TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-26-05

Real Time Multi Stream Video Transmission in Autonomous UAV

Josey Mol Sibi · Batbayar Battseren · Wolfram Hardt

Mai 2026

Chemnitzer Informatik-Berichte

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Real Time Multi Stream Video Transmission in Autonomous UAV

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc. Automotive Software Engineering

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Josey Mol Sibi
Student ID: 726634
Date: 29.01.2026

Supervising tutor: Prof. Dr. Dr. h. c. Wolfram Hardt
Dr. Batbayar Battseren

Abstract

This study develops and optimises a dual video streaming system to address the development of real-time multi-stream video transmission for autonomous Unmanned Aerial Vehicles (UAVs). UAV mission monitoring and decision-making depend heavily on real-time video feeds, particularly when multiple points of view are required for efficient target tracking, navigation, and ground control operations. The study's main goal is to improve the vision subsystem on an NVIDIA Jetson Nano with limited resources, which can only handle one video feed at a time. The project faces the technical difficulty of enabling dependable and effective dual-stream transmission without causing other mission-critical tasks to be disrupted or overloading the embedded platform. The upgraded streaming component allows the UAV to send and record two simultaneous video streams to the Ground Control Station (GCS) by utilising hardware-accelerated video encoding and effective utilisation of resources. In practical situations, this improves the operators' situational awareness and mission flexibility. The outcomes verify that the improved streaming architecture is completely compatible with the UAV's current system and accomplishes reliable, real-time dual-video transmission within the Jetson Nano's computational constraints. An important step towards flexible, scalable, and real-time visual situational awareness in autonomous UAV operations is demonstrated by this work.

Keywords: Jetson Nano, Hardware accelerated video encoder/decoder, Resource Management, G-Streamer, Ground Control Station(GCS)

Acknowledgement

I would like to show my deepest thankfulness to everyone who helped me and mentored me during the thesis process. At this point, I would like to acknowledge a great debt to Prof. Dr. Dr. h.c. Wolfram Hardt, Chair of Computer Engineering, Technische Universität Chemnitz, who has given me the chance to conduct this thesis. His academic advice, encouragement and quality insights have been of key help in directing this research and providing it the opportunity to become successful. I also wish to tell my great gratitude to my mentor, Dr. Batbayar Battseren, who has been giving me feedback, constructive suggestions, and overall encouragement during the thesis process. This detailed advice and guidance contributed to my being focused and motivated, particularly on the hard stages of the work.

I want to thank Joy Wilson, my partner, the most, who was patient, understanding, and never gave up on me, during this path. It was his faith in me, and constant motivation that enabled me to soldier on and complete this piece of work. Lastly, I would like to say a heartfelt thank you to my friends, family, fellow students and colleagues who have been encouraging, supportive and understood me during this process. I thank each and everyone who helped me in one way or the other to complete this thesis successfully.

Contents

List of Figures

List of Tables

List of Abbreviations

UAV Unmanned Aerial Vehicle

NVIDIA Next Generation Video and Image Development

GCS Ground Control Station

GPS Global Positioning System

WIFI Wireless Fidelity

LTE Long Term Evolution

RTP Real-time Transport Protocol

UDP User Datagram Protocol

WebRTC Web Real-Time Communication

RTSP Real-Time Streaming Protocol

RTMP Real-Time Messaging Protocol

AI Artificial Intelligence

CPU Central Processing Unit

GPU Graphics Processing Unit

SLAM Simultaneous Localization and Mapping

MP Megapixel

RGB Red, Green, Blue

GSD Ground Sample Distance

FOV Field of View

FPS Frames Per Second

HEVC High Efficiency Video Coding

MJPEG Motion JPEG

CBR Constant Bitrate

VBR Variable Bitrate

URL Uniform Resource Locator

OFDM Orthogonal Frequency Division Multiplexing

ROI Region of Interest

VR Virtual Reality

AR Augmented Reality

MR Mixed Reality

TCP Transmission Control Protocol

PC Personal Computer

FLIR Forward-Looking Infrared

IP Internet Protocol

HTTP Hypertext Transfer Protocol

WHEP WebRTC HTTP Egress Protocol

GUI Graphical User Interface

API Application Programming Interface

IO Input/Output

CGI Common Gateway Interface

USB Universal Serial Bus

HTML HyperText Markup Language

CSS Cascading Style Sheets

NVENC NVIDIA Encoder

1 Introduction

Autonomous Unmanned Aerial Vehicles (UAVs) are evolving promptly to become essential components of contemporary surveillance, monitoring, and inspection operations, revolutionising the execution of vital tasks in fields like environmental monitoring, public safety, and disaster response. One of the most important technological developments improving UAV capabilities is real-time video streaming, which makes it easier to monitor missions, make decisions remotely, and keep operators aware of the situation no matter how far away they are or how bad the weather is[21].

Numerous actual missions require a UAV with multiple cameras to capture, compress and transmit more than one video simultaneously over constrained wireless communications through a ground station or an edge server.Examples would include: a two camera inspection, with one camera being able to see the entire site and the other a close-up shot to land the plane; autonomous flying where the plane has a forward camera and a down camera; and search-and-rescue operations, where normal and thermal video is sent out simultaneously to enhance target visibility. In such situations the communication system must be high-speed, maintain high picture quality and must operate even in the event of packets being dropped, or bandwidth variations. Meanwhile the processing power and battery life of the onboard computer of the UAV, such as NVIDIA Jetson, is minimal.This is a tough puzzle to put it all together.

Conventional UAV video connections are typically provided to a single video channel, typically those used by individuals with special analog or digital connections. These solutions are difficult to append streams, and in most cases they do not allow you to modify resolution, codec options or adapt to network variations. Better picture or faster speed: many systems provide neither both nor the detailed controls that are required by the contemporary autonomous functions. As UAVs operate in more challenging environments such as city streets, busy factories or GPS-hard locations, there is an increasing demand to provide large numbers of synchronized video streams in real time. Without a good visual feedback, missions may be impaired, individuals may be unaware or the aircraft may become erratic.

Multi-stream transmission over unmanned aerial vehicles (UAVs) faces a number of constraints, as far as communications are concerned. To begin with, the wireless channel that can be used by the UAV is in most cases are bandwidth constrained and its communication is prone to interference, fading, and unstable link quality. Wi-Fi, LTE/5G or even point-to-point links all have trade-offs in terms of throughput, coverage, and latency. Secondly, the real-time encoding of multiple high-resolution video streams will require an effective use of hardware accelerators found on board-

level platforms; software-only encoding can easily overload the central processor and result in quick energy consumption. The network protocols too must provide low-latency transport (such as RTP/UDP or WebRTC) at the same time without conflicting with the existing visualization tools and allowing integration with cloud or edge services in the future. In turn, the creation of a multi-stream system that can fulfill all of these parameters is a complicated research and engineering task.

To manage this complexity modularly and flexibly, the modern multimedia paradigms should be implemented, and GStreamer has become one of the most popular to create a streaming pipeline. GStreamer provides a wide range of audio-video processing and transmission filling in the capture, processing, encoding, multiplexing and transmission of audio-video data and also supports hardware-accelerated codecs on small embedded platforms such as the Jetson series. In combination with real-time communications protocols such as WebRTC and RTSP/RTMP servers, GStreamer-based pipelines can be managed to provide both end to end solutions in real-time video streaming across heterogeneous network configurations. Such infrastructures are further simplified in containerization infrastructures such as Docker, which promote the deployment and reproducibility of such pipelines. This feature allows developers to develop and test pipelines in a more traditional laptop setting and then transfer them to an embedded system in the most seamless way possible, making the necessary changes as minimal as possible.

Still, existing examples and reference implementations mainly focus on the single-camera setup or basic demonstration pipelines, which provide limited information about designing and optimizing multi-stream pipelines that are simultaneously required to reach the efficiency and stability needed in the case of autonomous UAVs. Challenges related to synchronising streams, per-stream resolutions and bitrates, variable network conditions, streams separations or mixes among different consumers (e.g. operator view vs onboard AI module) are still poorly tackled in most available literature. It is evident that there is a need to carry out a systematic assessment of various multi-stream techniques, such as dual streaming, selective streaming, and stream stitching, under realistic constraints of UAV devices and wireless connections. In that regard, the future research can intensively examine the trade-offs of each of the approaches, including empirical measurements and theoretical modelling, so as to build robust and high-performance multi-stream architectures to be deployed in autonomous aerial vehicles.

The thesis addresses the above challenges by coming up with a Docker and Linux based multi-stream video transmission architecture in autonomous UAVs. The system is designed to work with an embedded platform, such as the NVIDIA Jetson Nano, and it supports the simultaneous operation of several inputs of cameras in one UAV. Expanding on the GStreamer, the framework studies and realizes multi-streaming strategies, such as simultaneous transmission of two camera streams and simultaneous transmission of streams to ground control station (GCS). The architecture is designed in a modular manner, which allows it to easily integrate into existing UAV software stacks and also it can be later augmented with new processing modules, e.g., object detection or video analytics, at a later stage.

The study attempts to create a real-time, end-to-end prototype whereby various video streams are recorded on separate cameras, analyzed and coded onboard and sent across a network to a ground station or browser-based client. In the process, specific focus is directed towards low latency maintenance, ensuring tolerable visual quality, and computational and bandwidth overhead in the constraints of the chosen hardware. The implementation of streaming pipeline in Docker containers facilitates the use of reproducibility and eases the deployment process making the same configuration used in both working environments and UAV hardware with minimal configuration changes. Finally, the paper herein is aimed at filling the gap between the primitive single-camera video demonstration pipelines and more credible, viable multi-stream solutions applicable to the practical autonomous UAV application.

1.1 Motivation

The advanced speed of the autonomous unmanned aerial systems and their application in more sophisticated operational environments, new solutions are necessary to ensure the reliability of perception, situational awareness, and safety of missions. UAVs have recently been used in inspection, surveillance, disaster response and environmental monitoring, where timely and accurate understanding of the environment is much needed. The use of a single camera stream is limiting in these situations: operators and onboard algorithms can either not have a wider situational view or cease to provide detailed visual granularity, especially when there is a lot of clutter or dynamic situations. This drawback is serious when dealing with time-sensitive missions, like search and rescue, disaster management, infrastructure inspection or low-altitude urban flights, where poor visual feedback or less information may cause missed targets, unsafe methods or mission failure.

Under the current paradigm, the need to develop multi-camera, real-time video transmission networks that have the capacity to relay augmented visual data based on the same UAV platform is very necessary. Dual streaming, or simultaneous delivery of two different camera feeds, explicitly responds to this need. An example of this is a forward-facing navigation display and a downward or zoomed-in inspection display. In turn, this arrangement allows human operators to maintain global situational awareness and at the same time focus on specific areas of interest, as well as autonomous algorithms to combine complementary views, and thus strengthen the robustness of decision-making. However, implementing one of those systems on the resource-limited embedded systems introduces significant issues: several streams should be recorded, coded, and sent with the bandwidth-constrained wireless connections with low latency and not overloading the onboard hardware.

Existing UAV video connections and sample pipelines are largely built for single-stream transmission and are frequently targeted for either human viewing or simple demonstration reasons, rather than robust multi-stream functioning in real missions. They often do not address how to manage and optimise two concurrent video streams in terms of quality, bitrate, latency, and reliability under actual network conditions.

In order to close that gap, this thesis designs, implements, and assesses a dual streaming architecture for autonomous UAVs that uses a Docker and Linux-based framework to capture, process, and transmit two camera streams in real time using hardware-accelerated pipelines. By illustrating that dual streaming can be carried out easily and safely on embedded hardware, this research aims to increase operator awareness, support sophisticated perception modules, and subsequently lead to safer and more successful autonomous UAV operations.

1.2 Problem Statement

For autonomous unmanned aerial vehicles (UAVs) to be really useful and safe in real-world tasks, they must be able to offer reliable, real-time visual feedback from the air to ground. In order to help human operators and onboard algorithms in making predictions under time constraints, many civil applications, including infrastructure inspection, environmental monitoring, security, and disaster response, rely on quick and precise visual information. Relying on a single camera feed is sometimes insufficient in such operations. While autonomy modules may benefit from numerous viewpoints to identify hazards, track objects, or confirm landing zones, operators may require both a broad situational overview and a zoomed-in or downward-looking view. This makes dual streaming—the simultaneous real-time transmission of two video streams from a single UAV—clearly necessary.

However, transmitting two concurrent video feeds from an aerial, resource-constrained aircraft is technically problematic. Many use cases demand minimal end-to-end latency (typically well below a second) so that operators can react fast and autonomous controllers may safely adapt to changing situations. However, the bandwidth of wireless networks accessible to UAVs, such as Wi-Fi, LTE/5G, or proprietary radio, is restricted and fluctuates. Existing academic and industry solutions suggest that low-latency UAV video streaming is conceivable, but they are either specialised to specific hardware or single-stream setups, or they are closed commercial systems that do not divulge their internal architecture. Problems including bandwidth sharing, bitrate and resolution selection for each stream, and minimal latency become even more crucial when many streams are involved.

At the same time, common onboard processors for UAVs—such as NVIDIA Jetson-based platforms—have limited CPU capacity and stringent power budgets. Particularly at higher resolutions and frame rates, encoding two video streams only in software might quickly overload the device or exhaust the battery. Modern multimedia frameworks like GStreamer and hardware-accelerated encoders on Jetson platforms give the building pieces to develop efficient pipelines, but transforming them into a stable, dual-stream, end-to-end solution still takes careful design and integration. Specifically, an architecture that achieves real-time speed while being modular, reliable (e.g., via Docker and Linux-based deployment), and simple to incorporate into current UAV control stacks is required.

Therefore, the main issue this thesis addresses is:How can we design and imple-

ment a real-time dual streaming framework for autonomous UAVs that uses an open, Docker- and Linux-based architecture that can run on embedded platforms like the Jetson Nano to capture video from two cameras, efficiently encode both streams using hardware-accelerated pipelines, and transmit them over bandwidth-limited wireless links with low latency and acceptable quality? To tackle this challenge, the effort focusses on constructing and testing a dual-streaming system based on GStreamer pipelines and containerised services. The system should allow two separate camera feeds to be taken and broadcast concurrently, expose customisable options such as quality and bitrate, and be appropriate for further integration with real UAV hardware and missions.

1.3 Research Objective

The main goal of this project is to create, build, and test a real-time dual streaming framework for autonomous UAVs that can continuously broadcast video from two cameras over a wireless network utilising a Docker- and Linux-based architecture. The system is intended to work on embedded systems such as the NVIDIA Jetson Nano and to facilitate integration into actual UAV control stacks in subsequent phases. Building GStreamer-based pipelines that effectively record, encode, and stream two concurrent video feeds while preserving low latency and acceptable visual quality under practical bandwidth restrictions is the major goal of this study.

To achieve this purpose, a modular streaming framework is designed that divides capture, processing, and transmission into well-defined components. In the first development phase, the framework employs a laptop webcam and a mobile camera to imitate a dual-camera UAV system. These two streams are processed by GStreamer pipelines, contained in Docker containers, and supplied to a ground-side client (e.g., browser or ground control station) for real-time visualisation. The architecture makes use of the platform's hardware-accelerated encoders to enable the same pipelines to be deployed on the Jetson Nano with no alteration.

The research also seeks to provide configuration and control techniques for the dual streaming system, such as per-stream resolution and bitrate settings, and the ability to start, stop, and monitor individual streams throughout runtime. This allows the operator or higher-level autonomous modules to modify the streaming behaviour in response to network circumstances and mission requirements. The viability of operating two concurrent streams on limited hardware is evaluated by gathering and analysing performance indicators, such as end-to-end latency, CPU/GPU utilisation, and effective throughput.

Lastly, the dual streaming system is assessed as a basis for future integration with modules for autonomous perception and decision-making. By proving that two video streams can be safely broadcast in real time from a single UAV aircraft, this research attempts to offer greater situational awareness for human operators and give multiple visual inputs for onboard algorithms. Reaching these goals advances the creation of open, reusable, and useful multi-stream video transmission systems

for autonomous UAV applications.

1.4 Thesis Outline

This thesis is divided into eight chapters. Chapter 1 presents the problem of research, motivation, objectives and the scale of work in general. Chapter 2 is a thorough review of related work, and other current methods to transmit videos in real-time and multi-stream in UAV platforms and their limitations, showing the gaps in the research. Chapter 3 outlines the system requirements, design considerations and the architecture of the proposed multi-stream video transmission framework in general. Chapter 4 gives an account of how the proposed system was implemented such as hardware platform, software environment and building video streaming pipelines. Chapter 5 describes experimental system and assessment methodology to measure the system performance. Chapter 6 introduces the results of the quantitative performance evaluation which are CPU, GPU, memory usage, stress testing, and system stability analysis. Chapter 7 presents the results gained, limitations, and future research directions and possible improvements. Last but not least, Chapter 8 will conclude the thesis by summarizing the contributions and the main findings of the work.

2 Background Research

Here we outline the current advancements and techniques related to real-time video transmission for autonomous UAVs, with a special focus on multi-camera and dual-streaming systems. The combination of onboard cameras, low-latency communication lines, and multimedia frameworks such as GStreamer has become crucial for giving accurate visual input to operators and autonomous algorithms in complicated missions. As a basis for understanding how these technologies support perception, navigation, and decision-making in UAVs, this chapter reviews prior work on UAV communication links, real-time video encoding and streaming protocols, multi-camera UAV platforms, and embedded hardware such as NVIDIA Jetson devices. The chapter also examines current designs that leverage Docker- and Linux-based deployments for streaming apps. Through this research, we highlight limitations in current single-stream or proprietary systems and motivate the need for an open, dual streaming framework that can effectively transmit two concurrent video feeds from a single UAV in real time.

