Bachelor's thesis

Information and Communications Technology

2024

Peter Ståhlström

Remote Controlled Video Processing and Serving for the eM/S Salama Unmanned Surface Vessel and its Remote Operations Center


TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

Bachelor’s | Abstract

Turku University of Applied Sciences

Degree Programme
Information and Communications Technology

2024 | 27 pages

Ståhlström Peter

Remote Controlled Video Processing and Serving for the eM/S Salama Autonomous Surface Vessel and its Remote Operations Center

eM/S Salama is an autonomous sea vessel testing platform developed by Turku University of Applied Sciences. The vessel is equipped with a variety of sensors for onboard autonomous decision making and feeding data to a remote operations center (ROC) where human operators monitor the vessels operations.

The goal of this work was to develop a system for reprocessing video streams from camera sensors mounted on the vessel, for transmission to the ROC over a limited bandwidth data link and give the remote operators the ability to view the produced video stream and control which input is processed and the output quality remotely.

The procedure for video processing was that hardware-accelerated video transcoding and modification was performed using the open-source Gstreamer multimedia framework and the produced video stream was made accessible through a standards-compliant real-time streaming protocol (RTSP) interface.

Remote control of video encoding properties and video selection were performed via the Robot Operating System 2 network already implemented for other systems on the vessel.

The result of this was a system that collects video streams from networked cameras on the vessel and allows a remote operator to use the remote control capabilities to adapt the video stream for different data link speeds.

Keywords:

USV, remote operations center, ROS2, video stream processing, Gstreamer

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Degree Programme in Information and Communications Technology

2024 | 27 sivua

Ståhlström Peter

Kauko-ohjattu videoprossessointi ja -palvelu eM/S Salama autonomiselle veneelle ja sen etävalvontakeskukselle

eM/S Salama on Turun ammatikorkeakoulun kehittämä itseohjautuvan merialuksen testialusta. Aluksessa on autonomisen päätöksenteon mahdollistamiseksi kokoelma sensoreita, jotka myös lähettävät dataa etäohjauskeskukseen, jossa henkilöstö valvoo aluksen toimintaa ja lähettää komentoja alukselle.

Työn tavoitteena oli kehittää järjestelmä, joka kerää eri kuvavirtoja alukselle asennetuista kameroista ja tuottaa kompressoidun kuvavirran joka voidaan lähettää aluksen etäohjauskeskukseen rajoitetun verkkoyhteyden kautta.

Kuvavirtojen prosessointi tehtiin avoimen lähdekoodin Gstreamer multimedia alustalla, ja kuvaprosessointiin käytettiin näytönohjaimien videokiihdytystoiminnallisuutta videon muuntamiseen ja transkoodaukseen.

Tuotettu kuvavirta on saatavilla tyypillisen RTSP-protokollan kautta ja videoprosessoinnin muuttaminen ja kuvanvalinnan etäohjaus tehdään ROS2-parametripalveluiden avulla.

Työssä kehitettiin yllä mainitun toiminnallisuuden omaava ohjelmisto, jossa käytetään etäohjaus toiminnallisuutta ulos tulevan videovirran sovittaminen eri datalinkkinopeuksille.

Asiasanat:

USV, etävalvontakeskus, ROS2, videoprossessointi, Gstreamer

Contents

Figures

List of abbreviations and symbols

4K	Video resolution with pixel count on at least one axis being approximately 4000 pixels
8K	Video resolution with pixel count on at least one axis being approximately 8000 pixels
LGPL	Lesser GNU Public License
NVENC	Nvidia Video Encoder
ROC	Remote Operations Center
ROS2	Robot Operating System 2
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
USV	Unmanned Surface Vessel

1 Introduction

The world is experiencing a growth in the development of fully or partially autonomous vehicles. Vehicles use a variety of different sensors and machine learning to operate autonomously but during the development phase or in some cases even in a final product, human supervision over the autonomous vessel is desired. The autonomous nature of these new platforms allows this supervision to be located at a centralized location that receives key information from remote platforms and can issue commands to the autonomous vessel.

Turku University of Applied Sciences has been working on an autonomous sea-faring vessel, eM/S Salama. This unmanned surface vessel (USV) is remotely monitored and controlled from a Remote Operations Center (ROC) that monitors various data streams generated on the vessel as shown in Figure 1.

Figure 1. Remote operations center (Kalliovaara, et al., 2024).

