Generating Robust Motion Maps To Improve Object Detector Performances

A thesis submitted in fulfilment of the requirements for the degree of

Engineering Degree

in

Mathematical Modeling and Artificial Intelligence

By

Sébastien Grand

Supervisor: Dr. Jonathan Kobold

ENSEEIHT, INSA
France

February 2023

Abstract

This report presents the work realized during my master's thesis with the Hensoldt Analytics computer vision team. This internship concluded my engineering course at ENSEEIHT. This work was part of developing the computer vision team's main product, called Object Detection, Classification, and Tracking (ODCT), whose main task is to secure vulnerable areas such as airports from threats like unknown drones with the help of an object detector.

Recent approaches involve deep learning methods to perform object detection. YOLO has shown great abilities to process data in real-time. It has also shown great results on well-known benchmark datasets like COCO or PASCALVOC. However, this model shows weaknesses regarding the detection of small or partially occluded objects.

To counteract the problems of small object detection, the previous approach of Analytics was to provide additional information to the object detector. This additional information can be motion maps. Motion maps can be computed using a simple pixel-wise difference between two consecutive frames or using a motion detection algorithm. The result of the frame difference is a frame highlighting moving objects and cancelling static background.

However, the frame difference method suffers from issues like ghosting or overlapping, which is discussed in this paper. To overcome these problems, a literature review is conducted to find reliable *motion detection* methods. The method has to fit the requirements discussed later.

Then, the results obtained from the motion detection are used to fine-tune a real-time object detector. Finally, a comparison of performance (regarding *average precision* and *average recall*) with and without the added motion map is made to conclude whether or not the added information is beneficial to the model.

Acknowledgements

The past six months have been an exciting and rewarding experience dedicated to completing my engineering studies, as well as my first work experience abroad, in Munich within Hensoldt Analytics. I would like to express my gratitude to the people that welcomed and helped me during these six months.

First of all, I would like to thank my master's thesis supervisor Jonathan Kobold, Head of Machine Learning and Computer Vision at Hensoldt Analytics for giving me this opportunity and for his trust. It was the greatest professional experience I had. Jonathan gave me a lot of great advice and a clear guideline to complete my work. I also could learn from his experience and technical knowledge. Also, I would like to thank the Computer Vision team for their help, their kindness, and the time they gave me.

I would like to thank the two supervisors Laurent Mirambell and Fabio Manzoni Vieira who accompanied me during my two-year apprenticeship at Hensoldt France, with whom I had my first professional experience and was able to learn and progress.

Finally, I would like to thank all the people who contributed to the success of my studies, and recently my internship and who helped me, advised me and read me through the writing of this report.

Contents

List of Figures

List of Tables

List of acronyms

Acronym	Full meaning
CV	Computer Vision OSINT
Open Source Intelligence	
ODCT	Object Detection, Classification, and Tracking
RGB	Red - Green - Blue
CNN	Convolutional Neural Network
YOLO	You Only Look Once
NLP	Natural Language Processing
RANSAC	RANdom SAmple Consensus
FAST	Features from Accelerated Segment Test
GFTT	Good Features To Track
CPU	Central Processing Unit
GPU	Graphical Processing Unit
MRF	Markov Random Field
PCA	Principal Component Analysis
RPCA	Robust Principal Component Analysis
PRPCA	Panoramic Robust Principal Component Analysis
SIFT	Scale-Invariant Feature Transform
FLANN	Fast Library for Approximate Nearest Neighbors
ADMM	Alternating Direction Method of Multipliers
TP	True Positive
FP	False Positive
FN	False Negative
P	Precision
R	Recall
AP	Average Precision
AR	Average Recall
mAP	mean Average Precision
mAR	mean Average Recall
IoU	Intersection over Union
UAV	Unmanned Aerial Vehicle
SVT	Singular Value Thresholding
soft	Soft thresholding

1 Introduction

1.1 About HENSOLDT Analytics

HENSOLDT Analytics has been founded in 1999 as SAILlabs and acquired by HENSOLDT group in 2021. Analytics is a global leading provider of Open Source Intelligence (OSINT) systems.
Analytics' main product is developed in the Vienna Analytics headquarters. It is a media mining Natural Language Processing (NLP) technology for collecting, monitoring, and analysing data from open sources. These sources include TV, radio, website, social media, etc.
Another product line is the Object Detection, Classification, and Tracking capability, developed by the Computer Vision (CV) team led by Dr. Jonathan Kobold in Munich. It is not only used for OSINT systems, but also for counter-UAV purposes. A counter-UAV system can be a ground station system, where cameras are on the ground and detect incoming drones. An active countermeasure is also part of the system. This is an interceptor drone which flies to the target drone to neutralize it, with an onboard CV system.

1.2 Context

Critical infrastructures are more and more threatened by drones or UAVs. To protect such infrastructures from drones, these need to be detected. The new generation of surveillance systems uses Deep Learning (DL) devices like real-time object detectors.
These new devices face challenges. They have to work in real-time, detect as many targets as possible, detect targets with different sizes, detect targets occluded, etc. However, current real-time object detectors like YOLO [3], have shown great results on benchmark datasets like COCO [4] or PASCALVOC [5], they still face the challenges listed above, and particularly they are not yet performant at detecting tiny objects.
A potential solution investigated by the CV team is to provide the object detector with additional information to help it in its detection. This additional information can be for example an image highlighting moving objects in a frame. One method was developed 2.1.3.1, but it has limits. So, new methods are investigated to solve challenges and improve the previous results (fig. 2).

1.3 Objectives

The objective of this work is to improve results from the previous one. The objective is separated into two distinct sub-objectives.
The first one consists of investigating the methods for motion detection 2.1.2 in the case of a moving camera capturing the scene. Moving the camera implies that the ego-motion of the camera has to be compensated. With the motion compensated, the method has to allow a separation between what is moving and what is not in the sequence of frames. This investigation is conducted through a literature review. Then, in accordance with the different requirements discussed in 2.1.1, a method is chosen, implemented and evaluated against the previous work regarding *precision*, *recall*, and *F1-score* defined in 3.1.1.2.
The second objective consists of using the results of the motion detection algorithm presented in section 2.1.5 to fine-tune YOLOX [2] an object detector and improve its detection performance regarding *average precision* and *average recall* defined in 3.2.3.

2 Methodology

In this section, the objectives are addressed separately. The first objective is to find a method for *motion detection* 2.1 in a moving camera scenario. So, a list of requirements 2.1.1 is drawn up for the motion detection model. Then, a literature review is conducted in order to know the different motion detection methods that exist. Among these methods, those that are in agreement with the requirements are selected. After discussion, a method is selected to be implemented and evaluated against a baseline model from previous work 2.1.3.1. The expected results are an improvement in motion detection.
The second objective is to exploit the results from the motion detection section 2.1.5 to train an object detector. The model selected is YOLOX [2], and it is slightly modified to train it with additional information (not only RGB frames). Using this additional information, the performances of the detector are expected to increase regarding *average precision* and *average recall*.

2.1 Motion Detection

In this section, different motion detection methods are investigated. Motion detection has to be reliable in the case of a moving camera capturing frames over time. However, to properly address the problem, it is important to define what motion and moving cameras mean.

- **Motion or moving object**: There are two different cases. The first one concern static cameras, in this scenario the background appears to be static over the frames captured by the camera. However, in the second case, the camera is moving. Thus, the background is also moving. At this point, it is difficult to make a distinction between a moving object and a static scene. The different possibilities for moving objects are very large. A tennis player playing tennis and a waving tree are both considered moving. But, only the tennis player has to be considered part of the motion. A waving tree is a part of what is called a dynamic background. In the end, it is important to separate moving objects from the dynamic background in a moving camera scenario.

- **Moving camera**: The major issue with a moving camera scenario is that a static object appears moving along frames. This is caused by the camera motion also called the *ego-motion*. There are different kinds of moving cameras. Those which are constrained in their motion, for example, surveillance cameras that can only perform right to left and zoom-in zoom-out motion and, those which are not denied and, are free to move. This second case is ours because for us the camera can be placed on a drone.

Then, to properly address motion detection in the case of a moving camera, a list of requirements is settled 2.1.1. According to this list, different methods are considered and one is chosen to be implemented and evaluated.

2.1.1 Requirements

A list of challenges is established in section 2.1.1.1. These challenges are issues that can be difficult to overcome. Then, a list of requirements, that the selected method has to respect, is drawn up. Finally, the challenges addressed and left unaddressed are stated.

2.1.1.1 Challenges

The problem of motion detection involves various challenges, which a model must try to tackle. Here is a list of challenges stated in the literature:

- **Moving camera**: The scene is captured by a moving camera, this makes motion detection even more difficult. The background due to the motion is also perceived as being in motion. So, the ego-motion compensation has to be performed to simulate a static camera scenario.
- **Low-contrast:** The frame can show low contrast, thus the moving object can be confused with the background (fig. 1b).
- **Dynamic background:** Waving trees or water on the move present motion but are remaining in the background (fig. 1a).
- **Illumination changes:** The illumination in one frame to another can quickly switch, leading to false positive pixels.
- **Low frame rate:** A low frame rate involves camera abrupt variations and makes the motion compensation difficult to perform.
- **Blurred pixels:** Appear when the camera is moving abruptly and so the motion is hard to be compensated again.
- **Motion parallax:** Happens for example when a drone camera is chasing a target. due to the high backwards-forward motion, parallax occurs and makes the motion estimation difficult to perform.
- **Shadows:** The object shadow can be understood by the model as a moving object, however, it is not.
- **Tiny objects**: Tiny objects represent only an area of 32^2, and are challenging to detect or identify (fig. 1).

With this list of challenges, different requirements are expected and listed in the section 2.1.1.2.
Also, figure 1, shows different challenges listed here. Note that the figures are cropped to make the drones visible, and the drones are circled by a red circle.

