AUTOMATIC TEST METHODS
FOR IMAGE AND VIDEO
VERIFICATION

MAGNUS HÅKANSSON
SVERKER RASMUSON

Master's thesis
2013:E28

Lund University
Faculty of Engineering
Centre for Mathematical Sciences
Mathematics
Automatic Test Methods for Image & Video Verification

Magnus Håkansson & Sverker Rasmuson

Supervisors: Magnus Oskarsson & Zoltan Michis

College Of Engineering, Lund University
DEPARTMENT OF MATHEMATICS
Abstract
In this thesis four methods for automatic verification of images and video on mobile platforms are developed. Both the case of recording images and video and the case of viewing images and video on the mobile LCD screen are considered. The first method is used to test the zoom function of the camera. It uses SURF descriptors along with clustering and histograms to determine which of six discrete zoom levels the current frame belongs to. The second method identifies color effects and color anomalies using histograms. The third method determines if the auto focus works correctly by measuring the average length of edges in the image. The fourth method is an artifact detection scheme using a non-reference implementation of the SSIM metric, used in conjunction with a for this purpose specially designed test setup. Together these methods form a tool kit for detecting the most common errors to occur in images and video during the development stage of mobile platforms.
## Contents

1 Background  
1.1 Problem Formulation  
1.2 Nomenclature  
1.3 Report Structure  
1.4 Notes about implementation  

2 Theory  
2.1 Zoom function  
  2.1.1 K-means clustering  
  2.1.2 Scale-invariant feature transform (SIFT)  
    2.1.2.1 Motivation  
    2.1.2.2 Scale-space extrema detection  
    2.1.2.3 Key point localization  
    2.1.2.4 Orientation assignment  
    2.1.2.5 Key point descriptor  
  2.1.3 Speeded Up Robust Features (SURF)  
2.2 Blur detection  
  2.2.1 Sobel Edge Detector  
  2.2.2 Canny Edge Detection  
  2.2.3 No-reference blur metric CPBD  
2.3 Artifacts - Non-Reference Methods  
  2.3.1 GMM  
2.4 Artifacts - Reference Methods  
  2.4.1 SSIM  
  2.4.2 Fast SSIM  
  2.4.3 Denoising  
    2.4.3.1 General  
    2.4.3.2 Morphological  
  2.4.4 Segmentation  
    2.4.4.1 Find Object Contour  
    2.4.4.2 Polygon Simplification  
    2.4.4.3 Perspective Transform  

3 Methodology  
3.1 Zoom function  
  3.1.1 Outline  
  3.1.2 Test setup  
  3.1.3 Reference phase
3.1.4 Testing phase ........................................ 36
3.2 Colors .................................................. 39
  3.2.1 Reference phase ................................. 41
  3.2.2 Testing phase ................................. 43
3.3 Blur .................................................. 44
3.4 Artifacts - Non-Reference Methods .............. 44
  3.4.1 Rate Distortion ................................. 44
  3.4.2 Normalized Compression Distance ............ 45
3.5 Artifacts - Reference Methods .................. 45
  3.5.1 Test Image ....................................... 46
  3.5.2 Algorithm ....................................... 47
  3.5.3 Setup Camera Test ............................. 50
  3.5.4 Setup Screen Test ............................. 50
  3.5.5 Pre- and post-processing ..................... 51

4 Results and Discussion .......................... 53
  4.1 Zoom .............................................. 54
  4.2 Colors .............................................. 54
  4.3 Blur ............................................... 55
  4.4 Artifacts .......................................... 59
    4.4.1 Non-Reference Methods ..................... 60
    4.4.2 Reference Methods ......................... 60

5 Improvements and Future Work .................. 65
1 Background

At the department of Multimedia Verification at ST-Ericsson, the multimedia aspects of mobile platforms are tested. The goal is to find errors in the platform that are noticeable from an end-users perspective. Multimedia incorporates images, video, sound, camera, browsing etc. with both playing/showing and recording when it comes to the first three.

Naturally, one likes to automate these tests whenever possible to increase the efficiency and repeatability of the testing. This has already been done to a large extent for most of the test cases, especially with regards to audio. This is however not true when it comes to images and video. Today almost all testing of video, imaging and the camera is done manually or semi-manually.

There are many reasons why the automatic testing of images and video are lagging behind. Images and video have several inherent problems that make it harder to deal with than e.g. audio.

- Image analysis methods are generally complex and puts high requirements on the used equipment.
- The possible errors represent a large number of diverse problems that are not possible to capture with a single method.
- It is not at all obvious what an optimal test setup should look like.
- Hard to get end-user perspective on the mobile screen without using some form of external device (e.g. film the screen with a camera).

1.1 Problem Formulation

The problem that we want to solve is the following: develop automatic test tools that evaluate the quality of images and video from a user perspective on mobile platforms in the development stage. This incorporates both the mobile camera and the mobile LCD screen e.g. both playing and recording content. It also includes various effects specified in Android OS as well as camera functions such as zoom and auto focus.
1.2 Nomenclature

Reference methods refer to methods that use a reference image or video for comparison with potentially erroneous test content. Non-reference methods are methods that analyze the content as is, with the test image or video as the only input. See Figure 2 for an illustration of these concepts.

![Figure 2: The difference between reference and non-reference methods in image and video processing](image)

1.3 Report Structure

Because of the diverse type of errors that the proposed tools should be able to detect, several different test methods have been developed with their own distinct purposes. Thus the thesis contains independent components, not really related to each other except in the purpose of detecting some form of error in images or video. To simplify for the reader and keep the line of argument clear, the following sections are divided into parts corresponding to these different test methods. The above-mentioned parts are regarded as separate and should be possible to read independent of each other. In Section 4.4 some notes about the relationship of the test methods are given.

1.4 Notes about implementation

The algorithms have been written in Python 2.7 using NumPy 1.7, with some minor parts written in C. The open-source image analysis library OpenCV
2.4 has also been used extensively. For image respective video editing the open-source software GIMP and Kdenlive has been used. The mobile platform used for evaluation has been a ST-Ericsson L8540 with a 5 Megapixel camera and Android OS 4.1. The camera could record in 1080p.
2 Theory

Many of the algorithms described in this thesis have a rather complex structure, using many subfunctions from e.g. OpenCV and other libraries. Therefore this section can be rather cumbersome, and readers are advised to mainly use it as a reference for later parts of the thesis.

If not otherwise mentioned, the referred images in this section are considered as being grayscale. An image is denoted as $I$, with two indices $I(x,y)$ for matrix indexing, or one index $I(x)$ for linear indexing i.e. when all image pixels are flattened to a single array.

2.1 Zoom function

The following chapters cover those methods that are used in chapter 3.1, where the zoom function is validated.

2.1.1 $K$-means clustering

$K$-means clustering is a technique from data mining. The aim is to partition $n$ observations into $k$ partitions. If denoting the clusters (sets of observations) as $S = \{S_1, S_2, \ldots, S_k\}$ and the mean of points for cluster $S_i$ as $\mu_i$ (centroid), the algorithm tries to find the solution to:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||_2,$$

where $x_j \in S_i$ are the observations in cluster $S_i$. In other words, it finds a partition so that the intra cluster distance is minimized. The metric used is Euclidean distance. One practical issue with $K$-means is that it is NP-hard[1]. Thus several heuristic algorithms have been developed. The most commonly used is the Standard algorithm (Lloyd’s algorithm) [2] which is an iterative algorithm:
initialize:

$\mu_1^{(1)}, \ldots, \mu_k^{(1)}$: Centroids for the clusters
do:

Assignment step: Assign every observation to the cluster with the closest centroid, i.e.

$$S_i^{(t)} = \left\{ x_p : ||x_p - \mu_i^{(t)} ||_2 \leq ||x_p - \mu_j^{(t)} ||_2 \quad \forall \ 1 \leq j \leq k \right\}$$

Update step: Compute new centroids given the new cluster assignments.

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

until The assignments will not change any more, i.e.

$$S_i^{(t+1)} = S_i^{(t)} \quad \forall \ 1 \leq i \leq k$$
end

The initializing can be done in several ways. The most commonly used methods are Forgy and Random Partition [3]. The Random Partition method randomly assigns each observation to a cluster. Given this random assignment, the initial centroids are computed. The Forgy method randomly assigns $k$ observations as the initial centroids. The different methods can produce different results. The Forgy method tends to evenly distribute the initial centroids across the whole data set. While the initial centroids with the Random Partition method tend to be more centered in the data set.

The Standard algorithm will always converge to a local minimum for equation 1. However it is not guaranteed to find the global minimum. To increase the chances of finding the global minimum one runs the algorithm several times with different initializing.

