UCLouvain

École polytechnique de Louvain

Smart Crossroad Surveillance in Urban Areas

Author: **Daniel Defrenne**
Supervisor: **Benoît Macq**
Readers: **Karim El Khoury, Jean-Didier Legat, Sébastien Lugan**
Academic year 2020–2021
Master [120] in Electrical Engineering

Contents

Introduction

Nowadays, compression algorithms are everywhere. Without them, we would be unable to save the same amount of data as we are saving today. Unfortunately, to view or fully understand the data, the need of decoder is mandatory. For videos, it is exactly the same. While being small in size, we can still enjoy them even if the resolution goes up to 4K.

Detection and tracking in decoded videos already exist and they perform quite well. But what if we could skip the step of decoding the video ? Could algorithms reach better performances ? What about the privacy of the processed data ?

Why would we use compressed video data?

There are multiple reasons why someone would try to use encoded video data. Less dataflow means less instructions, even memory, but in this thesis, the privacy of the surroundings will be the main focus. Today, privacy is at the centre of the attention. With data leaks happening everywhere, it would be unfortunate to let this happen to a camera system. But what happens if a data-leak contains partial encoded data ? The advantage here is that without having the full encoded data, it is nearly impossible to reconstruct the initial decoded video. In urban surveillance, detail is key for identification, but this detail becomes less pertinent for traffic analysis. It would also be more compliant to the General Data Protection Regulation (GDPR) [1]. Even if a hacker, as a man in the middle, recovers this partial data, he would have a hard time identifying anything.

This thesis will be split into different parts:

- The first part will be about how the video compression works and how it is performed to gather the necessary encoded data.
- The second part will be about detection and a focus will be brought more on privacy by removing a whole part of the compression data.
- The last part will study different tracking algorithms. The main focus will be on the usability of the motion vectors, which is a data type from encoded videos.

Chapter 1

Objective of this thesis

The objective of this thesis is to demonstrate if the available information after compression is enough to apply detection and tracking algorithms. The main focus will be the detection and tracking of vehicles on roads.
For the detection, two approaches will be analysed. Both will respectively remove a data type from the encoded videos, one removing the motion vectors and the other the residuals[1].
As for the tracking, two well known algorithms, IOU [2] [3] and Kalman IOU [4], will be used and one of them will be modified such that its tracking behaviour is adapted with the motion vectors.
If both detection and tracking can ignore the residuals, it could be removed completely and thus gain the advantage of privacy. If not, such that both data type give a great advantage to the detection and tracking quality, then it will be proven that the need of decoding videos could be skipped.

1.1 Dataset

The used dataset in this thesis is the same dataset that was used during the TRAIL. A workshop that took place in September 2020 [5]. AIC21 [6] benchmark is the dataset that will be used for the development of this work. This include the main videos and their ground truth position and identification number.

1.2 Metrics for the detection

For the detection, two metrics [7] will be used. The first one is precision (Eq. 1.1). It will count how much the detected bounding boxes, boxes which contains a detected object, compares to the ground truth. When a bounding box finds its matching ground truth, it will be considered as a true positive (TP), else it will be a false positive (FP). The precision will afterwards be computed using the following formula:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1.1}$$

[1] More details about what motion vectors and the residuals can be found in Chapter 2

The second metric is the recall (Eq. 1.2). It will look at the ground truth and compare it to the detected bounding boxes. When a ground truth finds a matching detected bounding box, it will be counted as a true positive (TP). This true positive should be equal to the amount of true positive found while computing the precision. If the ground truth bounding box does not find any match, it will be counted as a false negative (FN). The recall will then be computed with this formula:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1.2}$$

Both metrics will give a score between 0 and 1, with 1 the best score possible.
The reason to choose these metrics is that they both give a good idea of the quality of detection. A bad precision means the detections are not well placed or are bad detections (false positives). The recall gives an idea of how much ground truth has been detected. Therefore, a bad recall means the detections are missing lots of ground truths.

1.3 Metrics for the tracking

1.3.1 MOTA

To benchmark the tracking, the Multiple Object Tracking Accuracy (MOTA)[8] metric (see Equation 1.3) will be used.

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t} \tag{1.3}$$

Where GT_t is the amount of ground truth boxes per frame, FN_t the false negatives per frame, FP_t the false positives per frame and $IDSW_t$ the amount ID switches per tracking object.

This metric goes from 1 to $-\infty$. When the errors made by the tracker are greater than the amount of ground truth objects, the value of MOTA becomes negative.

Chapter 2

Selected model for the compression

The model that is used in this thesis is a simplified version of today's compression algorithms. There is no loss of data, contrary to High Efficiency Video Encoding (HEVC) [9] encoder.

2.1 Video compression

A video has redundancy, and video compression algorithms [10] take advantage of this redundancy to reduce the size of the video. There are two types of redundancy in a video:

- Spacial redundancy in each frame
- Temporal redundancy between consecutive frames

Image compression algorithms focusses on the spacial redundancy only. This spacial redundancy is also available in videos, since a video is a sequence of images. But using only spacial redundancy does not allow videos to reach a high compression ratio. However since a video is made of a sequence of images where the differences between two frames are small, the temporal redundancy is very high too. Therefore, video compression uses previous frames to predict the new frame, and using this prediction, the algorithm reduces the total amount of stored data, by removing this temporal redundancy. It will store the error produced by the prediction (residuals) and the necessary data to recreate this prediction (motion vectors).

2.1.1 Video encoding

The video encoder is the complex part of the compression of the data. Its first step is to remove a predicted frame from the actual frame. This results in an error frame, the residuals, which will be image-encoded before it is saved or sent to the decoder.

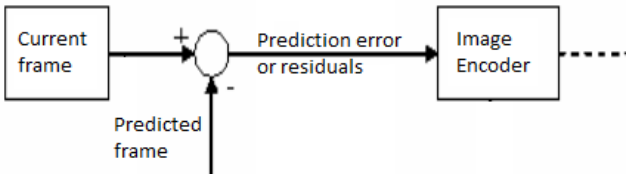


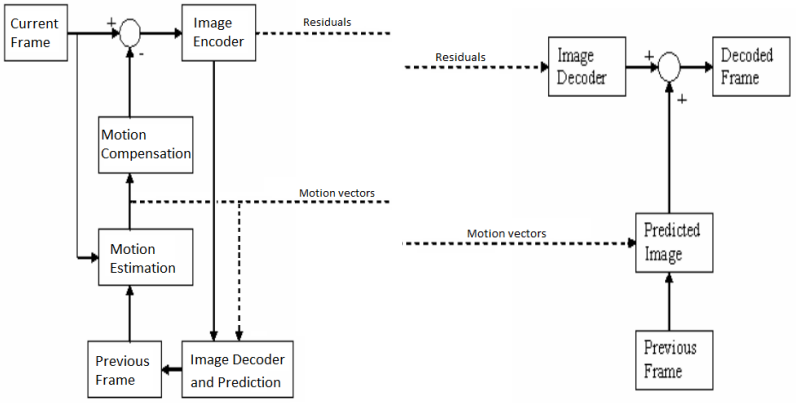
Figure 2.1: Encoder basic idea

The question resides in, how to create this predicted frame ?

