Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Jussi Judin

A data encoding approach to video communication system verification

Master's Thesis
Espoo, May 17, 2013

Supervisor: Professor Erkki Oja
Instructor: Jukka Lehtniemi, M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author:	Jussi Judin		
Title: A data encoding approach to video communication system verification			
Date:	May 17, 2013	**Pages:**	97
Professorship:	Information and Computer Science	**Code:**	T-61
Supervisor:	Professor Erkki Oja		
Instructor:	Jukka Lehtniemi, M.Sc. (Tech.)		

Standard software development practices emphasize more and more test automation that enables developers to focus on new functionality instead of manually making sure that the existing functionality still works. We mostly rely on our own eyes when we want to verify that images look as they should. This makes it especially cumbersome when we want to verify that a video communication system works correctly even after changes to the system. Video communication systems pose unique challenges from a test automation point of view as such systems makes the image go through various distortions that follow from the video encoding, transfer, and display processes that make it hard to automatically verify if the output has everything that is expected.

The goal of this thesis is to investigate methods that can be used for functional verification of video communication systems. The verification consists of checking if the output video data includes all expected input video streams depending on the system state. This includes the implementation and evaluation of one method that can encode data into video that will then make it possible to see if the data passes correctly through the video communication system under test.

The used method was a custom 9×9 matrix type binary barcode encoding 60 bits of raw data. Its detector is based on edge features that assume the use of linear interpolation as the resampling filter. This results in a detector that correctly decodes barcodes adhering to this model with the smallest module size of 1.9 pixels. When lossy JPEG image compression is applied to the barcode image, minimum fully successfully recognized module size varies between 2–4 pixels depending on the compression ratio.

Keywords:	information encoding, digital image processing, two-dimensional barcode, computer vision, video compression
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Jussi Judin		
Työn nimi: Videoviestintäjärjestelmän oikeellisuuden tarkastaminen dataa koodaamalla			
Päiväys:	17. toukokuuta 2013	**Sivumäärä:**	97
Professuuri:	Tietojenkäsittelytiede	**Koodi:**	T-61
Valvoja:	Professori Erkki Oja		
Ohjaaja:	Diplomi-insinööri Jukka Lehtniemi		

Ohjelmistotestausta automatisoidaan nykyään yhä enenevissä määrin, mikä mahdollistaa kehittäjille keskittymisen uusien ominaisuuksien tuottamiseen olemassaolevien ominaisuuksien toiminnallisuuden varmistamisen sijaan. Videokuvan oikeellisuutta tarkastellaan pääasiassa silmämääräisesti. Tämän seurauksena videoviestintäjärjestelmien oikeellisuuden tarkastaminen muutosten jälkeen on erityisen raskasta. Videoviestintäjärjestelmille ominaisena testiautomaatiota hankaloittavana haasteena on kuvan vääristyminen videopakkaus-, siirto- ja näyttöprosessien seurauksena. Kuvan vääristyminen tekee hankalaksi määrittää sen, löytyykö ulostulevasta kuvasta kaikki haluttu.

Tämän työn tavoitteena on tutkia keinoja videoviestintäsovellusten toiminnalliseen testaukseen. Tämä toiminnallinen testaus koostuu ulostulevan kuvan tarkastelusta järjestelmän oletetun tilan perusteella. Työssä toteutetaan ja arvioidaan yksi menetelmä datan sisällyttämiseksi videoon. Tällä menetelmällä voidaan varmistaa se, kulkeeko videokuva oikein testattavan videoviestintäjärjestelmän läpi.

Valittuna menetelmänä käytettiin mukautettua 9×9 moduulin kokoista matriisityyppistä kaksiarvoista raakakapasiteetiltaan 60 bittiä olevaa viivakoodia. Tämän viivakoodin lukemiseksi luotu tunnistin käyttää piirteinään reunoja ja olettaa kuvan kulkevan lineaarisen interpolaatiosuotimen läpi. Tunnistin tunnistaa tämän mallin perusteella oikein viivakoodeja, joiden moduulien koko on pienimmillään 1,9 kuvapistettä. Häviöllisellä JPEG-kuvanpakkauksella täysin onnistuneesti tunnistettujen viivakoodien moduulikoko vaihtelee 2–4 kuvapisteen välillä riippuen kuvanpakkausasteesta.

Asiasanat:	informaation enkoodaus, digitaalinen kuvankäsittely, kaksiulotteinen viivakoodi, konenäkö, videopakkaus
Kieli:	Englanti

Acknowledgments

I want to thank Jukka Lehtniemi for his long-term vision that led to this thesis and for his relentless push to get this thesis forward. I would also like to thank the people at Oy LM Ericsson Ab for supporting this work and providing means to accomplish this.

The appearance of this thesis would not be as good without the typographical advices from Miika-Petteri Matikainen and the written text did benefit from the keen eye of Jukka Lehtniemi. I also want to thank my supervisor, professor Erkki Oja, for his suggestions and corrections.

Espoo, May 17, 2013

Jussi Judin

Contents

Chapter 1

Introduction

The market for video communication is growing fast[1], as the rising end-user bandwidth[2] and ubiquitous integrated video cameras make it possible to experience the richness of real-time video communication in situations where formerly communication by voice or text were the only options. Systems for video communication pose unique challenges from the software testing point of view. They may provide different layouts and output stream combinations for the resulting video data depending on the viewing participant, participant count, video output system capabilities, and on other external factors.

Humans can easily check if a video communication system works correctly just by looking at the resulting output. This is demonstrated in figure 1.1 that shows a typical multi-participant video conferencing situation where a fault in mixing results in an incorrect output image. Although this is easy to verify during the development, verifying that all parts of the system work correctly after every change is slow and expensive to do manually. Automated software

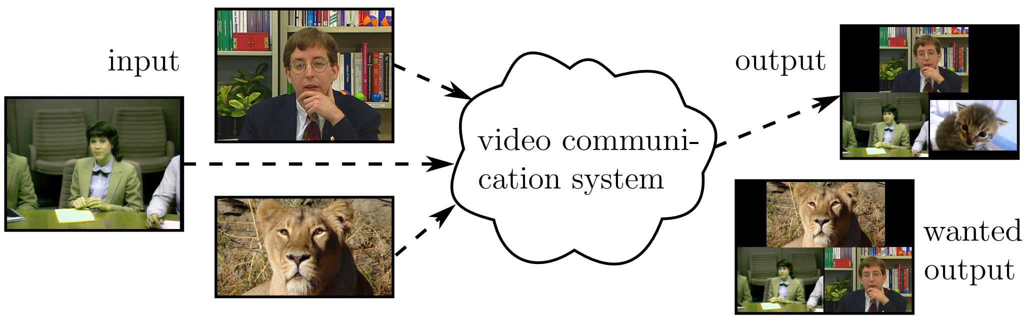


Figure 1.1: An example video communication system fault where video mixing results in an incorrect output image.

testing tries to ensure that defects resulting from changes to the software, in the system, or to the environment will be noticed as soon as possible.

Software testing by machine vision methods is already enabled by commercial and free software applications[3, 4]. These applications focus on user interface testing by searching for static graphical elements and text strings. A video communication system, however, can produce a picture that looks correct to a human observer even after the original video stream is subjected to various distortions. These distortions can come from scaling, occlusion, embedded user interface elements, video compression, and other filtering operations. Also, as the video image does generally not stay static and changes over time, these distortions can also change over time. From the software testing perspective these distortions combined with the high rate of change of the video image make it hard to verify if the resulting image looks correct.

The goal of this thesis is to investigate methods for encoding information into video data that can be then decoded and used for functional verification of the resulting output image. The output image is expected to undergo distortions that can occur in video communication systems.

1.1 The structure of this thesis

This thesis is divided into four major sections. First, chapter 2 describes how a video communication system transfers and modifies the video image. Chapter 3 then examines a few different methods that can be used to encode data into images and their suitability for this task. This then results in a custom barcode and its detector, that are described in chapter 4. Finally, the barcode detector is evaluated for its sensitivity and robustness in chapter 5.

1.2 A word about images

This thesis includes color images that are best viewed on a display device with color gamut that can show the full sRGB color space. The electronic copy of this thesis is available from the author and from places that distribute such copy.

Chapter 2

Video communication systems

High bandwidth communication links have enabled video communication in places where communication by voice or by text were the only viable options. Video communication applications range from simple static file transfer applications to real-time broadcasting applications including multiple participants with a large receiving audience[5].

This chapter focuses on describing how the video stream flows through video communication systems that are meant primarily for face to face communication. This also shows what challenges there are for testing the video output of such systems.

2.1 Services

Video communication services range from simple video on demand services that provide possibility to view pre-recorded content without too tight real-time constraints to highly interactive multi-participant, multi-endpoint video conferencing applications with more complex video stream structure[5, 6]. Here a video communication system refers to a collection of applications and links between them that may be used for two-way interactive video communication. These applications include video calling, video conferencing and remote collaboration with video image support. These generally have the view of one or more participants in the same video stream and optional user interface and shared content elements. Different participants can also receive different video image that depends on the participant count and the type of communication.

Fitting multiple streams onto a limited screen estate creates the need for output stream manipulation and selection. Output manipulation includes scaling, cropping, occluding, and arranging selected streams to the screen area that the display device provides. These operations make it harder to use direct video image comparison based methods, as streams from multiple clients get mixed together. Also, the possibility for varying amount of delay means that we can not rely on knowing the exact stream position for each input stream that is included in the output video stream.

2.1.1 Functional verification environment

The main challenge in the video communication system testing is to see if video streams go through the intermediate processing steps correctly. To do this in the testing environment, we generally need to simulate the video stream creation and consumption parts. This is done by injecting our own data through the processing pipeline and see if a correct looking picture comes out. We can inject this data in multiple places:

- by showing our own video stream to the video capturing device.
- by feeding the video capturing interface with our own video stream.
- by simulating the video capturing device.
- by using our own client software that only sends a predefined video stream.

The output can be captured in similar fashion. These different approaches have their advantages and disadvantages. The smaller part we try to simulate, the smaller are the set-up and run-time demands on the testing process. On the other hand, if the test verifies that only one part works, this does not guarantee that the whole system works properly under varying conditions. For example, simulating our own video capturing device enables us to be hardware-independent during testing. This does not, however, verify situations where the capturing device is accessed incorrectly and thus results in an incorrect or completely missing image in the real use case. Therefore the test method for video communication system should be flexible enough that it can be used in any input and output phase.

2.2 Video transmission

Video communication between two or more participants requires that the video stream is transferred from one participant to all participants that want to receive that stream. As video stream is read from the video capturing device, it has a certain frame rate, resolution, color space, color depth, and may be read by interlaced or progressive scanning method. Any of these properties may need to be changed to match the ones that the receiving end accepts. Video stream may also be modified to make it easier to match the available bandwidth constraints at the sending or the receiving end. The sender can also filter out the noise and add overlay graphics or text to the video stream before the stream is encoded and sent to the receiving end inside an appropriate container format.[6]

Sending video streams to receiving clients is usually done either by sending the same video stream to all receiving clients directly, or by using an intermediate service that then forwards the stream to receiving clients. This intermediate service usually lessens the bandwidth and video format requirements on the receiving end. The intermediate service can also combine multiple streams together and transcode them to match the receiving client's capabilities.[6] As the intermediate service can hide its own architecture from clients, it can be dynamically scaled to support much higher client counts than it would be possible to support even by highest bandwidth connections[7]. This intermediate processing, on the other hand, adds some delay that is especially undesired for real-time video communication.

As the receiving end receives one or more video streams, it then decodes them and makes the video image appropriate for viewing by applying necessary transformations, like scaling and moving the video image to correct position on the screen. As the video is subsequently shown on the display device, it may also include overlaid user interface components and other occluding elements.[6]

2.2.1 Transmission issues

Two-way video communication generally is a real-time application that suffers from the underlying transmission layer issues. Every method to overcome these issues is a trade-off between the required resources and service quality. Common quality issues arise from excessive delay, insufficient transmission capacity, and from packet loss and corruption[8, 9]. These issues bring out the need for various error recovery mechanisms on the receiving end to be able to show the picture as error free as possible[10].

Excessive delay reduces the real-time feeling of video communication and arises from video processing and network transfer delays. Video processing delay arises from the time required for encoding and decoding, from the requirements for rate control buffering, and how far into the future the video decoder can look to find redundant data[11]. When reducing the encoding delay, the trade-off is a loss of quality for the predefined bitrate, as the encoder has less means to find out redundant data. Network delay mainly results from the flight time of packets and from the intermediate buffering and processing by the networking equipment[12]. Reducing the network delay usually means using alternative service providers, network services with lower delay, or higher priority routes. Trade-off in this usually is the higher cost for such services and being physically bound to places where these better connections are available.

Varying transmission capacities of the video communication session participants due to different connection capacities or changing network conditions pose their own challenges for sending video that matches the available bandwidth. Approaches to mitigate this vary from sending the same video stream encoded with different bitrates to using layered approaches. The disadvantage of encoding the same stream multiple times with different bitrates is that it requires more resources from the sender side and the receiving side can not that gradually adapt to changing network conditions. In layered approaches video is encoded by having a base layer that requires a certain amount of bandwidth and subsequent enhancement layers that provide better image quality for higher bitrates[13, 14]. Layered approaches adapt to the varying bandwidth requirements with less abrupt changes but need higher bitrates to give the same image quality as non-layered approaches.

Various network issues may lead to lost, corrupt, or late video packets. These result in lost data that needs to be taken into account to provide smooth viewing experience. One way to do this is to ask the sender to re-transmit the already sent data. This adds to the decoding delay and increases the memory requirements of the sender, as the sent video data needs to be kept in memory for some time. Another way is to add forward error correction that includes error correction data in addition to the video data. This, however, reduces the available bandwidth for the actual video data and does not guarantee that all lost packets can be recovered[15]. When video data is irrecoverably lost, errors will propagate until the affected parts are refreshed by self-sustaining image data[16, 17]. Therefore some form of spatial or temporal error concealment, that tries to minimize the visible error until the affected part is refreshed, is useful so that the affected parts, or the whole video stream, do not need to be frozen. Error concealment techniques may need a large amount of processing power and therefore may not be applicable in devices that do not have much extra processing power to spare[18].

2.3 User interface

User interface is the part that affects how video streams are shown to individual participants. The resulting image on the display device is affected by

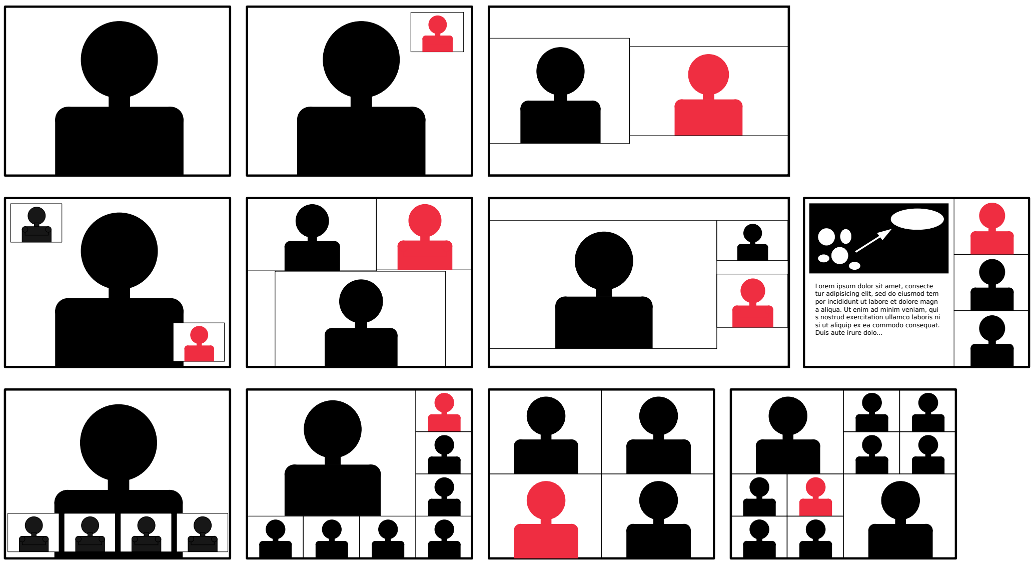

Lorem ipsum dolor sit amet, consecte
tur adipisicing elit, sed do eiusmod tem
por incididunt ut labore et dolore magn
a aliqua. Ut enim ad minim veniam, qui
s nostrud exercitation ullamco laboris ni
si ut aliquip ex ea commodo consequat.
Duis aute irure dolo...

Figure 2.1: Different layout choices for different numbers of participants and aspect ratios. Red color indicates sender's video stream.

the application type, the screen estate that is available, by the number of participants, and the output layout that is in use. This creates challenges for output verification as the position and the size of a single video stream picture is unknown. The video stream may also be partially cropped or occluded[19, 20].

The application type affects how much focus the actual video stream image gets. If the application is a general use collaboration application or if it primarily provides other means for communication, then the video image may take just a small spot on the screen and other user interface elements will take most of the screen estate. However, if the application is meant for video calls, video conferencing, or is a part of a telepresence system, then the video image has the most significant role and other user interface elements are there just to enable video communication between participants.

Participant count and the available screen estate generally decide the usable layout choices. High-end room and telepresence systems may focus on providing separate screens for different streams and rely on static camera positioning[21] to eliminate the need to show the outgoing video stream to the sender. On the other hand if cameras are not static, or are integrated into screens of the sending and receiving devices, viewing own picture creates one additional stream that needs to be displayed on the screen. Also, if the participant count is higher than can comfortably be fit on the screen at once, then there is a need for a scheme that decides which streams are shown on the

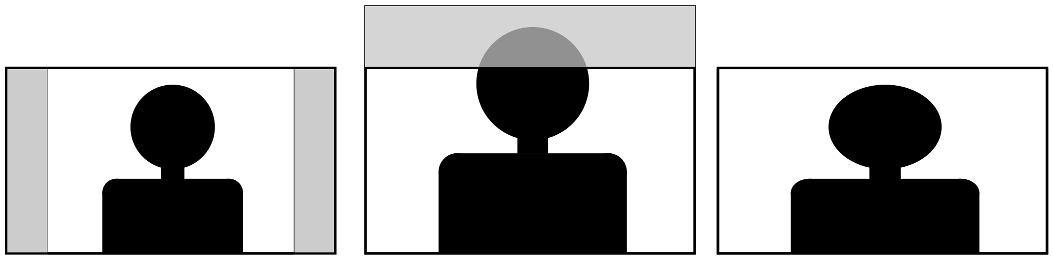

(a) Added borders. (b) Cropping. (c) Stretching.

Figure 2.2: Different methods for fitting different aspect ratio images together.

screen. This scheme can generally be based on static choices by the receiver, by the session moderator, on speaker activity, or by some mix of those schemes[6].

Figure 2.1 shows different layout variations that take into account different participant counts, communication modes, and client capabilities. Issues with different layout variations include image scaling to fit more participants on screen or to emphasize some participants over others, occlusion by user interface elements or by picture-in-picture functionality, and issues from differing aspect ratios. The picture-in-picture functionality is used to emphasize the main participant by giving it more screen estate and secondary streams can be laid on top of this main image. Figure 2.2 shows an example how different aspect ratios between sender's video capturing device and receiver's display device may be taken into account. The received image may need extra borders to be added to the top and to the bottom to be able to have the whole image visible on the screen. This, however, leads to unused screen estate. The received image may also be cropped or stretched to fill the whole screen area that is allocated for the image. Here cropping leads to visually visible lost image data and stretching makes people look too thin or too thick[22]. These operations can be combined and used intelligently to maximize the used screen estate with minimal distortions to the actually interesting parts of the image.

2.3.1 Scaling

Image scaling, or spatial interpolation, is one of the fundamental operations that enables showing the same video image on different resolution displays. Downscaling also makes the video image easier to compress for low bandwidth links. It also enables a better use of the limited screen estate by fitting multiple streams images on one screen. The scaling of the video stream can be done multiple times: when capturing the video, scaling the captured video for

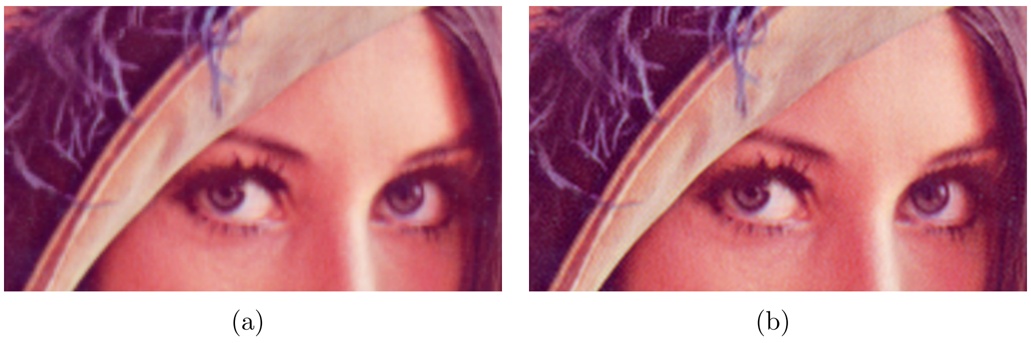


Figure 2.3: Scaling the picture of Lenna by a factor of 4 in (a) the spatial domain by bilinear interpolation and (b) the discrete cosine transform domain.

transfer encoding, in the transcoding phase, while generating the output layout, and when displaying the final image on the display device.

The real-time nature of video communication limits the choice of image scaling algorithms to ones that are fast enough to be used as a part of real-time processing. Still, there are multiple different algorithms in use that will result in slightly different scaled images[23]. To demonstrate the difference between various scaling methods, figure 2.3(a) shows the result of scaling with bilinear interpolation in the spatial domain and figure 2.3(b) shows the result of scaling the same image in the transform domain[24]. Bilinearly interpolated image results in blurring seen in the high detail areas, whereas scaling in the transform domain maintains the high detail areas sharp but results in rippling seen in the flat areas.