One of the data streams the ROC monitors is video streams from a set of panoramic surveillance cameras installed on the vessel. Displayed on an array of monitors as shown in Figure 1 these video streams are intended to provide a

remote operator with a broad view of the vessel's current operating environment enhancing their situational awareness.

The video stream generated by these cameras has extremely high resolution and changing the bitrate of the video stream from the cameras necessitates connecting to the camera's web interface and changing the parameter, followed by a reboot of the camera. The connection between the vessel and the ROC is over a consumer-class mobile broadband or satellite link which can experience degradation in transfer capacity, leading to the link being saturated when transmitting high resolution video streams. This could impact the transmission of other important data over the link. To avoid this, a method to quickly change the data-rate used by the video stream sent over the network is required to ensure reliable operation and save network capacity for other traffic.

As the vessel and ROC had no system to transmit these video streams between them in real-time, the objectives of this thesis work were to develop a software package that would collect and process video streams from sources on the vessel, process them into a format and resolution acceptable for transmission to the ROC, and provide access to dynamic configuration of video bitrate, resolution, video selection and in the future other parameters of the video.

This thesis is structured as follows: Chapter 2 introduces the USV eM/S Salama and its ROC as well as the requirements of the produced software. Chapter 3 describes the process of using the Gstreamer media processing framework to perform the task of collecting and processing the video streams from the USV. Chapter 4 goes on to describe how the final processed video stream is made accessible to the ROC via the widely used RTSP protocol. Chapter 5 explains how the Robot Operating System 2 is used to control the processing of the video and how to make the ROS2 framework interoperate with Gstreamer.

2 eM/S Salama and ROC background

eM/S Salama (Figure 2) is an electrically propelled catamaran equipped with a suite of sensors including RGB cameras, thermal cameras, and LIDAR sensors, designed to facilitate autonomous operation using computer vision and sensor fusion technologies. The vessel is designed to be a platform for developing technologies for USVs in the maritime industry.

Figure 2. eM/S Salama unmanned surface vessel (Kalliovaara, et al., 2024).

The vessel is monitored and remotely operated from a Remote Operations Centre where an operator can monitor data feeds from the vessel on an array of displays and issue commands to the vessel. The relationship is pictured in Figure 3.

