Universidade de Aveiro
Departamento de Eletrónica,
Telecomunicações e Informática
2018

Álvaro Rodrigues de Castro Mendes Martins

Compressão de dados sensoriais em sistemas robóticos

Compression of sensor data in robotic systems

Universidade de Aveiro
2018

Departamento de Eletrónica,
Telecomunicações e Informática

Álvaro Rodrigues de Castro Mendes Martins

Compressão de dados sensoriais em sistemas robóticos

Compression of sensor data in robotic systems

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor António José Ribeiro Neves, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Miguel Armando Riem de Oliveira, Professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

Aos meus pais José Maria e Maria Gabriela, aos meus irmãos Daniel, João e Henrique e à minha namorada Cláudia.

o júri / the jury

presidente / president

Prof. Doutor Armando José Formoso de Pinho
Professor associado com agregação da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Paulo José Cerqueira Gomes da Costa
Professor auxiliar da Faculdade de engenharia da Universidade do Porto (Arguente)

Prof. Doutor António José Ribeiro Neves
Professor auxiliar da Universidade de Aveiro (Orientador)

agradecimentos /
acknowledgements

Ao meu orientador, Doutor António Neves, agradeço todo o apoio, conselhos e voto de confiança. Sem a sua ajuda e incentivo este trabalho não iria ser possível.
Ao meu coorientador, Doutor Miguel Riem Oliveira, agradeço todos os ensinamentos e sugestões para que este trabalho fosse o melhor possível.
Aos meus pais e irmãos, expresso a minha profunda gratidão pelo apoio incondicional, ajuda e encorajamento contínuo ao longo deste percurso. Esta conquista não teria sido possível sem eles.
À Cláudia, que sempre acreditou em mim, agradeço toda a motivação e inspiração em momentos de maior necessidade.
Aos meus colegas, que sempre me apoiaram, um obrigado.

Palavras Chave ROS, Robótica, Visão por Computador, Compressão de dados.

Resumo

Um dos principais problemas no desenvolvimento e depuração de sistemas robóticos é a quantidade de dados armazenados em ficheiros contendo dados sensoriais (ex. ficheiros de log proprietários de ROS - Bags). Se considerarmos um robô com várias câmaras e outros sensores, que recolhem informação do ambiente diversas vezes por segundo, obtemos rapidamente ficheiros muito grandes. Além das preocupações com o armazenamento e, em alguns casos, a transmissão, torna-se extremamente difícil encontrar informações importantes nesses ficheiros.

Nesta dissertação, procuramos a melhor solução para os dois problemas estudando e implementando soluções de compressão de dados para reduzir os ficheiros referidos. O foco principal foi compressão de imagem/video, de longe, os dados que consomem mais armazenamento. Além disso, realizamos um estudo detalhado sobre o efeito de compressão com perdas no desempenho de alguns algoritmos de análise de imagem estado da arte.

Outra contribuição foi o desenvolvimento de um leitor de vídeo inteligente para ajudar os roboticistas no seu trabalho enquanto avaliam os dados gravados. Partes do vídeo que não contêm informações relevantes são aceleradas durante a leitura.

Com base nos resultados, concluímos que a compressão nativa de ROS não é suficiente. Além disso, soluções baseadas em ROS, ou de um modo geral qualquer sistema robótico que precise de lidar com dados de imagem/vídeo, beneficiaria com o uso de um codec H.265, uma vez que fornece o menor número de bits por pixel sem penalização significativa da eficiência dos algoritmos de análise de imagem.

Keywords ROS, Robotics, Computer Vision, Data Compression.

Abstract

One of the main problems in the development and debugging of robotic systems is the amount of data stored in files containing sensor data (ex. ROS proprietary log files - BAGS). If we consider a robot with several cameras and other sensors that collect information from the environment several times per second, we quickly obtain very large files. Besides the concerns regarding storage and, in some cases, transmission, it becomes extremely hard to find important information in these files.

In this thesis, we tried to solve both problems studying and implementing data compression solutions to reduce the referred files. The main focus was image and video compression, by far the most storage consuming data. Moreover, we conducted a detailed study about the effect of lossy compression methods in the performance of some state of the art image analysis algorithms.

Another contribution was the development of an intelligent video player to help roboticists in their work while they evaluate the recorded data after experiments. Parts of the video that do not contain relevant information are skipped during the play.

Based on the results, we concluded that ROS native compression is not sufficient. Furthermore, solutions based on ROS, or virtually any robotic system that has to deal with image/video data, would benefit with the use of a H.265 codec, as it provides the smallest number of bits per pixel without a significant penalty on the performance of image analysis algorithms.

Contents

List of Figures

List of Tables

Glossary

PNG	Portable Network Graphics
JPEG	Joint Photographic Experts Group
ROS	Robot Operating System
ISO	International Standards Organization
DCT	Discreet Cosine Transform
DST	Discreet Sine Transform
DWT	Discreet Wavelet Transform
codec	enCOder/DECoder
ROI	Region Of Interest
RGB	Red, Green and Blue
EBCOT	embedded block coding with optimized truncation
LOCO-I	LOw COmplexity LOssless COmpression for Images
HEVC	High Efficiency Video Coding
AVC	Advanced Video Coding
VCEG	Video Coding Experts Group
MPEG	Moving Picture Experts Group
BPG	Better Portable Graphics
CTU	Coding Tree Unit
CU	Coding Unit
SAO	Sample Adaptive Offset filter
AMVP	Advanced Motion Vector Prediction
MB	Macroblock
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable Length Coding
FLC	Fixed Length Codes
CRF	Constant Rate Factor
PSNR	Peak Signal to Noise Ratio
VBR	Variable Bitrate
CBR	Constant Bitrate
LBPH	Local Binary Patterns Histograms
LBP	Local Binary Patterns
MED	Median Edge Detection
YOLO	You Only Look Once
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robust Features
JSON	JavaScript Object Notation
CSV	Comma-separated values
XML	Extensible Markup Language

CHAPTER 1

Introduction

Robotic systems are becoming more and more common in society. Whether in supermarkets, on the road, on the industry, etc. At the development phase, when performing real world tests, there is a need for storing all sensor data. This stored sensor data will then be used to simulate and operate at the laboratory, rather than on-site. However, the longer the robot needs to operate autonomously during the testing phase, the more data needs to be recorded. Moreover, sometimes there is the need to acquire sensor data from a remote location (remote sensing). Not only the data has to be recorded, but it also has to be transmitted. The more sensors the system has, the more data has to be transmitted simultaneously.

One of the main problems in the development and debugging of robotic systems is the amount of data stored in files containing sensor data (ex. ROS proprietary log files - bags). If we consider a robot that contains several cameras and other sensors, that collect information from the environment several times per second, we quickly obtain very large files. Files that can grow to several Gigabytes in a matter of minutes. Besides the concerns regarding storage and, in some cases, transmission, it becomes extremely hard to find important information in these files. Figure 1.1 contains such an example.

In this thesis, we try to solve both problems, studying and implementing data compression solutions to reduce the size of the referred files. The main focus is image and video compression, by far the most storage consuming data. We performed experiments with the native Robot Operating System (ROS) compression standards Bzip2 [1][2], LZ4 [3][4], PNG [5], JPEG [6] and Theora [7] to measure their effectiveness. Besides the native compression standards, several state of the art compression standards are tested, more specifically JPEG2000 [8], JPEG-LS [9], Better Portable Graphics (BPG) [10], H.264 [11]–[13] and H.265 [14][15]. Moreover, we conduct a detailed study about the effect of lossy compression methods in the performance of some state of the art image analysis algorithms, in the areas of facial recognition, face detection, people detection, contour finding, color segmentation and feature extraction.

Another contribution of this thesis was the development of an intelligent video player to help roboticists in their work while they evaluate the recorded data after experiments.

Evaluating the average bitrate and the real time bitrate parts of the video that do not contain relevant information are skipped during the play.

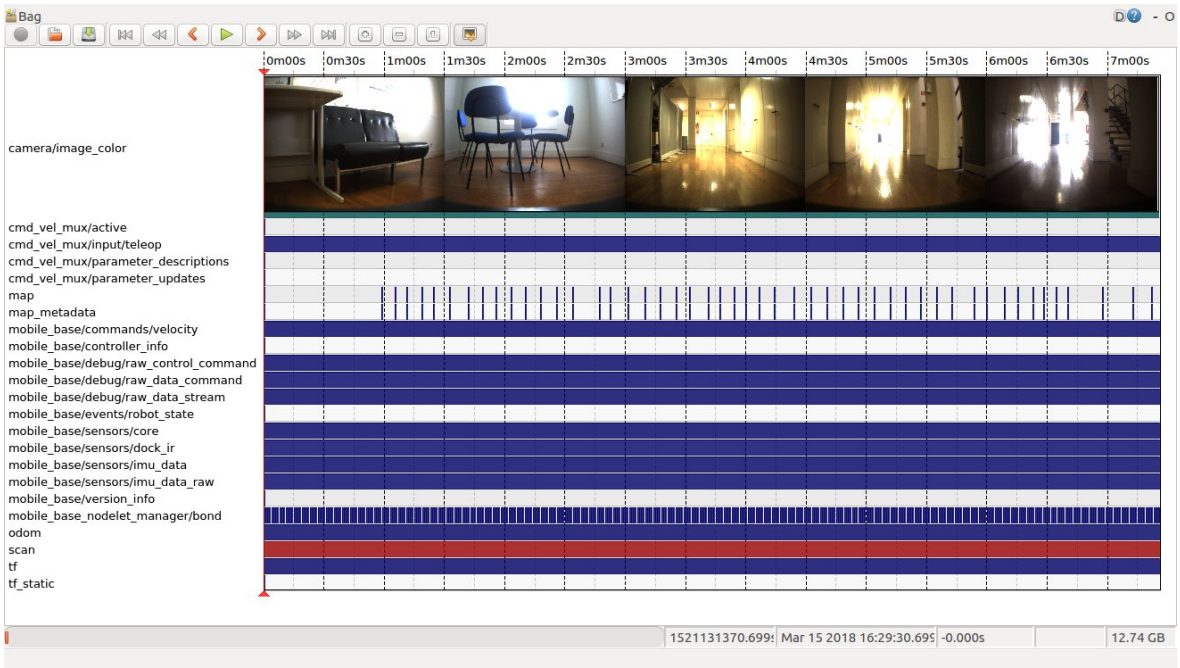


Figure 1.1: Example bag representation using the tool rqt_bag. On the left it shows each topic contained in the bag. On the right it presents the streams of data associated with each topic. The camera topic also presents thumbnails.

1.1 Robot Operating System

Throughout this thesis we make extensive use of a framework for the development of software for robots called Robot Operating System (ROS) [16]. Despite the name, ROS is not an operative system in the traditional sense. It provides, however, an abstraction layer on top of the host operating system. It is a modular framework that provides roboticists with a set of libraries, tools, and conventions to aid in the development of software for robots.

ROS promotes collaboration between roboticists due to its modular approach, with the use of packages. A package can contain nodes (processes), libraries, datasets, configuration files, third-party software, drivers or anything else that provides something useful.

ROS is also designed to be as distributed as possible, providing a built-in low level message passing interface that provides inter-process communication. The most used pattern of communication between nodes is the publisher/subscriber pattern (Figure 1.2), however, nodes can communicate using a variety of patterns, such as: publisher/subscriber, request/response, parameter servers and record/playback.

For recording data, ROS provides developers with a special file format and tools for dealing with the referred files. Such files are called bags.

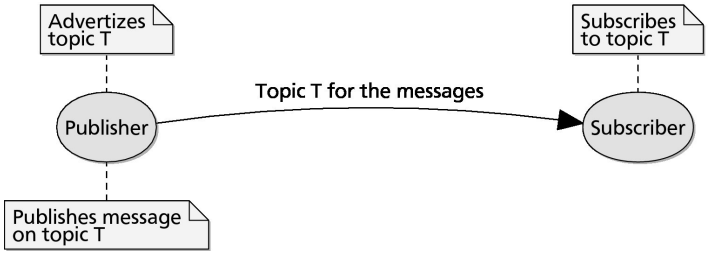


Figure 1.2: Diagram with an example of a publisher subscriber pattern [17].

The ROS bag file format is a logging format that is used for storing ROS messages. It consists of a field that indicates the bag format version number and sequences of records. Records contain headers and the data. There are 6 types of records (since version 2.0):

- **Bag header** holds information about the bag;
- **Chunk** holds connection and message records, which may be compressed using LZ4 or Bzip2;
- **Connection** holds the header of a ROS connection;
- **Message data** contains the serialized message data and the ID of the connection;
- **Index data** holds an index of messages in a single connection of the previous chunk;
- **Chunk info** holds information about messages in a chunk.

Figure 1.3 contains a representation of the bag file format 2.0. A detailed description of the underlying low level format is available at [18].

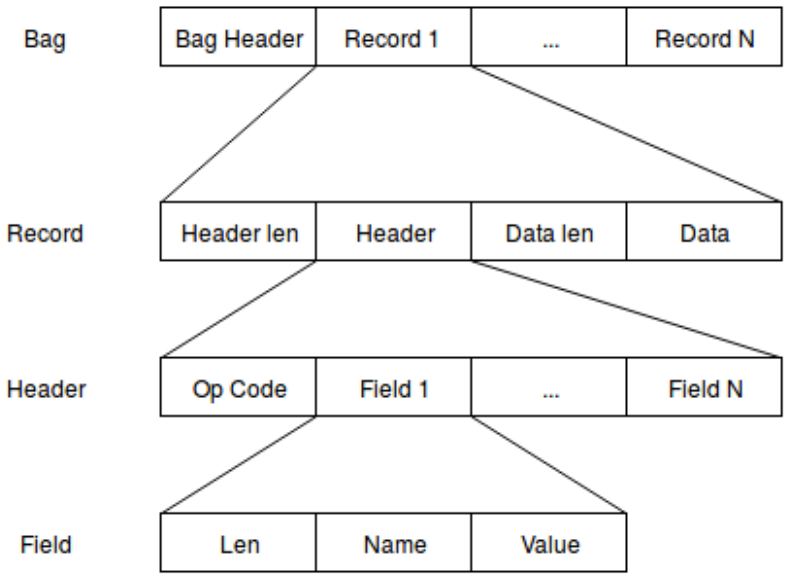


Figure 1.3: Block diagram representing the bag format 2.0 structure.

1.2 Main Contributions

The main objective of this thesis is to study the issues that arise in the development and debugging of robotic systems, associated with the amount of data that requires logging. The

main focus is image and video compression, due to the storage needs associated with this type of data. With this in mind, the major contributions of this thesis are:

- Study the compression standards native to ROS.
- Study the improvement introduced with state of the art compression algorithms and measure their benefits compared to the native standards.
- Conduct a detailed study on the effects of lossy compression in the performance of several image analysis algorithms.
- Develop an intelligent video player to help roboticists in their work while they evaluate the recorded data after experiments.

1.3 Thesis structure

This thesis is divided into 6 chapters and 4 appendixes.

- Chapter 2 presents a study of the compression standards and compressors native to ROS. It also provides a set of measurements to assess the efficiency of referred standards and compressors.
- Chapter 3 introduces state of the art compression standards and compressors. The ROS native compression standards are put to test against the state of the art standards.
- Chapter 4 we conduct a detailed study on the effects of lossy compression on several popular image analysis algorithms
- Chapter 5 presents the concept and implementation of the intelligent video player.
- Chapter 6 summarizes the conclusions of this work and indicates possible directions for future work.
- Appendix A presents details about the Bags used in the development of this thesis. Appendix B contains the details about the image datasets obtained from the referred Bags.
- Appendix C contains code samples from the code implemented during the development phase of this thesis.
- Appendix D contains information about all the software and hardware used throughout this thesis.

CHAPTER 2

Native compression

The Robot Operating System (ROS) already supports some compression mechanisms and standards, the ROS native compression. In this chapter the compression algorithms Bzip2, LZ4, PNG, JPEG and Theora will be introduced. The compression efficiency of these mechanisms and standards will be explored.

Bag files can contain data from a variety of sensors, each with its own type of data. As such, ROS can't make any assumption on the type of data stored in the bag files. Not knowing what kind of data the compressors are dealing with, the only type of compression that can be used is lossless compression. With the introduction of ROS bag format 2.0, the bag file is now divided into chunks that can be individually compressed, either using LZ4 or the Bzip2 compression algorithms.

The compression of bags and the compression of bag messages are important but different concepts. Essentially, bag files contain serialized message data. Which means that, despite the restriction on the type of compression that can be used on bag files, all the compression algorithms supported on messages will also affect the compression on bag files. This leads us to different compression paradigms, because, unlike bag files, messages are not type agnostic, i.e., a message can only contain a specific type of data, depending on the message type. With the Compressed Image Transport [19] plugin ROS is able to send Portable Network Graphics (PNG) or Joint Photographic Experts Group (JPEG) images on messages and consequently store them compressed in bag files, allowing the use of image specific compression standards. Moreover, in addition to supporting image compression standards, there is also the Theora image transport plugin [20], that supports Theora encoded messages.

2.1 Bzip2

Bzip2 [1] is a general purpose lossless data compressor. At the core, Bzip2 uses the Burrows-Wheeler block sorting text transform [21], which is a block-sorting lossless data compression

algorithm that works by applying a reversible transformation to a block of input. The block size affects both the compression ratio achieved, and the amount of memory needed for compression and decompression. This algorithm does not itself compress the data, but reorders it to make it easy to compress with. After the transform Bzip2 uses Huffman coding [22], which is a lossless data encoding algorithm that generates optimal variable length prefix codes.

Bzip2 supports block sizes from 100kilobytes to 900kilobytes. ROS uses Bzip2 with a block size of 900kilobytes, which provides the best compression ratios, while increasing memory usage and compression time. Moreover, Bzip2 provides a low-level programming interface library which allows for compression/decompression of data in memory, suitable for embedding on applications. While providing very good compression ratios Bzip2 is actually really slow for speed sensitive systems. [2]

2.2 LZ4

LZ4 is a lossless compression algorithm based on LZ77 [23], a lossless dictionary based compression algorithm that works by replacing repeated occurrences of data with a reference to a single copy of the referred data. One of the main goals of LZ4 is to be simple by design. It achieves reasonable compression ratios at really fast speeds [3]. LZ4 is often used for compression data before transmission due to its high speed. For long term storage LZ4 is unsuitable, as it trades compression efficiency for speed efficiency. A complete reference of the low level LZ4 data format can be found at [4].

2.3 PNG

PNG is an open extensible image standard that supports lossless compression of images. Developed in the mid 1990s, the project started because of patent problems surrounding the GIF file format. Designed to be open and flexible, suitable for internet usage and support many different types of images, PNG has three main features when compared to the older GIF format [24]:

- transparency (alpha channel);
- gamma correction;
- two-dimensional interlacing.

PNG supports filtering, which is a pre-compression step that converts data values into values which are easier to compress. Like predictive coding, a value is predicted for the pixel. This value is then subtracted from the pixel value and the residual is obtained. Although in some cases filtering does not improve compression, generally, PNG obtains much better results by applying filtering before compressing. As a non-realistic example, take into consideration a sequence of bytes which sequentially increments 1 from 1 to 255. By encoding this sequence, the encoder would achieve little compression or none at all. However, with a small modification to the sequence by replacing the bytes by the difference between them and their predecessor,

excluding the first byte, a sequence of 1s would be obtained, which, when encoded, obtains much better compression results [5].

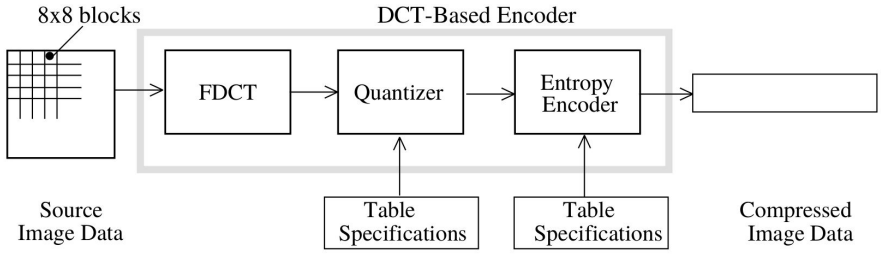
PNG uses 5 types of filters.

- none;
- sub;
- up;
- average;
- Paeth filter based on the algorithm of Alan W. Paeth [25].

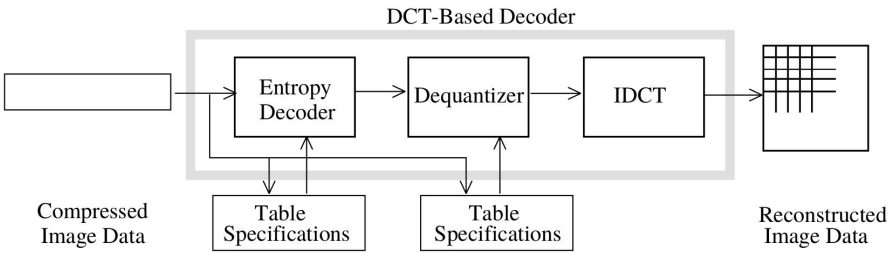
At the core of PNG's compression scheme is Deflate [26], a lossless compression data format that compresses data using a combination of the LZ77 [23] algorithm, a dictionary based compression algorithm, with up to 32kilobytes sliding window sizes and Huffman coding [22].

2.4 JPEG

JPEG is a image compression standard for continuous-tone still images, either gray scale or color. Its name is an acronym for Joint Photographic Experts Group (JPEG). It is a result of a collaboration between the International Standards Organization (ISO) and CCITT. JPEG is one of the most widely known standards for lossy image compression, and like other lossy image compression standards, it takes advantage of the human visual system perception of images. Figure 2.1a contains the baseline encoder block diagram of JPEG and Figure 2.1b contains the decoder block diagram.



(a) JPEG encoder block diagram.



(b) JPEG decoder block diagram.

Figure 2.1: JPEG codec block diagram [6].

JPEG uses a transform coding approach using a Discreet Cosine Transform (DCT) with an uniform mid-tread quantization. This process requires several steps, firstly each unsigned

pixel in the image gets level shifted, i.e., subtracted by $2^{(P-1)}$ where P is the number of bits per pixel, for example, on a 8-bit image, each pixel gets subtracted by 128, secondly the image gets divided into blocks of size 8 x 8, which are then transformed using an 8 x 8 forward DCT. After the transform, the coefficients are quantized and encoded. When there is heavy quantization some visible block artifacts will appear, like with any other block transform coding.

Figure 2.2 contains a practical example of the transform, quantization and the inverse process, adapted from Wallace [6].