In Figure 3 an example of $k$-means clustering is shown.
2.1.2 Scale-invariant feature transform (SIFT)

2.1.2.1 Motivation

A common task in computer vision is image matching and object recognition. One way of doing this is by describing the image in terms of features[4], which provides for a reduced information set instead of using the whole image itself. So for example if we have an image of an apple, we want to find the “interesting points”, or so called “key points”, of that image. Examples of key points can be corners, edges or other high-contrast regions of the image. When we are doing e.g. object recognition we want to compare the key points from an unknown image with the key points from the apple, thus being able to determine if there is an apple in the unknown image. Obviously we want to detect an apple in the new image regardless of its spatial location. Thus we want to describe the key points without using their spatial location. Subsequently we want to assign an information vector, also called “descriptor”, for every key point. Furthermore we want to be able to
detect apples regardless of the size, noise and illumination. Thus we also want the descriptors to be independent of these parameters as well.

In 1999 David Lowe[4] developed an algorithm that finds such key points and assign them descriptors which are invariant to uniform scaling, orientation and partially invariant to affine distortions and illumination changes. The method is called *Scale-invariant feature transform (SIFT)*.

### 2.1.2.2 Scale-space extrema detection

The first step of the SIFT-algorithm is scale-space extrema detection. This is done by building a Gaussian pyramid, a pyramid of images. Furthest down in the pyramid you have the original image which is smoothed with a Gaussian kernel. As you move up the pyramid the variance of the applied Gaussian kernel is increasing. Assume that the $n$ lowest images are all of the same scale (size). The first $n$ number of images are called an octave, and in particular they are called the first octave. The $(n + 1)$’th image is then a down-sampled version of the $n$’th image. The $(n + 2)$’th image is of the same size as the $(n + 1)$’th but with a Gaussian blur applied. The $(n + 1)$’th to the $(n + n)$’th images are called the second octave. As in the first octave, the applied Gaussian kernel has an increasing variance as you move up. This is then repeated depending on how many octaves one would like to have. The number of images per octave, $n$, is also specified by the user. Figure 4 illustrates the concept of a Gaussian pyramid.
Furthermore we also want to compute “Difference of Gaussians” (DoG) in every octave, i.e. the difference between two consecutive images in an octave is computed. This can be formulated as:

\[ D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \quad (2) \]
\[ L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y), \quad (3) \]
\[ G(x, y, k\sigma) = \frac{1}{\sqrt{2\pi(k\sigma)^2}} e^{-\left(\frac{x^2+y^2}{2(k\sigma)^2}\right)}, \quad (4) \]

where \( I \) is an image, \( x \) and \( y \) are image coordinates, \( \sigma \) is the deviation for the Gaussian kernel and \( k \) depends on the number of DoG’s per octave. Here we let the variance increase by a factor of two in every octave.

For every DoG the pixels are identified as local minimum/maximum by comparing it to its eight adjacent pixels in the current DoG and its nine neighbors in the upper and lower DoG, i.e. a 26 pixel neighborhood. If the current pixel’s value is greater (or smaller) than all of its 26 neighbors it is identified as an extreme pixel. Assuming one wants to compute extrema detection on \( s \) number of DoG’s per octave, one thus needs to compute \( s + 2 \)
DoG’s per octave and every octave must consist of \( s + 3 \) Gaussian blurred images. One can then compute \( k \) as: \( k = 2^{1/s} \).

### 2.1.2.3 Key point localization

The above section will provide key points candidates to the SIFT algorithm. However, the scale-space extrema detection will render too many of those as many are very unstable (occurred because of noise and will not be detected in another image of the same object). Firstly, key points candidates with a low contrast will be disregarded. Secondly, we want to remove the key points candidates at edges. The reason is that when one is executing image matching, the edges are not unique for matching. Thus, one wants to remove those candidates in favor of other candidates, such as corners.

### 2.1.2.4 Orientation assignment

Every key point is now assigned an orientation which is based on local image gradient directions. This step will ensure that the key point descriptors are invariant to image rotation. The computations are performed on the Gaussian blurred images in the pyramid, \( L \). The computation of the orientation, \( \theta(x, y) \), and magnitude, \( m(x, y) \), at a specific scale and coordinate is done by:

\[
\begin{align*}
    m(x, y) &= \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \\
    \tan(\theta(x, y)) &= (L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y))
\end{align*}
\]  

(5)

(6)

For every key point, an orientation histogram is assigned. The histograms have 36 bins representing 0-360 degrees in steps of 10 degrees. The neighbouring pixels of the key point are added to the histogram, however, they are weighted by their gradient magnitude and by a Gaussian-weighted circular window around the key point. The Gaussian window has a standard deviation which is 1.5 of the standard deviation of the Gaussian kernel that was used on that scale. The orientation with the greatest histogram bin is assigned as orientation to that key point. If there are bins that are within 80% of that peak, they are also assigned to that key point, i.e. there can be multiple key points on the same location.
2.1.2.5 Key point descriptor

Every key point will now be assigned a descriptor that is distinctive. It is also designed so that the descriptors are invariant to illumination, 3D viewpoint, etc.

They are computed as follows. A circular Gaussian filter is applied on a $16 \times 16$ neighbourhood around the key point. For every pixel the gradient magnitudes and orientations are computed. For every non-overlapping $4 \times 4$ subregion in this $16 \times 16$-neighbourhood, orientation histograms are computed. The orientation histograms have eight bins and the size of the bins are proportional to the gradient magnitudes of the pixels within the subregion. In total there are 16 subregions (of size $4 \times 4$) and the histograms (with 8 bins each) of the subregions are used as the descriptor, which implies that the descriptors are of length 128 ($4 \times 4 \times 8$). The descriptors are normalized to unit length. To avoid non-linear illumination changes the values in the normalized descriptor are thresholded so that no value is larger than 0.2. The descriptor is then renormalized.

2.1.3 Speeded Up Robust Features (SURF)

Speeded Up Robust Features is another feature detector that was developed by Herbert Bay et al. in 2006 [5]. It is based on the SIFT (see chapter 2.1.2) approach but it runs several times fast than the SIFT detection. Like the SIFT descriptors the SURF descriptors are invariant to several transformations such as scale, illumination, spatial location and rotation. However the rotation invariance can be left out in those cases that it is not necessary.

In contrary to the SIFT method, the SURF method does not use a Gaussian pyramid with down sampling for detecting the key points in the scale space analysis. Instead the images have the same resolution on all scales in the pyramid, but the Gaussian masks are increased in size. Lowe used DoG’s as an approximation for Laplacian of Gaussian. Bay used a greater approximation by using box filters to approximate second order Gaussian derivatives. These box filters can be computed fast by utilizing integral images (see section 2.4.2). The localization of the key points are done by studying the determinant of the Hessian matrices in every scale at every pixel. This determinant can be approximated using the box filters.

When computing the descriptors the SURF method uses Haar-wavelet re-
responses around the key points. These computations can also be done using integral images, thus saving computational time.

The SURF-descriptors are of length 64, which allows for faster matching compared to the SIFT-descriptors (length 128). One can also incorporate more information in the SURF-descriptors and let them be of length 128.

2.2 Blur detection

This section covers some of the methods that is used for blur detection. In particular a non-reference blur metric (CPBD, see 2.2.3) is described in detail. One major part in the CPBD algorithm is to determine the width on an edge. Thus two edge detection methods are also described in this section.

2.2.1 Sobel Edge Detector

An edge can be described as a rapid increase or decrease in pixel values in a small neighbourhood [6]. This implies that the gradient for pixels within that neighbourhood will be significantly higher compared to the gradient for pixels that are outside of this neighbourhood. The Sobel operator is a discrete differentiation operator. It has two kernels,

\[
S_x = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
\] (7)

and

\[
S_y = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{bmatrix}.
\] (8)

\(S_x\) is used to differentiate in the horizontal direction while the \(S_y\) kernel is for the vertical direction. The differentiation is done by convolving with the image,

\[
G_x = S_x \ast I
\] (9)
Thus the gradient magnitude can be computed as

\[ G = \sqrt{G_x^2 + G_y^2} \]  \hspace{1cm} (11)

and the gradient's direction as

\[ \tan \Theta = \frac{G_y}{G_x}. \]  \hspace{1cm} (12)

The gradient direction will be perpendicular to the edge and point from lower pixel values (dark area) to pixels with higher values (brighter areas). Since the gradient magnitude will be higher at edges compared to other areas in the image a threshold is used to determine if the pixel is an edge pixel or not. Since edge detectors are sensitive to high frequency noise one can also apply a Gaussian blur before using the Sobel operator. The Gaussian blur will reduce the high frequency noise.

### 2.2.2 Canny Edge Detection

In 1989 John Canny [7] proposed an edge detection method that is still widely used. Canny wanted the detector to be “optimal” by fulfilling three requirements:

- **Good detection**: There should be a low probability of failing to mark a real edge and also a low probability to falsely marking non-edges.

- **Good localization**: The marked points should be as close as possible to the center of the real edge.

- **Minimal response**: Only one marked edge per real edge.
The algorithm has the following methods:

- **Noise reduction**: Filter out the noise by using a Gaussian filter.

