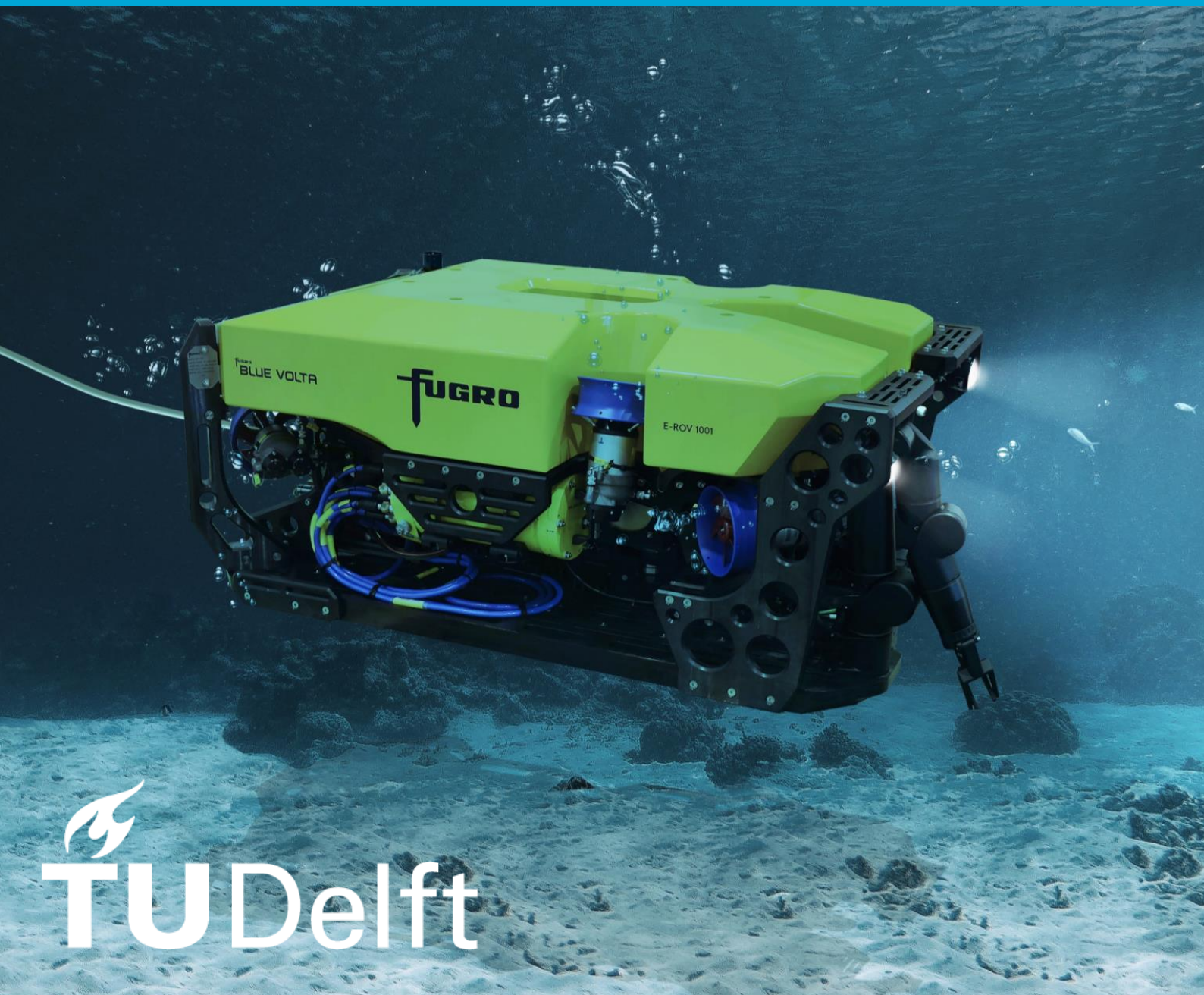
MSc Thesis
Embedded Systems

Compression Rate Optimisation for Streaming of Underwater Video

Daniël Panis
2023



TU Delft

COMPRESSION RATE OPTIMISATION FOR STREAMING OF UNDERWATER VIDEO

A thesis submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of

Master of Science in Embedded Systems

by

Daniël Panis

February 2023

Daniël Panis: *Compression Rate Optimisation for Streaming of Underwater Video* (2023)

The work in this thesis was made in the:

Embedded and Networked Systems group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Supervisors:	Prof.dr. Koen Langendoen ir. Igor Dorrestijn
Co-reader:	Prof.dr.ir. Inald Lagendijk

ABSTRACT

Compression of underwater video data can be a challenging task due to the unique environment of the underwater world. Low visibility, high noise levels, and a specific colour range all contribute to less efficient video encoding. This research aims to investigate ways to improve the video compression rate of Fugro's underwater Remote Operated Vehicle (ROV) for efficient transmission of video data over limited bandwidth satellite connections.

The Fugro project started by developing a streaming application for a ROV that locally processes and encodes the video signal from six onboard surround cameras. After the development of this application, ways to improve the compression rate were investigated.

Video codecs often exploit the correlation between neighbouring pixels. That is, a more homogeneous image in both space and time would require fewer bits to encode. When light scatters on small particles, such as plankton or sediment, it can create noise in the form of grains or speckles in the video. This noise can reduce spatial and temporal correlation in underwater video, which can negatively affect the encoding process.

In this research, methods to filter the video from this noise are investigated and optimised for compression efficiency. Therefore, two temporal filters were developed to remove noise from underwater videos: A weighted moving average filter and a median filter.

The filters were evaluated using two main metrics: computation time and compression rate improvement. The performance of the filters was evaluated on the basis of the improvement in compression rate, where the unfiltered compressed video was taken as the baseline. The temporal filter was able to achieve a significant reduction in noise and motion blur in the video sequences while preserving the sharpness of the frames. By increasing the temporal correlation with the temporal filters, the video can be compressed up to 30% more efficiently with respect to the unfiltered video. The results also showed that the filtered videos were of comparable or higher quality than the unfiltered video.

Temporal filters are effective in improving the compression rate and the quality of a video by reducing noise and other unwanted artefacts. They are, however, computationally inefficient in their current form, making it difficult to apply them in real-time applications in their current form.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

1 INTRODUCTION

1.1 CONTEXT

Robotics technology has revolutionised manufacturing, design, and communication across industries [1]. In particular, video data has become an essential part of the robotics sensing system because of its ability to provide high-definition images and real-time visual feedback about the environment and the robot's surroundings. These data can be used for navigation, path planning, and to detect objects and track their movement, so that they can be operated from a remote location or even fully autonomously.

Today, such remote-operated systems are often equipped with even more video cameras, while the quality of these cameras continues to increase. This requires more local storage space and a larger amount of bandwidth to transmit these video data. Especially the last can be problematic in remote areas, where these data typically get transmitted wirelessly via a satellite connection. The amount of data that can be transmitted in real time over a given amount of bandwidth is limited by the size of the transmitter and the type of subscription in such systems. This factor drives the need for more efficient video compression and field-specific solutions.

1.1.1 Fugro's remote operated vehicle

This research will focus on optimising the compression rate for Fugro's underwater ROV (Figure 1.1 a). This ROV is designed to be deployed from an Uncrewed Surface Vessel (USV) (Figure 1.1 b). This innovative technology is specifically used to conduct underwater structure inspections in remote locations and is designed to improve offshore operations through its remote and autonomous capabilities. It is a compact, highly flexible, and extremely powerful electric ROV, capable of operating at a depth of 450 metres.

(a) (b)

Figure 1.1: (a) Underwater remote operated vehicle, (b) Uncrewed surface vessel

The ROV is equipped with a sophisticated array of sensors that allows it to capture detailed data and images underwater. The collected data are then transmitted to the USV through a cable. From the USV, the data are streamed in real-time via a sattelite connection, allowing operators on shore to monitor and analyse the vehicle's progress. Additionally, the vehicle is equipped with a robotic arm that can be used to perform precise operations, such as collecting samples and manipulating objects.

The ROV offers a solution for performing submerged structure inspections, such as oil and gas platforms, wind turbines, and pipelines. Furthermore, the vehicle can be used to carry out research and exploration activities in areas difficult to access. The vehicle is also highly reliable and robust and can operate in harsh conditions, such as strong currents and low visibility. The ROV provides a safe and efficient way to inspect submerged structures and can be used to reduce the risk of human error. It is also a cost-effective solution, as it eliminates the need for expensive divers and manned vessels.

Fugro's latest surround vision system featuring embedded processing power will be embedded within the ROV. This high-fidelity vision system enables visual localisation, point cloud generation, target tracking, augmented reality, and will aid in future ROV autonomous navigation. The system will consist of six cameras that cover all angles around the ROV. To transmit the video data through the limited bandwidth satellite connection, it is essential to process and compress the video signal as efficiently as possible.

1.2 CHALLENGES

Video compression has a long history and is well understood, but compression of underwater videos remains a challenging task due to the unique environment of the underwater world. The lack of light, the lack of colours, the dynamic movement, and the slow shutter speeds all contribute to the difficulty of efficient video compression.

1.2.1 Low visibility

Underwater video is often characterised by low visibility due to the number of suspended particles in the water. These particles reduce the amount of light that reaches the camera and scatter the remaining light, resulting in a low-contrast video. This makes it difficult to distinguish objects in the video, making it harder to get good video quality with traditional compression techniques. Even with high bit rates, there may be significant compression artefacts in the video due to low contrast [2]. Low light in underwater environments makes it difficult to capture clear images; therefore, images are often blurry or distorted.

1.2.2 Colour

As light passes through the water, it changes in intensity and wavelength spectrum, resulting in a different colour than that seen on the surface. This means that underwater video compression must take into account the different colour palette of underwater objects to accurately represent them in the compressed video [3]. Furthermore, underwater video compression must also take into account the rapid changes in light and colour that can occur underwater. As light changes due to the movement of water, the colours of objects in the video may change rapidly [4]. This means that the video compression algorithm must be able to accurately represent these changes to maintain a high level of quality in the resulting compressed video.

1.2.3 Noise

Underwater video is often disturbed by so-called "marine snow" (Figure 1.2). Marine snow consists of an aggregation of organic detritus, microorganisms, and clay minerals [5]. These particles are called marine snow because they have a visual resemblance to snowflakes falling from the sky. Although marine snow has many vital applications for marine life, in underwater video it can also be considered as noise. This noise is caused by the scattering of light on these particles, which can make it difficult to accurately capture the details of objects in the video, resulting in a loss of quality.

Figure 1.2: Marine snow

Video codecs often exploit the correlation between pixels, both within a single frame (spatial) and between pixels in successive frames (temporal). That is, a more homogeneous image in both space and time would require fewer bits to encode. The presence of marine snow not only reduces the quality of the video, but also causes an attenuated spatial and temporal correlation, resulting in a lower performance of the compression algorithms [6].

1.3 PROBLEM STATEMENT AND RESEARCH QUESTIONS

This research will focus on optimising the compression rate for one of Fugro's underwater Remotely Operated Vehicles. The challenge of this task is compounded by the unique environment of the underwater world, which is characterised by low visibility, colour changes, and noise. The goal of this research is to investigate ways to improve the video compression rate of Fugro's ROVs for efficient transmission of video data over limited bandwidth satellite connections. The following research questions are being investigated in this thesis:

1. What technique can be used to reduce the amount of bandwidth required to transmit underwater video data wirelessly over a satellite connection?
2. How much bandwidth can be saved using this technique?
3. How does this technique affect other constraints, such as computation time and video quality?

1.4 SOLUTIONS AND CONTRIBUTIONS

In order to answer the research questions, first a literature study was conducted to determine what techniques were already available, providing a baseline for further

development. After extensive research, several filters were developed and tested to reduce noise and optimise the compression rate. Two types of motion compensation algorithms were compared for improving computational efficiency, quality, and compression rate. Experiments were conducted to compare the performance of the proposed filters with each other. Furthermore, an application for Fugro's ROV was developed and tested to stream the data of six video cameras in real time.

1.5 THESIS OUTLINE

In Chapter 2, background information about video compression is described. There, the fundamental techniques for the encoding of video signals are explained. Also, existing related research is analysed and described in this chapter. The design and performance of the streaming application developed for Fugro are described in Chapter 3. Chapter 4 describes the optimisation of the compression rate by denoising the video signal using specific filters. In Chapter 5, the results of the compression rate optimisation filters are presented and in Chapter 7 the results of this research are discussed and concluded. Future recommendations are also given in Chapter 7.

2 BACKGROUND

In this chapter, the background is presented, where the tools and concepts used to analyse the problems stated in Section 1.3 are described. In Section 2.1, the fundamentals of video encoding are explained, where information is given on some of the most widely used video encoding standards and the primary techniques of video encoding are briefly explained. Section 2.2 gives information on relevant research done by others on this subject. Finally, Section 2.3 discusses the research gap that this research focusses on.

2.1 VIDEO ENCODING

Video encoding is the process of converting video signals from one format to another for efficient transmission or storage. It involves compressing the video data to reduce its size, while maintaining its quality as much as possible. This is typically done using a specific set of standards, which dictate the specific methods and algorithms used to compress and encode the video data. There are several different video encoding standards that are used for this purpose, each with its own unique set of features and capabilities.