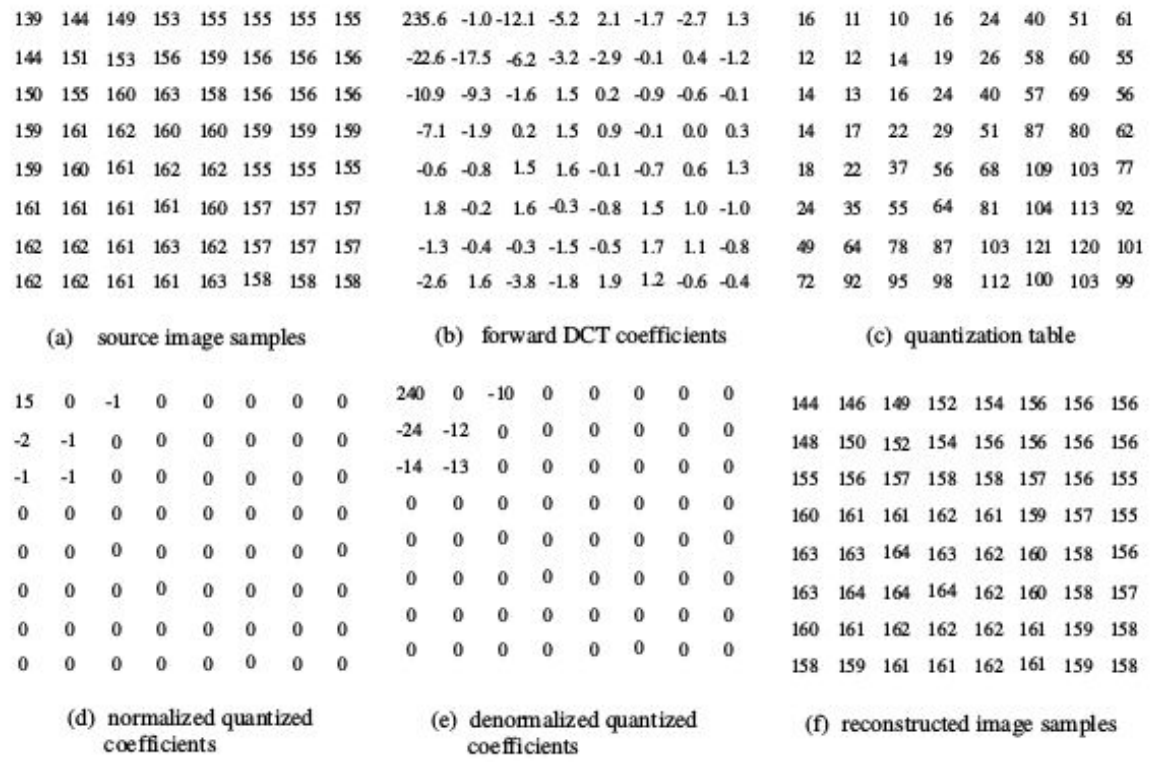


Figure 2.2: JPEG transform, quantization and the inverse example procedure [6].

Matrix (d) on Figure 2.2 contains the quantized coefficients, which are the ones that are encoded using Huffman codes, with a specific order [6] to ensure optimal compression (Zig-Zag).

JPEG supports four modes of operation: sequential, progressive, lossless and hierarchical, however, neither lossless nor hierarchical will be used throughout this thesis. Sequential is considered the *baseline* mode of operation and every enCOder/DECoder (codec) should contain this mode in order to be considered JPEG compatible. The progressive mode consists of the same steps as the *baseline*. However, each image component is encoded in multiple scans rather than in a single scan. By doing this the first scan encodes a recognizable version of the image. The resulting image can be transmitted faster when comparing with the total transmission time. The more scans the more refined the image gets, until it reaches the level of image quality specified by the quantization tables.

2.5 Theora

Theora is a patent free video compression format from the Xiph.org Foundation as part of their Ogg project, originally derived from ON2's VP3 [7].

Video compression can be viewed as image compression with a temporal component since video consists of a timed sequence of images. A video codecs takes advantage of the temporal correlation between sequential frames, removing eventual temporal redundancy. As discussed in Section 2.3, having some sort of prediction mechanism allows for better results when compressing images. In video compression this is taken a step further and, not only there is spatial prediction (like in image compression), but there is also temporal prediction. This means that previous (or even future) frames are used for prediction. The goal of this type of prediction is to reduce the redundancy between frames, by creating a predicted frame and subtracting this from the current frame. The output of this subtraction is a block of residuals.

The more accurate the prediction the more efficient will be the compression. The accuracy of the process can usually be improved by accounting for motion between objects in the reference frame and in the current frame, this is called motion compensation. The prediction errors, or residual blocks, are then transformed before coding, using an 8x8 DCT.

Frames that use spatial compression are called intra-picture frames and frames that use temporal compression are called inter-picture frames. There are more types of frames, however, Theora only supports this ones.

Although plagued by some troubles derived from the original VP3 code base, which led to bad results on comparisons [27], some updates resolved the problem and now it obtains more favorable results when compared with H.264 [28].

2.6 Experimental results

In this section we present several experimental results to show the speed and compression efficiency of the ROS native compression. We also present results on the usage of both ROS native image compression codecs and the ROS native general purpose compressors simultaneously. The results obtained are compared to the efficiency of using raw images. For this experiment we use the P0 Small bag (Appendix A.3, Figure 2.3). This bag contains images with a resolution of 1296x964 pixels, laser scans, odometry data and debugging system messages recorded on a robot at IEETA.

Figure 2.3: Example frames from the P0 Small bag.

The results are obtained by playing the pre-recorded bag and compressing a set with a maximum of 300 images at transport time. Both the Compressed image transport and the Theora image transport ROS plugins are used for compression. For this experiment we use PNG with the compression levels of 1, 3 and 6. JPEG with quality levels of 99 and 95 and Theora with quality levels of 63 and 44.

Figure 2.4 contains a block diagram of the system used to perform the experiments.

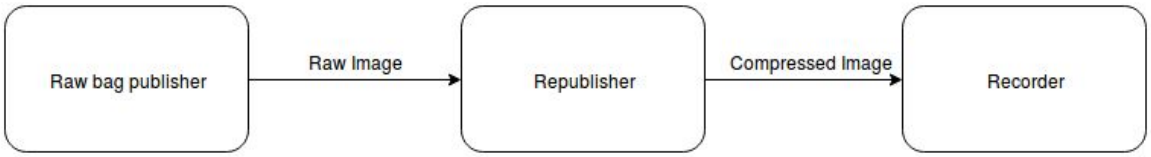


Figure 2.4: Measurement system block diagram.

Table 2.1 presents the compression results of PNG, JPEG and Theora. It also presents the reference value when using raw images.

Analyzing the results presented in Table 2.1 it is possible to divide the data into three major groups according to the compression performance, as expected. The first group is the lossless image compression group, which provides compression ratios between $\approx 2,4$ and $\approx 3,3$. The second group is the lossy image compression group which provides compression ratios up down to ≈ 7 and the last group is the lossy video compression group which provides compression ratios down to ≈ 135.

	Size(MiB)	Ratio
raw	1072,43	1
PNG 1	441,35	2,43
PNG 3	423,08	2,53
PNG 6	326,33	3,29
JPEG 99	151,42	7,08
JPEG 95	63,41	16,91
Theora 63	7,95	134,91
Theora 44	3,79	282,74

Table 2.1: Compression efficiency of PNG, JPEG and Theora during recording, and the reference value when using raw images.

It is possible to estimate the compressed size of the full bag by multiplying the ratio of each standard by the filtered full bag size, which means that with PNG we would be able to obtain a filtered compressed bag with size up to $\approx 1560Mib$ (compression level 1), with JPEG up to$\approx 540Mib$ and with Theora up to $\approx 28Mib$. Moreover, if the bag has 152 seconds of record time and considering each standard obtained size, only taking images into account, for a full hour recording it would occupy up to $\approx 37000Mib$ with PNG, $\approx 12800Mib$ with JPEG and $\approx 660Mib$ with Theora.

Table 2.2 contains the results for the speed efficiency of PNG, JPEG and Theora. According to the results presented in Table 2.2 both JPEG and Theora do not affect the frame rate of the recording however using PNG affects the number of frames per second that can be recorded, even at the lowest level of compression, which makes it unsuitable for systems that require high frame rate and compression during the recording phase.

	Duration(s)	Fps
raw	42,7	7,03
PNG 1	49	6,12
PNG 3	68	4,41
PNG 6	152	1,70
JPEG 99	42,7	7,03
JPEG 95	42,7	7,03
Theora 63	42,4	7,08
Theora 44	42,4	7,08

Table 2.2: Speed efficiency of PNG, JPEG and Theora during recording, and the reference value when using raw images.

Table 2.3 and Table 2.4 contain the results when compressing with the general purpose compressors LZ4 and Bzip2. Both compressors are not only tested with raw images but also with the other compression standards, PNG with compression level 3, JPEG with quality level 95 and Theora with quality level 63.

Looking at Table 2.3 it is possible to conclude that despite providing very fast compression, LZ4 provides little compression when used alone and no significant compression when used

simultaneously with the image compression standards, in fact, it might even increase the size of the bag file.

LZ4	Size(MiB)	Duration(s)	Fps	Ratio
raw	944,27	42,7	7,03	1,14
JPEG 95	63,39	42,7	7,03	16,92
PNG 3	423,22	70	4,29	2,53
Theora 63	7,94	42,4	7,08	135,04

Table 2.3: Image compression and LZ4 simultaneously during recording.

Looking at Table 2.4 it is possible to conclude that using Bzip2 at recording time with raw images, the frame rate drops by about 3 frames per second, while using Bzip2 simultaneously while the other compression standards drops frame rate slightly compared to using only the other compression standards.

Unlike LZ4, Bzip2 provides good compression ratios when used with raw images, however, it still does not compete with a image compression standard like PNG, even at the lowest compression level, and when used simultaneously with the other compression standards, like LZ4, it provides no significant improvements and, in fact, might even aggravate the results.

BZ2	Size(MiB)	time(s)	Fps	Ratio
raw	302,07	42,7	4,43	2,24
JPEG 95	62,57	42,7	7,03	17,14
PNG 3	424,20	74	4,05	2,53
Theora 63	7,89	47,4	6,33	135,98

Table 2.4: Image compression and Bzip2 simultaneously during recording.

It should be noted that, when recording with PNG with a compression level of 6, the end of the bag was reached before PNG could compress the 300 images, which means that, during the duration of the bag, using PNG, we only managed to compress 258 frames. The same thing happened when compressing at record time with Bzip2, the difference being that, with Bzip2, we only managed to compress 189 images for the duration of the bag.

In summary, PNG is not ideal for systems that require lossless compression as it is very slow and provides mediocre compression. As a result, newer and improved state of the art lossless compression standards need to be explored to overcome the weaknesses that PNG introduces in ROS.

JPEG provide sufficient compression, whenever lossy compression is possible, at the cost of image quality. Theora provides good compression ratios, however, like JPEG, at the cost of image quality. Moreover, using either LZ4 or Bzip2 on systems that, for the most part, only record images, provides no benefit as the overhead only increases size. For recording data on systems that record more than images, Bzip2 can be used for compression after the recording is finished, as it provides very good compression ratios on non image data, and LZ4 can be used during recording, as it is very fast at compressing.

CHAPTER 3

State of the art compression algorithms

In this chapter we provide two alternatives to Bzip2 and LZ4. We also present newer and improved state of the art image and video compression algorithms. The ROS native image and video compression algorithms will be put to test against the state of the art algorithms.

3.1 General Purpose Compression Algorithms

Both Bzip2 and LZ4 provide good results for their specific use case. However, if there is a need for better general purpose compressors we provide here two alternatives: Brotli [29] and lbzip2 [30]. Brotli is a general purpose lossless compression algorithm that uses a variant of the LZ77 algorithm [23], Huffman coding [22] and context modeling. It achieves slightly better ratios than Bzip2 at about the same speed [31], [32]. Lbzip2 is a general purpose multi-threaded lossless compressor that is fully compatible with Bzip2. It provides the same ratios as Bzip2 but at a much faster speed, due to the advantage of supporting symmetric multiprocessing [33].

3.2 JPEG2000

JPEG2000 is an image compression standard, created by the Joint Photographic Experts Group (JPEG) committee in the year 2000, with the intention of superseding their DCT-based image compression standard, JPEG. Based on wavelet decomposition, JPEG2000 is composed of many parts, that deal with a variety of applications. However, for this thesis, only the basic image compression part is relevant. Figure 3.1 contains the block diagram of a JPEG2000 encoder.

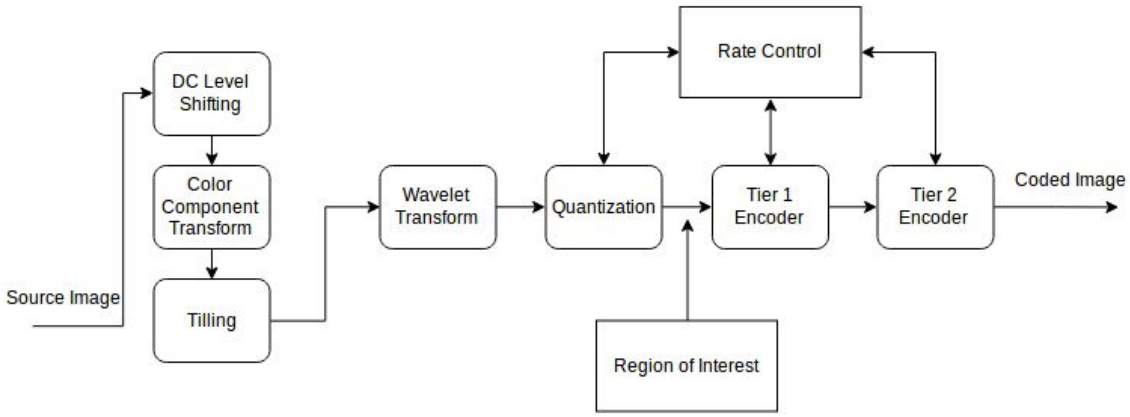


Figure 3.1: JPEG2000 encoder block diagram.

Some of the main features of JPEG2000 are:

- **Superior low bit-rate performance:** when compared to the baseline JPEG;
- **Continuous tone and bi-level image compression;**
- **Large dynamic range of the pixels:** JPEG2000 allows for the (de)compression of images with various dynamic ranges for each color component;
- **Lossless and lossy compression:** JPEG2000 can provide both the lossless and the lossy mode of image compression;
- **Fixed size can be preassigned:** The user can select a desired size for the compressed file;
- **Region Of Interest (ROI) coding:** Sometimes an image contains parts that are of greater importance, with JPEG2000 it is possible to encode parts of an image with higher quality(even lossless) compared to the less important parts;
- **Random access and compressed domain processing:**with JPEG2000 it is possible to manipulate certain areas (or regions of interest) of the image, for example, replace one object in the image with another.

The JPEG2000 compression system is divided into three main phases, the preprocessing phase, the compression phase and the formation of the bitstream.

3.2.1 preprocessing

The preprocessing phase is composed of three optional operations, *DC level shifting*, *multi-component transformation* and *tilling*.

DC level shifting

DC level shifting before transformation, and like in the baseline JPEG standard, the pixel values are level-shifted by 2^{B-1} where B is the number of bits per value used for each component of a pixel. DC level shifting is only applied if the pixel values are unsigned.

Multicomponent transform

Multicomponent transform is the transformation of the image color space into another color space. JPEG2000 supports two multicomponent transformations, a reversible color transform and a irreversible color transform. The *reversible color transform* transforms the original Red, Green and Blue (RGB) values into YUV values and allows for a lossless reversible transform back to RGB and is done using the following formulas:

$$\left\{\begin{array}{c} Y = \left[\frac{R+2G+B}{4}\right] \\ U = B - G \\ V = R - G \end{array}\right\},$$

and the inverse:

$$\left\{\begin{array}{c} R = V + G \\ G = Y - \left[\frac{U+V}{4}\right] \\ B = U + G \end{array}\right\}.$$

The *irreversible color transform* is the same used in baseline JPEG and introduces error because it uses non integer coefficients on the transformation formulas. The formulas used are:

$$\left\{\begin{array}{c} Y = 0,299R + 0,587G + 0,144B \\ U = -0,16875R - 0,33126G + 0,5B \\ V = 0,5R - 0,41869G - 0,08131B \end{array}\right\},$$

and the inverse:

$$\left\{\begin{array}{c} R = Y + 1.402V \\ G = Y - 0.34413U - 0.71414V \\ B = Y + 1.772U \end{array}\right\}.$$

Tilling

Tilling consists of dividing the image into non overlapping tiles (blocks), all the tiles have the same size, except those at the boundaries when image dimension is not a multiple of the tiles dimension. Tile size is variable up to the size of the original image (unlike baseline JPEG). Each image component is represented at the same tile, i.e., on a gray scale image a tile contains one component, on a multi component image a tile contains all components. For very low bit-rate compression some visible artifacts may appear due to heavy quantization, like in JPEG or any other block transform coding.

Figure 3.2: Example of the blocking artifacts [34].

3.2.2 Compression

After the preprocessing phase comes the compression, which can be divided into three sequential phases, the *Discreet Wavelet Transform (DWT)*, *quantization* and *entropy encoding*. Firstly the DWT coefficients are calculated and *quantized* using a midtread quantizer and then they are *coded* using the embedded block coding with optimized truncation (EBCOT) [35] algorithm. According to T. Acharya and P.-S. Tsai: "The main drawback of the JPEG2000 standard compared to current JPEG is that the coding algorithm is much more complex and the computational needs are much higher." [8].

3.3 JPEG-LS

JPEG-LS [9] is a standard for lossless and near-lossless compression of continuous tone still images. It has been developed by the Joint Photography Experts Group(JPEG), like JPEG and JPEG2000, with the aim of providing better compression efficiency than lossless JPEG whilst keeping the standard with low complexity. Generally JPEG-LS provides better compression ratios and coding speeds when compared with PNG [36], [37]. When the initial proposals for the new lossless compression standard were requested the following requirements were defined[38]:

1. Provide lossless and near-lossless compression.
2. Target 2 to 16 bit still images.
3. Applicable over wide variety of content.
4. Should not impose any size or depth restrictions.
5. Field of applicability include Medical, Satellite, Archival etc.
6. Implementable with reasonable complexity.
7. Significantly better than current lossless standards.

8. Capable of working with a single pass through data.

JPEG-LS is based on the LOw COmplexity LOssless COmpression for Images (LOCO-I) algorithm that, like PNG, relies on prediction however, unlike PNG, uses context modeling of the residuals prior to encoding. Figure 3.3 contains the basic block diagram of JPEG-LS.

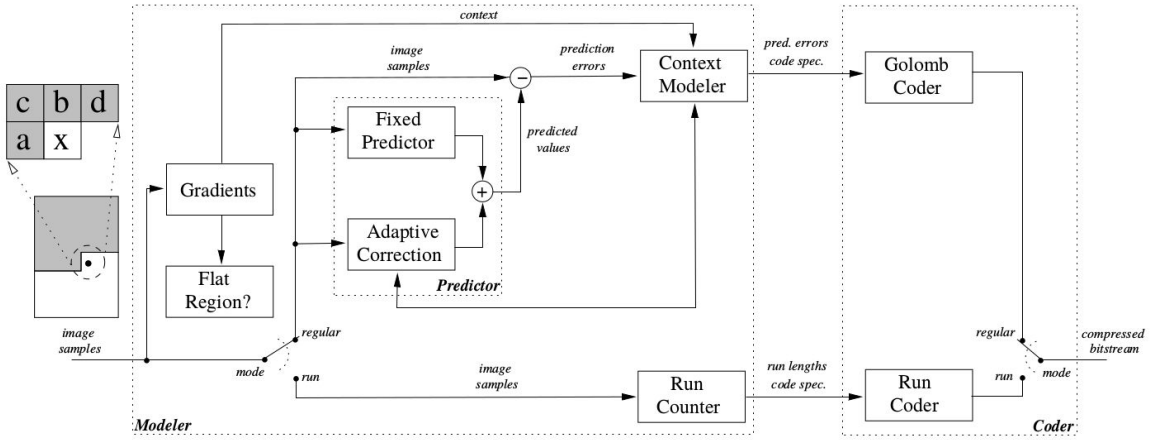


Figure 3.3: JPEG-LS block diagram [39].

3.3.1 Prediction

For prediction a fixed predictor is used , the Median Edge Detection (MED) predictor [40], which adapts whenever it finds local edges. This predictor takes into account vertical and horizontal edges, by examining the North, West and Northwest neighbors of the current pixel.

3.3.2 Context modeling

JPEG-LS makes use of a very simple context model where each pixel is assigned to a context. Contexts are determined by three values based on the neighboring pixels of the pixel to be predicted. These gradients represent an estimate of the local gradient, thus capturing the level of activity (smoothness, edginess) allowing JPEG-LS to exploit higher-order structures such as texture patterns and local activity in the image for further compression gains. A more detailed look at context modeling in JPEG-LS is available at [39].

3.3.3 Coding

In JPEG-LS, prediction errors are encoded using a special case of Golomb codes [41] which is also known as Rice coding [42]. These codes are optimal for data that follows a two-sided geometric distribution centered at zero, in which the occurrence of small values is significantly more likely than large values.

On low entropy images or image regions Rice codes are not optimal as the best coding rate achievable is 1 bit per symbol. JPEG-LS uses an alphabet extension mechanism, in order to effectively code this regions, that switches to a run-length mode when a uniform region is encountered.

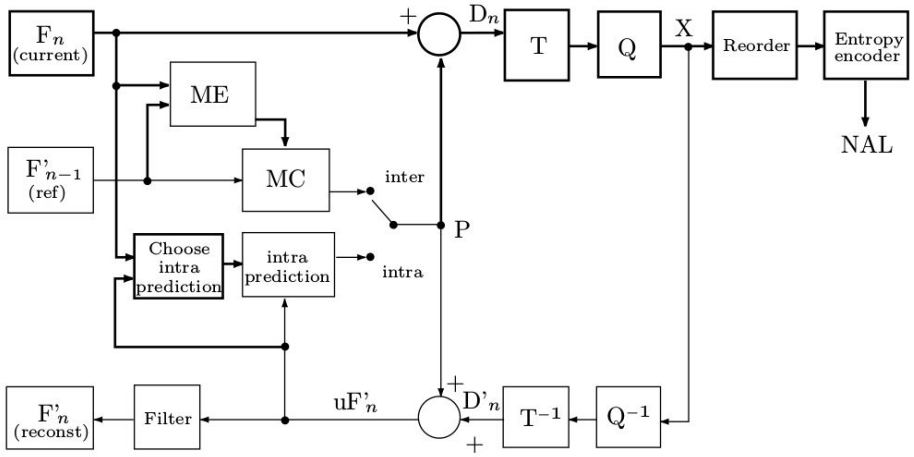
3.3.4 Near-Lossless Compression

Besides lossless compression, JPEG-LS also provides a lossy mode of operation where the maximum absolute error between the original and reconstructed values can be controlled by the encoder, known as the near-lossless mode. This mode will not be used throughout this thesis.

