

Compressed Domain Moving Object Detection based on H.264/AVC Macroblock Types

Marcus Laumer^{1,2}, Peter Amon², Andreas Hutter² and André Kaup¹

¹*Multimedia Communications and Signal Processing, University of Erlangen-Nuremberg, Erlangen, Germany*

²*Imaging and Computer Vision, Siemens Corporate Technology, Munich, Germany*

Keywords: H.264/AVC, Compressed Domain, Object Detection, Macroblock Type.

Abstract: This paper introduces a low complexity frame-based object detection algorithm for H.264/AVC video streams. The method solely parses and evaluates H.264/AVC macroblock types extracted from the video stream, which requires only partial decoding. Different macroblock types indicate different properties of the video content. This fact is used to segment a scene in fore- and background or, more precisely, to detect moving objects within the scene. The main advantage of this algorithm is that it is most suitable for massively parallel processing, because it is very fast and combinable with several other pre- and post-processing algorithms, without decreasing their performance. The actual algorithm is able to process about 3600 frames per second of video streams in CIF resolution, measured on an Intel[®] Core[™] i5-2520M CPU @ 2.5 GHz with 4 GB RAM.

1 INTRODUCTION

Moving object detection is probably one of the most widely used video analysis procedures in many different applications. Video surveillance systems need to detect moving persons or vehicles, trackers have to be initialized with the objects they should track, and recognition algorithms require the regions within the scene where they should identify objects. For this reason, several proposals for efficient object detection have been published. Most of them operate in the pixel domain, i.e., on the actual pixel data of each frame. This usually leads to a very high accuracy, but at the expense of computational complexity.

As most video data is stored or transferred in compressed representation, the bit stream has to be completely decoded beforehand in such scenarios. Therefore, attempts have been made to eliminate the costly step of decoding and to perform the analysis directly in the compressed domain.

Detection algorithms can therefore be divided into two categories: pixel domain detection and compressed domain detection. Thereby, pixel domain is well-defined as the entire video content is decoded and all video frames are available in pixel representation. Compressed domain on the other hand does not clearly express which part of the video content has to be decoded and which part may remain compressed. Several compressed domain detection methods that

achieve good results by analyzing different entropy decoded syntax elements have been presented.

The remainder of this paper is organized as follows. Section 2 introduces some state-of-the-art compressed domain detection algorithms. Section 3 provides a brief overview of the H.264/AVC syntax elements that are relevant for our algorithm and their extraction. It also describes how macroblock types are grouped to categories and defines which categories indicate moving regions. Section 4 describes the actual algorithm and the segmentation process in detail. After that, some experimental results are given in Section 5. Section 6 concludes this paper with a summary and an outlook.

2 RELATED WORK

Established moving object detection methods in hybrid video codecs are based on solely extracting and analyzing motion vectors. For instance, (Szczerba et al., 2009) showed an algorithm to detect objects in video surveillance applications using H.264/AVC video streams. Their algorithm assigns a motion vector to each 4x4 pixel block of the examined frame. Thereby, macroblocks with larger partitions than 4x4 are divided and their motion vector is assigned to the smaller blocks. Since intra-coded macroblocks

have no corresponding motion vector, the algorithm interpolates a vector from previous and consecutive frames. This results in a dense motion vector field. This dense motion vector field is further analyzed to estimate vectors that represent real motion by calculating spatial and temporal confidences as introduced by (Wang et al., 2000).

Other object detection methods do not solely analyze motion vectors but also exploit additional compressed information, like macroblock partition modes, e.g., (Fei and Zhu, 2010) and (Qiya and Zhicheng, 2009) or transform coefficients, e.g., (Mak and Cham, 2009) and (Porikli et al., 2010).

(Fei and Zhu, 2010), for instance, presented a study on mean shift clustering based moving object segmentation for H.264/AVC video streams. In a first step, their method refines the extracted raw motion vector field by normalization, median filtering, and global motion compensation, whereby already at this stage the algorithm uses macroblock partition modes to enhance the filtering process. The resulting dense motion vector field and the macroblock modes then serve as input for a mean shift clustering based object segmentation process, adopted from pixel domain approaches, e.g., introduced by (Comaniciu and Meer, 2002).

(Mak and Cham, 2009) on the other hand analyze motion vectors in combination with transform coefficients to segment H.264/AVC video streams to foreground and background. Quite similar to the techniques described before, their algorithm initially extracts and refines the motion vector field by normalization, filtering, and background motion estimation. After that, the foreground field is modeled as a Markov random field. Thereby, the transform coefficients are used as an indicator for the texture of the video content. The resulting field indicates fore- and background regions, which are further refined by assigning labels for distinguished objects.