2.1 Autonomous UAVs and Visual Perception

Autonomous Unmanned Aerial Vehicles (UAVs) have developed from basic remotely piloted platforms into more sophisticated systems that can coordinate with other agents, plan routes, and avoid obstacles. Their applications cover infrastructure inspection, precision agriculture, environmental monitoring, border surveillance, and search-and-rescue missions. In practically all of these scenarios, visual perception plays a crucial role: cameras give extensive information about the surrounding environment that cannot be gained via GPS and inertial sensors alone. Using onboard vision, UAVs may identify landmarks, detect obstacles, assess structural damage, monitor crops, or locate persons in distress, making vision a critical enabler for both autonomy and human-in-the-loop supervision[16].

Modern UAVs rely on computer vision and machine learning algorithms operating either onboard or at the edge to extract meaningful information from camera data. Semantic segmentation, visual odometry, simultaneous localisation and mapping (SLAM), and object identification and tracking are typical tasks[48]. For human operators, live video feeds serve as an extension of their own senses, helping them to retain situational awareness, evaluate mission progress, and respond to unforeseen situations.

The camera system becomes one of the most important parts since visual data is so essential to UAV operation. During the operation, UAV cameras control what

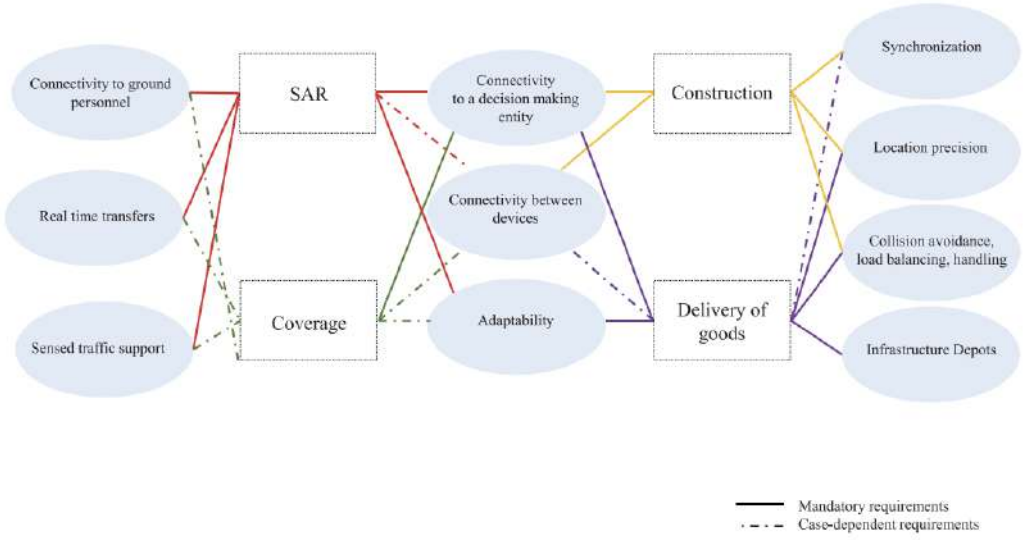

Figure 2.1: Illustration of the application scenario[16]

can be seen, how well it can be seen, and how efficiently the data can be processed or broadcast. The choice of camera basically depend upon the application domain and the different types of cameras used are:

- **RGB Cameras:** Standard colour cameras used for inspection, mapping, and monitoring. When flown at the right altitudes, high-resolution RGB sensors (12–48 MP) may attain centimeter-level ground sampling distance, making them perfect for high-detail mapping and structural examination[8].
- **Multispectral Cameras:** Capture reflectance in multiple spectral bands for vegetation analysis. Reflectance in a limited number of distinct spectral bands (usually 4–10), such as the visible and near-infrared ranges, is recorded by multispectral sensors[**?**]. When correctly calibrated, UAV-mounted microbolometers may provide accurate thermal maps, according to studies comparing UAV thermal cameras with terrestrial sensors.
- **Thermal Infrared Cameras:** Measure surface temperature for industrial inspection and search-and-rescue[19].
- **Hyperspectral Cameras:** Provide detailed spectral information for scientific monitoring. Large data volumes are produced by these sensors, necessitating reliable transmission lines and robust onboard processing[19].

Along with the cameras the parameters also strongly influence UAV performance in mapping, inspection and video streaming:

- **Spatial Resolution and Ground Sampling Distance (GSD):** The amount of detail that the camera can record depends on its spatial resolution. According to UAV research, GSD values of a few centimetres per pixel are typical,

allowing for the fine-scale identification of vegetation features, flaws, and fissures.

- **Field of View (FOV):** While narrow-angle or zoom lenses provide in-depth examination of far-off structures, wide-angle lenses improve situational awareness and are helpful for navigation and mapping.
- **Frame Rate:** How many pictures (frames) are displayed in the video per second?
- **Resolution:** The video's clarity and detail, expressed in pixels.
- **Bandwidth:** The maximum amount of data that can be sent across a network in order to deliver video.
- **Bitrate:** how much data is actually utilised every second to encode and send the video.

The limits of single-camera systems have become apparent as UAV applications have expanded. For instance, nadir-looking cameras are useful for mapping missions, but oblique views are necessary for inspecting facades, towers, or rooftops. Similar to this, low FOV and greater magnification are needed for defect identification or thermal diagnostics, whereas broad FOV is frequently required for navigation[32]. By concentrating on real-time multi-stream video transmission from such multi-camera UAV platforms—with a special emphasis on dual streaming for simultaneous navigation and inspection—this thesis expands on these advancements.

2.2 Video Streaming Fundamentals

One of the most crucial features of contemporary UAV systems is video streaming. In order for the operator or onboard computer to comprehend what is going on in the environment, live video must be received, regardless of whether the drone is being used for inspection, agricultural, or search and rescue. Because of this, it's critical to comprehend how video is produced, compressed, and sent from the UAV to the ground station.

A series of pictures (referred to as frames) sent constantly via a communication channel is all that makes up a video stream. For instance: 30 fps means the drone sends 30 images every second. A smoother video is one that has higher fps. Wireless connectivity (Wi-Fi, 4G/5G, radio modems) are typically used to transmit UAV footage from the drone to the ground. The video needs to be compressed before being transmitted since these links have a restricted bandwidth[7]. Research on UAV monitoring reveals that most real-world situations make it difficult to relay raw video, and effective compression is crucial for real-time performance.

After acquiring the video , the next step is to properly compress it because raw videos are extremely large. An Uncompressed 1080p video, for instance, might contain hundreds of gigabytes every second. UAVs must utilise video codecs to minimise

the data size since they frequently operate far away and rely on constrained wireless channels. H.264, H.265 (HEVC), and MJPEG are the most widely used compression formats. Because it performs well on embedded devices like the NVIDIA Jetson and offers high video quality at comparatively low bitrates, H.264 is the most popular. Although it demands more processing power, H.265 delivers even greater compression, allowing it to broadcast higher-quality video at lower bitrates[52]. Although MJPEG is straightforward and quick to decode, it consumes a lot more bandwidth and is typically not appropriate for long-range UAV flights. H.264 and H.265 are frequently identified as the best options for real-time aerial imagery in research on UAV video systems. After properly compressing the video, these fromated videos of appropriate parameters are send over the network. The network protocols and the techniques may vary accoring to the application which will be discussed in detail in further sections.

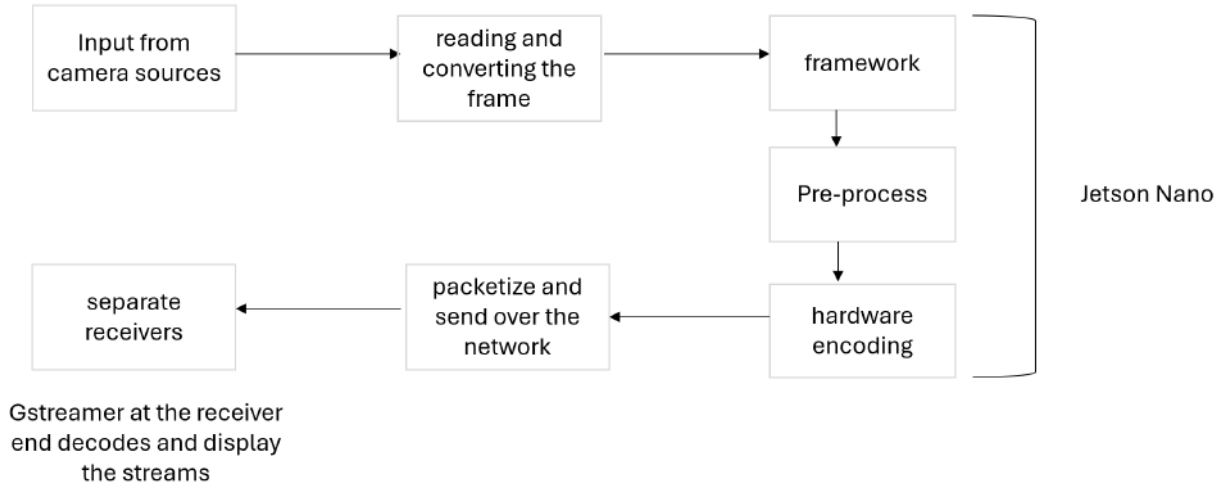

Figure 2.2: Illustration of video streaming process

There are certain key factors which has to focused on while discussing about the streaming process. The most important factor is the relationship between latency and bandwidth. The greatest amount of data that can be transmitted via a wireless link is known as bandwidth, while the time it takes for a drone to capture a frame and show it on the ground is known as latency[42]. Low latency is necessary for safe and responsive operation so that the operator does not notice a noticeable delay in what the drone observes. Bitrate management, which controls the amount of data sent per second, is another crucial idea. The two modes that UAV video encoders often use are Constant Bitrate (CBR) and Variable Bitrate (VBR). While CBR helps ensure consistent performance over poor or erratic wireless connectivity by maintaining a fixed bitrate, it may degrade video quality in complicated scenarios[20]. While VBR improves visual quality by allowing the bitrate to rise during high-motion or high-detail situations, it may result in traffic spikes that overwhelm the network.

When numerous video streams are delivered simultaneously, as in dual-stream or multi-stream systems, these principles become much more crucial. The UAV must handle video quality, latency, and stability much more carefully when several cameras are in use because this increases bandwidth usage. Analysing sophisticated multi-streaming strategies covered later in this thesis requires an understanding of these streaming fundamentals.

2.3 Basic Networking for UAV Video

For a UAV to transmit live video to the ground station, reliable networking is necessary. If the network link is unable to transfer the video smoothly, the system will fail even if the camera takes excellent pictures and the encoder effectively compresses them. The primary components of UAV video networking are wireless connectivity and lightweight transport protocols, which must function in ever-changing environments including mobility, interference, and different distances.

2.3.1 RTP, RTSP and RTMP Basics

The Real-Time Transport Protocol (RTP) is the foundation of most UAV video systems. RTP is designed for transferring audio and video with little latency, and it operates over UDP, which does not wait for missing packets to be retransmitted. This tendency is essential because real-time[40] video is more sensitive to delay than to an odd missing frame. RTP is commonly employed in drones, remote cameras, and surveillance systems because to its simplicity and minimal overhead. RTP is frequently used in conjunction with RTSP (Real-Time Streaming Protocol). Instead of carrying the video itself, RTSP functions as a "remote control" that instructs the server on how the client will receive the data, which codec to use, and when to begin or stop streaming. Due of the ease of integration with ground control software, many commercial drones publish their camera feed via an RTSP URL.

Another popular choice is RTMP (Real-Time Messaging Protocol), particularly when streaming video to distant servers or cloud platforms is required. Drone video is still sent to internet dashboards or distant processing systems via RTMP, which was first created for live broadcasting[38]. It is helpful for activities in industrial environments that are outside line of sight due to its consistent performance and broad support. The general working idea of these protocols are given in Figure 2.3 and 2.4.

Protocol	**Key Feature**
RTP	Sends audio/video packets in real time (delivery layer)
RTSP	Controls the stream (play, pause, stop)
RTMP	Best for low-latency live streaming

Table 2.1: Simple Comparison of RTP, RTSP, and RTMP

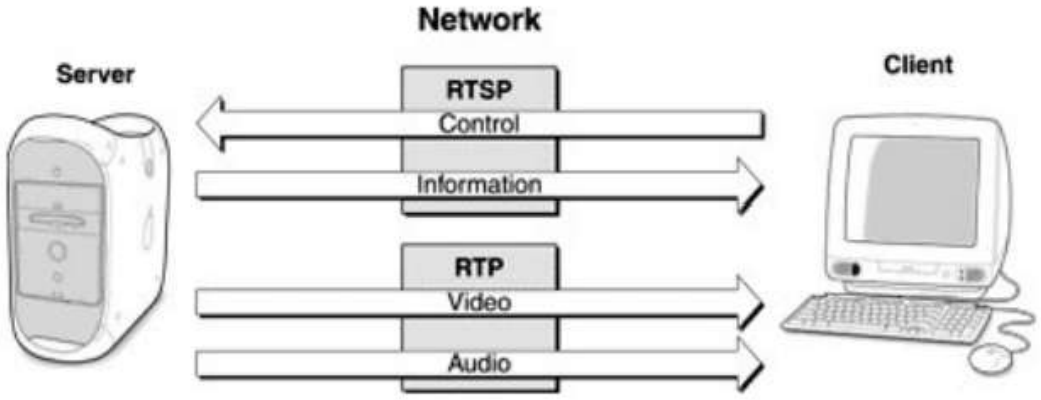

Figure 2.3: RTP Streaming Session[30]

2.3.2 WebRTC Basics (ICE, STUN/TURN, RTP over UDP)

Due of its very low latency and direct browser compatibility, WebRTC has lately gained popularity for UAV video. WebRTC, in contrast to RTSP or RTMP, is intended for interactive communication, including video calls[23]. Even in challenging wireless conditions, it intelligently adjusts the bitrate, manages network congestion, and keeps a steady connection. Eventhough internally it uses, RTP over UDP, there are some additional mechanisms.

- **ICE (Interactive Connectivity Establishment:** assists the drone and base station in determining the optimal connection method, even in the presence of firewalls.
- **STUN/TURN servers:** When direct peer-to-peer connections are not feasible, help devices find their public IP addresses and create relay pathways.
- **RTP over UDP:** guarantees that video packets are delivered with the least amount of delay feasible.

WebRTC performs better than conventional streaming protocols in terms of latency and flexibility, according to recent UAV communication tests, making it appropriate for multi-camera and dual-stream UAV systems[2].

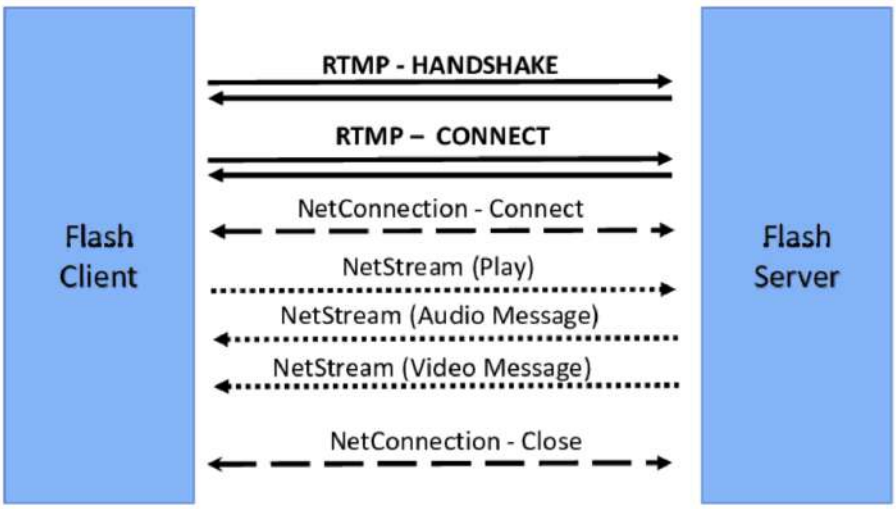

Figure 2.4: RTMP Streaming Session[38]

2.3.3 Wireless Link Constraints (5G, Wi-Fi, Long-Range Radio)

The wireless connection that transmits the data has a significant impact on UAV video streaming performance. Because Wi-Fi is easy to use and provides high bitrates at close range, it is frequently seen in tiny, short-range drones. Its range is constrained, though, as it is susceptible to interference from obstructions, other networks, and buildings.

4G and 5G mobile networks provide more consistent coverage and a greater range. For UAV applications, 5G in particular is appealing since it offers enormous bandwidth and extremely low latency[10]. However, because the link quality is highly dependent on the drone's position in relation to the cell tower, real-world testing reveal that even 5G performance varies greatly as the drone rises to higher altitudes or moves behind obstructions.

When the drone has to function many km distant, long-range radio solutions such bespoke OFDM connections or 900 MHz telemetry radios are utilised. Though they often have far less capacity than Wi-Fi or 5G, these systems offer a reliable connection across great distances. Drones must modify their video bitrate and quality due to these bandwidth constraints in order to prevent overloading the network. In general, range, interference, mobility, and environmental factors restrict UAV wireless networks[10]. One of the main issues with UAV video transmission, according to surveys, is unreliable bandwidth, particularly when numerous video streams need to be delivered simultaneously.

2.4 Multi-Stream Concepts

Several cameras are frequently carried by modern UAVs, such as a thermal camera for temperature monitoring, a zoom camera for inspection, or a wide-angle camera

for navigation. The drone must choose how to transmit its video feeds to the ground station when it has numerous cameras. There are several multi-stream notions, each with a distinct function and degree of intricacy. The more sophisticated streaming methods employed in dual-camera UAV systems are based on these ideas.