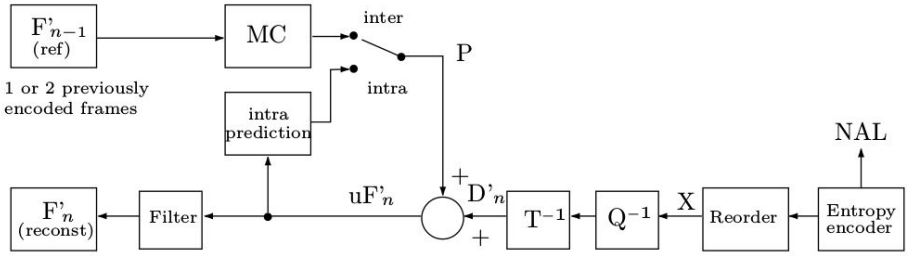
3.4 H.264

H.264, also known as Advanced Video Coding (AVC), is a video compression standard, created by the Joint Video Team (JVT), consisting of VCEG (Video Coding Experts Group) of ITU-T (International Telecommunication Union—Telecommunication standardization sector), and MPEG (Moving Picture Experts Group) of ISO/IEC in 2003.

It is a widely used standard that originated from the need for higher compression efficiency, when compared to the older standards, the need for support of special video applications, like DVD storage, video conferencing, video broadcasting and streaming, but also to achieve greater reliability. Figure 3.4a contains a typical encoder block diagram of H.264 while Figure 3.4b contains the typical decoder block diagram of H.264.



(a) Typical H.264 encoder block diagram.



(b) Typical H.264 decoder block diagram.

Figure 3.4: H.264 codec block diagrams [11].

H.264 defines four main profiles, baseline, extended, main and high. From these profiles

several other profiles are also defined. Each profile contains a set of encoding functions, and each is intended for different video applications. The baseline profile is designed for video telephony, video conferencing, and wireless communication. The extended profile is intended for streaming video and audio. The main profile is designed to handle television broadcasting and video storage. The high profile is designed for achieving significant improvement in coding efficiency for higher fidelity material, such as high definition TV/DVD. The main components of each profile can be found at Figure 3.5. A more in depth description of each profile can be found at [11], [12] and for the high profile [13].

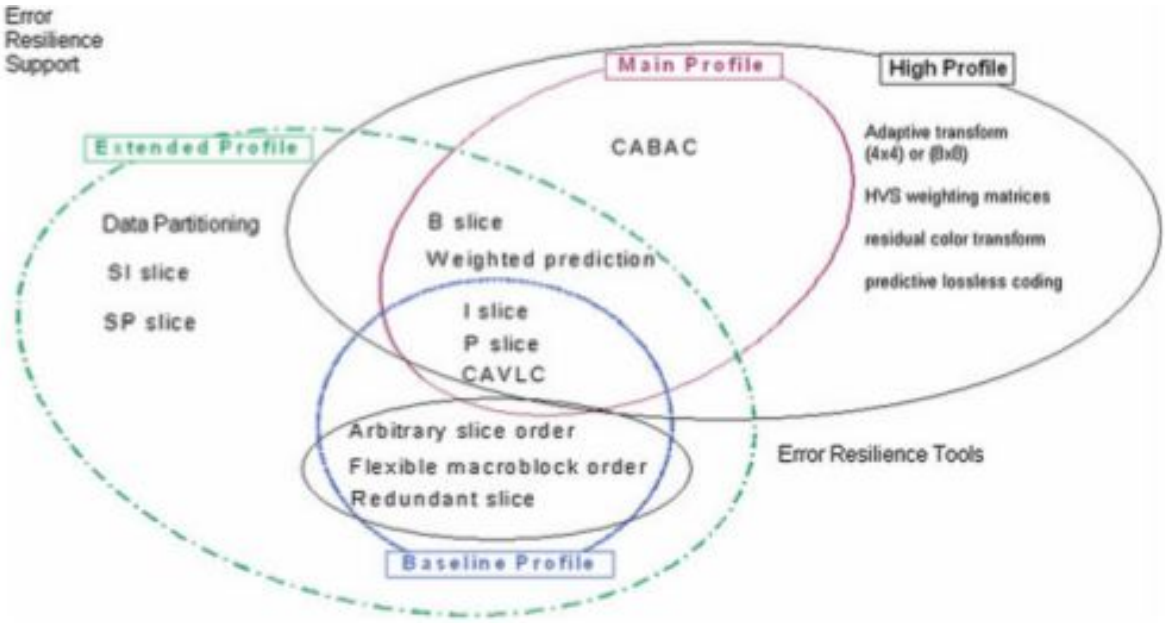


Figure 3.5: H264 profile components [13].

In H.264 each picture is divided into a series of Macroblocks (MBs). A MB corresponds to a block of 16x16 pixels which can be subdivided into smaller blocks and partitioned into luma and chroma components to be processed separately. MBs can be considered the basic coding unit in H.264.

To aid in transmission or streaming H.264 provides some high level networking headers and flags. However, the coded video data is stored in units, known as slices. Each slice is divided into the slice header and the slice data, which is a series of coded MBs. There are several types of slices:

- **I slices** contain intra predicted macroblocks.
- **P slices** contain intra or inter predicted macroblocks from only one reference.
- **B slices** contain intra or inter predicted macroblocks from one or two references (biprediction). Type B slices are used in the main profile.
- **SP and SI slices** are specially-coded slices that enable, among other things,efficient switching between video streams and efficient random access for video decoders. They are used in the extended profile [43].

A overview of the high level syntax architecture of H.264 is found on Figure 3.6.

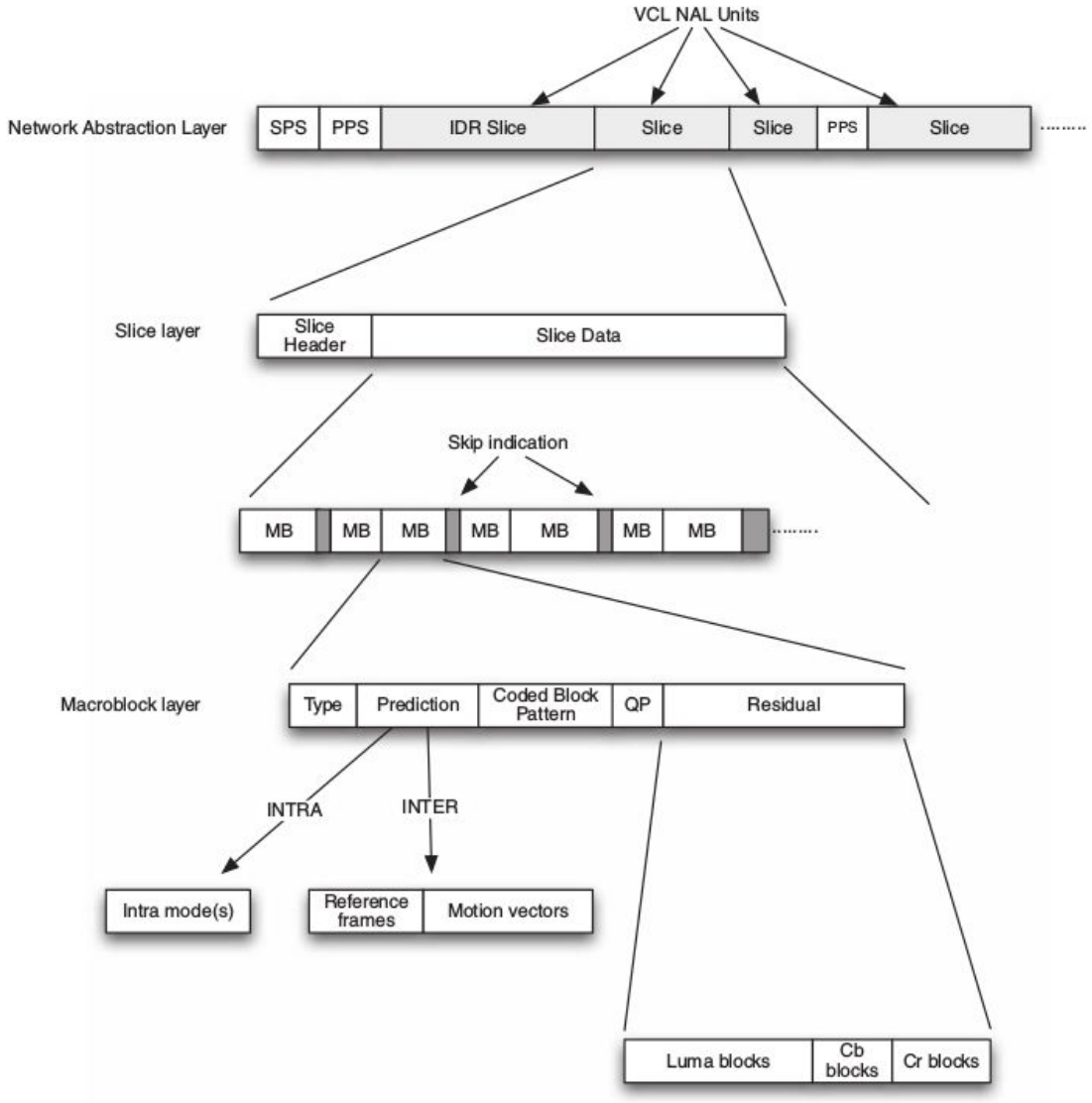


Figure 3.6: High level overview of a H.264 bitstream hierarchy [44].

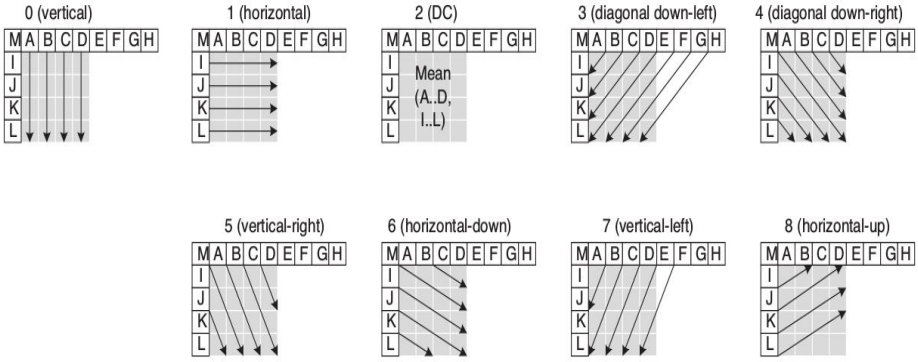
3.4.1 Intra-picture prediction

As previously mentioned, intra-picture prediction exploits the spatial correlation among pixels, like image compression.

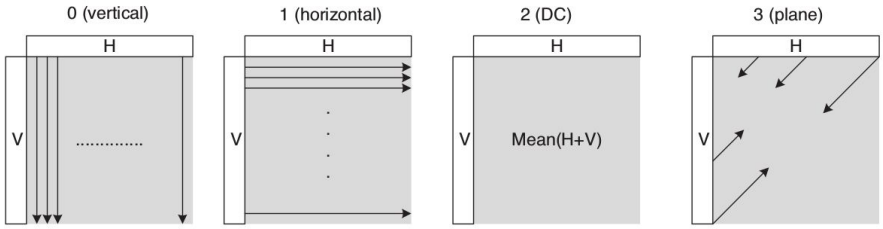
In H.264 the image is divided into blocks, which will be subtracted by a prediction block P, obtaining the residuals. P is based on the previously encoded block, exploiting the spatial correlation between sequential blocks.

In H.264 intra-picture prediction is divided into three types. The luma 4x4 block, the luma 16x16 macroblock and the chroma prediction. The luma block prediction modes consist of 8 directional modes for edge detection and a mean-based method (DC prediction). The luma macroblocks prediction modes only take into account horizontal or vertical edges, the same mean-based method (DC prediction) available for the luma blocks and a planar method that detects areas of smoothly varying luminance. The chroma components prediction is similar to

the luma macroblock prediction modes. Figure 3.7a contains a graphical representation of the 4x4 luma blocks prediction modes. Figure 3.7b contains a graphical representation of the 16x16 luma blocks and chroma blocks prediction modes.



(a) 4x4 luma block prediction modes.



(b) 16x16 luma block and chroma block prediction modes.

Figure 3.7: H.264 intra-picture prediction modes [44].

3.4.2 Inter-picture prediction

Inter-picture prediction, unlike intra-picture prediction, exploits the temporal redundancy in a sequence of frames, the temporal correlation is reduced by the use of motion estimation and compensation algorithms. Using a block-based motion compensation, H.264 creates a prediction model from one or more previously encoded video frames.

The most important differences between inter-picture prediction in H.264 and the older standards is the support for a range of block sizes (from 16x16 to 4x4) and more precise subsample motion vectors (quarter-sample resolution for the luma component). An example of motion estimation with integer and subpixel reference blocks can be found at Figure 3.8.

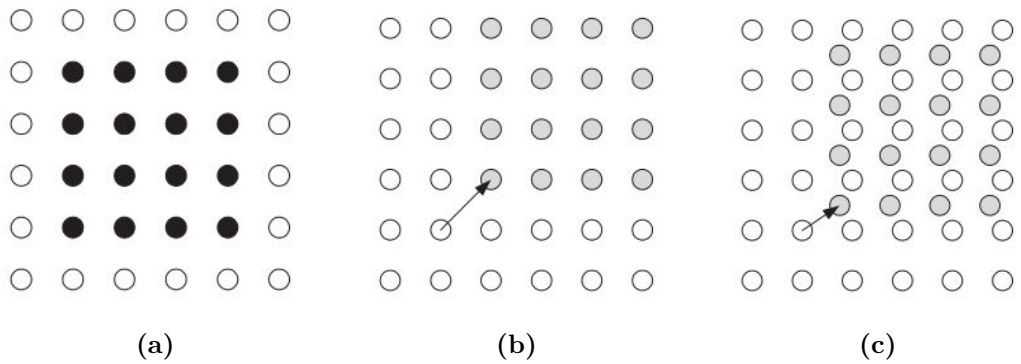


Figure 3.8: H.264 inter-picture prediction motion estimation [44]. (a) contains a 4x4 block in the current frame. (b) contains the reference block in the reference frame. (c) contains a subpixel reference block.

3.4.3 Transform and Quantization

H.264 uses different transform methods, depending on the type of residual data to be transformed. For the DC coefficient blocks obtained from the prediction of intra luma macroblocks and from the prediction of chroma DC coefficients a Hadamard transform [45] is used, for everything else a DCT-based transform is used. The output coefficients of the transform are then quantized. Quantization reduces the precision of the transform coefficients according to the quantization parameter (QP).

3.4.4 Deblocking filter

One of the new features that H.264 contains when compared to the older standards is the deblocking filter. The deblocking filter consists of a filter that is applied to each decoded macroblock to reduce blocking artifacts that are introduced after the transform is applied. The deblocking filter is applied after the inverse transform in the encoder and in the decoder. The filter smooths block edges, improving the appearance of decoded frames. The filtered image is the one that is used for motion-compensated prediction of future frames which might improve the compression efficiency, because the filtered image is often a better approximation of the original frame than the blocky image obtained from the inverse transform. Figure 3.9 contains the effect of the deblocking filter on a grayscale image.

(a) Without deblocking filter. **(b)** With deblocking filter.

Figure 3.9: Effect of the deblocking filter [12].

3.4.5 Coding

H.264 uses several coding methods depending on the profile chosen or data that is going to be coded. For the baseline profile, high level headers and flags are encoded using Fixed Length Codes (FLC). For the quantized coefficient values Context-Adaptive Variable Length Coding (CAVLC) is used. For the rest of the data, such as low level flags and parameters, exponential Golomb codes are used. For the main profile, instead of using CAVLC and exponential Golomb codes, H.264 uses Context-Adaptive Binary Arithmetic Coding (CABAC) [46].

3.5 H.265

H.265, also known as High Efficiency Video Coding (HEVC), is a state of the art video compression standard, the successor of the widely used AVC standard (H.264). Like H.264 it is the result of a collaboration between the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). H.265 can be considered an extension of the older H.264 standard, with improvements on the coding efficiency achieved by further exploring existing techniques but with the cost of increasing the complexity of the encoder. Despite having some new design aspects that improve flexibility for operation on several applications and network environments while improving robustness to data losses, the high level syntax architecture used in H.264 has mostly been retained in H.265. Figure 3.10 contains a typical H.265 encoder block diagram.

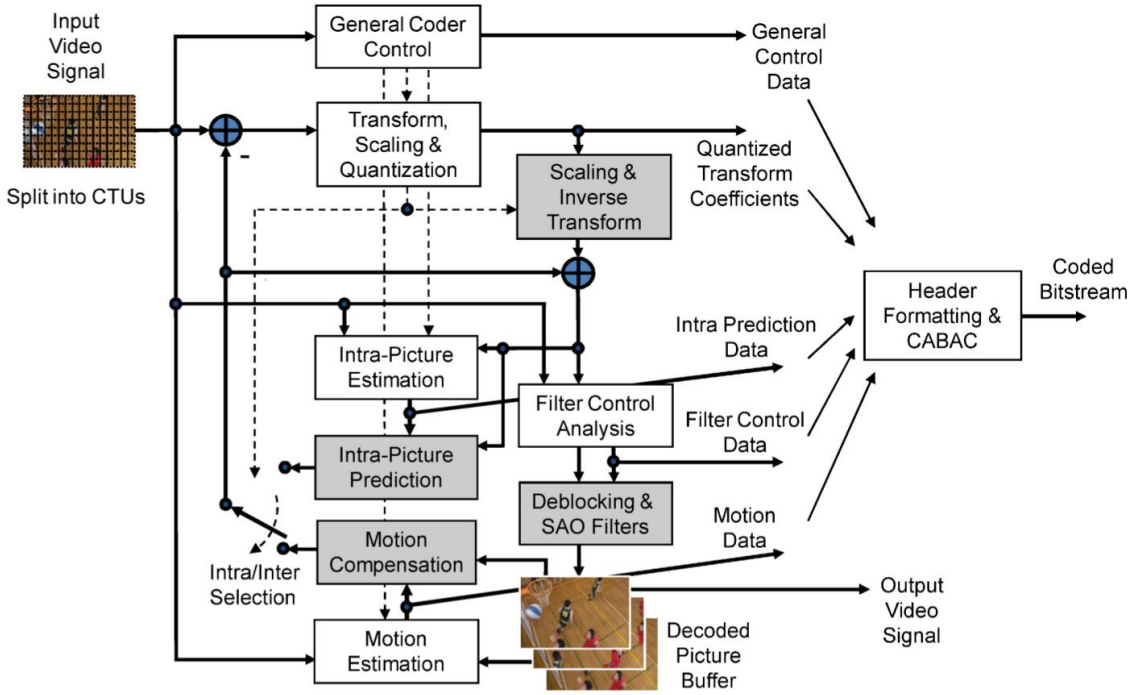


Figure 3.10: Typical H.265 encoder block diagram [14].

H.265 supports a broad range of applications. This is possible due to some variation in capabilities and functionalities in H.265. This variation is handled by specifying multiple profiles. In many applications, it is currently neither practical nor economical to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile. In order to deal with this problem each profile is subdivided into levels. A level is a specified set of constraints, imposed on values of the syntax elements in the bitstream. These constraints may be simple limits on values, i.e., levels can establish a limit on the picture resolution, frame rate, buffering capacity, and other aspects that are matters of degree rather than basic feature sets. However, this does not solve all problems. Professional environments often require much higher bit rates for better quality than consumer applications, for the same level. This was solved by introducing the concept of tiers, which define the bit rate that the level is able to handle. Several levels in H.265 have both a Main tier and a High tier, based on the bit rates they are capable of handling. H.265 defines several profiles, levels and tiers. For the first version of this standard only three profiles existed, the main profile, the main still picture profile and the main 10 profile. Following the newer releases of H.265, several other profiles were added that extend or improve upon these three profiles.

One of the new changes brought by H.265 is the discontinuation of Macroblocks (MBs) and the introduction of Coding Tree Units (CTUs), which conceptually are similar to MBs but with support for a wider range of sizes (16x16 up to 64x64) and improved sub-partitioning. A CTU represents the basic coding unit in H.265. Each CTU can then be partitioned into smaller blocks, the Coding Units (CUs).

3.5.1 Intra-picture prediction

H.265 has two intra-picture prediction categories, the *Angular prediction* methods form the first category and provide the encoder with the possibility to model structures with angular edges, the second category is composed of *planar prediction* and *DC prediction* that provide the possibility to estimate smooth image structures. The angular prediction methods are similar to the directional modes from H.264 but with support to a wider range of directions (angles). The planar prediction and the DC prediction are the same as the ones used in H.264 with the same name. The total number of intra prediction modes is thirty five, planar taking the first slot, DC the second and angular prediction taking the remaining 33 slots. Figure 3.11 contains a graphical representation of the intra-picture prediction modes of H.265.

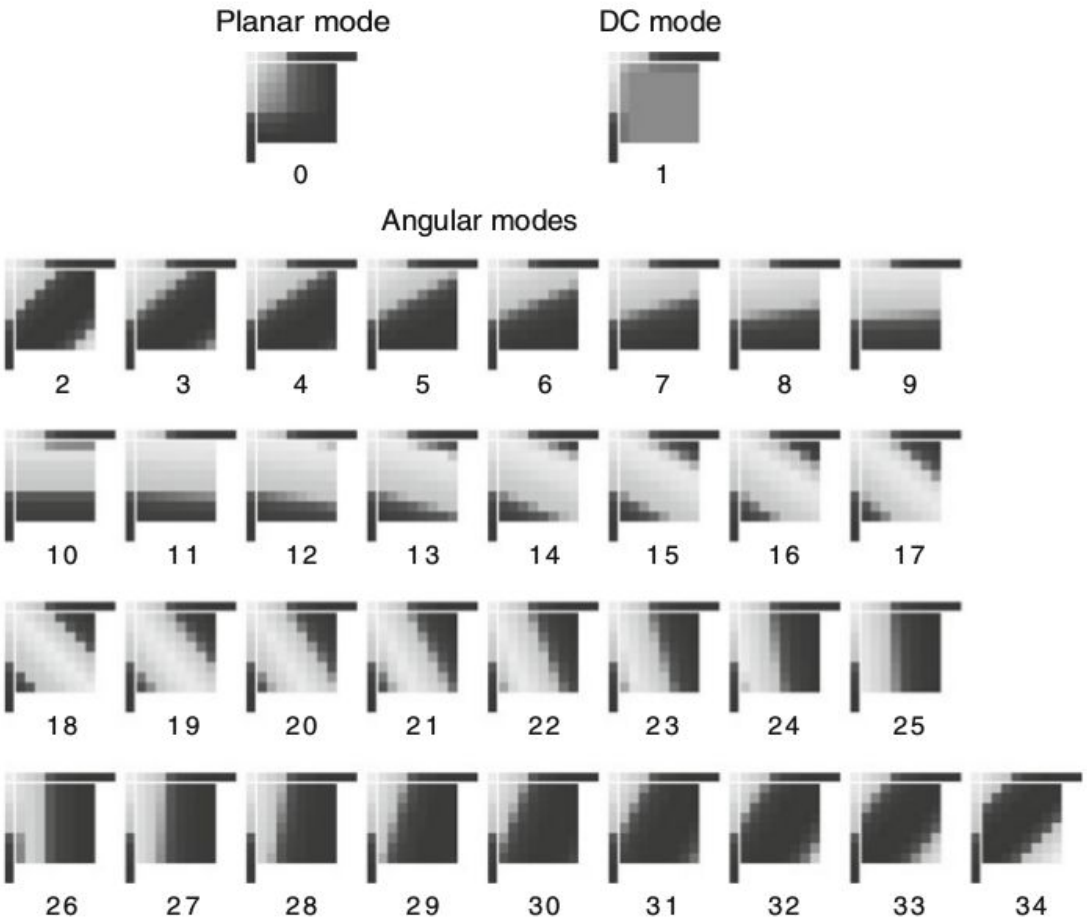


Figure 3.11: Intra-picture prediction in H.265 [15].