(Poppe et al., 2009) introduced an algorithm for moving object detection in the H.264/AVC compressed domain that evaluates the size of macroblocks (in bits) within video streams. Thereby, the size of a macroblock includes all corresponding syntax elements and the encoded transform coefficients. The first step of their algorithm is to find the maximum size of background macroblocks, which is performed in an initial training phase. During the subsequent analysis, each macroblock that exceeds this size is regarded as foreground, as an intermediate step. Macroblocks with less size are divided to macroblocks in Skip mode and others. Labeling of macroblocks in Skip mode depends on the labels of their direct neighbors, while all other macroblocks are directly

labeled as background. Subsequent steps are spatial and temporal filtering. These two steps are performed to refine the segmentation. During spatial filtering background macroblocks will be changed to foreground, if most of their neighbors are foreground as well. Foreground macroblocks will be changed to background during temporal filtering, if they are neither foreground in the previous frame nor in the next frame. The last refinement step is to evaluate boundary macroblocks on a sub-macroblock level of size 4 by 4 pixels.

Extracting motion vectors and transform coefficients from a compressed video stream requires more decoding steps than just extracting information about macroblock types and partitions. Hence, attempts have been made to directly analyze these syntax elements.

(Verstockt et al., 2009) proposed an algorithm for detecting moving objects by just extracting macroblock partition information from H.264/AVC video streams. First, they perform a foreground segmentation by assigning macroblocks to foreground and background, which results in a binary mask for the examined frame. Thereby, macroblocks in 16x16 partition mode (i.e., no sub-partitioning of the macroblock, including the skip mode) are regarded as background and all other macroblocks are labeled foreground. To further enhance the generated mask, their algorithm then performs temporal differencing of several masks and median filtering of the results. In a final step, objects are extracted by blob merging and convex hull fitting techniques. (Verstockt et al., 2009) designed their algorithm for multi-view object localization. Hence, the extracted objects of a single view then serve as input for the multi-view object detection step.

A more basic detection method than moving object detection is to detect global content changes within scenes. (Laumer et al., 2011) designed a change detection algorithm for RTP streams that does not require video decoding at all. They presented the method as a preselection for further analysis modules, since change detection can be seen as a preliminary stage of, e.g., moving object detection. Each moving object causes a global change within the scene. Their algorithm evaluates RTP packet sizes and number of packets per frame. Since no decoding of video data is performed the method is codec-independent and very efficient.

The algorithm we present in this paper solely extracts and evaluates macroblock types to detect moving objects in H.264/AVC video streams. It can either be performed as stand-alone application or be based on the results of the change detection algorithm pre-

sented by (Laumer et al., 2011). Once a global change within the scene is detected, the object detection algorithm can be started to identify the cause of this change.

3 MACROBLOCK TYPE CATEGORIES AND SYNTAX ELEMENTS

3.1 Categories and Weights

The H.264/AVC video compression standard was jointly developed by the ITU-T VCEG (VCEG, 2011) and the ISO/IEC MPEG (MPEG, 2010). It belongs to the class of block-based hybrid video coders. In a first step each frame of a video sequence will be divided in several so-called slices and each slice will be further divided in so-called macroblocks, which have a size of 16 by 16 pixels. In a second step, the encoder decides, according to a rate-distortion-optimization (RDO), how each macroblock will be encoded. Thereby, several different macroblock types of three classes are available. The first class is used if the macroblock should be intra-frame predicted from its previously encoded neighbors. The second and third classes are used in an inter-frame prediction mode, which allows to exploit similarities between frames. It is defined that macroblocks of the second class are predicted by just using one predictor, whereas macroblocks of the third class are predicted by using two different predictors. They are called I, P, and B macroblocks, respectively.

The same classification is defined for slices. In the scope of this work, the H.264/AVC Baseline profile is assumed. Within this profile, only I and P slices are allowed. The 32 macroblock types available for these two slice classes are grouped to six self-defined macroblock type categories (MTC):

MB_I_4x4. Intra-frame predicted macroblocks that are further divided into smaller blocks of size 4 by 4 pixels.

MB_I_16x16. Intra-frame predicted macroblocks that are not further divided.

MB_P_8x8. Inter-frame predicted macroblocks that are further divided into smaller blocks of size 8 by 8 pixels.

MB_P_RECT. Inter-frame predicted macroblocks that are further divided into smaller blocks of rectangular (not square) shape (16x8 or 8x16).

MB_P_16x16. Inter-frame predicted macroblocks that are not further divided.

Table 1: Macroblock type weights (MTW) of macroblock type categories (MTC).

Slice Type	MTC	Assumption	MTW
I	MB_I_4x4, MB_I_16x16	n/a	n/a
P	MB_I_4x4	most likely motion	3
P	MB_I_16x16	most likely motion	3
P	MB_P_8x8	likely motion	2
P	MB_P_RECT	likely motion	2
P	MB_P_16x16	maybe motion	1
P	MB_P_SKIP	most likely no motion	0

MB_P_SKIP. No additional data is transmitted for these macroblocks. Instead, the motion vector predictor that points to the first reference frame is used directly.

The decision of the RDO which macroblock type will be used for encoding the block heavily depends on the actual pixel data of this block and its difference to previous frames. Therefore, evaluating macroblock types can give a good guess of the location of moving objects within the scene. In order to determine which macroblock types indicate moving objects, an initial macroblock type weight (MTW) has to be defined for each category MTC first, which are shown in Table 1.