- **Gradient**: The intensity gradient of the image is computed. This can for example be computed by using the Sobel operator as described in section 2.2.1. The orientation is rounded to one of the four angles 0, 45, 90 or 135. These angles represent an edge in either vertical or horizontal direction or with the direction in one of the two diagonals.

- **Non-maximum suppression**: Identify the local maximums of the gradient magnitudes. The local maximums are found by identifying those pixels with higher gradient magnitudes compared to its two adjacent pixels in the gradient orientation. So for example if the gradient orientation is 0 degrees, a pixel is considered an edge candidate if its gradient magnitude is greater compared to the gradient magnitude of the pixels that are to the left and right of the pixel. This step will make sure that the found edges are thin. Those pixels that are not local maximums will be discarded in the next step.

- **Hysteresis**: Two thresholds are used here, an upper and a lower threshold. Those edge candidates with a gradient magnitude higher than the upper threshold will be marked as edges. Whereas those with a gradient magnitude lower than the lower threshold will be rejected. If a pixel has a gradient magnitude that is between the two thresholds, it will be marked as an edge only if it is connected to a pixel that has a gradient magnitude higher than the upper threshold.

### 2.2.3 No-reference blur metric CPBD

In 2011 Narvekar and Karam[8] proposed a non-reference image blur metric. The metric is based on the cumulative probability of blur detection and the metric is thus called CPBD.

The CPBD metric utilizes this by using the concept of “just noticeable blur” (JNB)[9]. The JNB is a measure on how much blurriness around an edge that is needed (in a probabilistic sense) before it is perceived by the human vision system. The amount of blurriness around an image is proportional to the edge width. Whereas the edge width is computed as the sum of the number of pixels with increasing grayscale values in the gradient direction and the number of pixels with decreasing grayscale values in the opposite
The edge width will increase with increasing blurriness. One can compute the probability of detecting blur around an edge, \( e_i \), as

\[
P_{\text{BLUR}} = P_{\text{BLUR}}(e_i) = 1 - \exp\left(-\frac{w(e_i)}{w_{\text{JNB}}(e_i)}\right)^\beta
\]

where \( w(e_i) \) is the edge width around edge \( e_i \) and \( w_{\text{JNB}}(e_i) \) is the JNB width. The \( \beta \) is equation 13 is a coefficient that the authors found out to be between 3.4 and 3.8 with a media value of 3.6. It was found that the JNB width depends on the local contrast, \( C(e_i) \), (assuming the image has 256 grayscales) in the neighbour of edge \( e_i \):

\[
w_{\text{JNB}}(e_i) = \begin{cases} 
5, & \text{if } C(e_i) \leq 50 \\
3, & \text{if } C(e_i) > 50 
\end{cases}
\]

In equation 13 one can see that when \( w(e_i) = w_{\text{JNB}}(e_i) \) there is a probability of 63\% to perceive blurriness around that edge. This probability (63\%) is denoted as \( P_{\text{JNB}} \). When \( w(e_i) < w_{\text{JNB}}(e_i) \) the blur will not likely be perceived. The authors’ metric, CPBD, is based on how many percentage of the edge widths that are below the JNB width. In previous work\[10\] it has been shown that it is actually sufficient to only consider the horizontal edges. Adding vertical edges did not provide for any improvements on Gaussian-blurred images and JPEG2000-compressed images.

The algorithm has the following steps:

1. **Canny edge detection**: Perform Canny edge detection on the image

2. **Remove smooth blocks**: The image is divided into blocks with a pixel size of 64x64. Given the edge previous edge detection, all blocks which has less than 0.2\% number of edge pixels are considered smooth blocks. The smooth blocks are discarded by algorithm in the next steps of the algorithm.

3. **Contrast**: Compute the contrast (difference between biggest and smallest grayscale value in the block) for every block. The contrast is used to determine the \( w_{\text{JNB}} \) for the block.

4. **Edge width**: Use the Sobel edge detector to find the horizontal edge pixels in the blocks. Compute the edge width of these edge pixels.
5. **PBLUR**: Given the edge width and the block contrast compute the probability to perceive blur.

6. **Compute histogram**: Given all probabilities create a histogram with 101 bins (from 0 to 100% with a bin size of 1%). The histogram is then normalized so that the sum of the histogram is equal to one. This means that the bin values are equal to the percentage of edge pixels where there is a specific probability to perceive blur.

7. **Compute the CPBD**: The metric is computed as:

$$CPBD = P(P_{BLUR} \leq P_{JNB}) = \sum_{P_{BLUR}=0}^{P_{BLUR}=P_{JNB}} P(P_{BLUR})$$

The summation in equation 15 is equivalent of computing the sum of the 64 first bins in the histogram.

Since the CPBD metric is proportional to the percentage of edge pixels where blur is not likely to be detected ($P_{BLUR} \leq P_{JNB}$), a higher metric implies a sharper image.

### 2.3 Artifacts - Non-Reference Methods

One approach used to create non-reference methods in data classification is to use concepts from information theory. The classical approach by Shannon [11] is to measure the information content transferred over a channel via the entropy

$$H = -\sum_i p_i \log_2(p_i)$$

where $p_i$ is the probability of producing one of $i$ symbols, and where $H$ is expressed in number of bits. A related notion is the Rate Distortion (RD) function [12], which describes the minimum amount of information necessary to approximately reconstruct an input signal sent over a channel, not exceeding a certain amount of distortion. It is defined as

$$RD(x, y, d) = \min_{d \leq D} J(x, y)$$

where $x$ is the input, $y$ is the output, $J(x, y)$ is the mutual information shared by $x$ and $y$, and $d$ is the distortion not exceeding the maximum
The mutual information is further defined as

\[ J(x, y) = H(y) - H(y | x) \]  

(18)

where \( H(y) \) is the entropy of the output signal and \( H(y | x) \) is the conditional entropy of the output signal given the input signal.

The Rate Distortion function can be thought of as an analytical expression representing the maximum compression ratio that can be achievable using a given lossy compression.

When assessing the information content of an object instead of the information sent over a channel, the concept of Kolmogorov Complexity can be used [13]. Using a more algorithmic approach it is given as the shortest program that can be written on a Universal Turing Machine ("theoretical computer") producing the data as its output. Mathematically this amounts to

\[ K(x) = \min_{q \in Q_x} |q| \]  

(19)

where \( K(x) \) is the string of length \(|q|\), which is the minimum length of all programs \( Q_x \) generating \( x \) [14]. Thus data with a high auto-correlation has low complexity and random data has high complexity. Note that this definition of Kolmogorov Complexity is a non-computable function, i.e. it is not possible to implement it on an actual computer in this form.

A related function to the Kolmogorov Complexity is the Normalized Information Distance (NID), which considering two data sources \( x \) and \( y \), and is defined as the length of the shortest program computing \( x \) knowing \( y \), and the shortest program computing \( y \) knowing \( x \). Formally we are interested in the quantities \( K(x|y) \) and \( K(y|x) \). NID is defined as the normalized metric [15]

\[ NID(x, y) = \frac{K(x, y) - \min\{K(x), K(y)\}}{\max\{K(x), K(y)\}} \]  

(20)

where \( K(x, y) \) is the Kolmogorov Complexity of the concatenation of \( x \) and \( y \), i.e. vector \( y \) appended to vector \( x \).

As stated earlier, the Kolmogorov Complexity is a non-computable function. This means that also the NID is a non-computable function. To be able to
use this metric in practice one can use a standard data compressor instead of the Kolmogorov Complexity, since it is somehow measuring how much the information content in a data source can be reduced. This leads to the metric Normalized Compression Distance (NCD) [16], which is defined as

$$ NCĐ(\mathbf{x}, \mathbf{y}) = \frac{C(\mathbf{x}, \mathbf{y}) - \min\{C(\mathbf{x}), C(\mathbf{y})\}}{\max\{C(\mathbf{x}), C(\mathbf{y})\}} $$

(21)

where $C$ is the used data compressor. This compressor is assumed to be a normal compressor, which means that the compressed size of the concatenation of the data $\mathbf{x}$ with itself should be the same as the compressed size of just $\mathbf{x}$, since no new information is added. Thus only lossless compressors will give expected results for this metric.

2.3.1 GMM

GMM is a clustering technique that estimates the parameters of a mixture of Gaussian distributions. Parameters to the Gaussian distributions are estimated so that the observations have some probability of belonging to one of the distributions. Thus a clustering with soft boundaries is obtained[17].