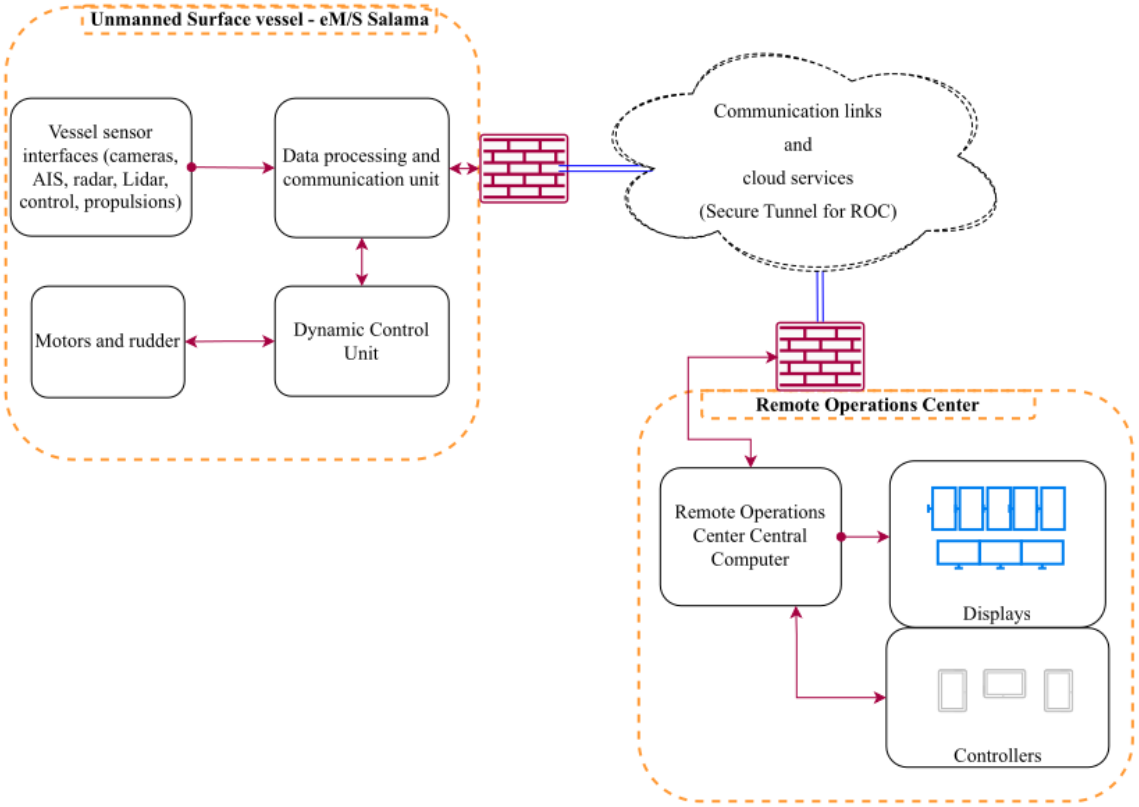

Figure 3. eM/S Salama and ROC network diagram (Kalliovaara, et al., 2024).

The main form of monitoring is intended to be a real-time view from a panoramic surveillance camera for the operator to have an awareness of the vessel's surroundings. This video feed needs to be reasonably low latency using the mobile broadband or satellite data link connecting the USV and ROC.

Video feeds from the RGB cameras specifically can have very high data-rates and can saturate the data link used to connect the vessel and ROC unless very low framerates or image quality is used. Furthermore, the model of camera used for the panoramic views sent to the ROC requires a reboot to apply changes to parameters such as bitrate, resolution, or framerate, during which the camera is inoperable.

By using the high-speed video encoding features present in modern graphics processing units (GPUs), even high-resolution, high-framerate video can be processed in real time to an adequately low bitrate to be efficiently transmitted

over the network while maintaining acceptable image quality for remote observation.

The video processing system developed in this thesis enables cameras to operate at consistently high video quality while utilizing a more flexible platform based on the widely used, open-source multimedia processing framework Gstreamer. This system reprocesses the video for transmission to the ROC and leverages the control network already implemented on the USV, the Robot Operating System 2 (ROS2). It allows the ROC to select which video to stream and modify the video encoding parameters to adapt to changes in network conditions without interruptions.

3 Video Processing

The structure of the video processing application as illustrated in Figure 4 is a typical object-oriented hierarchy where a top level object has another object as a component that it passes messages to achieve the desired functionality.

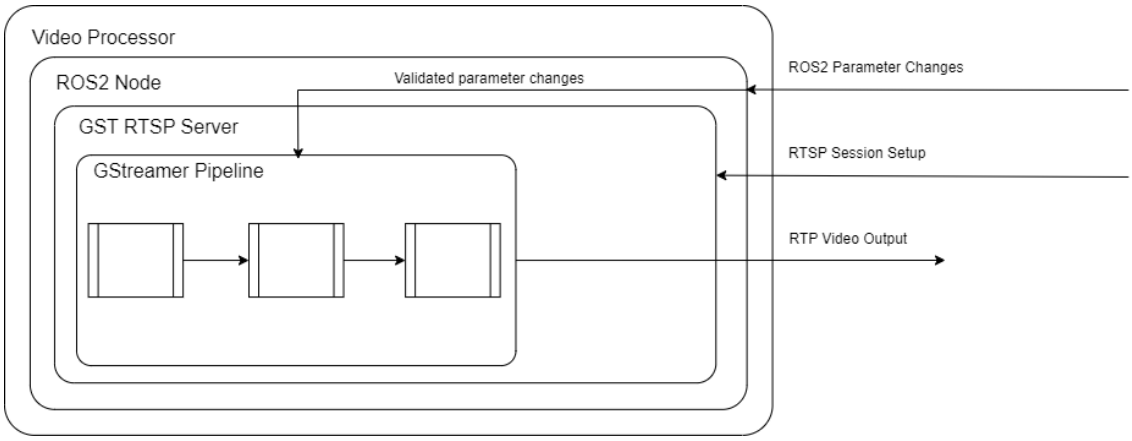

Figure 4. Overview of the program architecture.

