of the motion estimation problem is as follows. Given two blocks of 
two frames, we find a block from the second frame that is closest in appearance to the first 
block. The simplest abstraction is to consider the source block and a search window larger than 
that block. Or we may employ faster but approximate strategies, such as logarithmic search [1], to find a 
subblock that is close in appearance to the source block but is not necessarily the closest.

Whatever the search strategy, evaluating the \( l_1 \) metric on pixels of full intensity resolution is computationally expensive. To overcome 
this obstacle, we propose to transform the current and reference frame to frames of binary-valued pixels. We then apply one of the conven-
tional search strategies to these frames. The \( l_1 \) metric then amounts to computing the exclusive-or of a sequence of bits and adding up the 
number of ones in the result. This can result in substantial savings in software implementations 
as well as reduced complexity and power consumption in hardware implementations. Our experiments show 
that a careful choice of the one-bit transform can realize these gains with a small sacrifice in compression efficiency.

Previously, a one-bit modification of the \( l_1 \) metric was proposed in [3], and we will compare our approach to theirs later in this 
paper. Recently and independently, Feng et al. [4] proposed a one-
binary transform similar to ours, but exploited it as a preprocessing 
step to exhaustive search with the \( l_1 \) metric. Their approach differs 
from ours on three counts. 1) They use the block mean as the 
threshold. However, we have found that the block mean does not offer the best results in our experiments. 2) The complexity of 
their strategy is roughly six times that of ours. 3) Their strategy is adaptive and not suited for simple hardware implementation at low 

power consumption. In [5], Mizuki et al. describe a binary block 
matching architecture where block matching is performed on the 
binary edge maps of the current and the reference frames. They also present a custom hardware implementation that includes circuitry 
for edge detection and a two-dimensional (2-D) array of elementary 

processors, where the number of elementary processors is equal to 
the number of candidate blocks for full-search motion estimation. 

Compared to conventional block matching schemes, they estimate 
that binary block matching for motion estimation reduces the silicon 
area required by a factor of five.

In Section II, we establish the preliminaries and define the problem; 
In Section III, we give details of the proposed one-bit transform; in 
Section IV, we present a custom architecture for the one-bit motion 

estimation strategy, and in Section V, we present experimental results 
from applying our technique to sample video sequences.

II. Preliminaries

Let \( s \) denote the source block of \( b \times b \) pixels, with \( s_{i,j} \) being 
the pixel at row \( i \) and column \( j \). Similarly, let \( w \) denote the 


\text{distance between two blocks can be measured by a number of different metrics} [2], and typically the \( l_1 \) metric (mean absolute deviation) is used. Using this metric and a search strategy, we can 
evaluate candidate subblocks of the search window to find the 


\text{subblock that is close in appearance to the source block but is not necessarily the closest.}

Whatever the search strategy, evaluating the \( l_1 \) metric on pixels of full intensity resolution is computationally expensive. To overcome 
this obstacle, we propose to transform the current and reference frame to frames of binary-valued pixels. We then apply one of the conven-
tional search strategies to these frames. The \( l_1 \) metric then amounts to computing the exclusive-or of a sequence of bits and adding up the 
number of ones in the result. This can result in substantial savings in software implementations as well as reduced complexity and power consumption in hardware implementations. Our experiments show 
that a careful choice of the one-bit transform can realize these gains with a small sacrifice in compression efficiency.

Previously, a one-bit modification of the \( l_1 \) metric was proposed in [3], and we will compare our approach to theirs later in this 
paper. Recently and independently, Feng et al. [4] proposed a one-
binary transform similar to ours, but exploited it as a preprocessing 
step to exhaustive search with the \( l_1 \) metric. Their approach differs 
from ours on three counts. 1) They use the block mean as the 
threshold. However, we have found that the block mean does not offer the best results in our experiments. 2) The complexity of 
their strategy is roughly six times that of ours. 3) Their strategy is adaptive and not suited for simple hardware implementation at low 

power consumption. In [5], Mizuki et al. describe a binary block 
matching architecture where block matching is performed on the 
binary edge maps of the current and the reference frames. They also present a custom hardware implementation that includes circuitry 
for edge detection and a two-dimensional (2-D) array of elementary 

processors, where the number of elementary processors is equal to 
the number of candidate blocks for full-search motion estimation. 