The mixture of Gaussians is composed of $k$ density functions

$$ \phi_k(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |S_k|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \mu_k)^T S_k^{-1} (\mathbf{x} - \mu_k) \right\}. $$

(22)

where $\mathbf{x}$ is the random variable, $d$ is the dimension, $\mu_k$ is the mean and $S_k$ is the covariance matrix. The mixture of $m$ of these distributions is described by

$$ \phi(\mathbf{x}; \mu_k, S_k, \pi_k) = \sum_{k=1}^{m} \pi_k \phi_k(\mathbf{x}) $$

(23)

subject to

$$ \pi_k \geq 0, \sum_{k=1}^{m} \pi_k = 1 $$

(24)

where $\pi_k$ are the mixture weights.
The algorithm tries to compute the maximum-likelihood estimates of the parameters $\mu_k, S_k, \phi_k$ such that

$$L(x, \theta) = \log(\phi(x, \theta)) = \sum_{i=1}^{N} \log \left( \sum_{k=1}^{m} \pi_k \phi_k(x) \right) \rightarrow \max_{\theta \in \Theta},$$

$$\Theta = \left\{ (\mu_k, S_k, \pi_k) : \mu_k \in \mathbb{R}^d, S_k = S_k^T > 0, S_k \in \mathbb{R}^{d \times d}, \pi_k \geq 0, \sum_{k=1}^{m} \pi_k = 1 \right\}.$$  

This is done iteratively in two steps. First the expectation step, in which the probability $p_{k,i}$ of sample $x_i$ belonging to mixture $k$ is computed as

$$p_{k,i} = \frac{\pi_k \phi(x; \mu_k, S_k)}{\sum_{j=1}^{m} \pi_j \phi(x; \mu_j, S_j)}.$$  

In the second step, the so called maximization step, the estimates of the mixture parameters are updated using the $p_{k,i}$ probabilities such that

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} p_{k,i}$$  

$$\mu_k = \frac{\sum_{i=1}^{N} p_{k,i} x_i}{\sum_{i=1}^{N} p_{k,i}}$$  

$$S_k = \frac{\sum_{i=1}^{N} p_{k,i} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} p_{k,i}}.$$

Repeating this iterative procedure will result in soft boundaries of the observations $x_i$, expressed in parameters of the Gaussian mixtures.

### 2.4 Artifacts - Reference Methods

The most basic and maybe also most commonly used of the reference methods is the Mean Squared Error (MSE). It is a pixel-wise operation defined as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (I_1(i) - I_2(i))^2.$$
for two images $x$ and $y$, where $N$ is the number of pixels in each image. One frequently used variant of this is the peak signal-to-noise ratio (PSNR). It is defined as

$$PSNR = 10 \log_{10} \left( \frac{L^2}{MSE} \right)$$

(32)

where $L$ is the image depth, i.e. the number of discrete levels used to store color data in the image (e.g. for an 8-bit image $L = 256$).

Both these methods have the advantage of being simple and having low complexity. However, both also have a major caveat in having low correlation with the Human Visual System (HVS) [18]. Thus, the output from MSE and PSNR agree badly with what we perceive as good image or video quality. For this reason, a number of methods have been developed which try to compensate for this with more or less sophisticated models of HVS[19]. Due to the complex nature of HVS however, these models tend to be impractical and/or to computationally heavy for efficient implementation.

2.4.1 SSIM

In 2004 Wang et al. [20] presented a new reference metric called the Structural Similarity Index (SSIM). It has the advantage of being both simple and having reasonable correlation with HVS.

The basic idea of SSIM is to separate the metric into three components: luminosity ($l$), contrast ($c$) and structure ($s$). These components are then defined using the mean and standard deviations

$$\mu_1 = \frac{1}{N} \sum_{i=1}^{N} I_1(i),$$

(33)

$$\sigma_1 = \left( \frac{1}{N-1} \sum_{i=1}^{N} (I_1(i) - \mu_1) \right)^{\frac{1}{2}},$$

(34)

$$\sigma_{12} = \frac{1}{N-1} \sum_{i=1}^{N} (I_1(i) - \mu_1)(I_2(i) - \mu_2),$$

(35)
for two images $I_1$ and $I_2$ such that

$$l(I_1, I_2) = \frac{2\mu_1\mu_2 + C_1}{\mu_1^2 + \mu_2^2 + C_1}$$  \hfill (36)$$

$$c(I_1, I_2) = \frac{2\sigma_1\sigma_2 + C_2}{\sigma_1^2 + \sigma_2^2 + C_2}$$  \hfill (37)$$

$$s(I_1, I_2) = \frac{\sigma_{12} + C_3}{\sigma_1\sigma_2 + C_3}$$  \hfill (38)$$

where $C_i$ are safety constants to avoid division by zero. The mean and standard deviations are computed using a Gaussian window to avoid blocking effects. The three components are then combined as

$$SSIM(I_1, I_2) = l(I_1, I_2)^\alpha c(I_1, I_2)^\beta s(I_1, I_2)^\gamma$$  \hfill (39)$$

where the parameters $\alpha, \beta, \gamma$ are used to scale the relative importance of the components. It can be shown that SSIM fulfills the following criteria

$$SSIM(I_1, I_2) = SSIM(I_2, I_1) \quad \text{Symmetry} \hfill (40)$$

$$SSIM(I_1, I_2) \leq 1 \quad \text{Boundedness} \hfill (41)$$

$$SSIM(I_1, I_2) = 1 \iff I_1 = I_2 \quad \text{Unique maximum} \hfill (42)$$

which makes it a good candidate for a usable image metric.

SSIM is computed locally over the two images using a sliding $8 \times 8$ window. The mean of these local values, the Mean SSIM, is defined as

$$MSSIM(I_1, I_2) = \frac{1}{N} \sum_{i=1}^{N} SSIM(I_{1,i}, I_{2,i})$$  \hfill (43)$$

where $N$ is the number of local SSIM computations and $I_{1,i}, I_{2,i}$ are the respective local images. MSSIM can be used to produce an overall score of the image quality.

### 2.4.2 Fast SSIM

Since SSIM uses multiple convolutions with a Gaussian for each sliding window computation, it is a relatively complex operation. In 2011, Chen and Bovik [21] developed methods to reduce the computational complexity of SSIM while still retaining accuracy.
The luminance component, which only contains mean intensity values of the two images, can be efficiently computed using the concept of the integral image. The integral image $J$ is the sum of all pixel values above and to the left of the current pixel, including its own:

$$J(i, j) = \sum_{i' \leq i} \sum_{j' \leq j} I(i', j').$$  \hfill (44)

The integral image can be computed efficiently using the recursive formula

$$J(i, j) = I(i, j) + J(i - 1, j) + J(i, j - 1) - J(j - 1, i - 1).$$  \hfill (45)

However, the main advantage of the integral image is that, once it is computed, evaluating the sum of pixels in any arbitrary rectangle in the image can be done in $O(1)$. Defining a rectangle with corners $((i_a, j_a), (i_b, j_b), (i_c, j_c)$ and $(i_d, j_d)$, Figure 5, this sums up to

$$\sum_{i_a \leq i \leq i_c} \sum_{j_a \leq j \leq j_c} I(i, j) = J(i_a, j_a) + J(i_c, j_c) - J(i_b, j_b) - J(i_d, j_d)$$ \hfill (46)

As seen only four function evaluations are needed to compute the sum of the pixels in the rectangle.
For the contrast and structure component, it has been shown that the gradient can be used as a valid substitute for the standard deviation. Using the gradient can even improve the accuracy of SSIM by improving performance on blurred images [22].

The gradient can be computed efficiently by using the Roberts gradient templates

$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

(47)

and

$$R_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$  (48)

A simple approximation of the gradient magnitude can be obtained by

$$|\nabla I| = |R_1 \ast I| + |R_2 \ast I|.$$  (49)
where $\nabla$ is the gradient operation and $\star$ denotes the convolution operation.

The Structural Similarity Index is then calculated analogous with (39), only using the mean intensity computed from the integral image, and the approximated gradient magnitude instead of the standard deviation. Some critical parts of the Python code has also been ported to C further accelerate the algorithm. Together these measures speed up the algorithm up to a hundred times.

2.4.3 Denoising

2.4.3.1 General

For general purpose denoising of the input video, bilateral filtering has been used [23]. This filter has the advantage of preserving edges in the image while denoising. As a downside it is more computationally expensive than standard filter methods. The general idea in bilateral filtering is to use both geometric and photometric closeness in the averaging process. Mathematically this can be described as

$$h(i, j) = k^{-1}(i, j) \sum_{i'=1}^{N} \sum_{j'=1}^{M} I(i', j') c(i', j', i, j) s(I(i', j'), I(i, j))$$

where $I$ is the image, $c$ is the geometric closeness between $(i, j)$ and $(i', j')$, $s$ is the photometric closeness between the pixel at $(i, j)$ and the pixel at $(i', j')$ and $k$ is a normalizing term.