3.5.2 Inter-picture prediction

Inter-picture prediction in H.265 is an improvement from the older H.264. A significant improvement, besides the greater flexibility in partitioning the blocks, is the ability to encode motion vectors with much greater precision, giving a better predicted block with less residual error. There is also the introduction of a new tool called Advanced Motion Vector

Prediction (AMVP) that improves the predictive coding of the motion vectors. A detailed list of all improvements in inter prediction from the older H.264 can be found at [15]. Figure 3.12 contains a block diagram of HEVC inter-picture prediction module.

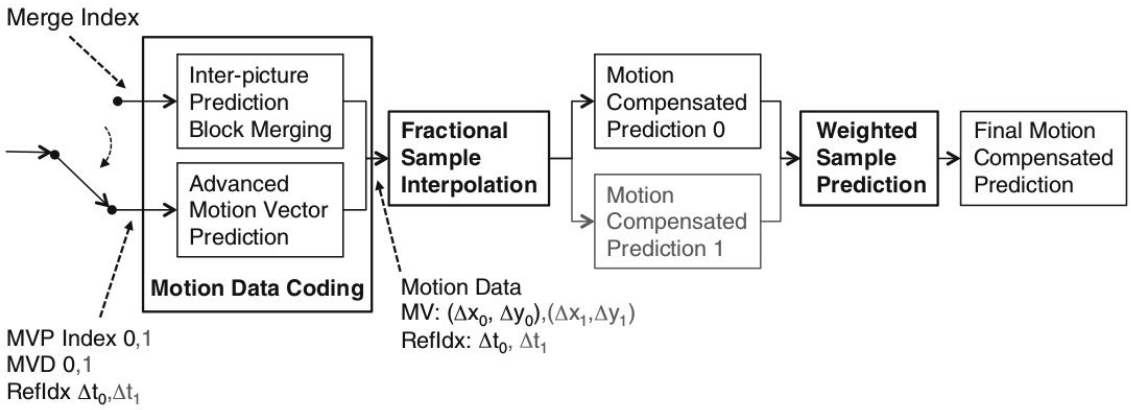


Figure 3.12: Inter prediction-picture in H.265 [15] ,blocks in faded gray represent the bi-prediction path.

3.5.3 Transform and Quantization

H.265 specifies two-dimensional transforms of various sizes from 4x4 to 32x32 that are finite precision approximations to the DCT. In addition, H.265 also specifies an alternate 4x4 integer transform based on the Discreet Sine Transform (DST) for use with 4x4 luma Intra prediction residual blocks. The H.265 quantizer design is similar to that of H.264.

3.5.4 Filters

Deblocking filter

The deblocking filter in H.265 is similar to the deblocking filter in H.264. However, some changes were made to improve efficiency and support for parallel processing. At the cost of subjective quality, the deblocking filter is only applied to the block boundaries that lie at the luma and chroma sample positions that are multiples of eight. This reduces computational complexity which improves efficiency. It also improves parallel processing by preventing cascading interactions between nearby filtering operations. In H.264 the deblocking filter is applied to vertical or horizontal edges of 4x4 blocks in a MB.

Sample Adaptive Offset filter

One of the new features introduced in H.265 is the Sample Adaptive Offset filter (SAO). The SAO consists of a filter that is designed to remove or reduce the mean sample distortion of a region. It works by first classifying the region samples into multiple categories with a selected classifier, obtaining an offset for each category, and then adding the offset to each sample of the category, where the classifier index and the offsets of the region are coded in the bitstream.

The SAO is applied to the output of the deblocking filter. A more detailed introduction to the Sample Adaptive Offset filter (SAO) is available at [47].

3.5.5 Coding

H.265 uses the same entropy coding method that the main profile of H.264 uses, Context-Adaptive Binary Arithmetic Coding (CABAC) [46] , with a mixture of zero-order exponential golomb codes [44] .

3.5.6 BPG

BPG is a file format for compressed images, based on a subset of the intra-picture mode of H.265. H.265 contains several profiles defined for still-picture coding. These profiles use the intra-frame encoding with various bit depths and color formats. Some of those profiles are the Main Still Picture, Main 4:4:4 Still Picture, and Main 4:4:4 16 Still Picture profiles. BPG is a wrapper for the Main 4:4:4 16 Still Picture Level 8.5 H.265 profile, with only up to 14 bits per sample. BPG uses a slightly different bitstream format that is based on H.265 but strips all the unnecessary headers for image compression. A complete specification can be found here [10].

3.6 Experimental Results

Most of the codecs used throughout this thesis provide extensive customization parameters. The idea of testing all the possibilities is unrealistic, so a subset of the codecs functionality was chosen for these experiments. The subset was the following:

- **PNG** with compression level of 1, 3, 6 and 9.
- **JPEG** with qualities 80, 90, 95 and 99.
- **Theora** with quality 10 and 7 (corresponds to qualities 63 and 44 used in the previous chapter).
- **JPEG-LS** with default settings.
- **JPEG2000** with compression ratios of 5x, 10x, 20x and 40x and also lossless mode.
- **H.264 and H.265** with a Constant Rate Factor (CRF) of 13 and 28, the lossless mode was also tested, each parameter was tested with two presets, medium and ultrafast.
- **BPG** with a quantizer parameter (similar to CRF) of 13 and 28, the lossless mode was also tested, each parameter was tested with a compression level of 1 and 5 (fast and medium).

To determine the efficiency of the codecs we execute a script that first compresses and decompresses the dataset sequentially with all the combinations provided in the subset above and lastly calculates the average Peak Signal to Noise Ratio (PSNR) for all the targets.

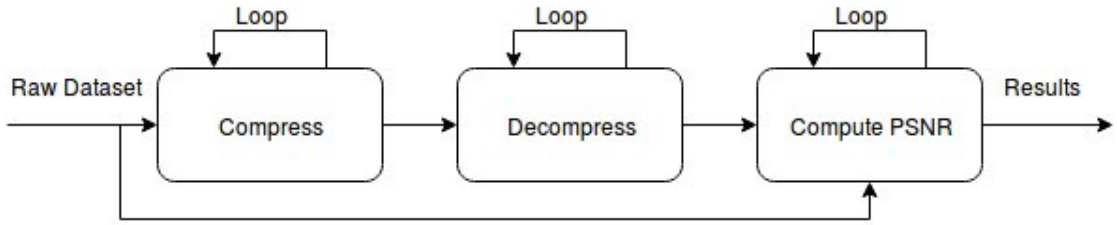


Figure 3.13: Simplified block diagram of the experiment procedure.

We focus on two important metrics when determining the efficiency of the compression standards, the speed efficiency and the compression efficiency. In order to determine the speed efficiency of a compression standard both the compression and decompression times are measured. All compression and decompression times are obtained with the Linux time utility. The experiment is executed with maximum priority to avoid context switches and other interruptions that would change the fairness of the experiment. The command to execute with maximum priority is presented bellow:

```
$ sudo chrt -f 99 bash experiment.sh &> result.txt
```

After the compression and decompression times are recorded, the size (in MiB) is taken into account and the average of bits per pixel (bpp) is calculated for each standard. For obtaining the relation between the number of bits per pixel (bpp) and PSNR a broader subset of codec qualities for each lossy standard was used:

- **JPEG** with a quality range of 10:100 with a step of 10.
- **Theora** with a quality range of 1:10 with a step of 3.
- **JPEG2000** with compression ratios of 5x, 10x, 20x, 40x, 80x and 100x.
- **H.264 and H.265** with a CRF range of 1:51 with a step of 5 at the ultrafast preset.
- **BPG** with a CRF range of 1:51 with a step of 5 and compression level 1.

The experimental results are discussed in three parts. The first part considers the results for the lossless codecs, the second part considers the results for the lossy codecs and the third part contains the relation between the number of bits per pixel (bpp) and the average PSNR of the lossy codecs. For this experiment we use three image datasets: The Alboi moving (Appendix B.1) and Alboi mixed (Appendix B.2) image datasets and the P0 Large image dataset (Appendix B.4).

Both the Alboi moving and Alboi mixed image datasets contain image sequences from the Alboi bag (Appendix A.1, Figure 3.14) with a resolution of 1624x1224 pixels. The P0 Large image dataset contains images from the P0 Large bag (Appendix A.2, Figure 3.15) with a resolution of 1296x964 pixels. The Alboi moving image dataset contains image sequences with movement. The Alboi mixed image dataset contains mixed image sequences, the first half consists of still image sequences and the second half of motion image sequences. The P0 Large image dataset contains still image sequences.

Figure 3.14: Example frames from the Alboi bag.

Figure 3.15: Example frames from the P0 Large bag.

3.6.1 Lossless

Table 3.1 presents the speed efficiency results for the lossless modes of the compression standards. According to the experimental results presented in Table 3.1 the fastest standard is H.264, followed by PNG with low compression levels and H.265 ultrafast preset. Moreover, JPEG2000, H.265 medium preset mode, JPEG-LS, BPG with low compression level and PNG with medium compression levels provide mediocre speeds while BPG with medium compression levels and PNG with high compression levels provide poor speeds.

In Chapter 3.3 it is mentioned that generally JPEG-LS provides much faster compression speeds than PNG, while this is true for higher compression levels, our experiments show that, with the codec that we used to represent JPEG-LS, PNG with low compression levels is much faster that JPEG-LS.

	Alboi moving		Alboi mixed		P0 Large	
	C Time(s)	D Time(s)	C Time(s)	D Time(s)	C Time(s)	D Time(s)
H.264 uf	8	15	8	14	5	9
H.264 md	73	22	70	20	42	11
PNG 1	85	35	97	33	50	22
H.265 uf	127	41	106	39	51	20
PNG 3	115	34	141	33	64	22
JPEG2000	223	161	204	147	135	94
H.265 md	262	35	252	37	125	17
JPEG-LS	329	315	304	290	178	173
PNG 6	364	35	324	33	183	22
BPG fast	388	104	373	97	223	59
BPG md	506	102	493	97	296	59
PNG 9	1087	34	1824	32	732	21

Table 3.1: Speed efficiency of the lossless codecs.

Table 3.2 shows the results for the compression efficiency of the lossless standards in size(MiB) and bits per pixel(bpp). Analyzing the results presented in Table 3.2 it is possible to conclude that BPG is the most compression efficient, followed by JPEG-LS and H.264 medium preset. The worst results belong to PNG, that, when compared to the most efficient standard tested, has more than double bits per pixel. JPEG2000 provides mediocre results in all situations. As expected, both H.264 and H.265 show improvement when we transition to datasets with little to no movement, due to the temporal prediction, which none of the image compression standards use.

	Alboi moving		Alboi mixed		P0 Large	
	Size(MiB)	bpp	Size(MiB)	bpp	Size(MiB)	bpp
BPG md	292,63	4,12	256,44	3,61	156,87	3,51
BPG fast	292,91	4,12	256,72	3,61	156,99	3,51
JPEG-LS	424,32	5,97	373,52	5,25	229,57	5,14
H.264 md	437,44	6,15	373,56	5,25	218,31	4,89
H.265 md	515,05	7,25	420,95	5,92	239,46	5,36
H.265 uf	550,40	7,74	440,59	6,20	246,70	5,52
H.264 uf	533,20	7,50	462,46	6,51	273,25	6,12
JPEG2000	551,32	7,76	495,71	6,98	307,68	6,89
PNG 9	597,41	8,40	528,07	7,43	325,34	7,28
PNG 6	606,68	8,53	546,57	7,69	332,23	7,44
PNG 3	673,02	9,47	606,73	8,53	371,90	8,32
PNG 1	708,91	9,97	644,27	9,06	391,38	8,76

Table 3.2: Compression efficiency of the lossless codecs.

3.6.2 Lossy

Table 3.3 presents the speed efficiency results for the lossy modes of the codecs used to represent the compression standards. According to the experimental results presented in Table 3.3, and when comparing similar quality levels, the fastest lossy standard is H.264, followed by JPEG and H.265. Theora shows little speed difference when changing quality levels. Both BPG and JPEG2000 are slow. Looking at decompression times, all standards provide similar results with the exception of BPG and JPEG2000 which require substantially more time to decompress.

Table 3.4 shows the results for the compression efficiency of the lossy standards in size (MiB) and bits per pixel (bpp). From the table we can observe, comparing similar qualities, that video compression standards, for the most part, provide the best compression when compared to image compression standards. Moreover, the efficiency of the video compression standards increases whenever movement in the image datasets reduces. Of all the video compression standards H.265 provides the best results followed by H.264 and lastly Theora. When the dataset contains high motion, the image compression standards provide very competitive results. BPG and JPEG2000 provide the best results of all the image compression standards, with BPG providing slightly better results than JPEG2000.

	Alboi moving		Alboi mixed		P0 Large	
	C Time(s)	D Time(s)	C Time(s)	D Time(s)	C Time(s)	D Time(s)
H.264 28 uf	5	11	4	10	2	7
H.264 13 uf	6	13	6	12	4	7
JPEG 80	15	19	13	19	10	11
JPEG 50	15	19	16	16	10	16
H.265 28 uf	28	12	17	12	6	7
JPEG 90	21	21	19	20	13	12
JPEG 95	25	22	22	22	16	13
H.264 28 md	38	12	23	11	8	7
H.265 13 uf	48	13	33	16	12	8
JPEG 99	37	27	33	26	22	15
Theora 7	62	11	53	10	31	7
H.265 28 md	83	12	55	14	21	7
Theora 10	65	11	56	11	33	7
H.264 13 md	85	14	73	13	37	8
H.265 13 md	148	15	123	19	58	9
BPG 28 fast	177	41	168	38	109	23
BPG 28 md	210	44	199	51	133	28
JPEG2000 40x	230	55	212	48	144	31
JPEG2000 20x	234	59	218	57	147	39
JPEG2000 10x	242	76	223	76	150	51
JPEG2000 5x	253	113	239	111	157	75
BPG 13 fast	300	75	271	68	174	41
BPG 13 md	381	75	339	72	220	45

Table 3.3: Speed efficiency of the lossy codecs.

	Alboi moving			Alboi mixed			P0 Large		
	Size(MiB)	bpp	PSNR	Size(MiB)	bpp	PSNR	Size(MiB)	bpp	PSNR
H.265 28 md	3,23	0,05	39,42	0,84	0,01	41,81	0,18	0,003	41,16
H.265 28 uf	2,59	0,04	38,87	1,38	0,02	41,05	0,14	0,004	40,19
H.264 28 md	6,71	0,09	40,30	2,62	0,04	42,38	0,35	0,01	41,57
H.264 28 uf	10,38	0,15	37,40	3,82	0,05	40,89	0,68	0,02	41,08
Theora 7	14,83	0,21	41,31	6,16	0,09	42,97	1,47	0,03	41,85
BPG 28 md	11,15	0,16	41,80	8,33	0,12	43,20	8,29	0,19	41,27
BPG 28 fast	11,24	0,16	41,77	8,38	0,12	43,17	8,29	0,19	41,23
Theora 10	27,44	0,39	42,40	11,6	0,17	43,56	4,34	0,10	42,93
H.265 13 uf	24,32	0,34	42,28	11,80	0,17	43,48	4,49	0,10	42,65
JPEG 50	19,74	0,28	39,74	15,1	0,22	41,15	12,97	0,29	39,21
JPEG 80	36,91	0,52	41,56	27,68	0,39	42,74	21,17	0,47	40,58
H.265 13 md	48,92	0,69	43,36	33,20	0,47	44,71	15,73	0,35	44,21
JPEG2000 40x	45,78	0,64	43,42	45,78	0,65	44,79	29,38	0,66	43,83
JPEG 90	87,26	1,23	44,55	65,83	0,93	45,54	48,23	1,08	44,30
BPG 13 md	104,28	1,47	45,23	75,38	1,06	46,18	54,74	1,23	44,19
BPG 13 fast	104,95	1,48	45,22	76,22	1,07	46,16	55,36	1,24	44,19
H.264 13 md	105,73	1,49	44,36	80,16	1,13	45,35	40,35	0,90	44,76
JPEG2000 20x	87,97	1,24	44,96	87,97	1,24	46,45	56,33	1,26	45,38
JPEG 95	143,12	2,01	45,57	115,56	1,63	46,54	80,01	1,79	45,73
H.264 13 uf	131,30	1,85	43,21	125,29	1,76	44,95	84,11	1,88	45,26
JPEG2000 10x	173,52	2,44	46,37	173,52	2,45	47,92	110,24	2,47	47,34
JPEG 99	368,17	5,18	48,47	302,79	4,26	48,91	201,38	4,51	49,19
JPEG2000 5x	344,60	4,85	50,03	344,61	4,85	51,78	217,14	4,86	51,52

Table 3.4: Compression efficiency and image quality of the lossy codecs.

3.6.3 Relation between bits per pixel and PSNR

It is clear from the results presented in this chapter that lossy compression provides better compression and speed efficiency when compared to lossless compression. Of all the tested standards, in general, the video compression standards provide the best results. However, as only a small subset of each standard is tested and in order to obtain a more detailed study on the relation between bits per pixel and PSNR, a wider range of compression levels needs to be chosen, going from high to low compression. For H.265, BPG and H.264 we use a range of CRF from 1 to 51 with a step of 5, all at the fastest preset. For JPEG a range of quality from 10 to 100 with a step of 10. For Theora a quality range from 4 to 10 with a step of 3, quality 1 was omitted because when decoded a different number of frames is obtained, instead of the 300. For JPEG2000 a compression ratio between 5 and 100 with a multiplication step of 2.

Figure 3.16 presents the results for the bits per pixel/PSNR relation for the Alboi moving image dataset (Appendix B.1). For smaller PSNR values H.265 dominates by a wide margin. For higher PSNR values both BPG and JPEG2000 provide the best results.

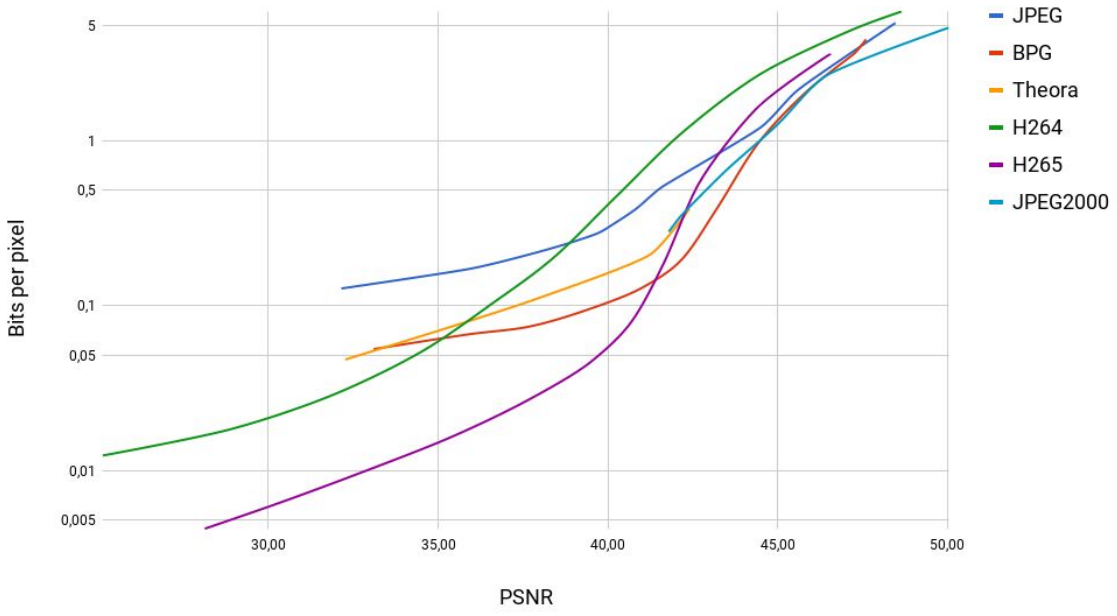


Figure 3.16: Bits per pixel/PSNR relation in the Alboi moving image dataset (B.1). The X axis represents the PSNR and the Y axis represents the bits per pixel.

Figure 3.17 contains the bits per pixel/PSNR for the Alboi mixed image dataset (Appendix B.2). As this dataset contains a sequence of frames with little motion, it provides better conditions for the video compression standards. As such both H.264 and H.265 provide much better PSNR at lower bits per pixel. Moreover, Theora also significantly improves, beating BPG at lower PSNR values. As such, the lower PSNR values are still dominated by H.265. For higher PSNR value both BPG and JPEG2000 continue to provide the best results.

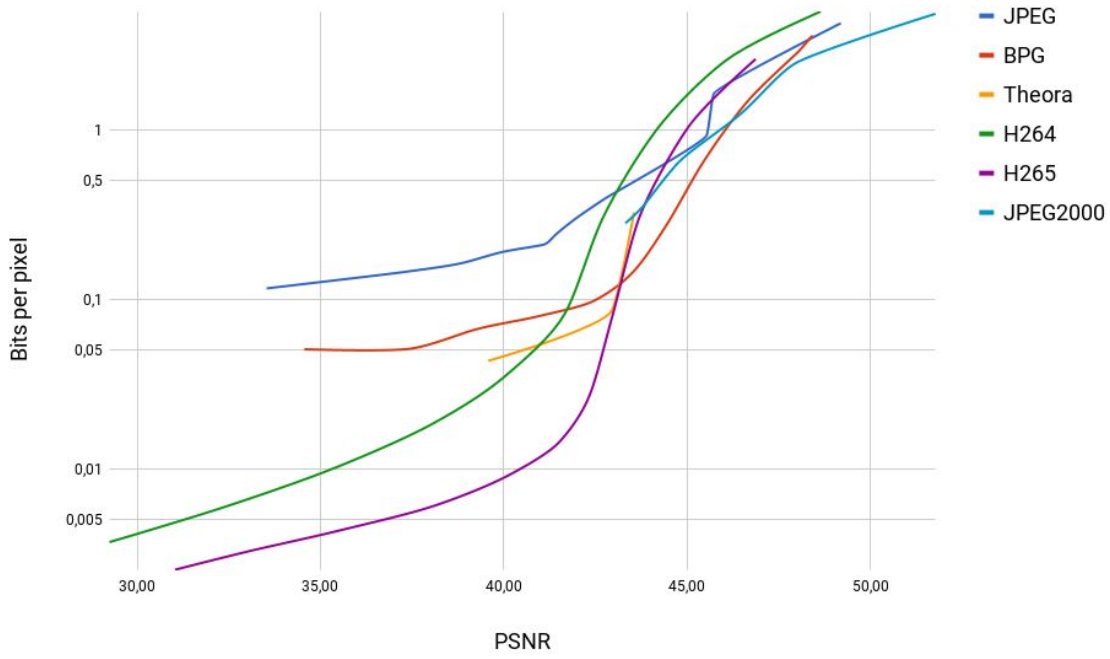


Figure 3.17: Bits per pixel/PSNR relation in the Alboi mixed image dataset (B.2). The X axis represents the PSNR and the Y axis represents the bits per pixel.