When predicting a frame, the encoder and decoder need to use the same information. If they do not, both input and output will differ. Therefore, it is crucial that they both have the same previous frame stored in memory. To be sure this happens, the encoder will not use the input video to recover the previous frame, but will recreate the previous frame $(i-1)$ as the decoder would. It will do so by using the residuals, previous motion vectors and the $i-2$ frame stored in its memory, the same information the decoder has. Once the previous frame has been recreated by the encoder, the algorithm will do the motion estimation. It splits the image in various blocks. These blocks can be of same size or they can use various sizes as will be explained in Section 2.2. Once the image separated in different blocks, the algorithm will estimate for each of them motion vectors using block matching algorithms (see Part 2.3.1).
Finally, the algorithm will use these motion vectors to predict the actual frame using motion compensation. These motion vectors are also stored and sent to the decoder. Then, the obtained prediction is removed from the new frame, and a new residual is created. This will continue until all frames are encoded.

2.1.2 Video decoding

The *image decoder and prediction* of the encoder is the same as what the decoder does. The encoder receives two data types, the residuals and the motion vectors. The residuals will immediately be image-decoded. The motion vectors will, with the help of the previously stored frame, recreate the prediction by splitting the frame in the same blocks as the video encoder has done and then apply motion compensation. Motion compensation uses the motion vectors to move parts of the previous frame, copy pasting the blocks following the motion vector's direction, and thus predicting the actual frame. The obtained prediction is afterwards added to the image-decoded residuals to obtain the new frame. This frame will then be stored as previous frame in this algorithm.



(a) Video encoder functional blocks

(b) Video decoder with functional blocks

Figure 2.2: Video encoder and decoder functional blocks[11]

2.2 Adaptative block matching algorithm (ABMA)

The video compression algorithm explained before can use a predetermined or adaptative block size. When using predetermined block sizes, it will bring some big drawbacks when the size is incorrectly chosen for a specific situation. A block size too big results in a poor compression capability. This is due to the fact that if a small part of a block is moving, but not the background, the predicted frame will rather stay static since it will result in less residuals than if the block was moved. This problem can be seen in Figure 2.3. The motion vector estimation will use a motion vector that reduces the error to the minimum possible. In this figure, two possibilities are shown, the one where the motion vector stays static, the other where the motion vector follows the object. The more orange the image has, the greater the prediction error will be, thus resulting in more residuals. The motion estimator will thus choose the left one and will avoid the right one. This is an example of what is called a blocking artefact created by using too big box sizes.

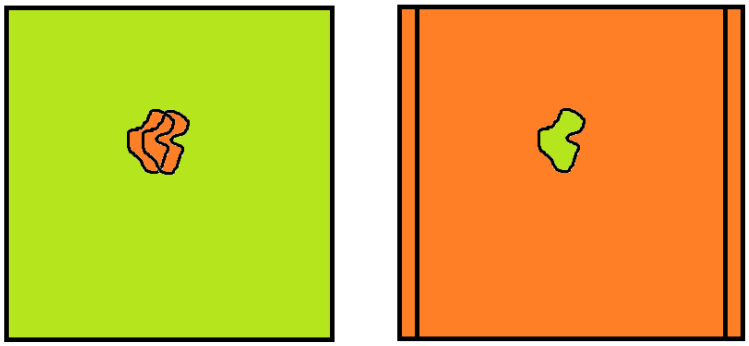
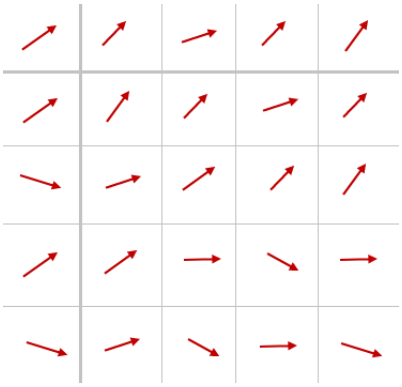


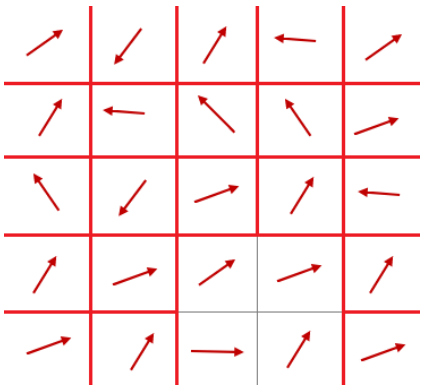
Figure 2.3: Images with no motion vector and with motion vector applied, left and right respectively

When a block size is too small, it will suffer from noise. Since noise is predominant in this case, the motion vectors are random. This is because of the different block matching algorithms' hypothesis, their behaviour will depend on noise, therefore not always finding the most optimal motion vector.

Using adaptative block size is necessary for better motion estimation and final compression ratio. The Distributed adaptative block matching algorithm (Dis-ABMA) [12] proposes a solution to estimate the block size by looking at the homogeneity of the motion vectors. When motion vectors tend to go in the same direction, it is more interesting to keep big boxes as shown in Figure 2.4a. When vectors are more random, smaller boxes would lead to better use of the detail of the motion and therefore give a better compression rate (see Figure 2.4b).
More detail of the algorithm and its results can be found in the paper Dis-ABMA[12].



(a) Homogenous vectors should take advantage of bigger boxes



(b) Less homogenous vectors should split in smaller boxes to have a better estimation of the motion vectors

Figure 2.4: Red contoured boxes will be splitted further, while ones with gray will not

2.3 Final model

The model used in this thesis bases itself on the Dis-ABMA [12] algorithm and adapts it slightly. It was adapted such that the algorithm could fully use the parallelisation of processes. It means that multiple computations can be run simultaneously.
A first change is that blocks are not broken down by comparing the motion vectors with their neighbours. It is broken down by comparing the absolute differences of the four sub-blocks of a block. If the variation of the absolute differences is too big between the four subblocks, the algorithm will break the block in four pieces. This has the advantage of it being faster to compute, but this does not take advantage of the homogeneity of the vectors. But since in this thesis, the goal is not to reach the best compression ratio, but rather to recover an estimation of the residuals and motion vectors, this difference of compression can be overlooked[1]. Here is a pseudocode of the algorithm which explains how it works:

[1]This algorithm could have been improved by computing the motion vector before computing the subboxes error and variance. This would have taken the vector information into account in the breaking process of the boxes. The drawback would be its execution time...

Algorithm 1: ABMA function

```
Inputs:
    previousImage, actualImage, boxPositionX, boxPositionY, boxSize
if boxSize % 2 == 0 && boxSize > 4 then
    Compute SubBoxes error & variance
    if variance > sensitivity then
        thread ABMA(boxSize/2, boxPositionX, boxPositionY)
        thread ABMA(boxSize/2, boxPositionX + boxSize/2, boxPositionY)
        thread ABMA(boxSize/2, boxPositionX, boxPositionY + boxSize/2)
        thread ABMA(boxSize/2, boxPositionX + boxSize/2, boxPositionY +
         boxSize/2)
    else
        Vectorize(boxPositionX, boxPositionY, boxSize)
    end
else
    Vectorize(boxPositionX, boxPositionY, boxSize)
end
```

With boxSize being the size of the processed box, boxPosition is the box's upper left position and actualImage and previousImage are RGB integers of the actual and previous image respectively.
The vectorize function is a function which will recover the motion vectors of a block using block matching algorithms.
The ABMA function only looks at its block and no other. Therefore, it is this function that is fully parallelised.