2.1.1.2 Requirements

The method for motion detection has to follow a list of requirements. Thes requirements are:

- A relatively low computational cost, so that a CPU can run the model. The GPU power is taken by the object detection model 2.2;
- A relatively low memory cost, so that an embedded system can handle it;

With the challenges coupled with the requirements, only certain challenges are tackled. They are listed in the section 2.1.1.3

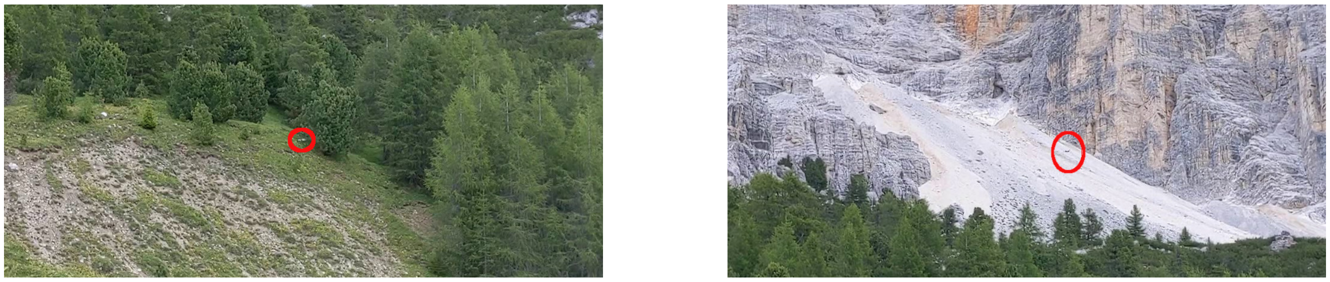


Figure 1: Two frames showing the three challenges listed in section 2.1.1.2. On both the object is tiny. In the left frame, the background is dynamic because of the moving trees. In the right one, the low-contrast challenge is faced.

2.1.1.3 Tackled Challenges

Here are listed the different challenges that the model tackle:

- **Moving camera**;
- **Low-contrast**;
- **Dynamic background**;
- **Tiny objects**;

These challenges can be viewed in figure 1 and have to be addressed with regard to the requirements. In the next section 2.1.2, different methods are investigated.

2.1.2 Literature Review

In this section, an overview of different possible techniques of motion detection in the scenario of a moving camera capturing the scene is presented. Also, according to the requirements, Deep Learning methods are not considered as they cannot be handled by a CPU.

2.1.2.1 Panoramic background modelling

Panoramic background modelling involves countering ego-motion by constructing an overview of the captured scene, to reduce the problem to a static camera scenario. This method is often used for Pan-Tilt-Zoom cameras involved in surveillance that are constrained in their moves. However, in our case, the camera motion is not constrained.
M. Brown and D. Lowe [6] present a method of image stitching using Scale Invariant Feature Transform (SIFT) [7] to stitch multiple images together, for an unconstrained camera. In the case of multiple frames (for example n frames), a point of view has to be defined to be the centre of the stitching. Using SIFT the idea is to extract pixels to track over n images. Then, taking one frame, each extracted pixel is matched to the extracted pixels of the next frame. The idea is to find the best matches which are supposed to best express the transformation that occurs from the first image to the next. The best matches are used to

estimate a homography [8] transformation which expresses the transformation that happens between two consecutive frames.
Once, all the homographies are known, each image can be aligned with its neighbouring image, the reference point of view being the $n/2$-th image. With all the images aligned together, the panorama is built, and the motion is compensated. An example of a panorama is presented in figure 4.
Using the motion compensation, it remains to separate the motion and static parts of the image sequence. C.Gao et *al*. [9] present a method using the low-rank and sparse approximation to divide the data into two parts. The first is the low-rank matrix concerning what remains unchanged over time. The second is the sparse matrix which contains the corrupted data or what is changing over the frames.

2.1.2.2 Sparse Optical Flow

Sparse optical flow [10] is the mostly used method to estimate and compensate for the ego motion of a camera. Sparse optical flow is a method for tracking the motion of pixels in a video. It works by identifying key points, or "feature points," in the image, and then tracking those points as they move from one frame to the next. Unlike dense optical flow [10], which computes the flow of every pixel in the image, sparse optical flow only tracks a subset of feature points, which makes it more computationally efficient. However, it also means that the resulting flow field will not be as detailed as a dense flow field. Additionally, it is prone to error when the feature points are not present in the next frame due to occlusion or other factors.
There are different methods to compute sparse optical flow. The first method is the Lukas-Kanade method [11] and is based on the assumption of small motion between consecutive frames, and uses the gradient of the intensity at each feature point to estimate the motion.
The second method is the Harris-Stephens corner detector [12], it detects feature points in the image and then tracks those points using the Lucas-Kanade method defined previously.
Another method is the Kanade-Lucas-Tomasi method, it is an extension of the Lucas-Kanade method that uses a more sophisticated feature point detector and a robust tracking algorithm, making it more resistant to errors caused by occlusion or other factors.
The last method presented here is SIFT defined in section 2.1.2.1, it is a feature descriptor and detector. It extracts features from an image and then tracks them over the next images.
Once one of these methods is implemented, ego motion is compensated. The problem is reduced to a static camera scenario. Different methods for motion detection (static camera case) can be employed. One method is frame difference [13], which is a technique used in computer vision and video processing to detect changes between consecutive frames in a video. It works by subtracting the intensity values of each pixel in one frame from the corresponding pixel in the next frame, resulting in a "difference image." Pixels in the "difference-image" that have a high-intensity value indicate a significant change in intensity between the two frames, while pixels with a low-intensity value indicate little or no change.
A second method is the Mixture of Gaussian (MoG) [14]. It is a statistical method for motion detection in video. It models the background of a video as a mixture of Gaussian distributions, with each distribution representing a different aspect of the background, such as colour or texture. The MoG model is updated dynamically, so it can adapt to changes in the background over time, such as lighting changes or the movement of objects in the scene. To detect motion, MoG compares the current frame to the background model and looks for pixels that deviate significantly from the model. These deviating pixels are likely to correspond to moving objects in the scene. The MoG algorithm typically uses a threshold to decide

whether a pixel is part of the background or a moving object. The threshold is adjusted based on the Gaussian distributions of the background model, and will likely change over time.
The last method presented here is SubSense [15] which is based on background subtraction. Subsense uses a background model that is learned over time and adapts to changes in the scene. Unlike other background subtraction algorithms, Subsense uses a combination of spatial and temporal information to create the background model, which improves its ability to handle changes in lighting and shadows. The algorithm also uses a technique called "shadow suppression" to remove shadows and other non-moving objects from the scene. The algorithm also uses a clustering method to the group moving pixels into regions corresponding to distinct moving objects. This clustering method allows Subsense to detect multiple objects moving independently in the same scene.

2.1.2.3 Subspace learning

Subspace learning for motion detection is a method for detecting moving objects in a video by analyzing the trajectories of feature points. The idea is to construct a subspace that captures the motion of the background, and then use this subspace to classify feature points as either part of the background or part of the foreground.
One way to construct a subspace is by using the long-term trajectories of feature points (as explained with SIFT section 2.1.2.1) as in the work of Sheikh et al. [16], where they use three long-term trajectories to construct a 3D subspace and classify feature points as background or foreground based on whether their trajectories belong to the subspace. Another way is to use trajectory affinities computed on motion and spatial location as proposed by Elqursh and Elgammal [17]. Once the subspace is established, trajectories of feature points can be analyzed by projecting them onto the subspace to separate the background from the foreground. The classification information is then spread throughout the image to get a pixel-level segmentation of moving objects. Subspace learning is an efficient method for motion detection as it is adaptive to changing backgrounds and lighting, and can detect multiple moving objects at the same time. However, it can be computationally intensive and determining the appropriate subspace dimensionality can be challenging.

2.1.2.4 Motion segmentation

Motion segmentation is a technique used in motion detection to separate moving objects from the background in a video. It involves analyzing the movement of feature points (section 2.1.2.1, SIFT), or regions in consecutive frames, and grouping them into different segments or clusters based on their movement patterns. Zhu and Elgammal [18] propose to cluster the trajectories regarding their affinities. Then, spread the label of trajectories, computing an intra-cluster variation. The clusters are automatically adapted according to the number of moving objects for each frame. To finish the process, an energy function minimization on a Markov Random Field (MRF) [19] is performed to propagate the background/foreground knowledge of the model to the unlabeled data.
Zhu et *al* [18] proposed multi-layer background subtraction and a multi-label segmentation and not only binary segmentation. Then, each cluster is linked to a layer. In each layer, pixel-wise motion estimation is performed. With this motion estimation, the appearance model and the prior probability map are updated and used to estimate the posterior map. MRF model is used to finish the multi-label segmentation on the posterior map.

2.1.2.5 Discussions

The methods presented in section 2.1.2, offer an overview of the possibilities to implement. Before choosing one method, the previous work made at HENSOLDT Analytics is presented and reviewed to find out about areas for improvement.

2.1.3 Previous Work

In this section, the previous work done by Analytics is presented. The first presents the method used for motion detection. The second part shows the approach for object detection. And the third part presents the results of this previous work.

2.1.3.1 Motion detection

Previous work focused firstly on motion compensation. To realize this motion compensation, sparse optical flow, discussed in section 2.1.2.2, was applied. The method employed to compute it was Lukas-Kanade [11] method, also presented in section [10]. The Lukas-Kanade method is used to extract "feature points" (pixels), to track them among frames. Using these feature points, a homography [20] transformation is sought. This homography allows warping consecutive frames together (see fig. 4). This warping procedure compensates for the motion of the camera.
Once the motion is compensated, the second objective is to compute motion detection. The result of the motion detection is a motion map, presented in figure 6b. To compute the motion detection, previous work focused on frame difference methods (discussed in section 2.1.2.2). The first experiment employed frame difference between two consecutive frames. Despite this method being easy and computationally efficient, it suffers from issues like ghosting and overlapping, an illustration of these phenomena is shown in figure 2b. Ghosting occurs when the same object is present twice in the same image. This is due to the difference between the images. Overlapping appears when the two images warped together are not covering the same area. This leads to pixels that are not subtracted and that are keeping their original value.
To overcome these problems, another approach was investigated, this approach uses 3 frames to compute a frame difference. This method is called Three-Frame Difference [13]. The idea is to use a frame from a future timestep to add it to the difference and reduce ghosting and overlapping effects. The formula employed to apply this method is defined by equation 1.

$$\text{Diff}_t = |I_t - I^*_{t+1}| + |I_t - I^*_{t-1}| - |I_{t-1} - I^*_{t+1}| \tag{1}$$

Where I_t is the frame at time t, and $*$ means that the image has been aligned to the reference image I_t. This method involving three frames reduces considerably ghosting and overlapping effects.
In the next section (2.1.3.2), the approach for the object detection part is presented.