Figure 3.18 contains the bits per pixel/PSNR for the P0 Large image dataset (Appendix B.4). As this dataset contains minimal motion all the video compression standards show significant improvement compared to the other datasets. For low PSNR values H.264 beats H.265 by a small margin and provides the best results. For higher PSNR values H.265 provides the best compression. For the highest PSNR values, that H.265 doesn't reach, JPEG and JPEG2000 have similar compression efficiency.

In summary, H.265, BPG and JPEG2000, for the most part, provide the best compression efficiency for the entire PSNR spectrum. Considering that BPG is a subset of H.265 and BPG and JPEG2000 obtain similar results we discard JPEG2000 from further tests.. With this in mind we conclude that ROS could benefit with the implementation of a H.265 codec. Moreover, since H.265 provides lossless and lossy video compression, with intra-picture coding only if necessary, ROS would also benefit from only having to deal with one code base instead of a separate code base for each standard that is already implemented.

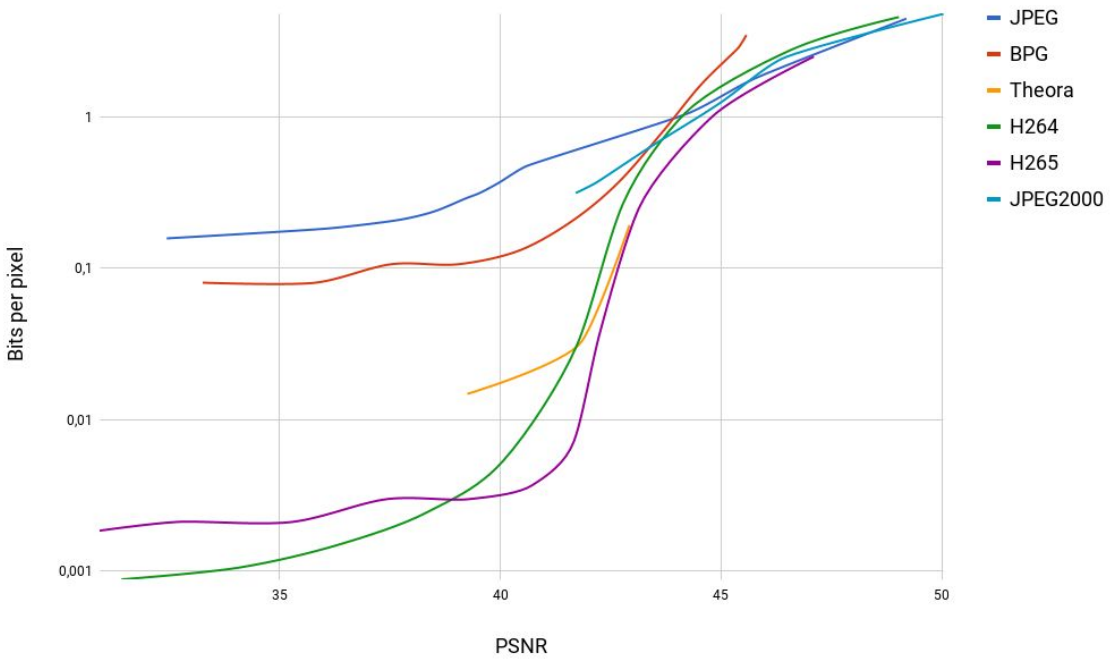


Figure 3.18: Bits per pixel/PSNR relation in the P0 Large image dataset (B.4). The X axis represents the PSNR and the Y axis represents the bits per pixel.

CHAPTER 4

Effects of lossy compression

In the previous chapter we concluded that lossy compression standards provide optimal results and that ROS would significantly benefit from the implementation of a H.265 codec. However, it is known that lossy compression works by discarding information. Typically lossy compression standards are fine tuned for discarding information based on human perception. In this chapter we conduct a detailed study on the effects of lossy compression in the performance of several image analysis algorithms when using H.265 or the native compression standards, JPEG and Theora.

It is known that over compressing JPEG images may take a toll on some image analysis algorithms [48]–[50]. In this chapter, we provide a detailed study on the effects that excessive compression with H.265, JPEG and Theora has on several popular image analysis algorithms, ranging from high level operations, such as facial recognition, object detection (face and body) and color segmentation, to low level operations, such as finding contours and feature extraction. We chose those algorithms with the intent of covering a wide range of image analysis operations used in robotics.

Each chosen algorithm is executed as a ROS node, which allows us to take advantage of ROS parameter passing. Each experiment is started individually with the proper parameters set and the help of a script (Appendix C.1). In order to carry on with the experiments, we select a range of compression levels and qualities for each standard. For the facial recognition experiment we use H.265 with a range of CRF from 1 to 51 with a step of 5. JPEG with range of quality from 10 to 100 with a step of 10. Theora with range of quality from 1 to 10 with a step of 3. For the rest of the experiments we divide each compression standard into three tiers.

- **High compression** (low quality). For this tier we select H.265 with a CRF of 51, JPEG with quality 10 and Theora with quality 1.
- **Mid compression** (mid quality). For this tier we select H.265 with a CRF of 26, JPEG with quality 50 and Theora with quality 7.

- **Low compression** (high quality). For this tier we select H.265 with a CRF of 1, JPEG with quality 100 and Theora with quality 10.

4.1 Face recognition

For facial recognition we use the popular Local Binary Patterns Histograms (LBPH) [51] algorithm. This method makes use of the Local Binary Patterns (LBP) [52]–[54] algorithm which is a texture operator that labels the pixels of an image by thresholding the neighborhood of each pixel. LBPH divides the results of the LBP into a grid and extracts the histogram for each region of the grid. Those histograms will then be concatenated to form a new and bigger histogram that will represent the characteristics of the face [55]. Figure 4.1 contains an example of the procedure for one face using the LBPH algorithm.

A complete implementation is available in the OpenCV library. However, there are also low-level hardware implementations [56], making this a very suitable method for facial recognition in real time systems.

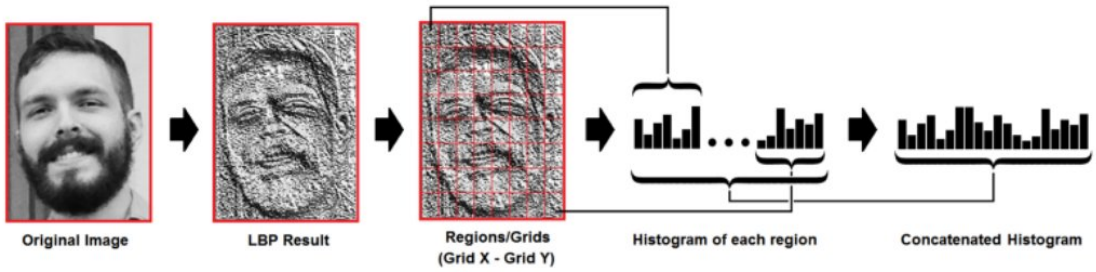


Figure 4.1: Example procedure for one face using the LBPH algorithm [55].

4.1.1 Procedure

This procedure is an adapted version of the older OpenCV facial recognition tutorial that uses LBPH [57]. To identify a subject and which photos belong to the subject, each compression configuration uses a simple Comma-separated values (CSV) file.

For this experiment we use the Database of Faces (Appendix B.7). Before creating the face recognition models, two testing datasets are created, the compressed dataset (Ct) and the raw dataset (Rt). Each dataset contains an image from each subject, extracted from the orl faces dataset. Two separate facial recognition models are created. One trained with the compressed images (Cm), and the other with the raw images (Rm). The Ct dataset is tested with both models, the Rt dataset is tested with the compressed model.

In this experiment we record the number of correctly recognized and the number of wrongly recognized faces for each compression level of each standard.

4.1.2 Experimental results

Table 4.1, 4.2 and 4.3 contain the results for each compression standard. "CmCt" represents the results for the model trained and tested with compressed data. "RmCt" represents the results for the model trained with raw data and tested with compressed data. "CmRt" represents the results for the model trained with compressed data and tested with raw data.

On Table 4.1 it is possible to see that if only compressed H.265 data is used the performance is the same as when using raw data, independently of the level of compression. However, when mixing raw and compressed data, after a certain threshold, the performance of the model decreases while the compression increases. Moreover, after CRF 36 the models performance plunges. We conclude that, if mixing compressed and raw data, the best CRF is 26, as it provides the best compression/accuracy ratio.

H.265	CmCt		RmCt		CmRt	
	Rights	Wrongs	Rights	Wrongs	Rights	Wrongs
1	38	2	38	2	38	2
6	38	2	38	2	38	2
11	38	2	38	2	38	2
16	38	2	38	2	38	2
21	38	2	38	2	37	3
26	38	2	38	2	37	3
31	38	2	36	4	37	3
36	39	1	35	5	33	7
41	39	1	29	11	28	12
46	38	2	14	26	20	20
51	38	2	11	29	15	25

Table 4.1: Results on the effects of lossy H.265 compression on facial recognition. "CmCt" represents the results for the model trained and tested with compressed data. "RmCt" represents the results for the model trained with raw data and tested with compressed data. "CmRt" represents the results for the model trained with compressed data and tested with raw data.

Looking at Table 4.2 we can observe that with JPEG the performance of the model is very consistent, even at higher compression. When mixing raw and compressed data the models performance slightly decreases after quality level 60. It is only after quality level 30 that the performance decreases significantly. When using JPEG data for training and testing the models performance matches that of using only raw data.

Table 4.3 shows that, when mixing raw data and Theora, after quality level 7 the models performance quickly goes down. When using quality level of 7 we found no noticeable decreases in performance. As such we conclude that when mixing data the best quality level for Theora is 7. When using only Theora compressed data the models offer similar performance compared to only using raw data, no matter the compression level.

In summary, when mixing compressed and raw data for facial recognition, high compression should be avoided. It is possible, however, to obtain good performance while mixing compressed and raw data, using low compression. If compressed data is used both in training and in

JPEG	CmCt		RmCt		CmRt	
	Rights	Wrongs	Rights	Wrongs	Rights	Wrongs
10	38	2	35	5	34	6
20	38	2	36	4	37	3
30	38	2	36	4	37	3
40	38	2	37	3	37	3
50	38	2	37	3	37	3
60	38	2	37	3	37	3
70	38	2	38	2	37	3
80	39	1	38	2	38	2
90	38	2	38	2	38	2
100	38	2	38	2	38	2

Table 4.2: Results on the effects of JPEG compression on facial recognition. "CmCt" represents the results for the model trained and tested with compressed data. "RmCt" represents the results for the model trained with raw data and tested with compressed data. "CmRt" represents the results for the model trained with compressed data and tested with raw data.

	CmCt		RmCt		CmRt	
Theora	Rights	Wrongs	Rights	Wrongs	Rights	Wrongs
1	39	1	31	9	34	6
4	38	2	35	5	33	7
7	39	1	38	2	37	3
10	38	2	38	2	38	2

Table 4.3: Results on the effects of Theora compression on facial recognition. "CmCt" represents the results for the model trained and tested with compressed data. "RmCt" represents the results for the model trained with raw data and tested with compressed data. "CmRt" represents the results for the model trained with compressed data and tested with raw data.

testing, high compression can be used without toll on the accuracy of the facial recognition models.

4.2 Face and body detection

For the detection of faces and bodies we use the popular Viola-Jones object detection method [58], [59]. It uses a machine learning approach to object detection. It is composed by four main stages. First it computes the integral image and then it extracts haar-like features (image features). These features are then selected with an adaboost algorithm. Lastly the selected features are used to train the cascading classifiers to detect the object [60].

This method provides reliable and fast object detection. It is capable of being used in real time systems, such as robots. A complete implementation is available in the OpenCV library. There are also pre-trained models (haar cascades) available.

There is still a lot of research going on concerning object detection, mostly through deep

learning, such as the Faster R-CNN [61] algorithm or the You Only Look Once (YOLO) framework [62], [63]. However, these algorithms are very computationally demanding and thus require better hardware in order to operate smoothly.

4.2.1 Procedure

For this experiment we adapted a version from the OpenCV face detection tutorial which can be found here [64]. This procedure makes use of the pre trained haar cascades that define a frontal face (haar cascade frontalface alt) and a full body (haar cascade fullbody). Both haar cascades were trained with images that come from a different sensor and use different compression than the images contained in the dataset we tested. They can be found here [65]. This procedure starts by converting the image into grayscale and equalizing the image histogram. Only then will the objects be detected. The image dataset the we use in this experiment is the People image dataset (Appendix B.5, Figure 4.2).

Figure 4.2: Images in the People image dataset.

In this experiment we record the number of detected objects in each tier of each standard

and compare them with the raw values. We also verify if the detected objects overlap with the raw values. We consider that the detected objects overlap if the area of the intersection between the detected objects is larger than one fourth of the area of the detected object in the raw image.

4.2.2 Experimental results

Tables 4.4 and 4.5 contain the results for face detection. Tables 4.6 and 4.7 contain the results for people detection.

Looking at Table 4.4 it is possible to see that no standard provides 100% matching results to when using raw data. Considering the lowest compression (highest quality) the closest to raw is JPEG. Both Theora and H.265 deviate from the raw results. At mid compression (mid quality) H.265 provides similar results to raw however at image #4 it detects less faces. Both JPEG and Theora deviate from raw in two images. At high compression (low quality) H.265 has trouble detecting faces due to the loss of details. JPEG detects too many faces due to the amount of noise introduced by the transform. Theora, unexpectedly, shows similar levels of performance when compared to mid compression levels.

Detections	#1	#2	#3	#4	#5	#6
raw	1	1	1	3	2	2
H 1	1	2	1	2	3	4
J 100	1	1	1	2	2	3
T 10	1	2	2	3	2	4
H 26	1	1	1	2	2	2
J 50	2	1	1	2	2	3
T 7	1	1	1	2	3	3
H 51	1	1	0	1	1	0
J 10	1	1	2	4	4	3
T 1	1	1	2	2	2	2

Table 4.4: Number of detected faces for each tier of each compression standard. The number of detected faces when using raw images is also presented as a reference.

Table 4.5 shows how many of those detected faces actually match the faces detected using raw data. Considering the deviation from the number of faces detected using lossy compression and raw data, most of the detected faces with compressed data are in line with the detected faces using raw data. With the exception of high compressed (low quality) H.265 on image #2. The excess of detected faces that do not overlap with the raw detected faces are mostly due to noise introduced with compression. Figure 4.3 shows one of this cases.

Overlap	#1	#2	#3	#4	#5	#6
H 1	1/1	1/2	1/1	2/2	2/3	2/4
J 100	1/1	1/1	1/1	2/2	2/2	2/3
T 10	1/1	1/2	1/2	3/3	2/2	2/4
H 26	1/1	1/1	1/1	2/2	2/2	2/2
J 50	1/2	1/1	1/1	2/2	2/2	2/3
T 7	1/1	1/1	1/1	2/2	2/3	2/3
H 51	1/1	0/1	0/0	1/1	1/1	0/0
J 10	1/1	1/1	1/2	3/4	2/4	2/3
T 1	1/1	1/1	1/2	2/2	2/2	2/2

Table 4.5: Number of overlaps between detected faces using raw images and compressed images for each tier of each compression standard.

(a) Image compressed with H.265 and CRF 51.

(b) Image compressed with Theora quality level 10.

Figure 4.3: Wrongly detected faces because of the noise added by the compression algorithm. Each rectangle represents a detected face. Only two detected faces are actually correct, on image (b).

Despite some wrongly detected faces due to compression induced noise, there are some cases where using compression actually improved the detection accuracy. For example, on Figure 4.4, using low compression (high quality) H.265 prevented a false positive, but using the raw image a false positive was detected.

(a) H265 compressed image with best quality. **(b)** Raw image.

Figure 4.4: Wrongly detected faces on raw image. Each rectangle represents a detected face. The raw image (b) contains a wrongly detected face while the compressed image (a) does not.

Considering body detection, Table 4.6 shows that on low compression (high quality) JPEG and Theora provide the closest results to raw while H.265 deviates on image #4, detecting one less body. At mid compression (mid quality) both JPEG and Theora deviate on one image, detecting one more body each. H.265 provides the same results as raw.

People	#1	#2	#3	#4	#5	#6
raw	1	2	1	2	2	2
H 1	1	2	1	1	2	2
J 100	1	2	1	2	2	2
T 10	1	2	1	2	2	2
H 26	1	2	1	2	2	2
J 50	1	2	1	3	2	2
T 7	1	2	1	2	2	3
H 51	1	1	1	2	2	1
J 10	1	1	1	3	2	1
T 1	2	1	1	2	2	1

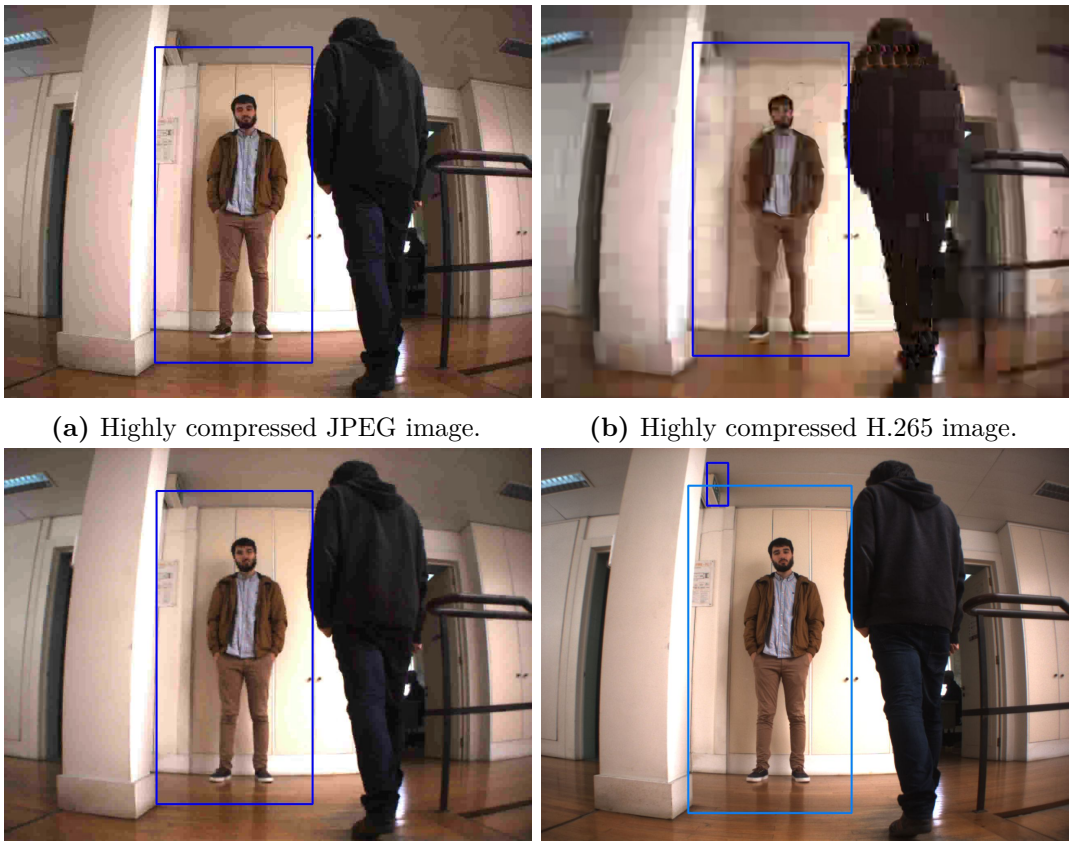
Table 4.6: Number of detected bodies for each tier of each compression standard. The number of detected faces when using raw images is also presented as a reference.

Table 4.7 shows the overlap between the detected bodies using compressed data and the detected bodies using raw data. Considering the number of bodies detected with raw data it is possible to see that, for the most part, low to mid compression (high to mid quality) provides very consistent body detection. On high compression (low quality), the number of detected bodies goes down. However, most of the detected bodies overlap with the bodies detected using raw data.

Looking at Figure 4.5, high compressed data can provide better results than using raw data. By using compressed data, in this example, we avoid a false positive. However, this is not always the case. Figure 4.6 and 4.7 contain two examples where using compressed data results in more false positives than when using raw data.

People	#1	#2	#3	#4	#5	#6
H 1	1/1	2/2	1/1	1/1	2/2	2/2
J 100	1/1	2/2	1/1	2/2	2/2	2/2
T 10	1/1	2/2	1/1	2/2	2/2	2/2
H 26	1/1	2/2	1/1	2/2	2/2	2/2
J 50	1/1	2/2	1/1	3/3	2/2	2/2
T 7	1/1	2/2	1/1	2/2	2/2	3/3
H 51	1/1	1/1	1/1	2/2	2/2	1/1
J 10	1/1	1/1	1/1	2/3	2/2	1/1
T 1	1/2	1/1	1/1	2/2	2/2	1/1

Table 4.7: Number of overlaps between detected bodies using raw images and compressed images for each tier of each compression standard.



(a) Highly compressed JPEG image.

(b) Highly compressed H.265 image.

(c) Highly compressed Theora image.

(d) Raw image.

Figure 4.5: Example where highly compressed images provide better accuracy than the raw image on frontal body detection. The raw image contains a wrongly detected body.

(a) Raw image. **(b)** Highly compressed JPEG image.

Figure 4.6: Example where the raw image provides better accuracy than the highly compressed JPEG image on frontal body detection. Despite the raw image also having a wrongly detected body, the JPEG image has two wrongly detected bodies.

(a) Raw image. **(b)** Mid quality Theora compressed image.

Figure 4.7: Example where raw image provides better accuracy than compressed (mid quality) Theora image on frontal body detection. Despite the raw image also having a wrongly detected body, the Theora image has two overlapped wrongly detected bodies.

4.3 Find contours

On robotic systems, detect contours on images have many use cases as a pre-processing stage, namely on high level segmentation, object detection, among others. In this section, the effects of lossy compression are tested on the outcome of contour calculation in images. The image dataset that we use for this experiment is the People Dataset (Appendix B.5).