A key aspect of video encoding standards is the bitrate, which is the rate at which the video is encoded. The bitrate is expressed in bits per second (bps) and is determined by the resolution of the video, the frame rate, and the type of video encoding used. A higher bitrate will result in higher quality video, but it will also use more bandwidth, which can increase the cost of streaming services.

2.1.1 Coding Standards

One of the most widely used video encoding standards is H.264, also known as MPEG-4 Part 10 or Advanced Video Coding (AVC). It was developed by the International Telecommunications Union (ITU) and the Motion Picture Experts Group (MPEG) [7]. H.264 is a relatively old standard, having been released in 2003, but is still widely used due to its efficiency and compatibility with many devices. This standard is commonly used for a variety of applications, including streaming video, video conferencing, and high-definition television.

H.265, also known as High Efficiency Video Coding (HEVC), is a more recent video encoding standard that was developed by the Joint Collaborative Team on Video Coding (JCT-VC) [8]. It uses advanced techniques such as sample adaptive offset (SAO) and adaptive quantisation (AQ) to achieve even higher levels of compression than H.264, making it ideal for applications where bandwidth is limited. H.265 is supported by many newer devices, including high-end smartphones and TVs, and is used by some streaming services for its 4K content.

In addition to H.264 and H.265, there are several other video encoding standards that are commonly used. The VP9 standard, developed by Google, is widely used to stream video over the internet. It uses a variety of advanced techniques, including interframe prediction and multiresolution encoding, to achieve high levels of

compression while maintaining good video quality [9]. The AV1 standard, developed by the Alliance for Open Media, is a royalty-free successor to VP9 that is also gaining popularity for streaming video over the Internet [10].

In general, video encoding standards play a crucial role in allowing us to efficiently transmit and store digital video. The development of these standards continues to evolve with the goal of providing higher video quality at lower bit rates.

2.1.2 Fundamental encoding techniques

Raw video signal consists of the data for each pixel in each separate frame. With a resolution of 1920x1080, that is, approximately 5 MB per frame. At 30 frames per second, this builds up to about 150 MB per second. However, video analysis has shown that there is a strong correlation between both spatial elements within one frame and temporal elements in successive frames. Most modern video compression techniques exploit these correlation features by reducing statistical redundancy. With these techniques, codecs manage to losslessly compress the video size by a factor of 5 to 12 or even with a factor of 20 to 200 with lossy compression [11]. There are three redundancy reduction principles that are fundamental for almost any video codec [12]:

1. Spatial redundancy reduction
2. Temporal redundancy reduction
3. Entropy encoding

These are the principles that will also be exploited in this research.

Spatial redundancy reduction

Spatial redundancy reduction focusses on the encoding and decoding of individual frames. Within one frame, there are usually large areas of pixels that are more or less the same. Transform coding and differential pulse code modulation (DPCM) are techniques that exploit this feature. DPCM uses a prediction of the value of the neighbouring pixel, based on the value of the previous pixel [13]. This prediction will be subtracted from the actual pixel value to obtain the prediction error. Due to spatial correlation, this prediction error is usually much smaller. To save bandwidth, only the prediction error will be transmitted. In the receiver, the prediction error can be added to the predicted pixel value to restore the original frame without loss of information.

Transform coding exploits spatial correlation by mapping pixels from the spatial domain to another (transform) domain. One of the most commonly used transforms is the discrete cosine transform (DCT). The DCT transforms the image into the frequency domain. Frames taken from natural scenes have the characteristic that the image energy is concentrated mainly in the low-frequency region [12]. This results in fewer transform coefficients. To further compress the image, the transform coefficients can be quantised, where the less significant coefficients will be discarded. This does not significantly affect image quality, but is, however, a lossy technique, since the original values cannot be recovered.

Temporal redundancy reduction

Temporal redundancy reduction focusses on reducing redundancy in successive frames. Since there are usually little to no changes in successive frames, it seems efficient to transmit only the difference in successive frames rather than the whole

frame. This is done by creating a prediction of the current frame based on the previous frames. Then the predicted frame is subtracted from the current frame, leaving you with the residual frame [13]. Figure 2.1 displays the current frame, the previous frame, and the inverted residual frame of the two.

(a) (b) (c)

Figure 2.1: (a) current frame, (b) previous frame, (c) inverted residual frame

To reduce the bitrate as much as possible, it is important to make an accurate prediction. When the prediction frame is accurate, there will be little data in the residual frame and after compression the frame size will be significantly reduced. When the frame is received, the decoder has to reverse the process by making the same prediction and adding the residual frame to the predicted frame [13]. This technique is called interframe coding (Figure 2.2).

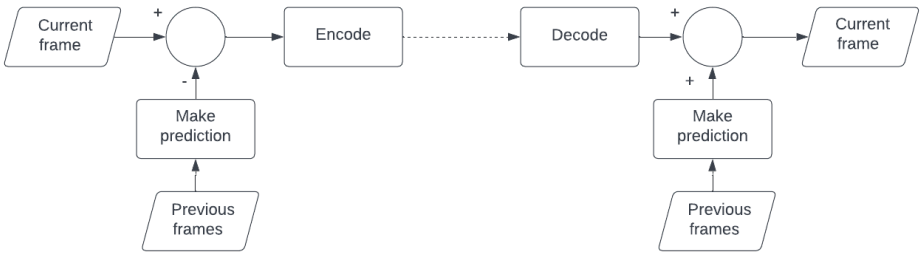


Figure 2.2: Inter-frame coding

Interframe coding is a very efficient technique for frames with little difference. However, when there is more difference between frames, caused by movement in the scene, this technique does not perform well. When frames are taken from scenes with more movement, a better prediction can be achieved by compensating for the motion first. A motion-compensated prediction usually consists of two steps:

1. *Motion estimation* is performed by comparing blocks of pixels from the current frame with neighbouring pixels from the previous frame, until a best match is found. One of the most basic techniques for motion estimation is the block matching method (BMM) [14]. Here, the frame is divided into blocks of $M \times N$ pixels, which will be displaced by a maximum of w pixels until the best match is found.

2. *Motion compensation* subtracts the region of the best match from the current frame to produce the residual frame.

To reconstruct the frame in the decoder, knowledge of the displacement is necessary on the receiving side. Therefore, displacement is usually stored as a motion vector and is also transmitted.

Entropy coding

Entropy coding is usually the final process in data encoding and is all about representing the data in the most compact form. A simple example would be to trans-

mit a sequence of symbols where many successive symbols are equal, as in Example 2.1.1. A more compact form of transmitting such a sequence would be, instead of transmitting every single symbol separately, to transmit the symbol followed by the number of successive occurrences. This is called run-length encoding.

Example 2.1.1.
Input sequence:
$[2,2,2,2,2,9,9,9,3,3,3,3,3,3,3,3,8,8,8,8,8,8,5,5,5,5,5,5,5,5,6,6,6]$
Encoded sequence:
$[(2,5)(9,3)(3,8)(8,6)(5,8)(6,3)]$

For this particular example, this seems like an effective compression method. However, when there is suddenly more variation in the sequence, run-length encoding leads to a larger file size.

A variant of run-length coding is run-level coding. This method is effective when encoding sequences with large numbers of successive zeros. In run-level encoding, every nonzero coefficient is represented together with the number of zeros preceding it. Since after transformation the image largely consists of zeros, this is an effective method in image compression.

To further reduce file size, the image coefficients are variable length coded (VLC). VLC represents the more occurring symbols in short code words, while representing the fewer occurring symbols in longer code words. The average number of bits needed to encode every code word can be calculated using Shannon's entropy [15], described in Equation (2.1):

$$H = -\sum_{i-1}^{n} p_i \log_2(p_i) \tag{2.1}$$

Here H is the entropy that can be translated as the average level of information or the average number of bits needed to describe the information. p_i is the probability of the occurrence of a specific symbol. The minimum number of bits needed to describe a single symbol is called self-information $I(x)$ and depends solely on the probability of occurrence $p(x)$. To achieve maximum compression in VLC, the number of bits used to encode a symbol must be equal to the self-information described in Equation (2.2).

$$I(x) = -\log_2(p(x)) = \log_2(1/p(x)) \tag{2.2}$$

Arithmetic coding is one of the most outstanding methods for achieving maximum compression [16]. An arithmetic encoder divides the data into sequences of symbols and encodes every sequence into a single fractional number. The length of the sequence determines the precision of the fractional number. Appendix A shows an example of arithmetic coding.

2.2 RELATED WORK

Many recent studies have already focused on reducing the bitrate of remote operating systems. The techniques proposed in the literature for optimisation of the bitrate in video encoding can be classified into two main categories. The first category mainly focusses on adapting the bitrate in such a way that the average overal bitrate is reduced. The second category focusses on preprocessing the video signal in such a way that the full performance of the encoders can be enabled. As described in Section 1.2, the lack of light, colours, and dynamic movement and the presence of noise in underwater video can all lead to decreased compression efficiency. Restoring the colour or removing the noise before encoding might result in more efficient compression.

2.2.1 Adaptive bitrate encoding

Adaptive bitrate (ABR) is a method of streaming media in which the bitrate of the video source is automatically adjusted according to the available bandwidth. ABR can be achieved using many different methods. Zvezdakov et al. [17] proposes a method of video encoding adapted to video content. Zvezdakov et al. are studying the selection of sufficient presets with new data sets to achieve maximum compression speed. They also proposed a method to predict the best coding parameters for the input video from a pre-defined list. The proposed methodology is independent of architecture and implementation and applies to various codecs and coding standards. Their experimental comparisons with real user video showed a 4–15% reduction in bitrate compared to standard presets and a similar compression quality and encoding time.

Another method is to adapt the bitrate according to the available signal. In the work of Wang et al. [18], the challenges and design of unmanned aerial vehicle (UAV) streaming in various applications, such as disaster response, surveillance, and gaming, are discussed. The proposed system design consists of two modules: an adaptive video streaming module, which adjusts video bitrates according to the drone's location and the receiver playback buffer status, and a content-aware compression module, which only transmits frames with target objects of interest. Preliminary experiments suggest that the proposed system outperforms conventional throughput-adaptive video streaming in terms of video freezing time and the amount of high-bitrate video segments received.

The work of Concolato et al. [19] presents a new method for tile-based adaptive delivery of HEVC tiled videos using MPEG-Dynamic Adaptive Streaming over HTTP (DASH). This method allows for streaming of high-resolution videos over bandwidth-constrained networks, by streaming only a tile or combination of tiles of interest, or by adaptively varying the quality of each tile. The experiment showed that the use of 9 or 25 tiles resulted in overheads of at most 3.5% and 7.3%, respectively, compared to when tiles were not used. This approach opens the way to new streaming possibilities and further research into the design of tile-based adaptive algorithms.

2.2.2 Video preprocessing

Many methods have already been suggested to improve colour in underwater video. In the work of Ancuti et al. [20], a novel approach is presented to remove haze from underwater images based on a single image captured with a conventional camera. Their approach relies on gamma correction and sharpening to deal with the hazy nature of the white-balanced image. This approach generally results in good perceptual quality, with significant enhancement of the global contrast, the colour, and the image structure details.

Methods for denoising underwater video have also been suggested. In the work of Banerjee et al. [21], a probabilistic approach to remove the so-called marine snow from underwater images is presented. The proposed method relies only on the luma channel of the image. A sliding window traverses this channel and calculates the probability of high luminance for every region of the image. If the region has a high variance in the pixel values and the probability is high, the high-luminance pixels are replaced by the median of the region. This method appears to be effective in reducing the effect of marine snow with an accuracy of more than 98%, while maintaining the true features of the image intact.

Although many methods to enhance underwater video and images have already been suggested, it seems that less research has been done on how this affects the compression rate. In the work by Rakhshanfar et al. [22] a temporal video denoization algorithm is presented that aims to reduce signal-dependent noise. Although this algorithm is not specifically developed to improve the compression rate, they mention that their algorithm has the potential to be used in denoiser-codec combinations to reduce the bitrate in noisy video. This algorithm uses temporal information to estimate motion errors and to better estimate the signal-to-noise ratios of each pixel. This is done using block motion estimation and then taking a weighted average of the pixel values. Smoothing filters are applied to further reduce blocking artefacts. Experiments have shown that the algorithm has promising results while also being significantly faster than other state-of-the-art temporal filters.

2.3 RESEARCH GAP

There are many techniques to improve the perception of underwater video by pre-processing the video signal. These techniques can help remove noise from the video, as well as improve contrast, sharpness, and colour saturation. While much research has been done on the effects of pre-processing on the perception of underwater video, less research has been done on how this affects the compression rate.

Given the importance of improving the compression rate of underwater video, investigating more about improving the compression rate by video pre-processing seems to be an interesting research opportunity. In particular, more research is needed to understand the effects of reducing the noise caused by marine snow particles on the compression rate and video quality.