2.3.2 Frame rate changes

Video image may also be distorted by frame rate changes by temporal interpolation[18]. Frame rate changes are needed for similar reasons as image scaling, namely to be able to support different client capabilities, transport media, and display device refresh rates. For example, a cheap web camera may have varying capture rate depending on the lighting condition of the room, but the encoder expects constant frame rate to be able to maintain desired bit rate and frame size. This stream may be then mixed with other streams that have different frame rates to form a single combining stream with the desired layout. This can then again lead to frame rate changes depending on the receiving client capabilities. Then the receiving client needs to display this video stream on its display device that has a certain refresh rate that

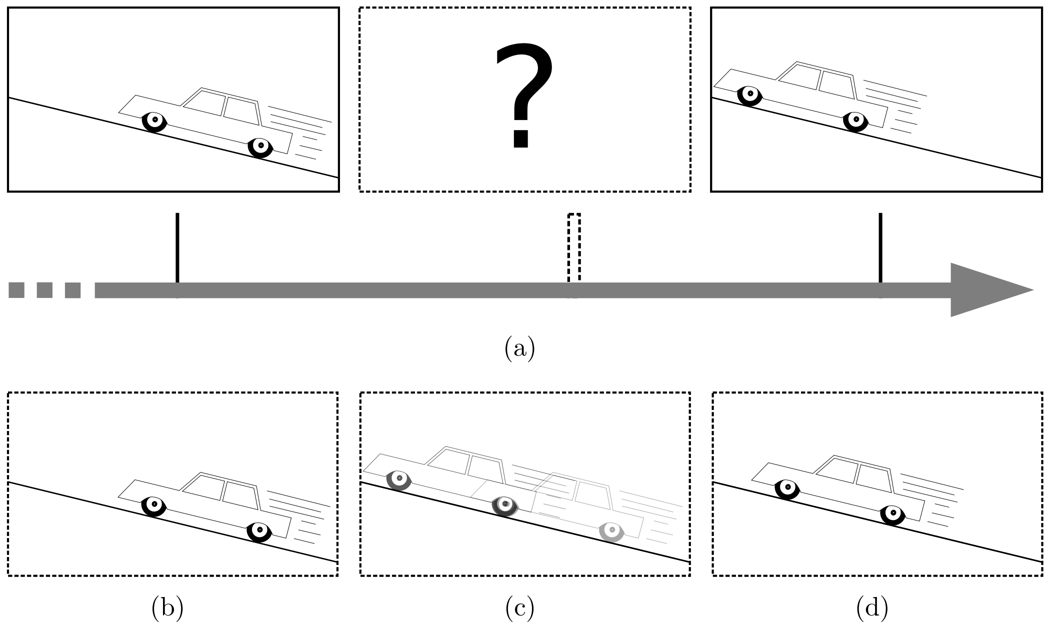


Figure 2.4: Frame rate change sampling methods with (a) a timeline where the frame sample lies between two existing frame locations. Sampling by using (b) the latest frame, (c) the weighted average of the two closest frames, and (d) motion compensated interpolation.

depends on the frequency at which the displayable data is read from the frame buffer[25].

Similarly, as for spatial interpolation, there exist multiple methods for temporal interpolation that have their own different compromising features[18]. Figure 2.4 demonstrates various temporal interpolation methods that may be used for frame sample creation in a case where the new frame sample lies between two existing frames. The most obvious method is to use the latest frame that is before the current sample point (figure 2.4(b)). More computationally demanding methods based on higher order filters use more than one frame as the basis for interpolation. We can, for example, create a linear interpolation filter that uses the weighted average of the temporal distance of the two closest frames to create an interpolated frame (figure 2.4(c)). And one less used scheme, that is especially useful for increasing the frame rate, is to use motion compensated spatiotemporal interpolation[26] shown in figure 2.4(d). Motion compensated interpolation tries to infer the movement of the content from existing frames and create new frames based on that.

For video communication applications, however, the more advanced interpolation methods that depend both on the past and future frames around the

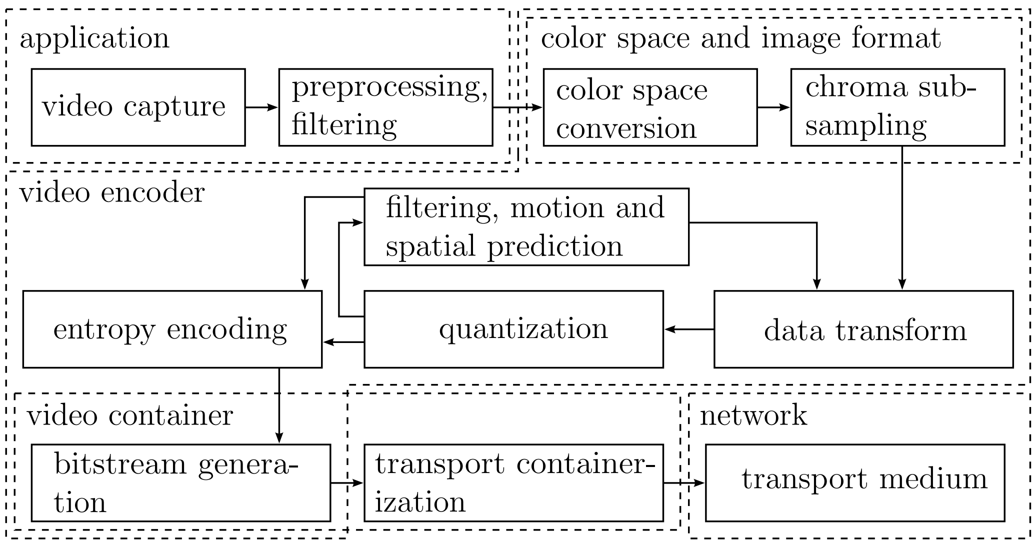


Figure 2.5: Image manipulation steps and manipulating components responsible for making the original captured image suitable for network transfer.

sample point obviously add delay to the encoding process. Therefore, even though those methods might produce better results, especially for dynamic parts of the image, the delay they add will definitely not be welcomed.

2.4 Encoding

Video is encoded to some compressed format, as the uncompressed video data requires too much bandwidth to transfer it affordably in real time over long distances. Nowadays most popular video encoders for real-time communication are based on MPEG-2[27], H.261[28], MPEG-4 part 2[29], H.263[30] or MPEG-4 part 10/H.264[31] standards where the video compression algorithm throws a part of the original image data away with the goal of minimal visible distortions for natural images[6]. This makes it harder to know what to expect from the resulting video data when we want to encode information to it.

The general transform based lossy video encoding process consists more or less of five steps that result in data rate reduction[5], shown in figure 2.5. First, encoding begins by transforming the source image to an appropriate color space. Then the actual data reduction process usually begins by reducing the resolution of color information encoding chroma channels. This data is then passed to transform coding based encoder that then tries to maintain as much visually relevant features as possible[32].

Transform based encoding begins by transforming the image, or parts of the image, so that the most relevant visual features can be encoded with the smallest amount of data and the less relevant image features get thrown away in the process[5, 33]. Transformation leads to a representation that provides coefficients describing the image features that then are quantized to the selected accuracy. Quantization reduces the effective accuracy of transformation coefficients and leads to the loss of visually less important features for the transformed block. Usually there also is high temporal similarity between subsequent frames, and therefore the difference between different frames can be transformed and quantized in a similar fashion as encoding pure source image data. This may also take the motion of objects between different frames into account to reduce the encodable difference between temporally close frames. And finally, when the image data has gone through these lossy encoding steps, it is encoded for transfer by using lossless encoding techniques[5]. Following subsections focus on what causes the image distortion in the various encoding phases.

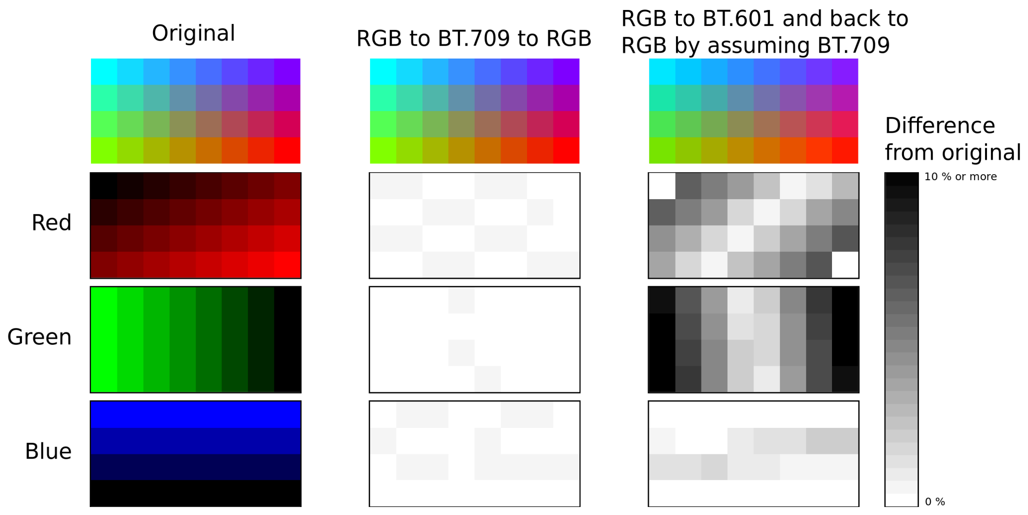


Figure 2.6: Absolute color channel value differences in comparison to the original resulting from converting correctly and incorrectly between $Y'C_BC_R$ and $R'G'B'$ color spaces.

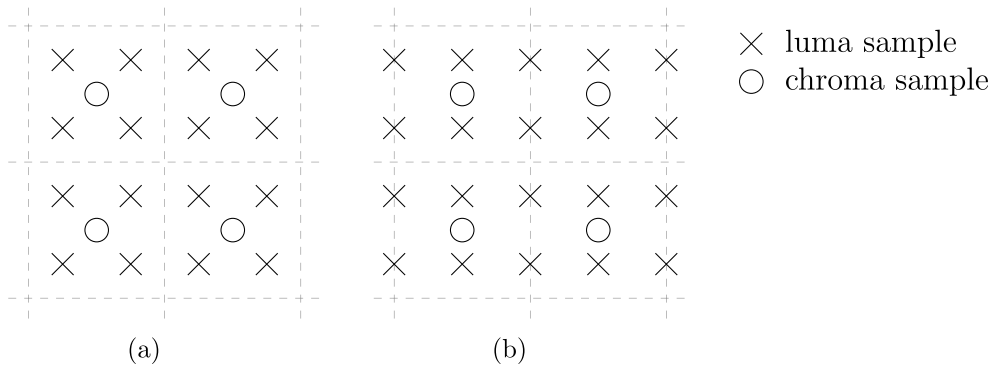


Figure 2.7: 4:2:0 chroma subsampling with (a) interstitial sample placement and (b) cosited sample placement. Dashed lines show the extent of a chroma sample.

2.4.1 Color space conversion

Video input devices and display devices generally handle colors in the gamma corrected nonlinear R′G′B′ color space with red, green, and blue color channel components. For data transfer purposes these color channels are then converted to the $Y'C_BC_R$ color space with one luma, and blue and red chroma channels[32]. The result from this conversion may vary depending on which conversion method is used, usually ITU-R BT.601[34] or BT.709[35]. Figure 2.6 demonstrates the difference between the original correct color values, even when the color space conversion is performed correctly, and when the assumption for the variant of the $Y'C_BC_R$ color space is wrong. This shows that even though we have the correct color space conversion method in use, we still get a small difference between the original and once converted R′G′B′ color values.

2.4.2 Chroma subsampling

One of the first intentional data reduction steps in video encoding is chroma subsampling. In chroma subsampling the resolution of the two chroma channels is reduced. This can be done because the human vision does not perceive the loss of the color detail as strongly as the loss of detail in the luma channel[32]. The usual resolution reduction method for distributable video content is to use 4:2:0 chroma subsampling where the number of chroma samples is reduced to one-fourth of luma samples. This sample reduction method, however, has various alternatives for doing the actual subsampling, for placement of the subsampled values in relation to luma samples, and for interpolation between

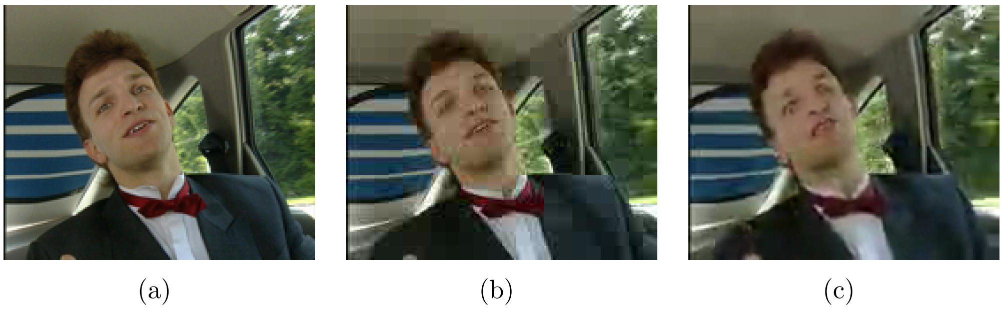


Figure 2.8: Blocking and blurring artifacts created by low bitrate encoding of (a) the original image by (b) H.263 encoder and (c) H.264 encoder.

samples. Although small variations in the subsampling generally do not lead to easily discernible differences, they still result in numerically different values[36].

To demonstrate two widely used 4:2:0 subsampling methods, figure 2.7 shows an example of two different chroma subsampling implementations. Figure 2.7(a) demonstrates a situation where a chroma sample is placed in the middle of four luma samples. Here a simple averaging filter can be used for the subsampling. On the other hand figure 2.7(b) shows the case where chroma samples are placed vertically between luma samples but horizontally coincident with luma samples. Then we need a weighted averaging filter to create chroma samples.[32]

2.4.3 Transform coding and quantization

Image data is usually transformed to more compressible format by using some form of reversible transform coding where parts of the image are transformed to a set of coefficients that are then quantized. The goal of the selected transform is to focus the most important visual data on the least amount of coefficients. This way stronger quantization can be used to eliminate the less important coefficients, that then result in reduced information content. This leads to a data representation that is then easier to compress by lossless compression methods[33].

In the most often used discrete cosine transform based video compression methods heavy quantization can cause distortions like blocking, edge busyness, mosquito noise, and quantization noise[37]. Blocking results in clearly visible separate blocks in the image, demonstrated in figure 2.8(b). This results from independent block based transformations that do not take dependencies between neighboring blocks into account. Quantization noise comes from coarse quantization steps and looks like random noise[38]. Edge busyness and mosquito noise are distortions resulting from the compression process that happen near sharp edges and can lead to image noise near edges or to time-varying edge sharpness[39].

Transform coding can be applied either directly to image blocks of the original image or to the difference between video frames[5]. When the transform coding is applied to the original image, it is called intra-frame coding. Intra-frame coding results in compressed data that is completely independent of surrounding video frames. This is needed as a starting point for image display and can also be used when recovering from data corruption. When the transform coding is applied to the difference between images, the difference is calculated between an already decoded block and between a block in some other frame. This is called inter-frame coding and it enables high compression ratio for video by taking the advantage of the slowly changing video data between consecutive frames. Inter-frame coding, however, at the same time makes the video stream vulnerable to lost packets and data corruption, as a single error will propagate until the affected image area is refreshed by intra-coded image data.

2.4.4 Transcoding

Video transcoding is re-encoding the whole or parts of the video stream, generally to some other video format or to change attributes of the current format. These changes may include bitrate changes, frame rate changes, or frame size changes[5]. This is usually done to support variations in network capacity and client capabilities.

The most obvious way to do transcoding is first to decode the encoded stream and then apply the wanted operations to the decoded stream. These operations may include scaling, frame rate changes, additional content addition, and deinterlacing. After these modifications are done, the stream is then re-encoded to the new format. This, however, results in additional image distortions in addition to the distortions that already come in the originally encoded video stream.

Video stream modifications can also be done for the transformed data without fully decoding it first, or by re-using some aspects of the original encoded video data, like motion vectors. These both tricks generally are done to reduce the computational burden of re-encoding, but can also lead to additional loss of quality. This loss of quality can result from less accurate motion vector data or from losing important visual features from the transform coefficient modifications that would otherwise be captured by the encoder if done for the decoded image. Alternatively, only a part of the frame can be transcoded so that only the modified image parts are decoded and re-encoded, and the untouched parts are passed through as they are. This trick to save processing time may be useful when doing content insertion, like adding subtitles or providing picture-in-picture functionality[40].

Transcoding generally also adds some extra delay to video stream forwarding. Even though this is not directly related to quality degradation, it will affect the perceived video communication quality. Therefore it is desirable to make sure that all aspects of transcoding can be done as quickly as possible without adding noticeable extra delay.

2.4.5 Other distortions

In addition to these distortions that generally happen for every block based encoder, various encoder and implementation specific distortions exist. For example in-loop deblocking filter[41] used by the H.264 encoder causes blurred features that are visible from figure 2.8(c). Also the H.265 encoder[42] includes a different deblocking filter than the preceding H.264 encoder, and two additional optional filters to reduce distortions caused by the preceding encoding steps[43, 44].

Implementation specific errors can also come from requantization errors in the transform phase due to different quantization step sizes between the encoder and the decoder[45], especially in older video encoders. This will lead to small differences in what the encoder and the decoder think is the reference image. This can cause cascading errors that lead to a distorted output image. Newer video encoders however eliminate these kind of implementation specific differences by removing the need to use floating point calculations, whose accuracy is machine dependent, by unambiguously defining fixed point encoding and decoding steps[46].

Chapter 3

Encoding data into images

Data generation is an often used approach in software testing when we try to see if some information properly flows through the system under test. In this method we generate a data pattern that is injected to input data and is then searched from the output data. We can use similar approach for testing video communication applications, where we encode data into the input video streams. As video stream consists of individual frames, we can encode the current stream and the frame number to the input stream. Then we can detect and decode the encoded data and use this information to find out what part of which video stream is visible, and where individual streams are in the resulting output.

This chapter will first look at the requirements that we have for testing a video communication system. This includes the resolution, frame rate, and bitrate limits that define how much data we can theoretically encode in the worst case situation. Then we take a look at a few methods for encoding information into images. These methods give us a basic understanding upon which we can then create our own data encoding method for video communication system verification.

3.1 Requirements

Video communication systems have their own requirements for what kind of processing the embedded data needs to tolerate to be successfully decoded. Chapter 2 describes what happens to the image when it goes through the video communication system. If we put some numbers to all those phases, we can have some guidelines that we can use when selecting the data encoding method.

Section 2.3 describes how the user interface can look and what limitations the data embedding process needs to overcome. The picture on the screen can be divided into smaller subpictures that take up somewhere between 1/4 and 1/5 of the screen width and height. This leads to the need to know the spatial position of a subpicture. If we look at the example frame sizes that video encoder standards[30, 31] or a recommendation for video terminals[47] give, we can see that 128×96 luma pixels often is the suggested minimum image resolution. Also 128 pixels is the minimum resolution for many mobile phone screens[48]. Thus we can consider 128×96 pixels as the minimum desirable video resolution that we can target. Also if the screen is divided into even 1/4 sized subpictures, this gives us 24 pixels on the shorter side for data encoding in the luma channel. Chroma subsampling, described in section 2.4.2, affects the resolution of color information encoding chroma channels. If we want to use these channels for information encoding, we need to be prepared for 1/4 resolution loss in them. The maximum realistic video resolution highly varies depending on the video encoding standard, its level, available bandwidth, and on the resolution that the receiving end supports. At the time of this writing 1920×1080 is the highest supported resolution in decently priced consumer devices. This leads to 11.25 times vertical resolution increase when scaling upwards and 1/45 resolution decrease when downscaling for small subpictures. The data encoding method should survive such resolution variations.

The bitrate of the video stream provides the upper limit how much unique data we can theoretically push through the video encoding pipeline. This is linked to the video frame rate that tells us how much data we can have in one frame on average. If we look at the guidelines in the H.264 specification where the maximum frame rate on the lowest encoding level is 30.9 frames/second for resolution of 128×96 and the minimum bit rate is 64000 bits/second this gives us the average data encoding capacity of 2071 bits/frame. If we combine 4×4 video streams to these frames, this gives us at most 129 bits/stream/frame as the upper limit that we can transport. However, as we are mainly transferring the difference between frames, as explained in section 2.4, we can treat that limit as the limit how much we can change between different frames on average. The video container and transport protocol headers, however, reduce the available capacity for data encoding somewhat. Anyway, we should be able to encode around 20 bits of information/frame so that we can do load testing for several dozens of clients for several minutes at about 30 frames/second without having to worry about the frame counts going around.

The data decoding speed is also an issue, as the detector should be fast enough that we can run the verification in real-time after video decoding or while doing screen capture. This is to make sure that the expected resource usage, verification setup complexity, and the time spent on verification stays low enough so that we can use automatic verification as the standard part in testing.

3.2 Information encoding

When we are looking at the resulting video stream from functional verification point of view, we can think of every frame from each stream as a separate discrete data point. We can determine that the video communication system transfers the image correctly if we can recognize these data points from the output despite distortions created by video transmission, mixing, and encoding. Besides treating single frames as discrete data points, we could also make the encoded data span multiple frames. This is used in video watermarking techniques[18], but it generally only is used to identify individual streams, not individual frames.

We can detect a range of faults if we can determine which input video streams are in the resulting output image, where they are located, and at which frame they are in. This information enables us to verify that correct streams are forwarded to correct clients, the encoding delay corresponds to what is expected, and that the resulting stream locations on the display correspond to what is expected. For example, if the video communication application is used in a presentation where the moderator chooses the main content stream from multiple choices, then the image of the selected stream should replace the old stream. This can be verified by checking that the image area that was previously occupied by the other stream has now changed to the requested one.

This section takes a look at methods that can be used for visualizing discrete valued data in such a way that the information that is encoded in the visualization can be read back. In the case of machine readable data, the ease of implementation and reading reliability are quite essential factors. There also are other data visualization methods besides the ones presented here[49], but the ease of visualization generation, the reading reliability, and locating accuracy of discrete values is not that high. This, for example, eliminates changes in textures as a data representation method, even though textures can be procedurally generated from the underlying data[50].

The data encoding methods examined here are glyphs, text, digital watermarks, and barcodes. Glyphs are mainly for human consumption and text is both machine and human readable method for discrete data encoding. Digital watermarks and barcodes, however, are designed to only be machine readable.

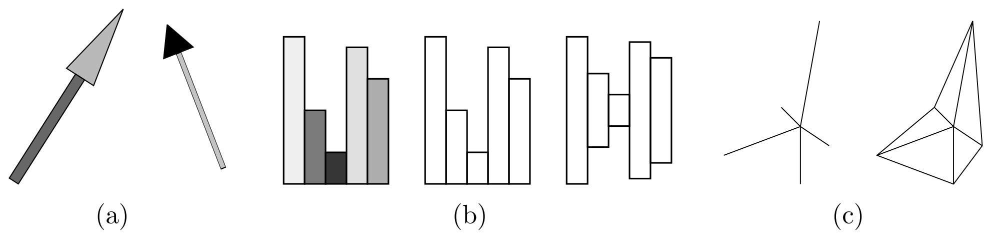


Figure 3.1: Different glyph types that can be used for multi-dimensional data representation. (a) Arrows, (b) profile graphs, and (c) stars.

3.2.1 Glyphs

Glyphs are graphical objects for presenting multi-dimensional discrete data values. They usually are designed in such a way that they can be rapidly and clearly distinguished from each other. This is done by designing glyphs to provide relevant information about the task at hand by using features that quickly enable distinguishing objects with the wanted property from the environment and other objects[49, 50]. Features that provide clear distinction can be position, color, shape, orientation, texture, and motion. All features are not independent of each other, as for example displaying orientation depends on the shape of the glyph. The goal to keep glyphs distinguishable from each, and the fact that different features are not independent from each other, limits the data encoding capability and presentation accuracy of glyphs[49].