2.3.1 Block matching algorithms

There exists multiple block matching algorithms [11] to find the best motion vector. The simplest, but the slowest is the exhaustive search. But better performing algorithms do exists. All of these algorithms, except the exhaustive search, uses the hypothesis that the error function has one and only one minimum and that the slope going towards this minimum is always negative.

Exhaustive search

The exhaustive search, as its name implies, will check every possible motion vector and recover the best one of them. To avoid doing all the possible motion vectors of an image, the search is limited inside an interval which is a square of size n. The search cost is thus equal to $\mathcal{O}(n^2)$ times the complexity of the motion vector's evaluation. This search method is avoided due to its high time complexity.

Three step search (3SS)

The three step search is a block matching algorithm which works in three steps. First, it computes the matching error of nine vector points, each equally spaced with a spacing of 4. The one that matches the most, the one with the smallest error will be selected. The

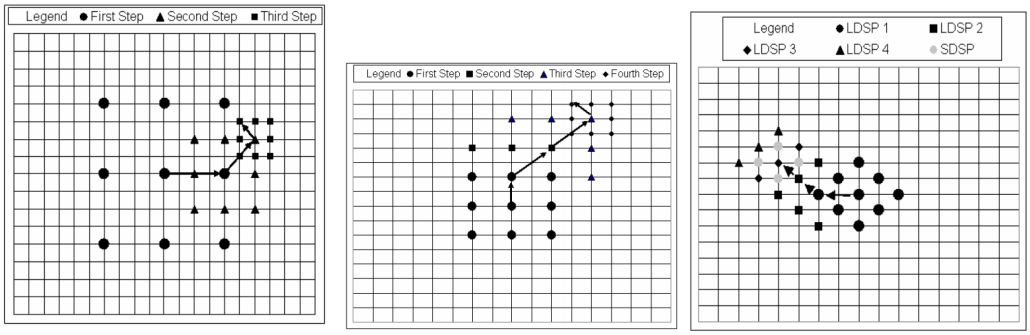
algorithm will then change its origin to the selected point. This step is repeated twice, but the space between each point will be reduced by a factor of two. Once it reaches the spacing of one, the algorithm's final origin will be the motion vector.

Four step search (4SS) [13]

The four step search is not as its name could imply, an extended three step search. The four step search will start by computing nine points with a fixed spacing of 2. Once it has selected the one which matches most, it will change its origin to that point. If the new origin is the same as before, the algorithm jumps immediately to step 4. If not, then depending on the new origin, it will compute the 3 or 5 new points. These points are spaced equally with the same spacing than the first step. It will do this a third time if the second step does a change of origin. In the fourth and last step, it will compute the eight neighbours of the actual origin but with a spacing of one. The point which matches most will be the motion vector.

Diamond search (DS) [14]

The diamond search is an algorithm which works exactly as the four step search algorithm, except that its search pattern is changed from a square to a diamond. There is also no limit to the number of steps it can take. It will use a diamond like pattern and change its origin accordingly to the best matched point. Once the origin does not change anymore, it will go to its last step. This time, the search pattern is a smaller diamond.



(a) 3SS (b) 4SS (c) DS

Figure 2.5: Block matching algorithms [11]

Used block matching algorithm

The used block matching algorithm is the three step search but with an extra step and a different space reduction factor between each step. The first step has its points spaced of 27 pixels, the second step is spaced of 9 pixels, the third 3 and the last one 1. This is a simple and quite efficient way to apply the block matching.

2.4 Block size adaptation

To show how the adaptative block matching algorithms behaves with noise, it will be tested with frames with no noise and a big amount of noise. The used algorithm here is the one used in the final model. Figure 2.6 shows that more noise means less organised

(a) No noise added to the image (b) Small amount of noise added to the image (c) Big amount of noise added to the image

Figure 2.6: Block sizes, darker means smaller block

Figure 2.7: Image of the situation before compression [6]

block splitting. With no added noise, the image is mostly split depending on the image information. The more noise is added, the more it will split the boxes randomly. Once a block has been split once, the probability of splitting it again becomes higher, reducing the amount of intermediate sized blocks.
A proof of this concept is given in the Appendix A.

Chapter 3

Detection algorithms

To fulfill the objective of this thesis, the next step is to verify the feasibility of detecting vehicles driving through the intersection. To do so, the motion vectors and frame's residuals will be tested individually. Both have the same disadvantage; non-moving vehicles are not detectable due to the compression which removes spacial and temporal redundancy.

3.1 Detection using the motion vectors

If the detection is possible using only the motion vectors, this means that the residuals will not be necessary to accomplish any parts of the objectives. This could allow the removal of the residuals from the algorithm procedure and thus harden the task of recreating the original images, making them more private.

3.1.1 Algorithm principle

The algorithm works as presented in Figure 3.1. The only used input here are the motion vectors. They will first pass through a temporal filter to remove random motion vectors appearing in some frames 3.2b. The remaining vectors are then clustered in a 2D plane. With these clusters, the bounding boxes will be created.

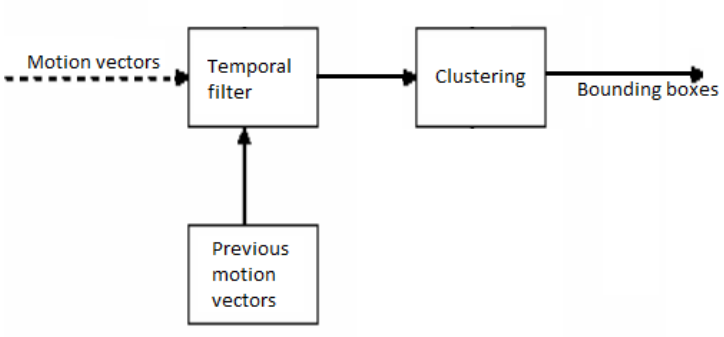
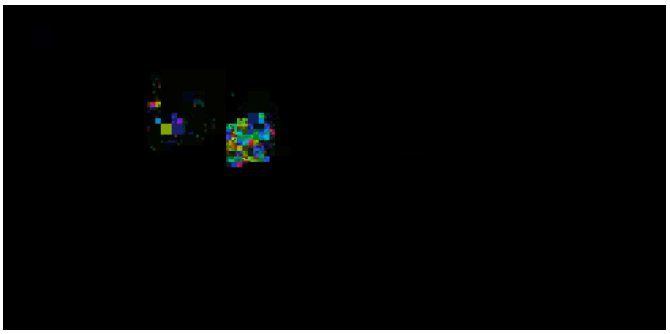


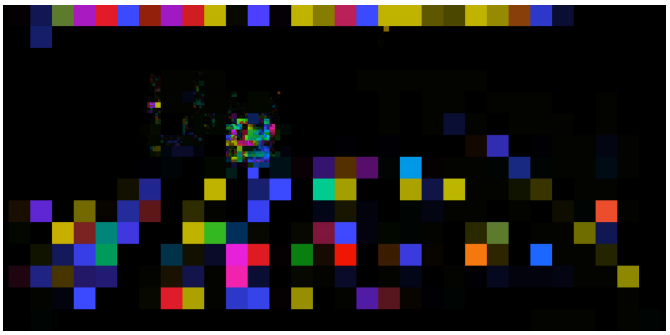
Figure 3.1: Proposed algorithm

3.1.2 Filtering the vectors

The used cameras, as most cameras are not static. Wind can make small movements of the camera, and those movements sometimes create unwanted motion vectors. But even if they are static, the encoder encodes some frames where lots of random motion vectors appear. It is for this reason that a temporal filter is necessary. This filter should not only remove unwanted vectors, but will also thin the clusters of motion vectors, which will give an advantage to the clustering function. As can be seen on Figure 3.2, some frames



(a) No random motion vectors



(b) Random motion vectors

(c) Color wheel indicating the direction of the vectors [15]

Figure 3.2: Comparison between an expected frame and the random ones that sometimes appear

(Fig. 3.2b) have unwanted motion vectors appearing. Figure 3.3 shows the results of the temporal filter. To understand how to observe the images, the colour intensity gives back the amplitude of the vector, while the colour gives the direction as the colour wheel in Figure 3.2c represents.