Marine snow has been found to negatively affect the video quality of underwater videos. Marine snow particles can obstruct the view of the camera and can distort the image or make it difficult to view the image. In addition to the negative effect on video quality, marine snow could negatively affect the video compression rate. Video codecs often exploit the correlation between pixels, both within a single frame (spatial) and between pixels in successive frames (temporal). The presence of marine snow causes an attenuated spatial and temporal correlation, potentially resulting in lower performance of compression algorithms. As Rakhshanfar et al. [22] also pointed out, denoising the video signal by means of a temporal filter potentially improves the compression rate. Therefore, this research will focus on improving the compression rate by reducing the noise caused by the marine snow effect.

3 STREAMING APPLICATION DESIGN

In this chapter, the design of the streaming application developed for Fugro is explained. Section 3.1 describes the scope of the project. In Section 3.2, the requirements of the application and the available tools are explained, and Section 3.3 describes the software design for the application.

3.1 SCOPE

The streaming application is being developed for use on Fugro's Remote Operated Vehicles (ROV's). The ROV is a tethered underwater vehicle equipped with video cameras, sonars, pipe tracking, navigation, and positioning sensors. ROVs inspect subsea system pipelines and other subsea equipment at depths of water where divers cannot operate, while manipulators and intervention tools enable them to carry out project-specific tasks.

At this moment, the ROV's are controlled from an on-shore station, where each ROV is controlled by an operator. Six additional surround cameras will be installed on the ROV to give the operator a better view of its surroundings. This project focusses on the development of a streaming application that locally processes and encodes the video signal from the six surround cameras.

3.2 TOOLS AND REQUIREMENTS

Each of the cameras will cover one side of the ROV. The video signal is collected on a locally installed processor, where it can be pre-processed and encoded. The encoded signal will be streamed to a remote-operated surface vessel via a wired connection, from where it will be transmitted to the on-shore control station via satellite. Due to the limited bandwidth of the satellite connection (5-30 Mb/s), it is essential to encode the video signal as efficiently as possible.

3.2.1 Requirements

The video signal must be encoded in a way that meets the limitations of the satellite bandwidth. Most of the time, only a couple of cameras capture significant data, while the other cameras capture only the emptiness of the ocean. This means that it seems inefficient to always divide the bandwidth between the video signals equally. To save bandwidth, there should be a possibility to control the bitrate of each individual video signal manually. In this way, the most significant signals can be streamed with a higher bitrate than less significant signals.

3.2.2 Hardware

GMSL cameras

The ROV will be equipped with six NileCAM25 GMSL cameras [23]. Gigabit Multimedia Serial Link (GMSL) is a technology that allows for the digital transmission of video signals without compression. The GMSL standard is a high speed point-to-point connection that is used in embedded, automotive, and autonomous driving applications. It can carry video, data, and power through a small and light coaxial cable [24].

NVIDIA Jetson AGX Xavier

A Jetson AGX Xavier will be used to process and encode the video signal. The Jetson AGX Xavier is a high-end single-board System-on-Module, designed for artificial intelligence and robotics applications. The Xavier is the second most powerful board in the Jetson family and is designed for applications that require the highest performance. It features an eight-core NVIDIA Carmel Arm processor, 512 CUDA cores, and 64 Tensor cores. It also has 32GB of LPDDR4 memory, 32GB of eMMC storage, and four 4K H.264/H.265 video encoders [25]. A camera platform expansion board is connected to the Jetson to connect up to eight GMSL cameras [26].

3.2.3 GStreamer media pipeline

To process video signals locally, the GStreamer software libraries [27] were used. GStreamer is a powerful framework for handling multimedia pipelines. The framework is written in the C programming language and is designed to work with a variety of operating systems. Some of the most popular applications that use GStreamer include video editors, media players, and streaming media servers. GStreamer can also be used to create custom applications. The tool comes with a set of plug-ins that can be used to add functionality to an application. For example, there are plugins to handle different types of media data, manage pipelines, and provide graphical user interfaces. The tool is also extensible, which means that new plug-ins can be added to add new functionality.

3.3 SOFTWARE DESIGN

The application is designed to stream video content from multiple onboard cameras on the ROV in real-time. The main goal of the application is to provide a reliable and efficient way of streaming the video data to remote operators, while also allowing them to remotely control certain parameters of the encoder.

To achieve this goal, the application uses the Gstreamer library to create a media pipeline that can process and stream video frames from the cameras. The pipeline includes elements for encoding the video, multiplexing the streams from multiple cameras, and sending the resulting stream to a Real-Time Streaming Protocol (RTSP) server.

The RTSP server, also built using Gstreamer, is responsible for receiving video streams and handling RTSP commands from clients. It manages client connections and forwards RTSP commands to the appropriate components of the application.

In general, the design of the application is focused on being simple, efficient, and easy to use. The pipeline handles all video processing and streaming, while the RTSP server handles network communication and authentication. This allows the

user to log into the RTSP server, view video streams, and remotely control certain parameters of the encoder and ROV.

3.3.1 GStreamer pipeline

The GStreamer pipeline is the core component of the application, responsible for processing and streaming video frames from the onboard cameras. The pipeline is built using the Gstreamer library and consists of several elements, graphically displayed in Figure 3.1, that perform specific tasks.

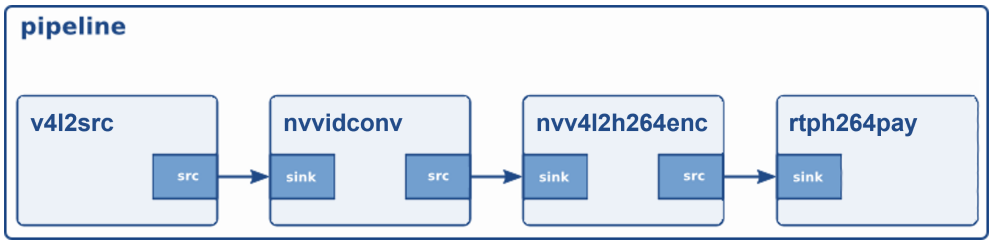


Figure 3.1: Schematic view of GStreamer pipeline used for application

The pipeline starts with the v4l2src element, which is a Gstreamer element that allows access to video capture devices such as webcams or cameras connected to the system through the Video4Linux2 (V4L2) interface. This element is responsible for capturing video frames from the onboard cameras and passing them to the next element in the pipeline. The v4l2src element can be configured with various properties, such as the resolution, frame rate and pixel format of the video device.

The next element in the pipeline is the nvvidconv element, which is a Gstreamer element that performs video format conversion using the NVIDIA hardware video codecs. This element receives the video frames from the v4l2src element, converts the video format, and passes it to the next element. This element uses the NVIDIA hardware video codecs to perform the conversion, which provides a significant performance boost compared to software-based conversion. The nvvidconv element can be configured with various properties, such as output resolution, pixel format, and colour space.

The element nvv4l2h264enc is a Gstreamer element that encodes video frames in H.264 format using the NVIDIA hardware video codecs. This element receives the video frames from the nvvidconv element, encodes them to the H.264 format, and passes them on to the next element in the pipeline. H.264 is a widely supported video codec that provides a good balance between video quality and compression. The nvv4l2h264enc element can be configured with various properties, such as output bitrate, intra-refresh mode, and profile.

Finally, the rtph264pay element is a Gstreamer element that takes H.264 encoded video frames and packages them into Real-Time Transport Protocol (RTP) packets. This element receives the video frames from the nvv4l2h264enc element, packages them into RTP packets, and sends them to the RTSP server. RTP is a standard protocol for delivering multimedia content over IP networks and is commonly used to stream video and audio. The rtph264pay element can be configured with various properties, such as maximum packet size, timestamp offset, and payload type.

The pipeline is designed to be highly modular, allowing for easy modification and customisation. Each element of the pipeline performs a specific task, and the pipeline can be extended or modified by adding or removing elements as needed.

The pipeline is optimised for efficiency and low latency, ensuring that the video streams are delivered to the users in real-time with minimal delay. The elements of the pipeline are connected so that the data flows from the source to the sink, and each element processes the data and passes them to the next element; in this case, the pipeline starts with the v4l2src element and ends with the rtph264pay element.

The GStreamer pipeline architecture is a crucial component of the application; it captures video frames from onboard cameras, converts the format, encodes them, and packages them into RTP packets, making them ready to be sent to the RTSP server. The pipeline architecture is designed to be highly modular, efficient and low latency, ensuring that video streams are delivered to users in real time with minimal delay.

3.3.2 RTSP server

The GStreamer RTSP server is a crucial component of the application, responsible for receiving video streams and handling RTSP commands from clients. The server is built using the Gstreamer RTSP server plugin, which provides a robust and flexible framework for creating RTSP servers.

The GStreamer RTSP server is based on a modular architecture, where different components handle specific tasks. The main components of the GStreamer RTSP server are:

- The RTSP server core: This is the backbone of the GStreamer RTSP server, it is responsible for managing the server's state, handling client connections, and managing the sessions. The server core is also responsible for authenticating clients, forwarding RTSP commands to the appropriate components, and receiving commands from the TCP socket to change parameters o pipeline elements.
- The Media Factory: This component is responsible for creating the media pipelines to stream video content. The media factory receives a request for a specific media and creates the pipeline for that media and returns it to the server core.
- The Media Manager: This component is responsible for managing the media pipelines and their corresponding sessions. The media manager receives commands from the server core and the TCP socket and forwards them to the appropriate pipeline.
- The RTP Manager: This component is responsible for managing RTP sessions. It receives RTP packets from the pipeline and forwards them to the appropriate client.

The GStreamer RTSP server also supports various authentication mechanisms, such as basic authentication, digest authentication, and token-based authentication. It also supports different transport protocols, such as TCP, UDP, and HTTP for video streaming, and a separate TCP socket for handling commands to change pipeline parameters.

The GStreamer RTSP server is designed to be highly scalable, with the ability to handle multiple clients and sessions simultaneously. The server can stream video to clients using various protocols, such as RTP/UDP, RTP/TCP, and HTTP. Additionally, the server can handle multiple video streams simultaneously, allowing different clients to view video streams and handle RTSP commands.

3.3.3 Implementation

For the final implementation, the GStreamer RTSP server application, described in Section 3.3.2, implements the media pipeline described in Section 3.3.1 to process and transmit video streams over a network. The RTSP server application listens for incoming RTSP requests from clients and establishes a connection with the client. Once the connection is established, the pipeline starts transmitting the media stream to the client, who can then receive and play the stream.

In a GStreamer-based RTSP server, the RTSP server takes ownership of the media pipeline by controlling its operation. The RTSP server sets up the pipeline and starts it running and is responsible for managing the flow of media data through the pipeline. This makes implementing such a RTSP server relatively straightforward when there is no need to make adjustments to the properties of the elements on the fly. Since making on-the-fly adjustments of element properties is a requirement, a workaround was necessary.

To be able to use the GStreamer RTSP server while maintaining control of the parameters of the element properties, the pipeline described in Section 3.3.1 had to be split into two, Figure 3.2.

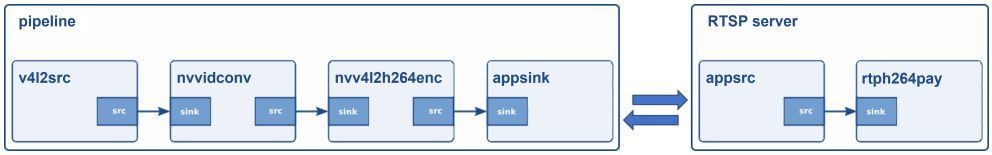


Figure 3.2: Schematic view of GStreamer pipeline after splitting in two sections.

The elements appsrc and appsink are two GStreamer-provided elements that can be used to create a connection between an application and a GStreamer pipeline. The appsrc element acts as the source pad for the RTSP server, allowing it to receive media data from the pipeline. The appsink element acts as the sink pad for the pipeline, allowing it to push media data into the server.

To establish a connection between the appsrc and appsink elements, the application creates and configures the elements, sets up the callbacks, and then links them together. The pipeline processes the data and then uses the appsink element to push the data to the server. The processed media are then available to the server via the appsrc element. From this, the server can transmit the media to the client via the rtph264pay element.

In this way, the RTSP server only takes control over the rtph264pay element, allowing it to handle data streaming to the client. The client is able to transmit new element property parameters to a TCP socket on the server. The server transmits this command to the pipeline where the adjustment will be made.

4 COMPRESSION RATE OPTIMISATION ANALYSIS

In this chapter, techniques to optimise the compression rate are explained. Section 4.1 explaines why denoising underwater video can improve the compression rate. In Section 4.2, two different types of motion compensation algorithms that were developed for this research are explained. Finally, in Section 4.3, two different temporal filters used for denoising are described.

4.1 DENOISING UNDERWATER VIDEO

When light scatters on small particles, such as plankton or sediment, it can create noise in the form of grains or speckles in the video. This noise can reduce spatial and temporal correlation in underwater video, which can negatively affect the encoding process.

Spatial correlation refers to the relationship between pixels in an image or video frame, while temporal correlation refers to the relationship between pixels in successive video frames. When an image or video has high correlation, the pixels are similar to their neighbours in terms of colour and intensity. This makes it easier to compress the video data, as there is less variation between the pixels. On the other hand, when an image or video has low correlation, the pixels are more varied, making it more difficult to compress the data.