Figure 3.1 shows an example of three different types of glyphs. Arrows in figure 3.1(a) can encode information in their direction, color, and size of the different parts. Profile graphs in figure 3.1(b) all encode the same information, where colored graphs also encode the intensity value as the color value. Each bar in the profile graph encodes value of one dimension and more dimensions can be easily added by adding more bars. More dimensions naturally make it harder to search for specific patterns for one dimension, as other dimensions interfere with the pattern. These problems can be somewhat mitigated by correctly ordering the displayed dimensions and adjusting the location of different glyphs in relation to each other[49]. Star graphs in figure 3.1(c) also enable the visualization of somewhat freely selectable dimensionality and have similar problems and remedies as profile graphs when the dimensionality and representable value count increases[49, 50].

Glyphs as a data encoding method are, however, mainly designed for humans. Even though there exist data encoding methods that encode data in glyph like fashion[51], they are closer to a specific barcode type than anything that can be read by a human.

3.2.2 Text

Text is the most used human readable form for visually identifiable information encoding for discrete data when accurate data representation is needed. Computers can recognize text by using optical character recognition engines that take images as their input and output the recognized text and its location[52]. This enables the use of text as data encoding method that can be recognized both by humans and, to a certain extent, a machine.

Written text for wider distribution consists of character combinations whose look is generally determined by the font in use. Humans generally have no problems in adjusting to slight variations in character shapes. Optical character recognition engines, however, need to be general and adaptive enough to be able to recognize text that is written in different fonts[53]. Even though most work on optical character recognition is done for the Latin alphabet, the challenges with different fonts has still led to creation of special fonts that are specifically made to make the optical character recognition task easier and are still readable by humans[54, 55].

Fonts created for optical character recognition, however, are not optimized for the rectangular sample grid that forms the picture on the screen. This leads to a situation where we need to use larger characters to be able to reliably recognize them. This reduces the data encoding density of text written in such fonts. If we, on the other hand, use small fonts that are designed to be used on a computer display[56, 57], we can encode the data by using text that takes smaller area than using more rounded fonts that are not designed for screen use.

Even though text can be used to represent machine decodable information, methods designed purely for machine reading generally offer much higher information encoding capabilities and robustness than encoding data as text and then using optical character recognition for detection[58, 59].

3.2.3 Digital watermarks

Digital watermarking is embedding data to the host signal in such a way that it can be read back by algorithmic means and does not change the host signal in a perceptible way[18]. Digital watermarks are generally designed to survive one or more intentional or unintentional signal manipulation operations. This way they offer a good overview of methods for data embedding and solutions and compromises that need to be used for watermarks to survive image distortions that happen during the video transmission process.

Digital watermarking schemes can be divided into two major categories, depending on their need to have the original signal or some derived attributes available. Having the original data or derived features available makes it possible to focus just on watermark detection. This reduces the need to guess where the watermark data might be in the image and what distortions are applied to the image as these can often be compensated for. Needing the original data to be available naturally limits the applicability of such watermarking scheme. Schemes that do not need the original data or derived attributes have much wider applicability. Naturally, not having the original data available makes detection harder and these watermarking schemes need to have some other method to overcome image distortions than comparison with the original image.[18, 60]

Watermarking schemes can be classified to systems that only detect if the signal under inspection is watermarked with a certain key or not. Extension to this is to embed multiple bits of data into the watermarked signal. This naturally leads to a need to decode this embedded data in addition to watermark detection. This embedded data may be encoded in a format that is intended to be decoded straight away, like in watermarking schemes that rely on correlating the watermarked data with some known signal. Another possibility is to embed data in a format where the watermark decoding process does not directly result in the actual data, but in something from which the actual data can be detected[60], like an embedded image in some transform domain[60, 61].

These data embedding schemes are then subjected to various quality degrading image manipulation operations like cropping, rotation, scaling, lossy compression, and color adjustment. The data can also be subjected to additional intentional watermark removal operations, like averaging multiple differently watermarked copies. This makes watermarking a trade-off between robustness against distortions, the amount of data that can be encoded, and the amount of perceptible changes that the watermarking scheme introduces to the original signal[60]. As the goal of watermarking often is to make itself hard to notice, this makes data embedding hard, as lossy compression algorithms aim to remove the imperceptible features to increase the compression ratio. This leaves very little room to embed data where the embedding is hard to notice. Also watermark embedding and decoding speed needs to be taken into account. Even though more expensive watermarking techniques may enable more robust watermark detection and extraction, in practical applications their price on the performance may be too high, especially for on-line video watermarking and detection[62].

Besides making the watermarking scheme resistant to various distortions in itself, there still is possibility for individual bit values to get improperly decoded. There are two main approaches for achieving robustness against incorrectly decoded data in different watermarking schemes. The first is to embed multiple copies of the same data into the target signal and the second is to use some more sophisticated forward error correction code that can detect and correct errors that occur in the decoded data, or some combination of these two[60]. Embedding the same data multiple times may also enable more robust detection, as partial data loss does not remove the features that may be used for watermark detection[63, 64].

Often used watermarking methods may be divided to several general classes. First, pixel values can be directly modified in the spatial domain and modifications are kept small enough that the embedded changes are not noticeable. These methods include correlation based methods that embed data by using pseudorandom sequences that are then added to the original image data[65], possibly taking the local image features into account[5]. Then there are transform based methods that embed data in some transform domain to the image. This embeds the transformed data over larger area and at the same time gains robustness to image manipulation operations against which the used transform is robust[65]. Although these transform domain based methods do not always directly map to certain locations in the spatial domain, often the same methods that are applicable to spatial domain hiding are used to hide detectable patterns in the transform domain[65]. Then there are combinations of these two methods, where the properties of the used image compression algorithm are used for data hiding, like embedding hidden data in the middle frequencies of 8×8 discrete cosine transform blocks of the JPEG compression algorithm[66]. Salient image features, like strong corners, may be used as locations for image watermarks and the data can then be hidden around these image dependent points[67]. These points can also be geometrically warped to lie near some pseudorandom pattern over the image[68]. When image resolution is high enough, like with high resolution printing, then the watermarked information may be included in the image creation process itself. Instead of using dots as the basic printing pattern, printed image can be generated by using data encoding glyphs as color elements[51]. Video provides additional possibilities for watermark embedding, as the watermark may be divided along multiple frames in the temporal direction. This can be done by using properties of video encoding, like motion vectors, for watermark embedding that are not available for static images[69].

Watermarking methods provide various data embedding possibilities that may be used to embed data into existing video streams or to create completely new video streams that just include the embedded data. Embedding data into an existing video stream creates possibility to verify that the video communication system works correctly with video streams that do not look artificially generated. On the other hand, using just the data embedding part of watermarking scheme enables the creation of more robust test cases that

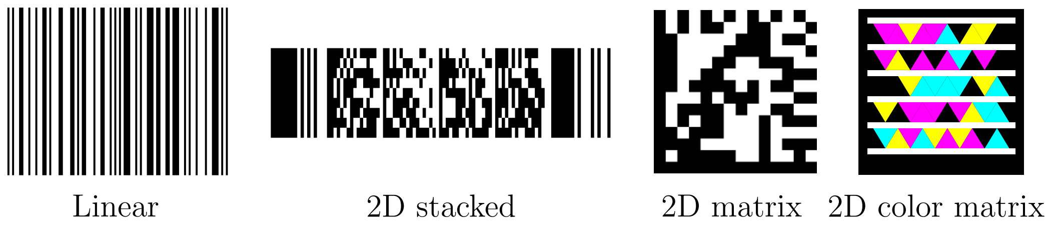


Figure 3.2: Evolution of barcodes from one-dimensional linear barcodes to color coded two-dimensional barcodes.

survive greater variety of distortion causing operations, as the perceptual invisibility part does not need to be taken into account. Consequently, many of the techniques that give watermarks their robustness and machine readability are also used in barcodes, whose properties are described next.

3.2.4 Barcodes

Various types of barcodes are the most often used method for machines to identify objects[52]. Barcode types range from encoding short identifiers[70–72] to barcodes that are designed to have enough capacity to encode hundreds or thousands of bytes of arbitrary data[73, 74]. Figure 3.2 shows how barcode formats have evolved from linear one-dimensional barcodes along with reader device capabilities to pack more information in the same space, first by using the second dimension for data encoding, and then encoding more bits per symbol by using color to encode data. In this section we are interested in the general properties of two-dimensional barcodes that are often used for detectability, error resilience, and information packing.

Barcodes usually consist of basic building blocks that make them machine readable. Blocks that are often encountered with different types of barcodes are visible from figure 3.3. The basic data measurement unit of a barcode is module. It defines the smallest discernible data unit from which the barcode is constructed. Modules may have different shapes, of which dot, triangle, square, and hexagonal patterns are the most often used ones. Modules are then used to form codewords that are divided to metadata, data, and error detection and correction codewords. These may then be encoded in different ways depending on their location in the barcode area and the general data content. In addition to data encoding areas, barcodes also have patterns to help with the code location and alignment. All these blocks with rules for their formation make the defined barcode format, usually called symbology.

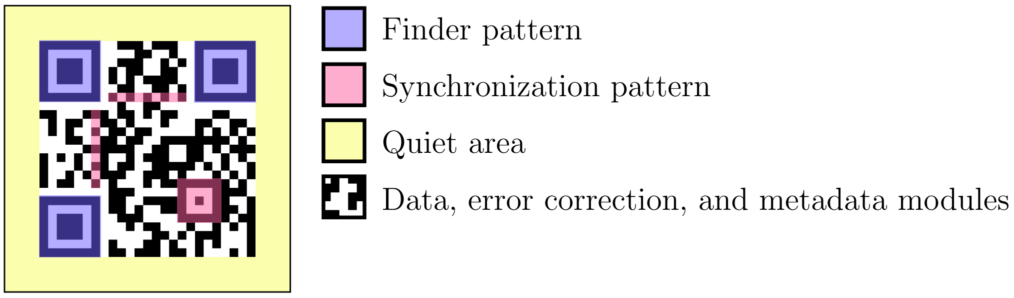


Figure 3.3: Typical areas of a two-dimensional barcode.

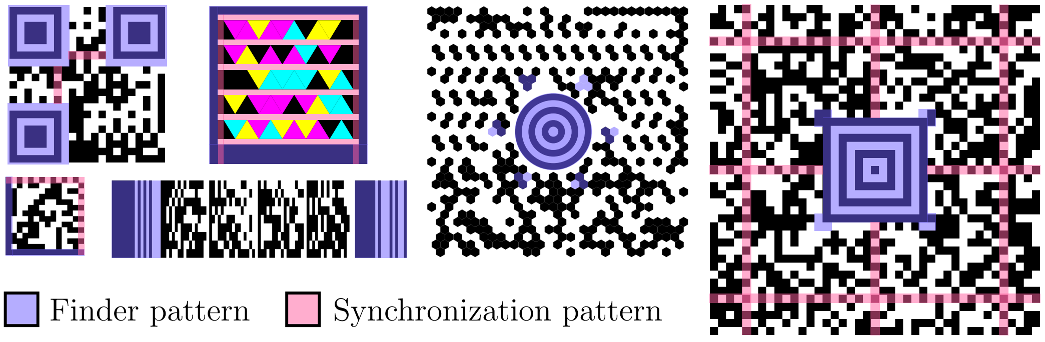


Figure 3.4: Different functional areas illustrated in different barcode types that aid in code area recognition and processing.

Fast machine readability is often the design goal for barcodes in situations where there are real-time constraints for barcode processing. To help with this, barcodes often include certain functional components to make their detection and data decoding easier and more robust. Three often encountered types of such areas are synchronization patterns, the quiet area, and finder patterns[73–78], shown in figure 3.3. Synchronization patterns often consist of timing patterns that are alternating light and dark colored modules, and sometimes there are predefined module patterns in certain locations that help in distortion detection. Figure 3.4 shows examples of such areas in various commonly encountered barcode symbologies. The quiet area is an evenly colored, often white, area around the visible barcode that helps to differentiate the barcode from the surrounding environment. Finder patterns are used to help with barcode location and orientation, as barcode search can be reduced to search for the finder pattern. Larger barcodes may include synchronization

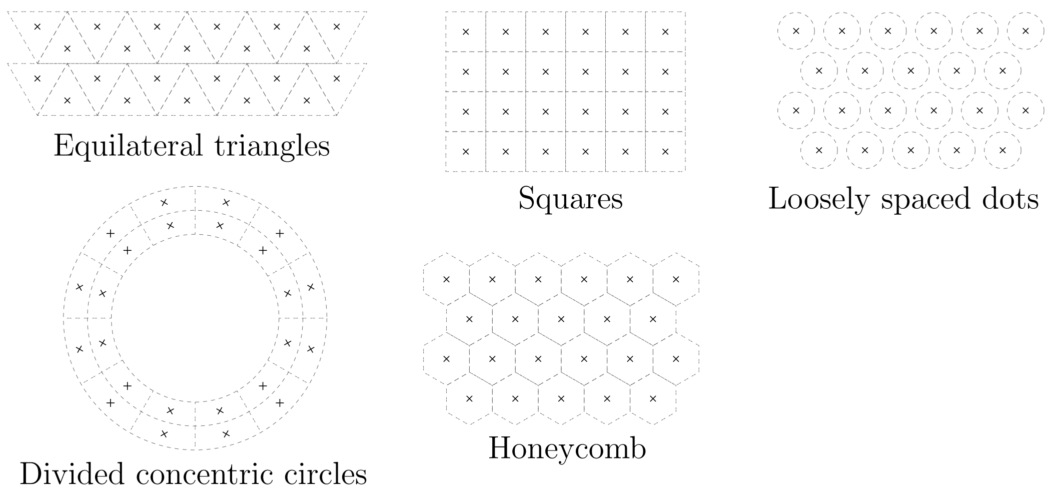


Figure 3.5: General module layout styles used in different barcode types. Cross ($\times$) indicates the module center and dashed line shows the extent of the module area.

patterns to help with small location errors and distortions that occur when the underlying surface has slightly changing geometrical properties[79, 80]. Additionally, synchronization patterns provide information about the data encoding module size over the whole barcode area. And when the barcode gets too big, some standards provide possibility to divide the larger barcode into multiple smaller ones and this way avoid the need to read the whole large code area at once[73, 74].

Besides being machine readable, barcodes often have a goal to encode the information as densely as possible over the small area that is reserved for them. This enables barcodes to have much higher information encoding density than using text and optical character recognition based decoders[58]. For example, experiments with information encoding on microfilms revealed that a checkerboard pattern based matrix type binary barcode[81] was able to encode over 7 times more information over the same area than UUEncoded[82] binary data that was read by an optical character recognition engine[59].

High information storage density is generally achieved by packing constant-sized data modules on a two-dimensional grid. Such barcodes are often called matrix type barcodes. An often used alternative construction strategy is to stack multiple one-dimensional barcodes on top of each other, thus creating stacked two-dimensional barcode[76, 83] (figure 3.2). These stacked barcodes often have a lower areal density than matrix type barcodes with the same printing accuracy, as the stacked barcodes have redundant data in the stacking direction.

There are also a few grid types that are used for laying out barcode modules. The most often used ones are visible from figure 3.5. The often encountered rectangular shape regular grid uses square modules[73, 74, 77]. This enables displaying such barcodes on display devices with similar grids, like on screens of desktop computers and handheld devices. Additionally, creating and reading such a grid will map quite easily to the familiar two-dimensional image processing world, as row and column start points lie on a straight line. Another alternatives that are also used are triangular[78] and hexagonal grids[75], possibly with disconnected circular module shape to enable high-speed barcode printing[84]. Encoding modules in a polar-coordinate fashion around the center of a circle has also been used to make the barcode more robust against rotation[85].

The actual data is then divided to larger codewords that represent integer numbers. Codewords are then encoded by using modules that the barcode format provides. The format of modules for the encoded data often has a few design goals that it tries to fulfill. There often is interest to avoid large areas of the same color to make the module grid more visible, even though synchronization patterns provide clues about the grid arrangement. One way to achieve this is by changing the module pattern in a controlled way so that the appearance probability of the undesired pattern is low. Another way is to use line coding principles of data transmission methods and many one-dimensional barcodes, where more modules are used for the data encoding than is strictly required[71]. These same module value changing principles that are used to avoid large areas of the same color can also be used to avoid areas that look like functional components of the barcode, such as finder patterns[73].

As barcodes are likely to be damaged, or the scanner may read module values incorrectly, there are a few error resiliency methods in place to minimize the probability for unsuccessful or incorrect reads. First, the modules encoding a single codeword are often arranged in such a pattern that a single small stain damages as few codewords as possible. This often leads to an arrangement pattern where codeword modules have roughly equal horizontal and vertical dimensions. Then when codewords eventually get damaged, this damage is detected and corrected by using an error correction methods, like BCH or Reed-Solomon codes[73, 74, 76, 77]. Another possibility is to use checksums to indicate erroneous reads and try again until a successful read is achieved. However, this is not used for two-dimensional barcodes as the primary error correction method, as they do not use redundant data encoding as the primary method for error resiliency.

3.3 Method of choice

Although there are multiple methods that can be used to encode data into images, most of them are not suitable for highly varying image sizes and low resolutions. Even though glyphs and text are human readable data encoding methods, they need a relatively high resolution for each encoded element. Glyphs also have the problem that automatically recognizing them depends on specific shapes that are used for the data encoding. This will result in a quite glyph specific and probably complex detector. Text recognition by optical character recognition is quite researched problem, although creating an optical character recognition engine is generally far from simple[53]. Therefore it is better to focus on codes that are meant for machines to read and only provide human readable codes as additional information.

From machine readable codes, image watermarking mostly includes tricks to make watermarks invisible to the naked eye. Watermarks also suffer from being easily damaged when the image resolution changes drastically. Watermarking methods often divide the encoded information over the whole image area and consequently lose the spatial information that is useful for layout verification. As barcode is used as the data carrier in some image watermarking schemes[61], this strongly suggests that the method that can be used as part of verifying video communication systems should be based on a barcode style information encoding method. This is the method of choice in this study.

Chapter 4

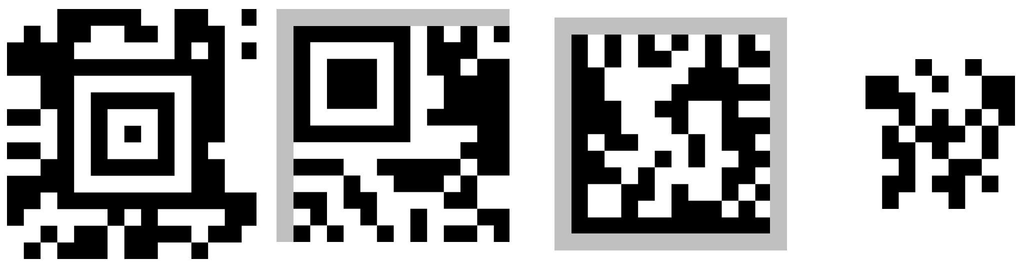
Implementation

This chapter describes the structure of the custom data encoding barcode and the implementation of its detector. Section 4.1 introduces motivation for custom barcode design and the structure of the barcode that is optimized for verifying video communication applications. Section 4.2 takes a quick look at the literature about barcode detection. Then section 4.3 explains the design of the detector for this custom barcode format.

4.1 Barcode design

Barcode design is generally a compromise between the barcode size, ease of detection, data encoding capacity, and damage resistance. We should first check if existing general use barcode standards are suitable for our use.

Some existing publicly available low-capacity general purpose small barcode standards are visible from figure 4.1. These are compared to the custom barcode that is described in the upcoming sections. The chosen existing barcode standards were selected for their ability to encode information in a small space and for the reason that their structure is based on a square grid. These barcode images also show features that are often used in other barcode standards. Compact Aztec code[77] (figure 4.1(a)) uses a finder pattern that is at the center of the barcode and data encoding modules are placed around it. Micro QR code[73] (figure4.1(b)) is a typical barcode that has a distinct finder pattern at a corner of the barcode area. These barcode types usually have the finder pattern in more than one corner for error resiliency. Finally, the Data matrix[74] (figure 4.1(c)) provides an example of a barcode where the actual data is inside the barcode area and has a finder pattern that surrounds the data area. These types of barcodes make it possible to search for rectangular objects from the image and have quite well defined boundaries. The downside



(a) 15×15 (b) 13×13 (14×14) (c) 12×12 (14×14) (d) 9×9

Figure 4.1: The size of the area that different barcode standards take when having the same module size. Additional 1 module wide quiet area is marked as gray, if required (in parenthesis). Each barcode encodes around 20 bits of data. (a) Compact Aztec code, (b) Micro QR code, (c) Data matrix, (d) the barcode introduced in this section.

of these types of barcodes is the fact that they often need a small quiet area around the barcode contents so that the finder pattern stands out from the background.

Section 3.1 gave us some requirements and boundaries for the encodable data amount, resolutions, and the data transfer capacity of a video stream. We established that about 20 bits would be enough to distinguish streams from each other. The size of the area that 20 bits requires for each barcode standard is shown in figure 4.1. We can further see from figures 4.2(a)–4.2(d) how these barcode images look like when they are downscaled to 24×24 pixel area. This corresponds to the smallest realistic subpicture size described in section 3.1. In addition to just scaling, figures 4.2(e)–4.2(h) show how these barcodes look like when they are compressed with 20 % quality JPEG compression (see appendix A). Here we can see how the compact Aztec code and Micro QR code are noticeably suffering from the small module size. Especially with the compact Aztec code in figure 4.2(e) there are module locations that dark enough to be considered black modules, whereas in the uncompressed figure 4.2(a) there clearly are white modules in those areas.