Compared to conventional block matching schemes, they estimate 
that binary block matching for motion estimation reduces the silicon 
area required by a factor of five.

In Section II, we establish the preliminaries and define the problem; 
In Section III, we give details of the proposed one-bit transform; in 
Section IV, we present a custom architecture for the one-bit motion 

estimation strategy, and in Section V, we present experimental results 
from applying our technique to sample video sequences.

II. Preliminaries

Let \( s \) denote the source block of \( b \times b \) pixels, with \( s_{i,j} \) being 
the pixel at row \( i \) and column \( j \). Similarly, let \( w \) denote the 


\text{distance between two blocks can be measured by a number of different metrics} [2], and typically the \( l_1 \) metric (mean absolute deviation) is used. Using this metric and a search strategy, we can 
evaluate candidate subblocks of the search window to find the 


\text{subblock that is close in appearance to the source block but is not necessarily the closest.}

Whatever the search strategy, evaluating the \( l_1 \) metric on pixels of full intensity resolution is computationally expensive. To overcome 
this obstacle, we propose to transform the current and reference frame to frames of binary-valued pixels. We then apply one of the conven-
tional search strategies to these frames. The \( l_1 \) metric then amounts to computing the exclusive-or of a sequence of bits and adding up the 
number of ones in the result. This can result in substantial savings in software implementations as well as reduced complexity and power consumption in hardware implementations. Our experiments show 
that a careful choice of the one-bit transform can realize these gains with a small sacrifice in compression efficiency.

Previously, a one-bit modification of the \( l_1 \) metric was proposed in [3], and we will compare our approach to theirs later in this 
paper. Recently and independently, Feng et al. [4] proposed a one-
binary transform similar to ours, but exploited it as a preprocessing 
step to exhaustive search with the \( l_1 \) metric. Their approach differs 
from ours on three counts. 1) They use the block mean as the 
threshold. However, we have found that the block mean does not offer the best results in our experiments. 2) The complexity of 
their strategy is roughly six times that of ours. 3) Their strategy is adaptive and not suited for simple hardware implementation at low 

power consumption. In [5], Mizuki et al. describe a binary block 
matching architecture where block matching is performed on the 
binary edge maps of the current and the reference frames. They also present a custom hardware implementation that includes circuitry 
for edge detection and a two-dimensional (2-D) array of elementary 

processors, where the number of elementary processors is equal to 
the number of candidate blocks for full-search motion estimation. 

Compared to conventional block matching schemes, they estimate 
that binary block matching for motion estimation reduces the silicon 
area required by a factor of five.

In Section II, we establish the preliminaries and define the problem; 
In Section III, we give details of the proposed one-bit transform; in 
Section IV, we present a custom architecture for the one-bit motion 

estimation strategy, and in Section V, we present experimental results 
from applying our technique to sample video sequences.
video coding, our one-bit motion estimation strategy consists of the following steps (Fig. 2): 1) apply the one-bit transform $Q$ to both the current frame and the reference frame; 2) use any motion-vector search strategy in combination with the metric defined in (2).

For two one-bit images this metric reduces to

$$\|u, v\| = \frac{1}{b^2} \sum_{i,j} u_{i,j} \oplus v_{i,j}$$

(2)

where $\oplus$ denotes the exclusive-or operation. The problem of motion estimation is to find the position $x, y$ so that the subblock $w_{x,y}$ is closest to the source block $s$, in that $\|s, w_{x,y}\|$ is minimum over all subblocks of $w$.