Noise caused by light scattering on small particles can reduce both spatial and temporal correlation in underwater video. Noise can obscure fine details and create variations in colour and intensity, making it more difficult to compress the video data. This can increase the file size of the video and make it more difficult to encode the video efficiently. To increase the efficiency of high-performance video codecs in underwater video, it seems wise to reduce unwanted noise in raw video frames. Various types of filters can be used to reduce the amount of noise in the video signal. Spatial filters are applied to each individual video frame, while temporal filters are applied to a sequence of video frames.

Marine snow particles often have a larger displacement per frame than their surroundings. Temporal information from successive frames can be used to construct a noise-free image and improve spatial and temporal correlation. The goal is to estimate a noise-free frame C_t, using the combined information from the original frame F_t and its $2n$ surrounding frames $\{F_{t-n}, ..., F_t, ..., F_{t+n}\}$. Due to the movement of the ROV, there is often a small displacement of the scenery between these frames. Therefore, these frames must first be compensated for their motion, after which the pixel values can be compared with those at the same location in successive frames within the window. Only noise-free pixels are chosen to construct the estimated noise-free frame. For this research, a combination of two different filters and two different motion compensation algorithms was developed to remove noise from underwater video. The different combinations have been tested and compared for their performance.

4.2 MOTION COMPENSATION

As described in Section 4.1, two types of motion compensation algorithms have been developed and tested for their performance. One uses a global motion estimation method, and the other uses a local motion estimation method. Global motion estimation refers to the process of estimating and correcting for large-scale movements of a camera or an object in a video sequence. These movements may include translations, rotations, and zoom-in or zoom-out. Local motion compensation, on the other hand, refers to the process of estimating and correcting for small-scale movements of objects within a video frame. These movements may include movements of small parts of an object or movements of an object relative to other objects in the scene.

4.2.1 Global motion estimation

Global motion estimation refers to the process of determining the overall motion of an object or scene in an image or video sequence. It is a crucial step in many computer vision and image processing applications, such as video compression, video stabilisation, and video tracking. There are several methods for estimating global motion that can often be classified into two categories: feature-based methods and model-based methods.

Feature-based methods involve the identification and tracking of distinctive points or features in the image or video sequence. These features can be points, edges, corners, or any other distinctive visual elements. The motion of these features is then used to estimate the global motion of the scene. One popular feature-based method is the Lucas-Kanade algorithm [28][29], which uses gradient information to track features in an image sequence.

Model-based methods involve the use of a mathematical model to represent the motion of the scene. Sometimes, these methods require a priori knowledge about the motion of the scene, such as the camera's intrinsic parameters and the 3D structure of the scene. There are several types of motion models that can be used in model-based global motion compensation, including translational, rotational, and affine models. Translational models assume that the motion of objects in the image is a simple translation or shift in position. Rotational models assume that the motion is a rotation around a fixed point. Affine models allow for more complex motion, including both translation and rotation, as well as scaling and shearing.

To estimate the motion using these models, algorithms typically search for the parameters of the motion model that best fit the observed motion in the image or video. This can be done using techniques such as least squares optimisation [30] or Enhanced Correlation Coefficient (ECC) maximisation [31].

Enhanced Correlation Coefficient Maximisation

For the estimation of global motion, ECC maximisation has been chosen to estimate the transformation parameters between two frames. ECC maximisation is a method to estimate the transformation between two images that are related by a geometric transformation, such as translation, rotation, and scaling [31]. The goal of ECC is to find the transformation that maximises the correlation between the two images, which can be used to transform the target frame such that it is approximately the same as the current frame and, as such, compensated for motion.

ECC has been widely used in a variety of applications, including image registration, object tracking, and stereo matching. In image registration, ECC can be used to align two images of the same scene taken at different times or from different viewpoints. In object tracking, ECC can be used to track the motion of an object in a video sequence by aligning the object in successive frames. In stereo matching, ECC can be used to compute the depth map of a scene by aligning the left and right images of a stereo pair.

ECC is an iterative optimisation method that starts with an initial estimate of the transformation and then iteratively refines it by maximising the correlation between the two images. The transformation is parameterised using a set of parameters $\mathbf{p}$, such as the translation and rotation angles, and the optimisation process involves finding the optimal values of these parameters that maximise the enhanced correlation coefficient $\rho(\mathbf{p})$ [32] in:

$$\rho(\mathbf{p}) = \frac{\bar{\mathbf{f}}_r^t \bar{\mathbf{f}}_w(\mathbf{p})}{\| \bar{\mathbf{f}}_r \| \| \bar{\mathbf{f}}_w(\mathbf{p}) \|}, \tag{4.1}$$

where $\bar{\mathbf{f}}_r$ and $\bar{\mathbf{f}}_w$ are the reference frame and the warped frame, respectively. This method has been chosen because of one of the key advantages of ECC, its robustness to noise and outliers [31]. ECC uses a robust cost function that is insensitive to outliers, making it less prone to errors caused by noise or occlusions. This makes ECC particularly well suited for applications where images are of poor quality or have significant amounts of noise or occlusions.

Affine transformation

In the developed filter, ECC is used to estimate the affine motion transformation parameters. Affine motion refers to the movement of an object in a plane or in space, where the object is allowed to translate, rotate, scale, and shear [33]. Affine transformations preserve parallelism, but not necessarily distances between points or angles between lines. In other words, if two lines were parallel before an affine transformation, they would still be parallel after the transformation, but the distance between the lines may have changed.

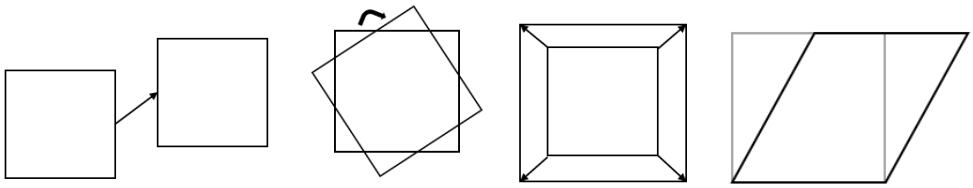


Figure 4.1: Affine transformations, with (a) translate, (b) rotate, (c) scale, (D) shear

The affine transforms rotate (4.3), scale (4.4), and shear (4.5) are all linear transforms and therefore can be represented as a matrix multiplication. The affine transform translate (4.2) is not a linear transform.

$$\text{Translate:} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}, \tag{4.2}$$

$$\text{Rotate:} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{4.3}$$

$$\text{Scale:} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} xs_x \\ ys_y \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{4.4}$$

$$\text{Shear:} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + yh_x \\ xh_y + y \end{bmatrix} = \begin{bmatrix} 1 & h_x \\ h_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{4.5}$$

where θ is the angle of rotation, s_x and s_y are the horizontal and vertical scale factors, respectively, and h_x and h_y are the horizontal and vertical shear factors, respectively. The rotation, scale, and shear transforms can be conveniently represented by a single matrix multiplication in Equation (4.6).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_0x + a_1y \\ a_2x + a_3y \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ a_2 & a_3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{or} \qquad \mathbf{x}' = A\mathbf{x}, \tag{4.6}$$

where $\{a_0, ..., a_3\}$ are the transform parameters. However, this matrix representation lacks the translation transform. As translation is not a linear transform, it is not possible to represent it as a matrix multiplication in two dimensions. This problem can be solved by representing the affine transform as a plane in three dimensions and setting the third coordinate to 1 (4.7).

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_0x + a_1y + a_2 \\ a_3x + a_4y + a_5 \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{4.7}$$

The translation in two dimensions is now represented as a shear motion in three dimensions. As such, all affine transformations in two dimensions can be described using six parameters $\{a_0, ..., a_5\}$. These are the parameters estimated by ECC. To receive new pixel coordinates for a motion-compensated frame, the pixel coordinates of the target frame can be multiplied by the affine transformation matrix in Equation (4.7).

Kalman filter

The affine parameters are constantly updated using a Kalman filter [34] on a separate thread. The Kalman filter is used to predict the state of a dynamic system from a series of incomplete and noisy measurements. It works by predicting the state of the system at the current time t based on the previous state $t-1$ and the dynamics of the system, and then updating this prediction based on new measurements. The resulting estimates are optimal in the sense that they minimise the mean squared error between the estimated and actual system states.

The Kalman filter consists of 4 steps:

1. Initialisation
2. Measurement
3. State update
4. Prediction

The initialisation step is performed only at the beginning. Here, the system is preloaded with a predefined initialisation value $\hat{x}_{0,0}$ for the affine transformation parameters and the uncertainty of the initial value $p_{0,0}$. In the measurement step, ECC maximisation is performed on the current and previous frame to measure the affine transformation z_n on time stamp n with measurement uncertainty r_n. During the state update, the current state estimate is calculated with the filtering equation (4.8):

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - \hat{x}_{n,n-1}), \tag{4.8}$$

where $\hat{x}_{n,n-1}$ is the previous state and K_n is the Kalman gain. The Kalman gain is used to weight the predicted state and the measured state to produce an updated estimate of the system's state that is more accurate than either the predicted state or the measured state alone. If the Kalman gain is high, it means that the measurement is more reliable and should receive more weight in the update. If the Kalman gain is low, it means that the predicted state is more reliable and should be given more weight in the update. The Kalman gain is calculated using the weight equation (4.9):

$$K_n = \frac{p_{n,n-1}}{p_{n,n-1} + r_n}, \tag{4.9}$$

where $p_{n,n-1}$ is the uncertainty of the previous predicted state estimate. Then, finally, in the state update step, the current estimate uncertainty $p_{n,n}$ is calculated using the corrector equation (4.10):

$$p_{n,n} = (1 - K_n)p_{n,n-1}. \tag{4.10}$$

In the last step, the next state is predicted based on the current state and the dynamic model of the system. Since the motion of the system is mostly constant, the assumption can be made that the predicted state estimate is equal to the current state estimate, and the predicted state estimate uncertainty is equal to the current state estimate uncertainty:

$$x_{n+1,n} = x_{n,n}, \tag{4.11}$$
$$p_{n+1,n} = p_{n,n}. \tag{4.12}$$

4.2.2 Local motion estimation

The second motion compensation method is a local motion compensation method. Local motion compensation takes advantage of the fact that objects in a scene tend to move relatively little over time. Accurate prediction of the motion of objects within a frame is essential for effective local motion compensation. There are several methods for local motion estimation, including full search, hierarchical search, and the three-step search algorithm. In this research, the focus lies on the three-step search algorithm as a specific method for local motion estimation.

Three-step search block matching

The Three Step Search (TSS) [35] block matching motion compensation algorithm is a technique used in video compression to predict the motion of blocks within a video frame. The algorithm works by dividing the image into blocks and then searching for the best matching block in a small region around the block in the previous frame.

The TSS algorithm operates in three steps:

1. Initial search: In this step, the algorithm searches for the best matching block in the previous frame by comparing the block in the current frame to a number of different locations in the previous frame (Figure 4.2a). The algorithm uses a predefined search window to limit the number of possible locations that are compared.

2. Refined search: In this step, the algorithm refines the search by using a smaller search window centred on the best matching block of the initial search (Figure 4.2b). This step is designed to improve the accuracy of the motion prediction by focussing on a more specific region of the previous frame.

3. Final search: In this step, the algorithm performs a final search using an even smaller search window centred on the best matching block of the refined search (Figure 4.2c). This step is designed to further improve the accuracy of the motion prediction by examining an even more specific region of the previous frame.

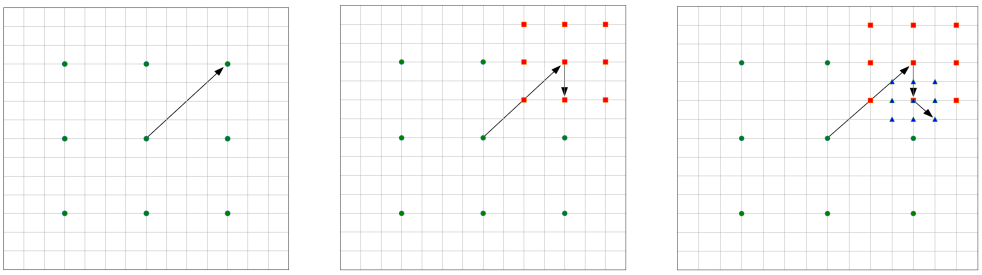


Figure 4.2: Three Step Search, with (a) initial search, (b) refined search, (c) final search.

In each step, the best matching block is selected by choosing the block with the lowest Mean Absolute Difference (MAD). MAD is a measure of the difference between two blocks of pixels. It is calculated by taking the absolute value of the difference between each pixel in one block and the corresponding pixel in the other block and then averaging these differences. For two blocks of pixels X and Y with size $L \times L$, the MAD is calculated using Equation (4.13).

$$MAD = \frac{1}{L^2} \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} |X_{ij} - Y_{ij}|, \tag{4.13}$$

MAD is a commonly used measure of difference because it is relatively simple to calculate and is not affected by the scale of the data. However, it can be sensitive to outliers, as a single large difference can significantly affect the MAD [36].