An example how barcodes can be used to test a video communication system is shown in figure 4.3. Here we have a 13-participant conference with high and low resolution versions. We might, for example, want to verify that the layout for such a conference stays the same for each video resolution. Figure 4.3(c) then shows how barcodes would look like for such a high compression and low resolution image. Here the smallest subimages are 24 pixels high with the same compression ratio as in figure 4.2. As the module size of the barcode is relatively large even for the smallest subimages, we can

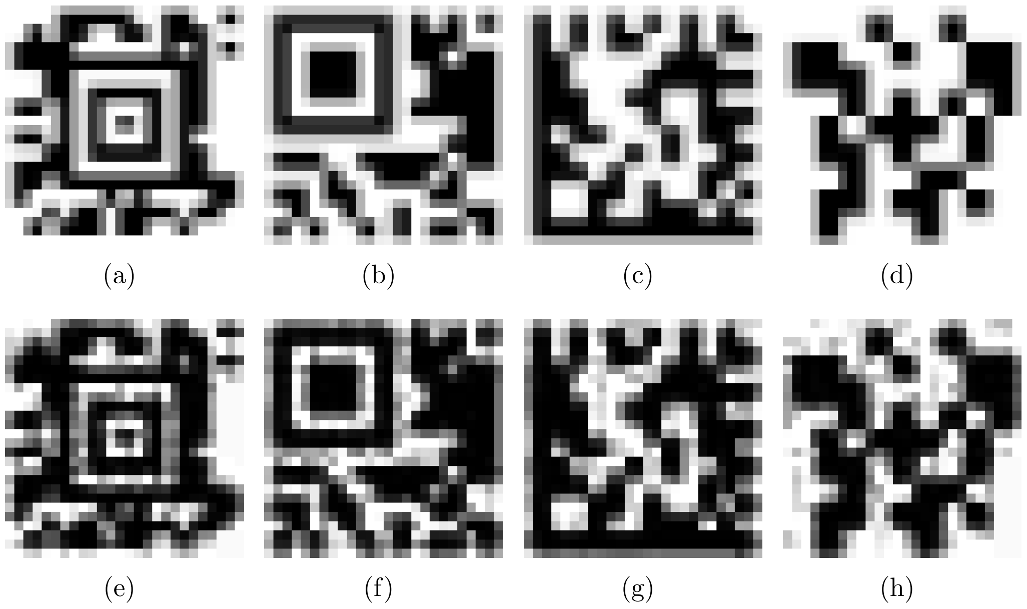


Figure 4.2: A small barcode where (a)–(d) each barcode area is scaled to fit 24 × 24 pixels with a small offset and then (e)–(h) compressed with JPEG quality set at 20%. Module sizes in pixels (a) 1.6, (b) 1.8, (c) 2.0, and (d) 2.7.

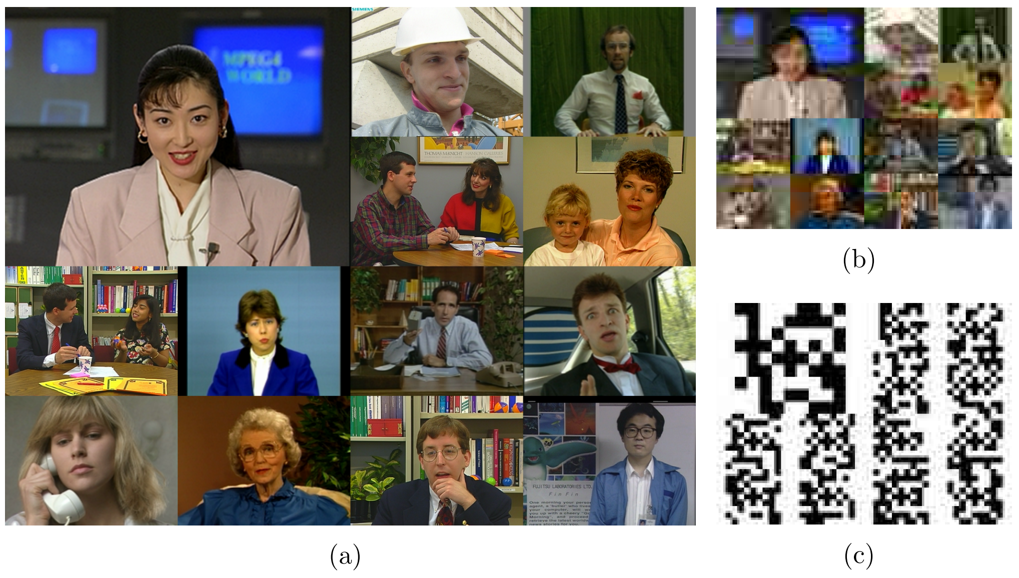
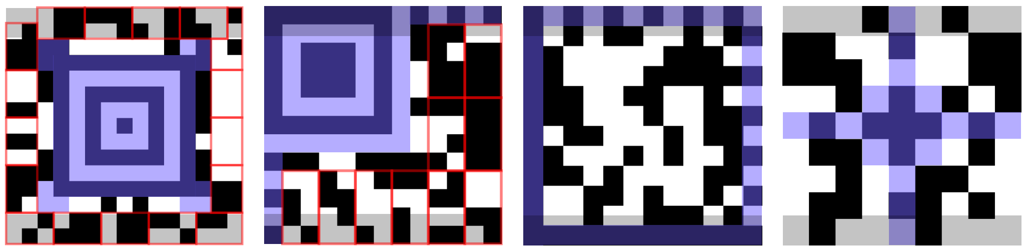


Figure 4.3: A 13-participant conference as (a) 1440 × 1080 pixel high quality version, (b) highly compressed 128 × 96 pixel low quality mobile version, (c) highly compressed 128 × 96 pixel low quality mobile version with encoded data for automatic layout determination.



(a) Compact Aztec code (b) Micro QR code (c) Data matrix (d) Presented barcode design

Figure 4.4: Damage caused to codewords (red) and the finder pattern (blue) by cropping 4:3 image to fit 16:9 screen (gray area). (a) Finder pattern is fully visible, but 12/17 = 70.6% of codewords are damaged. (b) 6/10 = 60.0% of codewords are damaged and partial damage to the finder pattern. (c) No codeword damage, but over half of the finder pattern is destroyed. (d) Depending on the chosen codeword layout and size (figure 4.7), 4 – 8/15 = 26.7 – 53.3% of codewords are damaged and partial finder pattern damage.

be somewhat confident that this could survive even higher compression ratios than the one in figure 4.3(b).

Cropping is an image manipulation operation that cuts out sides of the image. This is usually used to make the image fit a different aspect ratio screen without causing black borders. Figure 4.4 shows a situation where we crop often used 4:3 aspect ratio image to fit 16:9 aspect ratio screen. This cuts out 25 % of the image and causes a large amount of damage to barcodes that use the whole vertical area to maximize the visible module size. Both Aztec code and Micro QR code lose a large enough number of data encoding codewords that their error correction is unable to correct the damage. Data matrix does not really suffer any codeword damage but the finder pattern loses all the horizontal features and makes it impossible to detect the barcode if the detector primarily relies on the finder pattern shape. The suggested barcode shape loses only part of the finder pattern. The codeword damage depends on the codeword size and on the placement of codewords, that will be discussed in section 4.1.3. Occluding elements on the sides of the picture can similarly hide a large area of the Micro QR code's finder pattern.

Damage resistance against cropping and occlusion can be achieved simply by ensuring that there is enough empty space around the barcode and that the barcode includes enough data so that the relative damaged bit count stays low enough. This, on the other hand, leads to a smaller module size, and therefore to a lower resistance against lossy image compression and higher demands for the minimum resolution with the barcode is still recognizable.

4.1.1 Modules

Module is the basic building block that defines the shape and information encoding capabilities of a barcode. Usually the possible module shape and color choice depends on the available printing equipment, surface, scanning device, and scanning environment. We know that the common video encoding standards have a rectangular grid for samples forming the picture. This grid usually has an equal horizontal and vertical difference between image sample points. However, due to different image scaling strategies between resolutions, we can not always assume that this is the case. Anyhow, this gives us a strong indication that we should use rectangular modules to make them match the shape of the image grid.

Then we have the question how many different values a single module may have, as this affects the data encoding density of the barcode. For a general use barcode, black and white make it more easy to take into account varying lighting conditions and enables the usage of image binarization methods in detection. As we do not need to take such conditions into account, in the ideal case we could just use all the different values that image channels provide to encode data. On the other hand, this means that module values would have no tolerance for small value changes. We can expect those small deviations, as color space conversion (section 2.4.1), and transform coding and quantization (section 2.4.3) both change the colors in such a way that the actual image sample value can only be known to a certain precision.

In a typical video communication system we are dealing with 3-channel 8-bit R′G′B′ images that get converted to $\mathrm{Y'C_BC_R}$ color space when we create video out of them. In this case $\mathrm{Y'C_BC_R}$ color space has 220 values for luma channel and 225 values for both chroma channels[34, 35]. Then transform coefficient quantization (appendix A) will change the color value even further, especially when the module size gets smaller. Therefore, if we want to ensure that the module value can be reliably read even with higher compression ratios, we should use black and white modules. This enables encoding 1 bit data per module and thus leads to simpler detector design.

By using binary valued barcode modules we mainly are concentrating on the luma channel. But what if we would also use the color information encoding chroma channels for data? From section 2.4.1 we can see that a wrong color space conversion results in slightly wrong color values. Let us see what happens when all color channels have an equal value, namely $R' = G' = B' = a$, where R' is the value of the red color channel, G' is the value of green color channel and B' is the value of the blue color channel. Both ITU-R BT.601[34] and ITU-R BT.709[35] specify the value of the luma channel Y' and chroma channels C'_R and C'_B with following equations:

$$Y' = k_R R' + k_G G' + k_B B', \qquad C'_R = R' - Y', \qquad C'_B = B' - Y'$$
$$\text{ITU-R BT.601:} \qquad k_R^{601} = 0.299, \quad k_G^{601} = 0.587, \quad k_B^{601} = 0.114$$
$$\text{ITU-R BT.709:} \qquad k_R^{709} = 0.2126, \quad k_G^{709} = 0.7152, \quad k_B^{709} = 0.0722$$

where $k_R + k_G + k_B = 1$. Therefore, the two major standards differ only in multipliers used for converting color channels into the luma channel. For grayscale source image, where every channel has the same value, $Y' = k_B a + k_G a + k_B a = a$, $C'_R = a - a = 0$, and $C'_B = a - a = 0$. This means that the luma channel directly corresponds to the grayscale value of the image pixel independent of the used color space conversion method. And consequently, chroma channels will be easily compressible over the barcode area, as they only have the same constant value of 0.

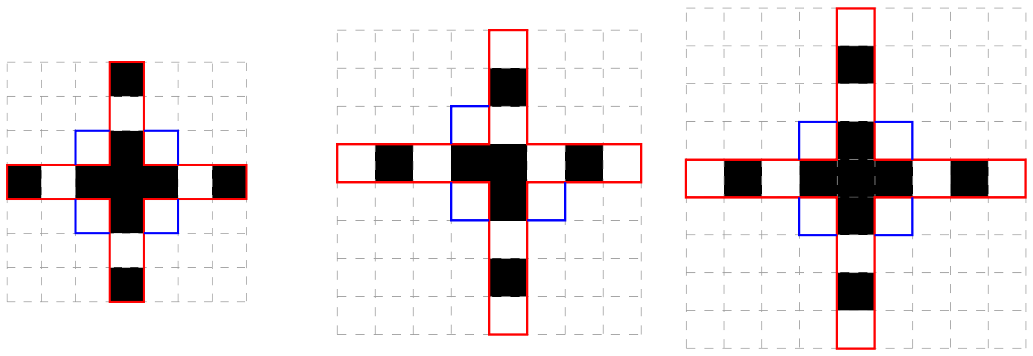
Section 2.4.2 describes the chroma subsampling and how the color information is not as important as the details for the human vision. Consequently, the standard JPEG quantization matrix for chroma channels (table A.1) has higher quantization factors than the corresponding example matrix for the luma channel. So if chroma channels are used to encode information, the information density of those channels will be smaller than for the luma channel. Also, if we do not have access to the original luma and chroma channels, and our color space conversion assumption is wrong, this may also result in a slightly wrong value for all color channels in the R′G′B′ color space.

Even though using more than 2 color values per module, or using chroma channels, could provide us with higher information encoding density in the ideal situation, distortions resulting from the video encoding and decoding process make purely black and white modules less ambiguous.

4.1.2 Finder pattern

Finder pattern has the role to make it easier to find out barcode location and orientation. The bigger and the more unique finder pattern is, the more easy it is to search for. Consequently a bigger finder pattern takes up area that could otherwise be used to encode data. Also, the location of the finder pattern relative to the rest of the barcode area matters. If the barcode is partially damaged, then it would be desirable that the finder pattern is still recognizable after encountering damage.

Some of the existing finder pattern shapes and their properties are described in section 3.2.4. The interesting properties for a finder pattern are: distinctiveness, immunity to rotation, and standing out from the background. When we want to use the barcode in video communication applications we



(a) 7×7, 32 bits (b) 8×8, 46 bits (c) 9×9, 60 bits

Figure 4.5: Finder pattern and barcode size alternatives. Red outline modules are meant to be used for finder pattern detection and blue outline modules prevent additional crossing finder pattern detection structures from forming near the center.

can ignore the rotational independence property, as video image is not usually rotated. This way we can get more usable area for the actual data.

The usual image distortions in video communication applications, namely scaling, cropping, and occlusion, are described in section 2.3. The most important information, like the face of the participant, is usually found at the center of the video frame. This way we can expect cropping to destroy information on sides of the video frame and we can also expect the smaller occluding pictures are to be placed at the sides of the main picture. This leads to a finder pattern design where the finder pattern lies in the middle of the barcode area.

By taking ideas from the finder pattern structure of the QR code[73], properties of Reed-Solomon error correction code[86], and some experimentation, the finder pattern structure is a pattern visible in figure 4.5(c). This results in a finder pattern whose center can be detected by searching in the horizontal and vertical direction. The finder pattern also does not need an additional quiet zone around the barcode, as it can be separated from the rest of the barcode and has enough unique parts that can be detected by themselves. Figure 4.6 shows the possible divisions of the axis-oriented finder pattern component that gives a known ratio of black and white modules that may be used to detect partial finder patterns. Partial finder patterns, however, may result in spurious finder pattern detections that are discussed in more detail in section 4.3.7.

Figures 4.5(a) and 4.5(b) show some alternative smaller barcode and finder pattern designs. Using a smaller barcode size would provide more robustness

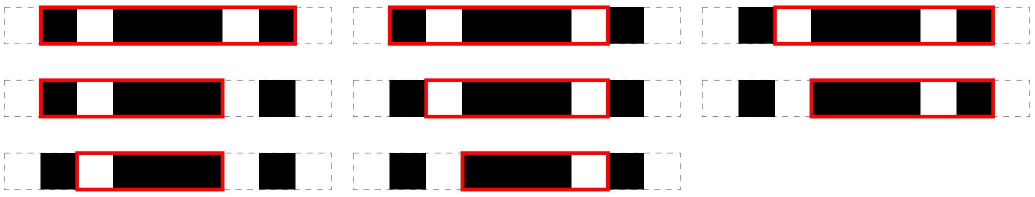


Figure 4.6: Unique divisions that include the center of the finder pattern based only on the edge ratio information.

against scaling, as the barcode area would take less space with the same module size. Consequently, a smaller barcode would allow modules to be larger over the same image area and therefore easier to distinguish. On the other hand, the finder pattern takes relatively more space over the barcode area and this way leads to fewer modules that can be used for error resilience. Also there are fewer alternatives to divide the finder pattern into unique shapes. This makes it harder to enable the detector to detect partially damaged finder patterns. The symmetry of the finder pattern location also plays a role when we try to make the relative damage that the barcode area encounters independent of the damage direction.

4.1.3 Barcode area and codeword placement

Multiple modules are grouped into codewords that form the basic addressable element of the barcode. This enables block based error correction that is used for damage resistance[73–77]. Codeword locations, conversion of the raw data into codewords, division between data and error correction codewords, and the order in which codewords are encoded depends on the use cases that the barcode standard tries to take into account.

The codeword based data encoding approach is somewhat sensitive to damage, as damage to one module makes the whole codeword damaged. Therefore we want to minimize the codeword size so that the damage to one module only covers as small area as possible. We can expect that the damage comes from cropping and overlaid elements. Cropping destroys large horizontal and vertical edge areas, whereas overlaid elements focus their damage over small areas. Figure 4.7 shows some possible codeword placement strategies around the barcode area depending on the damage that the barcode is expected to encounter. We can use the layout shown in figure 4.7(a) if we expect to encounter damage from cropping, but do not know from which direction the image is cropped. Figure 4.7(b) shows a situation where we can increase the horizontal cropping resistance with the risk of losing almost all codewords if sides of the barcode are cropped away.

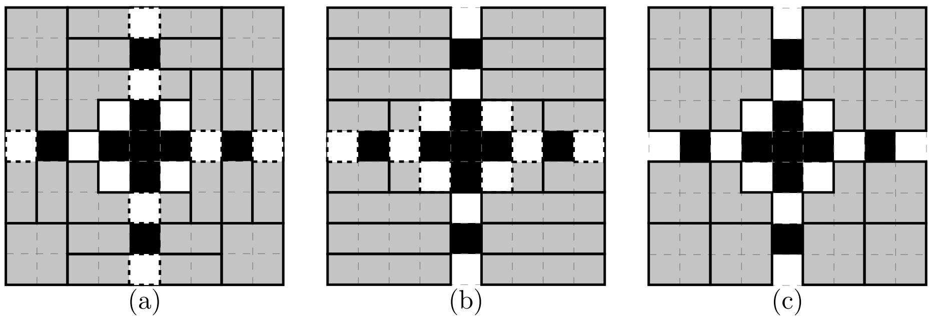


Figure 4.7: Alternative codeword placement strategies. (a) 4-bit code words with somewhat equal horizontal and vertical cropping resistance. (b) 4-bit codewords with higher horizontal cropping resistance. (c) Mixed 3-bit and 4-bit codewords for blob damage resistance.

These cropping resistant codeword placements, however, can lead to a large amount of codeword damage if the area of damage covers many codeword boundaries. Figure 4.7(c) shows a codeword encoding strategy that is often employed in general use barcodes where the damage to the barcode area is assumed to consist of horizontally and vertically roughly equally sized blobs. This can be a useful codeword placement strategy if picture-in-picture functionality (section 2.3) is used as part of the video output.

4.2 Existing detectors

Most barcode readers are part of commercial applications and therefore the exact algorithms that they use are trade secrets. However we can look at a few open source implementations and research on barcode detection to get some clues how existing barcode readers generally work.

Barcode detection can generally be divided to five distinct steps: pre-processing, feature extraction, regions of interest and code area location, code segmentation, and decoding. Pre-processing the barcode image converts the image data to a format that makes it possible to apply feature extraction. Feature extraction step provides the barcode detector the features that it can use to locate candidate barcode areas. Then the barcode area candidates are searched for by using the extracted features. When a barcode area is located, comes the step of locating different barcode regions, like the data encoding modules and their values. And finally, when the different module values are known, they need to be read in the order that the barcode symbology specifies, apply the possible error correction, and decode the data keywords that result from this.

Different pre-processing methods have the goal of making the original image more favorable for the feature extraction phase. Often barcode processing with black and white modules relies on grayscale image conversion, and possibly thresholding, of the original color image[79, 87–91]. Thresholding can be either global or locally adaptive, depending on if we take possibly varying image brightness into account[78, 87, 88]. This then results in only one image channel that can be used for feature extraction or additional pre-processing. More expensive pre-processing steps may include applying different filtering operations, like morphological operations[92, 93] or noise reduction[79, 93], and blur detection and correction. Blurring can come either from shaking or out-of-focus images[91, 94]. To reduce the computational complexity of the pre-processing, these steps may also be applied later in detection process if the potential barcode area can first be located by some other method[92].

The feature extraction phase provides the barcode detector features that it can use for the barcode area location. Features related to specific barcode component shapes include edges[87], line segments and their crossings[87], corners[95], and features arising from various Hough transforms[78, 89, 94]. Texture direction analysis[87, 89, 96] can be used for barcode area location that then enables the detection of multiple barcode type candidates without having to search for barcode specific features over the whole image. These features can then form larger feature groups that provide even stronger indicators of the barcode area, like the three similar finder patterns in the QR code[91].

The extracted features are then selected based on some criteria to detect valid barcode areas and achieve low false positive count while doing this. Depending on the selected features, this selection is done by having a measure for feature goodness and selecting features that have the largest threshold crossing value for such measure[87]. This measure may come directly from feature values or be combined by some function. These combining functions can be, for example, decision functions in machine learning methods[96].

The barcode area, its size, and orientation can be determined when patterns that make the barcode area are located. These patterns are found out by searching for known functional areas of the barcode[79, 88, 90], or by relying on the quiet area around the barcode, defining the edges of the barcode accurately enough[79, 87]. The detected barcode area often suffers from perspective projection distortion due to camera angle. This can be corrected by using inverse perspective transform that relies on the barcode area curvature being close to a plane[79, 88]. The barcode may also suffer from uneven surface that can result, for example, from bending due to being on a cylindrical surface, or due to wrinkles on the surface. The uneven surface can be overcome, for example, by fitting a deformable model on the estimated barcode area[94].

When the barcode area is determined and its distortions are taken into account, we can read individual module values that result in the final barcode value. There unfortunately is little information about reading the module value in the literature. Some focus is, however, given for determining the module position[78, 97]. By examining some freely available 2D barcode decoder libraries we can see that ZBar[98] uses the value of the thresholded pixel at the estimated module center as the module value. Libqrdecode[99], on the other hand, takes the average value of the thresholded pixel values over the estimated module area and checks if the resulting average is closer to 0 or 1.

4.3 Detector design

Detector design aims to take into consideration the finder pattern structure, known image distortions, and the situations that can cause spurious barcode area detections. Algorithm 1 shows the general steps that the detector performs. These steps are explained in more detail in following subsections. We can also take a look at the assumptions that this barcode detector does about the barcode distortion model and how it takes it into account.

1. Detect strong horizontal and vertical edges. (4.3.2)
2. Detect the finder pattern based on the detected edge information. (4.3.3)
3. Merge horizontal and vertical pattern detections. (4.3.4)
4. Find pattern group intersections. (4.3.5)
5. Create estimated barcode areas based on pattern group intersections. (4.3.6)
6. Clean up likely invalid barcode matches. (4.3.7)
   (a) Small intersection score.
   (b) Narrow area.
   (c) Overlapping barcode areas.
   (d) Invalid finder pattern.
7. Read barcode module values. (4.3.8)

Algorithm 1: General steps for barcode detection with this detector and the sections where they are described.

We do not need many of the advanced and often computationally expensive methods that are used in many barcode detectors to overcome the features of the environment. We also want to keep this detector fast so that it is possible to analyze the video image in real-time. This is about 20–30 frames per second, including the time it takes to decode the video. The smaller the barcode size we can detect is, the smaller video image size we can use in testing and this way either execute tests faster, or possibly run more tests simultaneously.

Distortions that our detector does not have to take into account are random noise, lens related distortions, blurring, uneven lighting, and rotation. We mostly need to take into account the large scale differences that occur with output streams including many participants. We may also be interested in how the video encoder distorts the image, but this is mostly for finding out the limits of the detector.

4.3.1 Distortion model

The theoretical barcode model has a certain module size and clear edges between modules. When this barcode is shown as an image, we only have single samples of the barcode in the image. Additionally, image size changes lead to resampling the image by either reducing or increasing the resolution of the source image. However, we know that our barcode can only have two different module values. This way we can create a model that enables us to do sub-pixel accurate barcode positioning. This leads to a more accurate module value reading when the module size is small.