3.1.3 Clustering of the vectors

When the motion vectors are filtered, the next step is to cluster them in groups. The clustering method is a special type of spectral clustering, known as DBSCAN (Appendix B). The only parameter of this algorithm is ϵ, which is the maximal distance between two points to which the DBSCAN will consider it to the same group. This parameter has been set to 2. The smaller this value, the more the algorithm will cluster groups that are not close to each other, it will be more strict. This will lower the cluster size and create small new clusters which should be avoided. While a bigger number could merge two clusters that should be separated.

Figure 3.3: Vectors after they have been filtered through the temporal filter

Once the vectors are clustered, they will be transformed in a bounding box only if their size surpasses the size of 200 vectors. If the size is not bigger than 200, the cluster is simply ignored, it is considered as a false positive. This size of 200 has been chosen such that it minimises the false positives and negatives. The advantage of doing this is that it avoids unnecessary bounding boxes of isolated motion vectors, but has the disadvantage that it removes true positives when the vehicle is too far and the cluster therefore becomes too small. Figure 3.4 shows the result of the clustering.

When the bounding boxes are available, it is possible to compute the precision and the

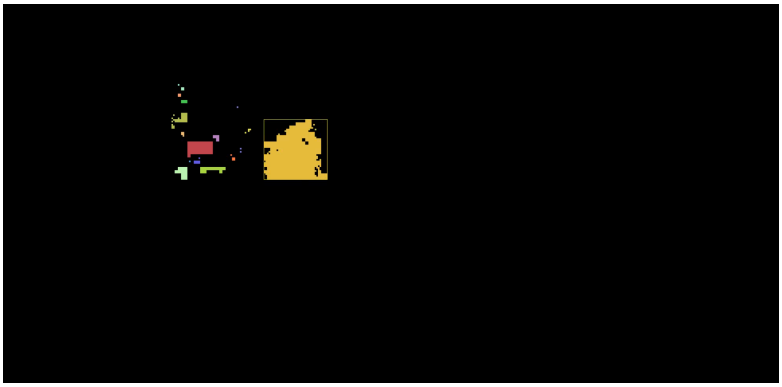


Figure 3.4: Vector clustering after a DBSCAN & a found bounding box

recall. The parameter ϵ of DBSCAN has been chosen such that the results of the precision and recall are optimised. Table 3.1 gives the obtained results.

ϵ	Precision	Recall
1	0.156	0.0475
2	0.158	0.0483
3	0.153	0.0473
4	0.153	0.0472
5	0.151	0.0463

Table 3.1: Precision and recall after the DBSCAN clustering

3.1.4 Results

The results obtained by this algorithm are quite good when two specific conditions are satisfied. The first one, is that the vehicle is in motion. When a vehicle is not moving, no vectors are generated making it disappear in the background. The second condition is that two vehicles should not be too close to each other. If they are not isolated, they will fuse into one detected vehicle as shown in Figure 3.5. This is unfortunately its biggest drawback which will result in a poor precision and recall. Do keep in mind that the precision and recall already have low values due to the nature of the used data. When a car is not moving, it cannot be detected at all, which lowers its precision and recall. The

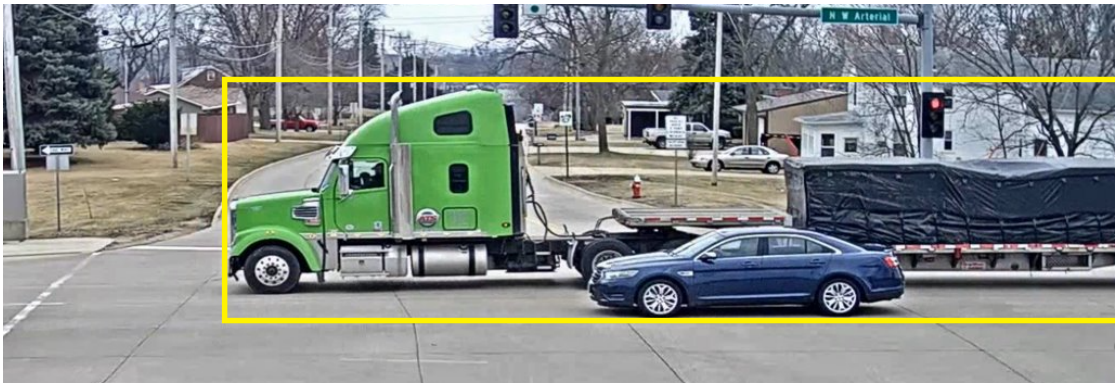


Figure 3.5: Car not being detected with semi passing by [6]

precision and recall are computed during the execution of the motion vector detection algorithm with the test sequence and they are:

$$\text{precision} = 0.1583 \tag{3.1}$$

$$\text{recall} = 0.0483 \tag{3.2}$$

The precision is not a great score and the recall is a very poor result. This is still true even when taking into account the impossible detections due to the lack of movement of a vehicle.

3.2 Detection using the frame's residuals

The frame's residuals contains the error occurred when predicting a frame, it is the second data type recovered from encoded videos. Therefore, it contains valuable information which could be used to detect vehicles. This data type closely matches the data type of the original frames, which is the usage of the RGB or YUV colour encoding. The usage of already existing algorithms on images is therefore possible.

3.2.1 Algorithm principle

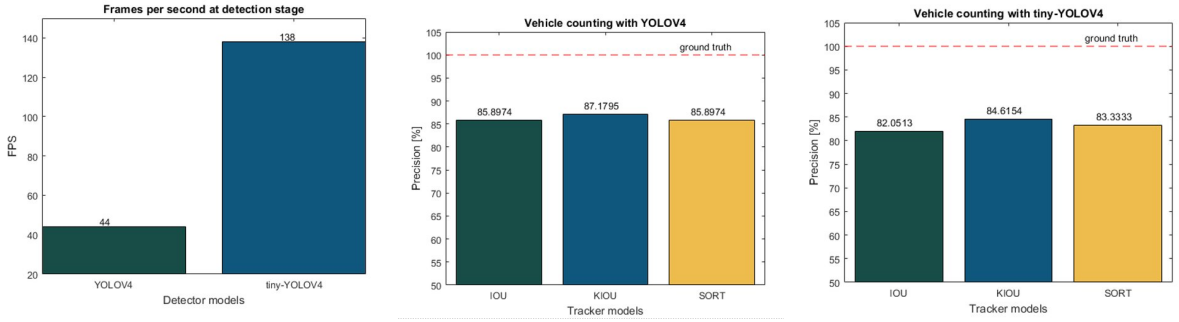
The principle of this algorithm is to create a model using a training set. Once this training is done, the model will be used to detect new vehicles. The training of the model was done outside of the thesis scope and is therefore not discussed in detail. The detail of this work can be found in the TRAIL [5] which took place this summer.

