Short Project VC 2023- Object tracking and detection

Iván Serrano Hernandez and Lluís Llull Riera

22/06/2023

Abstract

Object detection and tracking are fundamental tasks in computer vision with various applications, from autonomous vehicles to surveillance systems for example. The objective of this project is to perform a comprehensive comparative analysis of different techniques for object recognition and tracking, evaluating their performance, robustness, and suitability for real-world scenarios.

The study focuses some categories of object detection and tracking methods: traditional computer vision approaches, deep learning-based methods, and hybrid techniques combining both paradigms.

Furthermore, we analyze the robustness of the methods against challenges commonly encountered in real-world scenarios, such as occlusions, scale variations, and cluttered backgrounds. We examine their ability to handle object tracking under challenging conditions and assess their performance using appropriate evaluation metrics, including tracking precision and robustness.

Based on our findings, we provide insights into the strengths and weaknesses of each approach, offering guidelines for selecting the most appropriate technique based on the scenarios.

Contents

1 Introduction

In this project, we will explore a small range of tracking and object detection algorithms, covering both classical and machine-learning techniques. This exploration will involve implementing and evaluating different algorithms, analyzing their strengths, limitations, and trade-offs. We will investigate methods that we have seen in class, and compare them to others that are considered to be state of the art.

By undertaking this project, we seek to gain a comprehensive understanding of the principles and techniques employed in tracking and object detection. Furthermore, we aim to assess the performance of different algorithms in diverse scenarios and benchmark their accuracy, robustness, and suitability for real-world scenarios.

In order to test the algorithms build in real world situations, we will use the *TLP Benchmark dataset*, which will serve as a valuable resource for evaluating and refining our system's performance.

2 Dataset

Among the vast collection of videos available in the TLP Benchmark, we have specifically chosen four videos to focus on: **Aladdin**, **Drone**, **Bike**, and **Motorcycle Chase**. These videos have been carefully selected due to their diverse and challenging scenarios, providing a comprehensive evaluation of our system's performance.

MotorcycleChase: is about a persecution of a moto. It is relatively simple because the background is always gray because of the road, so the object is easy to distinguish. The main difficulty that it presents is the transformation in scale and angle that the motorcyclist undergoes at some point in the video.

Figure 1: MotorcycleChase

Dron: is about a drone filming a guy in the beach. It is not very complex because the background is also constant, the sand. Nevertheless, the man in he video moves a lo, changing its shape, and interacts with other objects giving place to occlusions and merging objects.

Figure 2: Dron

Bike: A man walking near his bike. This is a complex video for many things; the cyclist moves quite frequently so his shapes varies a lot during the video, the colours are quite similar, and the background has many relevant points, like a well distinguishable skyline.


Figure 3: Bike

Alladin: A play of Aladin. Aladdin is a complex video, there are many objects in the scene, all very closed and with very poor ilumination, which makes objects hard to differentiate. What is more, the main object is constntly moving and changing this form, waht it makes bigger the difficulty, above all tracking.


Figure 4: Alladin

3 Metrics

We have used two metrics to compare different techniques and check if we are getting good results.

The first one, is getting the match percentatge of the bounding boxes, is calculated intersection of the two boxes divided by the union, like this way: $\frac{A \cap B}{A \cup B}$. We are going to call this the **Intesection metric** .

Te second one, is calculated by the euclidean distance of the centroids of the bounding boxes. We are going to call this the **Distance metric**: $d = \sqrt{c(A)^2 + c(B)^2}$ where c(A) is the positions x and y of the centroid of the bounding box A

To make a comparison between the models studied in this project we are going to create a two lists with the each metric of every frame for every algorithm, this way later we can get relevant statistical data about both metrics. But there is some things that we have to keep in mind.

1. Some times the mean of the metrics can be not representative since they may be working very well all the time and in a specific frame lose the object, and the result is a bad mean. For this reason it is necessary also to check the plots.

2. In the tracking method the intersection of the bounding box is interesting, but not as the distance between centroids of the boxes, because in some cases the is not any intersection but the boxes are very close one each other.

3. With the ML based techniques, we have to consider that the results will not be always equal, they are stochastic.

4 Object detection

Object detection is the process of identifying and localizing multiple objects within an image or video. It involves detecting and classifying different objects of interest, and drawing bounding boxes around them to indicate their locations. Object detection algorithms use a combination of image processing techniques and machine learning models to achieve this task. These models can be based on traditional computer vision approaches like Haar cascades or more advanced deep learning architectures like Convolutional Neural Networks (CNNs). Object detection is a crucial component in a wide range of applications, including autonomous driving, surveillance, object recognition, augmented reality, and robotics, enabling systems to understand and interact with their visual environment more effectively.

4.1 Template Matching

4.1.1 Explanation

Template matching is a fundamental technique in computer vision that involves comparing a template image with different regions of a larger image to locate a specific object or pattern. By sliding the template across the image and calculating similarity measures, the region with the highest similarity score indicates the likely location of the template. While template matching is straightforward to implement and computationally efficient, it can be sensitive to changes in lighting conditions, occlusions, and scale variations.

In our case we are getting as a template the bounding box of the first frame of the video. Lets see the example of the *Moto* case. 5

Figure 5: Template of Moto dataset

The second step is create check the correlation of each step of the image we want to detect, to understand this better we can make a warmap of correlations 6

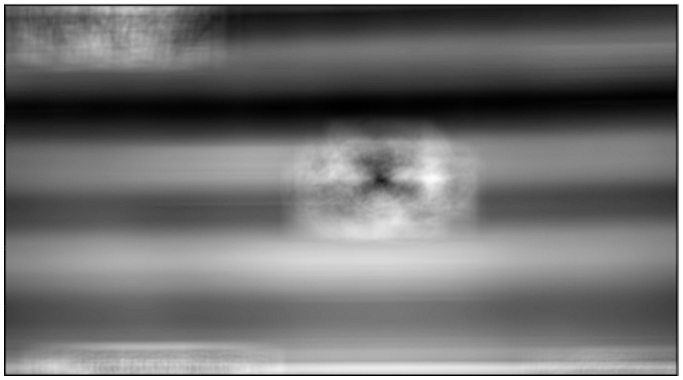

Figure 6: Correlation warmap of Moto dataset

Finally, we get the point of maximum correlation and we create the bounding box of the same size of the template.

4.1.2 Metrics and results

With the *Moto* dataset we can clearly detect one problem of the **Template Mathing**. We can clearly see that the first frames return us very good results. .

Figure 7: Template Matching frames 2 and 150

But the problem starts when we apply transformations to our image, for example scaling. Even this, in some cases we still found a good match, but now the object to predict is bigger than the template

Figure 8: Template Matching frames 360 and 570

In the next plot 9 we can see what we commented before, the first frames give us very good results, but at a certain point when the video starts zooming in starts getting bad results.

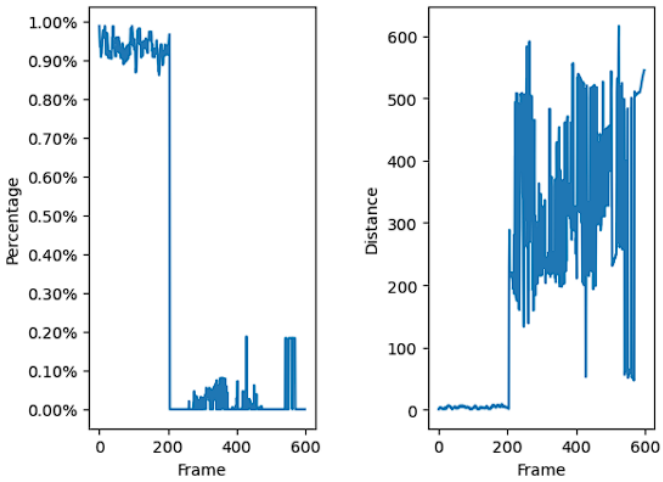

Figure 9: Plot of template matching Moto dataset

With the *Drone*, we are obtaining very similar results, but the problem in this case is not a transformation, is the lack of adaptability, Static templates may not accurately represent dynamic objects, in this case the guy is moving and the algorithm does not detect him accuretly. With the Bike we are not obtaining bad results.