When creating the initial image from the barcode model or reducing the image resolution, the image distortion model that we assume is an image averaging operation. The resulting target image value $F^{down}_{x,y}$ is created by taking the mean of the area in the source image that is closest to the corresponding source image point (x', y'):

$$F^{down}_{x,y} = \frac{1}{ab} \int_{x'-a/2}^{x'+a/2} \int_{y'-b/2}^{y'+b/2} \hat{f}_{\hat{x},\hat{y}} \, d\hat{x} \, d\hat{y}.$$

Here a is the width, b is the height of the sample pixel area in the source image, and $\hat{f}_{\hat{x},\hat{y}}$ is the value of the sample point in the source image. Sample point value determination varies depending on if we are reading from the barcode model or if we are resizing the image. When sampling the barcode model, $\hat{f}_{\hat{x},\hat{y}}$ is the binary module value at sample point $(\hat{x}, \hat{y})$. When we are downsampling an existing image we assume that coordinates that are not at an exact existing image sample point value $\hat{f}_{\hat{x},\hat{y}}$ will have the value of the nearest known sample point.

We assume that the image resolution increase is done by using bilinear interpolation[23]. In bilinear interpolation the interpolated sample value is

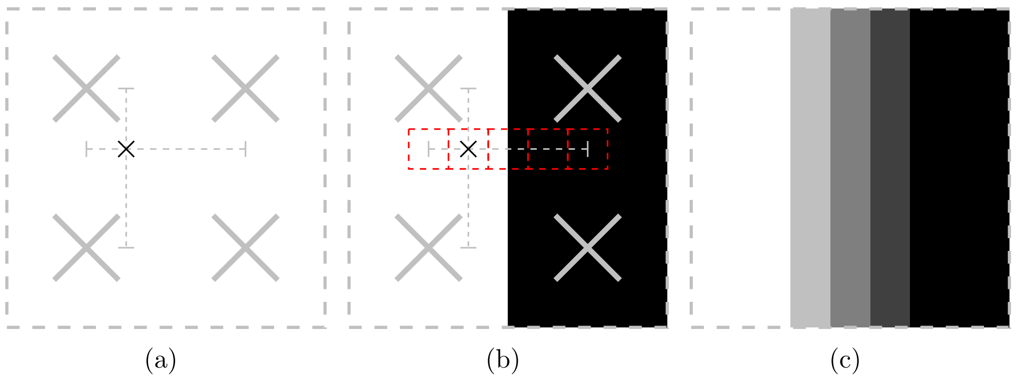


Figure 4.8: (a) Image sample points (gray crosses) and a new sample point (black cross) with its location compared to original image samples. (b) New sample locations when scaling the image by four and a nearest-neighbor representation of the black-white edge. (c) The final nearest-neighbor representation of the linearly interpolated image.

calculated by taking the weighed value of the 4 nearest image sample points. We can see how this works by assuming that the 4 nearest sample values reside in coordinates $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$. Then the bilinearly interpolated value $F^{up}_{\hat{x},\hat{y}}$ at $(\hat{x}, \hat{y})$ where $\hat{x}, \hat{y} \in [0, 1]$ comes from following equation:

$$F^{up}_{\hat{x},\hat{y}} = (1 - \hat{x})(1 - \hat{y})\hat{f}_{0,0} + \hat{x}(1 - \hat{y})\hat{f}_{1,0} + (1 - \hat{x})\hat{y}\hat{f}_{0,1} + \hat{x}\hat{y}\hat{f}_{1,1}.$$

We can see that $\hat{x}$ and $\hat{y}$ work as weights that depend on the horizontal and vertical distance from the nearest known original image sample points.

As our detector is based on detecting the image edges, we can examine how bilinear interpolation acts near edges. Figure 4.8 shows what happens when sample points that form a vertical edge are interpolated with 4-time resolution increase. This creates a staircase edge between the areas and is perceived as a blur. We can also see that when $\hat{f}_{0,0} = \hat{f}_{0,1}$ and $\hat{f}_{1,0} = \hat{f}_{1,1}$, the interpolated values tell the relative position between the original sample points that form the edge, as:

$$\hat{F}^{up}_{\hat{x},\hat{y}} = \hat{f}_{0,0}(1 - \hat{x}) + \hat{f}_{1,0}\hat{x}.$$

This works similarly for horizontal edges. We can see that if we are just concerned with horizontal or vertical edges between modules, we can get the interpolated sample point position relative to the original sample points forming the edge just by looking at its value.

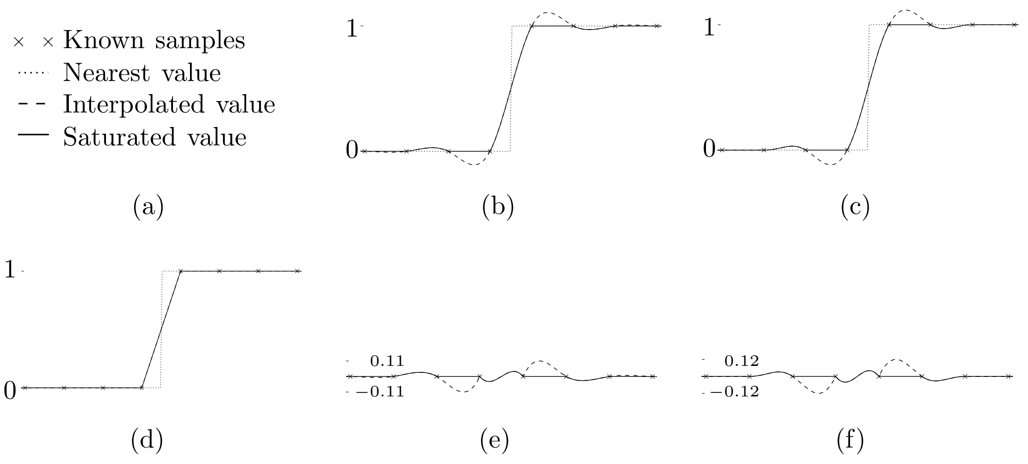


Figure 4.9: One-dimensional interpolation results for often used image interpolation filters on a discretely sampled step edge. Saturated values indicate the maximum and minimum values that an interpolated value can get in the final image. Interpolation results for (b) cubic interpolation, (c) Lanczos3 interpolation, and (d) linear interpolation. The difference between linear interpolation, (e) cubic interpolation, (f) and Lanczos3 interpolation.

Bilinear interpolation is not the only method available for image resizing. We can do a quick estimate for its applicability by looking how much it differs from other often used image interpolation methods. The interpolation situation that we are interested in is a step edge that goes from the maximum to the minimum value, or the other way around. Figure 4.9 shows how values from different often used interpolation methods compare when the interpolation is applied in one dimension. We can see that the cubic[23] and Lanczos3[100] interpolation methods result in high bumps at the sample edges. These result in ringing artifacts in grayscale images. In our case these artifacts will be cut away due to value range saturation in the image. Figures 4.9(e) and 4.9(f) show the resulting difference between linear interpolation, and cubic and Lanczos3 interpolation methods. As the difference for saturated values is around 4% at most, we can be quite sure that at least these methods will not cause a large error in comparison to our assumption that the image is resized by using bilinear interpolation.

4.3.2 Edge detection

The change between black and white modules forms a step edge. This step edge has a property of going from the maximum image intensity value to the minimum, and the other way around. We use this assumption to form the basic features that this detector uses for the finder pattern detection.

We can first look at how edge detection is generally done. Edge detection is often divided into three parts: image smoothing, image differentiation, and edge labeling[101]. Image smoothing removes noise from the image that results in spurious edge detections. Image differentiation brings out features that produce edges. Labeling involves edge localization and false edge suppression. In our case we are looking just for purely horizontal or vertical step edges. This makes it possible to ignore directional features of two-dimensional edge detectors. We can also ignore the smoothing step, as the goal of smoothing is to reduce noise, but at the same time it eliminates small features that look like noise. This is detrimental when we have a small module size that we want to detect in the barcode. We can also rely on large edge intensity changes, as black and white module values go from the image minimum value to the maximum, and the other way around.

As this barcode detector is designed for testing a video communication system, we can ignore barcode rotation. This way we can use one-dimensional differentiation on rows and columns to search for edge matches from axis-oriented sample sequences. This results in rows and columns that are independent sample sequences on which we can apply the edge detector without worrying about the detection direction. We can define an edge e_k to be a monotonically increasing or decreasing sequence of samples y^k where $y_n^k < y_{n-1}^k$ or $y_n^k > y_{n-1}^k$. We can do this by assuming that edge smoothing resulting from image upscaling described in the previous section will not result in two subsequent samples having the same value. This is likely the case, as we usually have an 8-bit image channel that has most of the intensity values available and image resizing will result at most in 10–20 times resolution increase.

The monotonically increasing sample sequence forms a staircase edge. We are, however, expecting a step edge, and therefore need to determine its location from the staircase edge. We can use finite difference $\Delta_n^k = y_n^k - y_{n-1}^k$ between samples to calculate the location of the step edge. First, we add virtual samples to the end and to the beginning of the monotonically rising or falling sample sequence to make sure that the staircase edge reaches the minimum and the maximum image values. Then these finite differences are normalized so that they form weights $w_n^k = (\Delta_n^k)/(\sum_j \Delta_j^k)$ such that $\sum_j w_j^k = 1$. These weights are used to calculate the step edge location $C(e_k)$:

$$C(e_k) = \sum_{j=p-1}^{q+1} w_j^k (s_{j-1}^k + \frac{s_j^k - s_{j-1}^k}{2}),$$

where s_j^k is the location of sample value y_j^k. p is the location of the first finite difference Δ_p^k of edge samples and q is the location of the last one and $p \leq q$. This calculates the mass center of finite differences.

In addition to calculating the location of an edge, edge intensity $I(e_k)$ provides us information about the edge direction (rising or falling) and about the intensity of the edge. The edge intensity can be used for false edge suppression, as we know that the edges that interest us theoretically go from the maximum to minimum value and the other way around. Edge intensity value is calculated simply by calculating the sum of finite differences, or equivalently the difference between the first and the last samples that belong to the staircase edge:

$$I(e_k) = \sum_{j=p}^{q} \Delta_j^k = y_q - y_{p-1}.$$

4.3.3 Finder pattern detection

When we have the information about edges, we can start using it for the finder pattern detection. We can use the intensity of an edge $I(e_k)$ and the estimated step edge location $C(e_k)$ to find out where the finder pattern, or a part of it, is located. This is done by searching for edge location sequences with specific ratios in edge location differences. We know that edge intensities change from the minimum to the maximum value, or close to it. This makes it possible to ignore false edges that are created by compression noise or by areas that do not belong to the barcode. This enables us to focus on probable barcode module edges. At the same time we want to be able to recognize small barcode finder patterns where edge intensities do not always go from the minimum to the maximum value and therefore we want to have some slack in edge intensities that we accept. This leads to accepting only the edges that have the absolute edge intensity $|I(e_k)|$ at least as high as the minimum edge intensity threshold τ_I.

The general process for finding out if edges that we have in the current sample sequence form a pattern match is outlined in algorithm 2. This algorithm goes through partial finder patterns listed in figure 4.6 and checks which sequences of strong edge intensities match with the current partial finder pattern under inspection. The strong edge list in the sample sequence is iterated in the rising order of edge locations. This filters out all edges whose absolute intensity does not reach the edge intensity threshold τ_I. This threshold is to make sure that we do not falsely register edges that come from compression artifacts and to ensure that small variations in the image

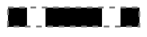
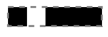
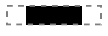
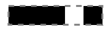
```
$l \leftarrow$ the amount of consecutive finder pattern edge differences;
forall the edges $e_k$ in a sample sequence, where $|I(e_k)| \geq \tau_I$ do
    Check that the last $l$ edge intensity directions match with pattern's directions;
    Check that the relative ratio of the last $l$ edge location differences matches with the pattern up to error $\epsilon_M$;
    if all checks succeed then
        Create a match object;
    end
end
```

Algorithm 2: General steps for pattern matching based on the location of strong step edges e_k for a pattern with $l + 1$ alternating black and white module groups.

intensity do not form interesting edge matches. Then when we have enough strong edges at our disposal, we can start comparing them with the current partial finder pattern under inspection. Table 4.1 shows partial finder patterns and their ratios that we use in our detector to establish a potential finder pattern match. As they mostly are partial finder pattern matches, they also enable the detection of partially damaged finder patterns. We can create normalized edge difference lists from these patterns by dividing each module color ratio by the sum of these ratios.

The first comparison using the partial finder pattern under inspection is to check if all edge intensity directions match. The strong edge matches in the image may quite possibly have two consecutive edges going to the same direction even though a pattern consists of black and white modules. This may arise when image intensity changes slowly from one extreme to another and has multiple consecutive weak edges between two strong edges.

Table 4.1: A list of partial finder patterns and their corresponding scores that are searched from the image to establish a potential barcode match.

Pattern	Module color ratios	Score	Description
	1:1:3:1:1	4	Full pattern
	1:1:3	1	Partial left/top pattern
	1:3:1	1	Partial center pattern
	3:1:1	1	Partial right/bottom pattern

When edge directions under inspection match with the ones that the partial finder pattern has, we can continue by checking that each normalized edge location difference Δ^{edges} matches closely enough to the corresponding normalized ratio $\Delta^{pattern}$ in the current partial finder pattern under inspection. Edge location difference is the difference between the location of two consecutive strong edges. We then normalize these differences by summing up the last l edge center differences and dividing each edge difference with this sum. Here l is the edge difference count for the current partial finder pattern candidate under inspection. This is not the only way that could be used to make ratios of the last l edge differences and partial finder pattern module color ratios comparable. We could, for example, normalize against the widest edge difference and this way get larger normalized edge location differences out when the finder pattern module is partially cropped or occluded.

The closeness of the normalized edge location differences is determined by checking if all normalized edge differences Δ_i^{edges} are relatively close enough to the corresponding normalized ratios $\Delta_i^{pattern}$ for the current pattern under inspection. Relative closeness is calculated by:

$$\frac{|\Delta_i^{edges} - \Delta_i^{pattern}|}{\Delta_i^{pattern}} \leq \epsilon_M,$$

where pattern match error limit ϵ_M is a constant for selecting how big pattern match errors are allowed. If all edge location differences pass this test, then current edge differences are used to generate a line segment that corresponds to the current partial finder pattern.

The line segment resulting from the current partial finder pattern match is perpendicular to the current sample sequence that is used for pattern match generation. Sample sequences generated from rows result in vertical line segments and sample sequences generated from columns result in horizontal line segments. This line segment of the partial finder pattern match indicates the assumed center of the finder pattern. The location and length of an individual match line segment is determined by the location of the sample sequence and the difference between the first and the last edges that are included in this partial finder pattern match. The center location of the actual finder pattern depends on which partial finder pattern we are currently trying to search for, but is still calculated by using the outermost edge locations of the current partial finder pattern match.

The resulting partial finder pattern match line segment is generated by using the module size estimate of the current matched partial finder pattern. The module size is estimated to be the difference of the first and the last edge location divided by the amount of edges in the current partial finder pattern match. The module size estimate then determines the match line segment length simply by multiplying the average module size with the module width factor δ^{module}, that in our case usually is 1.

The location of the line segment resulting from the partial finder pattern match is determined in such a way that it indicates the center of the full finder pattern. In the ideal case this results in multiple line segments that are on top of each other. Also, as each sample sequence results in an individual line segment, the amount of these line segments in the ideal case corresponds to the module size in pixels multiplied by the amount of partial finder patterns. The next section describes how these line segments resulting from partial finder pattern matches are then combined together.

The lengths of these line segments give an estimate for the module size. This, however, is wrong in a sense that matches result in line segments perpendicular to the current sequence direction. If we have modules with aspect ratio that is not equal in the vertical and the horizontal direction, the line segments that are supposed to intersect with each other will not have emphasis in the correct direction. This can lead to a situation, especially with small module sizes, where we detect the partial finder patterns correctly, but line segments resulting from them will not intersect. We can overcome this by increasing the module width factor δ^{module}. This can, however, lead to false finder pattern detections more easily. Another way to overcome this is to figure out the aspect ratio of modules in the barcode and weight the horizontal and vertical line segment lengths differently.

4.3.4 Match merging

The pattern detection phase results in a large amount of line segments that indicate where the detected finder pattern center is. Most of these resulting line segments form larger clusters that come from detecting the same finder pattern multiple times. If we then use each segment intersection to form a potential barcode match, we will have $\mathcal{O}(mn)$ potential barcode matches for modules that are n pixels high and m pixels wide. In the ideal case every match on the same pattern results in an intersection at the same point, so in this case we would have mn barcode candidates from segment intersections.

We can reduce the resulting intersection count by merging the vertical and horizontal match segments $\boldsymbol{l}_i^{match}$ separately and create new segments $\boldsymbol{l}_j^{merged}$ that maintain the approximate location and extent of the finder pattern match. Let us see how this works by looking at vertical line segments. Horizontal segments have the same operations but just with flipped coordinates.

First, assign some slack $\epsilon_i^{pattern}$ to each match segment $\boldsymbol{l}_i^{match}$ in the horizontal direction, when the horizontal segment location is at $x_i^{segment}$.

Segment area $A_i^{segment}$ is a rectangle that includes all points that are between the vertical endpoints of the line segment and horizontally $\epsilon_i^{pattern}$ away from the horizontal location $x_i^{segment}$ of the segment.

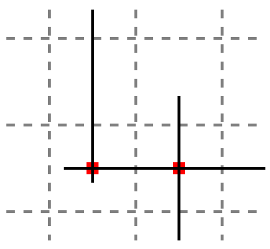
When we have formed areas $A_i^{merged} = A_i^{segment}$ from all vertical segments, then we go through all intersecting areas $A_j^{merged} \cap A_i^{merged} \neq \emptyset$ and remove the intersecting areas A_i^{merged} and A_j^{merged} and create a new rectangular area A_k^{merged} where the horizontal start and end coordinates of that area are the maximum and the minimum of horizontal coordinates of areas A_i^{merged} and A_j^{merged}, and vertical coordinates are similarly the maximum and the minimum of A_i^{merged} and A_j^{merged}. And then we repeat this process until we have no overlapping areas left. And repeat the same process with horizontal segments.

As a result, each area A_i^{merged} covers a group of vertical segments. We can then create a new segment $\boldsymbol{l}_i^{merged}$ that has the same vertical endpoints as area A_i^{merged} and the horizontal location is the median of all vertical segments inside the area. This median selection criteria is used to avoid the effect of outliers on the merged segment location. However, merged segment endpoints are still sensitive to outliers.

This merging process avoids the quadratic complexity that we would encounter if we would use the match segments as they are to create barcode area candidates. We can see that this line segment merging operation has $\mathcal{O}(n \log n)$ complexity, as that is the complexity for rectangular area overlap detection, where most of the complexity comes from sorting[102, 103]. By using balanced binary trees as the data structure can handle insertion and deletion with complexity of $\mathcal{O}(\log n)$. Therefore the removal and creation of new areas is cheap from the complexity standpoint. Although all areas could overlap with each other, the removal of one area on each merging iteration leads to at most $n - 1$ overlap removals in total.

We still have not discussed how the match segment slack $\epsilon_i^{pattern}$ is determined, even though it determines the resulting area size. Slack is determined by multiplying the pattern match error limit ϵ_M with the estimated module size. The minimum value for slack is 0.5. This is to ensure that we will get some decent sized rectangular areas $A_i^{segment}$ even when the module size is small (around 2 pixels).

This approach of pattern match line segment merging has its shortcomings as one standalone segment resulting from spurious pattern detections on module edges can merge two strong separate segment groups together. This is because the slack $\epsilon_i^{pattern}$ is relative only to the module size and this way may cover too large an area relative to the actual error when the module size is large. Then, by using median to select one single point to indicate the location of merged segments will return value that is bad at least for one group.



$$\begin{aligned}
&\Delta x_{start} = \mathbf{horizontal}(\boldsymbol{i}_{hv}^{merged}) - \mathbf{start}_x(\boldsymbol{l}_h^{merged})\\
&\Delta x_{end} = \mathbf{end}_x(\boldsymbol{l}_h^{merged}) - \mathbf{horizontal}(\boldsymbol{i}_{hv}^{merged})\\
&\Delta y_{start} = \mathbf{vertical}(\boldsymbol{i}_{hv}^{merged}) - \mathbf{start}_y(\boldsymbol{l}_v^{merged})\\
&\Delta y_{end} = \mathbf{end}_y(\boldsymbol{l}_v^{merged}) - \mathbf{vertical}(\boldsymbol{i}_{hv}^{merged})\\
&\rho_x = \min(\Delta x_{start}, \Delta x_{end}) / \max(\Delta x_{start}, \Delta x_{end})\\
&\rho_y = \min(\Delta y_{start}, \Delta y_{end}) / \max(\Delta y_{start}, \Delta y_{end})\\
&\mathbf{badness}_{hv} = (\min(\rho_x, \rho_y) + \lambda)^{-1}
\end{aligned}$$

(a) (b)

Figure 4.10: (a) An intersection with high badness (left) and low badness (right). (b) Intersection badness calculation.

4.3.5 Merged segment intersection

After we have reduced the amount of intersecting horizontal and vertical segments, we want to find out where the barcode center is. We do this by finding out the intersection point of a merged horizontal segment $\boldsymbol{l}_h^{merged}$ and a vertical segment $\boldsymbol{l}_v^{merged}$. The intersection of line segments can be determined, for example, in similar fashion as rectangle intersection[103]. For n line segments finding out the all intersections can be done with the complexity of $\mathcal{O}(n \log n + k)$, where k is the total number of intersections.

An intersecting horizontal segment $\boldsymbol{l}_h^{merged}$ and vertical segment $\boldsymbol{l}_v^{merged}$ will form an intersection $\boldsymbol{i}_{hv}^{merged}$. The intersection also has a badness measure $\mathbf{badness}_{hv}$ that tells how much the intersection point of the two intersecting merged segments differs from the ideal center point. The reason for intersection badness can be seen from figure 4.10(a) where the upper row results in an unwanted segment intersection. As these intersection points result in potential overlapping barcode matches, we can later use this intersection badness measure to clean up barcode areas that look like they result from accidental line segment intersections. The badness measure $\mathbf{badness}_{hv}$ is calculated with an algorithm presented in figure 4.10(b). Here **horizontal**() and **vertical**() functions refer to the horizontal and vertical locations of the intersection. Functions **start**() and **end**() refer to the start and end points of the line segment they are applied to. λ is some small constant to prevent the badness measure going to infinity.

Intersection has certain properties in addition to the badness measure. It has a list of horizontal and vertical segments where each segment corresponds to a partial horizontal or vertical finder pattern match. Table 4.1 assigns a score to each partial finder pattern and we can use these scores to assign a score to the intersection. The intersection score is the sum of scores of unique partial finder pattern match segments from which the intersection results from. We are interested in the unique partial finder patterns in the intersection, as the amount of segments that form the intersection only tells us how large the module size is.