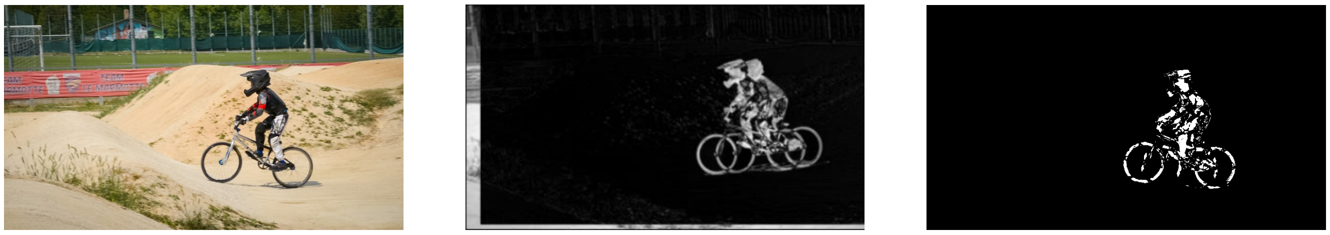
2.1.3.2 Object detection

In this section, the object detection improvements and experiments are presented. The aim is to understand the approach enabling to improve the detector performances.
An object detector is a type of computer vision algorithm that is designed to detect and locate objects

of a specific class within an image or a video. It typically consists of two main components: a feature extractor and a classifier. The feature extractor is responsible for extracting distinctive features from the image or video, such as edges, corners, and textures, that are relevant to the object class of interest. The classifier then uses these features to make a prediction about whether an object of the class of interest is present in the image or video, and if so, where it is located.
In the previous and present work, the class to be detected is the class *drone*. To improve the detector performance, an approach can be to add additional information to the input. Using the results of section 2.1.3.1, it is possible to modify the input layer of the object detector to add new inputs. In fact, object detectors use RGB images to detect objects. But, it is possible to concatenate the RGB image with a motion map that results from the algorithm in section 2.1.3.1. Thus, the input is not only the RGB image but also the motion information. The input is not a 3-channel image anymore, but a 4-channel image.
In the next section, the qualitative results of the motion and object detection are presented.

2.1.3.3 Results

The first result concerns the frame-difference method involving two frames. As discussed in section 2.1.3.1, this method showed limits like ghosting and overlapping, as shown in figure 2b. It is clear that the result of the two frame difference is not reliable and does not describe the moving object properly. Then, the result from the three-frame difference is shown in figure 2c.



(a) RGB frame (b) Motion map (2 frames) (c) Motion map (3 frames)

Figure 2: Figure 6a corresponds to the RGB frame. Figure 2b corresponds to the result of the two-frame difference method. Figure 2c corresponds to the result of the three-frame difference method.

The result of the three-frame difference method in figure 2c, shows a significant improvement compared to the two-frame difference method. However, some information about the motion is still not highlighted. Another method must therefore be tried to improve the results of motion detection.
In the next section (section 2.1.4), the different methods investigated in section 2.1.2 are discussed and one of them is chosen, with respect to the requirements (section 2.1.1), to be implemented.

The second result from the previous work is the result concerning the object detection experiments. As mentioned in the section 2.1.3.2, motion information was added to the input of the object detector to train it with 4-channel images. The results showed an improvement in the performance of the detector by adding the motion information to the input. The detector was found to be better at detecting drones in terms of the *average precision* and *average recall* presented in the section 3.2.3.
In the next section (2.1.4), different methods for improving motion detection are discussed. One of them is selected for implementation and testing.

2.1.4 Discussions

In this section, the different methods investigated in 2.1.2, are discussed. One method is chosen regarding the requirements (section 2.1.1).
The method *sparse optical flow* 2.1.2.2 also used in previous work 2.1.3.1 has shown limits. Feature points must be extracted and tracked over time. The features belong to both the static and the moving parts of the image. Thus, motion compensation is performed on both static and moving objects in the image. This results in inaccurate motion compensation, and therefore inaccurate motion detection.
The *motion segmentation* method (section 2.1.2.4) also uses feature point extraction. But here, all pixels in the image are classified as belonging to a moving object or a static object. This extraction can be difficult to achieve in the case of a very moving camera. Thus, if the motion segmentation is not accurate, the motion compensation is not accurate. Again, the motion detection is not accurate enough.
The method *subspace learning* is not appropriate as one of its characteristics is to be computationally expensive. Despite the approach being promising, it cannot be handled by a single CPU.
The *panoramic background modelling* (section 2.1.2.1) approach has shown interesting results recently. Especially, it is robust to noise, small object detection and is not computationally expensive. However, the storage of the frame, in memory, to build the panorama can be critical.

With regard to the different requirements and challenges, the **panoramic background modelling**, presented by [9] is chosen to be implemented and tested. In the next section (7), the problem formulation of the panoramic background modelling is presented.

2.1.5 PRPCA for Motion Detection

In this section, the Panoramic Robust Principal Component Analysis (PRPCA) is investigated. It has been presented by C. Gao et *al.* [9] as a foreground-background separation model. The method can be split into two different objectives.
Firstly, a registration step must be done to build a panorama made of the captured frames (N frames). Secondly, a low-rank and sparse decomposition is done to split the information of the panorama into the background (what is static in the frames), and into the foreground (what is moving in the frames)). This decomposition is performed using Robust Principal Component Analysis (RPCA) and is presented in [21].
The next section (2.1.5.1) presents the registration procedure aligned to two consecutive frames together.

2.1.5.1 Video Registration for Ego-motion Compensation

Video registration is used to compensate for the motion of the camera. Let X be a batch of N frames captured by a moving camera. The first objective is to register all the frames in the batch with respect to a reference point of view. This reference is the middle frame of the batch ($(N/2)$-th frame). It is named the *anchor frame*. As discussed in section 2.1.2.1, the idea is to build a panorama made of the frames in the batch. Once the registration is performed, the problem is reduced to a static camera scenario.
Thus, the procedure to register two consecutive frames is presented in section 2.1.5.1.1. Then, this procedure is extended to a multiple-frame registration in section 2.1.5.1.2.

2.1.5.1.1 Two frame stitching

Considering two consecutive frames $x(t)$ and $x(t-1)$. These two images, because they are consecutive are supposed to have a high correlation. When two images have a high correlation, it means that there is a strong similarity or match between the two images.
Using SIFT [7] as discussed in section 2.1.2.1, it is possible to extract feature points from $x(t)$ and $x(t-1)$. Let p be a feature point from $x(t-1)$, and p' from $x(t)$. These two points are known to correspond to each other in relation to a certain transformation from $x(t-1)$ to $x(t)$. If it is possible to extract many feature points. Then one can estimate the transformation that happens between the feature points in $x(t-1)$ and the feature points in $x(t)$. Thus, each feature point from $x(t)$ is matched to a feature point in $x(t-1)$, using Fast Library for Approximate Nearest Neighbors (FLANN) [22]. Figure 3 shows the extracted feature points from two consecutive images and their match.

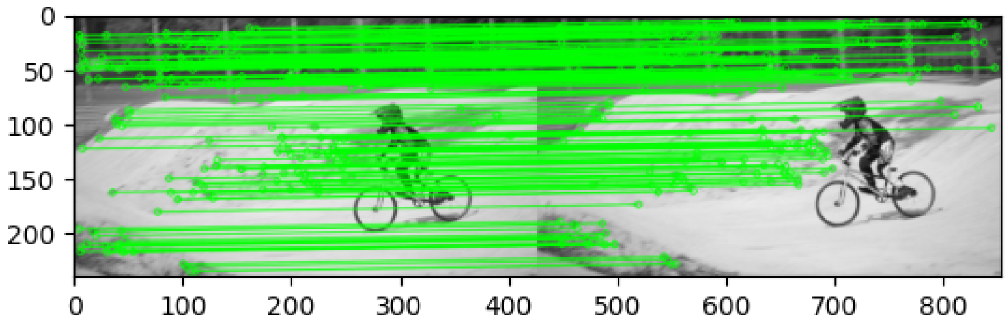


Figure 3: Extracted and matched feature points, respectively with SIFT and FLANN matcher, applied to two consecutive frames of *Bmx-bumps* video sequence from *Davis* dataset [1]

Finally, if the number of feature points matches is at least equal to four, it is possible to estimate a projective transformation H defined in equation 2

$$sp' = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} p \tag{2}$$

where $s \neq 0$ is a scaling constant and $H \in R^{3\times3}$ with $h_{33} = 1$.
This transformation estimates the projection of each pixel from $x(t-1)$ to $x(t)$ and is called a homography [23] transformation. Given a minimum set of 3 matched feature points, the homography H can be estimated using a least square minimization problem, as described in [24, p. 368]

$$\begin{aligned} \min_{h} \quad & ||Ah||^2 \\ \text{subject to} \quad & h_9 = 1 \end{aligned} \tag{3}$$

where $h = vec(H)$ is the vectorize version of the homography H. And $A^T = [A_1^T, ..., A_d^T]$ where d is the number of matched feature points, as presented in [9]. Then A_i the i-th component of A^T is defined as

$$\begin{bmatrix} 0 & p_i^T & -x'_{1i}p_i^T \\ p_i^T & 0 & -x'_{2i}p_i^T \end{bmatrix} \in \mathbf{R}^{2\times9} \tag{4}$$

where p_i is the i-th feature point of $x(t-1)$. x'_{1i}, x'_{2i} are respectively the projected feature points coordinates of $x(t)$, as explained in [9].
The solution of 4 is the smallest right singular vector of A. To estimate whether or not, the estimated projection is robust the RANdom SAmple Consensus (RANSAC) [25] algorithm can be employed to classify the estimations into inliers or outliers. The results that produce a projection with too much error are rejected.
Finally, the homography H with the fewest outliers is conserved to be the best estimation of the transformation.
In the next section (2.1.5.1.2), this registration procedure is extended to multiple-frame stitching as presented in [9].