In I slices, only intra-coded macroblocks are allowed. In this case, only two categories MTC are available and no information about moving objects can be derived. To solve this problem different solutions are imaginable. One approach is to inter- or extrapolate from neighboring slices if the current frame consists of several slices. If the encoder configuration just allows one slice per frame, the resulted fore- and background segmentation mask of the previous frame could be also used for the subsequent I frame. For further enhancing this result, the mask could be interpolated by also considering the mask of the subsequent P frame, if the system configuration admits.

Intra-coded macroblocks are also available in P slices. Within a P slice it is assumed that the two categories **MB_I_4x4** and **MB_I_16x16** indicate blocks with high motion, because usually the encoder decides to choose an I macroblock type if similar video content could not be found in previous frames. Therefore, it is most likely that an object has moved or just entered the scene within this region.

Macroblock types of the both categories **MB_P_8x8** and **MB_P_RECT** will usually be selected by the encoder if blocks that are smaller than 16 by 16 pixels can be encoded more efficiently than the entire macroblock. That usually means that these regions are very structured and/or have been slightly changed compared to previous frames. Hence, it is

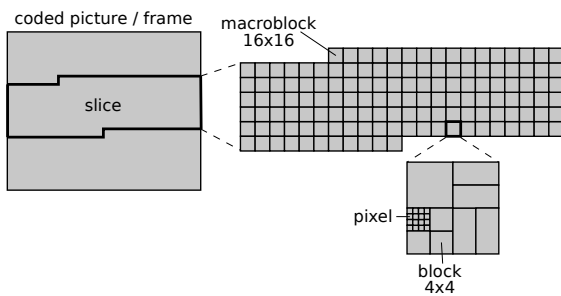


Figure 1: Sample hierarchical structure of block-based video coding.

assumed that likely a moving object is present here.

Macroblocks that are not further divided (i.e., of category MB_P_16x16) indicate high uncertainty concerning moving objects. On the one hand it is conceivable that slowly moving objects with constant directions are present in these regions, but on the other hand the corresponding motion vector could be quite short and this type has been selected because of a slightly noisy source. Therefore, the assumption here is that there is maybe motion.

The last category MB_P_SKIP is selected by the encoder if the predicted motion vector points to an area within the previous frame that is quite similar to the current macroblock. That means that it is most likely that there is no motion since there is nearly no difference between the current and the previous frame within this region.

Since objects usually extent over several macroblocks, the moving object certainty (MOC) of a macroblock highly depends on its neighboring macroblocks. This is further described in Section 4.

3.2 Syntax Extraction

To be able to assign macroblocks to the previously defined categories MTC, the macroblock types have to be extracted from the bit stream. As already mentioned, H.264/AVC is a block-based video compression standard and has a hierarchical structure consisting of five levels. Figure 1 illustrates this hierarchy.

The highest hierarchical level is a *coded picture*. Since the Baseline profile of H.264/AVC does not support interlaced coding, a coded picture within this profile is always an entire *frame*. On the next level a frame consists of at least one *slice*. If flexible macroblock ordering (FMO) is not used, which is assumed since FMO is rarely used in practice, a slice consists of several consecutive *macroblocks* on the third level. Each macroblock can be further divided in smaller *blocks*, at which the smallest available block has a size of 4 by 4 *pixels*.

H.264/AVC defines a huge number of syntax el-

ements. The most important for the presented algorithm will be discussed in the following. The `nal_unit_type` in the network abstraction layer (NAL) unit header indicates if the contained coded slice belongs to an instantaneous decoding refresh (IDR) or non-IDR frame. IDR frames can only consist of I slices while non-IDR frames are composed of slices of any type. The actual type of each slice is then encoded within its header by the syntax element `slice_type`. The beginning of the slices within the current frame is encoded by the element `first_mb_in_slice`, which can also be extracted from the slice headers. On macroblock level two elements are extracted. As already mentioned, no further information is transmitted if a macroblock is encoded with P_SKIP type. In this case, the bit stream contains an element called `mb_skip_run` that indicates the number of consecutive macroblocks in skip mode. For all macroblocks in non-skip mode the algorithm extracts the available syntax element `mb_type`.

As soon as all these syntax elements are extracted and parsed accordingly, the algorithm starts to evaluate them, as described in the following section.

4 MOVING OBJECT DETECTION ALGORITHM

4.1 Foreground/Background Segmentation

The presented object detection algorithm relies on the assumptions defined in Section 3. The H.264/AVC syntax elements are extracted from the bit stream and decoded, if required. The `nal_unit_type` is directly accessible without decoding. To access the other syntax elements the bit stream has to be parsed, i.e., entropy decoded. Already during the parsing process each macroblock is assigned to one of the six categories MTC and the corresponding weight MTW is set.

An example category MTC and weight MTW map is shown in Figure 2(b) and Figure 2(c), respectively. Thereby, the colors within the category MTC map are defined as follows.