2.4.1 Dual Streaming

When a UAV uses dual streaming, it simultaneously transmits two live video feeds. Typically, separate cameras or the same camera with various quality settings provide each feed. With this method, even the operator can see two viewpoints simultaneously, such as a zoomed view for detail inspection and a broad view for navigation. Both the video feeds are visible at the same time on the same screen. Dual streaming is extensively utilised in inspection drones and teleoperation interfaces because operators benefit from knowing both context and information concurrently[36]. Two complementing perspectives enhance situational awareness and save operator burden, according to research on multi-camera systems for UAV navigation and teleoperation.

2.4.2 Selective Streaming

A more adaptable kind of multi-streaming is selective streaming. The UAV only transmits the most crucial visual data when required, rather than sending two complete streams continuously. For instance, the drone may constantly broadcast a basic navigation stream but only send a zoomed-in or high-resolution video upon request from the operator or when an onboard algorithm finds anything intriguing (such a person or a fault). When the bandwidth of the wireless connection is constrained, selective streaming might be helpful. According to research on Region-of-Interest (ROI) streaming, delivering only the crucial portions of the image can drastically cut down on bandwidth use while maintaining the level of information required for tasks like monitoring or inspection[24].

2.4.3 Stitching

The process of combining several camera perspectives into a single, expansive, or panoramic video is known as stitching. The operator sees a single, big picture that combines all perspectives rather than many streams. This produces a uniform broad field of vision, which is helpful for mapping, surveillance, and 360-degree monitoring[25]. However, stitching can be difficult when the drone is flying swiftly and requires more processing power and accurate camera alignment. Stitching enhances global awareness, but it is computationally costly and sensitive to motion or illumination changes, according to studies on multi-camera panoramic systems[46].

2.4.4 How Multi-Stream Videos Can Be Displayed

Apart from these methods or techniques, there are different ways in which the videos can be visually seen at the receiver side. These possible ways make use of the above mentioned techniques to visually present the data at the front end of the receiver. Let us now discuss the common methods. The choice of display solely depends upon the situational awareness, decision making and other relevant factors.

Picture-in-Picture (PiP)

One video stream is shown as the primary view in Picture-in-Picture layouts, with the second stream appearing as a smaller window superimposed on top. Because it enables the operator to maintain a steady navigation view while concurrently watching a magnified or thermal image, this configuration is frequently utilised in UAV inspection[41]. PiP reduces the requirement for frequent view switching while preserving context, according to teleoperation research, particularly when one stream is obviously more significant than the other.

Side-by-Side (Tiled View)

Two video streams are shown next to one another at comparable sizes in side-by-side layouts. When both perspectives are equally important, such in RGB and thermal images or front- and downward-facing cameras, this method is helpful. Because they enable direct comparison of viewpoints without favouring one over the other, tiled layouts are frequently used in multi-UAV and multi-camera monitoring platforms[23]. However, research also shows that if the user needs constantly switch between windows, tiled screens may increase visual burden[35].

Switchable or Focus-Based Layouts

Only one stream is displayed at full size on some computers, even though numerous streams are always accessible. While the other streams stay reduced to thumbnails, the operator may rapidly change which stream is in focus. This method keeps all views accessible while minimising screen clutter. Studies on human-robot interfaces suggest quick switching techniques (controller buttons, keyboard shortcuts) to reduce reaction times during flight or inspection[22].

These are the common methods and other possible methods are also listed in the table below.

Layout	**Key Idea (Short Description)**
Picture-in-Picture (PiP)	Main view with a small inset window for secondary stream; good for navigation + detail.
Side-by-Side (Tiled)	Two equally sized streams shown next to each other for direct comparison.
Switchable / Focus-Based	One full-size stream with others as thumbnails; operator switches focus quickly.
Context-Plus-Detail	Zoomed view linked spatially to a wide-angle view (e.g., Monocle interface).
Overlay / Information Fusion	Thermal, detection, or other data overlaid directly on RGB to reduce window count.
Immersive / XR-Based	Streams placed in a 3D VR/AR/MR environment for enhanced spatial awareness.

Table 2.2: Concise summary of multi-stream display layouts for UAV interfaces.

3 State of the Art

Research on real-time video transmission systems has increased due to the increasing use of unmanned aerial vehicles (UAVs) for intricate tasks including environmental monitoring, infrastructure inspection, surveillance, and search and rescue. Both autonomous UAV operation and human-in-the-loop supervision rely heavily on visual data to assist tasks including navigation, obstacle avoidance, scene comprehension, and decision-making. Traditional single-stream video transmission methods are no longer adequate to satisfy the demands of contemporary applications as UAV platforms increasingly incorporate several cameras and sensors. Early UAV video systems prioritised either navigation or monitoring functions and mostly concentrated on sending a single camera feed to a base station. Recent developments in UAV technology, however, have made it possible to combine wide-angle, zoom, thermal, and multispectral cameras on a single platform enabling multi-camera and multi-sensor payloads[48]. Research into multi-stream video transmission—where numerous video feeds are concurrently recorded, encoded, sent, and displayed to give both global situational awareness and precise inspection capabilities—has been spurred by this progression[51].

Concurrently, networking and video compression technologies have advanced significantly. While more modern frameworks like WebRTC provide adaptive bitrate management and low-latency communication over diverse networks, traditional real-time transport technologies based on RTP and RTSP have long been employed for UAV video distribution[40][8]. The design space for UAV video streaming systems has grown as a result of these advancements, opening up new designs that can more effectively handle latency limitations and bandwidth fluctuations. Simultaneously, academics have investigated several approaches to handle the higher volume of data that comes with simultaneous video feeds. These include stitching-based panoramic techniques, which combine several camera views into a single wide-field stream; selective or region-of-interest-based streaming, which prioritises crucial visual information; and dual streaming, which transmits two complementary video feeds simultaneously[36]. Although each strategy has unique benefits, there are trade-offs in terms of latency, computational complexity, bandwidth use, and system robustness[26].

The state of the art in real-time multi-stream video transmission and multi-camera UAV systems is reviewed in this chapter. System designs, streaming protocols, bandwidth-efficient strategies, and visualisation approaches are all addressed in the literature. This chapter sets the stage for identifying unresolved issues and inspires the research path taken in this thesis by examining current methods and their shortcomings.

3.1 Multi-Camera and Multi-Sensor UAV Payloads

When compared to conventional single-camera platforms, a number of studies have shown that providing unmanned aerial vehicles (UAVs) with numerous cameras or heterogeneous sensors greatly improves their perception skills. For simple visual monitoring and mapping duties, early UAV systems usually depended on a single RGB camera. However, the drawbacks of single-view vision became more apparent as UAV applications grew to encompass search and rescue operations, autonomous navigation in challenging situations, and infrastructure inspection. Particularly in complex situations with occlusions or different views, a single camera frequently falls short of providing sufficient spatial coverage, redundancy, and detail.

Wierzbicki, for example, demonstrated a multi-camera imaging system for UAV photogrammetry that increased the effective field of view and ground coverage by mounting multiple cameras. The study shown that multi-camera payloads increase reconstruction quality and lower the number of necessary flying passes, especially in urban settings where occlusions are caused by buildings and other objects[48]. This study unequivocally showed that multi-camera setups allow for more thorough and dependable data collection than single-camera methods. Nadir and oblique camera combinations have been utilised in similar photogrammetry-focused investigations to concurrently collect images from numerous perspectives, enhancing geometric robustness and feature visibility in three-dimensional reconstruction.

Additionally, multi-camera payloads have been thoroughly investigated for duties pertaining to safety and navigation. Using five cameras positioned in various directions around the UAV to continually examine its surroundings, Zaměy et al. created a real-time multi-camera vision system for UAV collision warning and navigation[51]. Their findings demonstrated that multi-view perception enhances situational awareness and obstacle identification, especially in congested or dynamic surroundings where a single forward-facing camera is inadequate. Several perspectives lessen susceptibility to occlusions and motion blur, resulting in more consistent localisation and navigation performance, according to other studies on multi-camera visual SLAM[50]. These pieces demonstrate how various cameras frequently perform distinct functional tasks, such as lateral situational awareness, downward-looking localisation, or forward-looking obstacle detection.

Many UAV platforms incorporate diverse sensors in addition to numerous RGB cameras to collect additional data. Because thermal infrared cameras can detect temperature differences that are invisible in RGB video, they are frequently utilised in industrial inspection, energy audits, and search and rescue operations. Torres-Rua et al. assessed UAV-mounted thermal cameras and showed that they can produce accurate thermal readings appropriate for structural and environmental monitoring when calibrated correctly[44]. Another crucial sensing technique is multispectral cameras, especially in environmental monitoring and precision farming. According to a thorough investigation by Zhang et al., multispectral UAV photography outperforms satellite-based methods in many situations by enabling extensive crop-health assessments at extremely high spatial resolution[52].

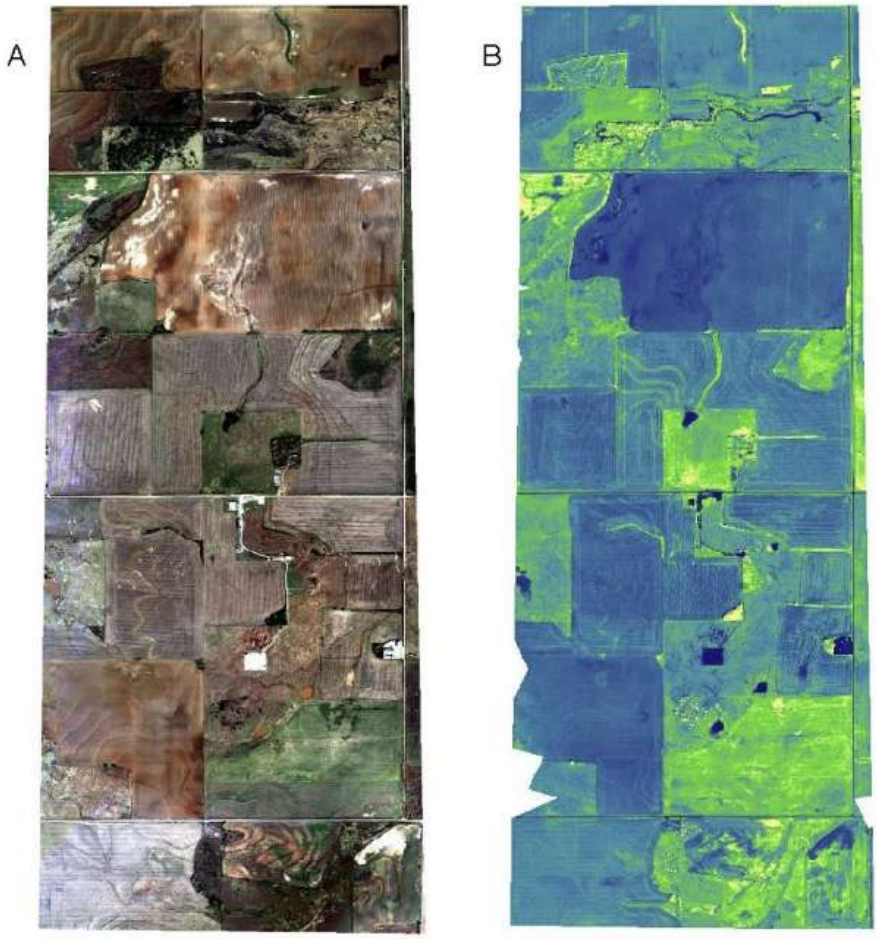

Figure 3.1: Sample photos taken at 820 meters above sea level (AGL) with the normalised difference vegetative index (NDVI) estimated on the right and actual colour on the left[12]

UAV perception capabilities are greatly expanded by the integration of many sensor types, but system complexity is also increased. Onboard processing and real-time data transmission are complicated by the fact that separate sensors usually operate at varying resolutions, frame rates, and noise characteristics. Furthermore, a significant increase in data volume results from many cameras or sensors operating simultaneously. This presents difficulties for embedded UAV platforms, which have constrained energy and processing capabilities. A number of technological difficulties are frequently documented in the literature, despite the obvious benefits of multi-camera and multi-sensor UAV payloads. One big issue is the increased volume of data produced by several high-resolution sensors, which puts a lot of load on wireless communication lines, which are sometimes unreliable and have limited capacity. Synchronisation and temporal consistency across sensor streams, which are necessary for dependable navigation, inspection, and multi-view visualisation, provide another

difficulty[51]. Additionally, real-time processing and encoding pipeline complexity is limited by onboard computing limits, especially on lightweight UAV platforms where hardware acceleration needs to be carefully controlled.

All things considered, current research unequivocally shows that multi-camera and multi-sensor payloads significantly enhance UAV perception and mission performance. However, the end-to-end issue of real-time multi-stream video transmission receives very little attention, with the majority of research concentrating mostly on sensor hardware and perception algorithms. Specifically, lightweight, repeatable solutions that specifically address how many camera or sensor streams should be encoded, transferred, and displayed to operators under stringent latency and bandwidth limits are lacking. This gap encourages more research into effective multi-stream transmission designs, such dual-stream systems, which are examined in this thesis's later sections.

3.2 Multi-Streaming and Multi-View Teleoperation Interfaces

Researchers became more aware that sending a single video stream is frequently insufficient for efficient teleoperation and supervision as UAV platforms developed to carry several cameras and sensors. In many missions, operators have to concentrate on fine-grained features like impediments, structural flaws, or people of interest while still maintaining global situational awareness. Multi-streaming techniques, which broadcast numerous video feeds simultaneously, and multi-view teleoperation interfaces, which show these feeds to the operator in a user-friendly and understandable way, have been developed as a result of this necessity.

When compared to single-view systems, a number of studies show that multi-camera visual feedback greatly enhances operator performance. Recchiuto et al., for example, examined the use of numerous cameras in a UAV human-swarm interface and shown that concurrently providing various perspectives improves situational awareness and spatial knowledge during remote operation[36]. Their findings show that when several visual viewpoints are provided, operators are better able to evaluate the surroundings and the status of the UAV, especially in complicated or dynamic settings.

Multi-view teleoperation interfaces have been investigated in more general monitoring and control contexts, going beyond dual-camera configurations. A WebRTC-based multi-UAV monitoring systems that shows many real-time video feeds in a browser-based ground control station was created by Kilic et al. The same interface ideas apply to a single UAV with several camera streams, even though their work mainly targets numerous UAVs[23]. The study shows that contemporary streaming frameworks can theoretically handle several video streams at once, but it also emphasises how crucial proper layout design is to prevent operator overload.

Research on human-robot interaction further demonstrates that the presentation

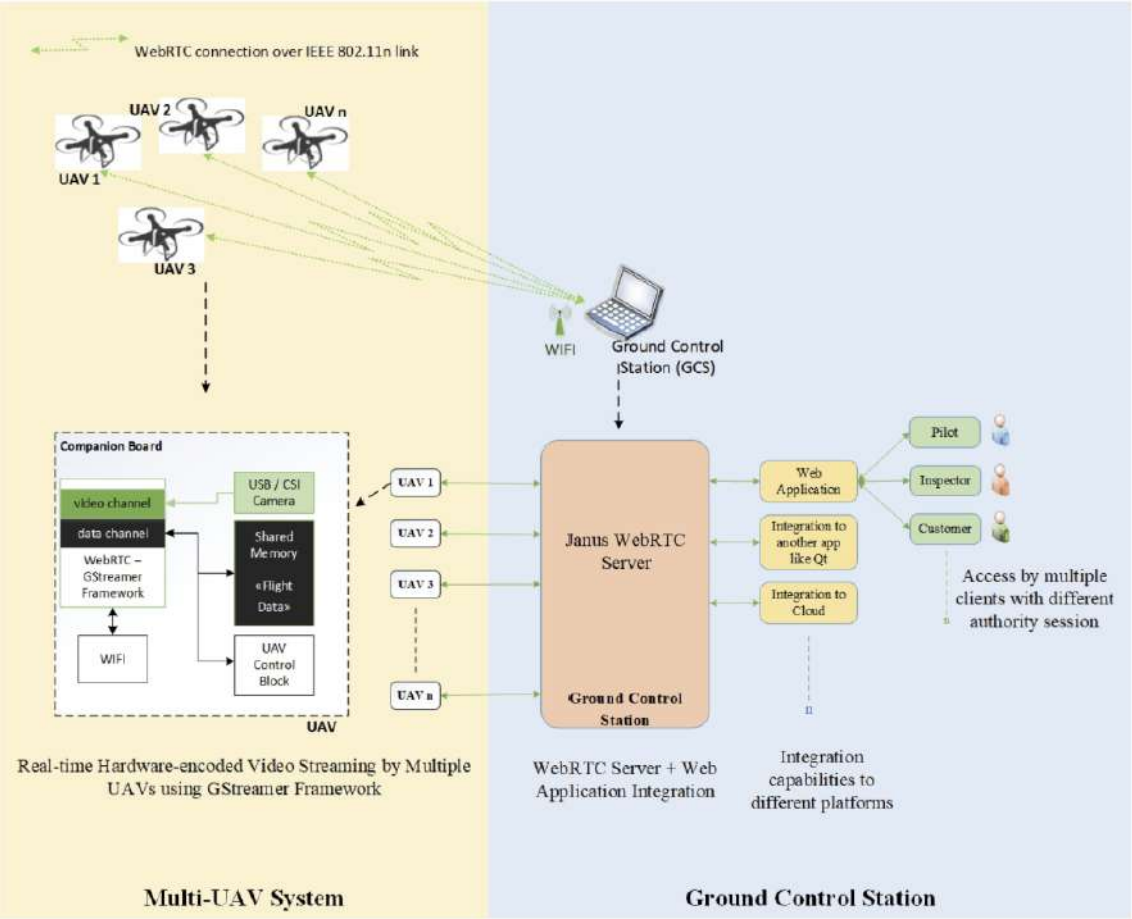

Figure 3.2: General architecture for multi-streaming[23]

of numerous streams is just as significant as their quantity. Multi-view interfaces can enhance work performance, according to surveys by Kazanzides et al. and Rea et al., but badly designed layouts may increase cognitive load and decrease efficacy[35][22]. Picture-in-picture views, side-by-side displays, and focus-based layouts—in which one stream is highlighted while others are shown as thumbnails—are examples of often observed layouts. A trade-off between visual intricacy and information richness is represented by each arrangement.