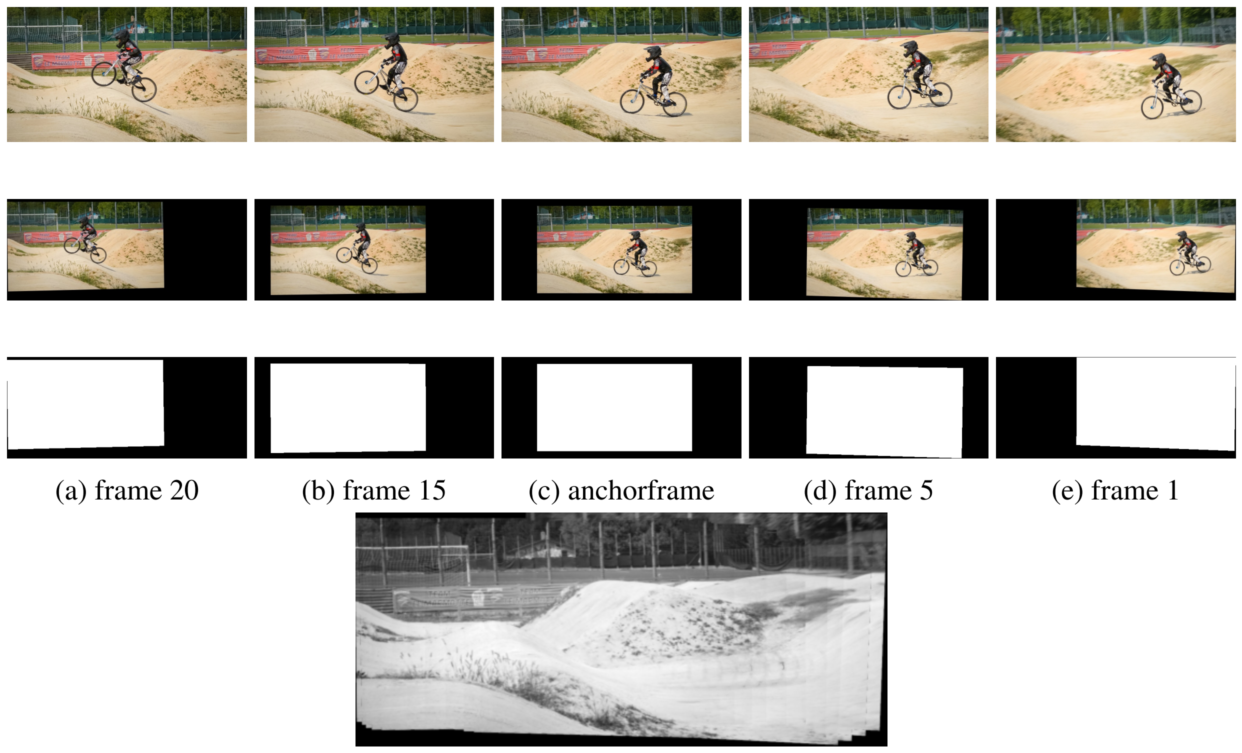
2.1.5.1.2 Multiple frame stitching

This work extends the section 2.1.5.1.1 to the case of multiple frame stitching proposed by C. Gao et *al.* [9]. A panoramic image is constructed using an anchor frame as a reference. In an online setting, new captured frames have to be placed regarding the reference.
This reference frame, also called the anchor frame, is the centre of the batch of images. The future panorama consists of all the images registered together with the anchor frame as a reference. As presented in section 2.1.5.1.1, homographies are computed to know the transformations that happen from one frame to another. Then, all the homographies have to be computed with respect to the anchor frame. The process is explained here.

Let $X = [x_1, ..., x_p] \in \mathbf{R}^{nm \times p}$ be the batch of vectorized frames of a moving camera video sequence. Let H_k be the projective transformation between the frame at time $k-1$ and the frame at time k. C. Gao et *al.* [9, eq. 5] propose to register the frames regarding the anchor frame $\tilde{k}$ as in equation 5.

$$\tilde{x}_k = \begin{cases} (\mathcal{H}_{\tilde{k}-1} \circ \mathcal{H}_{\tilde{k}-2} ... \circ \mathcal{H}_k)(x_k) & k < \tilde{k} \\ x_k & k = \tilde{k}, \\ (\mathcal{H}^{-1}_{\tilde{k}} \circ \mathcal{H}^{-1}_{\tilde{k}+1} ... \circ \mathcal{H}^{-1}_{k-1})(x_k) & k > \tilde{k} \end{cases} \tag{5}$$

With this registration procedure, all the frames are warped to each other with respect to the anchor frame as a reference. A new couple, (height, width) is then defined and informs about the panorama size formed by the union of the registered frames.
The results of the registration and the panoramic background (without moving objects) are shown in figure 4.



(a) frame 20 (b) frame 15 (c) anchorframe (d) frame 5 (e) frame 1

(f) Panoramic background made of 20 frames

Figure 4: Video registration with *Bmx-bumps* video sequence from *Davis* dataset [1]. The 1st row shows the original frame X_k. The 2nd row shows the registered frame $\tilde{X}_k$ regarding the anchor frame following 5 procedure. The 3rd row shows the support or mask M_k of each registered frame $\tilde{X}_k$. Black spaces represent unobserved pixels from the new point of view. The 4th row shows the resulting panoramic background without moving object

As it is presented in figure 4, the camera motion is compensated. A new camera point of view has been created and fakes a static camera video scenario. The mask shown in line 3 of figure 4e, is helpful for the next section 2.1.5.2, which presents low-rank and sparse approximation performed by the Robust Principal Component Analysis (RPCA).

2.1.5.2 Robust Principal Component Analysis

In this section, the Robust Principal Component Analysis [21] algorithm is presented, starting by making links between regular Principal Component Analysis (PCA) [26] and RPCA. Then, the method to solve an RPCA problem is described. Finally, the features showing the relevance of RPCA for motion detection are stated.

2.1.5.2.1 From PCA to RPCA

The Principal Component Analysis is a well-known technique for analyzing high dimensional data, which consists of finding orthonormal vectors to project data onto, in order to describe most of the data variation in fewer dimensions. Another formulation of the PCA can be an optimization problem.
Let $X = [x_1, ..., x_p]$ be a data matrix of rank r_0 composed by vectorized frames $\{x_i\}_{i=1...p}$ (one column of X is a vectorized frame x_i), and suppose that it exists another matrix L of rank r_1, $r_0 > r_1 > 0$. If it

is possible to remove noise or corrupted data from X, then one can compute the matrix L, without the noise of X. This matrix L is called a low-rank approximation of X. Solving the minimization problem 6 results in the best low-rank approximation L of the original data matrix X.

$$\min_{L} |X - L|_F^2 \text{ with } rank(L) < r \in \mathbf{N}^* \tag{6}$$

An important property of X to obtain an accurate low-rank approximation L is that the columns of X, consisting of the vectorized images in this work, must be correlated. The solution to the problem is demonstrated by the Eckart-Young-Mirsky theorem [27], where Singular Value Decomposition (SVD) is employed.
Here, X is reduced to L because the assumption that X is corrupted with irrelevant data is made. However, if the corrupted data are relevant and useful to extract, the previous minimization problem 6 must be slightly modified. In fact, in the case of motion detection, where motion is supposed to belong to corrupted data, accessing this corrupted data can be useful. Candes et *al.* proposed in [21] to decompose X into two different matrices. L remains the low-rank approximation of X. Then, S is introduced as the sparse component of X. This sparse matrix represents what changes over the columns of X. In the present work, a moving object could change position over the frame and be extracted in the sparse matrix S. So, the formulation 6 is not appropriate anymore. The sparse component S must be added to the problem. The new proposed formulation ([21]) is

$$\min_{L,S} ||L||_* + \lambda||S||_1 \text{ with } L + S = X \tag{7}$$

where $||.||_*$ is the nuclear norm and refers to the convex envelope of the rank minimization function $rank(.)$ and provides the best low-rank approximation L. The use of a convex norm helps in the minimization process. $||.||_1$ is the $l1$ norm, and is used to induce sparsity of S. λ is the regularization parameter to adjust the sparsity of S.
With this new definition, the minimization problem becomes a convex optimization problem. The formulation 7 is called the Robust Principal Component Analysis (RPCA) and has been introduced in [21]. The 7 formulation is interesting to apply to motion detection in the case of a moving camera. Using the registration formulation presented in 2.1.5.1.2 represents the matrix X. Then, decomposing X into L for the static background and S for the moving object can produce better results than the previous work presented in 2c.
In the next section (2.1.5.3), the application of the registration part 2.1.5.1.2 coupled with the RPCA, to a moving camera scenario is presented.

2.1.5.3 Application: Moving camera sequence

In this section, the application of the section 2.1.5.1.2 coupled with the section 7 is defined to extract motion from a video sequence captured by a moving camera.
RPCA is a great method for motion detection when the camera is static. However, in the present case, the camera is moving. But, in the scenario of a moving camera, if the registration of p frames can be done and thus the ego-motion compensated, RPCA can be applied as for a static camera. So, the video to process is stored in a matrix X.
Let $X \in \mathcal{R}^{mn \times p}$ be defined as

$$X = [vec(\tilde{x}_1), ..., vec(\tilde{x}_p)] \tag{8}$$

where $vec(.)$ is the function used to vectorize a frame, and $\tilde{x}_k \in \mathbf{R}^{m \times n}$ is the $k-$th registered frame (defined section 2.1.5.1.2, shown fig. 4e 2nd row) from a video captured by a moving camera. m and n are respectively the new *height* and *width* of the frame $\tilde{x_i}$ (also the size of the panorama). p is the number of columns in X and represents the number of frames collected in X. Then, let $M \in \{0,1\}^{mn \times p}$ (fig. 4e, 3rd row) be the matrix composed of all the supports of each registered frame. It allows knowing for each frame where the active (white pixels) and the inactive (black pixels) regions are located.
With this representation of X and with help of M, the problem is reduced to a static camera scenario. As each row of X refers to a fixed point in space, the problem is the same as a static case, with incomplete information represented by the black part in the frame (see fig. 4e, 3rd row). In this case, it is possible to solve the problem with the RPCA formulation, where the idea is to decompose the data matrix X as

$$\mathcal{P}_M(X) = \mathcal{P}_M(L + S) \tag{9}$$

where $\mathcal{P}_M$ is the projection of X on M space, where M represents a mask shown in line 3 of the figure 4. This projection is defined by [9, eq. 8] as follows

$$(\mathcal{P}_M(x))_{i,j} = \begin{cases} x_{ij} & \text{if } M_{i,j} = 1 \\ 0 & \text{if } M_{i,j} = 0 \end{cases} \tag{10}$$

where x_{ij} is a pixel from the x frame, (i, j) as coordinates.
As discussed in 7, it is possible to decompose X in L and S to obtain segmentation between the background (or what is static in the scene) and the foreground (or what is moving in the scene). It is important to notice that the mask M is indispensable to solving the minimization problem properly. The model has to know at each step where to focus (on the active regions and not the inactive ones). Otherwise, the problem does not simulate a static camera scenario.
In this case, the problem becomes.

$$\begin{array}{ll} \min\limits_{L,S} & ||L||_* + \lambda||S||_1 \\ \text{subject to} & \mathcal{P}_M(L+S) = \mathcal{P}_M(X) \end{array} \tag{11}$$

In the next, is explained how this minimization problem is solved using ADMM (5.1).