### III. The One-Bit Strategy

We now construct a transform $Q$ that maps a frame of multivalued pixels to a frame of binary-valued pixels. $Q$ is defined with respect to a convolution kernel $K$ and is denoted by $Q_K$. Let $F$ denote a frame and let $\hat{F}$ denote the filtered version of $F$ obtained by applying the convolution kernel $K$ to $F$. Let $G = Q_K(F)$ be the frame obtained by applying $Q_K$ to $F$. The pixels of $G$ are given by

$$G_{i,j} = \begin{cases} 1, & \text{if } F_{i,j} \geq \hat{F}_{i,j} \\ 0, & \text{otherwise.} \end{cases}$$

(3)

In this paper, we use the $17 \times 17$ convolution kernel $K$ given below

$$K_{i,j} = \begin{cases} \frac{1}{255}, & \text{if } i, j \in \{1, 4, 8, 12, 16\} \\ 0, & \text{otherwise.} \end{cases}$$

(4)

The motivation behind our method rests on the observation that the edges in an image are key to accurate motion estimation. A simple way to extract the edges is to carry out a high-pass thresholding, that is, compare the frame pixel by pixel to a high-pass filtered version of the frame, and threshold the pixels to zero or one, depending on the outcome of the comparison. Unfortunately, this would also cause the thresholded frame to track the high-frequency noise in the original frame. To overcome this, we use band-pass thresholding, wherein the smoothed version is a band-pass filtered version of the original frame, so that the thresholded frame represents the mid-frequency content of the original frame. The convolution kernel that we propose is motivated by this consideration, as well as the need to minimize the number of arithmetic operations. For comparison, [3] uses a block averaging kernel, which corresponds to using low-pass thresholding.

The operations for the one-bit transform are shown in Fig. 1. Note that there is no global threshold for all pixels in a frame. For video coding, our one-bit motion estimation strategy consists of the following steps (Fig. 2): 1) apply the one-bit transform $Q$ to both the current frame and the reference frame; 2) use any motion-vector search strategy in combination with the metric defined in (2).
finding \( k \) and \( l \in [0, 15] \) for which
\[
D(k, l) = \sum_{i=0}^{15} f(R_i, V_{i+k, l})
\]
is minimized, where \( f(\cdot) \) is defined as
\[
f(R_i, V_{i+k, l}) = \sum_{j=0}^{15} r_{i,j} \otimes v_{i+k, j+l}.
\]
That is, the \( f(\cdot) \) function computes the number of bits for which there is a match between the \( R \) and \( V \) binary vectors.

From (5), each \( V \) vector is used in the computation of multiple distortion values. For example, \( V_{15,0} \) (Fig. 3) is used in the computation of 16 distortions, namely \( D(0, 0), D(1, 0), \ldots, D(15, 0) \). Hence, if the \( V \) vectors are distributed to multiple processors, then one can compute multiple distortions in parallel.

Fig. 4 shows such an implementation using an array of 16 processors. This is similar to the implementation in [6], except that each processor operates on 16-b vectors instead of on 8-b pixels. The architecture of each processor is shown in detail in Fig. 5. The \( f(\cdot) \) function defined in (6) is computed using two 8-b exclusive-or arrays, a dual-port look-up table (LUT) with 256 entries, and a 4-b adder. The look-up table yields the total number of ones (or matches) at the output of each exclusive-or array. One xor-array operates on the eight most significant bits of the \( R \) and \( V \) vectors and the other one on the eight least significant bits.

Table I shows in more detail the data flow of operations on processors PE-0, PE-1, and PE-15 for the computation of the first 16 distortion values. At \( t = 0 \), only PE-0 is active with binary vectors \( R_0 \) and \( V_{0,0} \) as inputs. At \( t = 1 \), PE-0 processes \( R_1 \) and \( V_{1,0} \), and PE-1 processes \( R_0 \) and \( V_{1,0} \). Following this approach, \( D(0, 0) \) in PE-0, will be ready after \( t = 15 \), and all the first 16 distortion values will be computed in 16 + 15 cycles. However, as shown in Table I, by using two ports for the search memory, processing of the next set of distortions can begin at \( t = 16 \). As shown in Fig. 5, a multiplexor in each processor selects the appropriate input from the search memory. The complete set of 256 distortion values can then be computed in 16 + 15 = 271 cycles. In contrast, the traditional architecture [6] requires 4111 cycles. Thus, the one-bit transform allows for a roughly 15:1 speed improvement. This is consistent with the fact that at each cycle we now process 16 binary pixels instead of one 8-b pixel.