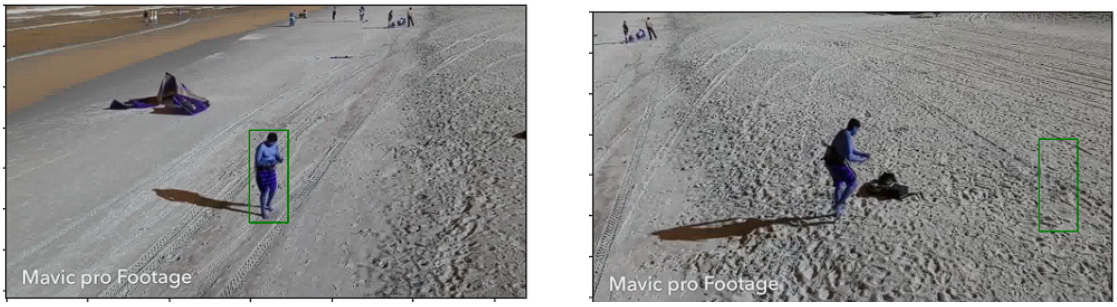

Figure 10: Template Matching frames

Lets observe the plot.

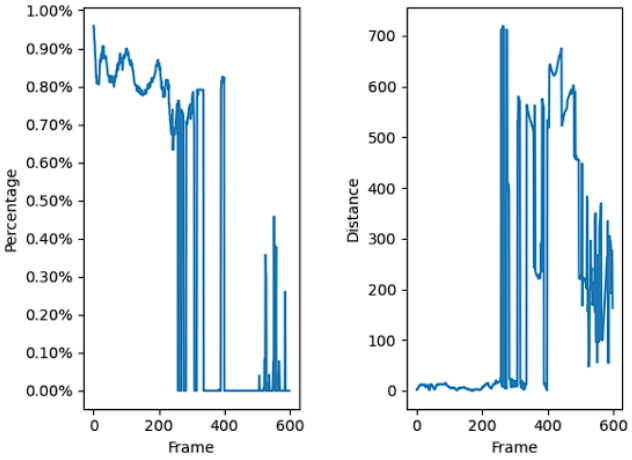

Figure 11: Plot of template matching Drone

The Bike seems a difficult dataset for TM because the object to detect is constantly moving, but in this case gives better results than we expected.

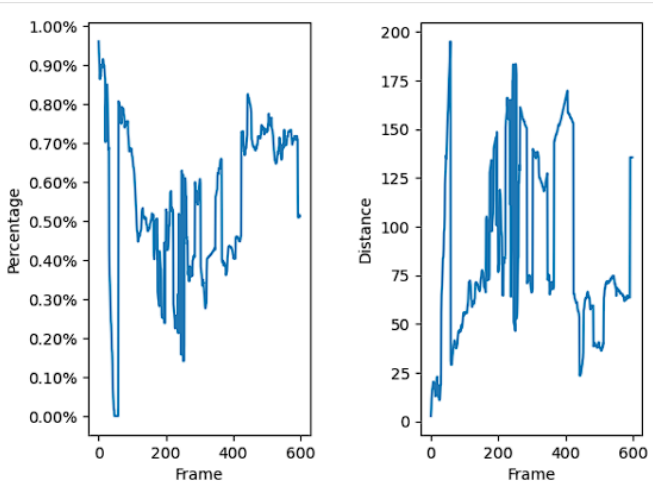

Figure 12: Plot of template matching Bike

Finally *Alladin* shows other debilities of this technique, that we are going to discus in the next subsection, like the changes of ilumination, the lack of adaptability, the transformations, oclusions...

Figure 13: Template Matching frames

4.1.3 Analysis of the results and associated problems

In general we can say that *Template Matching* is a good algorithm. The advantages of this technique are the next ones:

- **Simplicity and Intuitiveness:** Template matching is a straightforward and intuitive approach
- It is a good option for constant or not changins videos, that we could guarantee that will not be drastic changes.

But it haves very clearly problems:

- **Lack of adaptability**: Static templates may not accurately represent dynamic objects undergoing deformations or appearance changes.

- **Occlusion and partial object detection**: Template matching struggles with detecting partially occluded objects or objects with significant occlusions.
- **Sensitivity to variations**: Template matching struggles with scale, rotation, illumination, and perspective variations, often leading to inaccurate or failed detection.

4.2 YOLO

4.2.1 Explanation

YOLO (You Only Look Once) is a neural network for object detection. Unlike traditional methods, YOLO formulates object detection as a regression problem and directly predicts bounding boxes and class probabilities from a grid-based division of the input image. By optimizing a combination of localization, confidence, and classification losses during training, YOLO achieves accurate and efficient real-time object detection. Earlier detection frameworks, looked at different parts of the image multiple times at different scales and repurposed image classification technique to detect objects. This approach is slow and inefficient.

YOLO takes entirely different approach. It looks at the entire image only once and goes through the network once and detects objects. Hence the name. It is very fast. That's the reason it has got so popular.

For this project we get the pretrained YOLO model and we made a second training process with the *COCO dataset.*[1] The next step is defining the set of labels that the model want to predict. In our case we used the next list of objects.

['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'sofa', 'pottedplant', 'bed', 'diningtable', 'toilet', 'tvmonitor', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

4.2.2 Metrics and results

With the *Drone* dataset, we haven't encountered any strange news. In almost all the frames, the model correctly detects the person, and it also accurately detects other persons in the background, handbacks, and other objects. However, there are some cases where it fails and produces false detections, like the baseball bat example.

[1] COCO dataset that is a large-scale, widely used benchmark dataset for object detection, segmentation, and captioning tasks in computer vision.

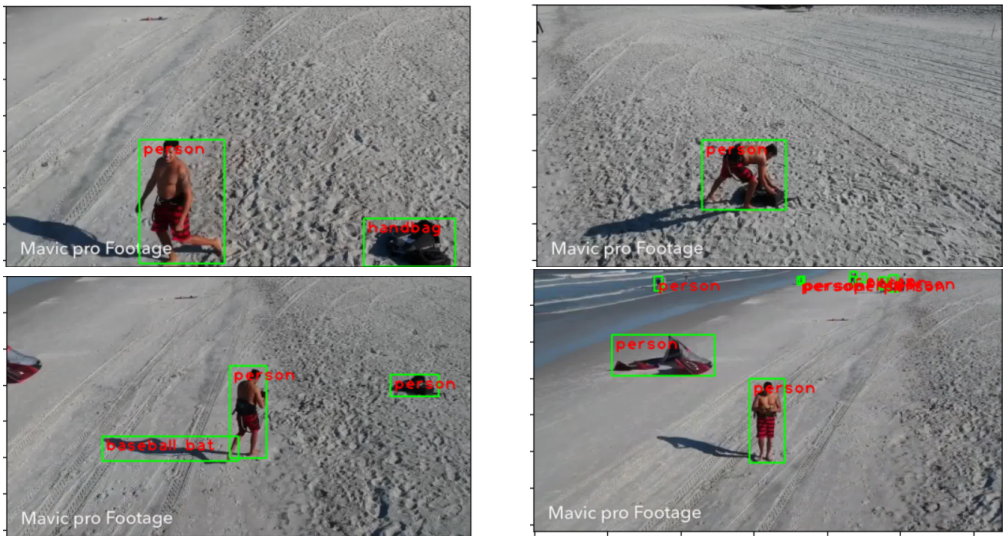

Figure 14: YOLO detection for Drone

The case of the Moto dataset is very similar. The model predicts the motorbike in almost all the frames, as well as the cars. Interestingly, in some frames, it detects the same person twice as different individuals.

Figure 15: YOLO detection for Moto

The Bike dataset yields incredible results as well. What's particularly interesting about this dataset is how the model is able to recognize the bicycle from various perspectives.