After the three steps have been completed, the TSS algorithm uses the results of the final search to generate a motion-compensated frame of the current frame. This motion-compensated frame is then used in the temporal filter to remove the noise in the current frame, reducing the amount of data required to represent the video.

In general, the TSS block matching motion compensation algorithm is a highly effective technique to improve video compression efficiency. It is widely used in a variety of video compression standards and has proven to be an important contributor to the development of modern video encoding and decoding technologies.

4.3 TEMPORAL FILTER

Temporal video filtering refers to the process of applying filters to a sequence of video frames, rather than just a single frame. These filters are designed to smooth out or remove unwanted variations or noise in the video over time, such as flickering or jitter.

There are several different techniques that can be used for temporal video filtering, including:

- Moving average filters: These filters work by calculating the average value of a group of neighbouring frames and using that value to replace the current frame [22]. This can help smooth out fluctuations in the video over time.
- Median filters: These filters work by selecting the median value of a group of neighbouring frames and using that value to replace the current frame. This can help remove outliers or sporadic noise from the video, which is often used for background subtraction [37].
- Kalman filters: These are a type of recursive filter that uses a mathematical model of the video process to predict the value of the current frame based on past frames. This can be useful to remove noise or compensate for missing data [38].
- Optical flow: This technique involves tracking the movement of pixels between frames in the video and using that information to interpolate or extrapolate the values of the current frame. This can be useful in removing jitter or other types of motion artefacts [39].

It is important to note that temporal video filtering can introduce some degradation in the quality of the video, such as loss of sharpness and resolution. As such, it is important to carefully balance the trade-off between smoothing the video and preserving its quality.

For this research, a weighted moving average filter and a median filter have been developed and compared for their performance.

4.3.1 Moving average filter

A moving average filter is a type of low-pass filter that works by calculating the average value of a group of neighbouring frames and using that value to replace the current frame. This process is repeated for each frame in the video. The size of the group of frames used for the average, also known as the window size, can be varied to control the amount of smoothing applied to the video. A larger window size will result in more smoothing, but may also introduce more delay and blur into the video.

A moving average filter can be modified to use a weighted average instead of a simple average. With a weighted average, each pixel in the window is assigned a weight, and the weights are used to influence the contribution of each pixel to the average. For example, if we want the a pixel in the current frame to have a larger influence on the average, we can assign it a higher weight. Similarly, if we want the influence of pixels in the past frames to be more pronounced, we can assign them higher weights as well.

Using a weighted average allows for more flexibility in the amount of smoothing applied to the video, as the weights can be adjusted to achieve the desired trade-off between smoothing and preserving detail. It is important to note that the weights

used in the weighted average should sum up to 1, in order to ensure that the overall brightness of the video is not affected.

Due to motion compensation, the background in all the frames should be more or less in the same position of the frame. Since the noise caused by light scattering on the marine snow particles often moves faster and in different directions than the background, the only difference between the motion-compensated frames should be the location of these particles, which makes constructing a frame without these particles more convenient.

After motion compensation, the developed weighted average filter consists of three steps:

1. Calculation of weights: Outlier pixel values should get a lower weight than other pixels.
2. Avereging: The weighted average of the pixel values is used to construct the new frame.
3. Local smoothing: Spatial low-pass filter to smooth out the last local irregularities.

Calculation of weights

There are two factors that are taken into account to calculate the weights. The first determines whether the pixel value is an outlier or not. This factor d_k is determined by taking the absolute value of the pixel value p_k subtracted from the median value M of pixels with the same spatial coordinates, Equation (4.14).

$$d_k = |M - p_k|, \tag{4.14}$$

where k denotes the k^{th} frame within the window.

To calculate the weights, the exponent of the outlier factor d_k from Equation (4.14) multiplied with an experimentally determined gain factor α is taken, Equation (4.15).

$$w_k = e^{-\alpha d_k} = e^{-\alpha |M - p_k|}. \tag{4.15}$$

Averaging

To determine the pixel values of the predicted frame, a weighted average is taken from the pixels with the same spatial coordinates within the window. The sum of the weights is multiplied with the pixel values and then divided by the sum of the weights, as described in Equation (4.16).

$$p' = \frac{\sum_{k=0}^{K-1} w_k p_k}{\sum_{k=0}^{K-1} w_k}, \tag{4.16}$$

where p' is the predicted value for the pixel.

Local smoothing

Finally, the resulting frame is smoothed locally to eliminate the last irregularities. A low-pass filter is applied only to the locations in the frame where the marine snow particles were in the original frame. To do so, the weights of the target frame are used as a mask. Since outliers receive low weights and outliers are often caused by noisy marine snow particles, low weights indicate the location of the marine snow particles. By thresholding the weights w_{ij}, a mask m_{ij} is created that indicates the location where the low-pass filter should be applied.

$$m_{ij} = \begin{cases} 0, & \text{if } w_{ij} \geq \theta \\ 1, & \text{if } w_{ij} < \theta, \end{cases} \tag{4.17}$$

where θ is an experimentally determined threshold value. To receive the filtered frame F', the predicted frame from Equation (4.16) F is low-pass filtered h^{lp} and element-wise multiplied with mask array $M = (m_{ij})$ (not to mistake with the median M in Equation (4.14)). The result of this is added to the element-wise product of the inverted mask array $1 - M$ and the predicted frame from Equation (4.16) F, Equation (4.18).

$$F' = M \circ (h^{lp} * F) + (1 - M) \circ F \tag{4.18}$$

4.3.2 Median filter

The median filter is a type of non-linear filter that is used to reduce noise in video sequences. It works by replacing the intensity value of each pixel in a frame with the median intensity value of that pixel in a sequence of previous and future frames within the window. This effectively smooths out noise that may be present in a single frame while also preserving important temporal information.

There are several variations of temporal median filters, including a simple median filter, a weighted median filter, and an adaptive median filter. The simple median filter uses an equal weight for all frames in the sequence, while the weighted median filter assigns higher weights to more recent frames. The adaptive median filter adjusts the weights of the frames based on the local noise level.

Since noise is often also present in locations similar to those of the most recent frames, assigning those frames a higher weight would not be desirable. Furthermore, it was desirable to compare the moving average filter from Section 4.3.1 with a faster and simpler filter, the simple median filter was chosen for this application. After motion compensation, this filter takes the median value of the pixels with the same spatial coordinates in successive frames within the window. This generates a new frame where all outlier values are filtered out.

5 RESULTS

In this chapter, the results of experiments on improving video compression for underwater video using a temporal video filter are presented. First, the setup used for the experiments is explained in Section 5.1. An overview of configurable parameters is given in Section 5.2. Here, the effects of these parameters are displayed, and optimal values for these parameters are given with the experimental results obtained. The most influential parameter is the window size of the temporal filters; this parameter is discussed in Section 5.3. Finally, the results of the experiments are shown in Section 5.4.

5.1 EXPERIMENTAL SETUP

All experiments for the analysis of compression rate optimisation, described in Chapter 4, were performed on a Dell Precision 7670 workstation, using a virtual machine with the following specifications:

Virtual machine settings:

- Operating system: Ubuntu 20.04.5 LTS
- Virtual cores: 12th Gen Intel® Core™ i7-12850HX x12
- Virtual memory: 16 GB
- Disk space allocated: 85 GB

Motion compensation algorithms described in Section 4.2 and temporal filters described in Section 4.3 were developed in Python 3.8.10 using the OpenCV library, version 4.2.0.

The data set used in this study consisted of 4 video sequences, with a varying number of frames and a resolution of 1920x900. The video sequences were acquired in different scenarios, capturing different types of motions and lighting conditions.

The performance of the filters was evaluated on the basis of the improvement in compression rate, where the unfiltered compressed video was taken as a baseline. Since the video data consist of different videos with varying numbers of frames, the average number of bytes per frame is often used as a metric.

In the experimental setup, FFmpeg was used to encode the video using the H.264 codec and the "ultrafast" preset. The ultrafast preset is designed for quick encoding, allowing for faster processing times at the cost of slightly lower compression efficiency.

To encode the video, the following FFmpeg command was used:

```
ffmpeg -i input.yuv -c:v libx264 -preset ultrafast output.avi
```

This command was used to encode all the videos in the data set.

5.2 CONFIGURABLE PARAMETERS

The two motion compensation algorithms described in Section 4.2 and the two temporal filters described in Section 4.3 each have their own configurable parameters, except for the median filter. Experiments were performed to individually optimise those parameters. In addition to algorithm-specific parameters, there is one key parameter that affects the performance of all algorithms, that is, the window size. The window size determines the number of frames that are used to compute the current frame values.

5.2.1 Three-step search

One of the key parameters in the TSS algorithm is the block size, which determines the size of the blocks used to divide the video frames. To simplify the calculations, it is conveniant to divide the frame into exactly an integer number of blocks. Therefore, for square blocks, the block size should be a common divisor of the height and width of a frame. Figure 5.1 shows how the block size affects the computation time and the average size per frame after compression. In the experiments, the optimal block size for the TSS algorithm was determined to be 30x30 pixels.

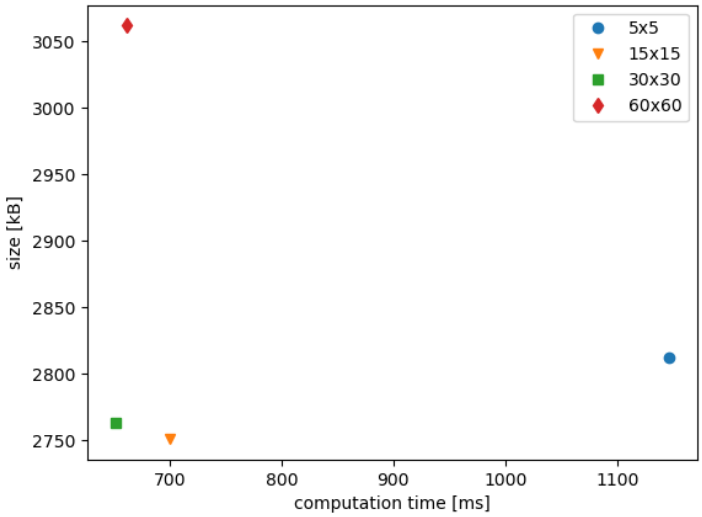


Figure 5.1: Affect of blocksize on computation time [ms] and size [kB] after compression.

The reason for choosing this block size is that it strikes a balance between computational complexity, estimation accuracy, and compression efficiency. Larger block sizes result in fewer blocks to search, which reduces computational complexity, but can also lead to a loss of accuracy in motion estimation. However, smaller block sizes increase the number of blocks to search, which improves accuracy, but also increases computational complexity, while there is no significant gain in compression efficiency.

Experiments have shown that using a block size of 30x30 pixels resulted in a good trade-off between computation time and the compression efficiency. Experiments have also shown that using larger block sizes led to a decrease in estimation accuracy and compression efficiency, while using a block size of 15x15 only slightly improved accuracy, but at a higher computational cost. When the block size is further reduced to a 5x5 block, the algorithm is able to track and compensate for all the specles individually, resulting in decreased filter performance. This explains why

the size per frame is higher for the 5x5 blocks.

Another configurable parameter for the TSS algorithm is the initial step size. The TSS algorithm uses a hierarchical search strategy, in which the search begins with a large step size and progressively decreases the step size at each step. The step size determines the distance between each search point in the motion estimation process and affects the accuracy of the algorithm. A smaller step size increases the precision of the motion estimation, as it allows for a finer search of the motion vector.

Furthermore, the choice of step size also affects the robustness of the algorithm to noise and errors in the input frames. A larger step size may be less sensitive to noise, as it allows for larger search areas, but may also miss small motions or details.

For this application, an initial step size of 4 has been chosen. After each step, the step size is halved to refine the search. Using a larger initial step size resulted in a significant drop in video quality and a lower compression rate.

5.2.2 Affine transformation

For the affine transformation motion compensation method, a Kalman filter, explained in Section 4.2.1 was used. One of the key input parameters of a Kalman filter is the initial state of the system, which is also known as the prior state. This input parameter is used to set the initial estimate of the state of the system before any measurements are taken.

The choice of the initial state of a Kalman filter is important as it affects the performance of the filter. If the initial state is accurate and the covariance matrix is appropriately set, the filter will converge quickly to the true state of the system. However, if the initial state is not accurate or the covariance matrix is not properly set, the filter may converge slowly or converge to an incorrect state.

The initial state values were determined by calculating the affine transformation parameters between frames in video sequences with similar motion and scenery. The uncertainty of the initial state was based on the variance of these measurements. The initial values used for the experiments are shown in Appendix B.

5.2.3 Moving average filter

One of the key parameters of the moving average filter is the gain factor α. This parameter controls the penalty given to outliers in the averaging process. In the experiments, the optimal value for α was determined to be 8. The value of α determines the penalty for outliers in the averaging process. A higher gain factor gives a higher penalty to outliers, whereas a lower value gives a lower penalty to outliers.

Using a gain factor α of 8 resulted in a good balance between preserving the details of the current frame and reducing noise. Using a higher gain factor increases the penalty for outliers and reduces noise, but also blurs the details of the current frame. However, using a lower gain factor not only reduces the penalty for outliers and increases the noise, but also preserves the details of the current frame. Figure 5.2 shows a graphical representation of the average size per frame after compression as a function of the gain factor α. The graph shows how alpha affects the compression rate.