2.4.3.2 Morphological

Another denoising approach, especially common if binary images are considered, are methods from mathematical morphology. The two most basic morphological operations erode and dilate, can be described easiest as changing your pen to a thinner one in the former case, and to a thicker one in the latter. This concept is depicted in Figure 6.
To define erosion and dilation, the common set operators of union and intersection ($\cap$, $\cup$), as well as translation are needed. With a vector $z$, translation is defined as

$$A + z = \{a + z \mid a \in A\}. \quad (51)$$

Erosion and dilation are closely related to Minkowski addition and subtraction, which are defined as

$$A \oplus B = \bigcup_{b \in B} (A + b) \quad (52)$$

and

$$A \ominus B = \bigcap_{b \in B} (A + b). \quad (53)$$

With these operators, erosion and dilation can be defined as

$$E(A, B) = A \ominus (-B) = \bigcap_{b \in B} (A - b) \quad (54)$$

and

$$D(A, B) = A \oplus B = \bigcup_{b \in B} (A + b) \quad (55)$$

where $A$ is the binary image and $B$ is the structural element (e.g. see Figure 7) [24].
Figure 7: An example of a structural element used in mathematical morphology.

If these basic morphological operations are combined, e.g. using erosion followed by dilation, denoising effects can be achieved.

2.4.4 Segmentation

Several techniques from image analysis and computer vision are used in combination to get a robust and stable segmentation method. The algorithms below are the current implementations in the open source library OpenCV.

2.4.4.1 Find Object Contour

To find the contour of a (binary) object, the standard method is to use an 8-pixel neighborhood and the so called Moore neighborhood algorithm [25]. A variant of this is implemented in the OpenCV library [26]. The algorithm is a border tracing algorithm described in the following pseudocode:
initialize:
    \( p \): to keep track of the current pixel
    \( s \): to keep track of the pixel from which \( p \)
    was entered from

do:
    find a starting pixel by scanning the image left-to-right, 
    bottom-to-top starting at the left bottom corner until 
    a 1-pixel is found
repeat:
    search clockwise around the current pixel \( p \) 
    starting from \( s \) until a 1-pixel is found
until termination criteria 
end

The termination criteria is usually defined as when the first pixel is revisited 
from the same direction as it was the first time. If several contours want to be 
found, and not just the outermost, the algorithm can be applied iteratively 
whilst removing the previously found contours as you go.

2.4.4.2 Polygon Simplification

To simplify complex polygons and/or extract corners from an object contour, 
the Douglas-Peucker line simplification algorithm can be used [27]. It works 
by defining anchor points, from which to perform the algorithm, and floaters, 
which are end points in the line/polygon to test against. The algorithm iteratively checks the largest perpendicular distance from the segment joined 
by the anchor and the current floater, and all of the other points on the line. 
If this distance is above a certain threshold, the current floater is changed 
to the point of the largest perpendicular distance. If no point is above the 
threshold, the current floater is converted to the next anchor point, and the 
process is repeated until the last anchor has been set at the end of the line. 
This set of anchor points now represent the simplified line. For a graphical 
explanation of this algorithm, see Figure 8.
(a) The algorithm first finds the largest perpendicular distance between the segment $a_0 - f_0$ and the line, which is the point $f_1$. This is done again with the segment $a_0 - f_1$, now the largest distance is at $f_2$. This is repeated until the segment $a_0 - f_3$ is tested. This time no perpendicular distance is found that is larger than the given threshold, hence $f_3$ becomes the new anchor point $a_1$.

(b) For the second iteration the same procedure is repeated using $a_1$ as anchor. This time the largest perpendicular distance between the line and segment $a_1 - f_2$ falls below the threshold and $f_2$ becomes the new anchor point $a_2$.

(c) Final result of the Douglas-Peucker algorithm after 5 iterations.
2.4.4.3 Perspective Transform

To compensate for aligning problems in the setup, a perspective transformation can be used for planar rectification. For an example of this see Figure 9.

Figure 9: An example of planar rectification using a perspective transform. The four corner points in the left image are mapped so that they form a perfect square in the right image. This can be used to compensate for aligning problems in camera setups.

A general description of the perspective transformation as a mapping between two points \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) with \(z_1 \neq 0\) is

\[
\begin{bmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{bmatrix} =
\begin{bmatrix}
  H_{11} & H_{12} & H_{13} \\
  H_{21} & H_{22} & H_{23} \\
  H_{31} & H_{32} & H_{33}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{bmatrix}
\]  

(56)

in homogeneous coordinates where \(H\) is the transform matrix. The above can be rewritten as

\[
x'_2 = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1}
\]  

(57)

\[
y'_2 = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1}
\]  

(58)

using inhomogeneous coordinates \(x'_2 = x_2/z_2, y'_2 = y_2/z_2\) and setting \(z_1 = 1\).
Since these equations are linear in the elements of $H$, they can be written as

\[ a_x^T h = 0 \]  \hspace{1cm} (63)
\[ a_y^T h = 0 \]  \hspace{1cm} (64)

where
\[ h = [H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33}]^T \]  \hspace{1cm} (65)
\[ a_x = [-x_1, -y_1, -1, 0, 0, 0, x_2', x_1', x_2'y_1, x_2'y_1']^T \]  \hspace{1cm} (66)
\[ a_y = [0, 0, 0, -x_1, -y_1, -1, y_2', x_1', y_2'y_1, y_2'y_1']^T \]  \hspace{1cm} (67)

The matrix $H$ has eight degrees of freedom, $h$ can therefore be determined using (at least) four point pairs. The final linear homogeneous equations are

\[ Ah = 0 \]  \hspace{1cm} (68)

where
\[ A = [a_{x_1} a_{y_1} \ldots a_{x_N} a_{y_N}]^T. \]  \hspace{1cm} (69)

This system can be solved using e.g. Singular Value Decomposition.
3 Methodology

3.1 Zoom function

This chapter covers how the zoom function of the mobile camera was validated.

3.1.1 Outline

In order to validate the zoom function an object with high spatial information was created. This object was then recorded in a reference video where all zoom levels was visited in a known order. To extract information from the image frames SURF features were used. These features were then clustered and reference histograms were computed for every zoom level. These histograms made up for the reference content in the algorithm.

During the testing phase a test video was recorded where the zoom level was set at random values. Given the SURF descriptors a histogram was created for every image frame. This histogram was then compared to its closest histogram from the reference video, and subsequently being classified to that zoom level.

3.1.2 Test setup

An object with high spatial information was created. High spatial information is needed to more easily discern between the different zoom levels. The creation of the object was done in an ad hoc manner. We believe that the object can be designed differently and still maintaining the same results. In Figure 10 the object is viewed without any zoom. The camera platform was located around 60 cm from the object and around 15 cm from the ground. This setup can be varied a lot and still maintaining the same results.
Figure 10: Test object that was used to validate the zoom function of the camera. The current image was taken in the actual test setup, without using zoom, with the camera. The green dots are the location of the SURF descriptors that was used in the algorithm.

3.1.3 Reference phase

An Android program that made the camera iterate through all different zoom levels while recording was created. This is done by going from zoom level zero (i.e. no zoom) and subsequently going to zoom level six in six steps, and then going back to zoom level zero. The camera records for one second at every zoom level with a frame rate of 30 and an image size of $1080 \times 1920$ (1080p). The video was recorded in color, but in further analysis the image frames are converted to black and white.
For the analysis we decided to use SURF features (see section 2.1.3). The motivation was that we wanted a scale invariant feature since the objects in the image appear at different sizes at different zoom levels. We chose SURF over SIFT (2.1.2) since the latter is computationally slower.

The algorithm was fed with several frames with the same image content (there are around 30 frames at every zoom level), thus the SURF descriptors had to stay the same throughout every zoom level, i.e. they must be resistant to noise. Thus the threshold of the SURF algorithm was adapted so that this was achieved. These chosen parameters were heavily dependent on the object that was being used for the reference video. Here the parameters given in table 1 was used. In Figure 12 the number of SURF features for every frame in the reference video is plotted. One can clearly see that there are more SURF features for lower zoom levels. The transitions between the zoom levels are also very visible. However between zoom level zero and one there is almost no visible difference. The reason behind this is that the difference in the image between zoom level zero and one does not have high spatial information. This problem can easily be avoided by using a more advanced test setup.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold for Hessian key point detector</td>
<td>1300</td>
</tr>
<tr>
<td>Number of pyramid octaves</td>
<td>7</td>
</tr>
<tr>
<td>Number of octave layers within each octave</td>
<td>4</td>
</tr>
<tr>
<td>Length of descriptors</td>
<td>128</td>
</tr>
<tr>
<td>Compute orientation of features</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 1: The SURF parameters that were used when validating the zoom function of the camera.

A simple algorithm was developed in order to automatically detect the transitions. The signal shown in Figure 12 (number of SURF features) was used as input to the algorithm, which is presented below:

- Lowpass (averaged) the signal to reduce the noise in the signal. Filter length of 11, i.e. the kernel was a vector with 11 ones.
- Highpass the above signal do detect the rapid transitions. Filter length of 7, i.e. the kernel was a vector of length 7 with ones and minus ones alternating (1, -1, 1, \ldots).
- Since the above signal was very noisy, the signal was again lowpass
(averaged) filtered. Filter length of 5.