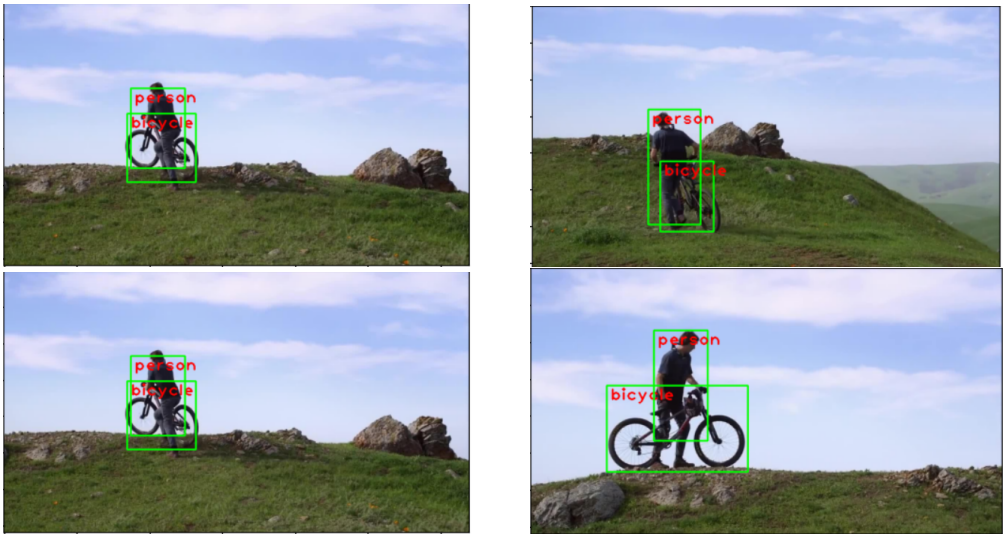

Figure 16: YOLO detection for Bike

Lastly, the Alladin dataset provides more diverse results. While it still performs well overall, we can expect some intriguing predictions due to the challenges posed by changes in illumination and the presence of numerous background objects. For instance, the model's prediction of a laptop is difficult to comprehend, highlighting one of the main issues with black box models. YOLO, in this dataset, also frequently identifies a 'knife' and various persons in the background. Although the dataset performs well when focusing solely on the intended object, it falls short when considering all the false predictions.

Figure 17: YOLO detection for Alladin

4.2.3 Analysis of the results and associated problems

YOLO demonstrates impressive performance across different datasets. However, it is not without its challenges. One of the mains problems are:

- **False detections**, where the model occasionally misidentifies objects or individuals.
- Another significant concern is the **interpretability** of the model's decisions, as exemplified by the perplexing identification of a laptop in the Alladin dataset. This occurs with the black box models

5 Tracking

Object tracking is a fundamental task in computer vision that involves continuously locating and monitoring a specific object as it moves within a video sequence. It aims to determine the object's position, size, and shape over time, enabling many real world applications. Tracking algorithms analyze visual information to overcome challenges like appearance changes, occlusions, and complex motion patterns, and mainly are formed of two main steps: analyzing the frame and processing it, and establishing relations with the previous one in order to determine some prediction, like the position of a given object's bounding box.

5.1 SIFT features based tracker

5.1.1 Explanation

SIFT, or Scale-Invariant Feature Transform, are local image descriptors widely used in the field of computer vision. They capture distinctive and invariant characteristics of image regions, making them valuable for object recognition, and in this case, for object tracking.
The main reasons why we have decided to build a SIFT features based tracker are the following.

- **Scale invariance**
  They can detect and describe objects at different scales, allowing them to handle variations in object size and changes in viewpoint. Additionally, SIFT features are rotation invariant, they can handle object rotations and establish reliable correspondences between images or frames. This makes SIFT features robust and accurate even when submitted to geometrical transformations, important capabilities for tracking purposes.

- **Distinctiveness**
  They capture unique characteristics of local image regions, such as shape, texture, and edges. This distinctive information enables effective object recognition, as they can accurately represent and differentiate different objects.

- **Changes in lighting conditions**
  By focusing on capturing structural information rather than absolute pixel values, they are less sensitive to variations in illumination, enhancing the reliability of SIFT features in different lighting environments.

Nevertheless, SIFT features also have their drawbacks, which we will explore in detail along the following sections. Let's proceed now with the algorithm we have proposed.

5.1.2 Tracking algorithm

The SIFT-based tracker we have build mainly performs the subsequent steps.

1. **Analysis of the first frame:** Using the provided bounding box, which can be found in the *groundtruth rect* file, we extract the SIFT features of the first frame.

2. **Tracking itself:** Once the first frame has been analyzed, we proceed with the following ones. The main idea is to extract SIFT features for the next frame, perform a matching with the previous one, and finally find the perspective transform for the bounding box, in order to delimit the newly found region of interest.

Although this idea may seem simple at a first glance, we have incorporated several improvements to our algorithm in order to make it more accurate and efficient.

One of the main changes we have done in comparison to a common SIFT feature extraction algorithm is in terms of the matcher. The regular *openCV* BFMatcher is going to try all the possibilities, which is the meaning of "Brute Force" and hence it will find the best matches.
Because of this, we decided to use a different matcher, much more efficient, called *FLANN*, meaning "Fast Library for Approximate Nearest Neighbors". This matcher will be much faster but will find an approximate nearest neighbors. It will find a good matching, but not necessarily the best possible one, although for the sake of this project we consider it good enough.

Another improvement worth mentioning is the use of *Lowe's ratio test* in order to select only the best matches once the features of both current and previous frame have been extracted.

For each potential match, the Lowe test compares the distances of the two closest matches, and if the ratio of the distance to the closest match and the distance to the second closest match is below a predefined threshold, the match is considered ambiguous and not good enough to pass the test.

The main idea behind the Lowe test is that when a match is genuinely correct, the distance to the closest match should be significantly smaller than the distance to other incorrect matches. By applying the ratio test, matches that have ambiguous or similar distances are discarded, reducing the likelihood of false positives.

After experimenting with different values, we found out that a 75% of the original distance suited well enough our algorithm.

Now that the algorithm itself has been explained properly, let's proceed with the results obtained with the different data sets.

5.1.3 Metrics and results

Using the metrics explained in a previous section, the results obtained are the subsequent.

	Distance	**Intersection**
Motorcycle chase	92.285	0.339
Drone	306.890	0.083
Bike	178.570	0.112
Alladin	444.151	0.0759

It's easily noticeable that the results obtained are quite poor. Let's analyze the behaviour of the algorithm with detail in order to find what's going wrong.

Firstly, let's take a closer look at the Motorcycle Chase data set. When checking the first frames, everything seems to be going nicely, just as it can be appreciated in the following track. Nevertheless, tracking losses quickly the ROI.

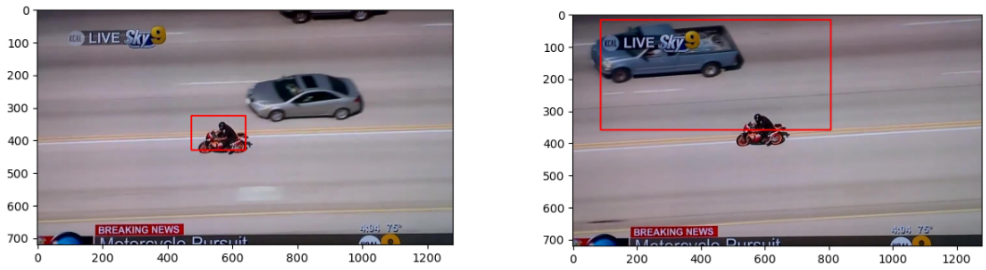

Figure 18: SIFT tracking samples on Motorcycle chase Data set

If we take a closer look to previous frames, moments before the ROI is completely lost, we see the following.