Experiments have also shown that the optimal value of α depends on the specific video sequence and its characteristics. In the case of high-movement video

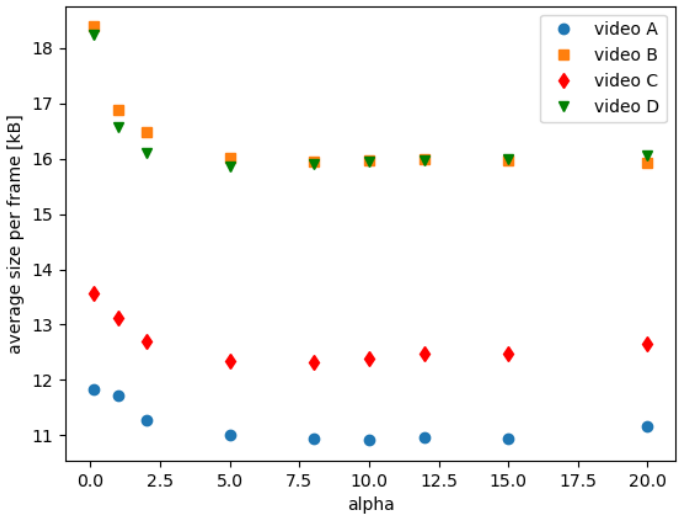


Figure 5.2: The average size per frame [kB] after compression as a function of gain factor α.

sequences, a lower value of α is recommended to preserve the details of the current frame, and in the case of low-movement sequences, a higher value of alpha is recommended to reduce the noise.

5.3 WINDOW SIZE

The window size, also known as filter size, is the most important parameter in temporal filtering. The temporal filter is used to reduce noise and improve the stability of a video sequence by averaging the pixel values of several consecutive frames. The window size determines the number of frames that are used to compute the average and affects both the effectiveness of the filtering and the computational complexity of the algorithm.

A larger window size increases the effectiveness of filtering, as it uses a greater number of frames to compute the average. This results in a smoother and more stable video, but also increases computational complexity, as more frames need to be processed. On the other hand, a smaller window size reduces computational complexity, but it may also result in less effective filtering, as fewer frames are used to compute the average.

The window size also affects the temporal resolution of the filtered video, which is the ability of the filtered video to preserve the temporal variations of the original video. A larger window size tends to smooth out these variations, resulting in a lower temporal resolution. A smaller window size preserves temporal variations, but may also result in a lower noise reduction. In practise, a trade-off between computational complexity and filtering effectiveness must be made when selecting the window size for temporal filtering.

Furthermore, it should be noted that the choice of window size depends on the specific requirements of the application and the level of noise present in the video. In cases where noise is high, a larger window size may be necessary to achieve an acceptable level of noise reduction. On the other hand, in cases where the temporal resolution is important, a smaller window size should be used to preserve the tem-

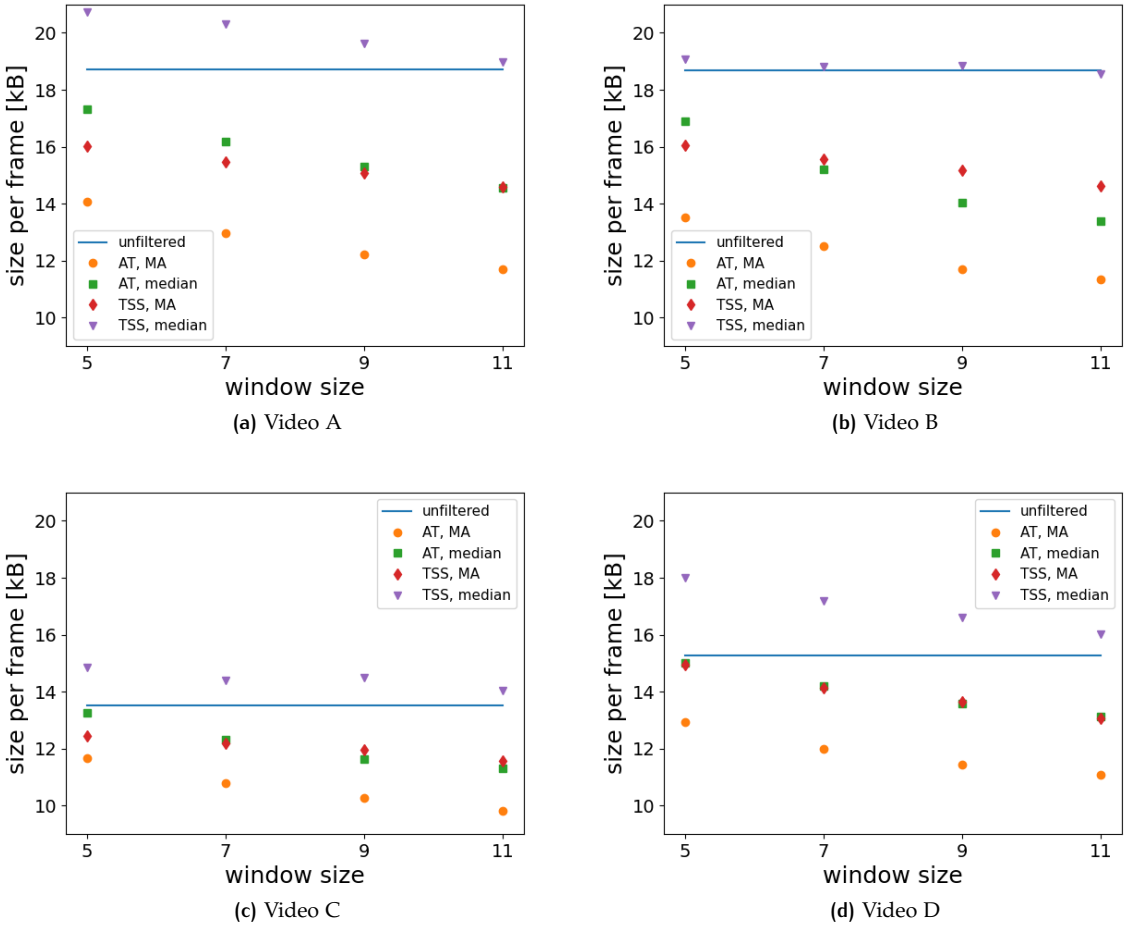
poral variations of the original video.

For this research, window sizes of 5, 7, 9, and 11 frames have been tested on several underwater video sequences to measure how they affect the compression rate and the computation time. The results of this are described below.

5.4 TEMPORAL FILTER RESULTS

In this section, the results of experiments on improving video compression for underwater video using a temporal video filter are presented. The most important parameter of the filter is the window size. The methods were evaluated using two main metrics: computation time and compression rate improvement as a function of window size.

To evaluate the results, four short video samples were used from underwater footage provided by Fugro. The samples have varying lengths, ranging between 10 and 15 seconds.



(a) Video A

(b) Video B

(c) Video C

(d) Video D

Figure 5.3: Average size per frame after filtering and compression. The blue line indicates the average size of the unfiltered compressed sample. The colors represent the different motion compensation and filtering combinations, where orange is affine transformation motion compensation with the moving average filter, green is the affine transformation motion compensation with the median filter, red is the three-step search motion compensation with the moving average filter, and purple is the three-step search motion compensation with the median filter.

Table 5.1: Improvement [%] of compression rate with respect to the unfiltered baseline.

video		A				B			
window size		5	7	9	11	5	7	9	11
motion compensation filter combination	at ma	24.8	30.8	34.8	37.4	27.7	33.1	37.4	39.3
	at med	7.5	13.6	18.2	22.2	9.7	18.6	24.9	28.4
	tss ma	14.3	17.4	19.4	22.0	14.1	16.8	18.8	21.8
	tss med	-10.6	-8.5	-0.5	-1.4	-0.2	-0.6	-0.7	0.8
video		C				D			
motion compensation filter combination	at ma	13.8	20.1	24.1	27.3	15.4	21.6	25.1	27.4
	at med	2.0	9.0	13.9	16.5	1.7	7.1	11.2	14.2
	tss ma	7.9	9.9	11.5	14.4	2.2	7.5	10.7	14.6
	tss med	-9.8	-6.4	-7.1	-3.9	-17.7	-12.4	-8.5	-4.7

5.4.1 Compression rate improvement

The results of the compression rate improvement of the four samples are presented in a graphical representation in Figure 5.3. The filter was tested with window sizes of 5, 7, 9, and 11. The results show that as the window size increases, the compression rate improvement also increases.

Figure 5.3 shows the average size per frame after filtering and compression. The blue line in the figure indicates the average size of the unfiltered compressed sample. Figure 5.4 shows the compression rate improvement as a percentage with respect to the unfiltered baseline. These values are also displayed in Table 5.1.

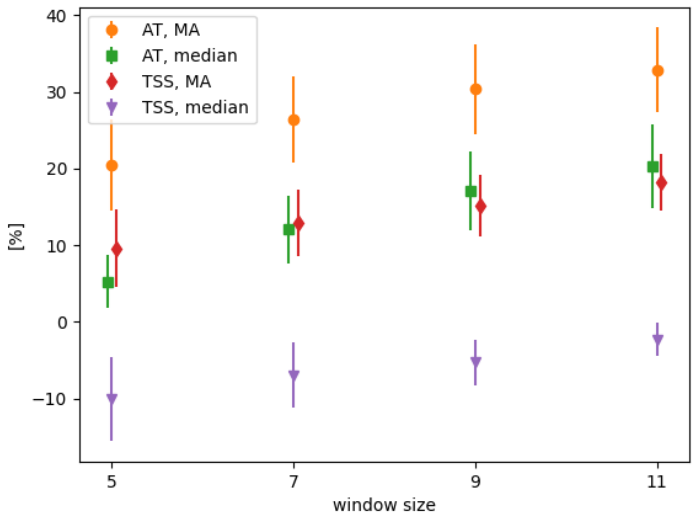


Figure 5.4: Compression rate improvement with respect to the unfiltered baseline. The percentage is an average of the videos A, B, C, and D. The colors represent the different motion compensation and filtering combinations, where orange is affine transformation motion compensation with the moving average filter, green is the affine transformation motion compensation with the median filter, red is the three-step search motion compensation with the moving average filter, and purple is the three-step search motion compensation with the median filter.

Exclusively looking at the compression rate improvement, the combination of the affine transformation motion compensation and the moving average filter seems to

have the best performance in all of the samples.

The combination of the three-step search motion compensation with the median filter has the worst performance in terms of compression rate improvement. This combination only shows counterproductive effects. One possible cause of counterproductive effects is the TSS motion compensation. The same blocks of pixels in different frames within the window might receive different motion vectors. This might reduce the temporal correlation even further.

One reason why the moving average filter is less affected by this is that the moving average filter smoothens the video both spatially and in time and therefore always improves the correlation between neighbouring pixels. This is also the reason why the moving average performs better, based on compression rate, with the affine transformation motion compensation.

5.4.2 Computation time

Figure 5.5 displays the average computation time per frame for video C. The graph displays all possible motion compensation and filter combinations, as well as the window size used. The average computation time per frame, consumed by the filter, is displayed in blue, while the average computation time per frame, consumed by the motion compensation, is displayed in orange.

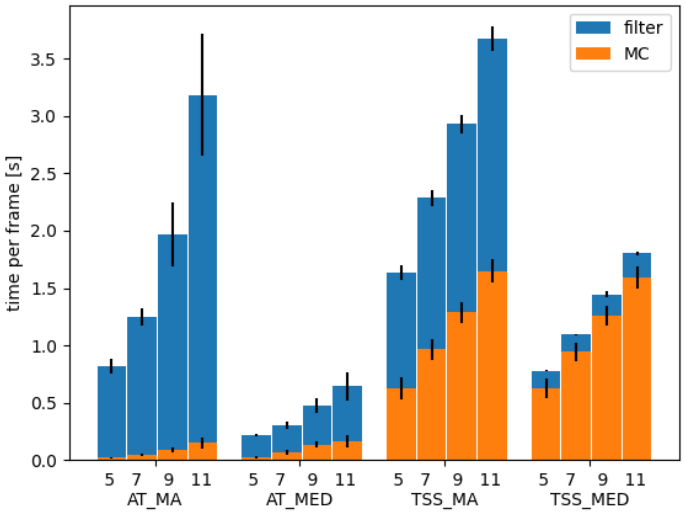


Figure 5.5: Average computation time per frame for video C. AT_MA means affine transformation motion compensation with the moving average filter, AT_MED is the affine transformation motion compensation with the median filter, TSS_WA is the three-step search motion compensation with the moving average filter, and TSS_MED is the three-step search motion compensation with the median filter. The errorbars indicate the standard deviation of the measurement data sets.