3.2.2 YOLO: Real-Time Object Detection

YOLO [16] [17] is a well known object detection algorithm. The one used in this thesis is the fourth version. The difference between YOLO and other classifiers or localisers, is that YOLO applies a single neural network on the full frame to detect. This network will divide the frame in regions to predict the bounding boxes and their respective probabilities. This approach is faster than R-CNN, because R-CNN requires thousands of neural networks for a single frame. This speed difference is also the main reason to choose YOLO over other object detection models (see Appendix C for speed results), YOLO can be run in real-time.

Why use TinyYOLO over YOLO ?

During the TRAIL [5], work has been done on the detection of the vehicles using YOLOv4 and TinyYOLOv4, a smaller version of YOLOv4. On the performance, TinyYOLOv4 was four times faster than YOLOv4 (see Figure 3.6a), but has a drawback that it has less precision on counting vehicles (Figure 3.6b & 3.6c). This difference of count is negligible, since it is maximum 5%. The choice has thus been made to use TinyYOLOv4. Figure 3.6 are results obtained from TRAIL.



(a) Speed of YOLOv4 and TinyYOLOv4 expressed in fps

(b) Precision of YOLOv4 with different trackers

(c) Precision of TinyYOLOv4 with different trackers

Figure 3.6: Comparison between YOLOv4 and TinyYOLOv4, results obtained using NVIDIA GTX 1080Ti GPU [5]

3.2.3 Filtering of the bounding boxes

Once the different bounding boxes are recovered from the model, it is necessary to filter overlapping bounding boxes. Filtering is done using the non-max suppression algorithm [18] [19]. This technique uses intersection-over-union score (IOU) (Appendix D) to remove bounding boxes that do detect the same object. An example is shown in Figure 3.7 . Using this algorithms removes most of the redundant bounding boxes in the frames and

Figure 3.7: Non-max suppression [20]

therefore makes the detection more precise.

3.2.4 Results

The results obtained from this algorithm depends primarily on the training of the TinyYOLO model. If it is poorly trained, it will give poor results compared to a more complete training sequence. As a reminder, when a vehicle is not moving, it becomes invisible and thus, most of the time, undetectable. This also lowers the precision and recall.
The precision and recall of this algorithm with the test sequence are:

$$\text{precision} = 0.3529 \tag{3.3}$$

$$\text{recall} = 0.4395 \tag{3.4}$$

Taking into account the impossible detection, the results obtained with the TinyYOLO algorithm are satisfying. They are not perfect, but with more training, it should reach higher precision and recall.

3.3 Comparing the two methodologies

Comparing both methodologies, it is clear that the TinyYOLO algorithms is superior over the vector detection algorithm proposed in this thesis. It has a better precision, and a non-negligible recall difference.
For the next part, the tracking of the vehicles, the bounding boxes of the TinyYOLO algorithm will be used as basis.

Chapter 4

Tracking algorithms

Be able to detect vehicles correctly is important, but the goal of the thesis is also to be able to track them in the whole test sequence. In this chapter, a comparison will be made between two existing algorithms and a new one that will use the motion vectors. The two existing algorithms are used as a basis of comparison; to see if the motion vectors add something which makes the tracking better, or worse.

4.1 Intersection-over-union (IOU) tracking algorithm

IOU [2] [3] is a tracking algorithm which uses only bounding boxes as input and outputs tracks it has found. It is a tracker in its most simple form and has the only advantage of being very fast at executing its task.

4.1.1 Algorithm working principle

For every bounding box in the frame i, the IOU tracking algorithm will compute the IOU-score (Appendix D) with all existing active tracks of the time $i - 1$. It selects the bounding box which has the highest score and if this score is higher than a certain threshold (σ_{IOU}), it will add the bounding box to the active tracks' list. If an active track does not find a matching bounding box, the track is removed from the active tracks. This track will afterwards be put in the found tracks' list only if its length is longer than a certain threshold (t_{min}). The short tracks are removed because they are mostly due to false positives.

A pseudo-code of this algorithm is written below (Algorithm 2). D_i stands for all the detections/bounding boxes in frame i, d_j stands for the j-th detection in a specific frame, T_a the active tracks, T_f the final/found tracks and F the number of frames. σ_l and σ_h are two threshold standing for the lower bound of confidence level to which a bounding box should be considered one and another lower bound of a level of confidence for a track to be considered one respectively.

Algorithm 2: IOU tracking algorithm [2] [3]

Inputs:
$D = \{D_0, D_1, ..., D_{F-1}\} = \{\{d_0, d_1, ..., d_{N-1}\}, \{d_0, d_1, ..., d_{N-1}\}, ...\}$
Initialize:
$T_a = \emptyset$,
$T_f = \emptyset$,
$D = \{\{d_i | d_i \in D_j, d_i \geq \sigma_l\} | D_j \in D\}$
for $f = 0$ *to* F **do**
  for $t_i \in T_a$ **do**
    $d_{best} = d_j$ where $\max(\text{IOU}(d_j, t_i))$, $d_j \in D_f$
    if $IOU(d_{best}, t_i) \geq \sigma_{IOU}$ **then**
      add d_{best} to t_i
      remove d_{best} from D_f
    else
      if $highest\_score(t_i) \geq \sigma_h$ **then**
        add t_i to T_f
      end
      remove t_i from T_a
    end
  end
  for $d_j \in D_t$ **do**
    start new track t with d_j and insert into T_a
  end
end
for $t_j \in T_A$ **do**
  if $highest\_score(t_i) \geq \sigma_h$ *and* $len(t_i) \geq t_{min}$ **then**
    add t_i to T_f
  end
end
return T_f

The IOU tracking algorithm suffers from false negatives. When the algorithm encounters one during one frame, it will lose the track and recreate a new track with a new ID the next frame. This will increase the ID switches for the found tracks, something to avoid as much as possible.

4.1.2 Results

These are the results of the 300 frames test sequence:

- There are 151 ID Switches
- 589 false negatives
- 1037 false positives
- 1044 ground truth objects

This results in the following score:

$$\text{MOTA} = 1 - \frac{\sum_t(\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t} \tag{4.1}$$

$$= 1 - \frac{151 + 589 + 1037}{1044} \tag{4.2}$$

$$= -0.699 \tag{4.3}$$

4.2 Kalman IOU

4.2.1 Algorithm working principle

Kalman IOU [4] can behave in two different ways. When the parameter frame_skip is set to false, it will behave like the IOU tracking algorithm, thus not having any advantage over it.
But this tracker has to be used with a frame_skip set to true. When it is enabled, the algorithm will for example use only one frame out of two. This frame skip ratio can be changed to any value wanted. When the algorithm skips a frame, it does not know anything about the bounding boxes. It will therefore estimate these using a Kalman Filter (Appendix E).
The algorithm can be optimised with the following parameters:

- σ_l: Same as in the IOU tracking algorithm, low detection threshold
- σ_{IOU}: Same as in the IOU tracking algorithm, it is the IOU threshold.
- σ_h: Same as in the IOU tracking algorithm, it is the high detection threshold.
- σ_{len}: Minimum track length to be considered a valid track
- n_{skip}: The program will use one out of n_{skip} frames for its tracking. Therefore discarding $n_{skip} - 1$ frames.