4.3.6 Barcode area candidates

After we have horizontal and vertical line segment group intersections, we can use this information to refine our estimate of the potential barcode area center. We can also use this combined information to refine the estimated module size and this way potentially get a better barcode area estimate.

The intersection location is first adjusted so that only the horizontal and vertical partial finder pattern segments that include the intersection point between their endpoints are selected for the final intersection point. This adjusted intersection location is calculated by taking the median of horizontal coordinates of vertical segments and similarly with the vertical coordinate. Even though we still could iteratively adjust this location, it would be unnecessary, as the biggest change to the intersection location comes from the elimination of segments resulting from pattern detections that are far away from the unadjusted intersection location.

Next we estimate the barcode area by using module size estimates of the partial finder pattern matches that are included in this barcode area intersection. We assume that the module size remains constant over the barcode area. We can do this for synthetic images, as we do not have to worry about sporadic sampling rate changes or distorted barcode surfaces.

The actual module size is then separately determined for both axes, by taking the median of module widths of all partial finder pattern matches for the current intersection. As we are mostly including partial finder pattern matches, the module size estimate comes from partial finder patterns near the center. This makes the module size estimate for a cropped and damaged barcodes more robust. Modules near the edges are often smaller than modules in the middle, especially in a cropped image.

The assumption of constant-sized modules is violated when the module size is small and a nearest neighbor type image filter is used for resolution reduction. If this filter does not have anti-aliasing properties built in, then it is quite likely that the module size estimate coming from center modules will result in an overly small estimate. This will lead to incorrect positioning of modules that are far away from the barcode center. We could use the mean width of module size estimates and this way nudge the module size estimate into the correct direction. This would, on the other hand, result in a smaller module size estimate for cropped images.

One solution to the incorrect module size and position estimates would be to relax the assumption of the constant module size for smaller sizes. It would be then possible to search for horizontal and vertical edge clusters over the estimated module border locations and use this information to refine the module location information. This, however, would need some measurement for the goodness of fit, a regularization method to prevent everything from collapsing to a single point, and an optimization method to achieve this.

4.3.7 Candidate area elimination

Barcode area candidates can be created by any matches that result in appropriately spaced edge sequences, especially as we are also searching for partial finder patterns. These edge sequences may also arise from user interface graphics or other content on the screen and from high frequency areas in natural images. Also, module values on the barcode area itself include potential for partial finder pattern sequence formation. We could eliminate these module groups that form partial finder patterns by approaches described in the finder pattern design section 4.1.2. However, these methods take away barcode area that could otherwise be used for encoding data. We can instead use other means to ensure that those areas will not easily result in spurious barcode matches.

There are four elimination phases that we use to clean up the potentially invalid barcode matches from the image. In the first phase we use the intersection score, described in the section 4.3.5. In this elimination phase we ignore all intersections that have score less than 4. This corresponds to a case where at most four module rows, or columns, are fully destroyed with maximum of two rows or columns on one side.

The second elimination phase is for spurious pattern detections, usually for areas that occur in natural scenes. Often the detected pattern match results in an overly narrow barcode area that can not occur in images that are stretched in any realistic way. Barcode narrowness can be determined from the aspect ratio of the barcode area:

$$\begin{aligned}\mathbf{aspect}(A^{candidate}) &= \frac{\mathbf{width}(A^{candidate})}{\mathbf{height}(A^{candidate})}\\ \mathbf{narrowness}(A^{candidate}) &= \max(\mathbf{aspect}(A^{candidate}), \mathbf{aspect}(A^{candidate})^{-1}).\end{aligned}$$

To accept the candidate barcode area as a valid one, we require that the narrowness measure is less than some predefined narrowness limit constant

narrowness$(A^{candidate}) < \epsilon_N$. This constant can be determined by application basis, for example by determining that a 11:9 CIF type image will be stretched to fit a 16:9 screen with some extra slack for error.

In the third elimination phase we filter out barcode area candidates that result from the barcode data area by module arrangements that form a full or partial finder pattern sequence. Figure 4.11(a) illustrates the places where such module groups that look like the finder pattern may form. The finder pattern shape, however, prevents the full finder pattern sequence from forming in other locations than the center of the barcode area. This information can be used to eliminate barcode area candidates that result from partial pattern matches inside the barcode area. To find out if two candidate barcode areas, $A_i^{candidate}$ and $A_j^{candidate}$, are from the same barcode, we first check the overlapping area, $A_{ij}^{overlap} = A_i^{candidate} \cap A_j^{candidate}$. Two different barcode area candidates are considered to be overlapping if the relative amount of overlap for both barcode area candidates is over some overlapping threshold $\tau^{overlap}$:

$$\frac{|A_{ij}^{overlap}|}{|A_i^{candidate}|} \geq \tau^{overlap} \quad \text{and} \quad \frac{|A_{ij}^{overlap}|}{|A_j^{candidate}|} \geq \tau^{overlap}.$$

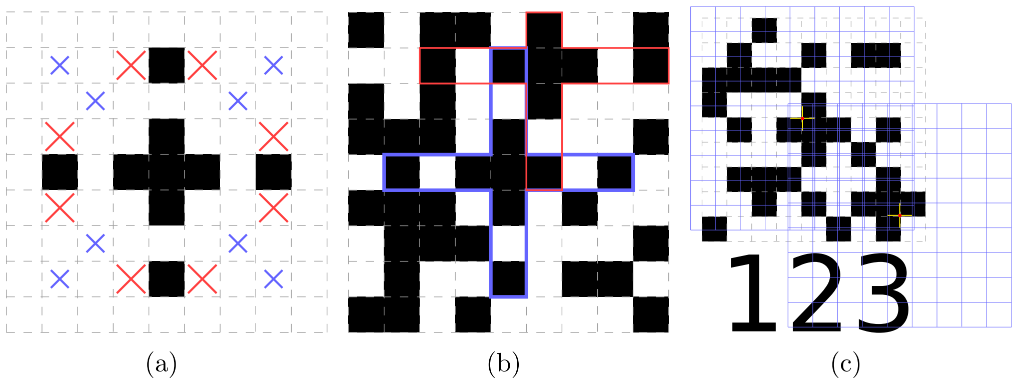


Figure 4.11: (a) Locations where module groups that look like a partial finder pattern may form fully inside the barcode area are marked with big red crosses. Locations where finder pattern like structures may form are marked with small blue crosses if there are appropriate patterns near the barcode area. (b) An example barcode where a partial finder pattern match (red) is detected in addition to the full finder pattern (blue). (c) Smallest overlap scenario for pattern matches (blue grid) resulting from the same barcode.

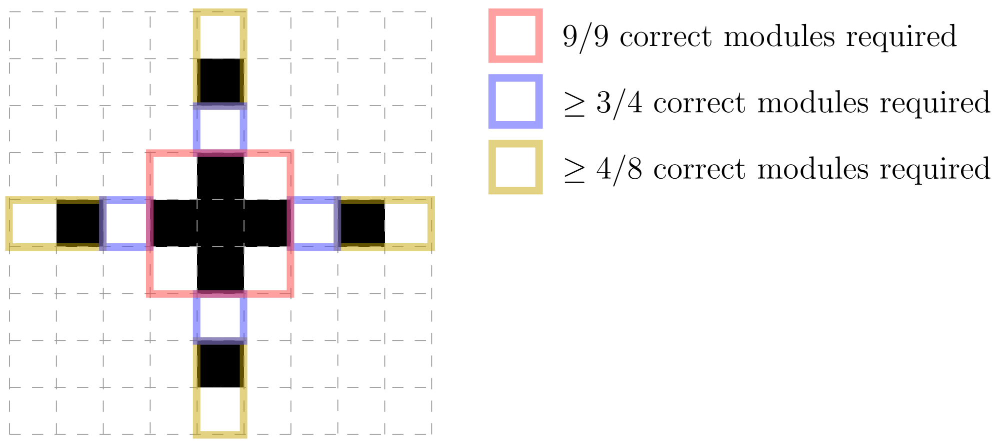

9/9 correct modules required
≥ 3/4 correct modules required
≥ 4/8 correct modules required

Figure 4.12: The division of the finder pattern to different categories and the minimum required modules for each category.

This prevents us from eliminating cases where a small barcode occludes a larger barcode. Overlapping threshold $\tau^{overlap} = (4/9)^2$ is selected so that we can notice if two equally sized barcode areas overlap at least with 4×4 module area. This can be derived from figure 4.11(c) where we have two overlapping barcode areas, where the second one is generated by the help of objects outside the actual barcode area.

When a barcode area overlap is detected, we try to find out if this is due to a partial pattern match. This can be detected by comparing intersection scores against each other and the area candidate with the smaller score is removed. If intersection scores are equal, we can look at the intersection badness to decide which intersection to remove. The intersection with the larger badness score gets removed. We keep both barcode area candidates if badness scores are equal.

The last elimination phase involves reading the actual module values of the finder pattern and checking its integrity. This ensures that we probably have the actual finder pattern. Figure 4.12 shows how different finder pattern modules are divided to three different categories. From each category we check that it has at least the minimum amount of correct modules. This category assignment tries to achieve a situation where we allow sides to be occluded or cut out, but expect the center area of the picture to remain intact.

We have more elimination steps that would be strictly necessary, as we could eliminate most barcode area candidates by checking for the finder pattern integrity. On the other hand we treat module value reading as an expensive operation and want to avoid it. These checks are also ordered in such a way that cheaper checks are executed first and the more expensive ones have less candidate barcode areas to go through.

4.3.8 Value reading

When the barcode location has been determined, then the next part is to read the barcode value. This is done by reading individual module values in some specific order and then determining the actual value of the barcode based on these module values. The value of a single module is determined by reading the mean value $\mu(A_{xy}^{module})$ of the image over the module area:

$$\mu(A_{xy}^{module}) = \frac{1}{hw} \int_{y-h/2}^{y+h/2} \int_{x-w/2}^{x+w/2} f(\hat{x}, \hat{y}) \, d\hat{x} \, d\hat{y},$$

where x is the horizontal center, y is the vertical center, h is the height and w is the width of the module, and $f(\hat{x}, \hat{y})$ is the intensity of the image pixel that is closest to the point (x, y). This enables us to read the module mean by weighting the edge and corner pixels by the area they cover and reading the pixel values that are fully inside the module area as they are.

The module value is then determined by what the module mean is. The allowed module values are white, black, or unknown. Unknown value results when the mean value over the module area is exactly at the image intensity midpoint or if the module area inside the image is less than 1/4 of the expected module area. Module areas outside the image can result from cropping the original image and can mark quite large module groups as unknown. A simple extension to the image intensity midpoint decision is a threshold that would mark the module value unknown if the module area mean does not differ enough from the midpoint intensity. This is, however, detrimental when module side length is around 2 pixels, as the effect of edges on the area mean gets quite high. This usually leads to a situation where clearly black modules that are surrounded by white modules, and the other way around, will be marked as unknown.

Marking the module value as unknown, however, is useful when we want to make the value decoding more resilient to errors. For example, Reed-Solomon error correction method[104] that is used in many barcode standards[73–77] can correct more errors if error locations are known[86]. This is helpful when a relatively large amount of modules gets cropped from the barcode image. Knowing that the barcode has a large amount of unknown modules can help us to decide if the area is really a barcode or just some noise that happens to have a similar structure as the finder pattern.

We could improve this module value reading method to detect unknown module values by using different module value calculation criteria for small module sizes than what is used for large modules. With small modules (side length around 2 pixels) we need to take into account the fact that pixel values near borders have quite a large impact compared to the whole module area. With larger module sizes we will have a situation where occlusion comes into play and the impact of edges over the module area diminishes. For example, picture-in-picture functionality shown in figure 2.1 will result in a small barcode covering a part of a larger one.

Another possible module value determination method for small modules would be to use weighted mean that gives higher weights to the pixels that are closer to the center of the module. We could also take the edge model into account instead of treating edge pixels as having just a single color value, they could be divided into black and white parts according to the estimated edge positions. This would also provide us with a prior value for the module and reading the actual pixel values would give us some level of confidence about the module being at that prior value.

Chapter 5

Evaluation

This chapter determines the limits of the information encoding and decoding method described in the previous chapter. The evaluation setup is first described in section 5.1. Then the test cases in section 5.2 focus on finding out how well this detector can detect the barcode from images that have a barcode and are distorted by scaling and image compression. Section 5.3 then tests how well this detector handles situations where there can also be other data besides the barcode in the image.

5.1 Evaluation setup

The different test cases have their own specific characteristics, but the test data creation and some detector parameters stay the same. Barcode images were created by using the bilinear interpolation model described in section 4.3.1 that creates image samples from the ideal barcode model. The resulting image samples are rounded to the nearest integer, rounding down if the sample value is exactly at the midpoint of two different integer values. The resulting image is an 8-bit image that provides us with 256 different intensity values. Image values are in an equally spaced scale of 0–1 in the processing phase. This enables us to express detector parameters in range of 0–1 instead of making them bit-depth dependent.

In the actual test cases we mainly change the edge intensity threshold τ_I and the pattern match limit ϵ_M (section 4.3.3), as these parameters make the largest difference on the detector performance. Values of these parameters are varied in a grid search fashion over the largest realistic value range for each parameter. The edge intensity threshold τ_I gets values 0.5, 0.6, 0.7, 0.8, and 0.9. Pattern match limit ϵ_M gets values of 1 %, 2 %, 5 %, 10 %, 15 %, and 20 %. The other selectable detector parameters stay constant during the test and only the test data changes. Pattern narrowness limit ϵ_N (section 4.3.7) has value that can stretch 16:9 image onto 9:16 with 10% error, or the other way around. The module width factor δ^{module} that determines the length of the line segment resulting from a successful pattern match (section 4.3.3) has value of 1.

Barcode readout is accepted as successful if the all barcode module values match with the expected ones. No effort was made to see how many of the invalid detections could be fixed by using error correction, as its results would depend on the codeword placement and size (section 4.1.3).

5.2 Detection limits

Finding out the detection limits of this detector provides us information about how small module size we can reliably put through the video communication system at different compression levels. First we try to find out the minimal module size in the ideal situation that only applies the distortion model described in section 4.3.1 and integer quantization of resulting values. Then the second test case applies varying levels of lossy image compression to the barcode image and tries to find out the minimal barcode size and decoder parameters for each compression level. The last test case related to detection limits looks how this detector behaves when the barcode image is heavily upscaled with different interpolation filters.

5.2.1 The minimal module size

Determining the smallest module size that is detectable by the detector in the optimal case provides us knowledge about how small barcodes we can find before we apply any image compression. The test case here is to create differently sized versions of the same barcode with module size of 1.5–3.0 pixels changing in steps of 0.1 pixels. Each barcode is displaced by 0.0–0.9 pixels in 0.1 pixel intervals in the horizontal and in the vertical direction. This displacement is to ensure that barcode modules will encounter similar distortions near module edges. Each displacement and module size has 500 randomly generated barcode values.

Table 5.1 shows what is the minimum module size for each selected parameter pair. All detector parameter variations result in the same minimum barcode module size and therefore table 5.2 shows what happens when the module size goes 0.1 pixels smaller than the module size that has 100 % detection rate with the test data. This shows more variation in detector

Table 5.1: The smallest module size and the strictest parameter values that lead to a successful detection of all barcode module values with different displacements.

$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9
1 %	1.9	1.9	1.9	1.9	1.9
2 %	1.9	1.9	1.9	1.9	1.9
5 %	1.9	1.9	1.9	1.9	1.9
10 %	1.9	1.9	1.9	1.9	1.9
15 %	1.9	1.9	1.9	1.9	1.9
20 %	1.9	1.9	1.9	1.9	1.9

Table 5.2: Successful detection rate for 0.1 pixels smaller module sizes than what is needed for 100% successful detection rate. Rounded down to the nearest 0.1 percentage point.

$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9
1 %	81.4 %	81.4 %	81.4 %	81.4 %	49.0 %
2 %	81.4 %	81.4 %	81.4 %	81.4 %	49.0 %
5 %	89.2 %	89.2 %	89.2 %	89.2 %	49.0 %
10 %	97.9 %	97.9 %	97.9 %	96.5 %	52.2 %
15 %	99.6 %	99.6 %	99.6 %	99.0 %	55.5 %
20 %	99.9 %	99.9 %	99.9 %	99.8 %	56.9 %

performance with relation to parameter values, where less strict detector parameters provide higher recognition rate.

These results can be explained by the fact that the detector is comparing only one row or column based sample sequence at a time. This leads to ignoring the neighboring sequences and this way results in an invalid edge position estimate when the edge is on sample points lying on corners between black and white modules. When the module size is at least 2.0 pixels, we will always have a sample sequence that results in a correct edge placement, as a module will always fully cover at least one sample point area. Module size smaller than 2.0 pixels will therefore result in edge location estimate error that goes over the pattern match limit ϵ_M. This leads to noticeably increasing detection rate for 1.8 pixel modules in table 5.2 when the maximum allowed pattern error is increased. Effects of the minimum edge intensity threshold τ_I are visible only with the most strict values, as factors affecting the edge intensity affect also its estimated location. The overlapping candidate area elimination (section 4.3.7) also causes failures in the barcode detection. This is because locations in the barcode area where a partial finder pattern may form (figure 4.11(a)) can result in finder patterns with higher intersection score than a partial detection of the actual finder pattern has.

5.2.2 Image compression

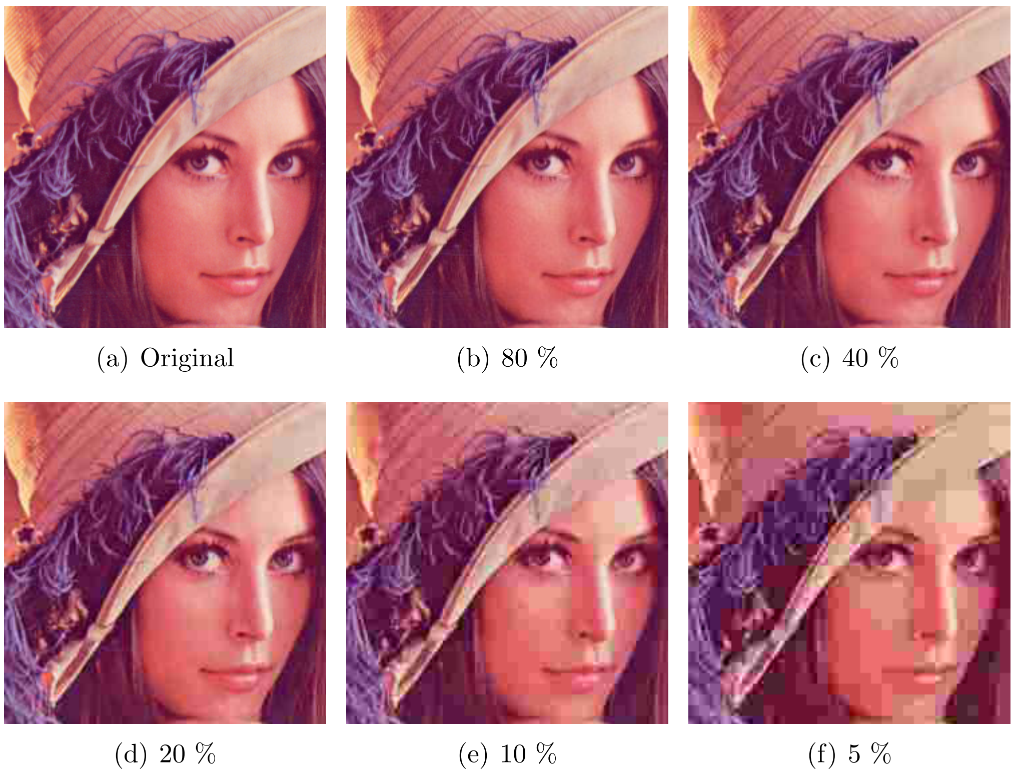
Image compression adds distortions that we do not take into account. This test case evaluates how varying compression amount changes the minimal fully recognized module size. The compression method used in this test is the JPEG image compression[105] described in appendix A. JPEG compression applies similar lossy compression method and data reduction operations as video encoders do, but with less complexity and less parameters to select. JPEG compression makes these test cases more easy to repeat, as it does not have that many dependencies on a certain encoder, encoder version, and on the variation of encoder parameters. We only have one parameter to vary, the JPEG quality level. In this test case we use JPEG quality levels 5 %, 10 %, 20 %, 40 %, and 80 %. The result of compressing a natural image by different quality levels is visible from figure 5.1.

Test images have module size of 2.0, 2.5, 3.0, 4.0, and 8.0 pixels. For each module size we randomly generate 10000 barcode images with random displacement of 0.0–8.0 pixels. This displacement is to make sure that the barcode will not always be at the same position relative to the JPEG block boundary. Then each image is compressed by the used JPEG library and fed to the barcode detector.

Table 5.3 shows the most strict parameters with the smallest module size where all the 10000 images were successfully recognized. The most strict parameter value pair was determined by selecting the largest minimum edge intensity threshold τ_I and then the smallest corresponding pattern match limit ϵ_M. We can see that compression levels that result in high quality natural

Table 5.3: The smallest module size and the strictest parameter values that lead to the successful detection of all lossily compressed test images for each quality level.

Quality	Minimum module size	Edge intensity	Location error
5 %	4.0	0.6	20 %
10 %	3.5	0.7	20 %
20 %	2.5	0.5	20 %
40 %	2.0	0.6	20 %
80 %	2.0	0.8	10 %



(a) Original (b) 80 % (c) 40 %

(d) 20 % (e) 10 % (f) 5 %

Figure 5.1: A comparison how a natural image looks like with the same compression levels that are used in the evaluation of quality setting influence on the detector.

images, like 40 % and 80 %, do not need a large module size for successful detection. Images with lower quality levels result in a larger module size and also require less strict detector parameters. This is due to the fact that when we want to find out the smallest detectable module size, then naturally the less strict detectors are able to detect smaller module sizes. However, when the module size gets larger, then parameters can also be less strict, as the influence of a single pixel gets smaller and image compression causes relatively smaller errors for the edge localization. There can also be spurious candidate barcode area eliminations, as in the test case determining the smallest barcode module size.

5.2.3 Heavily upscaled images

We are also interested in the behavior of this detector when the barcode image is heavily upscaled. Section 4.3.1 describes some often used image interpolation

Table 5.4: The amount of invalid detections for different interpolation methods. Total sample size is 10000 randomly generated source images with a scaling factor of 15.

	(a) Bilinear					(b) Bicubic					(c) Lanczos3				
$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9	0.5	0.6	0.7	0.8	0.9	0.5	0.6	0.7	0.8	0.9
1 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20 %	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1

methods and their properties when they are applied to step edges. The goal of this test case is to see if heavy two-dimensional interpolation affects the detector performance.

The test data for this test case is 10000 barcodes that originally have 8 pixel module size with random displacement of 0.0–8.0 pixels. This image is then separately upscaled by the factor of 15 by using bilinear[23], bicubic[23], and Lanczos3[100] interpolation. This results in a barcode that is 1080×1080 pixels in size.