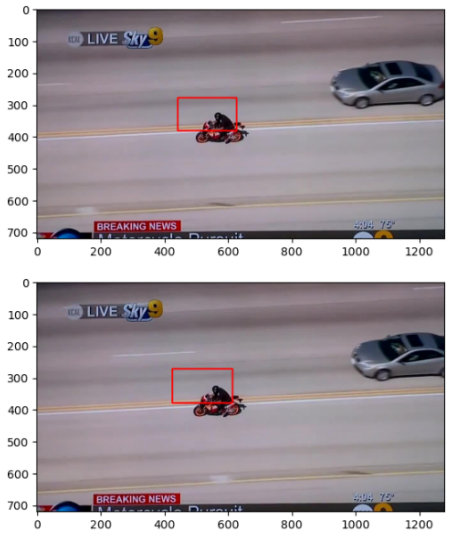

Figure 19: SIFT bad tracking sample on Motorcycle chase Data set

It seems like the bounding box progressively starts to grow away from the ROI. This same problem occurs with other data sets tested, for instance the Drone.

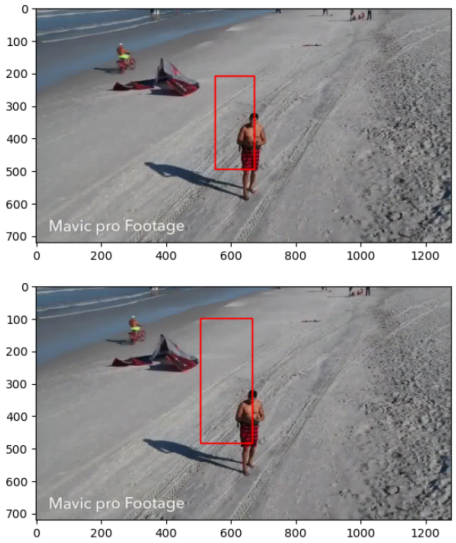

Figure 20: SIFT bad tracking sample on Drone Data set

After a certain amount of testing and investigation, we found out the the bounding box was sliding away from the object due to the fact that more SIFT features were being found outside the region of interest than on the inside, so the bounding box would inevitably grow outside the ROI in a small ammount of frames (around the 70th frame for most tests).

When analyzing the plots corresponding to the previously shown metrics, it's easily noticeable that the algorithm losses the object quite quickly.

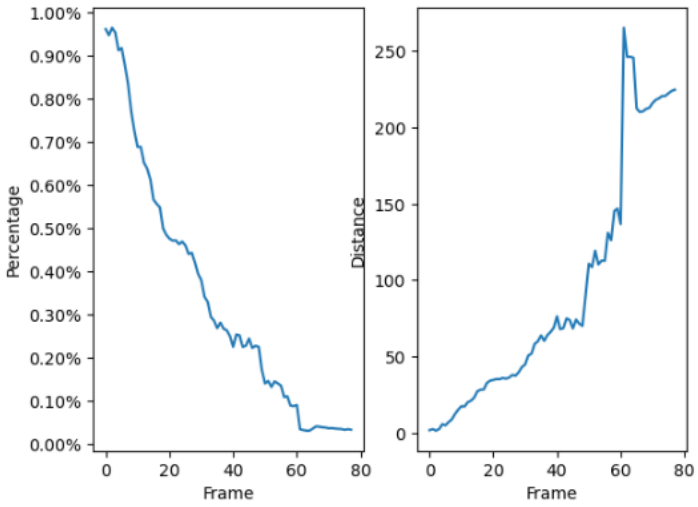

Figure 21: SIFT metrics plots on the Motorcycle chase data set

In fact, this issue got even worse when testing on the Aladdin data set, since there are plenty of details in each frame, so its even easier for the tracker to get lost when extracting and matching the SIFT features.

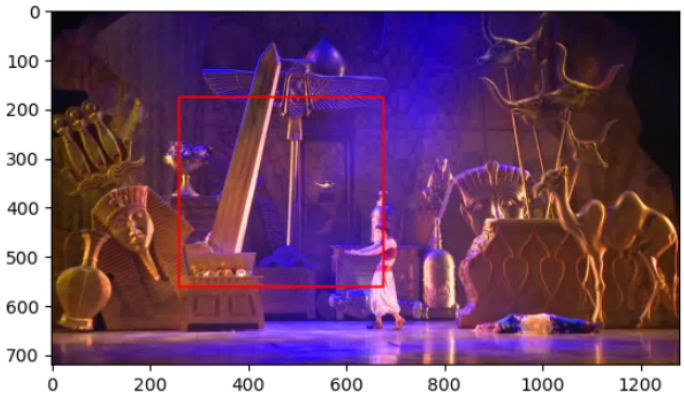

Figure 22: SIFT bad tracking sample on the Aladdin data set

The issue exposed in this section is known in the field of computer vision as the **drift problem**, and we are going to dive into it and how to solve it in the following section.

5.1.4 The drift problem

In the context of computer vision tracking, the drift problem refers to the cumulative error or deviation that occurs over time when tracking an object or feature in a video sequence. It arises due to various factors and can result in the tracked object gradually moving away from its true position, leading to inaccurate tracking results.

In our case, the reason why this phenomenon occurs is pretty clear. The main reason is that the SIFT feature extractor finds progressively more feature points far from the ROI than close to it.

Even though we delimit the finding of SIFT features to the bounding box detected on the previous frame, not the whole image, this error keeps taking place. Surely, the reason behind this has a lot to do with the fact that the bounding box captures a considerable region that is not strictly the object to track, leaving a lot of area for the feature extractor to capture points outside the object itself.

This issue gets worse when dealing with data sets with many possible points to extract features from, and the Aladdin and the Bike ones are good examples of it. For instance, in the Bike data set the skyline is a pretty differentiable point in the image because of its contrast and highlighted contour.

It's surely because of this that our baseline tracking algorithm gets lost when it starts to extract points from those remarkable regions instead of the object.

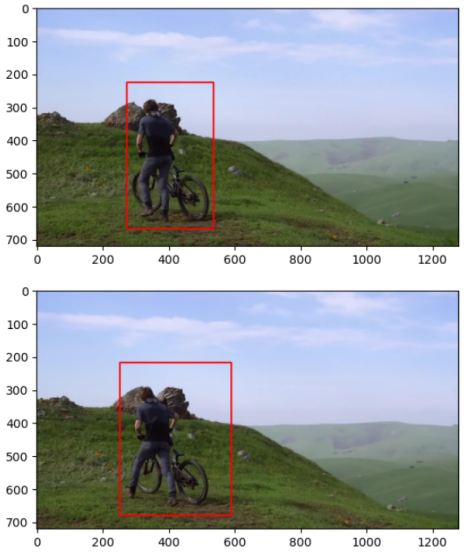

Figure 23: SIFT drift problem, Bike data set

It's easily noticeable that the ROI gets considerably bigger when is close to one of the mentioned areas, and from this point on the algorithm won't recover.

Since we weren't satisfied with the results obtained, we decided to propose a solution to this problem and test it in order to see in what extent it could possibly be solved.

5.1.5 Improved version for the drift problem and analysis of its performance

The solution we implemented is based on the idea of extracting in the first place feature points from the center of the previously detected ROI, and progressively expanding the area of search until a good enough match could be found. Concretely, what we have done is start to find the feature points in a bounding box centered on just a 30% of the original one, an keep increasing its size on a 10% each iteration until a good enough match is found (this method always converges, since it stops when trespassing the ROI original size).

Thanks to this improvement, the newly obtained results were the following.

	Distance	**Intersection**
Motorcycle chase	63.600	0.413
Drone	92.255	0.317
Bike	76.399	0.397
Alladin	41.314	0.531

It's pretty clear that the results obtained are way better than the previous ones. This enhanced version of the baseline SIFT tracker improved in this hereafter areas.

- **More precise tracking**: Although the drift problem was not completely solved, it was considerably delayed.
- **Efficiency**: Since the quest for a good match starts on a much smaller window than in the original algorithm, the tracker takes less time to run.

Nevertheless, this method has happened to have a considerable flaw, and is the possibility of getting to a frame where not enough feature points can be found too son. This can also happen in the baseline

algorithm, but not as soon as it has happened with the enhanced version, and this is due to the fact of continually focusing on a too small region of the image.