For higher throughput, multiple arrays could be used. For example, in pipelined mode, two such arrays (which is equivalent to using a 16-processor array of 32-b processors) could compute all distortions in 128 cycles.
Consider now the case of motion estimation using a search range of 
$[-16, 15]$ pixels. Then, the 1-b search window is $47 \times 47$ pixels and
we need to compute 1024 distortion values. Since our architecture
can compute 16 distortion values in 16 cycles, we can estimate that
the 16-processor linear array will require now $16 \times 64 + 15 = 1039$
cycles to compute all 1024 distortion values.

V. EXPERIMENTAL RESULTS

Custom architectures may provide the highest level of performance
for motion estimation, however, binary block matching schemes are
also ideally suited for software-only implementations on a general
purpose processor. We studied the performance of the following
search strategies on several sequences from the MPEG video test
suite.

1) $Full_8$: Full search on 8-b data, $l_1$ metric.
2) $Log_{8}$: Logarithmic search on 8-b data, $l_1$ metric.
3) $Full_1$: Full search after 1-b transform, distance metric of (2).
4) $Log_{1}$: Logarithmic search after 1-b transform, distance metric
   of (2).

Table II shows the computational complexity of the four strategies
for two different 32-b architectures. The first one, referred to as 32-b
ops, has a native instruction for counting the population of ones in a
register. This allows for 32 binary comparisons per instruction. The
second one, referred to as 1-b ops, is a traditional one where only
one binary comparison per instruction is performed. We also include
estimates for the pixel distance criterion (PDC) metric, which is the
scheme proposed in [3]. Note that the calculations for the PDC metric
in this table is for the full-search scheme. For $Full_1$ and $Log_1$, the
additional expense of the filtering and compare operations of Fig. 1 is
also included in the number of operations. From Table II, we note at
least a 200-fold reduction in complexity for $Log_1$ compared to $Full_8$.
The effectiveness of each of the search strategies can be measured in

Fig. 6. Motion-compensated prediction residual for the Miss America sequence using various search schemes.
several search strategies. To improve the performance of the Log

8-b data (Log

56

candidate logarithmic search [Log

49

a multicandidate logarithmic search scheme. For example, in a three-

scheme, we also examined simple extensions to this approach, namely

the motion-compensated difference frames. We compute PSNR as

terms of the peak signal-to-noise ratio (PSNR) and entropy values of

the motion-compensated difference frames. We compute PSNR as

\[ 10 \log_{10} \left( \sum_{i,j} \left( \frac{255}{F_{i,j}} \right)^2 \right) \text{dB} \] (7)

where \( F \) is the motion-compensated prediction residual image. Table III shows the PSNR of several video sequences, averaged over 100 motion-compensated difference frames for each sequence. It is clear that the performance of full search after the 1-b transform (\( \text{Full}_1 \)) is comparable to or better than that of logarithmic search on 8-b data (Log

56

). Also, for typical video sequences, the performance of Log

49

compares quite favorably with \( \text{Full}_s \), considering the 200-

fold reduction in complexity. In Table IV, we show the entropy of

the motion-compensated difference image. From this table, we note

that the one-bit transform scheme with a suboptimum search strategy

yields only 0.5 dB worse than \( \text{Full}_s \) and the one-bit, three-candidate logarithmic search [Log

56

(3)] yields only 1.65 dB lower performance than the exhaustive full-search (\( \text{Full}_s \)) method, while its complexity is 65 times lower.

VI. CONCLUSIONS

We presented a motion estimation strategy for digital video based

on a one-bit transform and gave an architecture for its hardware

implementation. The strategy can effectively integrate low complexity

search schemes, such as logarithmic search, to obtain complexity

reductions as large as 200 fold relative to classical exhaustive search.

The complexity reduction can translate into proportionate reduction

in power consumption of custom hardware. Experimental results

indicate that the reduced arithmetic complexity is accompanied by

acceptable levels of performance degradation.

REFERENCES


May 1990.


matching motion estimation algorithm using bit-plane matching,” in


binary block matching architecture with reduced power consumption


6, pp. 3248–3251.


the motion compensation block-matching algorithm,” IEEE Trans. Circuits


h263\_software/.