Capture and processing of the video from the panoramic cameras is achieved with the open-source Gstreamer multimedia processing framework (freedesktop.org, n.d.). Gstreamer is based on the GObject and GLib libraries developed by the GNOME desktop environment project and is included in most Linux distributions that offer multimedia functionality. Through the GObject library's introspection features Gstreamer provides bindings for its use in a variety of programing languages such as Python or Rust. Gstreamer was selected over competing multimedia libraries such as FFMPEG because of Gstreamer's easy extensibility that allows for a plugin module to be built independently and integrated into a system's Gstreamer installation without needing to rebuild any pre-existing part. This also allows vendors to deliver closed-source modules if they choose as allowed by Gstreamer's LGPL license.

Gstreamer works on the concept of a Pipeline constructed of several Elements that provide and consume data in a linear graph-like structure. Source elements produce data for the pipeline, sink elements consume data, and other elements

manipulate data in various ways such as splitting or combining streams of data, multiplexing or demultiplexing audio and video, and applying various transforms to video data such as cropping, scaling, color correction, etc.

Elements can be accessed from the pipeline by extracting a reference to them from the pipeline with various methods provided by the Gstreamer library. The simplest method is to assign a name to elements that you wish to access during the pipeline construction and then use the `get_by_name()` method of the pipeline object to get the desired object. With a reference to a pipeline element the parameters of the element can be changed.

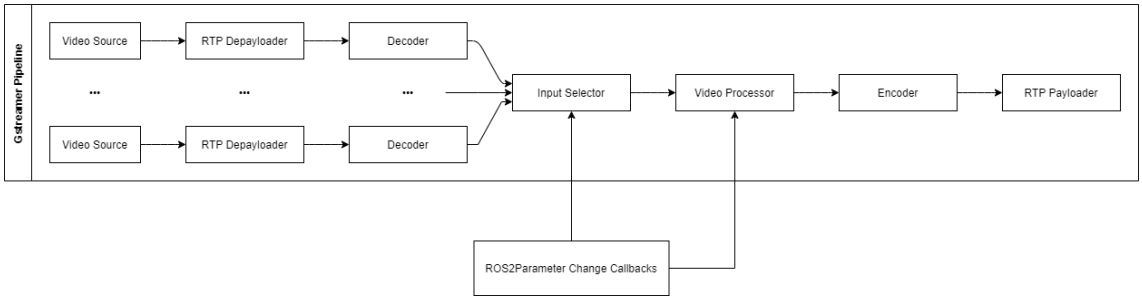

Figure 5. Gstreamer pipeline with key elements.

Figure 5 shows a block level layout of a possible pipeline for collecting and processing video streams from multiple sources and selecting one for retransmission, as well as points in the pipeline where dynamic configuration data can be used to modify the pipeline behavior.

3.1 RTSPSource, depayloader, and inputselector element

The installed surveillance cameras provide access to their video streams via standard RTSP protocol (Schulzrinne, et al., 2016). Gstreamer includes modules to request and receive RTSP streams using the ‘rtspsrc’ pipeline element. This element is configurable to adjust the size of its internal RTP jitterbuffer as well as the buffering strategy it uses. These parameters are tuned to fit both the incoming video stream, as well as the receiver’s processing

capabilities to control the smoothness and amount of latency it causes in the pipeline.

The RTSPsrc element produces a stream of RTP packets that need to be unpacked to get the bitstream for the transmitted video. This is done by passing the stream to an RTPdepayloader element, which is specific for the video codec being used. Gstreamer supports all common codecs specified in the RTP standard and provides depayloader and payloader elements for all. Due to the resolution of the video produced by the cameras and limitations in the resolution and codec combinations of hardware video decoders the cameras are configured to use the h.265 video encoding and the pipeline to use the corresponding depayloader when receiving data from the cameras

To handle multiple cameras the Gstreamer pipeline is built with an 'inputselector' element that takes in multiple streams of data and sends one on further into the pipeline. Enumerating the element's dynamically linked inputs allows for selection of which input will be passed to the element's single statically defined output.

3.2 Decoder and video converter

8K video requires significant processing power to decode so a hardware accelerated decoder is used. The three major graphics card vendors, Intel, Nvidia, and AMD, all have plugins for Gstreamer to provide this functionality on Linux.

Gstreamer's main repository and most common distributions provides a VA-API based plugin (freedesktop.org, n.d.) that was originally developed by Intel for use with Intel GPUs, but that also supports AMD GPUs using the open-source Mesa graphics driver.

Nvidia provides a plugin as part of their Deepstream Software Development Kit (SDK) for their discrete GPUs or is included in the base Linux image for Jetson platforms. (NVIDIA Corp., n.d.)