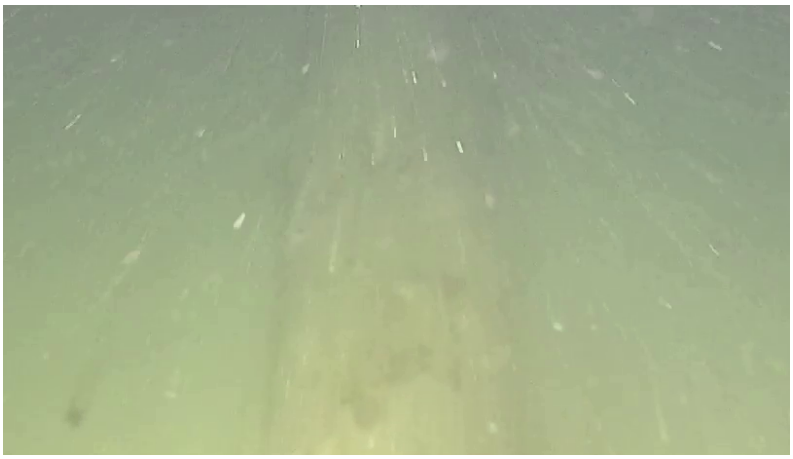
The graph shows that the affine transformation is the most efficient motion compensation method in terms of computation time. The TSS algorithm takes more time because it has to calculate the motion vector for each block in each frame separately while the transformation parameters for the affine transformation are being updated in the background. Furthermore, with the affine transformation, the whole frame can be transformed at once, while with the TSS, all blocks need to be displaced

separatly.

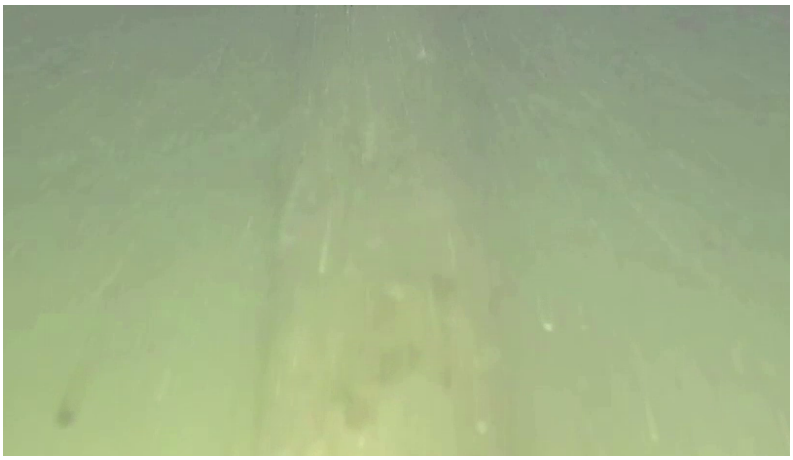
Compared to the moving average filter, the median filter is able to process video sequences much faster due to its simplicity. The moving average filter can be computationally expensive, especially for larger window sizes, as they consist of more steps. Where the moving average needs to identify the median to calculate weights, calculate the average, and apply a low-pass filter, the median filter operates by identifying the median value of a set of pixels and replacing the value of each pixel with the median value. This process is much simpler and faster than the moving average filter, as it only requires the sorting and selection of a small set of data.

5.4.3 Visual quality

A series of subjective evaluations were also conducted to assess the visual quality of compressed underwater videos. The results showed that with an affine transformation and a window size of up to and including 9 frames produced videos that were comparable to or higher quality than unfiltered compressed video. From a window size of 11 frames, the video tends to get blurry. The three-step search algorithm causes blocking artefacts, negatively affecting visual quality.



(a) Unfiltered.



(b) Filtered with affine transformation motion compensation and median temporal filter.

Figure 5.6: Zoomed underwater video snapshot.

Figure 5.6 shows a snapshot of the ROV inspecting a pipeline. This snapshot shows the difference between the unfiltered and filtered videos. In the snapshot of the filtered video, light scatterings on marine snow particles are almost completely

gone, whereas other details are still preserved.

With this experimental setup, the filters were able to achieve a noticeable reduction in noise and motion blur in the video sequences while preserving the sharpness of the frames. By creating more temporal correlation with the temporal filters, the video can be compressed up to 37% more efficiently compared to unfiltered video.

The weighted moving average filter outperforms the median filter based on compression rate, however, the median filter seems to be most promessing for streaming real-time underwater video. Because of the computational complexity, the median filter looks more promesing for real-time applications. While the median filter does not achieve the same compression rate improvements as the weighted moving average filter, it does show a significant improvement compared to the unfiltered baseline.

6 DISCUSSION

In this chapter, the results of the study will be analysed and compared to determine the strengths and weaknesses of each filter. This discussion will also provide insight into the limitations of the filters and highlight areas for improvement. Ultimately, the goal of this discussion is to provide recommendations for the use of these filters in real-world scenarios, particularly in the context of Remotely Operated Vehicles (ROVs) used by Fugro. Section 6.1 describes the challenges specifically for underwater video compression. The methods used for motion compensation are discussed in Section 6.2 and Section 6.3 discusses the temporal filters used for this research.

6.1 UNDERWATER VIDEO

This research is specifically focused on underwater videos. To reduce the amount of bandwith required to transmit underwater videos, it is wise to consider the specific challenges of underwater video compression. Underwater video compression is often less efficient than compression of other types of video due to the unique characteristics of underwater environments. The main challenges that make underwater video compression more difficult than compressing any other type of video are low visibility, high noise levels, and specific colour range.

Temporal video filtering is a technique used to improve the quality of a video by reducing noise and other unwanted artefacts by analysing multiple frames of the video over time. A temporal filter can reduce the impact of noise in a video by averaging the values of multiple pixels in multiple frames to produce a final image that is less affected by noise. Motion compensation is an important first step when using a temporal video filter because it enables the filter to align images and take advantage of interframe information.

For this research, a combination of two types of motion compensation techniques and two types of temporal filters was tested to assess their effectiveness in improving the compression rate and quality of underwater video. The motion compensation techniques that were used were three-step search block matching and affine transformation. The temporal filters that were used were a weighted moving average temporal filter and a median temporal filter.

6.2 MOTION COMPENSATION

The three-step search block matching technique is a motion estimation method that uses a block matching algorithm to find the best match between a block of pixels in the current frame and a block of pixels in the reference frame. The three-step search block matching is a local motion compensation method because it only considers a small region of the frame; therefore, this method is able to accurately estimate the motion of small objects or regions within the frame.

The affine transformation technique is a motion compensation method that uses a mathematical model to estimate the motion of an object in a video. Instead of

dividing the frame into blocks and compensating the motion for each block individually, this technique transforms the whole frame at once. Therefore, this method can be considered as a global motion compensation method.

To remove the noise caused by the marine snow particles, the affine transformation method seems to be more effective. The three-step search block matching method compensates for the motion of small blocks individually. Therefore, it also occurs that it compensates for the motion of the marine snow particles, which makes it less effective in filtering those particles than the affine transfromation method.

With an average computation time of 22, 64, 125, and 167 ms per frame for window sizes of 5, 7, 9, and 11 frames, respectively, the affine transformation method has a lower computation time per frame than the three-step search block matching method, which has an average computation time of 630, 949, 1264, and 1586 ms per frame for window sizes of 5, 7, 9, and 11 frames, respectively. As the affine transformation method uses a mathematical model to estimate the motion of the objects in the video, this model can be solved using linear algebra techniques, which are less computationally expensive than the exhaustive search performed by the three-step search method. It should be noted that the algorithms were not defeloped for efficiency in terms of computation time. This means that the values mentioned above should mainly be used as a relative comparison.

6.3 TEMPORAL FILTERS

The weighted moving average filter is a filter that uses a weighted average of pixels in multiple frames to produce a final image, while the median temporal filter is a filter that replaces each pixel in the current frame with the median value of the corresponding pixels in a set of previous and future frames.

One of the main advantages of the temporal median filter is that it is effective in removing impulse noise, which is characterised by isolated pixels with intensity values that are significantly different from their neighbours. Since the median value of a group of pixels is not likely to change dramatically unless there is a significant change in intensity within the group, the median filter is highly effective in preserving the edges in the image. This makes a median filter a good choice for applications where it is important to maintain the integrity of the original image.

With the combination of the moving average filter and the affine transformation motion compensation method, the highest compression efficiency was achieved. When filtering video sequences with window sizes of 5, 7, 9, and 11 frames, the total size after compression could be reduced by an average of 20%, 26%, 30%, and 33%, respectively, compared to the unfiltered video sequence. The median filter combined with the affine transformation motion compensation method performed worse in terms of compression rate improvement. When filtering the video sequences with the median filter with window sizes of 5, 7, 9, and 11 frames, the total size after compression could be reduced by an average of 5%, 12%, 17%, and 20%, respectively, compared to the unfiltered video sequence.

Taking into account the average computation time per frame, the median filter outperforms the moving average filter. Where the median filter has an average computation time of 204, 259, 371, and 510 ms per frame for window sizes of 5, 7, 9, and 11 frames, respectively, the moving average has an average computation time of 885, 1335, 2078, and 3232 ms per frame for window sizes of 5, 7, 9, and 11 frames, respectively. It should be noted again that the algorithms were not defeloped for efficiency in terms of computation time. This means that the values mentioned above

should mainly be used as a relative comparison.

The weighted moving average filter can show a better compression rate improvement compared to a median temporal filter. However, this advantage comes at a cost to viewing quality. The combined filter creates more correlation in the video stream by blurring, which leads to a higher correlation between neighbouring pixels, which improves the compression rate. However, this blurring process also has a negative effect on viewing quality as it reduces fine details and can create a hazy or soft appearance.

In contrast, the median temporal filter does not blur the video in the same way and, therefore, has a lower negative impact on the viewing quality. This makes the median filter a better choice when viewing quality is a priority, especially in applications where underwater video needs to be clearly visible and detailed. Furthermore, the median filter is less computationally expensive, making it better suited for real-time applications.

In conclusion, when considering compression rate improvement, computation time, and viewing quality in underwater video, there is a trade-off between the three. The weighted moving average temporal filter combined with a spatial low-pass filter can provide a higher compression rate improvement but at the cost of reduced viewing quality. On the other hand, the median temporal filter may provide a better viewing quality and lower latency, but with a lower compression rate improvement. The choice between the two filters will depend on the specific requirements and priorities of the application.

7 CONCLUSION

In conclusion, this thesis has explored techniques to improve the compression rate for underwater video captured by Fugro's Remote Operated Vehicles (ROV). The research focusses on reducing the amount of bandwidth required to transmit video data wirelessly over a satellite connection. Through the research, the following research questions were given an answer:

1. What technique can be used to reduce the amount of bandwidth required to transmit underwater video data wirelessly over a satellite connection?

2. How much bandwidth can be saved using this technique?

3. How does this technique affect other constraints, such as computation time and video quality?

7.1 METHOD

A number of methods to reduce the amount of bandwidth required to transmit video data have been suggested and presented in Chapter 2. The technique on which this research is focused is to reduce the amount of bandwidth required to transmit underwater video by improving the compression rate of underwater video. The challenges of underwater video compression can be overcome by preprocessing the video before encoding. By enhancing the colours and visibilty and by reducing the noise of underwater video, the video can often be compressed more efficiently.

7.1.1 Noise reduction in underwater video

To reduce the amount of noise present in underwater video, the characteristics of noise must be taken into account. Noise in underwater video is often caused by so-called marine snow particles. These particles are types of organic matter that sink from the surface waters of the ocean to the deep sea. The light reflecting from those particles into the camera sensor creates high-intensity specles on the frame. These light scatterings decrease the correlation between neighbouring pixels, as a result of which the encoder needs more bits to represent these data.

These specific types of noise-inducing particles have the characteristic that they often displace with a different velocity than the rest of the scenery in the video. Therefore, a temporal video filter is an efficient method of removing those particles from video frames.

For this research, a combination of two types of motion compensation techniques and two types of temporal filters was tested to assess their effectiveness in improving the compression rate and quality of underwater video. The motion compensation techniques that were used were three-step search block matching and affine transformation. The temporal filters that were used were a weighted moving average temporal filter and a median temporal filter.

7.2 COMPRESSION RATE IMPROVEMENT

With the combination of the moving average filter and the affine transformation motion compensation method, the highest compression efficiency was achieved. When filtering video sequences with window sizes of 5, 7, 9, and 11 frames, the total size after compression could be reduced by an average of 20%, 26%, 30%, and 33%, respectively, compared to the unfiltered video sequence. The median filter combined with the affine transformation motion compensation method performed worse in terms of compression rate improvement. When filtering the video sequences with the median filter with window sizes of 5, 7, 9, and 11 frames, the total size after compression could be reduced by an average of 5%, 12%, 17%, and 20%, respectively, compared to the unfiltered video sequence.

7.3 COMPUTATION TIME AND VIEWING QUALITY

Taking into account the average computation time per frame, the median filter outperforms the moving average filter. Where the median filter has an average computation time of 204, 259, 371, and 510 ms per frame for window sizes of 5, 7, 9, and 11 frames, respectively, the moving average has an average computation time of 885, 1335, 2078, and 3232 ms per frame for window sizes of 5, 7, 9, and 11 frames, respectively. It should be noted that the algorithms were not defeloped for efficiency in terms of computation time. This means that the values mentioned above should mainly be used as a relative comparison.