2.1.5.4 Solving the RPCA problem

The matrix X and the problem formulation defined in section 2.1.5.3 are addressed here. Note that $\mathcal{P}_M$ (eq. 10) is a linear transformation as it is a projection and can be written as

$$\mathcal{P}_M(L + S) = \mathcal{P}_M(L) + \mathcal{P}_M(S) \tag{12}$$

and based on the section 5.1 formulation, two functions f and g can be defined as

$$\begin{array}{rl} f(L) = & ||L||_* \\ g(S) = & \lambda||S||_1 \end{array} \tag{13}$$

and thus, the following problem is defined

$$\begin{array}{ll} \min\limits_{L,S} & f(L) + g(S) \\ \text{subject to} & \mathcal{P}_M(AL + BS) = \mathcal{P}_M(X) \end{array} \tag{14}$$

where f and g are both convex, and $A, B \in \mathbf{R}^{nm \times nm}$. Then, as presented in [28] an optimal solution p^* of the problem 14 is defined as follows.

$$p^* = inf\{f(L) + g(S) \mid \mathcal{P}_M(AL + BS) = \mathcal{P}_M(X)\} \tag{15}$$

With this problem formulation, the augmented Lagrangian can be written as

$$\begin{aligned} \mathcal{L}_\mu(L, S, Z) =& f(L) + g(S) + Z^T(\mathcal{P}_M(AL + BS - X)) + \frac{\mu}{2}||\mathcal{P}_M(AL + BS - X)||_F^2 \\ =& ||L||_* + \lambda||S||_1 + Z^T(\mathcal{P}_M(L + S - X)) + \frac{\mu}{2}||\mathcal{P}_M(L + S - X)||_F^2 \end{aligned} \tag{16}$$

where $\mu > 0$ is the Lagrangian multiplier, Z is called the dual variable, and both A and B are identity matrices.
There, ADMM consists of 3 updates at each iteration, as defined in [28, p. 14].

$$\begin{aligned} L^{k+1} &= \arg\min_L \mathcal{L}_\mu(L, S^k, Z^k) \\ S^{k+1} &= \arg\min_S \mathcal{L}_\mu(L^{k+1}, S, Z^k) \\ Z^{k+1} &= Z^k + \mu(L^{k+1} + S^{k+1} - X) \end{aligned} \tag{17}$$

where k refers to the k-th iteration.
Here, the two primal variables L and S are minimized alternately, and this two-step minimization is what allows to split f and g from eq. 14. Then, the ADMM formulation problem is rewritten in its scaled form for convenience. The idea is to combine the linear and quadratic terms of $\mathcal{L}_\mu$ and define the primal residual term R as shown in [28, p. 15]. R the primal residual is defined as $R = \mathcal{P}_M(L + S - X)$. This implies the following changes

$$\begin{aligned} Z^T R + \tfrac{\mu}{2}||R||_F^2 &= \tfrac{\mu}{2}||R + \tfrac{1}{\mu}Z||_F - \tfrac{1}{2\mu}||Z||_F^2 \\ &= \tfrac{\mu}{2}||R + U||_F^2 - \tfrac{\mu}{2}||U||_F^2 \end{aligned} \tag{18}$$

where $U = (1/\mu)Z$ is called the scaled dual variable. The ADMM iterations can be reformulated as in [28, p. 15].

$$\begin{aligned} L^{k+1} &= \arg\min_L(||L||_* + \tfrac{\mu}{2}||\mathcal{P}_M(L + S^k - X) + U^k||_F^2) \\ S^{k+1} &= \arg\min_S(||S||_1 + \tfrac{\mu}{2}||\mathcal{P}_M(L^{k+1} + S - X) + U^k||_F^2) \\ U^{k+1} &= U^k + \mu\mathcal{P}_M(L^{k+1} + S^{k+1} - X) \end{aligned} \tag{19}$$

Note that, L^{k+1} has the form of a *proximal operator* for the nuclear norm. Also called *Singular Value Threshold operator*. $SVT_\lambda(Y) = UD_\lambda(\Sigma)V^T$ is a solution of the problem 20

$$\arg\min_x \frac{1}{2}||x - y||_F^2 + \lambda||x||_* \tag{20}$$

where in the present case, $x = L$ and $y = \mathcal{P}_M(S - X) + \frac{U}{\mu}$, $\lambda = 1$, and $D_\lambda(\Sigma) = Diag([\sigma_i - \lambda])_+$, where σ_i is the ith singular value of the matrix involved in the singular value decomposition.
Then, S^{k+1} formulation refers to the *proximal operator* of the $l1$-norm, which stands as *Soft Thresholding operator*, $soft_\lambda(Y) = sign(Y)max(|Y| - \lambda, 0)$. It is a solution of the problem 21

$$\arg\min_x \frac{1}{2}||x - y||_F^2 + \lambda||x||_1 \tag{21}$$

where $x = S$ and $y = \mathcal{P}_M(L - X) + \frac{U}{\mu}$.
Finally, the 3 ADMM steps can be reformulated as follows

$$\begin{aligned} L^{k+1} &= SVT(L^k - \tfrac{1}{\mu}U^k) \\ S^{k+1} &= soft_\lambda(S^k - \tfrac{\lambda}{\mu}U^k) \\ U^{k+1} &= U^k + \mu\mathcal{P}_M(L^{k+1} + S^{k+1} - X) \end{aligned} \tag{22}$$

The objective is to perform these steps at each iteration, until both primal (R^k defined in 2.1.5.4) and dual ($H^k = ||S^k - S^{k+1}||_F^2$) residuals have decreased sufficiently with respect to a certain tolerance. Also, μ_k the lagrangian multiplier is updated according to eq.34.
At this stage, L and S are computed. It remains to readjust the point of view to have the same as the original image.

2.1.6 Postprocessing: Re-adjusting the point of view

Once the RPCA is completed it remains to perform postprocessing on L and S, to readjust the point of view and get back to the original one. For each frame warped into the panoramic view, the goal is to warp it back to the original point of view. At first, the four corners of the warped frame that are called *moving points* are selected. Using the mask M defined in equation 10 the active regions are known (pixels value is one. With these corners, a transformation T has to be found, to re-project the *moving points* into the *fixed points* that are defined by the width and height of the original frame. The problem is simply defined as

$$\begin{bmatrix} t_i x^{'} \\ t_i y^{'} \\ t_i \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{23}$$

where t_i is the scaling factor and T is the perspective transformation matrix, defined as,

$$T = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \tag{24}$$

where the a_i coefficients describe the transformations such as rotations, scaling, etc. The b_i coefficients define the translation vector and, to finish the c_i define the projection vector. The transformation of an image is presented in the scheme below.

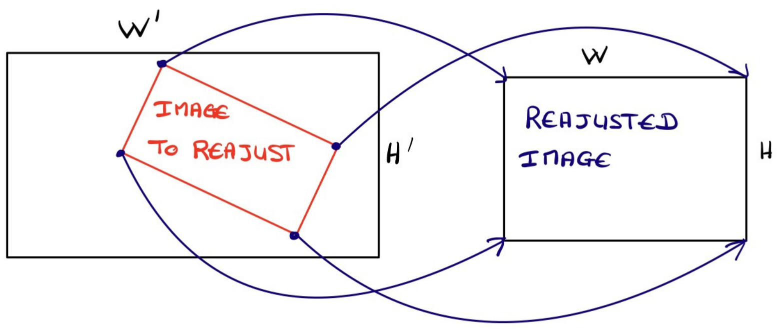


Figure 5: Reajusting the point of view of an image using a perspective transformation

2.1.7 Binary motion map

At this point, it is not yet possible to know the relevance of the extracted raw motion presented in figure 8. Post-processing on these motions is performed to get a binary version of this one. So, regarding the pixel values distribution of the sparse matrix, it is clear that the values are mainly centred around a grey value of 144 in the interval $[0, 255]$. These pixel values represent the background, while the other values represent the motion. These other values are out of the main distribution and are more likely to be found around 50 or 230. So, the idea is to simply centre the pixel values to get an image mean of 0 and then use the absolute value of the image. The obtained result is then an image where the majority of the values are around 0, while the motion pixels have a much higher value, ideally close to 255.
Then, applying a thresholding process to get a binary frame provides assessable results. A well-known method used to get a proper threshold is Otsu's method [29]. It is an optimal threshold calculation, where the idea is either to maximize the extra-class variance (where there are classes of pixels) or to minimize the intra-class variance. The separation of the two classes is made by the threshold value. The result of this process is presented in the 4th row of figure 6b.

2.1.8 Trajectory map

In this section, as it was investigated by the previous work presented in section 2.1.3.1, the idea is to compute a *Trajectory map*. The deal is to present to the added channel of the object detector not only the motion at the current timestep but the motion of the previous timestep also. So, a single frame can then highlight the trajectory of the moving object over time. This method aggregates past motion maps whose pixel values decayed over time. As the most recent motion or object location is the most relevant information for the detector, the pixels representing the current object location are highlighted in red, so that it is possible to distinguish the current location from the earlier ones. The older movements are all assigned to grey values, while their intensity decreases with an exponential decay rate concerning their temporal position in the sequence. The following process is described by the algorithm 1, and the result of this algorithm is presented in figure 6c with the original RGB frame and its corresponding motion map.
In the next section 2.2, the object detection part is addressed using the results from the motion detection.

Algorithm 1: Trajectory map algorithm

```
Data: $X \in \mathcal{R}^{N \times h \times w}$ ;      /* Binary motion map sequence (BGR format) */
Result: $Y \in \mathcal{R}^{N \times h \times w \times 3}$ ;      /* Trajectory map sequence (BGR format) */
$k \leftarrow 0$;
while $k < N$ do
    if k = 0 then
        for i = 1:h do
            for j = 1:w do
                $val\_px \leftarrow X_{kij}$;
                if $val\_px = [255, 255, 255]$ then
                    $X_{kij} \leftarrow [0, 0, 255]$;
                end
            end
        end
        $Y_k \leftarrow X_k$
    end
    else
        $X_{k-1} \leftarrow X_{k-1} \exp(-k \times 0.05)$;
        $X_k \leftarrow X_k \mid\mid X_{k-1}$;
        for i = 1:h do
            for j = 1:w do
                $val\_px \leftarrow X_{kij}$;
                if $val\_px = [255, 255, 255]$ then
                    $X_{kij} \leftarrow [0, 0, 255]$;
                end
            end
        end
        $Y_k \leftarrow X_k$
    end
end
```

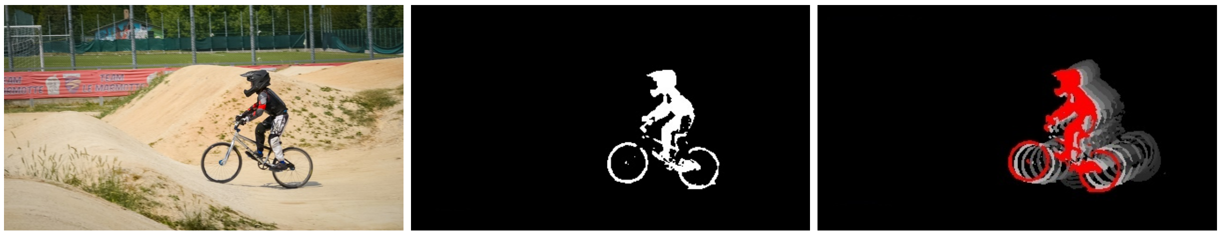

(a) RGB frame (b) Binary motion map (c) Trajectory map

Figure 6: Illustration of Trajectory map regarding its corresponding binary motion map and RGB frame

2.2 Object Detection

A real-time object detector is a system that can identify objects in a video stream or a single image and locates them in the frame while processing the video or image in "real-time," meaning fast enough for the output to be useful to a person or machine in near-instantaneous fashion. This is in contrast to a system that processes video or images in batch mode, taking a long time to produce output. Real-time object detection systems have many applications, including self-driving cars, surveillance, robotics, and augmented reality. They work by using machine learning algorithms to analyze images or video frames and identify patterns that correspond to known objects. The algorithms are typically trained on large datasets of annotated images, which contain bounding boxes around the objects of interest. There are several different approaches to building a real-time object detector, including using traditional machine learning algorithms, or more modern approaches like deep learning, which involves training large neural networks on the data. In general, deep learning approaches tend to be more accurate but also require more computational resources and are more difficult to implement. Overall, real-time object detectors are an important tool for enabling machines to understand and interact with their surroundings more intelligently and autonomously.