- The local maxima within a window size of 21 are then marked as zoom transitions. The size of the window depends on how often the zoom switches occur. Here they occur in intervals of approximately one second, i.e. around 30 frames.

One would think that the (since convolution is commutative) the two lowpass filters in the above steps could be concatenated to one single lowpass filter. However we could not find this possible. However this algorithm will not always detect the switch between zoom level zero and one. One thus need to manually set that frame as a transition. In Figure 13 the zoom transitions are shown as vertical lines.

In the following methods only the latter half of the video was considered, i.e. when the zoom level went from six to zero. This half was used to build the model, while the first half was used as validation.

K-means clustering (see section 2.1.1) was then performed on all features from all frames. The number of clusters was chosen to 500. The choice was made with the knowledge that the number of key points varies between 150 and 700, which gives an average of 425. However it was adjusted upwards since we wanted to account for the key points that are only visible in the lower zoom levels.

The clustering was used to compute normalized histograms (order 2) for every zoom level. Given the zoom transitions all features for all frames within a zoom level were collected. Each of these features were then classified to one of the centroids that was found in the k-means clustering. The classification is done using the nearest neighbour algorithm. The classifications was subsequently used to create reference histograms for every zoom level:

$$H_i, \quad 0 \leq i \leq 6$$

(70)

The histograms were normalized with respect to 2-norm. Now that the model was built, i.e. the seven histograms and centroids, we proceeded with validating the model. The following steps was performed on every frame in the first half of the video:

- Extract the SURF features
• Build a normalized histogram \((H)\) using the centroids from the modeling phase,

• Classify the frame to the zoom level which has the closest histogram. Among the many distance metrics for histograms\([28]\) the 2-norm was chosen, which provided for good results.

\[
\arg\min_i ||H - H_i||_2
\]  

(71)

• Given the zoom transitions we can now compare the true value to the classified level.

In the test runs the validation showed perfect results. The only anomalies were in the beginning of the video. The cause of this was that when the camera started recording, it took some tens of a second before the exposure and other parameters were set and stable. The validation also differed in the neighbour of a zoom transition. The cause was that a zoom transition could take longer than one frame. This implied an ambiguity for determining where the transition actually was.

3.1.4 Testing phase

An Android program was created that let the camera record for a certain amount of time while it randomly switched zoom level every second. The test video was then analyzed in the same manner as in the validation process.

In Figure 14 the result from one of these test videos is found. The black line is the true zoom level and the red line is the classified zoom level. It is very apparent in the plot that the lines follow each other. Since the camera was working correctly, we know that the algorithm worked. There are some differences, but these only occur at the zoom transitions which was mentioned in the previous section.

The program that was created warns the user if the classified zoom level differs from the true value. It also distinguish if the difference happened in the neighbour of a zoom transition or not. If true the difference might not be a real error, but just an error in the zoom transition.
Figure 11: Screen caps from the reference video. The images are taken at zoom level one to six, beginning at zoom level one at the upper left corner. The green dots are the location of the SURF descriptors. A frame from zoom level zero, i.e. no zoom, is found in Figure 10.
Figure 12: Number of detected SURF features at every frame for the reference video that is described in section 3.1.3.

Figure 13: Number of detected SURF features at every frame for the reference video that is described in section 3.1.3). The zoom transitions are also shown as vertical green lines.
Figure 14: The plot shows the result from a zoom test. The video recorded while randomly switching the zoom level. The red solid line represents what zoom level that was set in the camera parameters. The black dashed line represents what zoom level that was actually recorded. The black dashed line was acquired by using the proposed method to validate the zoom function. Since the black and red line is following each other at every frame, this indicates that the zoom function is working correctly in the camera.

3.2 Colors

The mobile platform that was used can apply the following seven color effects to its recorded videos and images:

- Aqua
- Mono
- Negative
- Posterize
- Sepia
- Solarize
- Whiteboard
Examples of these color effects can be found in Figure 15. To validate that the camera applies these color effects correctly a reference based approach was developed. Applying a color effect is essentially a histogram matching. Thus one wants to analyze the histograms of the images. Here color images are used, i.e. every image has three channels (red, green and blue). The word color histogram will refer to the histogram of all three channels:

\[ H_i = \{ H_{i,\text{red}}, H_{i,\text{green}}, H_{i,\text{blue}} \} \]  

(72)

Figure 15: Different color effects applied in the mobile camera.
Where $H_i$ is the color histogram for frame $i$ and $H_{i,color}$ is the histogram for a given channel for frame $i$.

### 3.2.1 Reference phase

An Android application for the reference phase was developed. The application made the camera loop through all color effects while recording a video, from here called the reference video. The object that was being recorded was the same object as described in section 3.1 (Figure 10). The application also saved a text file with information about the order of the applied color effects.

The video was then analyzed in the computer. The next step was to determine which frame in the reference video that belonged to which color effect. Since the text file contained the applied color effects, we needed to find the transitions in the video in order to classify the frames. The switches were found by iterating through the video and computing the difference between the current frame’s color histogram with the color histogram of the previous frame. When the frames had different color effects applied the computed difference was significantly higher compared to when both frames had the same color effect applied, see Figure 16. Furthermore a threshold was used on the differences to identify where the switches occur. This threshold was chosen to 0.10, a value that gave the desired results.
When studying the computed difference throughout the reference video one could also conclude that switching the color effects may take longer than one frame. This could lead to problems when classifying the frames, thus a “switch width” was introduced. All frames within this switch width were discarded by the algorithm. The switch width was chosen to 5.

The color histograms have 85 equally sized bins ranging from 0 to 255 and they were normalized so that their integral were equal to one. Since color images with three channels were used, the difference was the sum of the histogram differences for all three channels:

\[ \textbf{H}_i - \textbf{H}_j = \left| \left| H_{i,\text{red}} - H_{j,\text{red}} \right| \right|^2 + \left| \left| H_{i,\text{green}} - H_{j,\text{green}} \right| \right|^2 + \left| \left| H_{i,\text{blue}} - H_{j,\text{blue}} \right| \right|^2 \]

The difference between two histograms are computed in 2-norm. This norm provided accurate results.

The previous steps had classified all frames to a given color effect. Given this knowledge one could create an averaged color histogram for every color effect. This was done by summing the color histograms from all frames
with a given color effect. The result was then renormalized. The resulting eight (seven color effects + no effect applied) color histograms will be called “reference histograms” and denoted as $H^CE$ where $CE$ is a color effect.

### 3.2.2 Testing phase

After the reference histograms had been created one could use these to validate the color effects on another mobile. For the testing phase another Android application was developed. The application made the mobile camera switch color effects randomly while recording a video. It was also designed so that no two consecutive color effects were the same. Otherwise the algorithm failed when identifying the switches in the video. The video also had to use same object as in the reference phase, otherwise the classification would fail. Just as in the reference phase this Android application also saved a text file with the applied color effects. In conclusion the text file determined what color effects that was supposed to be applied in the video, while the video shows what color effect that was actually applied.

The first step was to determine when the mobile switched color effects. This was done by analyzing the video in the same manner as in the reference phase. Given this knowledge one could determine how many color effects that had been applied. If this number differed from the number of applied color effects in the text file, the algorithm would mark the test run as failed.

The next step was to iterate through the video and compute the color histograms. To classify what color effect that was applied to the specific frame the algorithm searched for the closest reference histogram, i.e.:

$$\arg \min_{CE} (H^{CE} - H_i)$$

The classified color effect was compared to the color effect in the text field. If a frame was classified to the wrong color effect the algorithm marked the test run as failed. The algorithm also returned which frame(s) that was misclassified so that the developer of the mobile platform could easily identify the error. If all frames were correctly classified the test run was considered a success.
3.3 Blur

To detect blur the CPBD-algorithm (see section 2.2.3) was implemented and images and videos from the mobile were analyzed. Very early it was discovered that the CPBD-algorithm were not suitable to use in a no-reference based approach for blur detection in the scope of this project. The reason was that images of different objects could render very different results from the CPBD-algorithm, even though the images were blur free. This implied that a no-reference based approach would not work.

A reference based approach to detect blur was also investigated. The results from this approach is presented in section 4.3.

3.4 Artifacts - Non-Reference Methods

In satellite imaging there is a strong need for non-reference artifact detection methods [13]. Satellites gather huge amounts of data every day, and it is not uncommon for it to be ridden with noise and artifacts from electronics and sensors. Using the assumption that artifacts change the local complexity and/or entropy of the image, information theoretic methods can be used in combination with clustering to try to identify anomalies.

Using these methods as a basis, two different artifact detection schemes have been implemented. One based on Rate Distortion (RD), and one based on Normalized Compression Distance (NCD).