4.2.2 Used parameters

To obtain the results obtained in Subsection 4.2.3, the parameters in Table 4.1 were used to optimise the results:

σ_l	σ_{IOU}	σ_h	σ_{len}	n_{skip}
0.3	0.4	0.5	4	3

Table 4.1: Used parameters for Kalman IOU tracking

4.2.3 Results

These are the results of the 300 frames test sequence:

- There are 13 ID Switches
- 191 false negatives
- 811 false positives
- 1044 ground truth objects

This results in the following MOTA score:

$$\text{MOTA} = 1 - \frac{\sum_t(\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t} \tag{4.4}$$

$$= 1 - \frac{13 + 191 + 811}{1044} \tag{4.5}$$

$$= 0.0278 \tag{4.6}$$

The MOTA score obtained with Kalman IOU is better than the one obtained using IOU. This is to be expected since this algorithm uses a Kalman Filter to predict frames, therefore ignoring if a frame's detection had a false negative during a skipped frame.

4.3 IOU using vectors

The IOU tracking algorithm has a big drawback when faced with false negatives. When a false negative appears, the algorithm removes the track from the active tracks and starts a new track when the object has been detected again, but giving it another ID. It does not link both tracks together. To counter this, the usage of the vectors when a false negative appears could help linking the tracks.

4.3.1 Algorithm working principle

The working principle of this algorithm is as follows. The tracking works based on IOU. As long as the tracking of an object has not been lost, motion vectors will not be used. If it is lost, the algorithm creates a new bounding box following the motion vectors available in the previous bounding box itself. It computes the mean and moves the bounding box accordingly to that direction. To remove those bounding boxes such as it does not stay indefinitely, a new variable is introduced. This new variable $life$, is the amount of consecutive usage of motion vectors for each active track. If this amount becomes higher than a threshold(σ_{life}), the track itself is removed from the active tracks and stored in the found tracks following the same conditions as for the IOU tracking algorithm.

The pseudo-code of this algorithm is written below (Algorithm 3).

In this algorithm, the following variables were used:

- D_i are all the detections of frame i.
- d_j is the j-th detection of a frame.
- F is the total amount of frames.
- T_a are the active tracks.
- T_f are the found/final tracks.
- σ_l is the lower confidence bound for detections.
- σ_{IOU} is the lower bound IOU-score for a detection to be considered a match.
- life is the consecutive amount of times the vectors were used for a track.
- σ_{life} is the maximum amount of consecutive usages of the motion vectors.
- $IOU(x, y)$ is a function which returns the IOU-score between detection x and y (App. D).

Algorithm 3: Vector IOU tracking algorithm, based on IOU [2] [3]

Inputs: $D = \{D_0, D_1, ..., D_{F-1}\} = \{\{d_0, d_1, ..., d_{N-1}\}, \{d_0, d_1, ..., d_{N-1}\}, ...\}$
Initialize: $T_a = \emptyset$, $T_f = \emptyset$, $D = \{\{d_i | d_i \in D_j, d_i \geq \sigma_l\} | D_j \in D\}$
for $f = 0$ *to* F **do**
  for $t_i \in T_a$ **do**
    $d_{best} = d_j$ where max(IOU(d_j, t_i)), $d_j \in D_f$
    if $IOU(d_{best}, t_i) \geq \sigma_{IOU}$ **then**
      add d_{best} to t_i
      set $life$ of t_i to 0
      remove d_{best} from D_f
    else
      if $life(t_i) < \sigma_{life}$ **then**
        Compute the mean (μ_{t_i}) of motion vectors in t_i's last bounding box
        Create d_{new} = translation(last_bounding_box$(t_i), \mu_{t_i}$)
        add d_{new} to t_i
        increment $life$ of t_i
      else
        if $len(t_i) \geq t_{min}$ **then**
          add t_i to T_f
        end
        remove t_i from T_a
      end
    end
  end
  for $d_j \in D_t$ **do**
    start new track t with d_j and insert into T_a
  end
end
return T_f

4.3.2 Results

These are the results of the 300 frames test sequence:

- There are 106 ID Switches
- 535 false negatives
- 1182 false positives
- 1044 ground truth objects

This results in the following score for MOTA:

$$\text{MOTA} = 1 - \frac{\sum_t(\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t} \tag{4.7}$$

$$= 1 - \frac{106 + 535 + 1182}{1044} \tag{4.8}$$

$$= -0.746 \tag{4.9}$$

One big advantage of using the vectors is the ability of having less ID switches compared to the IOU tracking algorithm. This happens when a moving vehicle comes to a stop. The motion vectors are static at that exact position, giving the track more time to re-detect invisible vehicles. This advantage is useful when σ_{life} is high, but a higher σ_{life} will add more false positives, since it removes the mechanism in IOU to remove too small active tracks.

4.4 Comparing those algorithms

As expected, the Kalman IOU does a better job than the IOU tracking algorithm. This can be explained by the fact it uses a Kalman filter to create missing boxes when false negatives occur, lowering the ID switches and the false negatives. Since the vector IOU tracker algorithm created in this thesis is based on the IOU algorithm but uses the additional vector motion's information when IOU fails, it should behave quite like IOU. The results obtained by taking the mean value of the motion vectors to adapt the bounding box when IOU fails, does not improve the tracking at all. This is due to the motion vectors random behaviour. The only advantage of this usage is when vehicles are stationary, it can help them hold their position without changing the ID of the vehicle for a longer time. But this advantage alone is not convincing enough to be a good alternative to IOU.

.	IOU	Kalman IOU	Vector IOU
MOTA-score	-0.699	0.0278	-0.746

Table 4.2: Obtained MOTA score for each algorithm

Conclusion

What to learn of the conclusions of Chapter 3 and Chapter 4 ?

The results obtained from using encoded video flux are still results of a new concept which needs further development. It is very clear that the residuals play a role in the detection of the vehicles. When a machine learning model is sufficiently trained, it can reach a usable precision and recall. This means that it is possible to use tracking algorithms such as SORT [21] or Kalman IOU [4] afterwards.
Unfortunately, the same cannot be said about the motion vectors. The randomness complementing the right motion clearly moves the mean direction of a group to a random direction. This does not mean that the vectors cannot be used, further development could be done to make the best usage out of them.

Ways of improving the tracking algorithm with motion vectors

There are multiple ways the algorithm could be improved.
A first improvement could be the use of the HOTA metric [22]. This metric' paper tells that MOTA is biased to the detection, and therefore does not give the best evaluation possible for the tracking. Using such a metric could help all trackers to optimise their ability to track.

Improvements to the Vector IOU algorithm could also be done. Suggestions such as time filtering the vectors, as has been done in the Vector Detection algorithm, could be a solution. When computing the mean of a bounding box, the usage of a weighted mean, where the centre of the bounding box has more weight than its extremities. This could avoid the fact other vector groups appearing at the extremities lead to a false movement of the bounding boxes.

Development could also be done at the encoding procedure. Right now, the goal of a encoder is to minimise the final size of the video file. What if the goal is changed to optimise this file size and tries to create more homogenous motion vectors ? This will of course be a trade-off between the file size and the homogeneity of the motion vectors.

The results obtained during this study are not very promising for the usage of the encoded video format. But video encoders are still improved today, and therefore further research should be done on how to use encoded videos to improve detection and tracking. The fact that encoded videos are harder to decode when incomplete, gives them a great advantage over privacy that decoded videos will never reach.