- MB_I_4x4: light red ■
- MB_I_16x16: red ■
- MB_P_8x8: light blue ■
- MB_P_RECT: blue ■
- MB_P_16x16: dark blue ■
- MB_P_SKIP: black ■

The weight MTW map (and also the certainty MOC map in Figure 2(d)) is illustrated by a gray

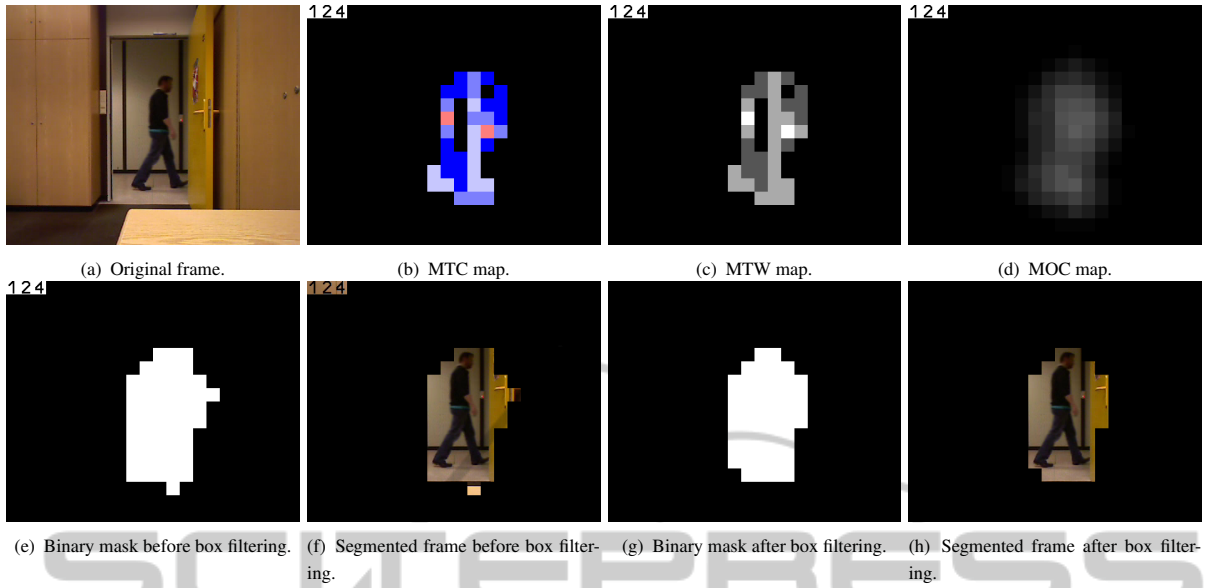


Figure 2: Sample maps and masks created by the algorithm (sequence: *door*).

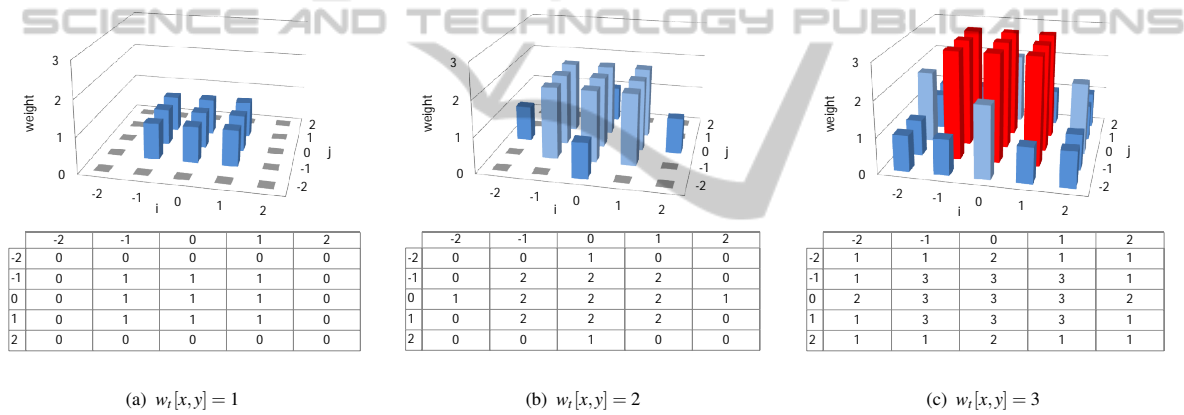


Figure 3: 3-dimensional illustration of discrete kernels for different MTWs.

scale picture, at which brighter gray levels denote a higher weight (or certainty in case of the certainty MOC map).