3.4.1 Rate Distortion

This method relies on implementing an artifact detection scheme using Rate Distortion, as described in 2.3. As Rate Distortion is a measure using lossy compression, it seems reasonable to use lossy image compression for this purpose [29]. The general idea is that areas in the image with high entropy will achieve a higher compression ratio than areas with low entropy. The image is divided into 64 × 64 blocks, and then each block is subjected to a number of different JPEG compressors with different compression ratios (and thus quality). These compressed images are then decompressed, and the difference between the original image and all the different compressed ones are put into a feature vector for each block. These feature vectors are then clustered using a Gaussian Mixture Model (GMM), see Section 2.3.1,
and a probability map is created. The implementation followed the scheme in Figure 17. This is the same basic approach as proposed in [30].

![Figure 17: Scheme for artifact detection using Rate Distortion.](image)

### 3.4.2 Normalized Compression Distance

The following scheme was implemented for the NCD approach. The image was once again divided into $64 \times 64$ blocks, and then for each block NCD was applied, as described in Section 2.3, with respect to all other blocks in the image. Thus feature vectors for each block is produced, with length equal to the number of blocks in the image. A GMM model is then applied to these feature vectors creating a probability map. The general scheme is depicted in Figure 18.

![Figure 18: Scheme for artifact detection using Normalized Compression Distance.](image)

### 3.5 Artifacts - Reference Methods

SSIM has lots of advantages in being simple, fast (at least with the modifications described in Section 2.4.2 and that it is in relatively good accordance with the Human Visual System, see Section 2.4.1 for details.

There are however several native problems with using a full reference method like SSIM:

- The images have to be fully aligned.
- Videos need to be synchronized frame by frame.
The method relies too heavily on perfect reference images and video. There is a big problem acquiring such data using phones in the development stage.

What would be desirable is to try to use SSIM in a non-reference or reduced-reference manner. This should be possible to achieve using the fact that there are no real restrictions on what test setup to use.

If a test setup is constructed with a suitable test image, SSIM can be used to measure the self-similarity of the image, by comparing different parts of the image with each other. With the reasonable assumption that areas that are artifact free should be most similar to each other, it would be possible to find artifacts in the image.

### 3.5.1 Test Image

The desired test image should have the following characteristics.

- Exhibit patterns that are sensitive to artifacts.
- Include information in all color channels.
- It should be arranged in such a way that SSIM metrics can be computed between areas of the image at different scales.

One way to design a pattern that fulfills these characteristics is to use a square, translational symmetric pattern with multiple colors, and a more complex structure at the smallest scale. The somewhat ad-hoc test image constructed with this design is depicted in Figure 19. This image is generated via a script so that the resolution, number of levels, colors etc. easily can be varied.
3.5.2 Algorithm

The algorithm will divide the test image into four equally sized blocks, and then compare these with each other using SSIM. Since there are four different blocks, the resulting SSIM computations will be six, see Figure 20.
Figure 20: The test image is divided into four equally sized blocks, and then SSIM is used to compare these to each other.

This scheme is then applied in a recursive manner, going down one scale in the image and following a clockwise pattern. This is repeated until the smallest scale is reached, see Figure 21 and Figure 22. Of the six MSSIM values that are computed for each recursion, each is compared to a threshold value, signifying what is considered a high enough MSSIM value to be produced from two artifact-free blocks.
Figure 21: The algorithm is applied in a recursive, clockwise manner until the smallest scale of the test image is reached.

One of these artifact-free blocks is then used to compute an SSIM probability map with respect to the other three blocks. If none of the MSSIM values are high enough to qualify as artifact free, a stored artifact free block is used to compute the map. All artifact free blocks are saved during the recursion so that the spatially closest block always can be used in SSIM.

The purpose of these measures is to keep the SSIM computations \textit{local}, since SSIM is sensitive to translations and other geometric transformations.

To ensure that artifact free blocks always are available, the recursion makes two clockwise iterations at each level. All the local SSIM computations are then stored in a global probability map equal in size as the test image.

Figure 22: The test image at its smallest scale.
3.5.3 Setup Camera Test

Since the main focus in this thesis is video quality, a moving target is preferred as a test object. The easiest way to create such a target is to use a rotating and translating version of our test image in Figure 19.

For this purpose a Lego Mindstorms robot was purchased, which with its three basic DC motors and numerous miscellaneous parts are capable of performing the required operations necessary for a basic evaluation.

A prototype was constructed which was capable of translating in one dimension, and which had a rotating axis which the test image could be fastened to, see Figure 23.

Since the only part of interest in the filmed video is the test image, a framed black fabric was used to put behind the robot.

![Figure 23: The test setup for the mobile camera.](image)

3.5.4 Setup Screen Test

To test the mobile phone screen for artifacts, a webcam was used to film the screen at close range. It was then set to display a computer generated
version of the rotating and translating test image. This setup can be seen in Figure 24.

Figure 24: The test setup for the mobile LCD screen.

### 3.5.5 Pre- and post-processing

To be able to use the above setup the rotating and translating test image need to be segmented out. The following steps has been taken to perform the segmentation in a robust manner.

- Threshold to a binary image.
- Remove dust and specks with morphological denoising, Section 2.4.3.2.
- Find the image contour, Section 2.4.4.1.
- Simplify given contour polygon to obtain the four corners of the square, Section 2.4.4.2.
- Compensate for aligning problems as well as rotate the image to its nominal position (Figure 19) using a perspective transform, Section 2.4.4.3.

As mentioned above, SSIM is quite sensitive to translations and other transformations. This makes the final probability map rather noisy, with lots of
lines where the blocks do not exactly overlap. To remedy this the image is again thresholded to a binary image, and then subjected to morphological denoising techniques.

To be able to analyze video without having to actually look at the generated probability map, we introduce an error metric as

$$e(P) = \sqrt{\frac{1}{M^2} \sum_i P(i)}$$

(77)

where $P$ is the probability map and $M$ is the length of the side of the used test image. This gives a value between 0 and 1 that corresponds to how large area of the test image that is affected by artifacts.

If the errors are so severe that the segmentation of the test square fails, that frame is skipped and its corresponding error is set to its maximum of one.
4 Results and Discussion

In this thesis a set of tools has been developed with the purpose of being a complete system for end user testing of video and images on mobile platforms in the development stage.

Both the case of viewing recorded image/video from the mobile camera and the case of viewing image/video on the mobile screen have been considered.

In its general form this is a hard problem due to the diverse nature of the errors in videos and images that can arise. It is not likely that one can find one solution to all these problems and still retain good performance. Therefore four distinct categories have been identified, representing the most common and severe problems in this context of mobile video and imaging.

These four identified categories are

- Zoom Function
- Colors
- Blur/Auto Focus
- Artifacts

For these four categories four corresponding solutions have been developed, to be able to tailor the algorithms to its specific task. These algorithms are then meant to be used in succession, to serve as a complete solution for finding possible errors and problems in videos and images.

Especially the last category, artifacts, relies on the image to be void of the errors of the other categories, e.g. that the image is not unfocused and tinted green. Would such errors be present when this step is reached, the algorithm would probably fail. Thus the order of the tested categories is somewhat important, especially that the testing for artifacts is done last, and only if the other tests have failed to detect anything.

A general difficulty when analyzing the mobile screen is that it is hard to get an objective view of how the media content will actually look to the end user. For example it has been tried to extract the video data from the frame buffer just before it reaches the screen. However there have always been problems with certain type of errors which has not been detectable using this approach.
To be sure that the same image or video that the end user sees is analyzed, an external camera setup has been successfully used to film the screen.

4.1 Zoom

The algorithm has been tested extensively. Initial results show that the algorithm works very well and provides a hit rate of > 99%. Test videos from different hours at the day, meaning different lightning conditions, has been analyzed and shows good results.

Since zoom level five and six are very similar to each other, the method is very sensitive to translations of the camera and object. A presumption for the analysis is that the video looks reasonable correct, e.g. no artifacts and that the image is correctly rotated.

It is very important that the camera’s auto focus is disabled and fixed to a constant value. We used focus at infinity. If auto focus is enabled the image can be more of less blurry, which affects the SURF extraction.

As described above the algorithm will just classify every frame to its closest zoom value. Thus the algorithm will not detect if the camera is stuck between two zoom levels. This was purposely done in order to avoid another level of thresholding and ambiguity. However one can warn the user if the histogram distance to its closest zoom level is above a certain threshold. But in order to definitely set a threshold, that would require extensive tuning of the algorithm.

All analysis is done offline since it is computationally heavy. Currently the analysis of a 20 second video takes approximately 5 minutes.

4.2 Colors

The proposed method to validate the color effects of the mobile camera was working perfectly. Several test videos was used and the algorithm successfully classified all frames.