AMD additionally provides a plugin as part of their Advanced Media Acceleration (AMD AMA) SDK. (Advanced Micro Devices Inc., n.d.)

All three plugins provide broadly the same functionality, but the initial testing was done using an embedded Nvidia system and AMD VAAPI PC.

The decoder elements are simple in that they take in a bitstream of their specified codec and output framebuffers of a format that the element they are linked to accepts. Gstreamer elements automatically negotiate compatible data formats between inputs and outputs of linked elements

The video converter elements all provide similar features, with crop, scale, and rotate at minimum. The Intel VAAPI postprocess element additionally offers the ability to perform color correction in the same step.

Figure 6 shows a block level layout of the flow of the decoder to scaler to encoder flow offered by the three major vendors. All elements in similar positions in the presented pipelines offer similar capabilities but the naming of parameters may be inconsistent so configuration data cannot be carried over one to one.

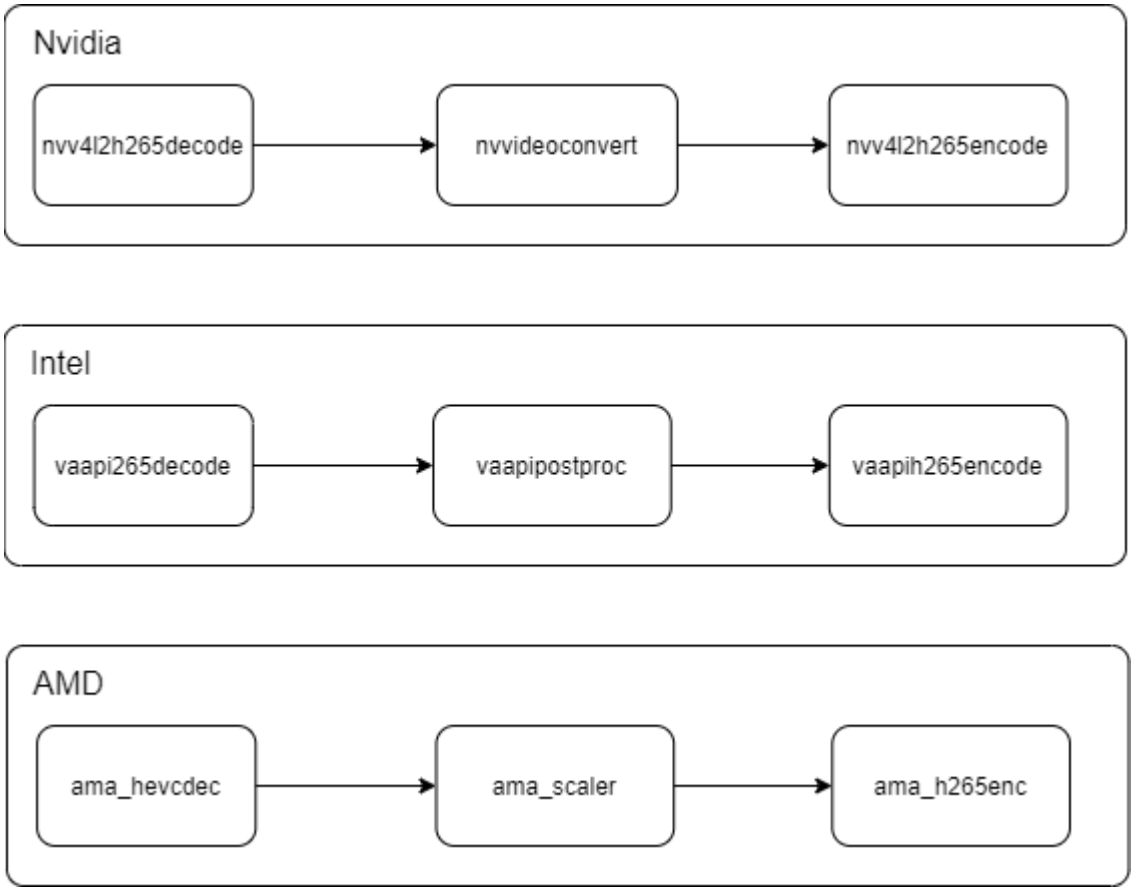

Figure 6. Gstreamer decode/process/encode pipelines for the three major GPU vendors.