The main challenge of the algorithm is to create a robust map that indicates where within the scene moving objects are located, or in other words to transform category MTC/weight MTW maps to certainty MOC maps. These maps significantly differ from each other, since weight MTW maps do not take dependencies between neighboring macroblocks into account while certainty MOC maps do. Macroblocks have a size of 16 by 16 pixels. The assumption that actual moving objects usually span over several macroblocks requires to process them jointly. The certainty $c[x,y]$ of a single macroblock $m[x,y]$ (with Cartesian coordinates $[x,y]$) that depends on the weights $w_t[x+i,y+j]$ of all macroblocks in a desig-

nated neighboring area (translation indicated by (i,j)) is defined as

$$c[x,y] = \sum_{j=-2}^2 \sum_{i=-2}^2 w^{ij}[x,y] \quad , \quad (1)$$

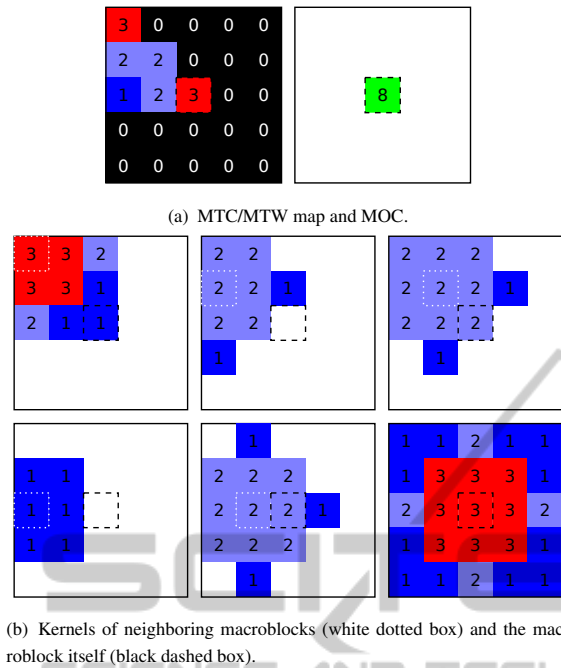
where

$$w^{ij}[x,y] = \begin{cases} w_t[x+i,y+j] & , \quad \forall i,j \in \{-1,0,1\} \\ (w_t[x+i,y+j]-1)^+ & , \\ \forall(i,j) \in \{(-2,0),(0,-2),(0,2),(2,0)\} \\ (w_t[x+i,y+j]-2)^+ & , \quad \text{otherwise} \end{cases}$$

Thereby, the operator $(\cdot)^+$ is defined as

$$+ : \mathbb{Z} \rightarrow \mathbb{N}_0, a \mapsto (a)^+ := \max(0,a) \quad .$$

According to (1) the certainty MOC of a macroblock depends on the weights MTW of its eight direct neighbors and on the weights MTW of the 16



(b) Kernels of neighboring macroblocks (white dotted box) and the macroblock itself (black dashed box).

Figure 4: Sample calculation of the MOC of a macroblock (black dashed box).

neighbors of their direct neighbors. Thereby, the values are weighted according to their distance to the current macroblock. Direct neighbors are weighted just like the macroblock itself. Neighbors in higher distance factor into the certainty MOC with decreased weight, since it is assumed that the mutual interdependence with respect to the presence of an object is also lower.

A more illustrative description of the algorithm is depicted in Figure 4. At each macroblock position (white dotted boxes in Figure 4(b)) a discrete kernel is set according to the macroblocks weight MTW. In case the weight MTW equals 0 all points of the kernel are also 0, i.e., the weight MTW of this macroblock does not affect any other macroblock. The three other kernels can be seen in Figure 3. Once the kernels of the relevant neighboring macroblocks are set, the certainty MOC of the current macroblock is calculated by summarizing all overlapping kernel values at its position (black dashed box). In the example in Figure 4(a) this equals $1 + 0 + 2 + 0 + 2 + 3 = 8$.

Note that if the current macroblock lies near the frame border, some of its neighbors will not exist. In this case the weight MTW map is extended to the required size and the weights MTW of the new border macroblocks are set to 0.

The next step of the algorithm is to segment the current frame to fore- and background. Thereto, the calculated certainty MOC map is thresholded by t .

Whether a macroblock $m[x,y]$ is part of the foreground is calculated by

$$m[x,y] = \begin{cases} 1 & , c[x,y] \geq t \\ 0 & , \text{otherwise} \end{cases}, \quad (2)$$

where 1 indicates the foreground and 0 indicates the background of the scene, which is illustrated within the binary masks in Figure 2(e) and Figure 2(g) by white and black blocks, respectively.

4.2 Box Filtering

The resulting binary mask of the segmentation process is then further refined by an $n \times n$ box filtering process. That means if most neighboring macroblocks in a surrounding $n \times n$ region (including the macroblock itself) of a single macroblock are labeled as foreground, the macroblock is also labeled as foreground, and vice versa. The purpose of this step is to eliminate very rarely occurring holes within objects and to filter out remaining single foreground labeled macroblocks. Furthermore, object edges are smoothed, as can be seen in Figure 2(g), which represents the filtered version of Figure 2(e).

5 EXPERIMENTAL RESULTS

5.1 Performance Measures

The performance of the method is measured by the following procedure. Since the analysis is frame-based, for each frame k a manually labeled ground truth states the set of pixels $S^{\text{pix}}[k]$ of the moving objects. The proposed algorithm segments macroblocks to fore- and background. This is the reason why we also defined the set of macroblocks $S^{\text{mb}}[k]$ for each frame k as ground truth. Thereby, a macroblock is denoted as foreground, if at least one of its pixels is considered foreground.