To appreciate better the results obtained, let's proceed to see a few of the results obtained with this enhanced algorithm.

Figure 24: SIFT good tracking sample on the Aladdin data set, enhanced version

The Aladdin data set is where the improvements are more noticeable. The tracker holds the ROI on its place for much more time and does not get lost when getting close to other objects, at least during the first part of the video. That's thanks to the fact that we are only using points from the center of the bounding box. Nevertheless, it runs out of points for the matching sooner, but this could be solved changing the matcher or improving the image resolution.

Another remarkable improvement is the ability to recover from a bad tracking, something that did not happen with the baseline version of the algorithm.

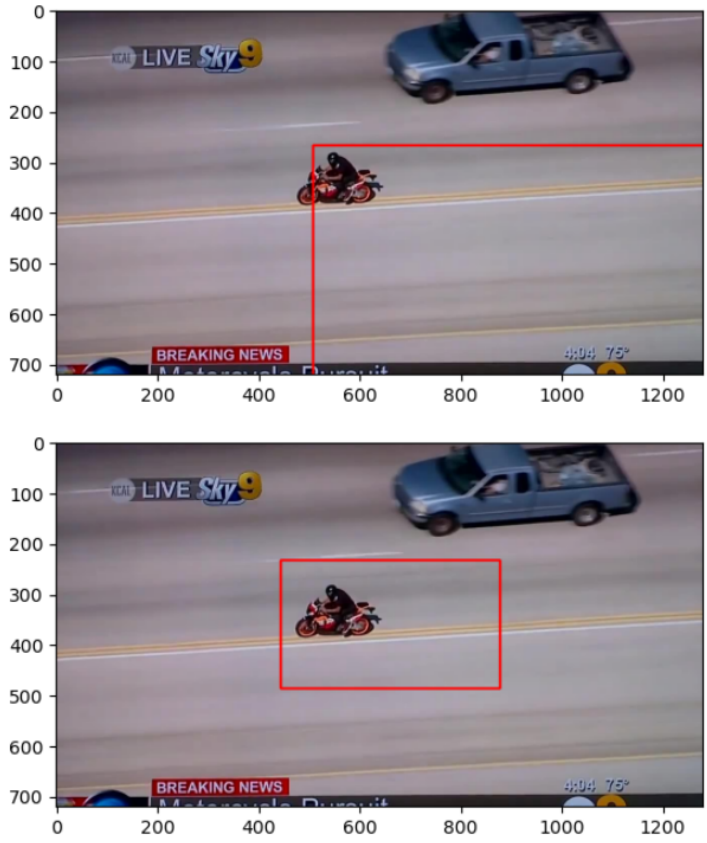

Figure 25: SIFT come back from bad tracking, enhanced version

In the frame where the detection is awful, the algorithm has possibly used the 100% of the previous box to extract the feature points, but on the next frame it doesn't have to be so, since it will always start by the smallest 30% of the centered ROI, giving it a chance to recover if it finds enough points.

To conclude with this version, let's observe the metrics plot for the motorcycle data sets with both the baseline and enhanced versions.

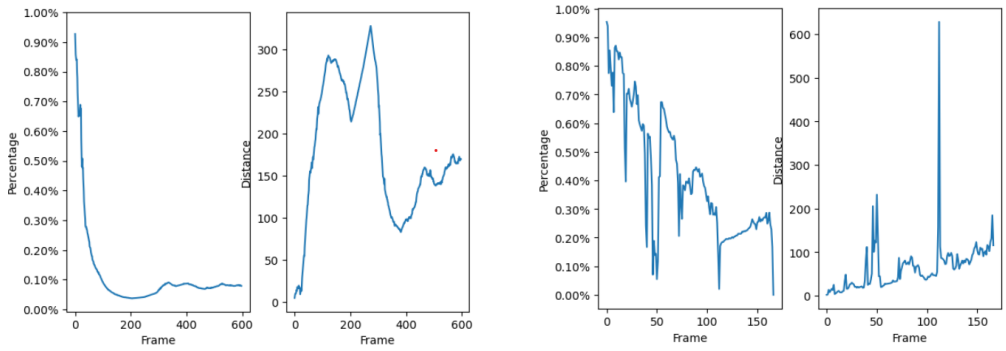

Figure 26: Comparison on the metrics plots, baseline (left) vs enhanced (right)

Even though the baseline version lasts for longer, since it is less possible to have the problem of lacking feature points due to the drift, it is remarkable how much better is the enhanced version. It is able to hold a much higher percentage of the intersection over union metric, and perform comebacks at some given points, holding the distance metric on reasonable values for longer.

5.2 Centroid distance based tracking algorithm

For this second version of tracking, our goal is to develop a tracking method with a more personal touch. The version that we are going to analyze next bases the tracking on the distance between the centroids of the objects that are recognized in the image. Due to the limitations of equipment and knowledge that we have, this is a simple method and we do not expect a performance much less excellent, and we understand that by its nature it will not be good at all in what scenarios, but we will see this in detail below.

5.2.1 Explanation

We can divide each processing of a frame in the following steps.

- **Detection on objects**: This is basically image segmentation in order to detect the different objects present in the image.
- **Computation of centroid distances**: Once the different objects have been identified, their distance to the previous frame's ROI is calculated in order to pick the closest one.

Regarding the image segmentation part, from all the techniques seen in class we decided to pick K-Means, since it is not any pre-trained model and we want this tracking algorithm to be as ours as possible. Some of the most used alternatives in this field are the following:

- **Convolutional Neural Networks (CNNs):** These deep learning models are specifically designed for processing visual data, they are specially good at learning hierarchical representations of objects and their features.
- **Histogram of Oriented Gradients (HOG):** This technique analyzes the distribution of gradients in an image, enabling the capture of shape and edge information.
- **Deep Learning-based Object Detection:** These are algorithms such as Single Shot MultiBox Detector (SSD) or You Only Look Once (YOLO), which we've used in this project. They mainly utilize convolutional neural networks.

The problem with this object detection techniques are that require a considerable ammount of training, and massive data sets of positive and negative cases for each particular object, hence, they are far too complicated to implement from scratch, taking into account that for this tracking algorithm we want to do it all by ourselves.

On the other hand, centroid based tracking is a popular and relatively simple approach for object tracking, easy to implement and efficient, since it only requires simple calculations that can be quickly performed, even for a huge number of detected objects, (its cost will be always O(n), being n the number of objects). However, it may face challenges in scenarios with complex object interactions, occlusions, or abrupt changes in appearance or motion, so we won't expect marvellous results.

5.2.2 Tracking algorithm

In this section we will analyze the algorithm in detail. Since it is a tracking algorithm, the main body of it will be similar to the other ones presented in this project: first we process the first frame, and then predict the next ROI for the following frames based on the previously collected data.

1. **Processing of the first frame**
   The groundtruth file is read, so we have the original ROI.

2. **Processing of the n-th next frames**

   (a) **frame segmentation**: Image segmentation of the next frame using k means clustering. Before performing the clustering, we reduce the noise of the image in order to make it easier for the clustering.

   (b) **contour extraction of the segmented image**: To do so, normally a morphological transformation is required previous to the contour extraction, but we have seen that this depends on the data set and its conditions.

   (c) **Identifying blobs from the contours**: First extracting its bounding rectangle, and then filtering by its size. We saw that complex images, just like the ones in the Aladdin data sets, can produce thousands of blobs using this technique, so we decided to filter the blobs detected. To do so, we only took into account in the computation blobs with plus/minus 50% of the original size of the previous frame's ROI.

   (d) **Perform the tracking itself**: With only the promising blobs, so pick the one with a closest euclidean distance to the previous frame's one, since the movement from one frame to another is pretty low.

5.2.3 Analysis of the results

Just as expected, the results obtained are quite poor, specially in some scenarios. The first point worth mentioning is efficiency, in every frame we are performing clustering, a calculation that involves all the pixels in the image, so it's very inefficient.