3.3 Encoder and Payloader

The final elements in the pipeline before handoff to the RTSP-Server library for transport to clients is the video encoder and RTP payloader elements. The main complexity in this stage is the encoder element which needs to be configured and tuned for the task. All the encoders provide many parameters for tuning the encoding quality and speed but also provide “fast or slow” presets and target bit-rate options which were used to configure the encoder element. The encoders provide modes for constant or variable bitrate encoding. As we desire to control the bitrate the constant bitrate mode is selected, and dynamic bitrate changes are done by modifying the bitrate parameter of the encoder element. This work was not concerned with careful tuning of encoder parameters so

simply the constant bitrate mode was selected, along with a fast encoder preset and then image quality was controlled via the target bitrate parameter.

For the payloader, the only concern is to select a payloader that matches the video codec output by the encoder. RTP has standardized definitions for most common codecs and Gstreamer provides payloaders for most common formats. Assigning a name of the format “pay0” will then make the GST-RTSP server collect the produced RTP packets to be served to clients that connect to the server.

4 GST-RTSP-Server

The Gstreamer project provides their own library for constructing an RTSP standard (Schulzrinne, et al., 2016) compliant server to stream media produced from Gstreamer pipelines. The library can be used to manage pipeline lifetimes, RTP transport modes, user authentication, and transport encryption (freedesktop.org, n.d.).

The programming pattern the library works by is that there is a central server module that listens for RTSP initiation messages and manages client authentication and media configuration. Once a client connects the server uses a factory module to construct a media object that encapsulates the Gstreamer pipeline and RTP connection to the client. The default assumption the library makes is that each media object manages an independent pipeline, but the media factory can be configured so that all constructed media share the same pipeline, reducing redundant processing and allowing for one single control channel to affect the views of all streaming clients.

The data stream delivered to the client is managed by an RTSPMedia class object. An instance of this class is constructed by an instance of the MediaFactory class, or a subclass thereof. A MediaFactory is associated with one or more URLs and when the server receives a request for an URL the linked MediaFactory's construct method is invoked and the media factory constructs the Media element that manages the network connection to the client. The Media element transmits the RTP packets generated by the payloader element in the pipeline and transmits them to the assigned client via the transport protocol defined by the server. The MediaFactory object holds a textual representation of the pipeline the media will use and will parse that representation and construct a pipeline the first time a Media object is constructed. The format of the text representation of the pipeline is the same format as used by the `gst-launch` tool and other tools in the Gstreamer ecosystem to describe pipelines or pipeline segments. If the MediaFactory is configured to share the pipeline between created Media objects it will hold a

reference to the constructed pipeline and use that reference when constructing any later Media objects. The only requirement on this pipeline is that it contains at least one RTP payloader element that is named along the pattern of “pay#” where the last digits are integers. The Media object will stream the output of each payloader element it detects in the pipeline to the client.

The server library automatically handles multithreading client connections and media pipeline processing via a customizable thread pool, as well as managing the pipeline’s running or pausing as clients connect. One possible drawback is that as new clients connect the pipeline clock is restarted, resetting any time elapsed overlays that may have been included in the pipeline.

5 Remote Control with ROS2

The various elements in the Gstreamer pipeline can be reconfigured at runtime but this requires the program to receive updated parameters from the ROC. The mechanism for delivering these parameters was chosen to be the Robot Operating System 2 (ROS2), a software platform that provides a distributed data delivery system and various related tools that an operator can use to record all data transmitted for later replay. (Macenski, et al., 2022)

Other systems on the vessel already use ROS2 to communicate commands and other data, making ROS2 an ideal candidate for the mechanism by which commands would be transmitted to the video processing application, obviating the need to use the lower-level Real-Time Control Protocol that the RTSP server could provide.

Elements of a ROS2 network are called Nodes and have several ways of communicating. These are Topics that operate on a publish/subscribe model where multiple nodes can publish and receive data from a topic. Node parameters can be read from a topic in a namespace belonging to the Node that defined the parameter. A second communication type is Services that operate on a request/response model with only one Node defining the service and being able to process requests but multiple nodes being able to issue requests. Parameter changes operate on the Service models with the Node whose parameters are being changed acting as the server. The server responds with a Boolean value to indicate whether the parameter change was accepted or not. (Open Robotics., 2023)

5.1 Video server paramaters

The processing server instantiates an ROS2 node which defines several ROS2 parameters corresponding to the configuration parameters of the video processing pipelines. The parameters in the initial version are the selected stream and the target bitrate of the encoder with. The current value of these

parameters is published on the ROS2 network, and the node operates a ROS2 Service that can receive messages requesting changes to the values of these parameters. When the node receives a parameter change request a programmer defined callback function is called that validates the value of the parameter being changed and attempts to apply the requested changes to the Gstreamer pipeline by calling the function for changing that property in the pipeline and finally returns a response on whether the parameter change was accepted based on the return value of the Gstreamer property change.