Appendix A

Adding noise in images: effect on block sizes

Let p be the pixel value, i the exact pixel value in the image and S_j the j-th subblock of the studied block of size N^2. x is additive noise with $X \sim \mathcal{N}(0, \sigma^2)$[1].

$$p(x, y, t) = i(x, y, t) + x(t) \tag{A.1}$$

The absolute difference between each pixel and its previous frame one is:

$$|\Delta p(x, y, t)| = |p(x, y, t) - p(x, y, t-1)| \tag{A.2}$$

When summing up these differences for a whole subblock, the error ϵ_j is obtained.

$$\epsilon_j = \sum_{\forall x,y \in S_j} |\Delta p(x, y, t)| = \sum_{\forall x,y \in S_j} |p(x, y, t) - p(x, y, t-1)| \tag{A.3}$$

$$= \sum_{\forall x,y \in S_j} |i(x, y, t) - i(x, y, t-1) + x(t) - x(t-1)| \tag{A.4}$$

$$\leq \sum_{\forall x,y \in S_j} (|i(x, y, t) - i(x, y, t-1)| + |x(t) - x(t-1)|) \tag{A.5}$$

When the block size S_j is big and of size N^2, the following approximation[2] can be done.

$$\epsilon_j \approx \underbrace{\sum_{\forall x,y \in S_j} |i(x, y, t) - i(x, y, t-1)|}_{=\gamma} + \sqrt{2} N^2 \sigma \sqrt{\frac{2}{\pi}} \tag{A.6}$$

When the subblock size is big, and the noise variance is high, the image information becomes negligible compared to the obtained mean due to the added noise ($2N^2\sigma\sqrt{\frac{1}{\pi}} >> \gamma$) and the following will apply:

$$\epsilon_j \approx \sum_{\forall x,y \in S_j} |\Delta p(x, y, t)| = 2N^2 \sigma \sqrt{\frac{1}{\pi}} = Z_j \tag{A.7}$$

[1] In the code, it is a sum of uniform distributions that are used. The central limit theorem establishes that a sum of independent random variables tends to a normal distribution.

[2] Using a normal distribution $\mathcal{N}(\mu, \sigma^2)$, then $E[|X - \mu|] = \sigma\sqrt{\frac{2}{\pi}}$ and $\text{Var}[|X - \mu|] = \sigma^2 - E[|X - \mu|]^2$ [23]

Where the mean and variance of the distribution Z_j is:

$$E[Z_j] = 2N^2\sigma\sqrt{\frac{1}{\pi}} \tag{A.8}$$

$$Var[Z_j] = 4N^4\sigma^2 - E[Z_j]^2 = 4N^4\sigma^2 - 4N^4\sigma^2\frac{1}{\pi} \tag{A.9}$$

$$\approx \frac{8}{3}N^4\sigma^2 \tag{A.10}$$

The mean and the variance between the four subblocks will thus be $Z = \frac{1}{4}\sum_{j=0}^{3} Z_j$, where $\forall i \neq j, Cov(Z_i, Z_j) = 0$:

$$E[Z] = E\left[\frac{1}{4}\sum_{j=0}^{3} Z_j\right] = 2N^2\sigma\sqrt{\frac{1}{\pi}} \tag{A.11}$$

$$Var[Z] = Var\left[\frac{1}{4}\sum_{j=0}^{3} Z_j\right] \tag{A.12}$$

$$= \frac{1}{16}\sum_{j=0}^{3} Var\left[Z_j\right] \tag{A.13}$$

$$= \frac{2}{3}N^4\sigma^2 \tag{A.14}$$

With the different hypothesises taken into account, this shows that big blocks can still be split randomly in smaller blocks. This is because blocks are split depending on the absolute error variance they have between each other. Therefore the blocks still split in a random way.
When the block sizes become small, some approximations that were used become untrue. The mean value obtained with the folded absolute normal distribution shall have even more variations. And more variation means more splitting of the boxes.
This phenomenon can be seen in Figure A.1a, A.1b and A.1c. The more the noise is added, the more randomly the smaller boxes will be split, making intermediate block size rare.

(a) No noise added to the image (b) Small amount of noise added to the image (c) Big amount of noise added to the image

Figure A.1: Block sizes, darker means smaller block

Appendix B

DBSCAN

The DBSCAN [24] clustering algorithm is a Density-Based Spatial Clustering of Applications with Noise. It uses density to estimate the different clusters.
The algorithm could be compared to a propagation algorithm. It uses a radius ϵ and a minimum population threshold min_samples as input and starts by taking a point randomly in the given set. Every point which lays in the radius ϵ of this starting point, will be added to the cluster. Afterwards, it uses all the points in this cluster to look for new points in their radius. It continues to propagate until no more points are added to the cluster. (Figure B.1)
Once a cluster has been finished, it will take a new random point not assigned to any cluster, and start this procedure again until all points are part of a cluster.
Once all points are clustered, DBSCAN will look at the size and confirm a cluster if its size is bigger than the min_samples threshold.

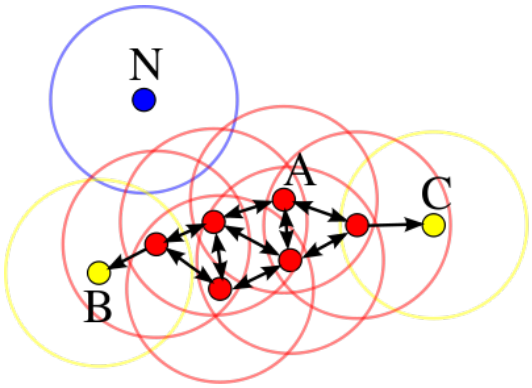


Figure B.1: DBSCAN clustering principle [25]

Appendix C

Performance of different models

Performance of different models can be found in Table C.1. All values are based on the COCO Dataset [26] [17].

Model	mAP	FPS
SSD300	41.2	46
SSD500	46.5	19
YOLOv2 608x608	48.1	40
Tiny YOLO	23.7	244
SSD321	45.4	16
DSSD321	46.1	12
R-FCN	51.9	12
SSD513	50.4	8
DSSD513	53.3	6
FPN FRCN	59.1	6
Retinanet-50-500	50.9	14
Retinanet-101-500	53.1	11
Retinanet-101-800	57.5	5
YOLOv3-320	51.5	45
YOLOv3-416	55.3	35
YOLOv3-608	57.9	20
YOLOv3-tiny	33.1	220
YOLOv3-spp	60.6	20

Table C.1: FPS and mAP obtained with different models on the COCO dataset. The higher the FPS, the faster the model, the higher the mean Average Precision (mAP), the better the results. [26]

Appendix D

Intersection over union (IOU)

When using bounding boxes, it is quite common to evaluate the detections with the ground truth. An evaluation method is the Intersection-Over-Union (IOU). The way it is computed is the following:

$$\text{IOU}(A, B) = \frac{\text{intersection of A \& B}}{\text{union of A \& B}} \tag{D.1}$$

$$= \frac{A \cap B}{A \cup B} \tag{D.2}$$

The obtained score is between 0 and 1, 1 being a perfect match, zero being no match at all between two bounding boxes. Some examples of the results are shown in Figure D.1.