For contour calculation we use the algorithm proposed by Suzuki and Be in [66]. This algorithm requires an input binary image. For this, we use a binary image that contains edges found by a canny edge detector [67], a multi-stage algorithm that detects edges. Both algorithm implementations can be found at the OpenCV library. There are two thresholds required for the canny edge detector. The first and second thresholds for the hysteresis procedure. These values can either be static or dynamic. However, both possibilities were tested.

The static values for these thresholds were 100 and 200 respectively. There are several ways to use dynamic threshold values for canny edge detection [50], [68], [69], the one we use in this experiment is an adaptation of the version provided by A. Rosebrock [69]. This version uses the median of the grayscale version of the image (M), the first value (F) is obtained using $F = max(0.0, (1-0.33)*M)$ and the second (S) is obtained using $S = min(255, (1+0.33)*M)$.

4.3.1 Procedure

This procedure is an adapted version of the OpenCV contour tutorial [70]. Some preprocessing is applied before finding the contours. The image is first converted into grayscale and then blurred. This will remove excess noise from the image. After we detect the edges using the Canny edge detector. With the binary image that contains the edges we proceed to find and draw the contours using the algorithm mentioned above.

In this experiment we record the number of contours found. Moreover, we also draw the contours on a binary image and use that image to calculate two metrics. The first and simpler metric is to calculate the percentage of different pixels between the binary images. The second metric is the Baddeley error metric [71]. This metric is defined as the p-th order mean difference between thresholded distance transforms of the two images.

4.3.2 Experimental results

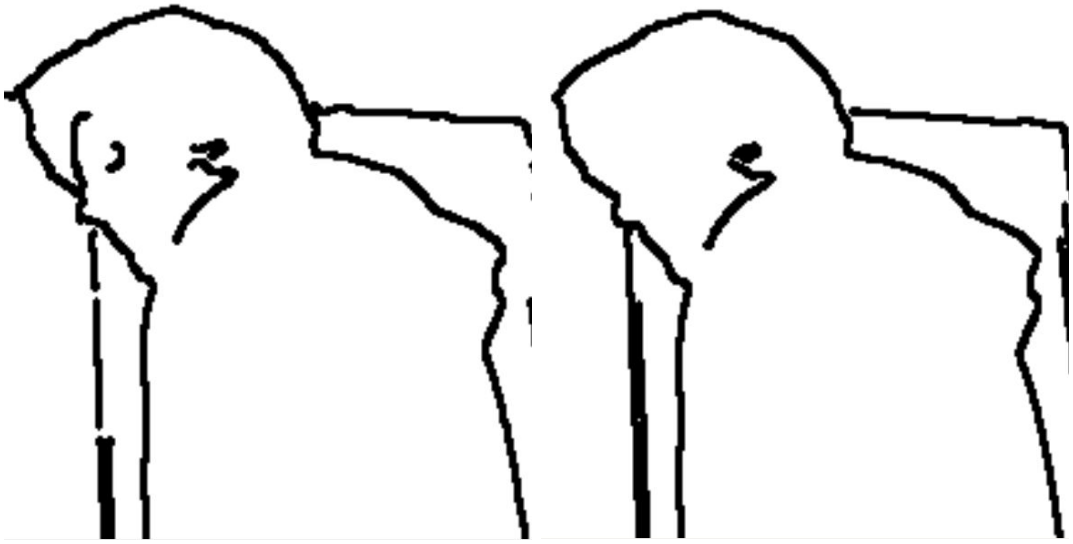
Tables 4.8 and 4.9 show that, no matter the threshold, the higher the compression (lower quality) on H.265, the less contours are found. The higher the compression (lower quality) on JPEG, the more contours are found. Theora shows no pattern on the number of contours found. This happens in H.265 because of the loss of details, and the bigger block size, which may remove certain contours or merge smaller contours into a bigger contour. In JPEG, this happens because of the smaller block sizes and the introduction of noise, which makes artifact contours appear or breaks large contours into smaller ones. Figure 4.8 shows an example, on the JPEG image a line is broken into separate contours whereas in the raw image this does not happen.

Dynamic threshold	#1	#2	#3	#4	#5	#6
Raw	114	143	127	147	157	200
H 1	115	149	134	148	156	191
J 100	115	144	122	141	158	197
T 10	108	147	129	136	159	193
H 26	103	129	114	116	138	164
J 50	115	147	123	135	161	205
T 7	113	131	109	127	139	189
H 51	60	113	115	115	113	165
J 10	162	167	168	172	193	240
T 1	102	151	126	122	161	213

Table 4.8: Number of contours found for each tier of each standard using dynamic thresholds. The number of contours found when using raw images is also presented as a reference.

Static Threshold	#1	#2	#3	#4	#5	#6
Raw	72	90	87	84	100	118
H 1	75	94	93	82	101	114
J 100	75	93	87	83	103	119
T 10	75	85	90	76	97	116
H 26	72	79	81	80	93	110
J 50	80	92	84	82	106	124
T 7	74	81	75	81	93	119
H 51	36	60	72	76	65	87
J 10	84	105	96	98	114	151
T 1	58	86	81	78	108	138

Table 4.9: Number of contours found for each tier of each standard using static thresholds. The number of contours found when using raw images is also presented as a reference.



(a) Dynamic threshold jpeg quality 10 image. **(b)** Dynamic threshold raw image.

Figure 4.8: Differences in contours with a JPEG and raw images. Image (a) contains the contours found with in the JPEG with a quality level of 10. Image (b) contains the contours found in the raw image. Both contours were found with a dynamic threshold.

Table 4.10 shows the Baddeley errors when using dynamic thresholds. Table 4.11 shows the Baddeley errors when using static thresholds. Table 4.10 shows that JPEG provides the smallest Baddeley errors. H.265 comes in second and Theora provides the biggest Baddeley errors. Naturally, the higher the compression (lower quality) the higher the errors.

Dynamic Threshold	#1	#2	#3	#4	#5	#6
H 1	0,76	0,97	0,87	1,02	0,94	1,01
J 100	0,72	0,60	0,58	0,57	0,62	0,72
T 10	1,14	1,48	1,48	1,46	1,44	1,48
H 26	1,61	2,03	1,96	2,05	2,02	2,05
J 50	1,50	1,76	1,61	1,64	1,58	1,79
T 7	1,68	1,93	1,95	1,91	1,89	1,91
H 51	3,11	3,39	3,69	3,66	3,45	3,66
J 10	2,45	2,52	2,57	2,55	2,59	2,77
T 1	4,12	4,37	4,47	4,58	4,27	4,59

Table 4.10: Baddeley errors between the results of using the compressed image and the raw image for each tier of each compression standard using dynamic thresholds.

Observing Table 4.11, for the most part, by using static thresholds we obtain slightly smaller Baddeley errors when compared to using dynamic thresholds. This has something to do with the fact that, when using static threshold the number of contours will be much smaller, as seen in Table 4.9. Logically, with smaller margin for error we obtain smaller errors. JPEG continues to provide the smallest Baddeley errors of the three tested standards. H.265 is still in between and Theora provides the biggest Baddeley errors.

Static Threshold	#1	#2	#3	#4	#5	#6
H 1	0,70	0,77	0,68	0,75	0,77	0,96
J 100	0,42	0,59	0,49	0,49	0,42	0,58
T 10	0,83	1,19	1,40	1,23	1,22	1,18
H 26	1,17	2,65	1,58	1,55	1,59	1,58
J 50	1,13	1,21	1,36	1,24	1,19	1,47
T 7	2,44	1,46	1,69	1,53	1,56	1,45
H 51	2,66	2,93	3,17	3,06	2,88	3,23
J 10	2,59	1,94	2,04	2,05	2,00	2,15
T 1	3,18	3,73	3,89	3,72	3,72	3,74

Table 4.11: Baddeley errors between the results of using the compressed image and the raw image for each tier of each compression standard using static thresholds.

Table 4.12 shows the percentage of different pixels between raw and compressed found contours when using dynamic thresholds. Table 4.13 shows the same type of errors but when using static thresholds instead.

At low compression (high quality) the pattern remains the same as above. JPEG provides the smallest errors with H.265 in the middle and Theora with the highest errors. However, at mid compression this pattern begins to change. Both Theora and H.265 provide similar results

at mid compression while JPEG still provides the smallest errors. At high compression (low quality) H.265 overtakes Theora as the standard with the highest error values. Theora comes in second while JPEG still provides the smallest errors. Table 4.11 shows that the static threshold Baddeley errors are only slightly smaller when compared to the dynamic threshold. However, looking at Table 4.13 the percentage of different pixels with static thresholds is significantly smaller than with dynamic thresholds, especially at high compression (low quality) levels. Which suggests that despite having less contours and a smaller margin for the appearance of errors, the errors that do appear are of higher significance when compared to the errors that appear with dynamic thresholds.

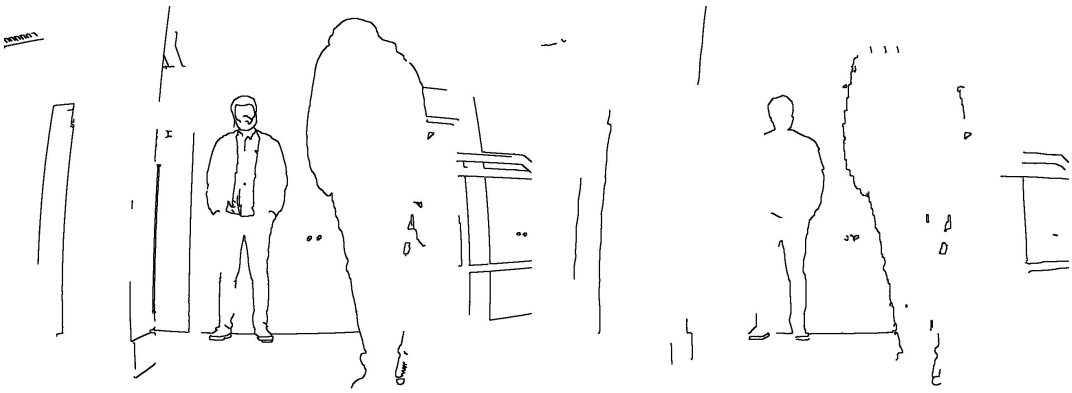
Dynamic Threshold	#1	#2	#3	#4	#5	#6
H 1	0,12	0,22	0,18	0,26	0,20	0,24
J 100	0,15	0,08	0,08	0,06	0,09	0,13
T 10	0,31	0,58	0,58	0,54	0,54	0,53
H 26	0,69	1,16	1,09	1,17	1,15	1,09
J 50	0,56	0,85	0,67	0,69	0,61	0,77
T 7	0,78	1,00	1,05	0,97	0,95	0,87
H 51	3,18	3,78	3,85	3,97	3,75	4,23
J 10	1,62	1,64	1,68	1,58	1,73	1,92
T 1	2,11	2,34	2,45	2,53	2,22	2,68

Table 4.12: Percentage of different pixels between the results of using the compressed image and the raw image for each tier of each compression standard using dynamic thresholds.

Static Threshold	#1	#2	#3	#4	#5	#6
H 1	0,14	0,15	0,11	0,14	0,16	0,28
J 100	0,04	0,10	0,06	0,06	0,03	0,10
T 10	0,17	0,41	0,61	0,44	0,42	0,37
H 26	0,37	0,64	0,69	0,64	0,70	0,64
J 50	0,33	0,36	0,52	0,40	0,32	0,58
T 7	0,40	0,58	0,85	0,65	0,68	0,54
H 51	1,91	2,50	2,69	2,63	2,50	2,71
J 10	0,94	0,97	1,06	1,11	0,97	1,16
T 1	1,11	1,59	1,73	1,59	1,56	1,62

Table 4.13: Percentage of different pixels between the results of using the compressed image and the raw image for each tier of each compression standard using static thresholds.

However these number are only useful for comparing performance between standards, as they do not give any feedback about real world usage. For that, we provide Figures 4.9, 4.10, 4.11 and 4.12. Figure 4.9 contains the worst and best case for high compression H.265. Looking at the worst case, we can see that objects start to loose their shape and lose too much detail. This will reduce the accuracy of the detected contours and may cripple the system. Looking at the best case we can see that, for the most part, the detected objects keep their shape, however, they continue to lose too much detail.

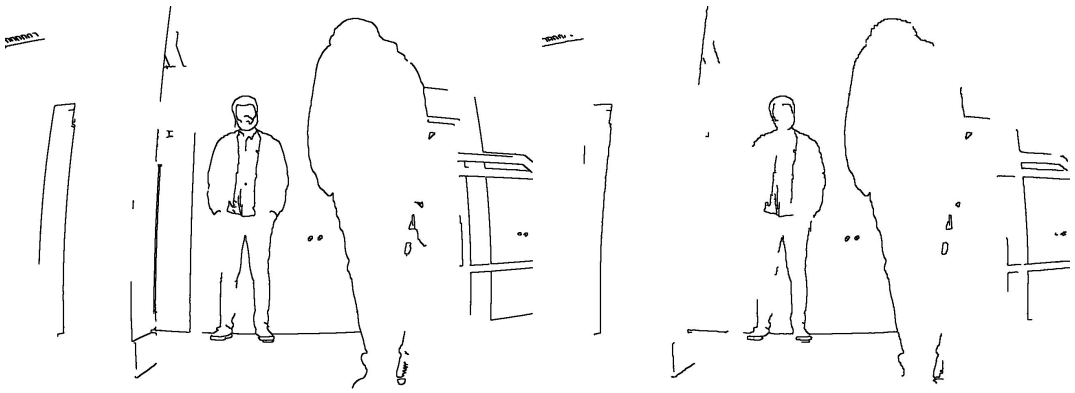


(a) Raw image. **(b)** High compression H.265.

(c) Raw image. **(d)** High compression H.265.

Figure 4.9: Best and worst case for high compression H.265.

Figure 4.10 contains the worst and best case for high compression Theora. Looking at the worst case, the detected contours lose a little detail. At the best case, the detected contours also lose a little detail, especially on very noisy areas. Both worst and best case provide good results.



(a) Raw image. **(b)** High compression Theora.

(c) Raw image. **(d)** High compression Theora.

Figure 4.10: Best and worst case for high compression Theora.

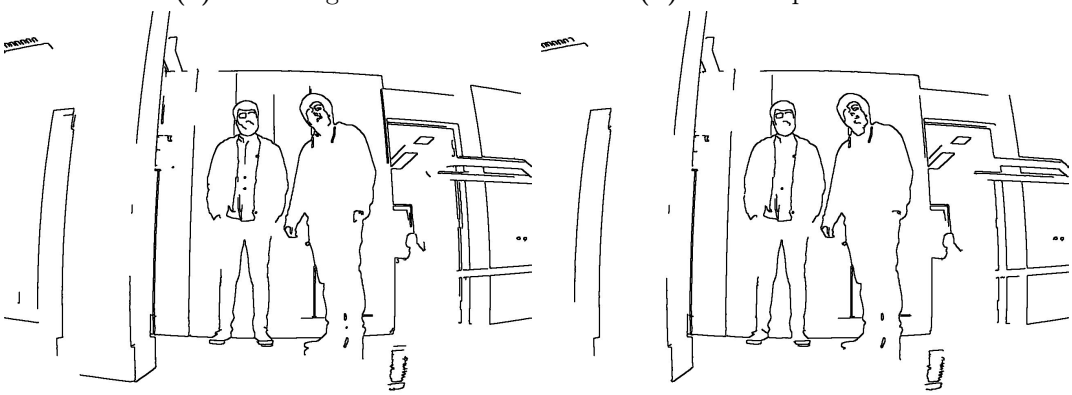
Figure 4.11 contains the worst case for high compression JPEG. Looking at the worst case, high compressed JPEG provides very good results, with only minor differences.

(a) Raw image. **(b)** High compression JPEG.

Figure 4.11: Worst case for highly compressed JPEG images.

Figure 4.12 contains the worst cases for mid compression H.265 and Theora. Looking at the worst case for mid compressed H.265, the only noticeable difference is at the left side of the pillar. Other than this H.265 provided very good results. Looking at the worst case for Theora, other than some small details, it achieved very good results too.

(a) Raw image. **(b)** Mid compression H.265.



(c) Raw image. **(d)** Mid compression Theora.

Figure 4.12: Worst case for high compression H.265 and Theora.

Looking at the presented results, we can conclude that, high compression H.265 is not suitable for finding contours, as it loses too much details and objects start to lose shape. At mid compression H.265 provides good results. Theora provides good results at high compression and even better at mid compression. JPEG provides excellent results even at high compression.

4.4 Color Segmentation

Image segmentation in robotic vision is very important. It is the basis for many other algorithms. Segmentation can be used for object detection, object recognition, object tracking, and many other applications. In this section, we will study the effects of lossy compression on high level color image segmentation. For this we set up the task of segmenting a person out of an image. The segmentation algorithm we use is Grabcut [72]. This algorithm, for the most part, requires user interaction. We chose this algorithm specifically because it allowed us to manually mark what was background and what we wanted to segment. By manually inserting the markers we reduce the error introduced pre-segmentation, allowing for more accurate results. The image dataset we use in this experiment is the People Dataset (Appendix B.5). Same as in Section 4.2 and 4.3.

4.4.1 Procedure

This procedure is an adapted version of the OpenCV python grabcut tutorial [73]. Before the experiment started we needed to manually mark what we wanted to be segmented from the background. Figure 4.13 contains the manual markers from one of the images.

(a) Original image. **(b)** Image with markers.

Figure 4.13: Example of manually inserted segmentation markers. Sure foreground is marked in red. Sure background is marked in blue.

After obtaining the markers this experiment starts with using the same algorithm tested in section 4.2 to detect a body. This body will only be detected in the raw image to avoid errors unrelated to the referred algorithm. With the detected body we get a region of interest. Everything outside this region of interest will be considered as background. We also create a mask with the manually marked image. With the region of interest and the mask we then apply the grabcut algorithm and obtain the segmented image.

In this experiment we record the same two metrics used in the finding contours experiment. The Baddeley error metric [71] and the percentage of different pixels metric.

4.4.2 Experimental results

Table 4.14 shows the Baddeley errors and Table 4.15 shows the percentage of different pixels, when comparing raw with compressed images. By looking at table 4.14, for low compression (high quality) JPEG provides, for the most part, the smallest errors. H.265 stands in between JPEG and Theora, Theora providing the highest errors. For mid compression (mid quality), Theora provides the smallest errors, H.265 continues to be in between and JPEG provides the highest errors. For high compression (low quality), Theora continues to provide the smallest errors, with JPEG taking the middle position and H.265 providing the highest errors. Table 4.15 shows similar results.

Baddeley error	#1	#2	#3	#4	#5	#6
H 1	1,00	0,94	0,73	1,13	1,88	1,17
J 100	0,59	0,90	0,91	1,10	1,01	0,90
T 10	1,34	1,37	1,07	1,47	1,48	2,01
H 26	1,58	1,71	1,39	1,84	2,44	2,19
J 50	1,99	1,95	1,32	1,91	2,48	2,26
T 7	1,63	1,63	1,22	1,63	2,22	1,96
H 51	2,38	2,23	1,79	2,93	3,08	2,69
J 10	2,26	2,06	1,62	2,11	2,62	2,61
T 1	2,31	2,19	1,61	2,19	2,39	2,46

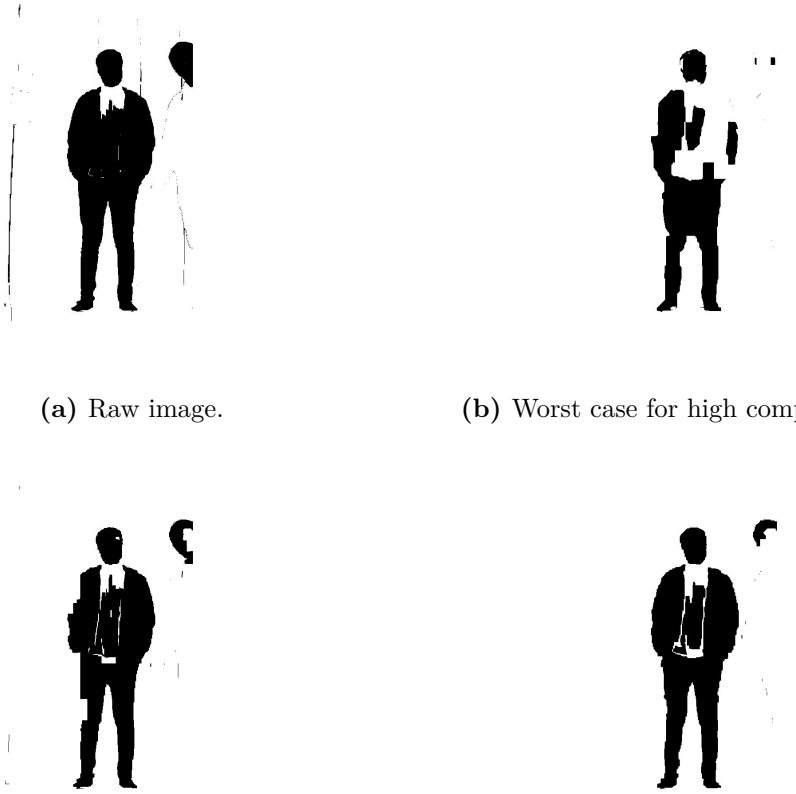
Table 4.14: Baddeley errors between the results of using the compressed image and the raw image for each tier of each compression standard.

percentage error	#1	#2	#3	#4	#5	#6
H 1	0,06	0,07	0,04	0,15	0,97	0,11
J 100	0,02	0,07	0,04	0,15	0,11	0,06
T 10	0,12	0,14	0,08	0,32	0,26	0,77
H 26	0,18	0,21	0,19	0,36	1,20	0,46
J 50	0,27	0,26	0,11	0,37	1,23	0,57
T 7	0,29	0,18	0,10	0,29	1,08	0,32
H 51	0,74	0,48	0,42	2,08	2,06	1,01
J 10	0,65	0,34	0,29	0,59	1,31	0,98
T 1	0,65	0,48	0,23	0,66	0,79	0,67

Table 4.15: Percentage of different pixels between the results of using the compressed image and the raw image for each tier of each compression standard.

As mentioned in Section 4.3, both the Baddeley error and the percentage of different pixels do not provide feedback about real world usage. They are only useful for comparing performance between standards. Figure 4.14, 4.15, 4.16 and 4.17 provide examples of the performance of the standards at segmenting the person.

Figure 4.14 contains the worst cases for each standard at high compression (low quality). Looking at the worst case for JPEG and Theora, we can see that, they deviate from the segmentation using the raw image. Using morphological operations it could be possible to fix this deviation. However, this requires pos-processing and would decrease system performance. Looking at the worst case for H.265, it significantly deviates from the segmentation using the raw image. At that stage fixing would be too costly.



(a) Raw image.

(b) Worst case for high compression H.265.