Two conventional measures are used to evaluate the results: *recall* and *precision*. For comparing set of pixels, they are defined as

$$r^{\text{pix}}[k] = \frac{N_c^{\text{pix}}[k]}{N_c^{\text{pix}}[k] + N_m^{\text{pix}}[k]}, \quad (3)$$

$$p^{\text{pix}}[k] = \frac{N_c^{\text{pix}}[k]}{N_c^{\text{pix}}[k] + N_f^{\text{pix}}[k]}, \quad (4)$$

where $N_c^{\text{pix}}[k]$ is the number of correctly detected pixels, $N_m^{\text{pix}}[k]$ is the number of missed pixels, i.e., pixels that are labeled foreground in the ground truth but

Table 2: Experimental results of several test sequences.

Sequence	r^{pix}	p^{pix}	r^{mb}	p^{mb}
<i>door</i>	0.96	0.81	0.95	0.88
<i>room1pFreeBlk</i>	0.99	0.40	0.98	0.60
<i>room2pXingDiagBlk</i>	0.97	0.47	0.95	0.65
<i>room2pXingEqMix</i>	0.58	0.50	0.57	0.64
<i>room2pXingDiagMix</i>	0.84	0.48	0.83	0.60
<i>campus4-c0</i>	0.75	0.48	0.71	0.74
<i>campus7-c1</i>	0.98	0.65	0.95	0.82
<i>laboratory4p-c0</i>	0.99	0.38	0.98	0.57
<i>laboratory6p-c1</i>	0.98	0.16	0.97	0.29
<i>terrace1-c0</i>	0.98	0.40	0.96	0.66
<i>terrace2-c1</i>	0.98	0.41	0.97	0.68

have not been detected, and $N_f^{\text{pix}}[k]$ is the number of pixels that have falsely been considered foreground.

Similarly, *recall* and *precision* on macroblock level are defined as

$$r^{\text{mb}}[k] = \frac{N_c^{\text{mb}}[k]}{N_c^{\text{mb}}[k] + N_m^{\text{mb}}[k]}, \quad (5)$$

$$p^{\text{mb}}[k] = \frac{N_c^{\text{mb}}[k]}{N_c^{\text{mb}}[k] + N_f^{\text{mb}}[k]}. \quad (6)$$

The final step to get *recall* and *precision* measures for a whole sequence is an averaging process. The pixel measures are defined as

$$r^{\text{pix}} = \frac{1}{N_{\text{frame}}} \sum_k r^{\text{pix}}[k], \quad (7)$$

$$p^{\text{pix}} = \frac{1}{N_{\text{frame}}} \sum_k p^{\text{pix}}[k], \quad (8)$$

and the macroblock level measures are defined as

$$r^{\text{mb}} = \frac{1}{N_{\text{frame}}} \sum_k r^{\text{mb}}[k], \quad (9)$$

$$p^{\text{mb}} = \frac{1}{N_{\text{frame}}} \sum_k p^{\text{mb}}[k], \quad (10)$$

where N_{frame} is the number of frames of the sequence.

5.2 Test Sequences and Setup

The algorithm has been tested with several H.264/AVC video sequences, including sequences from the data set of CVLAB (Berclaz et al., 2011) and self-created sequences. A detailed description for each test sequence is given in the Appendix.

The sequences have been encoded with variable bit rate by an own implementation of the H.264/AVC Baseline profile. The GOP size has been set to ten frames. During the segmentation process we set $t = 6$, which fits best to the defined macroblock weights MTW. For box filtering we applied a 3x3 filter.

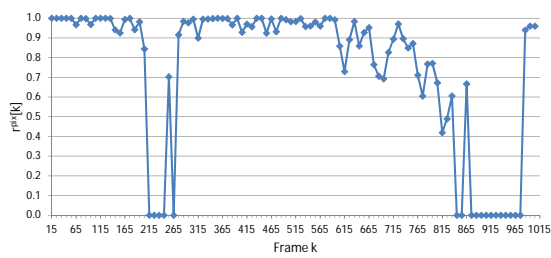
5.3 Result Discussion

An overview of the results is given in Table 2.

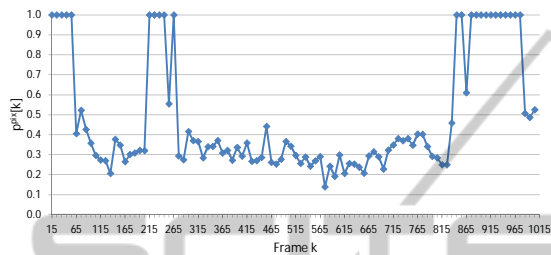
The first column r^{pix} represents the *recall* values of comparison between the ground truth in pixel accuracy with the results of the algorithm in macroblock accuracy. Although the resulting foreground masks are block-based, for the majority of sequences the method achieves 96% and above. That means that almost all foreground pixels could be detected correctly and only very little have been missed. This can also be seen in the third column r^{mb} , which represents the *recall* values of comparison between the results of the algorithm with the ground truth in macroblock accuracy. Macroblock accuracy in this scope means that each macroblock with at least one foreground pixel is regarded as foreground. In many cases, e.g., if the object is located at the edge of a macroblock row or column, this consideration will lead to more pixels labeled as foreground as their actual amount. That is the reason why values in the third column are always slightly smaller than in the first column.