5.2 Interoperability between ROS2 and Gstreamer

To work around both ROS2 and Gstreamer having their own blocking main-loop implementations two approaches were tested to have both systems run simultaneously.

In the first approach the program primarily runs in the Glib mainloop that Gstreamer is dependent on, with this mainloop being set up to regularly call the ROS2 Node's `spin_once()` method that runs one iteration of the ROS2 main loop to invoke the ROS2 message processing. ROS2 messages are thus regularly pumped as the Gstreamer RTSP server process runs and any Gstreamer pipelines will run normally in the context of the GLib mainloop.

The other uses the Python standard library's threading module to create a thread that will start the ROS2 Node's `spin()` method and run it in parallel with the GLib mainloop. This approach requires some additional setup to allow the program to keep responding to keyboard inputs to allow a user to manually shut it down without closing the terminal the program is running in.

6 Conclusion

The goal of this thesis was to produce a tool for the ROC operators to receive low-latency, high-quality video from the cameras installed on the USV and to adjust the bandwidth used by the video stream to compensate for varying connection quality. The product of this work is a software package that can be used to collect video streams from RTSP compatible sources, select one stream for further processing and retransmission to the remote operations center from where an operator can modify the parameters of the video stream processing or change the selected video stream by issuing parameter changes in the ROS2 network that the video processor will receive and give feedback on whether the change was applied to the processing pipeline.

Video processing is carried out with the open-source Gstreamer multimedia framework and utilizes hardware acceleration to improve speed and quality of video decoding and encoding. The modular nature of Gstreamer's GPU-accelerated processing enables the use of accelerators from different vendors with minimal modification to the software by merely changing the few relevant elements in the pipeline definition.

The processed video stream is accessible via the commonly used RTSP protocol, usable via many common media players or possibly even a dedicated viewer application also using Gstreamer. The application automatically manages threading and connections, but the implementation preserves the opportunities for extension and modification that the GST-RTSP library provides.

The system has been tested and has been delivered for integration and the code is available at:
https://github.com/TurkuAISLAB/Remote_controlled_video_processing

6.1 Further development

Further development of the software may involve integrating more camera streams using different input protocols. Another possibility is leveraging the fact that the main surveillance camera feed is higher resolution than the ROC displays to enable higher-quality digital zoom by cropping to a relevant region before encoding the stream for transmission.

References

Advanced Micro Devices Inc., n.d. *AMD Advanced Media Acceleration SDK Documentation.* [Online]
Available at: https://amd.github.io/ama-sdk/v1.0/index.html
[Accessed 5 2024].

freedesktop.org, n.d. *Gstreamer homepage.* [Online]
Available at: https://gstreamer.freedesktop.org/features/
[Accessed 05 2024].

freedesktop.org, n.d. *GStreamer RTSP Server.* [Online]
Available at: https://gstreamer.freedesktop.org/documentation/gst-rtsp-server/index.html?gi-language=python
[Accessed 5 2024].

freedesktop.org, n.d. *Gstreamer VAAPI.* [Online]
Available at: https://gstreamer.freedesktop.org/documentation/vaapi
[Accessed 5 2024].

Kalliovaara, J. et al., 2024. Deep Learning Test Platform for Maritime Applications: Development of the eM/S Salama Unmanned Surface Vessel and Its Remote Operations Center for Sensor Data Collection and Algorithm Development. *Remote Sensing,* Volume 16.

Macenski, S. et al., 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics,* Volume 7, p. eabm6074.

NVIDIA Corp., n.d. [Online]
Available at: https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_plugin_Intro.html
[Accessed 5 2024].

Open Robotics., 2023. *About parameters in ROS 2.* [Online]
Available at: https://docs.ros.org/en/humble/Concepts/About-ROS-2-Parameters.html
[Accessed 5 2024].

Schulzrinne, H. et al., 2016. *Real-Time Streaming Protocol Version 2.0.* s.l.:RFC Editor.