2.2.1 YOLO

The YOLO family [2, 3, 30–34] is probably the most popular object detection algorithm, it is fast and accurate, making it a famous choice for a wide range of applications. YOLO divides the input image into a grid of cells, and each cell is responsible for predicting the presence and location of objects within its region. YOLO uses a single convolutional neural network (CNN) to predict the bounding boxes and class probabilities for the objects in the image. It has several versions and one has improved upon one the previous one in terms of accuracy and speed. It has also several research projects, including the development of object detection models for specific tasks or datasets. However, due to the grid structure discussed in this section, its performance in detecting small objects is not good enough. If an object is too tiny and the grid is not thin enough the detection is difficult. And if the grid is too thin the computation becomes too important.
In the next section (2.2.2), a different version of YOLO is addressed to be the object detector to use.

2.2.2 From YOLO to YOLOX

YOLO is an anchor-based object detector, which means that it uses predefined reference bounding boxes (anchors) to identify the locations and sizes of objects in an image. The anchors are generated at various locations and scales in the image and processed by a convolutional neural network (CNN) to

classify them as an object or as background and to refine the position and size of the bounding box. However, YOLOX [2] is an anchor-free detector. Anchor-free object detectors, directly predict the bounding boxes and class probabilities for objects in the image without using anchors.
Anchor-free object detectors are generally simpler, generalizable and, more flexible than anchor-based detectors. They do not require the generation and processing of a large number of anchors and can handle a wide range of object sizes and aspect ratios without the need to predefine the shapes and sizes of the anchors.
Flexibility is an important parameter for an embedded device. YOLOX is the detector used for experiments in this work.

2.2.3 YOLOX

2.2.3.1 General Description

YOLOX has been built using YOLOv3 [31] as a baseline with Darknet53 as the backbone. The loss used is a Binary Cross Entropy loss for the *confidence* and *classification* branches and an IoU loss (equation 25) for the *regression* branch, that predicts the bounding box coordinates. Then, for the head part, YOLOX proposes a different approach compared to other YOLOs architecture. A Decoupled Head architecture as presented in [2, Fig. 2] is used to overcome the conflict problem between classification and regression tasks. A diagram of the architecture is shown below.

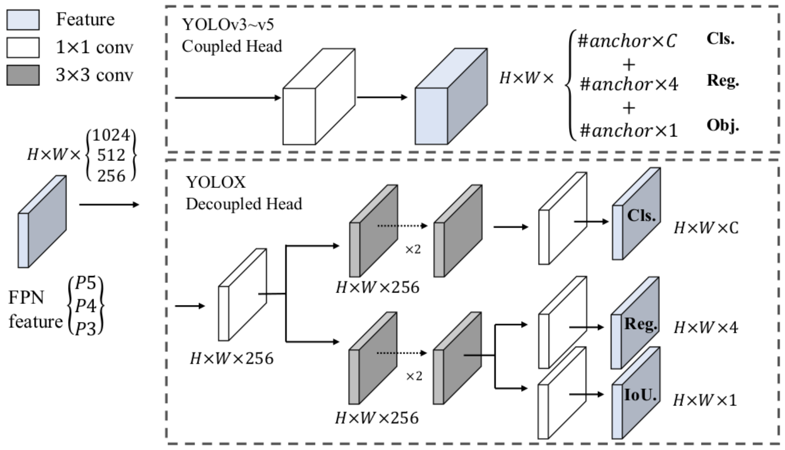


Figure 7: Illustration of the difference between YOLOv3 and YOLOX decoupled head architecture from [2].

2.2.3.2 Modifications

From this work, the input layer was modified for the experiments. Using the motion map result from section 2.1, the input layer had to be modified to take into account the motion map. On the first model, 1 channel was added for the motion map 6b, and on the second 3-channel for the trajectory map 6c. This was done by adding a 2D convolutional layer with uniform weight initialization, followed by batch normalization and a *silu* [35] activation. The results from the original input layer and the new one were then merged and passed through another 2D convolution layer to fit the right dimension for the remaining parts of the model.

Fine-tuning was employed for the training on the Analytics dataset. Fine-tuning is a process of training a pre-trained model on a new dataset. It allows the model to adapt to the new data and improve its performance for a specific task. This process is done by unfreezing a certain number of layers from the pre-trained model and training them with the new data while keeping the remaining layers frozen.
In the section 3, the results of motion detection and object detection are presented and discussed.

3 Experiments & Results

The experiments and results of 2.1 and 2.2 are presented in this section. The different setups for hardware and dataset are also addressed.

3.1 Motion detection Evaluation

In this section, the results of motion detection (2.1.5) and Otsu's thresholded motion map 2.1.7 are presented. The quantitative results are evaluated using three metrics, *precision*, *recall* and *F1-score*. Then, to visualize the results, qualitative results of the motion detection are presented. Experiments were made on two different well-known datasets for motion detection and change detection, which are *Davis* [1] and CDnet [36].
The quantitative results have been compared between the current work and the previous one. Only the binary motion map (section 2.1.7) is used, as it is the only result that can be compared, to the previous work results. After the presentation of the results, they are discussed.
Then, the results of the object detector are presented depending on the input data type, discussed in 3.2.2. The results are then discussed.

3.1.1 Motion Detection setup

The experiments were conducted in python on a laptop with an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz. Batch sizes of 15 frames were used to avoid overloading the CPU. The datasets for the experiments are briefly presented, and the metrics to evaluate the results are defined.

3.1.1.1 Dataset

The two datasets used for the evaluation of the motion detection, are *Davis* [1] and *CDnet* [36]. Each one provides different short video sequences captured by either moving cameras or jittering cameras. The aim is to compare the results of the PRCPA 2.1.5 to the results from the previous work 2.1.3.1.
Four videos are used to evaluate the model, two videos from *Davis* and two videos *CDnet*. The videos from *Davis* dataset show a simple movement of an object going from the right to the left and followed by the camera. The two videos from *CDnet*, show dynamic backgrounds with flowing water, waving trees and a camera jittering that have to be ignored. The motion to be detected has a less dynamic motion than the dynamic background.

3.1.1.2 Metrics

The results are evaluated using $precision$, $recall$ and, $F - measure$. These notions are defined below.

- **True Positives (TP)**: The model prediction matches the ground truth
- **False Positives (FP)**: The model predicted a label that does not match the ground truth
- **False Negatives (FN)**: The model does not predict a label, but a ground truth label exists
- **Precision (P)**: TP / (TP + FP)

- **Recall (R)**: TP / (TP + FN)
- ***F*-measure**: $2\,\frac{PR}{P+R}$

These notions allow knowing if the model produces too many False Positives or False Negative, and give an overview of the performances of the model on motion easily compensable.

3.1.2 Results

The results of the Panoramic Robust Principal Component Analysis are presented in this section. The quantitative results are presented first, and a comparison is made with the previous work results. Then, the qualitative results on the *Davis*, *CDnet* and *Analytics* datasets are shown.

3.1.2.1 Quantitative results

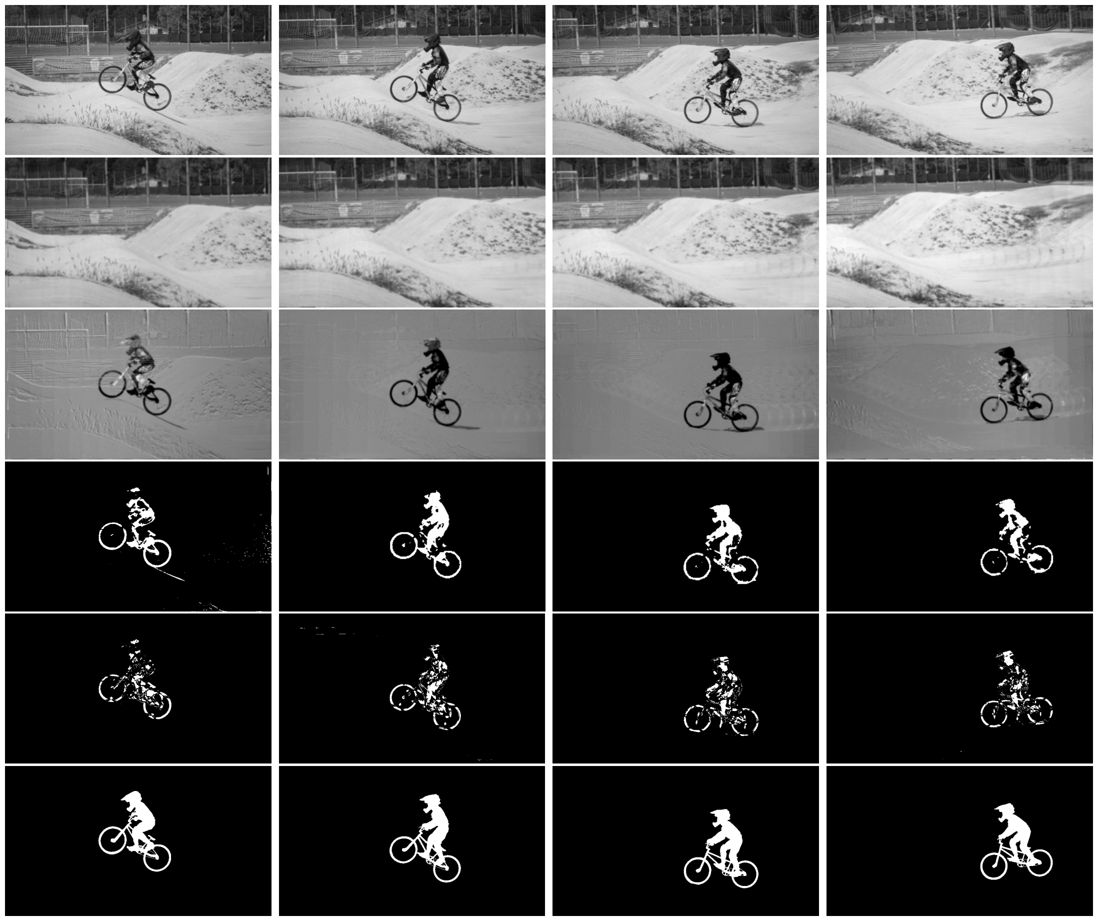
The quantitative results are displayed in table 1. Results are evaluated using $precision$, $recall$ and, $F - measure$. these notions are defined below. The results are discussed in the section 3.1.3.