Table 5.4 shows how many invalid detections there are for each scaling method with different parameters. We can see that there always is one unsuccessful detection for the largest allowed pattern match limit ϵ_M in bilinear interpolation. These invalid detections result from spurious finder pattern detections in the blurred edge areas. This leads to merging of different finder pattern areas into one, as show in figure 5.2. In this figure pattern matches related to real partial finder pattern detections are shown in red and invalid finder pattern detections resulting from the blurred edges are shown as yellow bars in figure 5.2(a). Blue color shows how large merge area the 20% pattern match limit results in. When blue areas overlap, they make the different line segments to merge with each other, as explained in section 4.3.4. This will make spurious pattern matches to act as a bridge between the two red finder pattern match groups and makes these groups merge with each other.

The median pattern match segment merging method will then pull the resulting final pattern match segment to the invalid upper pattern match, shown as a green bar in figure 5.2(b). The red cross in the same figure indicates the center of the correct finder pattern.

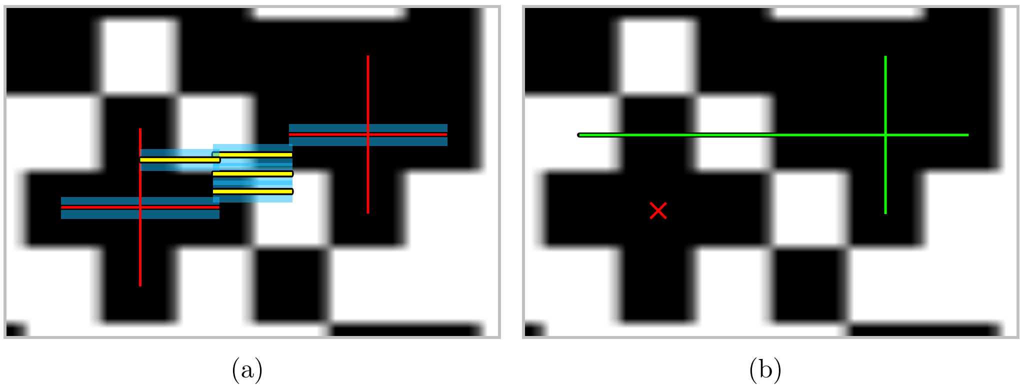


Figure 5.2: Interpolated image with (a) segments resulting from different pattern matches, and (b) merged line segments. Red color shows pattern match segments closest to the real pattern center. Yellow indicates spurious matches and matches that are not placed at the real pattern center. Blue shows the area where pattern match line segments get merged.

This situation where few outliers can cause detection failure on otherwise clear finder pattern match groups could be mitigated by making the relative area that a single pattern match covers relatively smaller as the module size increases. This would be in line with the fact that single image pixels are also smaller in relation to the barcode area. Currently the detector is mostly designed for small barcode module sizes where there is no strong blurring and therefore a smaller chance for spurious finder pattern matches to arise.

5.3 Testing for false positives

False positive testing concentrates on finding out how robust this detector is when the video image is something that we can encounter in a normal video communication situation. This enables us to see if we can use this detector to include the barcode image as a marker alongside with normal video data. The two test cases for false positives are meant to determine how well this detector works with video streams that could be captured by a camera and with images that result from data sharing.

5.3.1 Natural video sequences

Natural video sequences provide us a way to see if the detector returns false positive matches for situations that may occur in face-to-face video conferencing situations. The used test sequences are listed in appendix B and they include material with mostly low-frequency components corresponding to the situation for which most video encoders and their parameters are optimized for. The test setup is the same as the test setup for detection limits in section 5.2 where the edge intensity threshold τ_I and the pattern match error limit ϵ_M are varied between their realistic minimum and maximum values.

Table 5.5(a) shows how many barcode candidate patterns we have detected for each parameter pair. As we can expect, the potential for erroneous barcode detections rises sharply when the maximum allowed edge location error is increased. Also, the edge intensity threshold clearly affects the potential false positive rate, but to a lesser extent. Table 5.5(b) then shows how many of those potential detections result in an actual false positive pattern detection after candidate finder patterns are eliminated by using the elimination phases described in section 4.3.7. We can see that we have 1 false positive detection that does not repeat with higher allowed edge errors. This false positive detection gets eliminated by the elimination phase that removes overlapping barcode area candidates as there are more potential barcode candidates in the same image that partially cover each other.

Table 5.6 shows how many potential barcode areas each elimination phase removes. These elimination phases are applied in order and naturally phases that are applied first remove more invalid matches than phases that are applied later. We can see from table 5.6(a) that the intersection score elimination phase eliminates most of the spurious matches. Intersection score elimination phase, however, has a property where the higher required minimum edge intensity criteria with higher allowed pattern match error limits (15 % and 20 %) make the intersection score elimination method to work less well. However, the narrowness elimination phase takes up most of what the intersection

Table 5.5: Total candidate barcode areas in 36131 natural images and spurious pattern detections resulting from these areas.

	(a) Barcode candidate count.					(b) Spurious detection count.				
$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9	0.5	0.6	0.7	0.8	0.9
1 %	1	0	0	0	0	0	0	0	0	0
2 %	50	45	27	7	0	0	0	0	0	0
5 %	1327	1084	556	113	16	0	0	0	0	0
10 %	16459	13029	5989	1882	229	0	0	0	0	0
15 %	57433	45054	22910	7300	2384	**1**	0	0	0	0
20 %	124600	94251	52291	19654	6823	0	0	0	0	0

Table 5.6: Relative elimination rates in natural video samples for each elimination phase. Elimination phases are applied in the order listed and one sample is eliminated at most only once.

	(a) Intersection score.					(b) Narrowness.				
$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9	0.5	0.6	0.7	0.8	0.9
1 %	100.0	-	-	-	-	0.0	-	-	-	-
2 %	100.0	100.0	100.0	100.0	-	0.0	0.0	0.0	0.0	-
5 %	100.0	100.0	100.0	100.0	100.0	0.0	0.0	0.0	0.0	0.0
10 %	99.7	99.7	99.9	99.9	99.6	0.0	0.0	0.0	0.1	0.0
15 %	96.9	96.8	94.8	88.7	71.6	1.4	1.9	4.2	9.4	21.1
20 %	92.4	92.0	88.7	79.6	64.5	2.5	3.7	7.2	14.9	24.8

	(c) Overlap.					(d) Finder pattern.				
$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9	0.5	0.6	0.7	0.8	0.9
1 %	0.0	-	-	-	-	0.0	-	-	-	-
2 %	0.0	0.0	0.0	0.0	-	0.0	0.0	0.0	0.0	-
5 %	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10 %	0.0	0.0	0.0	0.0	0.0	0.3	0.3	0.1	0.0	0.4
15 %	0.0	0.0	0.0	0.0	0.0	1.7	1.2	0.9	2.0	7.3
20 %	0.2	0.1	0.0	0.0	0.9	4.9	4.2	4.1	5.5	9.9

score elimination phase lets through. This strongly suggests that when the minimum required edge intensity is set higher, the accepted edges matches create patterns that look more like the full finder pattern, but are narrower. The higher required edge intensity makes the accepted edge matches to be further away from each other and this way it quite naturally leads to a situation where there are more narrow barcode areas with higher edge intensity. Wider pattern matches result in a larger potential area where horizontal and vertical segment intersections can occur, as described in section 4.3.4. This leads to a situation where one large horizontal segment match can easily cover small vertical segment matches and result in a large and narrow barcode area match.

Table 5.6(c) shows that the area overlap check elimination phase is mainly applied when larger allowed errors result in more potential barcode area candidates. And as table 5.6(d) shows, most barcode matches that pass the intersection and narrowness elimination phases are eliminated by the finder pattern validity elimination phase. Although it now catches all but one invalid finder pattern matches, it still lets one through that could be eliminated by making the module value reading give errors if the average module value is not far enough from the intensity midpoint for larger module sizes.

5.3.2 Data sharing

Data sharing is a use case scenario where video communication participants view commonly shared data that usually is computer generated. This data may come from screen sharing, from shared work area, or from other data sources, like lecture slides. The general screen sharing situation usually displays application windows and some text and images among them. The shared work area enables the creation of content in writing and drawing between participants. Other use cases for screen sharing could be, for example, a remote lecture where remote participants may view the lecture slides and the lecturer at the same time. As this detector uses strong edges as the basis for detection, these data sharing cases should provide favorable ground for making this detector to detect false positives.

Data sharing use cases have quite a large variation in the content that they can present, even though the content usually is computer generated. This computer generated content includes application windows, varying sized text with different scripts, diagrams, and other artificially generated drawings and graphical elements. Data sharing cases may also include natural images as part of the video data, especially as a part of a presentation and a web page. Therefore instead of trying to generate artificial data, a sample of web pages with varying content was selected to include a representative sample of possible elements that can occur in data sharing.

A sample of 50 different sites was selected from the 500 most popular sites on the web according to Alexa top 500 global sites ranking[106]. These rankings are biased towards English and Chinese language, social media, blog, search engine, advertising, shopping, company, various content sharing, news, and entertainment sites. These may not provide a large enough content type variation in itself and therefore sites with similar content were manually reduced. Each site was then sampled for 20 randomly selected pages that were accessible through the site's home page. These subpages were selected to get a more representative content sample than just the home page would provide. A screen capture of the full page content was taken and these screen captures were then used to further reduce the sample count to 200 pages out of 1000 pages that were then put through the barcode detector.

Table 5.7 shows how many barcode candidates were detected for each parameter value pair of the edge intensity threshold τ_I and the pattern match limit ϵ_M. We can see that when the allowed error by the pattern match

Table 5.7: The amount of spurious barcode matches in 200 web page capture samples.

$\epsilon_M \backslash \tau_I$	0.5	0.6	0.7	0.8	0.9
1 %	0	0	0	0	0
2 %	0	0	0	0	0
5 %	0	0	0	0	0
10 %	0	0	0	0	0
15 %	0	0	0	0	0
20 %	6	1	1	1	1

limit ϵ_M goes to 20%, we start seeing some false positive matches. Those single false positive matches for all edge intensity thresholds τ_I come from a single image that includes a white font on black background that has spacing and black-and-white ratios similar to the finder pattern. This suggests that we could have much more false positive matches if we had white text on black background with appropriate font. Currently most page samples have the standard black text on white background style. Barcode area candidate elimination results for different elimination phases are similar to the ones for natural video sequences shown in table 5.6.

We can compare the data sharing results to the results from the natural video sequence test case in the previous section. The natural video sequence case includes 27 times more image sample points in total than this data sharing test case but results only in 1 false positive detection over the whole parameter and sample set. The difference in the data sharing case compared to natural video data is that in natural videos the image changes a little bit all the time so all false positive detections will probably last only a short while. In the data sharing case, however, the same image will probably stay on the screen until it disappears out of the view. This makes it hard to use temporal checks to eliminate spurious matches that last only for one frame.

Chapter 6

Discussion

Using barcode that is optimized for video communication system testing provides an easy and efficient way to encode small amounts of data into images. This barcode then provides spatial and temporal information about where the encoded data resides in. In addition, image compression tests in section 5.2.2 show that even with quite heavy compression we can have a decodable barcode over a small 36×36 pixel area.

This data encoding method, however, has a few limitations and assumptions that make at least this implementation unusable for certain use cases. The detector has an assumption that we have full control of the input and output images of the video communication system. We also assume that image distortions come from video encoding and layout generation. In cases where we can not control the video input and output, we need to use some external video capture/display device. Then we need take into account the properties of the display device, illumination intensity and its variations, image noise, rotation, skew, and frame rate differences between the capture and display device. This makes the selected barcode structure and also the horizontal and vertical edge matching strategy less optimal than structures and search methods that also take rotation into account.

Different phases that work on the matched finder pattern could also be improved. The linear relationship between the module size and finder pattern match error in the finder pattern match merging phase makes this detector quite sensitive to pattern match outliers when the module size and pattern match error limit are large. This could be improved in such a way that the area for finder pattern match merging would not depend linearly on the module size. Match merging could also have a separate parameter managing the area where the mergeable pattern is searched for. As a larger module size also has more matches in module's center, match merging could benefit from more intelligent clustering approaches and outlier detection than just using a simple rectangle overlapping detection criteria for determining if a certain pattern match point belongs to the same cluster.

Reading the module value currently depends only on the average intensity of the area where the module is estimated to be in. We could, for example, use the standard deviation of sample values in case of a larger module and mark modules with high standard deviation as invalid. This would give the error correction a better chance to recover from barcode area damage, as we would know the location of the damaged modules. Different data encoding strategies were briefly discussed in section 4.1.3 with relation to error resiliency but nothing final was presented. This is because module placement depends on the error resiliency properties we want to take into account. The codeword size and placement also depends on the used error correction algorithm and these also depend on how many bits we want to use for the actual data in relation to error correction.

The barcode design and detection tries to take some of the distortions that it encounters into account. However, we treat distortions caused by the transform coding and quantization, and video transcoding as a black box. This is compensated by giving a higher error margin for finder pattern matching. Even though the detector can detect the barcode from one encoding pass that results in bad quality, multiple encoding passes with different transform block alignments might give worse results.

Different elimination phases to reduce barcode area candidates, introduced in section 4.3.7, also cause detection errors, as analyzed in the evaluation chapter. These could be avoided by using a finder pattern that is two modules wide. This would enable the detection of a smaller module size, at least for the finder pattern, but would lead to barcode occupying 10×10 module area instead of 9×9 area and would therefore reduce the size of data encoding modules over the same image area.

Chapter 7

Conclusions

By using a custom barcode type as a data encoding method we created a method that can be used as a part of automatic verification process for a video communication system. We first examined how the video image goes through such a system and what distortions it will encounter. We then examined what methods there are that can be used to encode discrete data into images and selected two-dimensional barcode as the information encoding method. This resulted in the custom barcode that is designed to take video communication system constraints into account. The detector to detect this barcode uses edges and their relative distance difference as the basic feature for pattern detection. This results in ability to detect JPEG compressed barcode images that have 2.0–4.0 pixel sized modules depending on the compression level.

Barcodes are widely used for object identification and with reduced complexity can be applied to create an efficient system for functional verification of a video communication system. There have been no previous references about using barcodes for this type of testing in the literature. However this is a natural use case for barcodes as a data encoding method and there still can exist proprietary implementations of such testing methodology. This can be compared to using text and optical character recognition to transmit data by using images, but with higher data encoding density.

Testing by data injection limits it to cases where we can modify the data that the communication system gets and can read the processed image. From testing point of view this is not a problem, as we usually control the environment in which tests are run. If, on the other hand, we would like to check that existing video data goes through the system without changing the video image itself, we would have to use something like scale invariant feature detection. In this case we would need to rely on the passing video image to have enough features that can be detected and use them to distinguish different streams from each other. This approach was, however, not selected as we aim for the situation where we can control the inputs and analyze the output with high accuracy.

This detector implementation is just one of many possible alternatives and it can be improved, especially for small module sizes in cases where the image interpolation does not behave like an averaging filter. Also the transform coding part of the image compression is treated as a black box. Possible distortions rising from intra- and inter-frame transform coding and from multiple compression steps are mainly taken into account by increasing the error limits of the detector. Also, this detector does not use any information about consecutive frames and video encoding process. Taking the temporal information into account could possibly improve detector's performance by taking into account that when we know the location and the exact value of a barcode, we could treat that part of the frame as the original uncompressed frame.

Bibliography

[1] Global Industry Analysts, “Video conferencing – a global strategic business report,” 2011, [Accessed: 20 Jun. 2012]. [Online]. Available: http://www.strategyr.com/Video_Conferencing_Market_Report.asp

[2] Ookla, “Household download index for period Dec 17, 2009 — Jun 17, 2012,” 2012, [Accessed: 18 Jun. 2012]. [Online]. Available: http://www.netindex.com/

[3] T. Yeh, T.-H. Chang, and R. C. Miller, “Sikuli: using GUI screenshots for search and automation,” in *22nd annual ACM symposium on User interface software and technology*, Victoria, BC, 2009, pp. 183–192.

[4] TestPlant, “eggPlant,” [Accessed: 1 Apr. 2013]. [Online]. Available: http://www.testplant.com/products/eggplant/

[5] A. C. Bovik, Ed., *Handbook of Image and Video Processing (Communications, Networking and Multimedia)*, 2nd ed. Elsevier Academic Press, 2005.

[6] S. Firestone, T. Ramalingam, and S. Fry, *Voice and video conferencing fundamentals*. Indianapolis: Cisco Press, 2007.

[7] A. Hoff, “Justin.tv’s Live Video Broadcasting Architecture,” Mar. 2010, [Accessed: 2 Jul. 2012]. [Online]. Available: http://highscalability.com/blog/2010/3/16/justintvs-live-video-broadcasting-architecture.html

[8] D. Ferrari, “Client requirements for real-time communication services,” RFC 1193, Nov. 1990.

[9] C. Nicolaou, “An architecture for real-time multimedia communication systems,” *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 391–400, Apr. 1990.

[10] Y. Wang and Q.-F. Zhu, “Error control and concealment for video communication: a review,” *Proceedings of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.

[11] J. Garrett-Glaser, “x264: the best low-latency video streaming platform in the world,” 2010, [Accessed: 6 Jul. 2012]. [Online]. Available: http://x264dev.multimedia.cx/archives/249

[12] J. Gettys and K. Nichols, “Bufferbloat: Dark buffers in the internet,” *ACM Queue*, vol. 9, no. 11, pp. 40–54, Nov. 2011.

[13] T. Kim and M. Ammar, “A comparison of heterogeneous video multicast schemes: Layered encoding or stream replication,” *IEEE Transactions on Multimedia*, vol. 7, no. 6, pp. 1123–1130, Dec. 2005.

[14] H. M. Radha, M. van der Schaar, and Y. Chen, “The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming over IP,” *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 53–68, Mar. 2001.

[15] J.-W. Suh and Y.-S. Ho, “Error concealment techniques for digital TV,” *IEEE Transactions on Broadcasting*, vol. 48, no. 4, pp. 299–306, Dec. 2002.

[16] J. M. Boyce and R. D. Gaglianello, “Packet loss effects on MPEG video sent over the public Internet,” in *6th ACM international conference on Multimedia*, Bristol, 1998, pp. 181–190.

[17] Q.-F. Zhu and L. Kerofsky, “Joint source coding, transport processing, and error concealment for H.323-based packet video,” in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 3653, Dec. 1998, pp. 52–62.

[18] A. C. Bovik, Ed., *The Essential Guide to Video Processing*, 2nd ed. Elsevier Academic Press, 2009.

[19] T. Levent-Levi, “What Layouts Should a Dual Video System Support?” Dec. 2009, [Accessed: 1 Aug. 2012]. [Online]. Available: http://blog.radvision.com/voipsurvivor/2009/12/07/what-layouts-should-a-dual-video-system-support/

[20] T. Levent-Levi, “What Layouts Do You Need in Your HD Videophone?” Nov. 2009, [Accessed: 1 Aug. 2012]. [Online]. Available: http://blog.radvision.com/voipsurvivor/2009/11/30/what-layouts-do-you-need-in-your-hd-videophone/

[21] F. Ghetti, “Top, Side or Bottom – Camera Positioning Matters,” Mar. 2010, [Accessed: 28 Aug. 2012]. [Online]. Available: http://blog.radvision.com/voipsurvivor/2010/03/10/top-side-or-bottom-camera-positioning-matters/

[22] T. Levent-Levi, “HD Scaling Made Easy(ier),” Dec. 2009, [Accessed: 9 Aug. 2012]. [Online]. Available: http://blog.radvision.com/voipsurvivor/2009/12/14/hd-scaling-made-easyier/

[23] N. A. Dodgson, “Image resampling,” University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-261, Aug. 1992.

[24] R. Dugad and N. Ahuja, “A fast scheme for image size change in the compressed domain,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 4, pp. 461–474, Apr. 2001.

[25] P. Heckbert, “Color image quantization for frame buffer display,” *SIGGRAPH Computer Graphics*, vol. 16, no. 3, pp. 297–307, Jul. 1982.

[26] E. Dubois and J. Konrad, “Estimation of 2-D Motion Fields from Image Sequences with Application to Motion-Compensated Processing,” *Motion Analysis and Image Sequence Processing*, vol. 220, pp. 53–87, 1993.

[27] *Information technology – Generic coding of moving pictures and associated audio information: Systems*, ISO/IEC Standard 13 818-1:2007.

[28] *Line transmission of non-telephone signals – Video Codec for Audiovisual Services at P X 64 Kbits*, ITU-T Recommendation H.261, 1993.

[29] *Information technology – Coding of audio-visual objects – Part 2: Visual*, ISO/IEC Standard 14 496-2:2004.

[30] *Series H: audiovisual and multimedia systems – Infrastructure of audiovisual services – Coding of moving video – Video coding for low bit rate communication*, ITU-T Recommendation H.263, 2005.

[31] *Series H: audiovisual and multimedia systems – Infrastructure of audiovisual services – Coding of moving video – Advanced video coding for generic audiovisual services*, ITU-T Recommendation H.264, 2012.

[32] C. Poynton, *Digital Video and HDTV Algorithms and Interfaces.* San Francisco: Morgan Kaufmann Publishers, 2003.

[33] R. C. Gonzalez and R. E. Woods, *Digital image processing*, 2nd ed. Upper Saddle River: Prentice Hall, 2002.

[34] *Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios*, ITU-R Recommendation BT.601, 2011.

[35] *Parameter values for the HDTV standards for production and international programme exchange*, ITU-R Recommendation BT.709, 2002.

[36] G. Chan, "Toward better chroma subsampling," *SMPTE Motion Imaging Journal*, vol. 117, no. 4, pp. 39–45, 2008.

[37] *Series P: Telephone transmission quality – Audiovisual quality in multimedia services – Principles of a reference impairment system for video*, ITU-T Recommendation P.930, 1996.

[38] M. Robertson and R. Stevenson, "DCT quantization noise in compressed images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 1, pp. 27–38, Jan. 2005.

[39] C. F. John, J. Libert, and P. Roitman, "Mosquito noise in MPEG-compressed video: test patterns and metrics," in *SPIE Proceedings: Human Vision and Electronic Imaging V*, vol. 3959, Jun. 2000, pp. 604–612.

[40] Y. Michalevsky and T. Shoham, "Fast H.264 Picture in Picture (PIP) transcoder with B-slices and direct mode support," in *15th IEEE Mediterranean Electrotechnical Conference (MELECON)*, Apr. 2010, pp. 862–867.

[41] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 614–619, Jul. 2003.

[42] *Series H: audiovisual and multimedia systems – Infrastructure of audiovisual services – Coding of moving video – High efficiency video coding*, ITU-T Recommendation H.265, 2013.