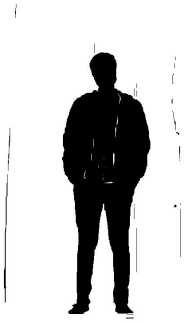
(c) Worst case for high compression JPEG.

(d) Worst case for high compression Theora.

Figure 4.14: Worst cases for the three standards using high compression and the reference case using raw image.

Figure 4.15 contains the worst cases for each standard at mid compression (mid quality). Looking at the worst case for JPEG and Theora, they slightly deviate from the segmentation using the raw image. Like mentioned above, using morphological operations it could be possible to fix this deviation, however, this requires pos-processing and would decrease system performance. Looking at the worst case for H.265, it manages to completely segment the person, without flaws in the body. Furthermore, it does not segment the detached head, contained in the other results. Although it achieves very good results, it deviates from the results obtained using the raw image.

(a) Raw image.



(b) Worst case for mid compression H.265.

(c) Worst case for mid compression JPEG.

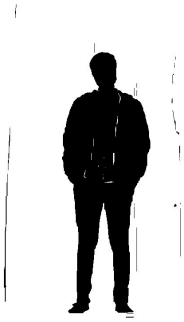
(d) Worst case for mid compression Theora.

Figure 4.15: Worst cases for the three standards using mid compression and the reference case using raw image.

Figure 4.16 contains the best case for H.265 high and mid compression (low and mid quality) and the best case for highly compressed JPEG images. Looking at this figure we can see that, H.265 was able to segment the person at both high and mid compression (low and mid quality). JPEG was also able to segment the person, however it contains a deviation at the person shoulder. Moreover, it also removed the noise lines found at the segmentation using the raw image.

(a) Raw image.

(b) Best case using high compression H.265.

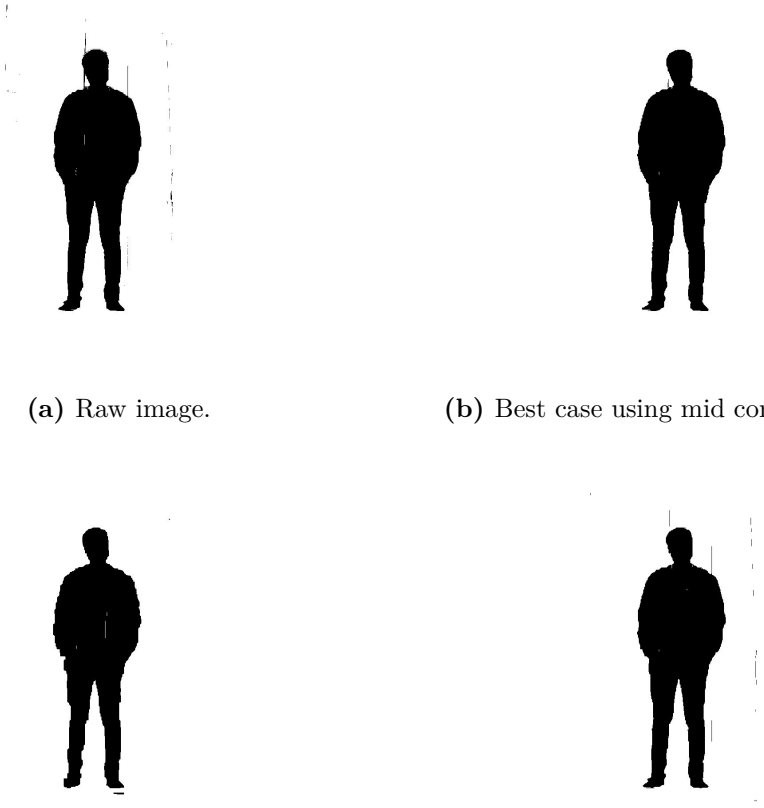


(c) Best case using mid compression H.265.

(d) Best case using high compression JPEG.

Figure 4.16: Best cases for high and mid compression H.265 and high compression JPEG and the reference case using raw image.

Figure 4.17 contains the best case for Theora high and mid compression (low and mid quality) and the best case for mid compression (mid quality) JPEG. Looking at this figure, both mid compression JPEG and high to mid compression Theora obtained similar results. They were all able to segment the person. The only deviation, compared to the results obtained with the raw image, was with mid compression (mid quality) JPEG and high compression (low quality) Theora, because they both removed the noise lines. This did not happen with mid compression (mid quality) Theora.



(a) Raw image.

(b) Best case using mid compression JPEG.

(c) Best case using high compression Theora.

(d) Best case using mid compression Theora.

Figure 4.17: Best cases for high and mid compression Theora and mid compression JPEG and the reference case using raw image.

4.5 Feature extraction

Feature extraction from images in robotic vision is used in several applications. Such applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, among others. In this section we will test the effects of lossy compression on two popular feature extractors. The Scale-Invariant Feature Transform (SIFT) [74] and the Speeded Up Robust Features (SURF) [75] extractors. Both are implemented in the OpenCV library.

4.5.1 Procedure

Unlike some previous experiments, this experiment does not require any preprocessing applied before feature extraction. Required procedures by feature extractors are already applied by the OpenCV library before, such as converting the image into grayscale. Each feature extractor will be executed sequentially per image. The SURF extractor was executed with a `minHessian` of 800. The larger the `minHessian` the fewer, but theoretically more salient,

interest points are detected.

The image dataset we use in this experiment is the set of images that contain supermarket products on a monotone background, the Features Dataset (Appendix B.6, Figure 4.18).

Figure 4.18: Images in the Features image dataset.

In this experiment we record the number of features extracted with each tier of each standard and compare them with the raw reference values.

4.5.2 Experimental results

Table 4.16 shows the number of features detected for each extractor. By observing this table it is possible to see that, when using low compression (high quality), the number of features detected doesn't deviate much from the reference raw values. When using mid compression (mid quality), both H.265 and Theora show less features than the reference raw values. JPEG shows an increase in the feature count. At high compression (low quality), H.265 shows a significant reduction in the number of detected features. Theora shows a less significant reduction. JPEG, on the other hand, shows a massive increase in the number of detected features. This massive increase is attenuated when using the SURF extractor, due to the high `minHessian` value.

	SIFT				SURF			
	#1	#2	#3	#4	#1	#2	#3	#4
Raw	961	225	529	396	738	181	504	329
H 1	953	211	518	396	735	179	506	326
J 100	962	225	536	402	740	182	502	328
T 10	957	209	525	407	727	183	502	332
H 26	996	184	493	329	731	163	473	281
J 50	1071	230	537	422	742	188	504	312
T 7	937	177	465	355	726	180	471	301
H 51	535	150	336	122	539	173	385	149
J 10	1305	302	645	537	766	215	486	343
T 1	774	167	419	313	671	168	421	284

Table 4.16: Number of features extracted using SIFT and SURF for each tier of each compression standard. The number of features extracted when using raw images is also presented as a reference.

Figure 4.19 contains the worst and best case for H.265 high compression (low quality). Looking at the worst case, we can see that, there are lesser features and they are more scattered. Features also appear outside of the objects. Which means that, if we are trying to obtain features that describe the objects, we would lose details that could be important. Looking at the best case, several features disappeared. We can also see that, the features that remained are mostly inside the objects. This will improve the system performance because there are less features to process. Figure 4.20 contains the worst case for mid compression (mid quality) H.265. This figure shows that, out of the already scarce features detected using the raw image, a few disappeared using compressed H.265.

(a) High compression H.265 worst case reference.

(b) Worst case using high compression H.265.

(c) High compression H.265 best case reference.

(d) Best case using high compression H.265.

Figure 4.19: Worst and best cases using high compression H.265 and the respective raw image references.

(a) Mid compression H.265 worst case reference.

(b) Worst case using mid compression H.265.

Figure 4.20: Worst case using mid compression H.265 and the respective raw image reference.

Figure 4.21 contains the worst and best case for highly compressed (low quality) JPEG images. Looking at the worst case, a significant amount of features appeared. This will negatively impact the performance of the system. However, for the most part, the detected

features are inside the object. Apart from the performance, the system won't be affected. Looking at the best case, the obtained results are very similar to the results obtained when using the raw image. Figure 4.22 contains the worst case for mid compression (mid quality) JPEG images. This figure shows that, like highly compressed JPEG, some extra features appeared. However, they aren't in significant numbers. This will not significantly affect the system performance.

(a) High compression JPEG worst case reference.

(b) Worst case using high compression JPEG.

(c) High compression JPEG best case reference.

(d) Best case using high compression JPEG.

Figure 4.21: Worst and best cases using high compression JPEG and the respective raw image references.

(a) Mid compression JPEG worst case reference.

(b) Worst case using mid compression JPEG.

Figure 4.22: Worst case using mid compression JPEG and the respective raw image reference.

Figure 4.23 contains the worst case for high compression (low quality) Theora. This figure shows that, for the most part, the results obtained are very similar to the results obtained using the raw image. Which means that, for extracting features, using High compression Theora provides good results.

(a) High compression Theora worst case reference.

(b) Worst case using high compression Theora.

Figure 4.23: Worst case using high compression Theora and the respective raw image reference.

4.6 Final remarks

Looking at results of facial recognition, we can see that, when mixing compressed and raw data, high compression (low quality) data should be avoided. It is, however, possible to mix data while keeping a good performance, by using better quality (lower compression) data. JPEG provides the best results at low compression as the models performance did not plunge like with H.265 and Theora. Unexpectedly, if only compressed data is used, the data can be highly compressed without toll on the facial recognition accuracy.

Looking at the results of face and body detection, if only low to mid compression (high to mid quality) is used then we are able to obtain results similar to using raw data. If we use

high compression (low quality), the results will be more mixed, sometimes better, sometimes a lot worse. For face detection, the results are a bit inconsistent, which might have something to do with the way the haar cascade was trained.

Looking at the previously presented results, high compression H.265 is not suitable for finding contours, as it loses too much details and objects start to lose shape. At mid compression H.265 provides good results. Theora provides good results at high compression and even better at mid compression. JPEG provides excellent results even at high compression.

Based on the results obtained when testing color segmentation, using compressed data is possible. We can conclude that, it is possible to segment using highly compressed (low quality) data, however, there are cases where this is not advised. Using mid compression allows for better results. Highly compressed H.265 provided poor results in this experiment. The use of post processing operations might help to obtain better results if necessary.

Looking at the results of feature extraction, it is possible to use compression and obtain good results. For this, Theora provides the best results, allowing the correct extraction of features even when using high compression. Both H.265 and JPEG can obtain good results using high compression. However, poor results are also possible. So we conclude that it is best to avoid using high compression H.265 and JPEG when using feature extraction. When using mid compression both standards provide consistently good results.

What these results show is that, except on a few cases, high compression H.265 provides poor results. Theora and JPEG provide better results at high compression. For mid compression all three standards provide very good results and can be used without any major effects on the outcome of the image analysis algorithms. Looking at these results, and taking into account the average number of bits per pixel required by each standard, we can see that high compression (low quality) JPEG requires as many bits per pixel as H.265 with a CRF between 11 and 21 (depending on the data), high compression (low quality) Theora requires as many bits per pixel as H.265 with a CRF of about 16. All three CRF values are significantly above what we considered mid compression (mid quality). Therefore, mid compression H.265 uses significantly less bits than high compression JPEG and Theora resulting in best lossy compression in the experiments we conducted.

With this in mind, we conclude that it is possible to use compression without compromising the system and ROS would benefit the most with the implementation of a H.265 codec.

CHAPTER 5

Intelligent video player

In this chapter we discuss the development of an intelligent video player to help roboticists in their work while they evaluate the recorded data after experiments. The first section begins by answering the question "Why is this player necessary?". The following sections discuss the theoretical concept and implementation of the player. Some results are also presented.

One of the main problems in the development and debugging of ROS systems is the amount of data stored in the bag files. If we consider a robot that contains one or more cameras, and other kinds of sensors, that record information from the environment several times per second, the difficulty to find the required information in the bag file will increase rapidly. Although sometimes there is a possibility to only connect the system in small time intervals(to avoid the overflow of information), this isn't always the case nor it is very practical for the developer(e.g. autonomous car, surveillance camera, supermarket robot). For roboticists, the process of finding relevant information in long videos becomes very time consuming and tedious.

5.1 Concept

When codecs are encoding they generate a number of bits per unit of time, this is called the bitrate. Codecs can produce files with different types of bitrate, in this section two types will be discussed, Constant Bitrate (CBR) and Variable Bitrate (VBR).

When using a constant bitrate the codec will always generate a fixed bitrate, i.e., the size of the encoded data per unit of time will always be the same, no matter the amount of information. This could be useful for streaming data over a limited capacity channel. However, when used in video compression, it may severely reduce the quality or increase the size of the compressed video. By forcing the codec to use less bits than those required, for example on motion heavy scenes, or forcing the codec to use more bits than those required, for example

on a scene with no movement. Depending on the input, the use of padding may be necessary to obtain a constant bitrate.

When using VBR the codec can decide the amount of bits that can be used per unit of time. Either because there is too much or too little information to be encoded per unit of time. What this means for video codecs, is that when there is little temporal or spatial change between sequences of frames the encoder will drastically reduce the amount of bits produced. It is possible to conclude that whenever the real time bitrate is much smaller than the average bitrate there is little to no motion in the frame sequence. When there is little to no motion in the frame, it is possible to speed up the play of the video without roboticists missing important information.

5.2 Implementation

There are two basic things that the player requires in order to work. The first is a decoder for the video. The second is a tool that allows us to output the video to the screen and handle events (keyboard, mouse).

For the decoder we use the FFMPEG libraries, AVCodec and AVFormat. These libraries provide a high level access to the video metadata and data. For the video user interface and events handler we use the SDL library. This library provides a low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. With this two tools we implemented a very basic video player that speeds up whenever there is little to no motion in the video.

However, in order for the video player to be scalable, we also need to make use of a configuration file that contains pre-processed data to avoid the overburden of processing during play. For this pre-processing configuration file we needed to chose a lightweight data format. For this we had three options. CSV, Extensible Markup Language (XML) and JavaScript Object Notation (JSON). CSV was discarded because it simply did not have the required features for the player use case. XML was discarded because it is too complex and has too many features, making the code more complex and slower. With this in mind, we chose JSON, as it contains the necessary features at a much faster parsing speed [76]–[78], typically. The JSON parser we use is the cJSON parser.

On top of referred before, we also implement some important features, namely the ability to manually speed up or slow down the play of the video and also the ability to save a sequence of frames during play, for separate processing.

Figure 5.1 contains the basic block diagram of the architecture of the video player.

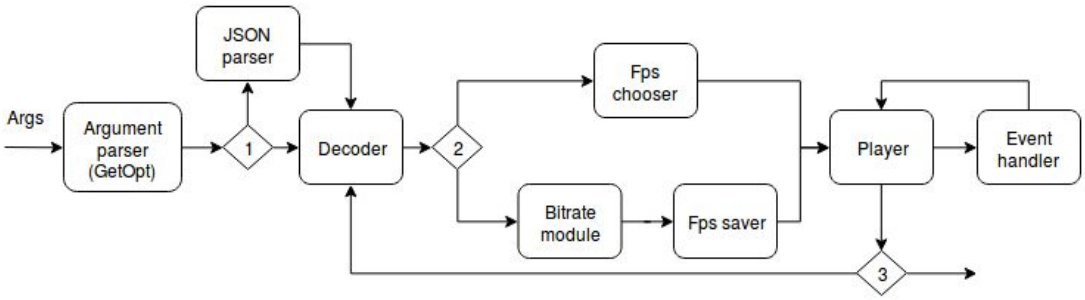


Figure 5.1: Intelligent player block diagram. Blocks 1 and 2 represent decision paths based on whether the JSON configuration file already exists or needs to be created. Block 3 represents a decision path based on whether the video has reached the end or not.

To test the implementation of the player, we used the Alboi Mixed dataset (Appendix B.2) to create a video. This video lasts 42.7 seconds, 20 of which contain little movement. The player is able to play this video in 27 seconds, taking only 4 seconds to play the parts with little movement, which originally lasted 20 seconds. Figure 5.2 contains the frames where the player started and stopped accelerating the play of the video. It is possible to see that the player stopped the acceleration when the movement started.

(a) Frame where acceleration started. **(b)** Frame where the acceleration stopped.

Figure 5.2: Interval of frames where the player accelerated the play. Between the start and stop frames there are 151 frames.

CHAPTER 6

Conclusion

In this thesis we present studies on the ROS native compression and state of the art compression standards in the context of sensor data in robotic applications. Furthermore, we studied the effects of those standards in several popular image analysis algorithms. An intelligent video player is also developed that may help roboticists in their work.

Chapter 2 presents a study on the ROS native compression standards and general purpose compressors. In this chapter we conduct some measurements on the performance of Bzip2, LZ4, JPEG, PNG and Theora. LZ4 proves to be very fast and good at compression during recording, albeit providing poor compression ratios. Bzip2 proves to be good at compression after recording, providing very good compression ratios. For lossless image compression PNG proves to be unsuitable for real time systems and provides mediocre compression performance. For lossy image compression, JPEG provides sufficient compression ratios that, for long recording times, do not completely solve the problem at hand. For lossy video compression Theora provides good compression ratios.

Chapter 3 presents a study on newer state of the art image/video compression standards. Moreover, we present alternatives to the general purpose compressors Bzip2 and LZ4. In this chapter we conduct some tests to measure both the native standards and the state of the art standards. We also measure the benefits that using the referred state of the art standards would bring to ROS. We conclude that H.265 could partially solve the storage and transmission problem. Furthermore, H.265 also provides a plethora of tools and configurations that not only allow for lossy and lossless image compression (using only intra-frames) but also for lossy and lossless video compression. As such, ROS would benefit the most with the implementation of a H.265 codec

Using H.265 lossy compression would significantly reduce the storage and transmission problem. Since, lossy compression works by discarding information, it is necessary to see how image analysis algorithm deal with lossy compressed images. Chapter 4 presents a detailed study on the effects of lossy compression on several popular image analysis algorithms. We test the effects of using H.265, JPEG and Theora on facial recognition, object detection (face and

body), finding contours, high level color segmentation and feature extraction. We conclude that, by using H.265 there would be no significant penalty on the referred image analysis algorithms while maintaining the smallest number of bits per pixel.

Despite H.265 significantly reducing the storage needs and helping with data transmission, it does not help directly with finding useful information. For this, we develop a intelligent video player. Chapter 5 describes the concept and implementation of a player. This player helps skip irrelevant information during video play, improving the problem of finding information.

6.1 Future work

At the end of this thesis, we observe that more work could be done, either in the storage and transmission problem or in the information finding problem. Some possible directions for future work are presented next.

- Although image/video data is one of the most consuming data, it is not the only one. Work could be done in other types of data that also consume a lot of bits, for example, point clouds.
- We tested several image analysis algorithms. There are many more that are relevant to robotics. Some other algorithms, for example image analysis algorithms based on deep learning, could be tested to provide a deeper understanding of the effects of lossy compression.
- If the provided compression is not enough, in some cases, a semantic compression algorithm could be developed that analyses images and discards non-relevant information.
- Improve the usability of the intelligent player, either by developing a GUI or by incorporating the concept into an open source video player, such as VLC.

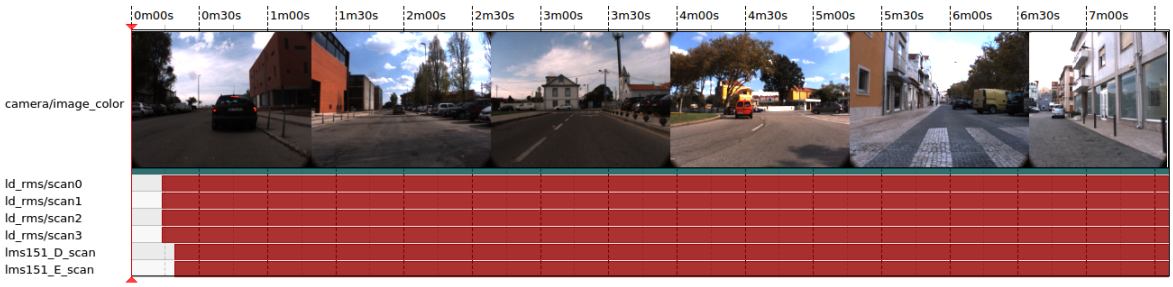
APPENDIX A

Bags

In this appendix we present a detailed description of all the bags used throughout the development of this thesis.

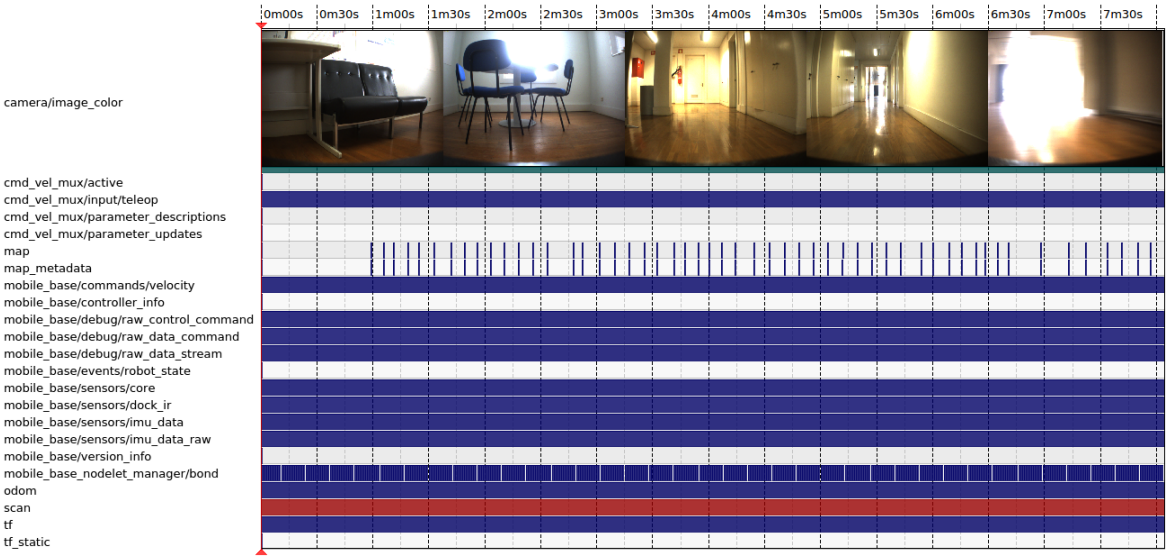
A.1 Alboi

The alboi bag contains data from sensors contained in the ATLAS car from the university of aveiro. It contains data from one color camera with a resolution of 1624x1224 and several lasers. It contains 135379 messages for a total size of 18430MiB. If we filter the image messages, we obtain a bag with 3194 messages and a total size of 18166MiB. This bag has a duration of 484 seconds.