Then we have the common issues of the k-means clustering, it requires an input k, and the result that it provides is not a global optimum, just a close enough local one. On top of it, the fact that the clustering uses color to classify pixels, we have a serious problems with occlusions and background and foregrounds with similar colors, since its very complicated to decide which part of each detected blob belongs to whom once the contours are extracted. This issue could be solved relatively well with some morphological processing in some data sets, but that created problems of its own.

Nevertheless, this algorithm behaved considerably well taking into account its limitations in scenarios that suited it well, like the motorcycle chase or the drone data set. Those data sets provided good results due to the fact that the object to track was easily distinguishable from the foreground, so the clustering had it easier. On the other hand, in the Bike and Aladdin data sets it got awful results for exactly the opposite reason: the object to track was hard to differentiate from the background, and the image were also saturated with many forms and objects, which made it harder for the object detection.

Because of this, we are only going to take a look at the Motorcycle chase and Drone data set, since the other two did not provided enough results to even study them.

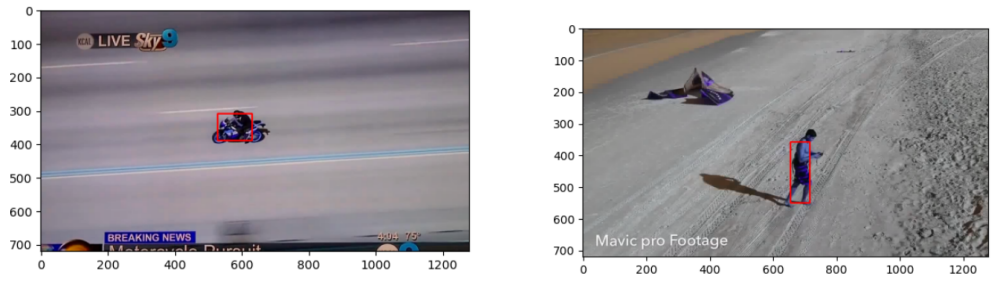

Figure 27: Centroid based tracking good tracks

In both images we see that the tracking does a pretty good job, the only issue that could be pointed out is the fact that the ROI is too small, but that occurs because of the morphological process we perform in order to simplify the segmented image contours, concretely an erode an a closing to simplify the shape of the detected objects and fill its gaps.

As it was mentioned before, this model is pretty sensible to occlusions. For instance, in the drone video, whenever the man that is being tracked gets close to another object with a similar shape and color, the tracker gets confused and detects the other object as the one to track.

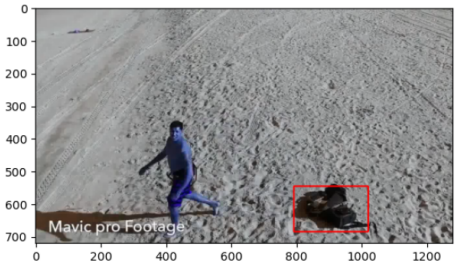

Figure 28: Centroid based tracking miss-tracking

To a detailed idea of how this algorithm performs, let's take a look at the metrics obtained.

	Distance	**Intersection**
Motorcycle chase	273.400	0.151
Drone	82.798	0.452

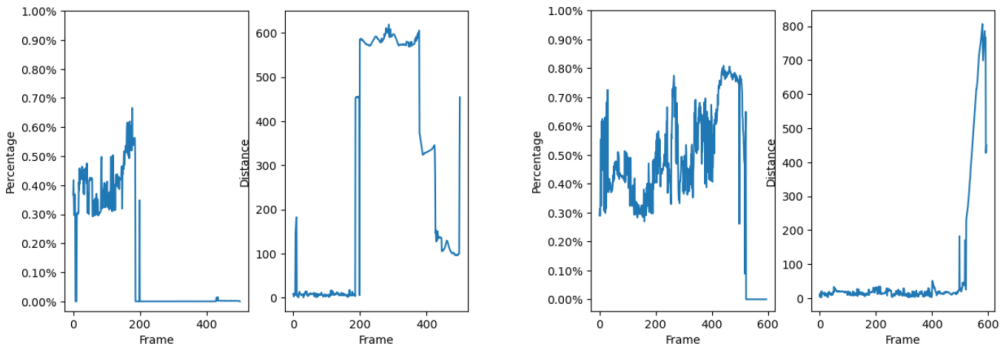

Figure 29: Centroid based tracking plots

The plots show a really decent performance, a least until the object gets lost (once the curve falls forming a step), specially in the case of the drone (the right plot), which we consider is the best suited data set for this model, since there are no occlusions and the object to tract is easily distinguishable most of the time.

5.2.4 Issues we have encountered

The main issues that we have faced with this method have been strictly related to the object segmentation part. The main difficulty has been to find a method that suits the maximum ammount of scenarios possible. Some methods that we tried but did not work well enough were the following ones:

- **Basic binarization**: Using the Otsu threshold to binarize. The main issue with this technique was the fact that it wasn't possible to distinguish the object to tract from other objects when they got too close, just like bellow.

Figure 30: Otsu binarization based object detection results

It could have been possible to split blobs once binarized, using the distance transformation and a watershed for example, but this should be adapted to every single frame where this issue was encountered, so it's not suitable for tracking.

- **object detection based on contours**: We tried to use Canny algorithm to extract the image contours and from that point identify the different blobs in the frame. However, Canny is a really good algorithm for contour detection, and we ended up having hundreds of ROI just as seen in the following image.

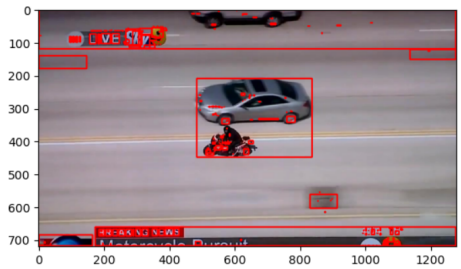

Figure 31: Canny binarization based object detection results

As wih the binarization, this issue could have been solved tweaking the parameters of the Canny algorithm and doing some morphological transformations, but again, this needed to be a particular solution for each problematic frame, so we did not consider it well enough for a tracking algorithm.

To conclude with this section, it must be said that it is difficult to find a method that suits well every single situation, specially without leaning on machine learning. The point was to find a trade off between scenarios that suited well the algorithm and others which not, so that why we decided to use k-means clustering.

5.3 Machine Learning techniques(CSRT, KCF and MIL)

5.3.1 Explanation

KCF (Kernelized Correlation Filter) employs machine learning by training correlation filters and kernel functions. It computes a correlation response map to measure the similarity between the target

and candidate regions. KCF is known for its speed and accuracy, making it suitable for real-time tracking.

CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) combines correlation filters with spatial and channel reliability maps. This integration enhances robustness by accounting for object scale changes, deformations, occlusions, and background clutter. CSRT also employs a multi-resolution search strategy to handle scale variations effectively.

MIL (Multiple Instance Learning) is a tracking method that deals with ambiguous training data. It represents the target object as a collection of positive and negative instances. MIL learns a classifier to differentiate between positive and negative instances, enabling robust tracking even when the target's location or appearance varies over time. *MIL* is particularly useful in scenarios with uncertainty or variation in the target's attributes.

5.3.2 Metrics and results

	MIL		**CSRT**		KCF	
	Intersection	Distance	Intersection	Distance	Intersection	Distance
MotorcycleChase	0.528	10.242	0.864	7.473	0.759	10.325
Dron	0.64	24.012	0.739	10.097	0.6864	9.27
Bike	0.236	149.59	0.569	86.4	0.203	154.56
Alladin	0.422	724.51	0.483	153.137	0.2607	72.15

We can see that all three methods give us very good results with the *Motorcycle* dataset. We could imagine that the MIL is not as good as the others because of the intersection, but if we check the distance is almost the same, this is because of the problem 2. We can observe it in the 32 that is making a good tracking but the box is not exaclty the same.

Figure 32: Failure of intersection metric.

For example if we check the plots for the *CSRT* method, we will see that in fact is a good method, the intersection is constantly high and the distance between centroids is low.