One limitation of the algorithm is that it only classifies every frame to its closest color effect. Suppose that the mobile fails to apply a certain color effect and the result is not a valid color effect. Thus the proposed method will classify this frame to the closest color effect. Worst case scenario it will classify it to the color effect that the frame was supposed to have, and
the test will be marked as a success. This drawback can be avoided by applying a threshold on the difference between the frame's color histogram and the reference histogram for the classified color effect. If the difference is above a certain value, the test is considered a fail. During the scope of this project, the value of such a threshold has not been investigated. This is because we wanted to keep the number of parameters and ambiguity down to a minimum. Setting such a threshold requires extensive tuning of the algorithm.

By further analyzing how the mobile applies the color effects, i.e., find the functions for the histogram matching, one should be able to reduce the reference phase in the proposed method. Thus one would only need to take one image or record a short video with no color effect applied. Given the color histogram when no color effect is added, one can then compute the reference histograms.

4.3 Blur

In Table 2, a comparison of the CPBD values shows that for different images with the same object. The CPBD value is clearly lower for the image where blur has been added with an image software (Figure 27), compared to the original image (Figure 25). However, when the object was moving, i.e., motion blur, the CPBD did not decrease compared to the original value (Figure 25). When visually comparing Figure 25 and 26, one can see a tremendous difference in sharpness. So ideally, the algorithm's result would reflect this difference in sharpness. As described in Section 2.2.3, the metric is based on the edges with a horizontal direction. This might imply that if the motion blur is directed in the perpendicular direction, the metric will fail to find the blur. However, rotating Figure 25 did not provide for a lower metric.

In this project, the main reason for detecting blur is to validate the autofocus of the camera. If an object is out of focus, the object will obviously be more blurry compared to that if it is in focus. However, we could not manually set the mobile camera out of focus (i.e., introduce a focus error) and analyze such images. This might be since the mobile camera has such a low aperture which leads to a very high depth of field (measure of how deep the focused area is). However, by using a conventional camera, the algorithm was tested on images which were out of focus. The initial results showed a satisfied behavior, i.e., the CPBD metric decreased when the image was out.
of focus. An image where the object was in focus (Figure 28) gave a CPBD value of 0.4226, whereas an image where the same object was out of focus (Figure 29) gave a CPBD value of 0.2971.

The problem of detecting whether an image has too much blur or not is also a very delicate problem. Firstly one wants the algorithm’s decision to reflect how the human perceives the image. This is a very hard problem regardless of what artifacts one are trying to detect. Secondly one needs to decide the specifications given by the mobile developer, i.e. how much blur should be allowed in this specific model. This would require extensive tuning of the algorithm and its parameters. Since this method was investigated in the end of this project, there was not sufficient time to fully explore the capabilities of the CPBD algorithm.

In conclusion one can say that the current implementation of the CPBD algorithm is not sufficient for use in the scope of this project. Clearly the algorithm should, if working, classify Figure 26 as too blurry. Further adjustments of the algorithm have to be made. However since the algorithm responded well to blur that was added during the post processing, there is definitely a possibility that one could adjust it so that also Figure 26 and similar will be classified as too blurry.

<table>
<thead>
<tr>
<th>Image</th>
<th>CPBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image with no blur (fig 25)</td>
<td>0.5604</td>
</tr>
<tr>
<td>Image that was taken when the object was moving, i.e. motion blur</td>
<td>0.5583</td>
</tr>
<tr>
<td>(Figure 26)</td>
<td></td>
</tr>
<tr>
<td>Image with blur added with an image software (Figure 27)</td>
<td>0.0366</td>
</tr>
</tbody>
</table>

Table 2: Comparison of images used for blur detection
Figure 25: Image with no blur, taken with mobile camera, CPBD: 0.5604

Figure 26: Image that was taken when the object was moving, i.e. motion blur, taken with mobile camera, CPBD: 0.5583
Figure 27: Image with blur added in an image software, taken with mobile camera, CPBD: 0.0366

Figure 28: Image with no blur, taken with conventional camera, CPBD: 0.4226
4.4 Artifacts

The detection of artifacts is probably the hardest of the four categories to address, mainly due to the very loose and general definition of "artifacts". Artifacts in this thesis is defined as any type of structural anomaly or error in the image/video that is easily visible by a potential end user. This applies to both the mobile camera and the mobile screen. There is a general problem to match these kinds of algorithms to what we actually perceive as artifacts, which often have a low correlation with the output of standard error detection algorithms.

Further problems with evaluating the accuracy of the artifact detection algorithm was the lack of data from actual error reports at STEricsson. Hence the information on what types of artifacts that are actually common when developing mobile platforms has been incomplete, and only based on discussions and a minor survey sent out by email to colleges at the multimedia testing division.

The testing and evaluation has thus relied on one part with artificially constructed videos in video editing software, and in one part on compressed videos with introduced bit-errors.
The first test suit is meant to test how small errors and artifacts the algorithm can find, when considering the affected area of each frame in the video. Since the chosen artifact type in the first suite is chosen somewhat ad hoc, the second test suite is chosen to cover a broader set of artifacts, since the errors associated with bit errors are very stochastic in nature, producing everything from color changes, blockiness, stripes; and in all different sizes. The first test is thus made to estimate the accuracy of the algorithm. The second test is meant to be so general that if the algorithm manages to find the diverse types of errors and artifacts in those videos, it will probably find most actual errors in videos and images as well, regardless of their more specific nature.

### 4.4.1 Non-Reference Methods

Generally these methods performed poorly on the test material, and at a pretty early stage it was concluded that they were not robust enough for this application. These methods are still experimental and in early development, and only seem to have a few successful applications e.g. for satellite imaging as previously mentioned. Nonetheless they seemed interesting enough to investigate further because of their truly non-reference nature. However there seem to be lots of work left before usable and robust non-reference methods are available for use in artifact detection.

### 4.4.2 Reference Methods

The method based upon SSIM gave better results than the previously mentioned. According to the results in Figure 32, the first test suite seemed to estimate the resolution of the algorithm to $2 \times 2$ blocks of the type in Figure 22. Below that limit the algorithm could not reliably distinguish between artifacts and the level of the noise floor. Although when visually inspecting the corresponding probability map, the artifact is clearly visible so this may be caused by inadequate denoising.

Due to the random nature of errors introduced by bit-errors in compressed video, the relationship between perceived artifacts in the video, and the number of bits flipped is highly non-linear. The number of bits flipped can therefore not really be used as a measure for how distorted the associated videos are. Instead the bit-error videos have been visually inspected and
chosen because of there varying characteristics. In Figure 31 and corresponding Figure 30 plots of errors and extracted frames from the bit-error videos are shown.

Figure 30: Plots of errors (using the introduced metric (77)) in six different videos with added bit-errors. The plots corresponds to the video frames in Figure 31. The numbers in the filenames signify between which frames the errors are introduced, and how many of the bits that are flipped. The severity of the errors however does not seem to have a strong correlation with this last number. Videos have rather been chosen for their visual characteristics.
Figure 31: Example frames from six videos with different artifacts introduced by bit-errors.
Figure 32: A comparison of errors (using the error metric \((77)\)) in videos with artificially introduced errors via a video-editing program. The introduced errors have different areas signified by the length of the side in blocks of the type in Figure 22. The purpose of this test is to find the resolution limit of the algorithm. The red threshold is somewhat arbitrarily chosen, but is meant to lie just above the noise floor.
Figure 33: An example frame from a video with an introduced error with an area of $4 \times 4$ blocks.

From these figures it can be concluded that the algorithm detects most types of major artifacts and errors. There are still some cases however where the rather high noise floor masks minor errors in the videos.

In general the performance of the algorithm is satisfactory, especially with regards to the complexity of the problem and the ad hoc, low quality setup that has been used.
5 Improvements and Future Work

There are several improvements in the test setup for the SSIM-based artifact detection scheme which should be able to increase the performance of the algorithm significantly.

- High resolution photo quality print on a thick and rigid type of paper.
- Dedicated setup for rotating and translating the image that is robust and properly aligned.
- A fixed and robust tripod or other type of camera support where the mobile phone can be mounted.
- A controlled environment with a uniform one-color (black) background and uniform lightning not casting any shadows or causing glare.

The artifact detection algorithm has been developed and evaluated using rather ad-hoc test material. Using a collection of actual artifact ridden images and video should help to tune the algorithm to be more effective and concentrated on plausible and common problems. As it is now the algorithm has had to be kept very general which probably have had a negative impact on its performance.

One possibility is also, since the tool kit is made to mimic the judgment of actual human observers, to try to tune the algorithms in accordance with that. This could be done either using existing data sets that are used for e.g. codec evaluation or make a new survey focusing on for example artifact severance in different types of images and video.

Because of the generality of the proposed toolkit there should be other interesting applications using the whole of or parts of it. Some appealing uses for the algorithm could for example be

- General camera evaluation of either specific functions (zoom, focus, lens aberration etc) or the system as a whole
- Image/video codec evaluation
- Live or steaming video quality assessment over network or Internet. The algorithms probably need further optimizations/simplifications for this to work since the performance is pretty far from real time as of current.
References