[43] C.-M. Fu, C.-Y. Chen, Y.-W. Huang, and S. Lei, "Sample adaptive offset for HEVC," in *IEEE 13th International Workshop on Multimedia Signal Processing (MMSP)*, Hangzhou, Oct. 2011, pp. 1–5.

[44] K. Miyazawa, T. Murakami, A. Minezawa, and H. Sakate, "Complexity reduction of in-loop filtering for compressed image restoration in HEVC," in *Picture Coding Symposium (PCS)*, Krakóv, 2012, pp. 413–416.

[45] P. Assuncao and M. Ghanbari, “A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 8, pp. 953–967, Dec. 1998.

[46] S. Shen, W. Shen, Y. Fabn, and X. Zeng, “A unified 4/8/16/32-point integer IDCT architecture for multiple video coding standards,” in *IEEE International Conference on Multimedia and Expo (ICME)*, Melbourne, Jul. 2012.

[47] *Series H: audiovisual and multimedia systems – Infrastructure of audiovisual services – Systems and terminal equipment for audiovisual services – Terminal for low bit-rate multimedia communication*, ITU-T Recommendation H.324, 2009.

[48] GSM Arena, “Phone finder,” [Accessed: 7 Jan. 2013]. [Online]. Available: http://www.gsmarena.com/search.php3

[49] C.-h. Chen, W. Härdle, and A. Unwin, Eds., *Handbook of Data Visualization*, 1st ed. Springer-Verlang Berlin Heidelberg, 2008.

[50] C. Ware, *Information Visualization: Perception for Design*, 2nd ed. San Francisco: Morgan Kaufmann Publishers, 2004.

[51] J. Breidenbach and D. L. Hecht, “Foreground/background document processing with dataglyphs,” U.S. Patent US 6 641 053, Nov. 04, 2003.

[52] L. Eikvil, “Ocr – optical character recognition,” 1993. [Online]. Available: http://www.nr.no/~eikvil/OCR.pdf

[53] I. Marosi, “Industrial OCR approaches: architecture, algorithms, and adaptation techniques,” *SPIE Proceedings: Document Recognition and Retrieval XIV*, vol. 6500, 2007.

[54] *Character Set for Optical Character Recognition (OCR-A)*, ANSI INCITS Standard 17-1981, 2002.

[55] *Alphanumeric character sets for optical recognition – Part 2: Character set OCR-B – Shapes and dimensions of the printed image*, ISO Standard 1073-2:1976.

[56] R. Pointer, “A very tiny, monospace, bitmap font,” [Accessed: 6 Sep. 2012]. [Online]. Available: http://robey.lag.net/2010/01/23/tiny-monospace-font.html

[57] M. C. Koss, “Tiny (3x5) Font Created for the Apple II program The Terminal,” [Accessed: 6 Sep. 2012]. [Online]. Available: http://mckoss.com/jscript/tinyalice.htm

[58] T. Pavlidis, “A new paper/computer interface: two-dimensional symbologies,” in *15th International Conference on Pattern Recognition (ICPR)*, Barcelona, 2000, pp. 145–151.

[59] S. W. Schilke and A. Rauber, “Long-term archiving of digital data on microfilm,” *International Journal of Electronic Governance*, vol. 3, pp. 237–253, 2010.

[60] F. Petitcolas, “Watermarking schemes evaluation,” *IEEE Signal Processing Magazine*, vol. 17, no. 5, pp. 58–64, Sep. 2000.

[61] P. Premaratne and F. Safaei, “2D Barcodes as Watermarks in Image Authentication,” in *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, Melbourne, Jul. 2007, pp. 432–437.

[62] I. J. Cox and M. L. Miller, “A review of watermarking and the importance of perceptual modeling,” in *Conference on Human Vision and Electronic Imaging II*, San Jose, CA, Jun. 1997, pp. 92–99.

[63] M. Kutter, “Watermarking resisting to translation, rotation, and scaling,” in *SPIE Conference on Multimedia Systems and Applications*, Nov. 1998, pp. 423–431.

[64] G. B. Rhoads, “Method for monitoring internet dissemination of image, video, and/or audio files,” U.S. Patent 7 653 210, Jan. 26, 2010.

[65] J. J. ÓRuanaidh and T. Pun, “Rotation, scale and translation invariant spread spectrum digital image watermarking,” *Signal Processing*, vol. 66, no. 3, pp. 303–317, May 1998.

[66] I. Cox, J. Kilian, F. Leighton, and T. Shamoon, “Secure spread spectrum watermarking for multimedia,” *IEEE Transactions on Image Processing*, vol. 6, no. 12, pp. 1673–1687, Dec. 1997.

[67] M. Kutter, S. Bhattacharjee, and T. Ebrahimi, “Towards second generation watermarking schemes,” in *Proceedings of IEEE International Conference on Image Processing*, vol. 1, 1999, pp. 320–323.

[68] P. M. J. Rongen, M. J. Maes, and K. W. van Overveld, “Digital image watermarking by salient point modification: practical results,”

Proceedings of SPIE: Proceeding of the Security and Watermarking of Multimedia Contents, vol. 3657, pp. 273–282, 1999.

[69] J. Zhang, J. Li, and L. Zhang, "Video watermark technique in motion vector," in *Proceedings of XIV Brazilian Symposium on Computer Graphics and Image Processing*, Oct. 2001, pp. 179–182.

[70] *Information technology – Automatic identification and data capture techniques – Code 128 bar code symbology specification*, ISO/IEC Standard 15 417:2007.

[71] *Information technology – Automatic identification and data capture techniques – EAN/UPC bar code symbology specification*, ISO/IEC Standard 15 420:2009.

[72] *Information technology – Automatic identification and data capture techniques – Code 39 bar code symbology specification*, ISO/IEC Standard 16 388:2007.

[73] *Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification*, ISO/IEC Standard 18 004:2006.

[74] *Information technology – Automatic identification and data capture techniques – Data Matrix bar code symbology specification*, ISO/IEC Standard 16 022:2006.

[75] *Information technology – International symbology specification – MaxiCode*, ISO/IEC Standard 16 023:2000.

[76] *Information technology – Automatic identification and data capture techniques – PDF417 bar code symbology specification*, ISO/IEC Standard 15 438:2006.

[77] *Information technology – Automatic identification and data capture techniques – Aztec Code bar code symbology specification*, ISO/IEC Standard 24 778:2008.

[78] D. Parikh and G. Jancke, "Localization and Segmentation of A 2D High Capacity Color Barcode," in *IEEE Workshop on Applications of Computer Vision*, Copper Mountain, CO, Jan. 2008, pp. 1–6.

[79] E. Ohbuchi, H. Hanaizumi, and L. Hock, "Barcode readers using the camera device in mobile phones," in *International Conference on Cyberworlds*, Tokyo, Nov. 2004, pp. 260–265.

[80] Y. Liu, J. Yang, and M. Liu, “Recognition of QR Code with mobile phones,” in *Chinese Control and Decision Conference (CCDC)*, Jul. 2008, pp. 203–206.

[81] PAPERBACK (version 1.00). [Online]. Available: http://www.ollydbg.de/Paperbak/index.html

[82] *Standard for Information Technology - Portable Operating System Interface (POSIX(R))*, IEEE Standard 1003.1-2008.

[83] *Uniform Symbology Specification – Code 16k*, ANSI/AIM Standard BC7, 1995.

[84] *ISS Bar code symbology specification – DotCode*, AIM Specification ISS DotCode, 2012.

[85] H. Kato and K. Tan, “2D barcodes for mobile phones,” in *2nd International Conference on Mobile Technology, Applications and Systems*, Guangzhou, Nov. 2005, p. 8 pp.

[86] S. B. Wicker, *Error control systems for digital communication and storage.* Upper Saddle River: Prentice-Hall, 1994.

[87] E. Ouaviani, A. Pavan, M. Bottazzi, E. Brunelli, F. Caselli, and M. Guerrero, “A common image processing framework for 2D barcode reading,” in *Seventh International Conference on Image Processing and Its Applications*, vol. 2, Manchester, 1999, pp. 652–655.

[88] M. Rohs, “Real-world interaction with camera phones,” in *2nd International Symposium on Ubiquitous Computing Systems (UCS)*, Tokyo, 2005, pp. 74–89.

[89] H. Hu, W. Xu, and Q. Huang, “A 2D Barcode Extraction Method Based on Texture Direction Analysis,” in *Fifth International Conference on Image and Graphics*, Xi’an, Shanxi, Sep. 2009, pp. 759–762.

[90] Y. Liu and M. Liu, “Automatic Recognition Algorithm of Quick Response Code Based on Embedded System,” in *Intelligent Systems Design and Applications, 2006. ISDA ’06. Sixth International Conference on*, vol. 2, Jinan, Oct. 2006, pp. 783–788.

[91] C.-H. Chu, D.-N. Yang, Y.-L. Pan, and M.-S. Chen, “Stabilization and extraction of 2D barcodes for camera phones,” *Multimedia Systems*, vol. 17, pp. 113–133, 2011.

[92] L. K. Leong and W. Yue, "Extraction of 2D Barcode Using Keypoint Selection and Line Detection," in *10th Pacific Rim Conference on Multimedia: Advances in Multimedia Information Processing*, Bangkok, 2009, pp. 826–835.

[93] D.-T. Lin and C.-L. Lin, "Multi-symbology and Multiple 1D/2D Barcodes Extraction Framework," in *Advances in Multimedia Modeling*, Taipei, 2011, vol. 6524, pp. 401–410.

[94] H. Wang and Y. Zou, "2D Bar codes reading: solutions for camera phones," *International Journal of Signal Processing*, vol. 3, no. 3, pp. 164–170, 2006.

[95] Q. Liu, X. Li, M. Zou, and J. Zhou, "The multi-QR codes extraction method in illegible image based on contour tracing," in *IEEE International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, Xiamen, 2011, pp. 51–56.

[96] M. Wang, L.-N. Li, and Z.-X. Yang, "Gabor filtering-based scale and rotation invariance feature for 2d barcode region detection," in *International Conference on Computer Application and System Modeling (ICCASM)*, vol. 5, Taiyuan, Oct. 2010, pp. V5–34–V5–37.

[97] H. Wang and Y. Zou, "Camera Readable 2D Bar Codes Design and Decoding for Mobile Phones," in *IEEE International Conference on Image Processing*, Atlanta, GA, Oct. 2006, pp. 469–472.

[98] ZBar bar code reader (version 0.10). [Online]. Available: http://zbar.sourceforge.net/

[99] libdecodeqr (version 0.9.3). [Online]. Available: http://trac.koka-in.org/libdecodeqr

[100] K. Turkowski, "Filters for common resampling tasks," in *Graphics gems*, A. S. Glassner, Ed. San Diego, CA USA: Academic Press Professional, Inc., 1990, pp. 147–165.

[101] D. Ziou and S. Tabbone, "Edge detection techniques - an overview," *International Journal of Pattern Recognition and Image Analysis*, vol. 8, pp. 537–559, 1998.

[102] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *17th Annual Symposium on Foundations of Computer Science*, Houston, TX, 1976, pp. 208–215.

[103] J. Bentley and T. Ottmann, “Algorithms for reporting and counting geometric intersections,” *IEEE Transactions on Computers*, vol. C-28, no. 9, pp. 643–647, Sep. 1979.

[104] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[105] *Information Technology – Digital Compression and Coding of Continuous-tone Still Images – Requirements and guidelines*, ITU-T Recommendation T.81, 1992.

[106] Alexa Internet, Inc., “Alexa Top 500 Global Sites,” [Accessed: 12 Dec. 2012]. [Online]. Available: http://www.alexa.com/topsites/global

[107] JPEG library (version 8d). [Online]. Available: http://www.ijg.org/

Appendix A

Discrete cosine transform

The two-dimensional discrete cosine transform and its approximations are the basic workhorse of the most used lossy transform based image and video compression standards. The discrete cosine transform enables to apply information reduction on each image block that results in a high compression ratio with visually desirable results compared to the amount of data reduction. This appendix describes the use of the discrete cosine transform that is used in many lossy image and video compression methods[30, 33, 105] and some of its implementation details for JPEG compressed images[105].

A.1 Block based discrete cosine transform

When applying the discrete cosine transform onto images, the original source image is divided into a $N \times N$ pixel blocks and each block then has the discrete cosine transform independently applied to it. The often used discrete cosine transform for $N \times N$ sized blocks is defined by:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos(\frac{(2x+1)u\pi}{2N}) \cos(\frac{(2y+1)v\pi}{2N})$$

where

$$\begin{array}{rcl} x, y & = & \text{pixel domain coordinates } \{0, \ldots, N-1\}; \\ u, v & = & \text{transform domain coordinates } \{0, \ldots, N-1\}; \\ C(u) & = & 1/\sqrt{2} \text{ for } u = 0, \text{ otherwise } 1; \\ C(v) & = & 1/\sqrt{2} \text{ for } v = 0, \text{ otherwise } 1. \end{array}$$

The transformation is often carried out in such a way that the original pixel domain values $f(x,y)$ are centered around 0 and then transformed. The transformed block is then quantized by using a quantization table Q in such a

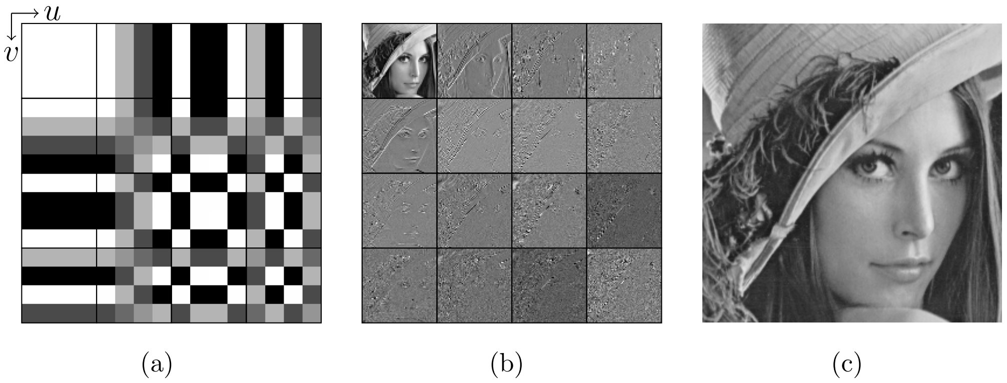


Figure A.1: Properties of the discrete cosine transform. (a) Normalized basis functions for 4x4 discrete cosine transform. Black indicates the smallest value of the 4x4 basis function and white indicates the largest value. (b) Normalized images resulting from transform blocks corresponding to each basis function, showing the feature extracting properties of each basis function. (c) The original 256×256 image to which the 4×4 discrete cosine transform is applied.

way that the quantization table element $Q(u, v)$ divides the transform domain value $F(u, v)$. This results in a quantized value $\tilde{F}(u, v) = \mathbf{round}(\frac{F(u,v)}{Q(u,v)})$ where $\mathbf{round}()$ function rounds to the nearest discrete encodable value by some rounding rule. The accuracy of quantized values depends on the amount of bits that are used for encoding. Then these quantized values are compressed by using some lossless coding method and encoded into a bit stream that forms the image.

The decoded image value $\tilde{f}(x, y)$ is calculated from quantized transform values $\tilde{F}(u, v)$ by the inverse discrete cosine transform:

$$\tilde{f}(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)\tilde{F}(u,v) \cos(\frac{(2x+1)u\pi}{2N}) \cos(\frac{(2y+1)v\pi}{2N}).$$

Properties of the discrete cosine transform can be visualized by visualizing the basis functions for each value $F(u, v)$ of the transform. These basis functions are visualized in figure A.1(a) for 4×4 discrete cosine transform, where the basis function image is normalized in such way that the maximum value of the basis function is white and the minimum value of the basis function is black. The upper left corner has basis function value for $F(u, v) = F(0, 0)$ which corresponds to equally weighted values for all block samples. Higher u and v values correspond to higher frequencies in horizontal and vertical directions. The result of applying these basis functions into the face image shown in figure A.1(c) can be seen from figure A.1(b). Each block of figure A.1(b) has been normalized in such a way that the minimum value corresponds to a black pixel and the maximum value corresponds to a white pixel. The upper left image results in $1/N$ resolution reduced version of the original image. Rows correspond to vertical features, that is visible from the nose, and columns correspond to horizontal features, that is visible from the mouth. Higher values of u and v correspond to smaller scale features, higher frequencies. This, combined with the properties of the human visual system, leads to quantization matrices that reduce the importance of high frequency components by having larger quantization constants in the lower right area of the quantization matrix.

A.2 Discrete cosine transform in JPEG

The JPEG image format is probably the most widely used image format for distributing natural images, as it is the default image format in most digital cameras and in web based distribution. JPEG applies the discrete cosine transform on 8×8 non-overlapping blocks, separately for each image channel. The quantization matrix in JPEG is also defined per channel basis and can be different for each channel depending on the desired compression level. Even though the JPEG standard does not force any specific quantization matrix, an often used JPEG library implementation[107] derives the used quantization matrices from the quantization tables given in Annex K of the JPEG standard[105]. The JPEG image compression also often has a specific quality level that is related to the desired compression level

To calculate the desired quantization matrices for the given quality level, the JPEG library takes an integer valued quality level $Q \in 1 \ldots 100$ and converts it to a multiplier that creates the final quantization matrices from the standard JPEG quantization tables shown in table A.1. The quality level is first converted to an integer scaling factor C where

$$C = \begin{cases} 5000/Q & \text{if } 1 \leq Q < 50 \\ 200 - 2Q & \text{if } 50 \leq Q \leq 100 \end{cases}$$

Then this scaling factor C is applied to the example JPEG quantization tables for the luma and subsampled chroma channels. The scaling factor C is applied to the quantization table element T_{ij} so that this results in an integer valued quantization matrix element Q_{ij} by:

$$Q_{ij} = \frac{CT_{ij} + 50}{100}.$$

Table A.1: The example JPEG quantization matrices specified in the standard for (a) the luma channel and (b) for chroma channels.

(a)

16	11	**10**	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	**121**	120	101
72	92	95	98	112	100	103	99

(b)

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

This value is then cut between $[1 \ldots Q^{\max}]$ to fit to the values that the used bit-depth supports and to prevent the division by zero. Often used bit depth is 8 bits per channel. This provides 256 distinct values between $[0 \ldots 255]$ which results in $Q_8^{\max} = 255$. Figure A.2 shows how the minimum and the maximum quantization value changes for 8-bit quantization matrices based on the JPEG quality level. These values come from the minimum and the maximum value in the example quantization table for the luma channel (table A.1(a)), as the quantization table for the chroma channel (table A.1(b)) includes values that fall between these extremes.

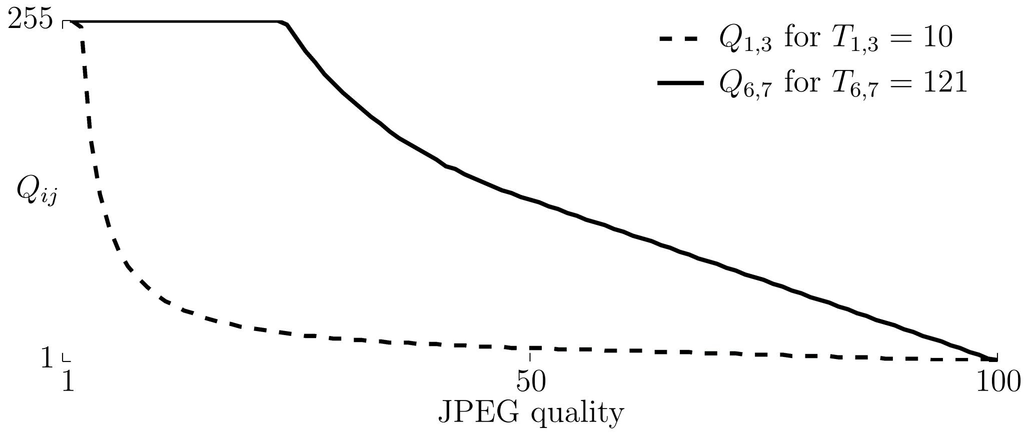


Figure A.2: The minimum and maximum quantization matrix value depending on the quality level that an often used JPEG library[107] produces for 8-bit images.

Appendix B

Video test sequences

Test sequences that are used in section 5.3.1 to see if the detector detects false positives in natural scenes are part of Xiph.org Video Test Media collection[1]. Many of the images showing natural sequences are also created from screen captures of these sequences. Table B.1 shows the name and other properties of the used video files.

Table B.1: Video test sequences used in the evaluation section.

Filename (.y4m)	Frames	Resolution	Size (bytes)	Filename (.y4m)	Frames	Resolution	Size (bytes)
720p50_parkrun_ter	504	1280×720	696 732 659	intros_422_ntsc	360	720×486	251 944 605
akiyo_cif	300	352×288	45 621 044	mad900_cif	900	352×288	136 863 044
aspen_1080p	570	1920×1080	2 363 907 467	mobile_calendar_422_ntsc	360	720×486	251 944 605
blue_sky_1080p25	217	1920×1080	674 958 138	mobile_cif	300	352×288	45 621 054
bowing_cif	300	352×288	45 621 044	mother_daughter_cif	300	352×288	45 621 044
carphone_qcif	382	176×144	14 524 448	mthr_dotr_qcif	961	176×144	36 539 186
claire_qcif	494	176×144	18 782 912	news_cif	300	352×288	45 621 044
coastguard_cif	300	352×288	45 621 044	pamphlet_cif	300	352×288	45 621 044
crowd_run_1080p50	500	1920×1080	1 555 203 036	paris_cif	1065	352×288	161 954 594
deadline_cif	1374	352×288	208 944 224	park_joy_1080p50	500	1920×1080	1 555 203 036
elephants_dream_720p24	15691	1280×720	21 691 332 590	salesman_qcif	449	176×144	17 071 922
flower_garden_422_ntsc	360	720×486	251 944 605	soccer_4cif	600	704×576	364 957 238
football_422_ntsc	360	720×486	251 944 605	stefan_sif	300	352×240	38 017 846
foreman_cif	300	352×288	45 621 044	students_cif	1007	352×288	153 134 534
galleon_422_ntsc	360	720×486	251 944 605	suzie_qcif	150	176×144	5 703 344
garden_sif	115	352×240	14 573 536	tempete_cif	260	352×288	39 538 244
grandma_qcif	870	176×144	33 079 184	tennis_sif	150	352×240	19 008 946
hall_monitor_cif	300	352×288	45 621 044	trevor_qcif	150	176×144	5 703 344
harbour_4cif	600	704×576	364 957 238	tt_sif	112	352×240	14 193 358
highway_cif	2000	352×288	304 140 044	vtc1nw_422_ntsc	360	720×486	251 944 605
husky_cif	250	352×288	38 017 544	washdc_422_ntsc	360	720×486	251 944 605
ice_4cif	480	704×576	291 965 798	waterfall_cif	260	352×288	39 538 244
Total	36131	-	33 032 747 339				

[1]`http://media.xiph.org/video/derf/`