Beyond interface design, multi-streaming presents new technological difficulties from a system standpoint. Real-time transmission of multiple video streams uses more bandwidth and imposes more stringent constraints on network stability and encoder design. Research on UAV video transmission highlights that although inspection or analysis streams may accept somewhat increased latency in exchange for improved quality, navigation-critical streams must retain low latency[6]. Even though the whole system is referred to as multi-streaming, this finding has prompted many researchers to tacitly or explicitly adopt dual-stream arrangements, where one stream prioritises low latency and another prioritises information.

Multi-streaming and multi-view teleoperation interfaces have been shown to have advantages, but there are still a number of drawbacks. Many current research approach the underlying streaming pipeline as a secondary concern, concentrating instead on user-interface ideas or perceptual performance. Furthermore, the majority of systems are tested on powerful ground-based hardware or in controlled laboratory environments, raising concerns about their viability on embedded UAV platforms with constrained computational capabilities.

Overall, the research unequivocally demonstrates that multi-streaming in conjunction with multi-view teleoperation interfaces enhances work performance, operational safety, and situational awareness. Nevertheless, complete, end-to-end solutions that simultaneously take into account interface design, real-time transmission restrictions, and stream prioritisation are still lacking. Few studies, in particular, offer repeatable implementations that show how multi-stream video pipelines may be effectively implemented on embedded UAV hardware while enabling useful layouts like side-by-side or picture-in-picture views. This gap drives the objective of this thesis, which is to design and develop a real-time, lightweight dual-stream system as a scalable and useful example of multi-streaming.

3.3 UAV Video Streaming Protocols and Frameworks

For many UAV applications, such as remote inspection, surveillance, autonomous navigation, and search and rescue operations, real-time video transmission is a basic prerequisite. A range of video streaming protocols and communication frameworks have been investigated to allow low-latency, dependable video distribution via wireless links, since UAV systems increasingly depend on live visual feedback for both onboard decision-making and human supervision. In terms of latency, resilience, scalability, and flexibility to shifting network circumstances, the protocol selection has a significant impact on system performance.

RTP-based streaming was frequently used in early UAV video systems, frequently in conjunction with RTSP for session management. Operating over UDP, RTP (Real-Time Transport Protocol) prioritises quick packet delivery above dependability and was created especially for real-time multimedia distribution. According to Schulzrinne et al., RTP is a low-latency, lightweight transport method appropriate for audio and video streams when sporadic packet loss is tolerated[40]. RTP is frequently utilised in UAV applications because TCP-based protocols' retransmission delays can seriously impair situational awareness and real-time control. RTP is commonly used in conjunction with RTSP (Real-Time Streaming Protocol) to manage streaming sessions, enabling a client to initiate, terminate, or modify the video feed. Since many commercial UAV systems use RTSP endpoints to expose their camera feeds, integrating them with ground control software is not too difficult. Although RTP/RTSP-based systems offer minimal latency, they don't have built-in ways to adjust to changing network circumstances, which are typical in mobile UAV situations.

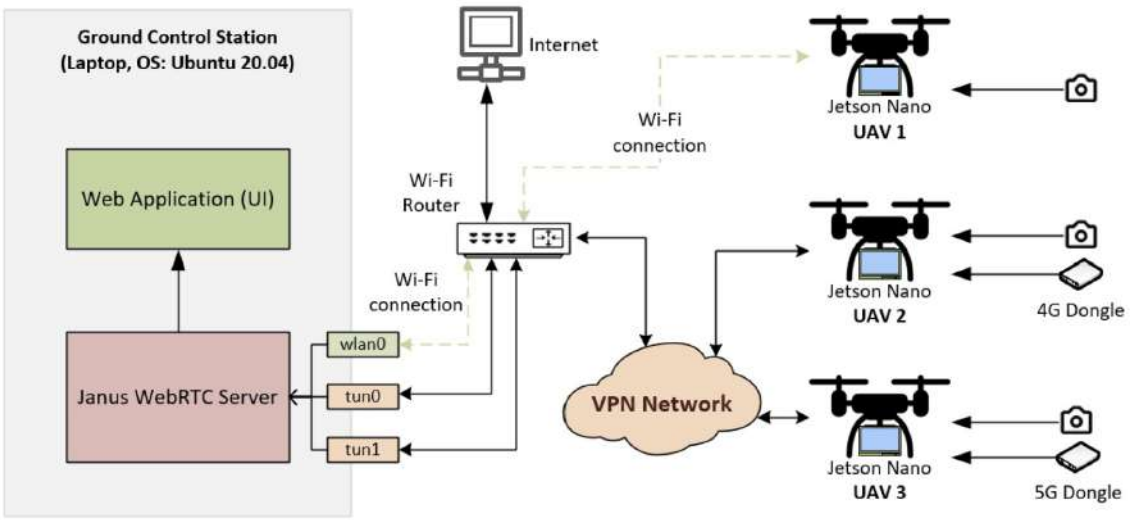

Figure 3.3: RTP/RTSP-based UAV video streaming architecture (adapted from [37])

Even though these protocols offer the fundamental components for UAV video streaming, using them to multi-stream or multi-camera UAV systems still presents a number of difficulties. The simultaneous transmission of numerous video streams raises bandwidth needs and puts more strain on congestion management systems[6]. Research on UAV video communication highlights that whereas inspection or analysis streams may accept increased latency in exchange for improved quality, navigation-critical streams must continuously maintain low latency[23]. Nevertheless, the majority of current protocol-level research is on single-stream situations and does not specifically address how several streams ought to be prioritised, synchronised, or set up inside a single framework.

The research also notes that UAV video streaming systems' deployment difficulty is a drawback. Reproduction and deployment on embedded UAV systems are challenging since many experimental configurations rely on closely connected software stacks or unique implementations. Although they are not yet extensively standardised in UAV research, recent movements towards containerisation and modular streaming pipelines seek to overcome this problem. All things considered, current research shows solutions that are straightforward and have minimal latency but are not very flexible. Despite these developments, there is currently a dearth of research that methodically investigates how these protocols might be applied to enable real-time multi-stream UAV video transmission, especially on embedded systems with low resources.

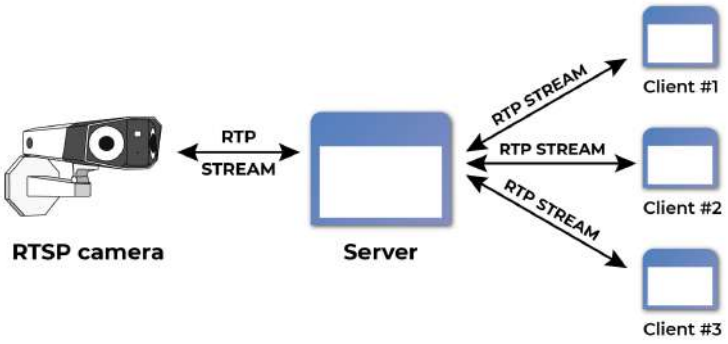

Figure 3.4: Multiple UAV video feeds are received by a browser-based ground control station using WebRTC[23]

3.4 Implementation Platforms and Frameworks for UAV Video Streaming

A number of works have been carried out to explore the practical implementation issues of real-time video streaming systems for UAV, in terms of multimedia frameworks, onboard computing platforms, and deployment strategies, under strict latency and resource constraints. When the UAV video transmission moved from desktop-based experiments toward embedded and real-world deployments, one of the critical factors that affected system performance and reliability was determined by the implementation platform choice.

For example, most of the existing streaming prototypes on UAVs use GStreamer as the base multimedia infrastructure. GStreamer has been extensively used for forming end-to-end video streaming pipelines on the UAV, which involve camera frame grabbing, video encoding using either H.264 or H.265, and streaming of the video using RTP or RTSP to a ground station[13]. The GStreamer Application Development Manual contains information on how such video streaming pipelines can be formed using modular elements, which allow detailed control on buffering, latency, and synchronization[4]. GStreamer has also been extensively used on most UAV systems together with the gst-rtsp-server library for forming RTSP servers on the UAV or on the edge device, which enable dynamic connections of ground stations for streaming video.

Some applied research papers and engineering documents elaborate on extending GStreamer pipelines from streaming to more complex operations. In these research papers, the authors discuss the use of branching elements in order to enable live video streaming and simultaneously record or transmit the video feed to an onboard processing module from a single camera source. However, in these research papers, the authors conclude that real-time streaming is affected by buffer sizes while work-

ing with multiple streaming outputs. Consequently, in the vast majority of research papers involving GStreamer on UAVs, the research is only conducted on pipelines involving a single streaming operation.

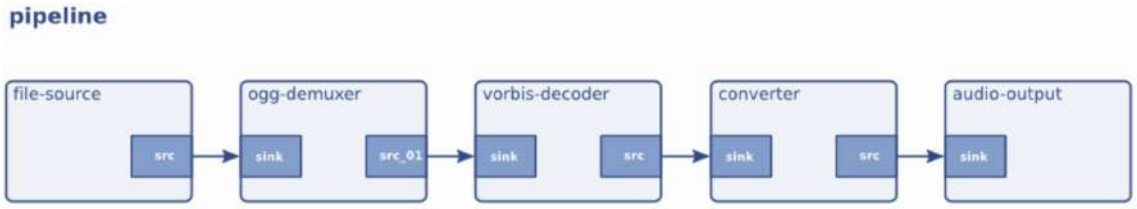

Figure 3.5: Conceptual Gstreamer Pipeline[11]

As the need for processing in UAV platforms has been on the rise, existing literature has focused prominently on embedded GPU platforms, especially on products developed by NVIDIA, including Jetson-series platforms[27]. For example, several papers on UAVs focus on Jetson Nano, Jetson TX2, and Jetson Xavier platforms as onboard computers for live video streaming in UAVs. The literature has demonstrated that hardware-based encoders in Jetson platforms make it possible to conduct real-time H.264 or H.265 encoding in UAVs with greatly reduced CPU overheads as compared to software-only solutions[27]. For example, reference pipelines presented in NVIDIA Jetson Linux documentation include GStreamer elements nvv4l2h264enc.

Other research has built on this by integrating video streaming and on-board perception. For example, applications developed on Jetson TX2 and Xavier boards have been shown to perform video streaming and computer vision or deep leaning tasks, such as object detection, in real time[51]. These studies illustrate that there is enough computation left over on embedded GPUs, even with video encoding, to enable inspection and monitoring UAVs.

Despite these benefits, there are consistent reports of limitations in the state of the art of embedded GPU platforms: memory bandwidth limits, thermal throttling, and a limited number of instances of each hardware encoder restrict scalability of such systems[47]. For example, various studies report that while a single stream can be reliably processed, the scaling to multiple streams often results in rapid latency increases or dropped frames unless pipelines are painstakingly optimized[33]. These findings underline that hardware acceleration is not enough in itself; pipeline design has to be done carefully for multi-s

Aside from hardware and multimedia platforms, another area of concern in more modern research on UAV systems is deployment. Some studies focus on containerization via Docker to make video streaming pipelines and their dependencies containerized. The use of containerization in UAV systems enables one to reuse the same streaming software on development PCs, ground control, and even the onboard computer with little modifications. Related studies in robotics and edge computing emphasize containerization in enabling system reproducibility. For instance, the latest systems developed using Jetson Xavier platforms integrate GStreamer or

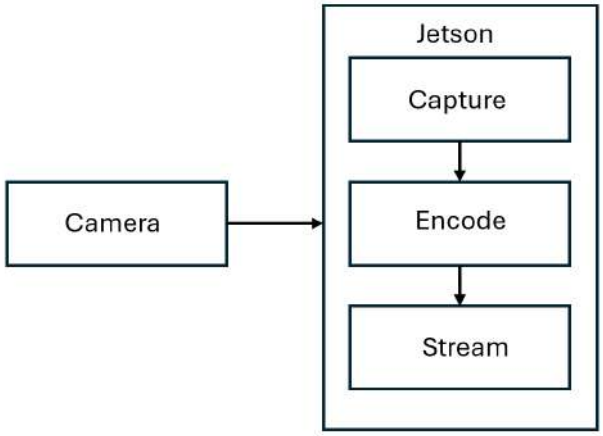

Figure 3.6: Conceptual Embedded GPU based UAV Processing Architecture

NVIDIA DeepStream pipelines and a Docker deployment framework[39]. In these studies, video streaming and analytics functionality are provided within containers and managed using Docker Compose, thus facilitating flexible experimentation with various streaming protocols including RTSP, RTMP, and WebRTC[6].

In parallel with open-source solutions, another category of alternatives and proprietary solutions has been identified in the literature. Commercial UAVs use closed hardware-software stacks that are optimized for specific sensor components and communication links. Such solutions may present excellent real-world performance but fall short in terms of transparency, flexibility, and research purposes[34]. The review of UAV communication systems in the literature has shown that in closed solutions, control over encoding and streaming is limited in low-level research.

In general, existing literature shows that GStreamer-based pipelines, embedded GPU platforms like NVIDIA Jetson devices, and containerized deployment strategies are at the heart of most modern UAV video streaming systems. However, previous studies predominantly evaluate these components in isolation or strive for single-stream transmission. Relatively few works propose integrated, end-to-end implementations that marry multi-stream GStreamer pipelines, hardware-accelerated embedded platforms, and portable container-based deployment under realistic UAV constraints. Specifically, few past studies analyze how multiple streams with different latency and quality requirements could be managed simultaneously on embedded hardware. This gap provides motivation for the implementation approach pursued in this thesis.

3.5 Bandwidth-Efficient, Selective, and Panoramic Streaming Approaches for UAVs

As a growing number of UAV applications require high-resolution visual information, it has been becoming increasingly unfeasible to transmit live, full-resolution video. To overcome these challenges, bandwidth-efficient streaming techniques, targeting the reduction of the amount of data that need to be transmitting while maintaining the ROI in the video, have received considerable attention. The two promising areas of research in this region include ROI-based streaming and stitching of panoramic video.

There are also some studies on selective streaming techniques where only the most relevant portions of the video frame need to be streamed at high resolutions. For example, Meuel et al. suggested the use of "region-of-interest encoding techniques" on video sequences captured from aerial perspectives, whereby models of the landscape are employed to detect regions of interest which should be processed at a more detailed level than the background[26]. Their experiment demonstrated that the "ROI-based encoding technique can lead to substantial savings on the amount of transmitted data without compromising the relevant detection detail needed on buildings or roads." Selective streaming techniques, as demonstrated by their experiment, find relevance in aerial imagery where "large areas of the scene are redundant."

More recent work continues to extend the above idea and covers the integration of ROI selection with adaptive video coding. Lee et al. proposed a multi-level UAV detection and tracking approach, wherein the ROIs are identified using the detected objects and are then encoded using higher bit rates, while the other regions are more aggressively compressed[24]. The system adjusts the levels of the encoding parameters based on the scene's contents and achieves better channel bandwidth without compromising the detection accuracy.

On one hand, selective streaming has proven to successfully cut down on bandwidth usage. There have been some drawbacks to selectively streaming that have been pointed out in earlier studies. When ROI-based techniques are used, they are largely dependent on the detection and comprehension process, which again is a process-consuming task and fails under certain conditions like occlusion and lighting. Selectively streaming might overlook human perception tasks. Another approach investigated in the literature is stitching-based panoramic video streaming, where multiple camera views are combined into a wide field video stream. Instead of transmitting multiple video feeds, panoramic systems attempt to unify them all into one. For instance, Ullah et al. designed a cost-effective multi-camera system for a UAV capable of streaming 360-degree mono and stereo panoramas in real-time[46]. The system proved the potential of panoramic stitching in offering all-round awareness using a single video stream, which would be very helpful for surveillance purposes. Likewise, Cui et al. presented a panorama generation strategy within multi-camera systems and used an object-distance estimation technique to eliminate geometric distortion[9]. The strategy demonstrated the effectiveness of proper geometric mod-

eling on the continuity and stitching artifacts of the resultant panorama.