For a few sequences in Table 2 the algorithm does not achieve very high *recall* values. This occurs when objects stop moving within the scene. In case an annotated object stops but is still visible, it is correctly labeled foreground in the ground truth, but most encoders will decide to use Skip mode for its macroblocks. Hence, our algorithm is not able to detect these objects anymore because they do not differ from the background. This happens in sequences *room2pXingEqMix*, *room2pXingDiagMix*, and *campus4-c0*. Figure 5 illustrates the *recall* r^{pix} and *precision* p^{pix} values for each frame with available ground truth of sequence *campus4-c0*. Approximately between frames 200 and 250 the only visible person stops moving. During this period, *recall* values drop to almost 0, while corresponding *precision* values increase to 100%. This means that on the one hand this object can admittedly not be detected correctly, but on the other hand also no false detections occur. The same behavior can be seen at the end of the sequence, where the three visible persons stop one after another to talk to each other.

The second and fourth columns in Table 2 represent the *precision* of the algorithm. The values for pixel accuracy comparison p^{pix} do not achieve that high percentage than their corresponding *recall* values. The main reason for this is that it is not possible to completely eliminate false detections with an accuracy of macroblock size. Therefore, comparing the detection results to ground truth in macroblock accuracy, as can be seen in column p^{mb} , achieves a significantly increased *precision*, up to 27 percentage points.



(a) Recall.



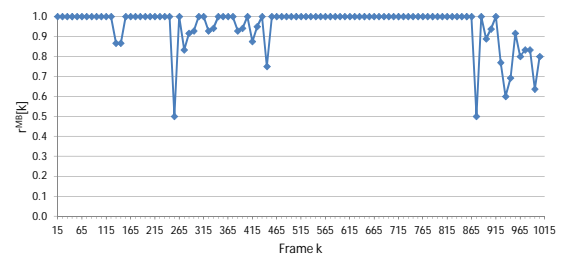
(b) Precision.

Figure 5: Recall and precision in pixel accuracy of sequence *campus4-c0*.

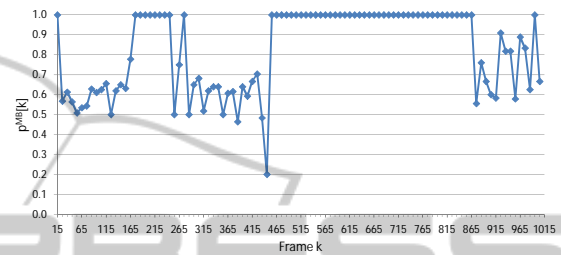
Even though macroblock accuracy comparison improves the results, *precision* values mostly do not exceed 70%. There are mainly two reasons: object shadows and image noise.

Mainly in outdoor sequences moving objects have shadows that will move accordingly. The algorithm detects these shadows as moving regions as well, because it is not possible to distinguish between actual objects and shadows on macroblock level. This leads to an increased number of false detections. Figure 6 shows *recall* and *precision* values of sequence *campus7-c1*. Within this scene, approximately between frames 175 and 250 and frames 455 and 865 no visible moving objects occur. During this period of frames both *recall* and *precision* are constantly at 100%. The latter demonstrates that during the absence of moving objects no macroblocks are falsely detected as moving regions, i.e., in this setup mostly false detections are caused by shadows.

The second reason for false detections is image noise. The video content of the sequences *laboratory4p-c0* and *laboratory6p-c1* is quite noisy. In such sequences the difference between frames with similar content is significantly larger than in high-quality sequences. Therefore, macroblocks in not partitioned or Skip modes are rarely used during the encoding process, i.e., the algorithm will not only detect the actual objects as moving regions but significantly noisy regions as well.



(a) Recall.



(b) Precision.

Figure 6: Recall and precision in macroblock accuracy of sequence *campus7-c1*.

5.4 Processing Speed

The processing speed of the algorithm depends on the video resolution of the test sequence and the number of moving objects that are present within the scene. Several measurements pointed out that our C++ implementation (without code optimizations or parallel processing) is able to process about 3600 frames per second of sequences in CIF resolution and 1900 frames per second of sequences in VGA resolution, measured on an Intel® Core™ i5-2520M CPU @ 2.5 GHz with 4 GB RAM. The average number of processed frames per second for each sequence is given in Table 3.

6 CONCLUSIONS

In this paper, we presented a novel compressed domain moving object detection method based on analyzing macroblock types only. The frame-based algorithm extracts and evaluates the type of each single macroblock. Thereby, the macroblocks get assigned a moving object certainty that is calculated by factoring in the types of neighboring macroblocks. The results could demonstrate that this approach reaches suitable detection rates within the limits of compressed domain processing, despite its very low complexity. This enables its use as an adequate preselection step within a multi-tier parallel processing system. It is

Table 3: Average number of processed frames per second.