The median filter is considered a more efficient and practical choice for real-time applications where processing speed is a concern and viewing quality is a priority. Unlike a weighted moving average filter, the median filter does not require any complex computations for determining the weights, is less sensitive to outliers and, therefore, does not require low-pass spatial filtering. The median filter does not negatively impact viewing quality by blurring the video, which can occur with the combined filter. The median filter is often a better choice when real-time processing speed and viewing quality are important considerations.

Temporal filters are effective in improving the compression rate and quality of a video by reducing noise and other unwanted artefacts. However, it should be noted that improvements are still necessary for both filters to meet the demanding requirements of real-time applications. The median filter may still be computationally intensive for large video streams, and the combined filter may still blur the video to a degree that is unacceptable for some applications. As such, more research is needed to improve the efficiency and performance of these filters in real-world scenarios. The goal should be to find a balance between compression rate improvement and viewing quality, while also ensuring real-time processing speed and efficiency.

7.4 RECOMMENDATIONS

In this thesis, techniques have been investigated to improve the compression rate of underwater video, with a focus on temporal video filtering methods. The results have shown that a combination of motion compensation techniques, such as affine transformation, along with temporal filters, such as weighted moving average and median filters, can effectively improve the quality of underwater video while reduc-

ing the amount of bandwidth required for transmission.

However, it should be noted that these filters may not be well suited in their current form for use with the ROVs used by Fugro. The demanding requirements of ROVs, such as real-time processing speed, stability, and efficiency in varying underwater environments, may require additional improvements to these filters.

Therefore, it is recommended that further research be carried out to tailor these filters specifically to the needs of the ROVs used by Fugro. This could involve exploring alternative temporal and spatial filtering techniques, incorporating hardware acceleration, and improving the efficiency and stability of the filters in challenging underwater environments. The goal should be to find a filter that provides a good balance between compression rate improvement and viewing quality, while also ensuring real-time processing speed and efficiency for the specific requirements of Fugro's ROVs.

Based on the findings, future recommended research in this field is focused on the following areas:

- Development of more advanced motion estimation methods: Although the affine transformation method has proven to be effective, there is still room for improvement in terms of motion estimation accuracy and computational efficiency.
- Exploration of other types of temporal filters: Although weighted moving average and median filters have been shown to be effective, there may be other types of temporal filters that can achieve even better results in terms of noise reduction and compression rate improvement.
- Reducing the computation time: Future research should focus on reducing the computation time of the proposed methods by exploring techniques such as parallel processing, approximate methods, and other optimisation techniques. Further improvements can me made by programming the filters specifically for the Jetson AGX Xavier in a more efficient programming language.

BIBLIOGRAPHY

[1] G. C. Fernandez, S. M. Gutierrez, E. S. Ruiz, F. M. Perez, and M. C. Gil, "Robotics, the new industrial revolution," *IEEE Technology and Society Magazine*, vol. 31, no. 2, pp. 51–58, 2012.

[2] C. Tang, U. F. von Lukas, M. Vahl, S. Wang, Y. Wang, and M. Tan, "Efficient underwater image and video enhancement based on Retinex," *Signal, Image and Video Processing*, vol. 13, pp. 1011–1018, 7 2019.

[3] S. Raveendran, M. D. Patil, and G. K. Birajdar, "Underwater image enhancement: a comprehensive review, recent trends, challenges and applications," *Artificial Intelligence Review*, vol. 54, pp. 5413–5467, 10 2021.

[4] S. Serikawa and H. Lu, "Underwater image dehazing using joint trilateral filter," *Computers and Electrical Engineering*, vol. 40, no. 1, pp. 41–50, 2014.

[5] A. L. A. Dredge and M. W. Silver, "Characteristics, Dynamics and Significance of Marine Snow," tech. rep., 1988.

[6] C. Chen, J. Han, and Y. Xu, "A Non-local Mean Temporal Filter for Video Compression; A Non-local Mean Temporal Filter for Video Compression," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020.

[7] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, 7 2003.

[8] 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.

[9] D. Mukherjee, J. Bankoski, A. Grange, J. Han, J. Koleszar, P. Wilkins, Y. Xu, and R. Bultje, "The latest open-source video codec VP9 - An overview and preliminary results," in *2013 Picture Coding Symposium, PCS 2013 - Proceedings*, pp. 390–393, IEEE Computer Society, 2013.

[10] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi, C. H. Chiang, Y. Wang, P. Wilkins, J. Bankoski, L. Trudeau, N. Egge, J. M. Valin, T. Davies, S. Midtskogen, A. Norkin, and P. De Rivaz, "An Overview of Core Coding Tools in the AV1 Video Codec," in *2018 Picture Coding Symposium, PCS 2018 - Proceedings*, pp. 41–45, Institute of Electrical and Electronics Engineers Inc., 9 2018.

[11] D. Vatolin, I. Seleznev, and M. Smirnov, "Lossless Video Codecs Comparison '2007," tech. rep., Moscow State University, Moscow, 2007.

[12] M. Ghanbari, *Standard Codecs: Image Compression to Advanced Video Coding*. Institution Electrical Engineers, 2003.

[13] I. E. Richardson, *Video Codec Design: Developing Image and Video Compression Systems*. USA: John Wiley & Sons, Inc., 2002.

[14] T. Ishiguro and K. Iinuma, "Television bandwidth compression transmission by motion-compensated interframe coding," *IEEE Communications Magazine*, vol. 20, no. 6, pp. 24–30, 1982.

[15] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[16] A. Said, "Introduction to Arithmetic Coding-Theory and Practice," tech. rep., HP, Palo Alto, 4 2004.

[17] S. Zvezdakov, D. Kondranin, and D. Vatolin, "Machine-Learning-Based Method for Content-Adaptive Video Encoding," *2021 Picture Coding Symposium, PCS 2021 - Proceedings*, pp. 12–16, 2021.

[18] X. Wang, A. Chowdhery, and M. Chiang, "SkyEyes: Adaptive video streaming from UAVs," *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, pp. 2–6, 2016.

[19] C. Concolato, J. Le Feuvre, F. Denoual, F. Mazé, E. Nassor, N. Ouedraogo, and J. Taquet, "Adaptive Streaming of HEVC Tiled Videos Using MPEG-DASH," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1981–1992, 2018.

[20] C. O. Ancuti, C. Ancuti, C. De Vleeschouwer, and P. Bekaert, "Color Balance and Fusion for Underwater Image Enhancement," *IEEE Transactions on Image Processing*, vol. 27, pp. 379–393, 1 2018.

[21] S. Banerjee, G. Sanyal, S. Ghosh, R. Ray, and S. N. Shome, "Elimination of marine snow effect from underwater image-An adaptive probabilistic approach," in *2014 IEEE Students' Conference on Electrical, Electronics and Computer Science, SCEECS 2014*, IEEE Computer Society, 2014.

[22] M. Rakhshanfar and A. Amer, "Motion blur resistant method for temporal video denoising," in *2014 IEEE International Conference on Image Processing, ICIP 2014*, pp. 2694–2698, Institute of Electrical and Electronics Engineers Inc., 1 2014.

[23] e-con systems, *NileCAM25 - Full HD GMSL2 Global Shutter color camera with 15m cable support*, 2021.

[24] M. Li, H. Zhao, and L. Shi, "Design of high-speed video data transmission circuit based on GMSL technology," in *Journal of Physics: Conference Series*, vol. 2026, IOP Publishing Ltd, 10 2021.

[25] NVIDIA, *NVIDIA Jetson AGX Xavier Series System-on-Module*, 2022.

[26] Connect Tech, *Connect Tech GMSL CAMERA PLATFORM*, 2021.

[27] GStreamer Team, "GStreamer," 4 2019.

[28] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, vol. 2, (Vancouver), pp. 674–679, 1981.

[29] B. Lucas, *Generalized Image Matching by the Method of Differences*. PhD thesis, Carnegie-Mellon University, Pittsburgh, 7 1984.

[30] G. B. Rath and A. Makur, "Iterative least squares and compression based estimations for a four-parameter linear global motion model and global motion compensation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 7, pp. 1075–1099, 1999.

[31] G. D. Evangelidis and E. Z. Psarakis, "Parametric Image Alignment using Enhanced Correlation Coefficient Maximization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 1858–1865, 2008.

[32] E. Z. Psarakis and G. D. Evangelidis, "An enhanced correlation-based method for stereo correspondence with sub-pixel accuracy," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. I, pp. 907–912, 2005.

[33] D. House and J. C. Keyser, *Foundations of physically based modeling and animation*. AK Peters/CRC Press, 2016.

[34] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems 1," *Journal of Basic Engineering*, vol. 82, pp. 35–45, 3 1960.

[35] T. Koga, "Motion compensated interframe coding for video-conferencing," *Proc. Nat. Telecommun. Conf.*, pp. G5.3.1–G5.3.5, 1981.

[36] S. Yitzhaki, "Gini's Mean difference: a superior measure of variability for non-normal distributions," *METRON-International Journal of Statistics*, vol. LXI, no. 2, pp. 285–316, 2003.

[37] M.-H. Hung, J.-S. Pan, and C.-H. Hsieh, "Speed Up Temporal Median Filter for Background Subtraction," in *Proceedings - 2010 1st International Conference on Pervasive2010 First International Conference on Pervasive Computing, Signal Processing and Applications*, pp. 297–300, 2010.

[38] J. Kim and J. W. Woods, "Spatio-temporal adaptive 3-D Kalman filter for video," *IEEE Transactions on Image Processing*, vol. 6, no. 3, pp. 414–424, 1997.

[39] H. C. Chang, S. H. Lai, and K. R. Lu, "A robust and efficient video stabilization algorithm," in *2004 IEEE International Conference on Multimedia and Expo (ICME)*, vol. 1, pp. 29–32, 2004.

A ARITHMETIC CODING EXAMPLE

In arithmetic coding, every symbol is given a subrange within 0 - 1 based on its probability of occurrence. Table A.1 shows the symbols 'A', 'B', 'C', 'D', 'E' and their occurrences in some application. Since symbol 'A' has a probability of occurrence of 0.1, it has been given the subrange 0 - 0.1. The next symbol in the list 'B' has a probability of occurrence of 0.2, so 'B' will be given the subrange 0.1 - 0.3, and so on.

Table A.1: Symbol probabilities and their range

Symbol	Probability	Subrange
A	0.1	0-0.1
B	0.2	0.1-0.3
C	0.4	0.3-0.7
D	0.2	0.7-0.9
E	0.1	0.9-1

Table A.2: Arithmetic encoding procedure

Step	Range	Symbol	Subrange
1. Set initial range	0 - 1		
2. Find corresponding subrange		C	0.3 - 0.7
3. Set range to this subrange	0.3 - 0.7		
4. Find subrange for next symbol		B	0.1 - 0.3
5. Set range to this subrange within previous range	0.34 - 0.42		
6. Find subrange for next symbol		C	0.3 - 0.7
7. Set range to this subrange within previous range	0.364 - 0.396		
8. Find subrange for next symbol		E	0.9 - 1
9. Set range to this subrange within previous range	0.3928 - 0.396		

Table A.3: Arithmetic decoding procedure

Step	Range	Subrange	Decoded symbol
1. Set initial range	0 - 1		
2. Find subrange in which received fraction falls, this is the first symbol		0.3 - 0.7	C
3. Set range to this subrange	0.3 - 0.7		
4. Find subrange from new range in which received fraction falls, this is the second symbol		0.34 - 0.42	B
5. Set range to this subrange within previous range	0.34 - 0.42		
6. Find subrange from new range in which received fraction falls, this is the third symbol		0.364 - 0.369	C
7. Set range to this subrange within previous range	0.364 - 0.396		
8. Find subrange from new range in which received fraction falls, this is the fourth symbol		0.3928 - 0.396	E

B KALMAN FILTER INITIAL STATE VALUES

The initial state value depends on the postion of the frame F_n with respect to the reference frame F_0. Therefore, each frame F_n needs its own set of initial state values. The initial state values used for this experiment are displayed in Table B.1.

Table B.1: Kalman filter initial state values

	a_0	a_1	a_2	a_3	a_4	a_5
F_{-6}	0.989695	0.000420	9.980498	-0.000454	0.964372	3.369549
F_{-5}	0.989426	0.000329	10.124234	-0.000338	0.962081	3.692857
F_{-4}	0.991333	0.001398	8.308937	-0.000820	0.970328	1.788813
F_{-3}	0.994273	0.001146	5.421419	-0.000549	0.976860	2.080395
F_{-2}	0.997026	0.000843	2.763927	-0.000268	0.984109	1.923968
F_{-1}	0.998934	0.000408	0.967567	-0.000081	0.992381	1.123456
F_1	1.001261	-0.000423	-1.156159	0.000064	1.007683	-1.093070
F_2	1.003384	-0.000881	-3.168234	0.000246	1.016157	-1.852247
F_3	1.006399	-0.001224	-6.095303	0.000502	1.023783	-1.946351
F_4	1.009654	-0.001508	-9.321694	0.000742	1.030651	-1.462865
F_5	1.011884	-0.000367	-11.372150	0.000463	1.038223	-3.564496
F_6	1.015156	-0.000227	-14.624343	0.000619	1.045639	-3.712919