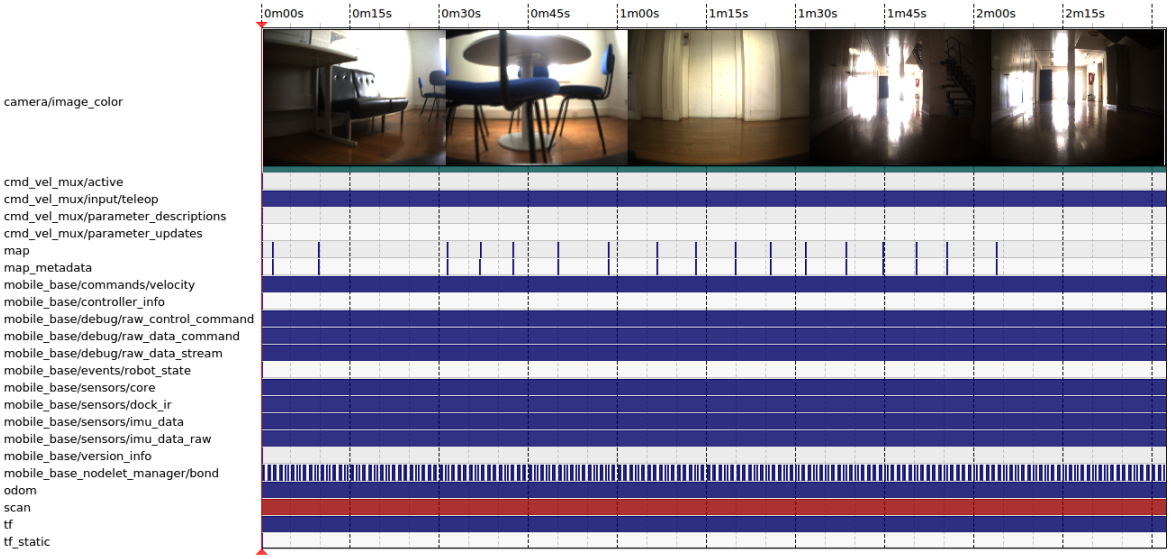
A.2 P0 Large

The P0 Large bag contains data from sensors contained in a robot that is used in IEETA. It contains images with a resolution of 1296x964, laser scans, odometry data and debugging system messages. It contains 155337 messages for a total size of 13051MiB. If we filter the image messages, we obtain a bag with 3387 messages and a total size of 12108MiB. This bag has a duration of 456 seconds.



A.3 P0 Small

The P0 Small bag contains the same type of data found in the P0 Large dataset but at different recording times and locations. It contains 48546 messages for a total of 4120MiB. Filtering the images we get 1065 messages and a total size of 3808MiB. This bag has a duration of 152 seconds.



APPENDIX B

Image Datasets

In this appendix we present a detailed description of each image dataset used throughout this thesis.

B.1 Alboi Moving

The Alboi Moving image dataset consists of 300 sequential frames contained in the Alboi bag. The sequence of frames contained in this dataset only contains motion.

B.2 Alboi Mixed

The Alboi Mixed image dataset consists of 300 sequential frames contained in the Alboi bag. The first half of this image dataset contains no motion. The second half only contains sequences with motion.

B.3 P0 Small

The P0 Small image dataset consists of the first 300 sequential frames contained in the P0 Small bag. This dataset only contains data with motion.

B.4 P0 Large

The P0 Large image dataset consists of the first 300 sequential frames contained in the P0 Large bag. This dataset only contains data with no motion and small variations in the lighting conditions.

B.5 People

The People image dataset contains 6 images. All images contain at least a body and a face.

B.6 Features

B.7 Database of Faces

The Database of Faces (formerly 'The ORL Database of Faces) dataset from AT&T. [1]

[1] https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

APPENDIX C

Code Samples

C.1 Lossy effects experiment launcher script

```
source /opt/ros/kinetic/setup.bash
source /home/alvaro/Documents/Universidade/Tese/thesis_workspace/devel/setup.bash
roscore &
sleep 3
IMAGES='image%04d.ppm'

##FACE RECOGNITION
echo -e "\n FACE RECOGNITION \n"
for (( COUNTER=1; COUNTER<=51; COUNTER+=5 )); do
echo -e "\n h265 $COUNTER \n"
rosparam set lsy_csv "/home/alvaro/Documents/orl_faces/csv/h265_$COUNTER"
rosrun bca_effects bca_effects_facerecon_node
done

for (( COUNTER=10; COUNTER<=100; COUNTER+=10 )); do
echo -e "\n jpeg $COUNTER \n"
rosparam set lsy_csv "/home/alvaro/Documents/orl_faces/csv/jpeg_$COUNTER"
rosrun bca_effects bca_effects_facerecon_node
done

for (( COUNTER=1; COUNTER<=10; COUNTER+=3 )); do
echo -e "\n theora $COUNTER \n"
rosparam set lsy_csv "/home/alvaro/Documents/orl_faces/csv/theora_$COUNTER"
rosrun bca_effects bca_effects_facerecon_node
done
```

C.2 JPEG and PNG image (de)compression methods

```
my_cvt() {
time(
for file in $(ls *.ppm); do
convert $file -quality $1 ${file%.*}.$2;
done
)
}

my_decvt() {
time(
for file in $(ls *.$1); do
convert $file ${file%.*}.ppm;
done
)
}
```

C.3 JPEG, Theora and H.265 dataset compression methods

```
for (( COUNTER=10; COUNTER<=100; COUNTER+=10 )); do
echo "jpeg $COUNTER quality"
mkdir -p ../jpeg_$COUNTER
my_cvt $COUNTER "jpg"
mv *.jpg ../jpeg_$COUNTER/
done

for (( COUNTER=1; COUNTER<=10; COUNTER+=3 )); do
echo "theora $COUNTER"
mkdir -p ../theora_$COUNTER
ffmpeg -framerate 7 -f image2 -i image%04d.ppm
-codec:v libtheora -qscale:v $COUNTER  theora_$COUNTER.ogv
mv theora_$COUNTER.ogv ../theora_$COUNTER
done

for (( COUNTER=1; COUNTER<=51; COUNTER+=5 )); do
echo "h265 $COUNTER"
mkdir -p ../h265_$COUNTER
ffmpeg -framerate 7 -f image2 -i image%04d.ppm
-c:v libx265 -crf $COUNTER -hide_banner -loglevel panic
-preset ultrafast -pix_fmt yuv444p h265_$COUNTER.mp4
mv h265_$COUNTER.mp4 ../h265_$COUNTER
done
```

APPENDIX D

Software and hardware information

D.1 Software

D.1.1 Operative system

The used operating system was Ubuntu version 16.04.4 LTS.

D.1.2 ROS

The used ROS version was ROS Kinetic Kame LTS version.

D.1.3 OpenCV

The used OpenCV version was version 3.3.1.

D.1.4 codecs

All results and benchmarks presented throughout this thesis have been obtained using open source implementations. Their implementations and necessary steps for installation are presented below (where applicable).

H.265

For H.265 ffmpeg static version 3.4.2 was used with the embedded codec x265 version 2.6+39-01b685d6fa33.

H.264

For H.264 ffmpeg static version 3.4.2 was used with the embedded libx264 version.

Theora

For Theora ffmpeg static version 3.4.2 was used with libtheora version 1.1.1.

PNG

For PNG libpng version 1.2.54 was used.

JPEG

libjpeg version 8c was used

BPG

For BPG version 0.9.7 was used. For compilation the makefile needs to be patched and additional dependencies must be installed on ubuntu 16.04 LTS.

```
first install dependencies:
$ sudo apt-get install -y libsdl-image1.2-dev libsdl1.2-dev libjpeg8-dev yasm
download libpng version 1.6.34
unpack and enter directory
$ ./configure
$ make
$ sudo make install
Download BPG v0.9.7 from https://bellard.org/bpg/
unpack and enter directory
before proceding there is a need to patch the makefile.
insert the two following directives:
"CFLAGS+=-I/usr/local/include" after the line "CFLAGS+=-I."
"LDFLAGS+=-L /usr/local/lib" before the line "CFLAGS+=-g"
replace the following line:
LIBS:=-lrt
with:
LIBS:=-lrt -lnuma
$ make
$ sudo make install
$ sudo ldconfig
```

JPEG2000

For JPEG2000 OpenJPEG version 2.3.0. For installation the following steps where necessary:

```
download version v2.3.0 source code from https://github.com/uclouvain/openjpeg/releases/
unpack it and enter the unpacked directory
$ sudo apt-get install liblcms2-dev libtiff-dev libpng-dev libz-dev
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_C_FLAGS="-O3 -msse4.1 -mavx2 -DNDEBUG"
$ make
$ sudo make install
```

JPEG-LS

For JPEG-LS a patched libjpeg that supports JPEG-LS was used. To avoid conflicts with the already installed libjpeg, the executable obtained by this steps is not installed in the system

but is used on the current directory. Later when doing the experiments the full path to the executable needs to be given. For compilation the following steps where taken:

```
download version v2.3.0 source code from https://github.com/thorfdbg/libjpeg
unpack it and enter the unpacked directory
$ ./configure
$ make
```

D.2 Hardware

D.2.1 Computer

Laptop	CPU	RAM	GPU	Disk
Asus	i5-5200U	12GB	Intel HD 5500	Crucial Mx200 250GB

D.2.2 Camera

All images we recorded were obtained using a Chameleon CMLN-13S2C-CS camera.

References

[1] J. Seward, *Bzip2 homepage*. [Online]. Available: http://www.bzip.org/index.html (visited on 02/16/2018).

[2] ——, "Bzip2 and libbzip2, version 1.0.5 manual", 2007.

[3] Y. Collet, *Lz4 homepage*. [Online]. Available: http://lz4.github.io/lz4/.

[4] ——, *Lz4 explained*, 2011. [Online]. Available: https://fastcompression.blogspot.pt/2011/05/lz4-explained.html.

[5] G. Roelofs, *PNG: The Definitive Guide*. O'Reilly Media, 1999, p. 340, ISBN: 978-1565925427.

[6] G. K. Wallace, "The jpeg still picture compression standard", *The JPEG Still Picture Compression Standard*, pp. 1–17, 1991.

[7] X. Foundation, *Theora faq*. [Online]. Available: https://theora.org/faq/%7B%5C#%7D10.

[8] T. Acharya and P.-S. Tsai, *JPEG2000 Standard for Image Compression*. 2004, p. 296, ISBN: 9780471653745. DOI: 10.1002/0471653748.

[9] ITU (International Telecomunication Union), "Information technology – lossless and near-lossless compression of continuous-tone still images – baseline", *Source*, vol. 87, pp. 1–75, 1998.

[10] F. Bellard, *Bpg specification*, 2018. [Online]. Available: https://bellard.org/bpg/bpg_spec.txt.

[11] I. Richardson, *H.264 and MPEG-4 Video Compression*. 2003, p. 306, ISBN: 0470848375.

[12] T. Wiegand, "Overview of the h. 264/avc video coding standard", *... and Systems for Video ...*, vol. 13, no. 7, pp. 560–576, 2003, ISSN: 1051-8215. DOI: 10.1109/TCSVT.2003.815165.

[13] K. R. Rao, D. N. Kim, and J. J. Hwang, *Video coding standards*. 2014, ISBN: 978-94-007-6741-6. DOI: 10.1007/978-94-007-6742-3.

[14] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012, ISSN: 10518215. DOI: 10.1109/TCSVT.2012.2221191.

[15] M. Budagavi, G. J. Sullivan, and V. Sze, *High Efficiency Video Coding (HEVC)*. 2014, p. 384, ISBN: 978-3-319-06894-7. DOI: 10.1007/978-3-319-06895-4.

[16] O. S. R. Foundation, *Ros homepage*. [Online]. Available: http://www.ros.org/.

[17] ——, *Ros-pub-sub image*. [Online]. Available: http://docs.ros.org/independent/api/rep/html/rep-0106/ros-pub-sub.png.

[18] ——, *Bags format 2.0*. [Online]. Available: http://wiki.ros.org/Bags/Format/2.0 (visited on 06/01/2018).

[19] P. Mihelich and J. Kammerl, *Compressed_image_transport*. [Online]. Available: http://wiki.ros.org/compressed_image_transport (visited on 02/16/2018).

[20] P. Mihelich and E. Dreyfuss, *Theora_image_transport*. [Online]. Available: http://wiki.ros.org/theora_image_transport (visited on 02/16/2018).

[21] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm", *Algorithm, Data Compression*, no. 124, p. 18, 1994, ISSN: 15708667. DOI: 10.1.1.37.6774. arXiv: 0908.0239.

[22] D. A. Huffman, "A method for the construction of minimum-redundancu codes", *A Method for the Construction of Minimum-Redundancu Codes*, pp. 1098–1102, 1952.

[23] J. Ziv and A. Lempel, "A universal algorithm for sequencial data compression", *IEEE Transactions on Information Theory*, vol. IT-23, no. 3, pp. 337–343, 1977.

[24] G. Roelofs, *An introduction to png*. [Online]. Available: http://www.libpng.org/pub/png/book/chapter01.html#png.ch01.div.2.

[25] J. Arvo, *Graphics gems II*. 1991, pp. 93–100, ISBN: 0120644819.

[26] P. Deutsch, "Deflate compressed data format specification version 1.3", pp. 1–15, 1996, ISSN: 2070-1721. DOI: 10.17487/rfc1951.

[27] T. Halbach, "Comparison of open and free video compression systems", *International Conference on Imaging Theory and Applications (IMAGAPP)*, 2009.

[28] X. Foundation, *Theora update 20090507*. [Online]. Available: https://people.xiph.org/~xiphmont/demo/theora/demo7.html.

[29] J. Alakuijala and Z. Szabadka, "Brotli", pp. 1–128, 2016, ISSN: 2070-1721.

[30] M. Izdebski, *Parallel bzip2 compression utility*. [Online]. Available: http://lbzip2.org/ (visited on 05/28/2018).

[31] Squash, *Squash compression benchmark*. [Online]. Available: https://quixdb.github.io/squash-benchmark/ (visited on 05/28/2018).

[32] M. Mahoney, *Large text compression benchmark*, 2018. [Online]. Available: http://mattmahoney.net/dc/text.html.

[33] G. Liu, *Multi-threaded compression benchmarks*, 2012. [Online]. Available: https://vbtechsupport.com/1614/.

[34] A. Jakulin, *Jpeg and jpeg2000 artifacts*. [Online]. Available: http://www.stat.columbia.edu/~jakulin/jpeg/artifacts.htm (visited on 02/20/2018).

[35] D. Taubman, "High performance scalable image compression with ebcot", *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000, ISSN: 10577149. DOI: 10.1109/83.847830.

[36] G. Schaefer, R. Starosolski, and S. Y. Zhu, "An evaluation of lossless compression algorithms for medical infrared images", *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pp. 1673–1676, 2005, ISSN: 1557-170X. DOI: 10.1109/IEMBS.2005.1616764.

[37] V. Bui, L. C. Chang, D. Li, L. Y. Hsu, and M. Y. Chen, "Comparison of lossless video and image compression codecs for medical computed tomography datasets", *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, pp. 3960–3962, 2016. DOI: 10.1109/BigData.2016.7841075.

[38] R. Ansari and N. Memon, "The jpeg lossless stantards", pp. 1–24, 1999.

[39] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls", *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, 2000, ISSN: 10577149. DOI: 10.1109/83.855427.

[40] S. Martucci, "Reversible compression of hdtv images using median adaptive prediction and arithmetic coding", *IEEE International Symposium on Circuits and Systems*, pp. 1310–1313, 1990, ISSN: 0278-0062. DOI: 10.1109/ISCAS.1990.112371.

[41] S. W. GOLOMB, "Run-length encodings", *IEEE Transactions on Information Theory*, vol. IT-12, no. 3, pp. 399–401, 1966, ISSN: 15579654. DOI: 10.1109/TIT.1966.1053907. arXiv: arXiv:1011.1669v3.

[42] R. F. Rice and J.-j. Lee, "Some practical universal noiseless coding techniques , part ii", *JPL Publication*, vol. 17, 1983, ISSN: 03610748.

[43] M. Karczewicz and R. Kurceren, "The sp- and si-frames design for h.264/avc", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 637–644, 2003, ISSN: 10518215. DOI: 10.1109/TCSVT.2003.814969.

[44] I. E. Richardson, *THE H . 264 ADVANCED VIDEO COMPRESSION Second Edition*. 2010, ISBN: 9780470516928.

[45] W. Pratt, J. Kane, and H. Andrews, "Hadamard transform image coding", *Proceedings of the IEEE*, vol. 57, no. 1, pp. 58–68, 1969, ISSN: 0018-9219. DOI: 10.1109/PROC.1969.6869.

[46] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, 2003, ISSN: 10518215. DOI: 10.1109/TCSVT.2003.815173.

[47] C. M. Fu, E. Alshina, A. Alshin, Y. W. Huang, C. Y. Chen, C. Y. Tsai, C. W. Hsu, S. M. Lei, J. H. Park, and W. J. Han, "Sample adaptive offset in the hevc standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, 2012, ISSN: 10518215. DOI: 10.1109/TCSVT.2012.2221529.

[48] W. C. Feng, R. Feng, P. Wyatt, and F. Liu, "Understanding the impact of compression on feature detection and matching in computer vision", *Proceedings - 2016 IEEE International Symposium on Multimedia, ISM 2016*, pp. 457–462, 2017. DOI: 10.1109/ISM.2016.140.

[49] J. D. Paola and R. A. Schowengerdt, "The effect of lossy image compression on image classification", *International Geoscience and Remote Sensing Symposium (IGARSS)*, vol. 1, pp. 118–120, 1995. DOI: 10.1109/IGARSS.1995.519665.

[50] P. Korshunov and W. T. Ooi, "Video quality for face detection, recognition, and tracking", *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 7, no. 3, pp. 1–21, 2011, ISSN: 15516857. DOI: 10.1145/2000486.2000488.

[51] T. Ahonen, A. Hadid, M. Pietikäinen, and M. Pietik, "Face recognition with local binary patterns", *Proc. of the European Conference on Computer Vision (ECCV)*, pp. 469–481, 2004, ISSN: 03029743. DOI: 10.1007/978-3-642-25449-9_2.

[52] L. Wang and D.-C. He, "Texture classification using texture spectrum", *Pattern Recognition*, vol. 23, no. 8, pp. 905–910, 1990, ISSN: 00313203. DOI: 10.1016/0031-3203(90)90135-8.

[53] T. Ojala, M. Pietikainen, and D. Harwood, "Performance evaluation of texture measures with classification based on kullback discrimination of distributions", *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, pp. 582–585, 1994, ISSN: 00313203. DOI: 10.1109/ICPR.1994.576366.

[54] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions", *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, 1996, ISSN: 00313203. DOI: 10.1016/0031-3203(95)00067-4.

[55] Kelvin Salton, *Face recognition: understanding lbph algorithm – towards data science*, 2017. [Online]. Available: https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b (visited on 06/01/2018).

[56] N. Stekas and D. Van Den Heuvel, "Face recognition using local binary patterns histograms (lbph) on an fpga-based system on chip (soc)", *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pp. 300–304, 2016. DOI: 10.1109/IPDPSW.2016.67.

[57] OpenCV, *Face recognition with opencv*. [Online]. Available: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html (visited on 05/29/2018).

[58] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–511–I–518, 2001, ISSN: 1063-6919. DOI: 10.1109/CVPR.2001.990517. arXiv: arXiv:1011.1669v3.

[59] M. J. Jones and P. Viola, "Robust real-time object detection", *Workshop on statistical and computational theories of vision*, vol. 266, p. 56, 2001, ISSN: 09205691. DOI: http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb. arXiv: arXiv:1011.1669v3.

[60] Y.-Q. Wang, "An analysis of the viola-jones face detection algorithm", *Image Processing On Line*, vol. 4, pp. 128–148, 2014, ISSN: 2105-1232. DOI: 10.5201/ipol.2014.104.

[61] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017, ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[62] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection", 2015, ISSN: 01689002. DOI: 10.1109/CVPR.2016.91. arXiv: 1506.02640.

[63] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017, ISSN: 0146-4833. DOI: 10.1109/CVPR.2017.690. arXiv: 1612.08242.

[64] OpenCV, *Cascade classifier*. [Online]. Available: https://docs.opencv.org/3.0-beta/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html (visited on 05/29/2018).

[65] ——, *Opencv haarcascades*. [Online]. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades.

[66] S. Suzuki and K. A. Be, "Topological structural analysis of digitized binary images by border following", *Computer Vision, Graphics and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985, ISSN: 0734189X. DOI: 10.1016/0734-189X(85)90016-7.

[67] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986, ISSN: 01628828. DOI: 10.1109/TPAMI.1986.4767851.

[68] X. M. Zhao, W. X. Wang, and L. P. Wang, "Parameter optimal determination for canny edge detection", *The Imaging Science Journal*, vol. 59, no. 6, pp. 332–341, 2011, ISSN: 1368-2199. DOI: 10.1179/136821910X12867873897517.

[69] A. Rosebrock, *No titlezero-parameter, automatic canny edge detection with python and opencv*, 2015. [Online]. Available: https://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/.

[70] OpenCV, *Finding contours in your image*. [Online]. Available: https://docs.opencv.org/2.4.13.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html (visited on 05/29/2018).

[71] A. Baddeley, "An error metric for binary images", *Robust Computer Vision: Quality of Vision Algorithms*, no. march 1992, pp. 59–78, 1992. DOI: 10.1.1.52.3879.

[72] C. Rother, V. Kolmogorov, and A. Blake, ""Grabcut": interactive foreground extraction using iterated graph cuts", *ACM Transactions on Graphics*, vol. 23, no. 3, p. 309, 2004, ISSN: 07300301. DOI: 10.1145/1015706.1015720. arXiv: arXiv:1011.1669v3.

[73] OpenCV, *Interactive foreground extraction using grabcut algorithm*. [Online]. Available: https://docs.opencv.org/trunk/d8/d83/tutorial_py_grabcut.html (visited on 05/29/2018).

[74] D. Lowe, "Object recognition from local scale-invariant features", *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1150–1157 vol.2, 1999, ISSN: 0-7695-0164-8. DOI: 10.1109/ICCV.1999.790410. arXiv: 0112017 [cs].

[75] H. Bay, A. Ess, T. Tuytelaars, and L. Gool, "Speeded-up robust features (surf)", *Computer Vision and Image Understanding (CVIU)*, no. September, pp. 404–417, 2006.

[76] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of json and xml data interchange formats: a case study", *Scenario*, vol. 59715, pp. 1–3, 2009, ISSN: 09505849. DOI: 10.1016/j.infsof.2010.03.005. arXiv: 1003.3338.

[77] B. Gil and P. Trezentos, "Impacts of data interchange formats on energy consumption and performance in smartphones", *Proceedings of the 2011 Workshop on Open Source and Design of Communication - OSDOC '11*, p. 1, 2011. DOI: 10.1145/2016716.2016718.

[78] K. Maeda, “Performance evaluation of object serialization libraries in xml, json and binary formats”, *2012 2nd International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2012*, pp. 177–182, 2012. DOI: 10.1109/DICTAP.2012.6215346.