Video informations				PRPCA			Baseline
Dataset	Sequence	Camera	Frames	Precision	Recall	F-measure	F-measure
Davis 2017	bmx-bumps	moving	40	0.78	0.72	0.75	0.53
Davis 2017	tennis	moving	35	0.89	0.54	0.67	0.45
CDnet 2014	canoe	jittering	80	0.25	0.26	0.22	0.11
CDnet 2014	fall	jittering	80	0.80	0.53	0.61	0.22

Table 1: Results of PRPCA and baseline model on 4 different videos.

3.1.2.2 Qualitative results

After presenting the quantitative results, the qualitative results are displayed below to visualise. The frist shown result is on *bmx-bumps* video from *Davis*. The second shown result in on two frames from the *drone dataset* provided by HENSOLDT Anlaytics presented in 3.2.2.



(a) frame 20 (b) frame 15 (c) frame 10 (d) frame 5

Figure 8: Results of PRPCA 2.1.5 motion detection for *bmx-bumps* video sequence (from right to left). 1st row: Orignal frame, 2nd row: Extracted background, 3rd row: Extracted motion, 4th row: Binary motion map, 5th row: Baseline model results, 6th row: Ground truth

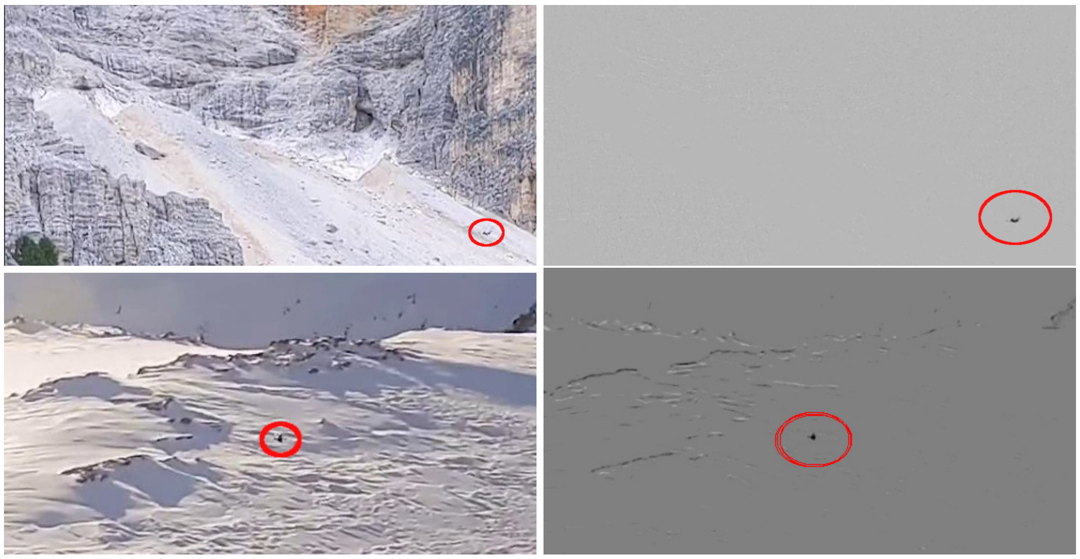
3.1.3 Discussions

Analyzing the results of these four videos, PRPCA approach improves the baseline algorithm results regarding the metrics used. However, some challenges, discussed in 2.1.1.1, remain unsolved.

- **Clutter**: It appears mainly when the videos present moving trees or flowing water. Of course, as motion detection model is used, moving trees are classified as moving objects even though trees are part of the background. Then, dealing with the camera motion and at the same time dynamic background is difficult and the results for some videos is not perfectly accurate. Machine learning approaches [37] propose to solve the problem of clutter, but it does not fit to the requirements 2.1.1.

- **Chasing a target**: This case is particularly interesting because one of the HENSOLDT use case is the drone chasing. However, as in this situation, the drone flies fast, and a lot of pixels specifically around the borders are blurry. This leads to a less accurate homography estimation and thus a less accurate motion compensation.

In the section 3.2, the experiments on the object detection are presented.



(a) RGB frame (b) Sparse matrix from PRPCA

Figure 9: Results of PRCPA on two different videos

3.2 Object Detection Evaluation

In this section, the experiments carried out by YOLOX are presented. Thus, using the results of PRPCA, new input data are generated. These are added to the original RGB frame. The aim is to show that this addition of information to the detector input improves its detection performance.
The different input types are presented in section 3.2.2, when the metrics used are presented in 3.2.3

3.2.1 Object Detection setup

The experiments were conducted in python on a laptop with an NVIDIA GeForce RTX 2070 Mobile. Batch sizes of 8 frames were used to avoid memory problems. The dataset for the experiments is briefly presented, and the metrics to evaluate the results are defined.

3.2.2 Dataset

The dataset used for the experiments, is given by HENSOLDT Analytics. It is composed by drone videos taken by moving camera in different situation. The drone can fly fast of be relatively stable. However, on the videos the drone are quite small and challenging to detect. Thus, the different input data are presented in section 3.2.2.1, and a split in the dataset is explained is section 3.2.2.2.

3.2.2.1 Different inputs

The training data is organized in four different possibilities. Thus it is possible to verify, if the addition of new data to the original one helps the detector to perform better.

- **3-channels** data is the original RGB frame, which is normally used to train an object detector;

- **4-channels** data represents the addition of the original RGB frame and either the raw motion map resulting from the PRPCA algorithm presented in section 2.1.5 or binary motion map presented in section 2.1.7;
- **6-channels** data is defined by the addition of the original RGB frame and the trajectory map presented in section 2.1.8.

3.2.2.2 Data split

For the experiments, the idea is not to retrain YOLOX from scratch but to fine-tune it with pre-trained weights, to gain time and performance. YOLOX has been trained on thousands of images from COCO or PASCALVOC datasets for example. So, to fit the model to our data, the weights of the pre-trained YOLOXs model are used.
Also, it was decided to split the dataset, provided by Analytics, into two datasets for the first experiment. The first dataset contains videos where the camera motion is easily compensated (slow motion of the camera) and, the second dataset contains videos where the camera motion is more difficult to compensate, due to abrupt movement changes, or high-speed movements for example. In the second stage, the two datasets are coupled and fine-tuning is performed. This will help to find if the camera motion has a great impact on the detection of drones. Thus, in the two datasets, there are dozen of different videos, and each video has around 50 frames. Then, one video is selected in each dataset (slow and high motion) for the validation step. To get accurate results, the validation videos have to have pixel values that belong to the pixel values distribution of the training videos. The next section (3.2.3) presents the metrcis used for the evaluation.

3.2.3 Metrics

In COCO 2017 challenge [4], the results are evaluated using mAP or *mean Average Precision*. This is made by averaging the AP over the classes of a dataset and over 10 different IoU thresholds from 0.5 to 0.95 with a step of 0.05. In this case of stude, the work only focus on drone detection, as the dataset is only made of drone videos. The object detection task localizes objects using bounding boxes associated with their confidence score to report the certainty of the bounding boxes of the detected objects. So, the Intersection over Union (IoU) is introduced to quantify how many objects are correctly detected and how many false positives are generated by the model. Before going further, IoU is introduced.

$$IoU = \frac{\text{Intersection of the predicted and true bounding box}}{\text{Union of the predicted and true bounding box}} \tag{25}$$

Then, mAP or as the COCO challenge defines it $AP^{.5:.05:.95}$ is computed following three steps. Firstly, an *average precision* is calculated for the different IoU thresholds. Then, the *precision* is calculated for every *recall* value from 0 to 1, and for each threshold value. To finish, the *average* is taken over all the classes, and over all the IoU thresholds. This is only for the primary challenge metric. The second metric used to evaluate the model is the mAR *mean Average Recall* or $AR_{small}^{.5:.05:.95}$ for the COCO challenge. The idea is to compute *recall* over the different IoU threshold values. Then, AR is defined as 2 times the area under the *recall-IoU* curve for one at a time. Then, mAR is computed by averaging all the AR by the number of classes. In this work, there is only one class which is the drone class. So, $AP^{.5:.05:.95}$ and AR_{small} are the metrics used for the evaluation.

3.2.4 Results

This section presents the quantitative results obtained by the object detector on the different datasets and for the different types of inputs. The metrics used have been presented in section 3.2.3.
The data presented in Table 2 permit several inferences to be made. Specifically, when analyzing the dataset labeled as *slow motion*, it is apparent that the inclusion of additional information in the input of the model leads to a corresponding enhancement in the performance of the detector.
The results demonstrate that the use of additional information in the form of raw motion, binary motion, or trajectory maps improves the performance of the detector when applied to single RGB input images. Furthermore, the combination of RGB and binary motion map inputs yields the highest level of accuracy. This can be attributed to the nature of the *slow motion* dataset, which is characterized by sparse PRCPA motion detection (7). As a result, the creation of binary maps is relatively simple and highly accurate. In other words, the location of the UAV is clearly evident in each image, making it more readily identifiable by the detector.