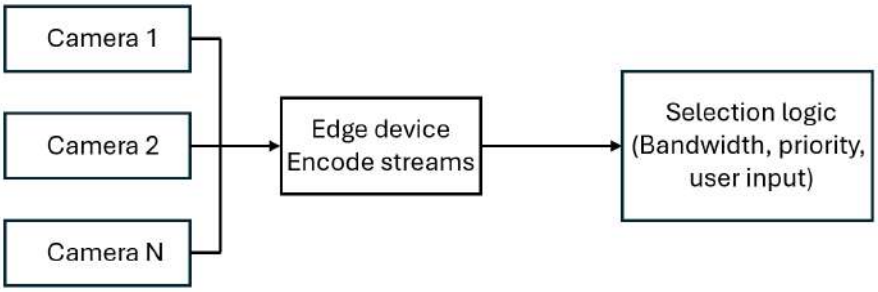

Figure 3.7: Conceptual Selective Streaming Architecture

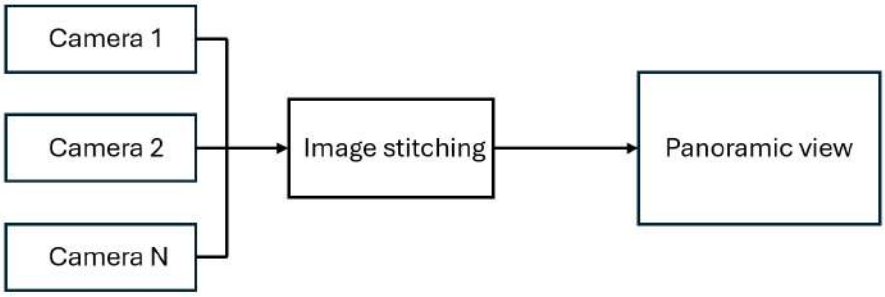

Figure 3.8: Conceptual Panoramic Streaming or Stitching Architecture

Panoramic systems minimize the number of streams that need to be transmitted and offer a global field of view; however, they also pose new challenges to the system. According to various studies on panoramic systems, stitching is a process that requires accurate camera calibration and synchronization and consumes more resources, which may not be ideal for lightweight drones. Additionally, stitching can cause visual artifacts due to fast movements of drones and objects in a scenario that may impact human observation and computer-assisted interpretation. In general, as made clear by the existing literature, methods for more bandwidth-efficient and panoramic streaming can help decrease the communication burden in UAV systems. Still, applying them in real-time video processing systems for drone media, which involve multiple video streams, has been very limited so far. In most related studies, the goal was optimized bandwidth or optimized coverage, rather than exploring how they can also be effectively integrated with real-time navigation or human-controllable interfaces.

3.6 Current Challenges

Although considerable advancements have been made in UAV video streaming, multi-camera configurations, as well as real-time transmission frameworks, some

central challenges still persist. The challenges have often manifested in literature and become even more important when multi-stream or dual-stream architectures are realized on the embedded UAV platform. The following subsections highlight the most significant challenges listed earlier.

3.6.1 Bandwidth Variability and Network Stability

One of the most demanding aspects of video streaming using a UAV is the unpredictable nature of wireless communication channels. Drones communicate via Wi-Fi networks, cellular networks (4G and/or 5G), or a custom radio link that is prone to distance, obstacles, and movement as the drones fly, causing the available bandwidth to vary rapidly. There are many researches available which point out that despite adequate average bandwidth, dramatic changes can still lead to the freezing of video, loss of packets, or blocking of streams. This issue becomes a serious problem in M/S systems, as several video signals share the same uplink for transmission. Although adaptive protocols such as WebRTC can take care of some of those problems, it has not been possible to eliminate all such problems created due to unstable wireless communication.

3.6.2 End-to-End Latency Constraints

End-to-end latency is an overall delay that occurs between the time a video frame is captured by the UAV and the time it is displayed on the ground control station. It is a latency that should ideally be as low as possible if the UAV is to be safely controlled remotely and if there is to be a high level of situational awareness. It has been observed from previous works that the factor of latency is impacted by several aspects in video streaming such as video encoding, buffering, transmission over the network, or decoding. The issue of latency further becomes complex in multi-stream applications where each stream can have unique requirements. It is noted that the stream responsible for navigation needs to be highly low-latency. However, in the inspection stream, slightly increased latency can be acceptable if the visual output improves.

3.6.3 Embedded Compute and Power Limitations

UAVs are restricted to stringent limits on weight, power consumption, and onboard computation. Embedded platforms, like Jetson-class devices, are equipped with hardware acceleration but still have limited processing capacity and thermal limits. Multiple running camera pipelines, encoders, networking stacks, and optional perception algorithms quickly deplete the available resources. As shown by prior work, while single-stream video encoding is generally feasible on embedded platforms, scaling to multiple streams often leads to increased CPU/GPU load, memory pressure, and thermal throttling. Power consumption is also a concern, as high computational

load directly reduces flight time. These constraints make the deployment of complex multi-stream systems difficult without careful optimization and prioritization.

3.6.4 Pipeline Complexity and Synchronization

Further, with advancements in UAV video systems from single camera to multi-camera and multi-stream processing, the overall pipelines have increased in complexity. This may include complex pipelines having multiple cameras, encoders, branched paths, network sinks, and optional analytics nodes. Real-time control and effective management of such complex pipelines are no trivial tasks. One problem noticed and reported in the literature involves synchronization. For multi-camera setups, synchronization among different streams must be maintained so that different views accurately describe the same state. This problem causes synchronization losses, thereby creating problems in using views from different cameras, making it difficult to rely on such a system. Further, complex pipelines are difficult to debug, thus providing room for potential buffer overflow, frame drops, and unsought latency variations.

3.6.5 Multi-View Interface Usability

Although multi-stream and multi-view systems supply additional information to vision, they cause difficulties regarding human usability. There are observations according to which additional video streams do not necessarily contribute to a higher performance of operators. It has been noted that a cluttered user interface can cause difficulties regarding concentration on relevant information. A picture-in-picture arrangement and a side-by-side layout each exist with their specific advantages and disadvantages. The picture-in-picture layout assists in maintaining a higher-level overview regarding information but hides crucial information. Side-by-side displays require a wider screen area and a permanent shift between both sides. It is difficult to decide which layout is to be preferred based on specific missions and available screen equipment.

4 Concept and Methodology

This chapter describes the conceptual designs and methodologies involved in designing a real-time multiple stream video transmitting system for autonomous unmanned aerial vehicles (UAVs). Although the former chapters outlined the background technologies and discussed the current status of the field, this chapter will pinpoint a proposed solution and a rationale for a set of designs. The discussion and explanation of this chapter are based on clarity, replicability, and a connection with realistic UAV usage constraints. Visual perception becomes the most critical source of information for autonomous UAVs to support navigation, inspection, monitoring, and human supervision. For instance, live video streams form a basis for onboard autonomy and represent the primary feedback channel for human operators. However, most existing UAV systems still operate with single-stream video transmission, forcing a trade-off between global situational awareness and fine-grained inspection detail[36]. It has been shown that multi-camera and multi-view systems indeed have larger human performance gains in inspection and teleoperation tasks. Results of this nature motivate the requirement to study video transmission architectures that can support multi-stream video transmission for each complementary visual perspective simultaneously.

From various multi-stream strategies proposed in current literature, including dual streaming, selective streaming, and stitching-based panoramic streamings, each has its own merits and drawbacks. Although selective streaming can conserve bandwidth and in many cases relies on high accuracy of region-of-interest identification and dynamic quality adjustment, it makes systems more complicated and challenging for implementation[26]. Stitching-based strategies offer a unified wider FoV; however, they add extra complexity in terms of processing requirement and alignment issues and hence may not be ideal for real-time implementation in Embedded Mini-UAVs[9]. Dual-streaming strategies involve simultaneous transmission of two separate video streams and offer a good compromise on system complexity and richness of visual experience, hence this chapter proposes dual-streaming as a premise based on an overview + details design paradigm where one stream is for navigation and situation awareness and another for high-detail inspection data.

In terms of system design, the proposed concept is specifically aimed at realistic UAV hardware and software environments. In terms of computing platforms, the NVIDIA Jetson line of platforms is popular in UAV research because of their capability to accelerate video encoding[31]. In relation to software design, multimedia platforms such as GStreamer are usually utilized to implement real-time video pipelines. Furthermore, communication platforms such as WebRTC are gaining attention for their capability to execute real-time video transfer in a browser without

incurring high latency[13]. These platforms serve as the building blocks for the proposed concept.

For the purpose of easy mobility and reproducibility of the system, the system concept also involves Docker containerization. Docker containerization is a technique that has been widely adopted in edge and distributed computing and will be implemented in the system architecture for encapsulating the components of the stream into isolated services for future integration with the drone platform[34]. Besides the existing implementation, the chapter also discusses future-proofness, especially the inclusion of thermal imaging sensors such as FLIR cameras. In particular, thermal cameras used for inspection, surveillance, and search and rescue applications are commonly employed; however, their implementation also brings about increased complexity due to their own specific format, frame rate, and processing pipeline[44].In summary, this chapter bridges the gap between theoretical background and practical implementation by presenting a well-motivated system concept and a structured methodology for developing a real-time dual-stream video transmission system for autonomous UAVs. The following sections detail the problem definition, design choices, architectural concepts, and methodological steps that guide the development and evaluation of the proposed system.

4.1 Design Requirements and Constraints

Real-time video transmission solutions for autonomous UAVs are more complex than other multimedia transmission systems. Firstly, the dynamic environments in which the UAV operates are much more complex than the typical environments in which other multimedia is currently delivered. UAV communications are always wireless. Another factor is the restricted computational capacity and battery power of the host platform. Finally, the video is a significant Safety Critical component of the overall system.

Therefore, because of these factors, the design for the proposed system must take into consideration not only the objectives, but also a number of very specific requirements and constraints. The requirements specify what must be accomplished by the design related to specific objectives, and the constraints inform how the design must operate within a set of physical limitations. For multi-stream video transmission, latency, bandwidth variability, hardware limitations, and human operators are very much related to each other, and if any one aspect thereof is neglected, it can tend towards creating an unsteady video stream process, poor human operator experience, or even unsafe operation of the UAV. In this section, the primary design considerations that were taken into account in the present effort will be discussed. These considerations will lay the basis for designing a dual-stream system architecture that will be used in coming sections. A quick overview of the rrequirements are listed below:

- **Real-Time and Low-Latency Requirements**

  - The end-to-end latency must be kept within a *few hundred milliseconds* to ensure safe navigation and stable operator control.
  - The navigation stream must prioritize minimum delay, even if it results in a slight reduction in video quality.
  - Quality-intensive streams must not be treated the same as latency-critical navigation streams.

- **Network Bandwidth Variability and Unpredictability**
  - UAV wireless connections experience bandwidth variation, jitter, packet loss, and intermittent link loss during flight.
  - Multi-stream transmission must remain functional under limited and unstable throughput.
  - Streams must support autonomous prioritization to ensure the critical navigation feed remains available even under poor network conditions.

- **UAV Hardware Limitations**
  - Embedded UAV platforms (e.g., Jetson Nano) are constrained in terms of CPU/GPU resources, memory availability, and power consumption.
  - Running multiple encoders along with transmission tasks can overload the system, leading to frame drops and increased latency.
  - The system must follow a lightweight and hardware-friendly design, preferably using hardware-accelerated encoding.

- **Operator Interaction Requirements**
  - The operator must be able to perceive and respond to live video feeds promptly for real-time decision-making.
  - Multi-stream viewing should be presented in a user-friendly interface without overwhelming the operator.
  - The system should provide independent control of streams (start/stop streaming and recording) and support an adaptive viewing layout.

4.1.1 Real-Time and Low-Latency Requirements

Real-time ability is one of the major requirements for the transmission of UAV videos, particularly those related to navigation, control, and safety monitoring. Latency is described as the time elapsed between capturing a frame of a video on the UAV and its reception on the ground station. In control systems, large latency values pose a threat to the ability to respond to changing surroundings and could be a cause for mission failure. Research on remote control of drones has shown that an end-to-end latency of a few hundred milliseconds and higher can impair the accuracy of control and awareness of the environment, especially in a crowded

environment[10]. Because of this, it has been deemed necessary for UAV navigation videos to emphasize low latency with possible sacrifice of video quality.

In multi-stream systems, not all video streams have the same latency requirements: While navigation streams require the least delay, inspection or analytical streams can tolerate higher latency in favor of better image quality. There is a need to provide such differentiation for multi-stream systems so that control safety is not compromised. Previous efforts on real-time communication stress that latency-sensitive streams should be separated from bandwidth-intensive or quality-intensive streams to ensure predictable behavior[40].

4.1.2 Bandwidth and Network Variability

Contrary to wired networks, UAVs communicate through wireless networks that can experience varying levels of quality within a short period, depending on distance, mobility, interference, and environmental barriers. Bandwidth can also experience varying levels within a short period when in flight, causing losses, jitter, or connectivity losses. Such variability creates one of the greatest challenges in transmitting video in real-time when several videos are being communicated. Studies on UAV communication networks have shown that bandwidth variability is one of the prominent constraints in aerial video streaming. In multi-stream environments, this issue is magnified as two or more video streams share access to limited wireless resources. In environments where each stream is given priority equally, an abrupt fall in bandwidth can lead to an increase in latency or stream termination.

In view of this challenge, it is necessary for the system to have independent handling and prioritization of streams. This will ensure that more important streams, such as video during navigation, continue to function even when network conditions are poor. The study on low latency in UAV real-time streaming sourced from research emphasizes network bitrate stability and independence of streams for achieving network stability during times of variance in network conditions[6].

4.1.3 Embedded UAV Hardware Constraints

The platforms of UAV systems place tight constraints on size, weight, power, and computational requirements. The available specialized platforms for computational calculations in UAV systems, for instance, Jetson platforms, allow hardware acceleration for video processing. However, the platforms have certain limitations on CPU, GPU, and memory, and energy consumption. All these factors influence the processing of multiple video streams, the complexity of encoding schemes, and the possibility of performing complex video processing operations. Real-time video encoding in a UAV is one of the busiest tasks in terms of computing. Simultaneous execution of multiple encoders, communication tasks, and other analysis tasks can easily overload the system if carefully designed. The importance of efficiency and hardware-friendly design in a UAV system with multiple cameras is recognized in literature, and this is necessary for reliable real-time performance[51].

Such conditions firmly oppose strategies which involve heavy computation, such as real-time video stitching or adaptive streaming strategies. In contrast, it becomes strategically preferable to apply light and modular strategies that efficiently incorporate hardware acceleration and eschew processing when possible. This requirement is an important driving factor in adopting a dual-stream strategy.

4.1.4 Usability and Operator Interaction Requirements

In most UAV operations, particularly in inspection and surveillance activities, the human operator is still part of the control process. For this case, where a human is part of the control loop or process, the usability of the video interface is an essential requirement. Operators should be able to quickly understand the video feed and make critical decisions related to the control process.

In research into teleoperation of UAVs and distance control of robots, it was demonstrated that multiple views of data can aid task completion, as long as it is presented in an intuitive interface[36]. On the contrary, complex and nonintuitive design of interfaces tends to overload the operator and decrease effectiveness. From an interface design point of view, it implies that there are demands for proper stream segregation, independent control of streaming and recording, and convenient visualizing layout designs. The operator must be able to independently start and stop streams and recording sessions and concentrate on what is of most interest to him at any particular time or juncture.

In conclusion, the design of the proposed multi-stream video transmission system is driven by a total of four highly integrated requirements: the provision of low-latency and real-time performance, bandwidth variability resistance, suitability for implementation on an embedded system in a UAV, and human operability. Meeting the requirements on the conceptual level ensures a practical, safe, and easily expandable proposed system, forming a good basis for the dual-stream system in the next section.

4.2 Selection of Multi-Stream Strategy

The choice of a suitable multi-stream video transmission scheme is one of the key design considerations in this research. This is because the current trend in the use of modern UAVs is the adoption of multiple cameras. However, the real-time transmission of video streams from several cameras is a challenge. There exist various multi-stream video transmission schemes in the existing literature. This research will consider an analysis of the different advantages of the available multi-stream video transmission schemes.

This part offers an overview of the most common multi-stream strategies applied to the area of UAVs, analyzes their properties discussed in the context of real-time systems, and explains why the selection of the dual streaming approach was justified for the proposed system.

4.2.1 Overview of Multi-Stream Approaches

Existing literature and practical systems based on UAVs use either dual streaming, selective streaming, or stitching-based streaming methods in order to process multi-camera video streams. The method depends on video processing and transmission.

- Dual Streaming means the transmission of two non-interrelated streams of video, normally corresponding to two different cameras or two different tasks, like navigation and inspection. In Dual Streaming, each stream is compressed and streamed independently, giving the ability to be managed separately for quality, bit rate, and latency. Dual Streaming can be found in inspection drones and teleoperation robots, meaning multiple perspectives need to be viewed simultaneously.

- The main purpose of selective streaming is the reduction of bandwidth consumption through the delivery of the most pertinent video information at a high quality. This pertinence may be established through the use of region-of-interest detection, event commands, as well as operator commands. The rest of the video information may be delivered at a lower quality or may not be delivered at all. It is common for selective streaming to be used along with computer vision algorithms.

- Stitching-based Streaming technology combines the views from various cameras and then stitches them together to create one wide field view video. This technology enables the creation of only one video instead of sending several videos when using other streaming technologies, and it is applied in applications such as mapping and surveillance systems.

4.2.2 Comparison of Dual Streaming, Selective Streaming, and Stitching

While all three methods allow for the use of several cameras, their suitability varies much for real-time UAV operation. Selective streaming is much more appealing from a bandwidth perspective due to the fact that significant data transmission reduction is achievable by focusing on important regions only. The studies related to ROI-based aerial video coding illustrate that quite substantial bitrate savings can be achieved with this selective streaming while still preserving critical information. However, this covers additional system complexity: reliable detection and tracking of ROIs are needed, and errors in the process may result in missing or degraded visual information[26]. Moreover, selective streaming leads to less predictable system behavior, which may be problematic in at least safety-critical UAV applications.