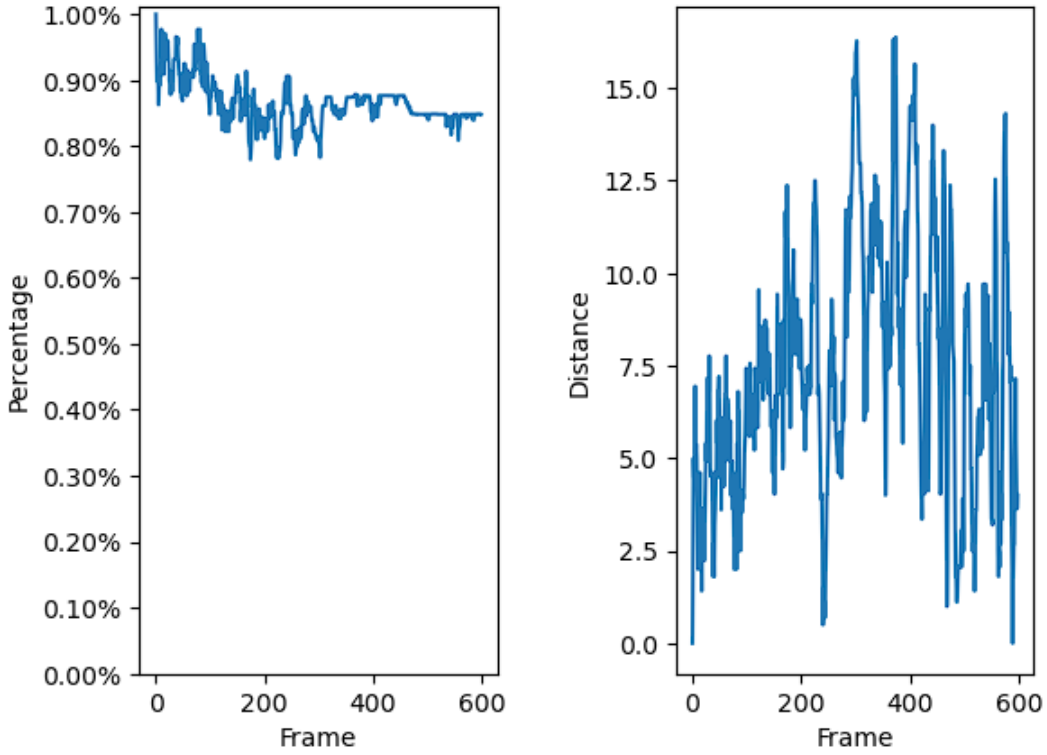

Figure 33: Plots for CSRT Tracking for Motorcycle.

We can clearly see that is also giving very good results for the *Drone* dataset

Figure 34: KCF fot Dron dataset

In the case of *Bike* it is interesting to see that it seems that metrics are note very good, specially in *KCF* and *MIL*, but what is really happening is that the algorithms for some reason are not tracking exactly the person but the bike, specifically the fron wheel.

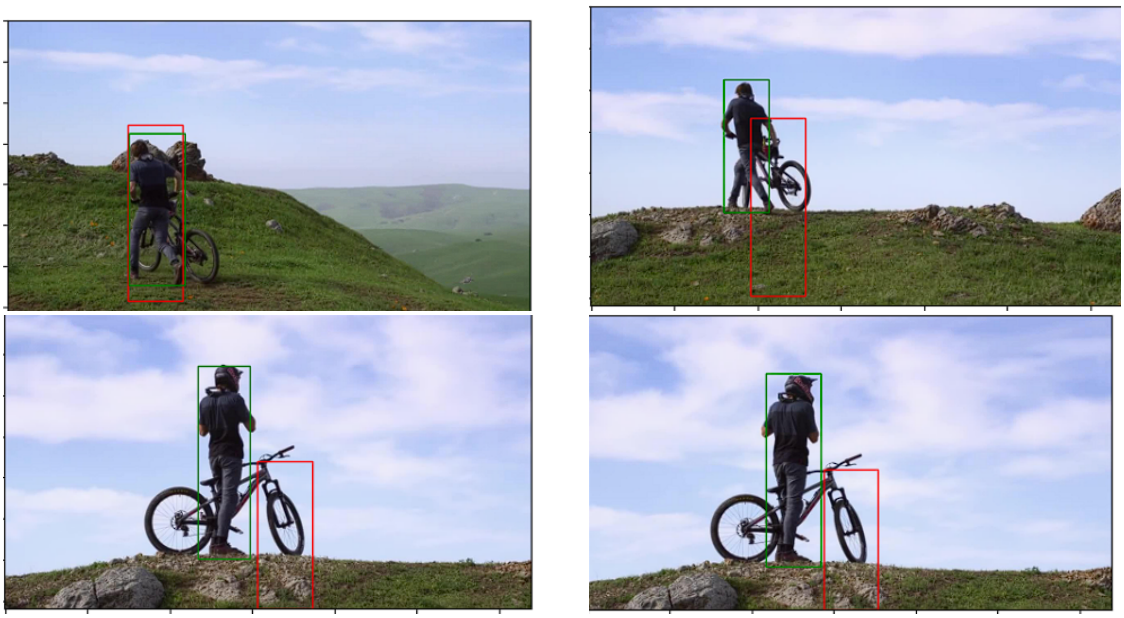

Figure 35: MIL fot Bike dataset

We can observe it in the plots, that in a certain frame the intersection begins to beeing zero, but the distance is not incrementing a lot.

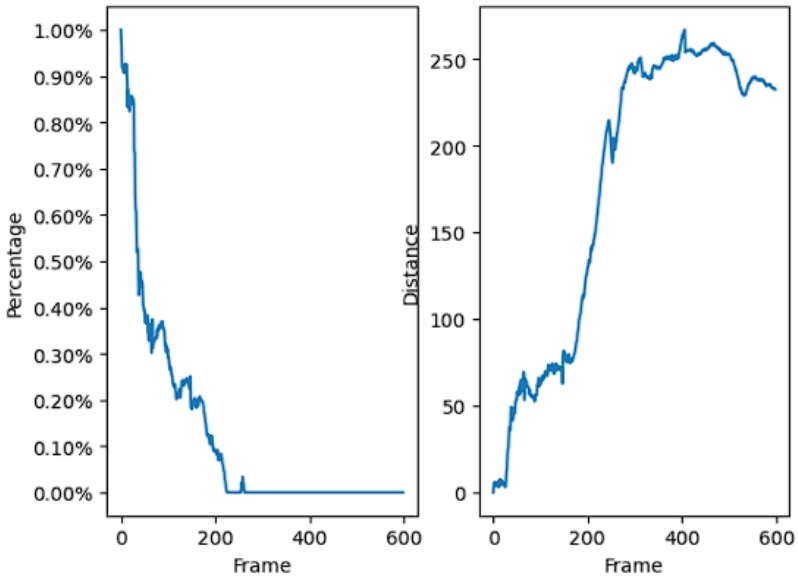

Figure 36: MIL tracking for Bike.

Finally in the *Alladin* case we can see more differences beetween the different models. With the KCF, the tracking is working more or less good unitl certain point that the object to track is get lost, how we can see in the next plot 37.

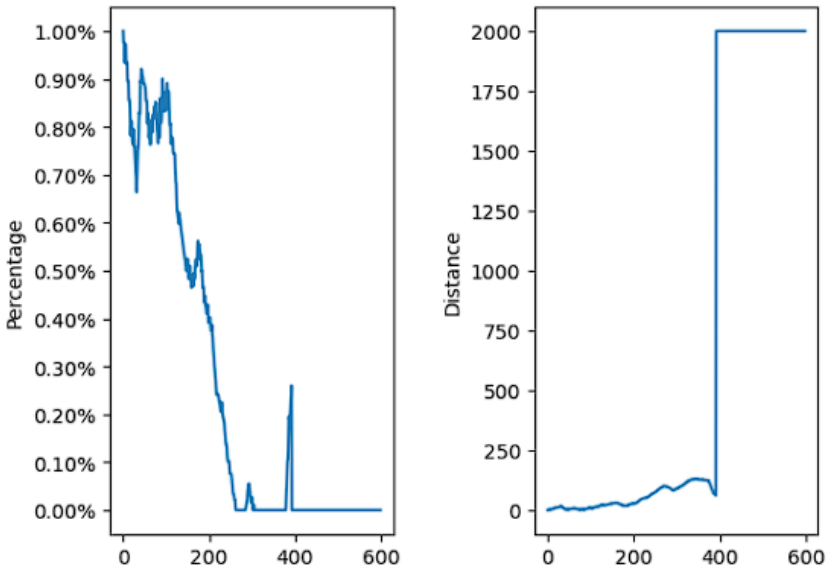

Figure 37: KCF tracking for Alladin.