Figure D.1: IOU score examples [27]

Appendix E

Kalman Filter

The Kalman filter[1] is a filter used to estimate states of discrete linear dynamical time-invariant systems. This is necessary since in all systems, noise is unavoidable. The system in Equation E.1 refers to the state x at time k of a system, where u_k is the input, $w_k \sim \mathcal{N}(0, Q)$ a zero-mean additive Gaussian noise of variance Q. A and B are the transition and the control input matrix respectively.

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{E.1}$$

Measurements y_k are recovered from the state as shown in Equation E.2, where C is the observation matrix and $v_k \sim \mathcal{N}(0, R)$ is a zero-mean additive Gaussian noise of variance R.

$$y_k = Cx_{k-1} + v_{k-1} \tag{E.2}$$

All values of the system, A, B, C, Q and R, are known inputs, as well as u_k. The main role of the Kalman filter is to provide an estimation of the state x at the time k. This estimation is based on the initial state x_0 and the different measurement y_k for $k = 0$ to k-th observation.
The Kalman filter algorithm works in two different steps, the first one being the prediction step, followed by the update step.
In the first step, the algorithm will compute a predicted state estimate (Eq. E.3):

$$\hat{x}_k^- = A\hat{x}_{k-1}^+ + Bu_{k-1} \tag{E.3}$$

And a predicted error covariance P_k (Eq. E.4):

$$P_k^- = AP_{k-1}^+ A^T + Q \tag{E.4}$$

With these two predictions, it enters the second step by updating the measurement residual (Eq. E.5), the Kalman gain (Eq. E.6), the state estimate (Eq. E.7) and the covariance error (Eq. E.8) one after another.

$$\widetilde{y}_k = y_k - C\hat{x}_k^- \tag{E.5}$$

[1]This appendix is based of the course *LINMA1731: Stochastic processes: Estimation and prediction* and from the following reference [28]

$$K_k = P_k^- C^T \left(R + C P_k^- C^T \right)^{-1} \tag{E.6}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \widetilde{y}_k \tag{E.7}$$

$$P_k^+ = \left(I - K_k C \right) P_k^- \tag{E.8}$$

The - or + sign are written to separate the state or covariance error during the prediction step and the update step respectively.
The Kalman filter can be extended to non-linear systems (Eq. E.9) where it will approximate the non-linear system by a linear one (Eq. E.10).

$$\begin{cases} x_k &= f\left(x_{k-1}, u_{k-1}\right) + w_{k-1} \\ y_k &= h(x_k) + v_k \end{cases} \tag{E.9}$$

$$\begin{cases} A_{k-1} &= \frac{\delta f}{\delta x}\Big|_{\hat{x}_{k-1}^+, u_{k-1}} \\ B_{k-1} &= \frac{\delta f}{\delta u}\Big|_{\hat{x}_{k-1}^+, u_{k-1}} \\ C_k &= \frac{\delta h}{\delta x}\Big|_{\hat{x}_k^-} \end{cases} \tag{E.10}$$

The procedure is exactly the same as for a linear system, first a prediction step, followed by an update step. For the prediction, a state estimate (Eq. E.11) and a error covariance (Eq. E.12) will be computed:

$$\hat{x}_k^- = f\left(\hat{x}_{k-1}^+, u_{k-1}\right) \tag{E.11}$$

$$P_k^- = A P_{k-1}^+ A^T + Q \tag{E.12}$$

For the update step, the measurement residual (Eq. E.13), the Kalman gain (Eq. E.14), the state estimate (Eq. E.15) and the covariance error (Eq. E.16) are computed:

$$\widetilde{y}_k = y_k - h\left(\hat{x}_k^-\right) \tag{E.13}$$

$$K_k = P_k^- C^T \left(R + C P_k^- C^T \right)^{-1} \tag{E.14}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \widetilde{y}_k \tag{E.15}$$

$$P_k^+ = \left(I - K_k C \right) P_k^- \tag{E.16}$$

Prior knowledge about the dynamical system is therefore crucial to use the Kalman filter.

Bibliography

[1] European Union Law. General data protection regulation (gdpr). https://www.belgium.be/fr/justice/respect_de_la_vie_privee/protection_des_donnees_personnelles.

[2] Erik Bochinski, Tobias Senst, and Thomas Sikora. Extending iou based multi-object tracking by visual information. In *IEEE International Conference on Advanced Video and Signals-based Surveillance*, pages 441–446, Auckland, New Zealand, November 2018. URL http://elvera.nue.tu-berlin.de/files/1547Bochinski2018.pdf.

[3] Erik Bochinski, Volker Eiselein, and Thomas Sikora. High-speed tracking-by-detection without using image information. In *International Workshop on Traffic and Street Surveillance for Safety and Security at IEEE AVSS 2017*, Lecce, Italy, August 2017. URL http://elvera.nue.tu-berlin.de/files/1517Bochinski2017.pdf.

[4] siyuanc2. Kalman-iou tracker. https://github.com/siyuanc2/kiout.

[5] Karim El Khoury and Jonathan Samelson. Privacy-preserving object detection in the compressed domain.

[6] Milind Naphade, Rama Chellappa, David Anastasiu, Anuj Sharma, Ming-Ching Chang, Xiaodong Yang, Shuo Wang, Zheng Tang, Liang Zheng, Pranamesh Chakraborty, and Stan Sclaroff. Ai city challenge. https://www.aicitychallenge.org/.

[7] Scikit learn. Precision-recall. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

[8] Anton Milan, Laura Leal-Taixé, Ian Reid, and Konrad Schlinder. Mot16: A benchmark for multi-object tracking. https://arxiv.org/pdf/1603.00831.pdf, 2016.

[9] MulticoreWare. High efficiency video encoding (hevc). https://www.x265.org/.

[10] Prof.S. Sengupta. Video coding : Basic building blocks. https://www.youtube.com/watch?v=OdYlNAFYq44, 2008.

[11] Aroh Barjatya. Block matching algorithms for motion estimation, 2004.

[12] "F. Vermaut, Y. Deville, X. Marichal, and B. Macq. A distributed adaptive block matching algorithm: Dis-abma. https://dial.uclouvain.be/pr/boreal/fr/object/boreal%3A42972, 2001.

[13] Lai-Man Po and Wing-Chung Ma. A novel four-step search algorithm for fast block motion estimation, 1996.

[14] Shan Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block-matching motion estimation, 2000.

[15] Wikipedia. Colorwheel image. https://fr.m.wikipedia.org/wiki/Fichier:Colorwheel.svg, .

[16] Joseph Redmon and Ali Farhadi. Yolo: Real-time object detection. https://pjreddie.com/darknet/yolo/, .

[17] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[18] Sambasivarao. K. Non-maximum suppression. https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c, .

[19] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Improving object detection with one line of code. *arXiv*, 2017.

[20] Sambasivarao. K. Non-maximum suppression. https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c, .

[21] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016. doi: 10.1109/ICIP.2016.7533003.

[22] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. https://arxiv.org/pdf/2009.07736.pdf.

[23] Wikipedia. Folded normal distribution. https://en.wikipedia.org/wiki/Folded_normal_distribution, .

[24] Scikit-learn. Dbscan. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

[25] Wikipedia. Dbscan. https://fr.wikipedia.org/wiki/DBSCAN, .

[26] Joseph Redmon and Ali Farhadi. Yolo: Real-time object detection. https://pjreddie.com/darknet/yolo/, .

[27] Adrian Rosebrock. Intersection over union (iou) for object detection. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[28] Youngjoo Kim and Hyochoong Bang. Introduction to kalman filter and its applications. https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**