Stitching techniques give a single wide or panorama view through integrating multiple camera feeds. Evidence from panorama generation multi-camera literature indicates that stitching enhances overall situational awareness. However, real-time stitching demands high accuracy in camera calibration and processing, contributing

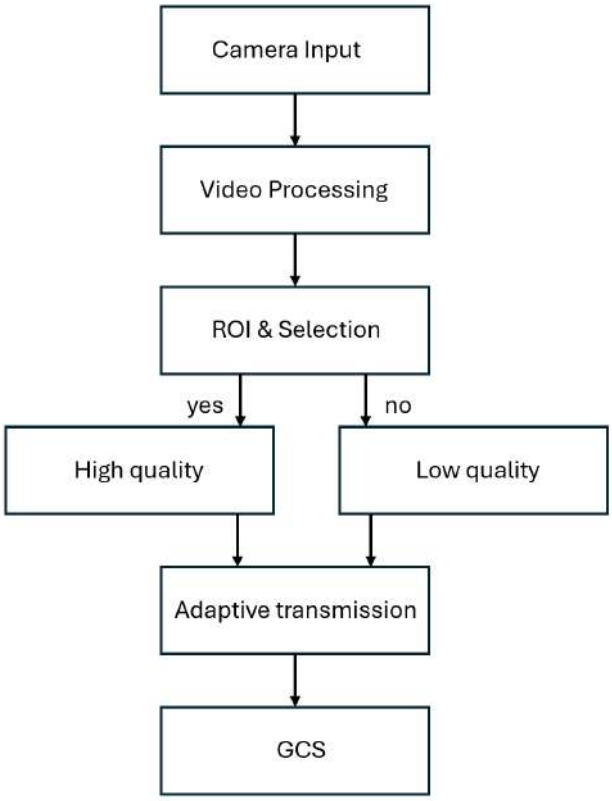

Figure 4.1: Conceptual Selective streaming architecture

to higher latency and instability in mini-UAVs[46]. A further consideration is that stitching inherently reduces resolution per camera, thereby being less applicable to inspection applications.

Dual streaming offers a good compromise between these two options. In dual streaming, two separate streams are transmitted, thus eliminating the complexities of selective streaming, which uses an algorithm to stitch the streams. Additionally, by transmitting two streams, the visual details are not lost, which would be the case if selective streaming was adopted. Studies on operating multi-camera drones have shown that offering the operator multiple views enhances situational awareness without adding complexities to the system[36].

let us now compare the different techniques with the essential criteria that has to be achieved in the approach selection(These data and scores are prepared by studying the aspects and written manually for better understanding). For instance, on a scoring scheme of 1 to 3 with 3 being the highest and 1 being the lowest and mark all the features accordingly, we would be able to get the below mentioned table. Clear differences between these three approaches can be observed when evaluated against system-level criteria relevant to real-time UAV applications. Indeed, selective streaming obtains high scores with respect to bandwidth efficiency and latency performance since it streams only selected regions or streams. Its overall

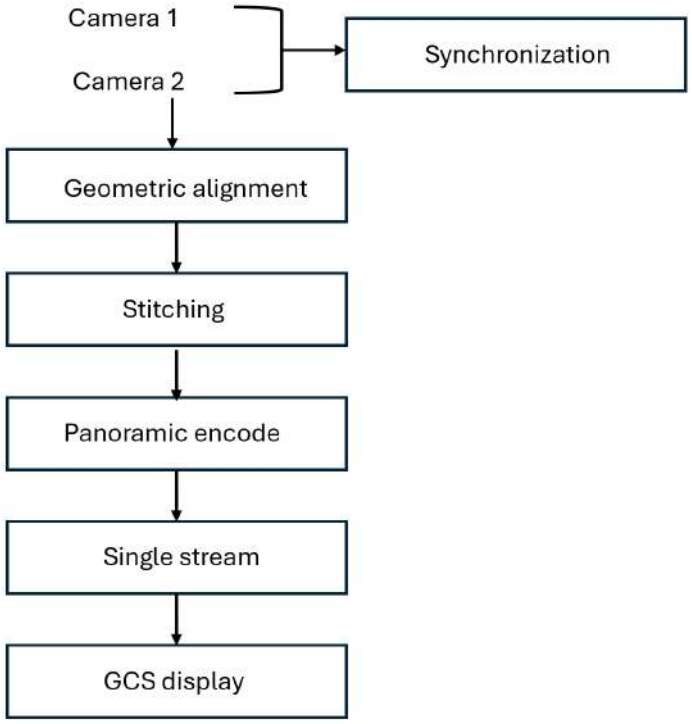

Figure 4.2: Conceptual Stitching streaming architecture

score is decreased due to low ease of implementation, limited scalability, and weak multi-camera support. These emerge due to the requirement for a reliable region-of-interest detection and complex stream management. Stitching-based approaches perform better in scalability and video quality because of the extensive and common field of view offered. Nonetheless, its performance is poor in terms of bandwidth utilization, latency, and resource requirements. The system requirement of computing the stitching in real-time and camera calibration makes this method not fully compatible for embedded platforms like the Jetson Nano. Dual-streaming has the best total score of the three approaches. Its capabilities include high performance in real-time, high video quality, excellent support for multiple cameras, and ease of implementation that is also the simplest of all approaches. Though its bandwidth is slightly worse compared to selective streaming, this is acceptable given the level of robustness, predictability, and simplicity incorporated in system design. Comaparing all these, we opt for Dual streaming.

4.2.3 Justification for Choosing Dual Streaming

Selection of an appropriate multi-stream strategy is a very important design decision, since the latter directly influences the performance and complexity of the system and its feasibility on embedded UAV platforms. From the analysis of the design

Criterion	Selective Streaming	Stitching	Dual Streaming
Real-Time Capability	3	2	3
Bandwidth Efficiency	3	1	2
Resource Utilization (Jetson Nano)	3	1	2
Ease of Implementation	1	1	3
Compatibility with GStreamer	2	2	3
Scalability	1	3	2
Latency Performance	3	1	2
Video Quality	2	3	3
Multi-Camera Support	1	2	3
Total Score	**19**	**16**	**23**

Table 4.1: Comparison of Streaming Methods

requirements presented in the previous section and the comparison in Table 4.1, dual streaming emerges as the most suitable approach for real-time multi-stream video transmission in autonomous UAVs. Table 4.1 offers a comparison between selective streaming, stitching, and dual streaming based on various parameters in qualitative-quantitative form that are very relevant in the context of UAV. Each parameter has been measured on a three-point scale. The parameters are scored in such a way that higher values are assigned to better-suited parameters. The parameters could be ranked based on their priority in real-time performance.

Selective Streaming produces high scores in terms of bandwidth and latency, indicating its effectiveness in minimizing the volume of data being transmitted, especially through focusing on regions of interest. However, these improvements are realized at the expense of higher system complexity and ease of implementation. This low score in ease of implementation indicates the use of additional elements in the computer vision or decision component, including region-of-interest identification and adaptive encoding logic, which are less desirable in high-predictability applications, especially in the realm of safety-critical drone applications. In addition, Selective Streaming produces low scores in terms of scalability and multi-camera support. Scalability, video quality, stitching-based streaming performs well in terms of these factors since this technique offers an overall wide-field view, accomplished by stitching the inputs of multiple camera views together. Yet, Table 4.1 indicates that the scores for bandwidth, hardware resource utilization on the embedded hardware, and latency are lower for the stitching-based technique. This indicates that the technical complexity, especially the high intensity of computations, when performed on the Jetson Nano, contributes to slow performance, lag, or even latencies for the purpose of navigating the UAV.

On the other hand, dual streaming yields the highest total score of 23 out of all the tested methods, implying that dual streaming has the most balanced performance on all criteria tested. Dual streaming has strong scores on real-time support, ease of implementation, GStreamer compatibility, video quality, as well as support for multi-cameras. Essentially, the strong points of dual streaming immediately correspond to the main design requirements of the new system designed for redundancy handling. Also, compared to selective streaming, dual streaming does not require complicated content analysis. Contrarily, compared to the stitching method, dual streaming does not involve intensive computational calculations that could be a hindrance for UAV-embodied computer systems.

A central principle of the proposed dual-streaming design is independence of handling multiple video streams throughout the system. Each stream is treated separately from the point of acquisition to visualization and recording. Configuration, control, and potential failures of one stream do not directly affect the other. This independence allows methodological optimization of each stream for different requirements. For instance, the navigation stream can be optimized for latency and real-time smooth feedback, while the inspection stream can be optimized for image quality and high spatial detail. Such a separation also aligns with guidelines from the literature on teleoperation and visualization, indicating the need for adaptation of visual feedback to task-specific requirements[22].

From all these facts and data presented, Dual-streaming has thus been selected as the core technique for multi streaming in this study.

4.3 Multi-Stream Video Transmission Framework

The aim of this section is to introduce the conceptual design and rationale of the proposed multi stream video transmission system for autonomous UAVs. The proposed system is intended for the concurrent streaming of several video streams with different configurations for real-time video streaming with low latency at a ground control station, as well as independent video stream recording without compromising the video streaming process. The proposed design rationale has been triggered by the real-time requirements for operation, network dynamics, and limited computation power on embedded UAV systems. The concept design rationale focuses on a modular design paradigm, where video capturing, processing, streaming, recording, and control are decoupled. Such design modularity ensures robustness, scalability, and maintainability of the system, which are fundamental for practical applications on UAVs.

4.3.1 Conceptual System Architecture

The conceptual architecture has a three-layer logical structure consisting of the UAV (edge) layer, media routing layer, and ground station layer. These are properly engineered with a clear function for each that communicates with other layers using

standardized interfaces and protocols.

In the UAV layer, the system is based on an embedded computer processor (Jetson) interfaced to two physical cameras. The two cameras are considered as distinct video sources and can be parameterized separately with respect to resolution, FPS, and bitrate. It is thus possible to prioritize the video, like giving high parameters to the primary camera for navigation and lower parameters to another camera for surveillance. Video processing in this layer is accomplished by employing the GStreamer media framework, which is quite popular for media processing in real-time applications and also supports embedded hardware-encoded video rendering[13].

Before this transmission, some overlay operations embed this contextual data, which includes camera name, resolution, frame rate, bitrate, IP address, encoder, and timestamp, into the actual video frames. With this embedding procedure, this data can be accessed at the receiving end for analysis without necessarily synchronized issues, which can be exhibited, for example, when an individual watches this video on an offline device, and it can also be transmitted online for distant analysis.

The media routing layer is realized through MediaMTX, a real-time media server with the functionality of routing media streams between publishers and subscribers through various protocols[3]. MediaMTX operates within a Docker container environment on the Jetson for isolation from the remaining environment. Its main objective is stream separation between video production and video playback. The UAV encodes video signals into data streams transmitted through RTMP protocol to MediaMTX, while the ground station receives video signals through WebRTC protocol with assistance from the WHEP (WebRTC HTTP Egress Protocol) interface[28] for signal reception at ground level.

The ground station level is made up of a browser-based GUI. This GUI makes WebRTC connections to MediaMTX and shows the camera feeds side-by-side. WebRTC is chosen for live viewing because WebRTC is intended for low-latency communications and has congestion control, jitter buffering, and rate adaptation built-in [4]. In fact, along with video playback, there is a continuous collection process by the GUI on receiver-side performance measurements such as jitter, percent packet loss, round-trip time, and frames per second using the WebRTC getStats() API.

4.3.2 End-to-End Data Flow from Cameras to Ground Station

The end-to-end data flow process includes the transformation of the raw camera images to video at the ground station. All the cameras have the same logical process but with varied parameter values.

The first step involves capturing the frames from the cameras through the Video4Linux2 interface. The frames are then decoded into a raw video which can be further processed for analysis. This process is known as normalization; as a result, the video data is treated equally regardless of the original output by the camera. The next component involves the addition of metadata directly into the video frames. A timestamp overlay is incorporated with the date and time values, while a text over-

lay includes other camera parameters such as the camera name, resolution, frames per second, bitrate, IP address, as well as the type of encoder. This makes it feasible for each frame to have additional data related to its creation process, for example, UAV missions whose videos are likely interpreted when public log files are inaccessible.

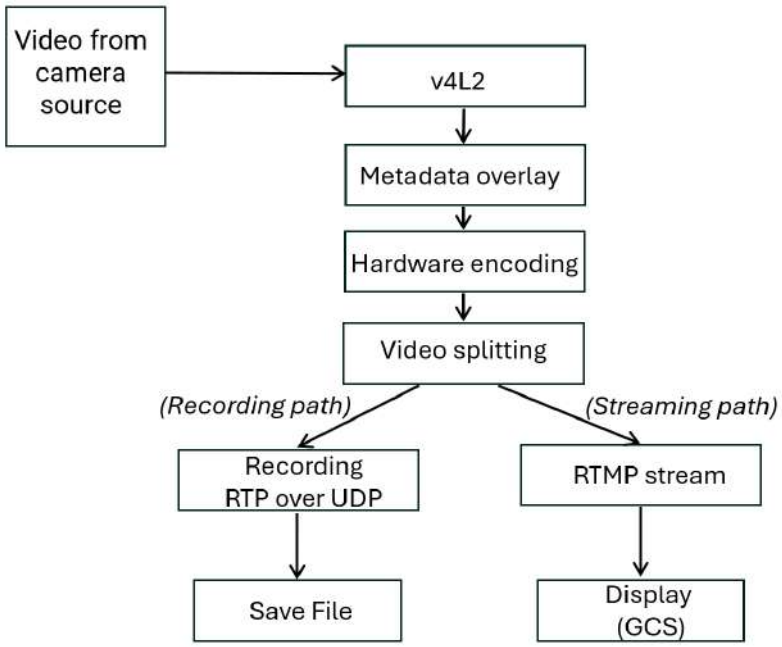

Figure 4.3: End to end data flow from camera to GCS

Following metadata insertion, the video is encoded with the H.264 standard using hardware acceleration. H.264 is chosen for its broad compatibility and efficient compression, making it suitable for applications with bandwidth constraints and/or that need to be performed in real time[18]. The encoding parameters are configured independently for each camera, thus enabling differentiated stream quality. Once encoded, the video stream is duplicated into two branches using a GStreamer tee element. One branch handles live streaming and the other handles recording. The live streaming branch takes the encoded video and sends it as RTMP to MediaMTX, which in turn makes it available for WebRTC consumption. The recording branch packetizes the encoded video into RTP and sends it over UDP to a local port, where it can be recorded without interfering with live streaming.

On the ground station side, the browser retrieves the streams through WebRTC with the use of the WHEP interface. WebRTC is capable of managing the necessary changes to the wireplay process depending on the jitter or the bandwidth of the stream. The browser will render the video signal for the operator to view the statistics on the reception side.

4.3.3 Separation of Streaming and Recording Functionality

Methodologically speaking, one of the main choices made in the proposed solution framework relates to the clear distinction made between the functionalities related to streaming and recording. Rather than recording the stream from the WebRTC stream, the recording takes place from an RTP stream in the sender side. Separation of the streaming and recording branches helps the system prevent any impact of recording delays on live streaming. A recording process may include disk I/O operations as well as multiplexing, possibly resulting in some delays, especially if blocking is involved. In the proposed solution, in case some delays occur in the recording branch, the live streaming process will not be affected because both branches act as independent processes after the stage of splitting the stream into two branches. There are individual queues in every branch, and it is a good practice in the development of GStreamer-based solutions[13].

Recording is accomplished through a remuxing process, in which the H.264-encoded and RTP packetized stream is repackaged into an MP4 file. This requires little computational complexity and thereby ensures the best possible encoded video quality. As mentioned, since the metadata information is overlaid on the video streams before they are encoded, the recorded files contain the same metadata as those from the live stream. Control over the recording is implemented via a lightweight control plane that is based on HTTP. A simple CGI script is used to control the recording of each of the cameras separately. This approach keeps its control plane separate from the media plane. This separation of concerns in media transport and signaling follows best practices when designing a real-time multimedia system[40].

4.4 Dual Stream Design and Functional Roles

This section explains the conceptual reasoning and methodological design of the dual-stream strategy in the proposed research. This dual-stream strategy allows the proposed system to concurrently fulfill its operation requirements in terms of latency requirements and its inspection requirements in terms of visual inspection quality. This has been achieved by ensuring that the proposed system uses two video streams for navigation and inspection that are complementary in nature by making use of the two cameras.

The purpose of the navigation stream is to enable real-time control of UAVs. The main function of the inspection stream is to increase the awareness and to monitor special areas. Both the streams are generated by a specialized camera with different frame rate and resolution. These streams undergo processing by a GStreamer pipeline specially designed for real-time processing of the streams with the use of hardware-assisted H.264 compression. During the processing of the stream, the actual metadata of the stream, including timestamps, resolution, frame rate, bit rate, as well as IP addresses, are directly encoded within the video stream frames. This

helps ensure that whenever the ground station operator is viewing the stream online, he or she is constantly assured of the ability to verify the actual stream quality. After the stream undergoes successful compression as a result of the H.264 processing, the encoded stream is transmitted to the media routing server for the purpose of streaming the stream to the ground station through WebRTC. The stream received at the ground station is displayed within a web browser with additional statistics related to the stream reception capabilities of the ground station operator. Either of these streams can be visualized later for the respective purposes like inspection and navigation. Moreover, the stream used for inspection can also be recorded for later inspections and decision making. Recording is done by making use of RTP without affecting the live stream facility.

The seperation of both the streaming and recording functionality is the core principle of this methodology. In the GStreamer pipeline, the output of each camera is duplicated using the tee element. This enables the camera output to be processed for live transmission and recording without incurring the overhead of re-encoding the video. It is essential to note that the two branches of the tee element in the GStreamer pipeline communicate through the use of queue elements to avoid backpressure and transmission delays in the system. This is an important design principle in the implementation of GStreamer in a real-time system[13]. Recording achieved by implementing a remuxing process that re-packages the H264 stream in an RTP protocol into an MP4 container without actually decoding and re-encoding it, thereby offloading any significant computing involved in this process and preventing this function from impacting live streaming on the device or network being streamed on. Additionally, this distinct service for live streaming and recording allows for dynamic control for starting and terminating recordings via control interfaces that utilize HTTP. That allows the system to scale naturally with independent handling of multiple streams. Additional cameras or alternative stream profiles can be easily introduced without redesigning the entire architecture. Moreover, failures or performance degradation in one stream do not propagate to the other, which is important for safety-critical UAV operations.