The *CSRT* it not detects like it has lost the object to track, but we can clearly see that in some point is not returning the correct object.

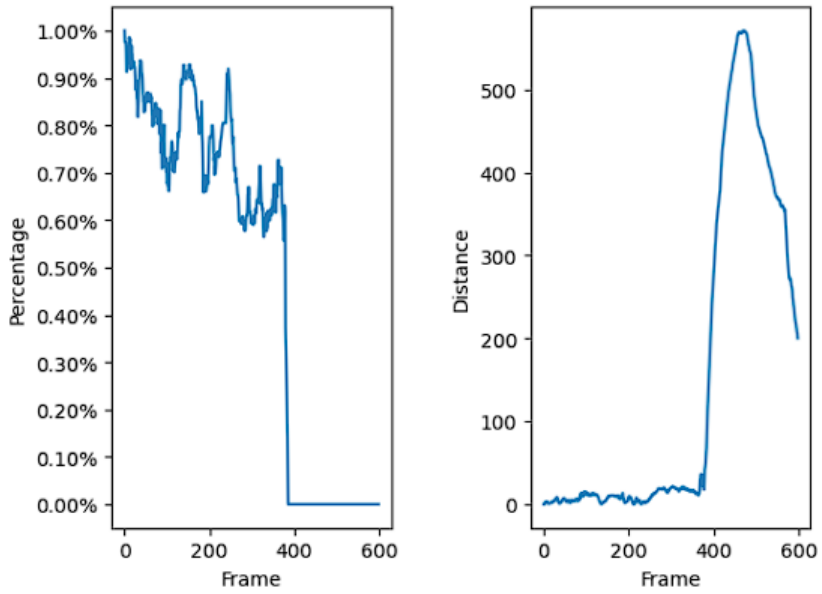

Figure 38: CSRT tracking for Alladin.

Finally, the *MIL*, unlike in the other datasets, its the one that performs better.

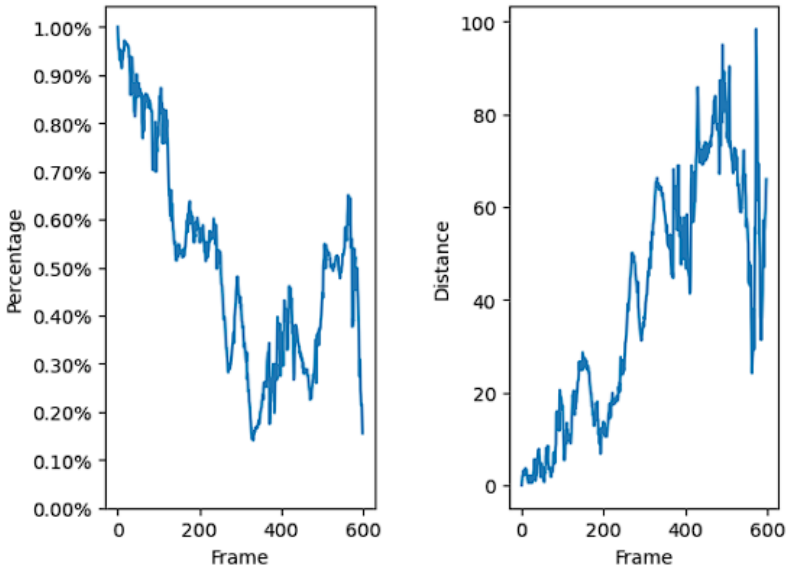

Figure 39: MIL tracking for Alladin.

5.3.3 Analysis of the results and associated problems

In general, we are getting very good results with the Machine Learning techiniques for tracking. What is more, we can see that *MIL* and *KCF* get very similar results. On the other hand, it seems that *CSRT* is getting better results.

Even this, the main problems we found with this concrete datasets are this ones.

- **Fast motion:** Rapid and abrupt motion of the object (for example in the *Alladin*)
- **Drift:** the cumulative error in the estimated position of the object over time

6 Results comparison

Just as we expected from the start of the project, algorithms that implemented machine learning techniques and pre-trained models have behaved much better than others more basic, like the SIFT based tracker or the template matching.

Regarding the tracking part, results have differed considerably from one method to another. Starting by the machine learning based methods, they are the ones who have set the benchmark in this section, since their results are really good, specially in the case of CSRT. Although they also got lost in the tracking, and their performance on the Aladdin data set wasn't as good as with the other data sets due to its complexity, they have performed solidly all round.

The next tracking method in order of performance has been the SIFT based one. SIFT features have proofed to be a good technique, mainly because of its robustness to changes in the orientation, occlusion and scale. Concretely, it has been the enhanced version the one that has provided better results, being our only self made method with the ability to recover from nearly fully-lost trackings thanks to its solution to the drift problem.

And lastly in this part, the centroid based tracker. The centroid based tracker is the perfect proof that is really complicated to create an all round tracker without using machine learning or AI, since it has behaved pretty good in some scenarios and incredibly bad in some others.

On the side of object recognition, YOLO (You Only Look Once) demonstrates impressive performance in detecting and recognizing the main object of interest. Its ability to accurately predict various objects in an image is commendable. However, one drawback of YOLO is the high occurrence of false detections of background objects. This limitation can affect the precision and reliability of the overall recognition system.

On the other hand, Template Matching, although a useful technique, has significant limitations. It relies heavily on finding exact matches between a predefined template and the input image. Consequently, it struggles to deliver reliable results when the frames deviate significantly from the template frame. The technique's effectiveness diminishes as the temporal gap between the template and the frames increases, making it less suitable for scenarios with dynamic or rapidly changing environments.

While both YOLO and Template Matching have their strengths and weaknesses, YOLO emerges as the superior choice for object recognition. Despite being a more complex technique that requires additional time and computational resources, YOLO achieves remarkable accuracy in identifying objects within a given frame. Its ability to detect multiple objects simultaneously and generalize well to diverse image contexts surpasses the limitations of Template Matching.

7 Next steps

In order to further enhance this project, we consider that the next logical step would be to implement a tracking method from scratch and train it from the ground up. While the project has utilized pre-trained models for tracking and object recognition, developing a custom tracking method would be really interesting in order to learn how it is to go through all the stages of a state of the art tracking algorithm. Two potential approaches we would like to explore are Haar features and Fast Convolutional Neural Networks (CNN), which we have seen in class.

Nevertheless, implementing and training custom tracking methods can be time-consuming and computationally demanding, requiring significant expertise in machine learning and computer vision. Additionally, it may involve large labeled data sets, which we would need to build from scratch (or find a suitable one) and substantial computational resources.

References

[1] Hao Zhou, Jian He and Zhangqin Huang, *Multiple Centroid-Based Multi-Object Tracking by Decision Making*

[2] Huiyu Zhou, Yuan Yuan, Chunmei Shi, *Object tracking using SIFT features and mean shift*

[3] Analytics Vidhya, *SIFT Algorithm — How to Use SIFT for Image Matching in Python*

[4] Abhinav Moudgil, Vineet Gandhi, *Long-Term Visual Object Tracking Benchmark, https://amoudgl.github.io/tlp/*

[5] OpenCV Official Documentation, *https://opencv.org/*

[6] Readthedocs.io, *https://opencv-tutorial.readthedocs.io/en/latest/index.html*