Upon examination of the *high motion* dataset, it is noteworthy that the outcomes diverge significantly from those observed in the "slow motion" dataset. Specifically, the use of single RGB input images yields the most favorable results in the *high motion* dataset. This discrepancy can be attributed to the increased noise and extraneous information present in the motion and trajectory map data, resulting in a negative impact on the performance of the model when additional information is provided. However, it is important to note that the values of AP and AR have not decreased significantly when compared to the results obtained in the *slow motion* dataset. This indicates that the detector's performance remains satisfactory. A particularly noteworthy result is the significant improvement in the RGB-only input, which can be explained by the fact that in the "slow motion" dataset, the smaller size of the drones makes them more difficult to detect, while in the *high motion* dataset, the larger presence of the drones on the frame makes them more easily identifiable. In conclusion, it is crucial to ensure that any additional information provided to the detector is both relevant and accurate to enhance the detector's performance.

Finally, the results obtained from the *coupled* dataset are also noteworthy. The inclusion of motion and trajectory map information leads to an improvement in performance when compared to the use of single RGB input images alone. Additionally, the AP and AR values for the RGB-only input are lower, indicating that there are either a significant number of false positives or false negatives present. A particularly interesting outcome is that the combination of RGB and trajectory map inputs results in the highest level of accuracy, which was not the case previously.

Dataset	Input	$AP_{drone}(\%)$	$AR_{drone}(\%)$
Slow motion camera	RGB	49.7	53.2
	RGB - Raw motion	52.3	54.2
	RGB - Binary motion	**58.4**	**62.0**
	RGB - Trajectory	54.5	57.0
High motion camera	RGB	**56.4**	**61.0**
	RGB - Raw motion	55.7	58.0
	RGB - Binary motion	48.1	52.1
	RGB - Trajectory	52.2	54.0
Coupled dataset	RGB	45.6	49.8
	RGB - Raw motion	51.2	55.7
	RGB - Binary motion	49.7	52.0
	RGB - Trajectory	**53.7**	**56.3**

Table 2: YOLOX results on different datasets (Analytics)

4 Conclusion

In this work, the goal of the research was to examine the impact of incorporating additional information into the input of an object detector on its performance. To accomplish this, different method for motion detection were investigated trough a litterature review 2.1.2. A panoramic view-building approach 7 was used to cancel out the ego-motion of the camera capturing the scene. Next, a low-rank and sparse decomposition of the data matrix containing the compensated motion frames, using the Robust Principal Component Analysis [21] algorithm, was employed to extract motion information and highlight potential moving objects present in the video sequence. These motion maps were then used to train an object detection model, YOLOX, with four different inputs.

The results obtained from the motion extraction process demonstrated an improvement in the $F1 - score$ compared to previous work, as evaluated on the Davis challenge and ChangeDetection datasets. However, when applied to the drone dataset provided by Analytics, which includes videos with high camera motion, the method sill encountered some unsolved challenges.

The results of the object detection showed an improvment of the detection when additionnal informations were provided to the model. This confirms the previous work result concerning the object detection.

Future work includes exploring methods to compute a more accurate transformation between consecutive frames, as the current method is not precise enough, particularly when dealing with high camera motion. Additionally, the motion extraction model performed well for slow camera motion and provided accurate motion maps.

On a personal level, the internship was a valuable learning experience in which the author was able to develop skills in data science and computer vision, improve communication and English proficiency, and gain exposure to different working styles.

References

[1] DAVIS: Densely annotated VIdeo segmentation. [Online]. Available: https://davischallenge.org/

[2] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021." [Online]. Available: http://arxiv.org/abs/2107.08430

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection." [Online]. Available: http://arxiv.org/abs/1506.02640

[4] COCO - common objects in context. [Online]. Available: https://cocodataset.org/#home

[5] The PASCAL visual object classes homepage. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/

[6] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," vol. 74, no. 1, pp. 59–73. [Online]. Available: https://link.springer.com/10.1007/s11263-006-0002-3

[7] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*. IEEE, pp. 1150–1157 vol.2. [Online]. Available: http://ieeexplore.ieee.org/document/790410/

[8] S. Lee. Understanding homography (a.k.a perspective transformation). [Online]. Available: https://towardsdatascience.com/understanding-homography-a-k-a-perspective-transformation-cacaed5ca17

[9] B. E. Moore, C. Gao, and R. R. Nadakuditi, "Panoramic robust PCA for foreground-background separation on noisy, free-motion camera video." [Online]. Available: http://arxiv.org/abs/1712.06229

[10] Optical flow in OpenCV (c++/python) | LearnOpenCV #. [Online]. Available: https://learnopencv.com/optical-flow-in-opencv/

[11] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision."

[12] baeldung. Harris corner detection explained | baeldung on computer science. [Online]. Available: https://www.baeldung.com/cs/harris-corner-detection

[13] G.-W. Yuan, J. Gong, M.-N. Deng, H. Zhou, and D. Xu, "A moving objects detection algorithm based on three-frame difference and sparse optical flow," vol. 13, no. 11, pp. 1863–1867. [Online]. Available: https://scialert.net/fulltext/?doi=itj.2014.1863.1867

[14] Prantik. Background extraction from videos using gaussian mixture models. [Online]. Available: https://medium.com/@prantiksen4/background-extraction-from-videos-using-gaussian-mixture-models-6e11d743f932

[15] P.-L. St-Charles, G.-A. Bilodeau, and R. Bergevin, "SuBSENSE : A universal change detection method with local adaptive sensitivity." vol. 24.

[16] Y. Sheikh, O. Javed, and T. Kanade, "Background subtraction for freely moving cameras," in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, pp. 1219–1225. [Online]. Available: http://ieeexplore.ieee.org/document/5459334/

[17] A. Elqursh and A. Elgammal, "Online moving camera background subtraction," in *Computer Vision – ECCV 2012*, ser. Lecture Notes in Computer Science, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Springer, pp. 228–241.

[18] Y. Zhu and A. Elgammal, "A multilayer-based framework for online background subtraction with freely moving cameras." [Online]. Available: http://arxiv.org/abs/1709.01140

[19] Markov random fields - an overview | ScienceDirect topics. [Online]. Available: https://www.sciencedirect.com/topics/mathematics/markov-random-fields

[20] homo. Wayback machine. [Online]. Available: https://web.archive.org/web/20171226115739/https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec04.pdf

[21] E. J. Candes, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" [Online]. Available: http://arxiv.org/abs/0912.3599

[22] flann. OpenCV: Feature matching with FLANN. [Online]. Available: https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

[23] OpenCV: Basic concepts of the homography explained with code. [Online]. Available: https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html

[24] D. Forsyth and J. Ponce, *Computer vision: a modern approach*, 2nd ed. Pearson.

[25] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," vol. 24, no. 6, pp. 381–395. [Online]. Available: https://doi.org/10.1145/358669.358692

[26] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (PCA)," vol. 19, no. 3, pp. 303–342. [Online]. Available: https://www.sciencedirect.com/science/article/pii/009830049390090R

[27] G. Golub, A. Hoffman, and G. Stewart, "A generalization of the eckart-young-mirsky matrix approximation theorem," vol. 88-89, pp. 317–327. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0024379587901145

[28] S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," vol. 3, no. 1, pp. 1–122. [Online]. Available: http://www.nowpublishers.com/article/Details/MAL-016

[29] N. Otsu, "A threshold selection method from gray-level histograms," vol. 9, no. 1, pp. 62–66, conference Name: IEEE Transactions on Systems, Man, and Cybernetics.

[30] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger." [Online]. Available: http://arxiv.org/abs/1612.08242

[31] ——, “YOLOv3: An incremental improvement.” [Online]. Available: http://arxiv.org/abs/1804.02767

[32] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection.” [Online]. Available: http://arxiv.org/abs/2004.10934

[33] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, “YOLOv6: A single-stage object detection framework for industrial applications.” [Online]. Available: http://arxiv.org/abs/2209.02976

[34] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.” [Online]. Available: http://arxiv.org/abs/2207.02696

[35] S. Elfwing, E. Uchibe, and K. Doya, “Sigmoid-weighted linear units for neural network function approximation in reinforcement learning.” [Online]. Available: http://arxiv.org/abs/1702.03118

[36] Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, “CDnet 2014: An expanded change detection benchmark dataset,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, pp. 393–400. [Online]. Available: https://ieeexplore.ieee.org/document/6910011

[37] Z. Zhang, Y. Chang, S. Zhong, L. Yan, and X. Zou, “Learning dynamic background for weakly supervised moving object detection,” vol. 121, p. 104425. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885622000543

5 Appendix

5.1 Alternating Direction Method of Multipliers

The principles of the ADMM is described in this section. It is presented in [28, section 3], as a method for convex optimization.
Thus, let P be a convex optimization problem defined as

$$P = \begin{cases} \min\limits_{x,y} f(x) + g(y) \\ \text{with } Ax + By = c \end{cases} \tag{26}$$

where it is assumeed that f and g are convex, with variables $x \in \mathbf{R^n}$, $y \in \mathbf{R^m}$ and $A \in \mathbf{R}^{p\times n}$, $B \in \mathbf{R}^{p\times m}$ and $c \in \mathbf{R^p}$.
The optimal value p^* of the problem is

$$p^* = inf\{f(x) + g(y)|Ax + By = c\} \tag{27}$$

Then, the augmented Lagrangian is defined as

$$\mathcal{L}(x, y, z) = f(x) + g(y) + z^T(Ax + By - c) + \frac{\mu}{2}||Ax + By - c||_2^2 \tag{28}$$

The Alternating Direction Method of Multipliers consists in updating x, y, z following these three update equations

$$x^{k+1} = \arg\min_x \mathcal{L}_\mu(x, y^k, z^k) \tag{29}$$

$$y^{k+1} = \arg\min_y \mathcal{L}_\mu(x^{k+1}, y, z^k) \tag{30}$$

$$z^{k+1} = z^k + \mu(Ax^{k+1} + By^{k+1} - c) \tag{31}$$

where $\mu > 0$. Thus, primal and dual residuals are introduced, r^k and h^k respectively defined as

$$\begin{aligned} r^k &= Ax^k - By^k - c \\ h^k &= \mu_k A^T B(y^k - y^{k+1}) \end{aligned} \tag{32}$$

The ADMM algorithm can be stopped when both primal and dual residuals are below certain quantities such as

$$\begin{aligned} ||r^k||_2 &\leq \epsilon^{primal} \\ ||h^k||_2 &\leq \epsilon^{dual} \end{aligned} \tag{33}$$

where $\epsilon^{primal} > 0$ and $\epsilon^{dual} > 0$. To finish, the update of The Lagrangian multiplier is computed as described below

$$\mu_{k+1} = \begin{cases} \rho\mu_k & \text{if } ||r^k||_2 > \tau||h^k||_2 \\ \mu_k/\rho & \text{if } ||h^k||_2 > \tau||r^k||_2 \\ \mu_k & \text{otherwise} \end{cases} \tag{34}$$

where $\tau > 1$.