4.5 Ground Station and User Interaction Concept

The ground station is the main interaction point between the human operator and the UAV video transmission system. The proposed system considers a light and web-based interface that leverages real-time visualization of several video streams, monitoring of their performance, and control of the recording functionality directly. Simplicity, ease of access, and responsiveness are considered in the design to ensure that the system is operable through standard web browsers with no dependency on specialized software or hardware. Conceptually, the ground station is designed to separate the concerns of visualization, monitoring, and control while keeping a consistent user interface. The separation will enable intuitive operation while preserving flexibility and scalability for future extensions.

A web-based visualization technique is used at the ground station where the live video feed is viewed through a browser using WebRTC. WebRTC helps avoid the need for software applications on the client side, allowing the service to be used across different systems with ease. WebRTC was chosen for its low-latency capabilities, as it has built-in congestion control mechanisms, along with the advantage of being widely supported by different browser systems, making it ideal for real-time UAV video services[1].

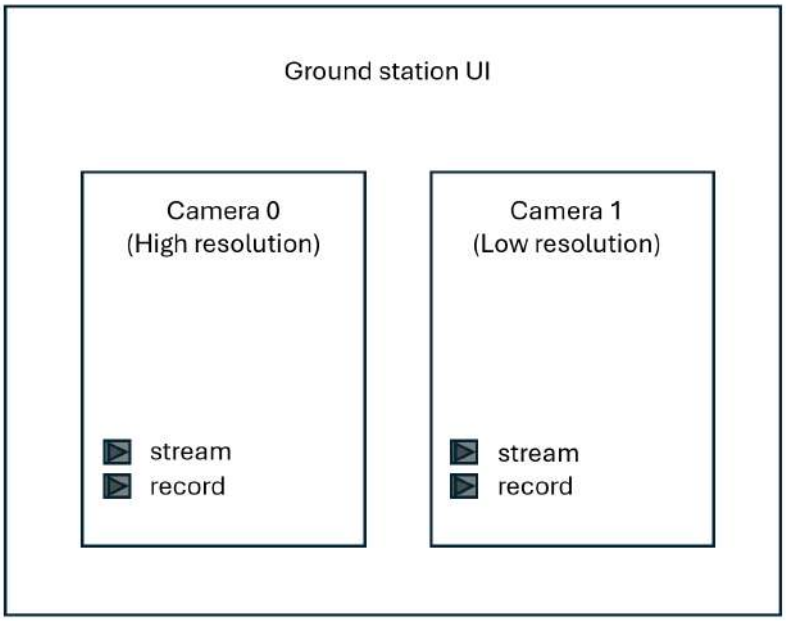

Figure 4.4: Dual Stream Visualization layout at the received end

It is ensured in the visualization arrangement of the ground station such that multiple camera videos could be viewed at the same time without compromising upon clarity and functionality. It has been taken care of in the designed interface of the ground station such that multiple videos of the camera views are represented in separate visualization containers in the designed interface in such a manner that the operator is able to view both videos at the same time. Every media file comes with a small info overlay showing reception-side statistics like the frame rate, jitter, packet loss, and round-trip time measured by the receiving side. To ensure these are distinguished from the metadata embedded into the video pictures on the sending side, the info overlays are made to be separately distinct from them. The receiving side info overlays tend to be dynamic, in contrast to the sending side info overlays, which contain static reception-side information.

One of the essential elements of the ground station concept is the capability to independently manage streaming and recording operations for every video feed. The capability to manage operations independently ensures that actions performed by the operator to start and stop a recording, for instance, do not affect other operations such as the visualization of live video feeds. The start and stop operations

for every camera feed have been managed through a lightweight control plane based on the widely used HTTP protocol and CGI scripts in the proposed system. The start/stop operations for every camera feed have been managed through specific start and stop URLs. Whenever a start or stop recording request is made, the browser sends an HTTP request to the control server, which subsequently starts or stops the recording pipeline at the UAV side. This control mechanism is, by design, independent of the media streaming plane. The streaming process is uninterrupted, irrespective of the recording status, and recording processes do not need restarts or reconfiguration of the live video stream. This improves system resilience and enables the recording process to be switched on only when the need arises, thereby obviating unnecessary storage and processing overheads. The independence of stream control further improves human-factor usability. The control system operator has the flexibility to select whether to record one, both, or none of the streams depending upon the requirement of the mission. The results of control actions are feed-backed to the browser.

4.6 Evaluation Methodology

As the system is intended for real-time UAV applications, the evaluation methodology is progressive. It starts with functional validation in order to confirm that the system is behaving correctly, continues with the qualitative performance assessment from the operator's perspective, and concludes by summarizing the preparation steps required for systematic quantitative evaluation. The evaluation approach is designed intentionally to resemble realistic conditions under which a system like that may operate rather than the traditional testing of separate parts. This ensures that valid conclusions can be drawn from such an evaluation for practical UAV deployment.

4.6.1 Functional Validation Criteria

Functional validation is the first and most fundamental step in evaluating the proposed system. The goal of this stage is to verify that all system components operate correctly and interact as intended under normal operating conditions. Functional validation focuses on correctness, not performance optimization. In the implemented system, functional validation begins at the UAV side by confirming that both cameras can be accessed simultaneously and that independent GStreamer pipelines are successfully instantiated for each stream. Each pipeline is validated to ensure that camera capture, metadata overlay, encoding, and publishing are functioning without errors. The successful display of both streams at the ground station via WebRTC confirms the correctness of the end-to-end streaming path.

Another crucial part of functional validation is the need to ensure that metadata is embedded and maintained correctly along the way. In the proposed system, the metadata, consisting of timestamp, resolution, FPS, bitrate, IP address, and encoder, is embedded into the video itself through burning prior to the encoding pro-

cess. An integral part of functional validation is, therefore, confirming whether the metadata overlay is present in the live WebRTC stream as well as in the recorded file, namely the MP4 file, and whether the metadata is readable in each case. Another crucial functional requirement test is related to the independence of both streaming and recording operations. The test is conducted by starting/stopping recording on individual cameras independent of ongoing live streaming. The successful production of playable MP4 files without affecting live playback confirms independent execution of both streams. The test of control functionalities is also done by manipulating the web ground station interface. The recording control operations initiated from the browser should result in activation of related recorder pipelines on the UAV platform. Instantaneous feedback on these operations confirms proper functioning of control functionalities.

4.6.2 Qualitative Performance Assessment

Once the functional correctness is ascertained, it moves on to qualitative performance analysis. In this phase, emphasis is on how it behaves from a human user point of view. Qualitative analysis is very important, especially with respect to UAV, as it involves human perception. One of the primary elements of qualitative analysis is Latency perception. In this, the user monitors the delay between the physical movement of objects as reflected by the camera sensor from one location to another and how it is reflected at the ground station. Though it is not a numeric value analysis at this stage, if there is a lag, stuttering, or freezing of frames, it is recorded. The analysis between the navigation or inspection streams enables qualitative analysis of how different configurations impact latency perception.

Another crucial qualitative parameter involves visual clarity and stability. The inspection stream will be assessed for clarity, detail resolution, and compressibility, whereas the navigation stream shall be assessed for fluidity and temporal stability. The burns on relevant metadata further help with qualitative evaluation, which enables the operator to make associations between perceptions and configurations on a real-time basis. On the receiver side, WebRTC statistics shown in a browser window regarding jitter, packet loss, and frame rate render qualitative understanding at this stage, although these parameters are quantitative at this point, which enables qualitative analysis for detecting patterns, for example, jitters and frame rate on instabilities and heavy loading, respectively.

4.6.3 Quantitative Assessment Criteria

The objective performance metrics used in the quantitative assessment of the proposed multi-stream video transmission system are based on the objective measures of system efficiency, scalability, and suitability in real-time to operate within the context of use by UAVs. The main evaluation criteria are the CPU load, the GPU load, memory load, the encoder load, and thermal stability. The metrics were chosen to determine the efficiency of the system in terms of using the available hardware

resources in different operational conditions. To evaluate the computational overhead, CPU utilization is applied in order to ascertain the availability of enough headroom in the processing capacity to ensure other UAV activities can be added. The use of a graphics card and the use of encoder are measured to confirm the efficient application of the hardware-accelerated video encoding. Its usage of memory is monitored in order to detect buffer increase and maintain constant performance without memory exhaustion. Temperature measurements are added to ensure that the system temperatures are within safe operating range when running is executed in a long period. System-level monitoring tools are used to capture all metrics at predefined intervals and average them throughout the experiment period which will in turn guarantee reproducibility and comparable metrics across scenarios. The combination of these quantitative requirements creates an all-inclusive framework on which it is possible to assess the performance of the system and confirm that the suggested architecture meets the needs of both normal and high-load performance scenarios. This is in detail discussed in the Results and Evaluation Chapter.

5 Implementation

This chapter presents the implementation of the proposed real-time multi-stream video transmission framework for autonomous UAV systems. Although the preceding chapters introduced a conceptual design and methodology, this chapter will address how the implementation was achieved through the use of real hardware and software components. The implementation of the real-time multi-stream video transmission framework will be built upon an embedded Linux platform for video capturing and processing of multiple cameras, where media routing services will be carried out through containerized services with web-based visualization capabilities. Moreover, the proposed system will ensure a separation of concerns for real-time video transmission, recording, overlaying metadata information, as well as interacting with the developed framework for testing purposes.

5.1 Tools and Technologies

In this section, the necessary tools and tech being used for the development of the proposed system have been described. Each tool has been chosen in relation to its suitability for the processing of the real-time video feed and the embedded system. In this work, apart from the tools and platforms mentioned below, a suitable hardware setup was also created so as to achieve the final output.

5.1.1 NVIDIA Jetson Platform

The NVIDIA Jetson platform serves as the onboard computer for the video system of the UAV. This is because the NVIDIA Jetson platform offers a low-power computing solution that features a GPU-enabled media solution. Therefore, in the proposed system, the Jetson computer senses video feeds from the multi-camera implementation, adds metadata, encodes the video feeds, and transmits them to the media routing server. Perhaps the most useful functionality of the Jetson computer is the ability to encode the video feeds. This function is particularly useful in the proposed system since it enables the video feeds to be processed in real time while consuming fewer resources. The Jetson platform is also the host system for all other system components, such as GStreamer pipelines, Docker containers, and control scripts. Its hosting nature provides a unified system for managing system components and ensures integration between hardware and software system components.

5.1.2 Ubuntu Linux Environment

The Jetson platform runs its operating system, which is Ubuntu Linux. Having an environment based on the Linux operating system offers many advantages, such as reliability, many drivers available, and the benefit of many open-source multimedia libraries. Having the Ubuntu operating system enables camera devices to be accessed directly through interfaces provided by the Linux environment, which supports process control, scripting, as well as networking functionalities required for real-time processing. Using the Linux environment also helps in easily programming system start, stop, or recovery from failures, which is critical for the reliable functionality of the UAV. It is also possible to host the backend processing for the front-end user interfaces on the same platform with the aid of the Ubuntu environment.

5.1.3 Docker and Containerized Deployment

Docker is used to containerize selected system components, especially media routing server and web-based ground station interface. Containerization provides isolation between services, thus preventing changes in one component from affecting others. In the implemented system, MediaMTX and the web server run in separate Docker containers. Such a setup can ease deployment, allow reproducibility on different systems, and permits independent starting and stopping of services. Maintaining a container also increases separation, since configuration changes do not touch the host system. It also enables a clean separation of the host environment from application services through Docker, combined with automation scripts in a shell; this helps not only for development purposes but also for evaluation.

5.1.4 Gstreamer Multimedia Framework

Gstreamer is the primary multimedia engine leveraged for video capturing, processing, encoding, streaming, as well as recording. Gstreamer is a flexible pipeline-based multimedia engine capable of assembling complex multimedia graphs from a set of modulated components. In this particular implementation, Gstreamer pipelines have been employed to carry video capturing from multiple cameras, overlaying metadata on video images, hardware-accelerated video encoding, as well as broadcasting encoded streams to both live and recording streams. Additionally, the pipeline concept of Gstreamer enables reusing one encoded video stream for multiple tasks without requiring repeated processing. Gstreamer's efficiency in supporting hardware-accelerated codecs as well as real-time streaming protocols makes it ideally suited to embedded UAV systems. Even more, Gstreamer is amenable to further modifications, such as adaptive streams.

5.1.5 MediaMTX Streaming Server

MediaMTX is used as a media routing server that sits between the video producers and the video consumers. It receives video streams from the GStreamer video pipelines and delivers the video streams to the clients using appropriate video streaming protocols. In the implemented configuration, MediaMTX receives video streams from the Jetson platform as RTMP flows and provides them at the ground station using WebRTC. The approach keeps the video pipeline configuration straightforward but enables the flexibility for support at the clients. There is, however, less complexity introduced at the sender side by utilizing MediaMTX as a protocol translation manager. MediaMTX further enables this configuration to be flexible and scalable because additional support from other clients or other video streaming protocols can be supported without upgrading the video pipeline configuration.

5.1.6 Communication and Streaming Protocols

"Real time multi stream video transmission system" involves the usage of a number of protocols so as to ensure the effective functionality of the system. For this objective, as a means of avoiding the usage of a number of protocols for the different tasks involved within the system architecture, the system proposes the usage of a multi-layer protocol as a means of getting the best possible end result with regard to the parameters of minimizing latency as well as ease of implementation.

From the onboard system, encoded video streams are sent to the media routing server using the RTMP. Implemented in this system, RTMP is used as an internal transport protocol between all the video processing pipelines and the routing layer. With RTMP, publication of streams and session management is much easier. Another option for live video delivery to the ground station is WebRTC. WebRTC is specifically designed for low-latency media transmission and includes mechanisms for congestion control, jitter buffering, packet loss recovery, and adaptive playback. In order to make the realization of a WebRTC session easier, it uses the WebRTC-HTTP Egress Protocol. With WHEP, it is possible to access WebRTC streams by utilizing standard HTTP requests without relying on complex signaling servers or any custom signaling logic.

For the recording control and handling, the Hypertext Transfer Protocol (HTTP) protocol is utilized. This HTTP protocol supports the use of lightweight HTTP requests, which enable functionalities such as the start and stop recording process for individual video streams. On the network transport level, the User Datagram Protocol (UDP) protocol is utilized inside the system for the real-time packet delivery, specifically in the RTP-based recording taps. Using the combination of all the utilized protocols inside the system, the entire setup enables the separation of concerns. This includes the use of RTMP for the internal stream publishing, WebRTC (with WHEP) for the low-latency browser-based viewing, HTTP for the control signals, and finally the use of UDP for the internal real-time packet delivery.

5.1.7 Hardware setup

The hardware configuration designed for the proposed system supports real-time capture, processing, and transmission of multi-stream video in a UAV-oriented environment. The hardware configuration designed for the proposed system supports real-time capture, processing, and transmission of multi-stream video in a UAV-oriented environment. The hardware system for the proposed solution includes an NVIDIA Jetson embedded computing platform, acting as the onboard processing unit for the UAV application. The Jetson board is responsible for interacting with camera sensors, performing video processing pipelines, as well as managing the communication with the ground station. The hardware system for the proposed solution includes an NVIDIA Jetson embedded computing platform, acting as the onboard processing unit for the UAV application. The Jetson board is responsible for interacting with camera sensors, performing video processing pipelines, as well as managing the communication with the ground station. Such hardware configuration gives a practical approximation of the onboard video processing of a UAV environment and at the same time is flexible enough for experimental testing. Using off-the-shelf components also allows for reproducibility.

5.2 System Setup and Hardware Configuration

This section introduces the configuration related to hardware components for the real-time dual-stream transmission capability on the camera platform. This includes camera integration, device management, network frameworks, as well as storage arrangement parameters that form a basis for all video processing and recording functionality on the camera platform. This is crucial for ensuring functionality effectiveness as well as experiment reproducibility on the camera platform.

5.2.1 UAV Camera Setup and Interfaces

The video capture subsystem consists of standardized USB cameras directly connected to the NVIDIA Jetson platform. These cameras act as the critical visual sensors of the whole UAV system, supplying continuous video feed data throughout the running software on the aircraft. When the UAV system initializes, a scan of each camera by the Linux kernel results in their access through the Video4Linux2 (V4L2) API, which presents a standardized way of dealing with cameras. Accessing the cameras through V4L2 means it becomes possible to work with the cameras using standardized device nodes (e.g., `/dev/video0 or /dev/video1`). The cameras will be set up to run at a predefined format rails as defined by their capabilities, facilitating stable video capture and steady frame delivery. During startup, the initialization of the cameras will be checked through the availability of these devices prior to activating the video pipelines. In this way, pipelines will never fail because of unavailability or disconnection issues of the devices, improving the stability of the entire system.

5.2.2 Dual Camera Configuration and Device Management

In order to facilitate dual-stream, each camera is modeled as a separate video source with distinct configuration parameters. The configuration parameters of each physical camera are given a logical name, such as `cam0,cam1` etc. At startup, based on these parameters, a separate GStreamer pipeline is created. This way, both streams run simultaneously, independent of each other. Also, to manage these physical devices, it is necessary to check on the status of each pipeline. In case of unavailability of a camera or unexpected termination of a pipeline, it is possible to sense this trouble or failure to prevent other components from failing as well, which would not be necessary or possible in a UAV scenario, as some functionality needs to be available even if it's not ideal.

5.2.3 Network Configuration and Management

Connectivity to the network is another important element of the real-time video transmission system. The Jetson system is connected to the ground station via the local network, either via Ethernet cable connectivity or wireless connectivity. A constant IP connectivity solution is necessary for continuous video streaming transmission. IP address management is done dynamically. When the system is initialized, the software in the Jetson system automatically identifies the active IP address of the Jetson system, thereby using that IP address for video streaming transmission. All of this data is also overlayed on the video streaming transmission. Different protocols run on the same network infrastructure but have separate functionalities. Video transmission from the Jetson system to the media routing server uses the RTMP protocol, while the WebRTC protocol is used for transmission to the browser-based ground station. Commands for recording as well as system control are transmitted over the HTTP protocol.