Resolution	Sequence	Frames per Second
CIF	<i>door</i>	7587.82
	<i>campus4-c0</i>	4636.33
	<i>campus7-c1</i>	4878.68
	<i>laboratory4p-c0</i>	1953.28
	<i>laboratory6p-c1</i>	1366.58
	<i>terrace1-c0</i>	2496.29
	<i>terrace2-c1</i>	2628.99
	Average:	3649.71
VGA	<i>room1pFreeBlk</i>	1693.33
	<i>room2pXingDiagBlk</i>	1921.42
	<i>room2pXingEqMix</i>	2202.27
	<i>room2pXingDiagMix</i>	1820.84
		Average:

envisioned to further enhance the method by refining the segmentation process to be able to eliminate inappropriate objects caused by, e.g., shadows, and exploiting temporal dependencies between consecutive frames. The latter also enables the algorithm to track moving objects.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme ([FP7/2007-2013]) under grant agreement no. 285248 (FI-WARE).

REFERENCES

- Berclaz, J., Fleuret, F., Turetken, E., and Fua, P. (2011). Multiple Object Tracking Using K-Shortest Paths Optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1806–1819.
- Comaniciu, D. and Meer, P. (2002). Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.
- Fei, W. and Zhu, S. (2010). Mean Shift Clustering-based Moving Object Segmentation in the H.264 Compressed Domain. *IET Image Processing*, 4(1):11–18.
- Laumer, M., Amon, P., Hutter, A., and Kaup, A. (2011). A Compressed Domain Change Detection Algorithm for RTP Streams in Video Surveillance Applications. In *Proc. IEEE 13th Int. Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6.
- Mak, C.-M. and Cham, W.-K. (2009). Real-time Video Object Segmentation in H.264 Compressed Domain. *IET Image Processing*, 3(5):272–285.

- MPEG (2010). ISO/IEC 14496-10:2010 - Coding of Audio-Visual Objects - Part 10: Advanced Video Coding.
- Poppe, C., De Bruyne, S., Paridaens, T., Lambert, P., and Van de Walle, R. (2009). Moving Object Detection in the H.264/AVC Compressed Domain for Video Surveillance Applications. *Journal of Visual Communication and Image Representation*, 20(6):428–437.
- Porikli, F., Bashir, F., and Sun, H. (2010). Compressed Domain Video Object Segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(1):2–14.
- Qiya, Z. and Zhicheng, L. (2009). Moving Object Detection Algorithm for H.264/AVC Compressed Video Stream. In *Proc. Int. Colloquium on Computing, Communication, Control, and Management (CCCM)*, volume 1, pages 186–189.
- Szczerba, K., Forchhammer, S., Stottrup-Andersen, J., and Eybye, P. T. (2009). Fast Compressed Domain Motion Detection in H.264 Video Streams for Video Surveillance Applications. In *Proc. Sixth IEEE Int. Conf. on Advanced Video and Signal Based Surveillance (AVSS)*, pages 478–483.
- VCEG (2011). H.264: Advanced Video Coding for Generic Audiovisual Services.
- Verstockt, S., De Bruyne, S., Poppe, C., Lambert, P., and Van de Walle, R. (2009). Multi-view Object Localization in H.264/AVC Compressed Domain. In *Proc. Sixth IEEE Int. Conf. on Advanced Video and Signal Based Surveillance (AVSS)*, pages 370–374.
- Wang, R., Zhang, H.-J., and Zhang, Y.-Q. (2000). A Confidence Measure Based Moving Object Extraction System Built for Compressed Domain. In *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, volume 5, pages 21–24.

APPENDIX

A detailed description for each test sequence is given in Table 4 and Table 5. Column 'GT Distance' indicates the distance of frames with available ground truth.

Table 4: Detailed description of self-created test sequences.

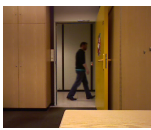


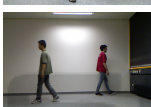
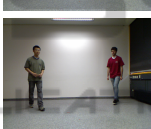



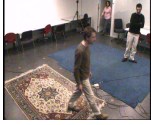

Sequence	N^{frame}	Resolution	FPS	GOP Size	GT Distance	Sample Frame
<i>door</i>	794	352x288	30	10	1	
<i>room1pFreeBlk</i>	423	640x480	30	10	1	
<i>room2pXingDiagBlk</i>	196	640x480	30	10	10	
<i>room2pXingEqMix</i>	246	640x480	30	10	10	
<i>room2pXingDiagMix</i>	174	640x480	30	10	10	

Table 5: Detailed description of CVLAB test sequences.

Sequence	N^{frame}	Resolution	FPS	GOP Size	GT Distance	Sample Frame
<i>campus4-c0</i>	1005	352x288	25	10	10	
<i>campus7-c1</i>	1005	352x288	25	10	10	
<i>laboratory4p-c0</i>	1005	352x288	25	10	10	
<i>laboratory6p-c1</i>	1005	352x288	25	10	10	
<i>terrace1-c0</i>	1005	352x288	25	10	10	
<i>terrace2-c1</i>	1005	352x288	25	10	10	