5.3 Dual-Stream Video Pipeline Implementation

The dual-stream video pipeline represents the technical heart of this developed system. Basically, its purpose is to achieve the concurrent acquisition, processing, encoding, streaming, and recording of two independent camera streams within an embedded platform. Unlike traditional single-stream systems, this developed system has been engineered to handle each camera stream as an independent processing unit, so different parameters or functionalities can be assigned to them. The dual-stream video pipeline has been developed using the GStreamer multimedia framework, which has an architecture suitable for real-time video processing that can handle multiple independent pipelines. Both camera streams are processed in their respective GStreamer pipelines, which have the same logical pipeline architecture but work under different parameter values in terms of resolution, frames per second, and bitrates.

5.3.1 Gstreamer Pipeline Architecture

Each camera stream is processed at the architectural level by a dedicated pipeline, which is made up of several sequential stages. These transform raw data from cameras into encoded video streams that are suitable for real-time transmission and recording. The deliberate design of this pipeline is to be linear up to the encoding stage and to branch afterward, ensuring efficient reuse of the already encoded data. The source element, interfacing directly with the camera hardware through the Linux Video4Linux2 (V4L2) subsystem, begins the pipeline. This element continuously captures video frames from the physical camera device and injects them into the pipeline. Since cameras might output video in different formats, this source stage is responsible only for data acquisition, allowing format handling to be done by subsequent elements. Following capture, decoding and conversion elements are part of the pipeline. This aims to ensure that video frames coming out of capture are converted into a raw, uncompressed format compatible with downstream processing and hardware encoding. This step is quite indispensable because most hardware encoders expect the input frames to be in a specific memory layout and color space.

After obtaining normalized raw video frames, it needs to move on to metadata integration and preparation for encoding. Timestamp overlaying, stream labeling, and additional details such as resolution and frame rate fall into this category. Executing this step prior to encoding ensures that metadata gets encoded directly into video frames and stays visible throughout the system, from recorded files to final data processing. Following pre-processing, encoding is the next stage in the pipeline. The video frames are encoded, using hardware acceleration into an H.264 bitstream. This stream is the most computationally expensive transformation of the pipeline; hence, it is offloaded to the dedicated hardware of the embedded platform. The last architectural element of the pipeline is the stream duplication stage-actually done with a tee element. Here, the encoded in-video stream is split into two parallel paths: one to be used for live and the other for recording. Notably, this is done after encoding has been carried out, so that at every frame, encoding is done only once.

The detailed architecture of the pipeline is given in Figure 5.1 below and each of the individual elements are explained in short in the coming sections below. For better understanding, we can divide the elements into the video capture phase, pre-processing phase, Hardware encoding phase and the streaming and recording phase. Both the pipelines run simultaneously in case of dual streaming and the choosen one runs in case of single streaming. Both the sections are identical and does the same and hence in the figure only one is shown

The detailed Gstreamer piepline code is given below :

```
gst-launch-1.0 -e \
  v4l2src device=${dev} io-mode=2 do-timestamp=true ! \
  image/jpeg,width=${w},height=${h},framerate=${fps}/1 ! \
  jpegdec ! videoconvert ! video/x-raw,format=I420,width=${w},height=${h},
    framerate=${fps}/1 ! \
  clockoverlay time-format="${CLOCK_FORMAT}" valignment=${CLOCK_VALIGN}
    halignment=${CLOCK_HALIGN} font-desc="${OVERLAY_FONT}" shaded-background=${
    OVERLAY_SHADED} ! \
  textoverlay text="${overlay_text}" valignment=${TEXT_VALIGN} halignment=${
    TEXT_HALIGN} font-desc="${OVERLAY_FONT}" shaded-background=${OVERLAY_SHADED}
    ! \
  nvvidconv ! video/x-raw(memory:NVMM),format=NV12,width=${w},height=${h},
    framerate=${fps}/1 ! \
  nvv4l2h264enc bitrate=${br} preset-level=1 iframeinterval=${fps} insert-sps-pps
    =true control-rate=1 maxperf-enable=true ! \
  h264parse config-interval=1 ! video/x-h264,stream-format=avc,alignment=au ! \
  tee name=t \
    t. ! queue max-size-buffers=0 max-size-bytes=0 max-size-time=0 ! \
         flvmux streamable=true ! rtmpsink location=rtmp://${RTMP_HOST}:${
    RTMP_PORT}/${cam} sync=false async=false \
    t. ! queue max-size-buffers=0 max-size-bytes=0 max-size-time=0 ! \
         h264parse config-interval=1 ! rtph264pay pt=96 config-interval=1 mtu
    =1200 ! \
         udpsink host=127.0.0.1 port=${tap_port} sync=false async=false
```

5.3.2 Pre-processing Before Encoding

All the processes after the video capture and before the hardware encoding comes in the pre-processing phase. Pre-processing is highly important for ensuring stable and predictable pipeline behavior. The raw outputs from cameras are often not directly amenable for hardware encoding and streaming. Thus, a series of processing steps are required to transform the video into a format ready for encoding. The different activities that come under this stage are discussed below.

- **Caps Filter :** Immediately after capture, a caps filter is applied to explicitly define the expected resolution and frame rate of the video stream. This prevents unexpected renegotiation during runtime and ensures that the pipeline behavior is stable and predictable. Caps is set differently in different cameras, which enables independent configuration for each stream.

- **jpegdec (Input Decoding) :** "If the camera's video stream is compressed using MJPEG, then the `jpegdec` element is utilized to decompress the frames into raw video." Decompression of the video frames is required because hardware encoding and other processing are carried out using the raw video frames only.

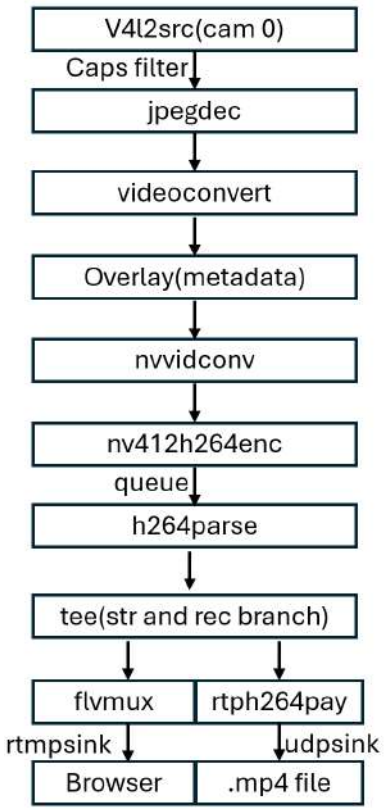

Figure 5.1: Detailed Dual-Streaming Pipeline Architecture

- **videoconvert (Color Space Conversion) :** The `videoconvert` element is responsible for converting raw video frames into a universal color format. The purpose of this element is to ensure compatibility with other elements and eliminate any format negotiation problems. The element essentially works as a normalization stage in the entire processing pipeline.

- **clockoverlay and textoverlay :** `clockoverlay` is an element that displays system time along with date directly onto each frame of the video. This system time stamp is significant because it gives the time at which the video was captured or processed at the sender's end. The `textoverlay` element is used to overlay metadata information related to the specific stream. The information may include the following: Camera identifier, resolution information, framerate information, bitrate information, IP information, and type of encoders. The overlay takes place before encoding.

- **nvvidconv (Memory and Format Conversion for GPU) :** The `nvvidconv` element performs the conversion of the video frames to accessible memory for the GPU in the system prior to encoding. The conversion allows for zero-copy memory transfers from the CPU to the hardware encoder and enhances the

performance on the embedded system.

```
v4l2src device=${dev} io-mode=2 do-timestamp=true ! \
  image/jpeg,width=${w},height=${h},framerate=${fps}/1 ! \
  jpegdec ! videoconvert ! video/x-raw,format=I420,width=${w},height=${h},
    framerate=${fps}/1 ! \
  clockoverlay time-format="${CLOCK_FORMAT}" valignment=${CLOCK_VALIGN}
    halignment=${CLOCK_HALIGN} font-desc="${OVERLAY_FONT}" shaded-background=${
    OVERLAY_SHADED} ! \
  textoverlay text="${overlay_text}" valignment=${TEXT_VALIGN} halignment=${
    TEXT_HALIGN} font-desc="${OVERLAY_FONT}" shaded-background=${OVERLAY_SHADED}
    ! \
  nvvidconv ! video/x-raw(memory:NVMM),format=NV12,width=${w},height=${h},
    framerate=${fps}/1 !
```

This part of the pipelines deals with the pre-processing stage of the video which includes how th eframes are decoded, normalized along with the addition of metadata and later the placement of all the data from CPU to GPU.

5.3.3 Hardware-Accelerated Video Encoding

The most CPU-intensive task involved is video encoding. The reason why the processing of raw video images to compressed images is a resource-intensive task is that it can easily overload general-purpose processors. In this context, hardware-accelerated video encoding is implemented to counter this problem. The platform provides an integrated hardware-based video encoder that supports the H.264 video compression standard. The hardware-based video encoder is designed and optimized to function without any intervention from the processor. The hardware-based video encoder also ensures that multiple video streams can be processed by the system without increasing CPU usage. The part of the pipeline dealing with hardware encoding is :

```
nvv4l2h264enc bitrate=${br} preset-level=1 iframeinterval=${fps} insert-sps-pps=
    true control-rate=1 maxperf-enable=true ! \
  h264parse config-interval=1 ! video/x-h264,stream-format=avc,alignment=au ! \
```

The encoder is designed with real-time considerations in mind. The `nvv4l2h264enc` element is a hardware H.264 encoder that supports real-time encoding on Jetson-class hardware. This element is useful because CPU-intensive encoding tasks are handled by hardware and because multiple feeds can be encoded simultaneously without latency. The parameters like bitrate and keyframe interval are set at values that aim at finding a compromise between compression efficiency and latency. A smaller value of keyframe interval is beneficial for error recovery and lower latency, while proper bitrate allocation is optimal for network bandwidth usage. Since each camera pipeline has its own encoder configuration, the system allows different streams to use different encoding schemes. This allows, for instance, the generation of a navigation stream with low latency and an inspection stream with a focus on image quality.

5.3.4 Independent Stream Configuration

A major requirement in the dual-stream system entails the capability to set each video stream individually. Instead of forcing the same parameters in the different video streams, the system provides the flexibility to set the resolution, frame rate, and bitrate of each camera individually. These are the parameters that are set through a centralized configuration file in the project directory when the pipelines are initialized. Each pipeline has its own configuration parameters, which are utilized during the pre-processing and encoding process. These enable experimentation and the allocation of functional roles for each stream. Independent configuration allows for greater system robustness. In this way, if there is a problem with one of the streams that needs to be configured, this is not a problem for the other stream. In fact, this is very important for a UAV system. Here, if there is a problem with the system but it is still functional at a certain level, this is okay. For various future activities related the inspection and navigation, the parameters can be changed accordingly from the config file without affecting the whole code.

These are the parameters used in the current development and it can changed for the use cases differently.

```
CAM0_WIDTH=1280
CAM0_HEIGHT=720
CAM0_FPS=30
CAM0_BITRATE=4000000

CAM1_WIDTH=640
CAM1_HEIGHT=480
CAM1_FPS=20
CAM1_BITRATE=2000000
```

5.4 The Streaming Subsystem

This section describes how the proposed dual-stream UAV framework implements its live video streaming and media routing subsystem. Complementing the previous section, which described the process of capturing, processing, and encoding of video streams on the onboard platform, the following section will explain in detail how the encoded streams are actually transmitted, routed, and delivered to the ground station in real-time. The path designed for streaming has been kept simple, modular, and satisfying low-latency requirements. In order to satisfy these demands, the system assumes a layered architecture: the onboard video pipelines act as producers, and an intermediary is provided by a dedicated media routing server while the ground station acts as the consumer. This will allow each component to evolve independently and simplify the management of the system.

5.4.1 RTMP based Stream Publishing

After the video frames are compressed and packaged in H.264 format, they are then distributed on the onboard system and posted on the media router server through the Real Time Messaging Protocol (RTMP). The Real Time Messaging Protocol has an inherent connection model that allows for a persistent and reliable connection and is very well supported in multimedia frameworks such as GStreamer. This connection model makes it very ideal for handling continuous distribution in embedded systems such as in this system. The system utilizes RTMP as a transport layer from the source to the route layer but it is not meant for playing back.

However, one key benefit of using the RTMP protocol is its simplicity in design. While in other protocols the process of session establishment can be quite intricate, in the case of RTMP, the publishing of streams to predetermined destinations can be accomplished easily via the URL method. Even though the RTMP protocol is not the one with the lowest latency as far as the transmission of streams is concerned, the requirement in the proposed system is only limited to the ingestion of streams in the internal platform. The requirement of latency in the transport of streams is later fulfilled by WebRTC.

5.4.2 MediaMTX Configuration and Deployment

The media routing server is an integral part of decoupling the producers from the consumers in a video distribution scenario. The system will utilize MediaMTX as the media routing server for decoupling. MediaMTX is a lightweight and real-time media server supporting a variety of transport protocols and can be considered apt for an embedded and experimental system scenario. MediaMTX operates as a service in a containerized domain through the utilization of Docker. The server will listen for the incoming stream from the onboard pipelines based on RTMP and provide it for other clients to access. One of the major benefits of MediaMTX is that it is a protocol-agnostic system. After ingesting a stream, a system with MediaMTX can provide access to a given media object using multiple protocols without any modifications in the producer side pipelines. It is possible to evaluate various distribution protocols while maintaining the same onboard system. The system is also easy to configure and administer in comparison with a full-featured media server. The configuration file of MediaMTX contains a straightforward way for specifying protocols and access control.

Why MediaMTX and not other similar servers?

Although WebRTC-enabled servers like Janus are very powerful, their main objective is to act as webRTC gateways rather than media routers. The reason for choosing MediaMTX over Janus is based on a variety of aspects such as: First of all is the fact that MediaMTX allows for ingest of RTMP directly. In fact, this directly complements the use of GStreamer-pipes for media processing. The disadvantage of using Janus seems to be needing media protocol conversion if either RTMP or raw RTP media needs to be consumed. Second is that MediaMTX allows for WHEP so

that WebRTC access is easily allowed through HTTP. The process seems difficult with Janus. Third, it has low overhead and is easier to deploy on resource-poor systems. In the case of a research-focused UAV platform, less overhead in the system may, in fact, be more valuable than the ability to perform conferencing, for instance. Lastly, it provides a clean fit with the architectural principle of decoupling the routing of the media from the application logic. It functions more as a transparent relay than a active media handler, hence making it easier to evaluate and analyze.

5.4.3 WebRTC Stream Exposure Using WHEP

The low latency delivery of the live video streaming service to the ground station is achieved through the use of Web Real-Time Communication (WebRTC). The reason why WebRTC is solely used for audio and video streaming in real-time is because this protocol already supports facilities for handling congestion control, jitter buffering, and playback adaptation. In the developed prototype, the use of WebRTC streaming is made possible through the use of the WebRTC-HTTP Egress Protocol (WHEP). From the ground station point of view, playing back a live stream becomes as easy as fetching a given URL. The large reduces in implementation complexity is accompanied by vastly improved usability. Session establishment, media decoding, and playback, are automatically performed by the browser. A further benefit of WebRTC is that the statistics are available on the receiverside. In particular, metrics like jitter, packet loss and rendered frame rate can be queried directly from within the browser. This metric is later useful during evaluation for assessing the quality of streaming and network behavior.

Session Stream Management : Streaming session management is the management associated with live streaming sessions. In the proposed architecture, streaming sessions are managed implicitly by the media routing server and WebRTC. Every stream is published to a distinct path on the media server. When a ground station client subscribes for a stream, a dynamic WebRTC session for that client is established by MediaMTX. A number of clients may subscribe for the same stream without impacting the producer pipeline. Life cycle management for sessions is facilitated due to statelessness in HTTP-based WHEP requests. When a client disconnects, its WebRTC session is automatically released. The onboard system remains stream-publishing irrespective of whether clients are connected or not. This approach prevents any need for handling session control logic directly at the onboard system level. It also enhances fault tolerance, since any error or disconnection at the client level will not be reflected at the producer pipelines.

The `tee` element receives the encoded stream and separates it for streaming and recording respectively. The part of the pipeline that deals with streaming and the individual elements are discussed below:

```
t. ! queue max-size-buffers=0 max-size-bytes=0 max-size-time=0 ! \
        flvmux streamable=true ! rtmpsink location=rtmp://${RTMP_HOST}:${
   RTMP_PORT}/${cam} sync=false async=false \
```

- **queue :** The elements of the queues are inserted right after the Tee objects in each of the branches. The queues remove dependencies from downstream processing so that any lag related to recording purposes - for instance, disk I/O - does not contribute to any issues with live streaming.
- **flvmux(packaging) :** The live streaming branch consists of an element called `flvmux` that puts the H.264 video stream in a "suitable" format for RTMP transport. The effect of this process allows it to be compatible with the media routing server.
- **rtmpsink(publishing) :** The `rtmpsink` element is responsible for transmitting this live video stream into the MediaMTX server. The video is then rebroadcast throughout the ground station via WebRTC.

5.5 Recording Subsystem Implementation

The task of the recording subsystem is to record each camera stream locally on the onboard system as live streaming continues unabated. The requirement to record in UAV operations may be necessary for analysis purposes post-operational activities or for evidence and data collection purposes, while live streaming may be necessary for situational awareness as it provides such in real-time. The reason the subsystem for recording purposes is a distinct path and does not affect live streaming latency is that it has different requirements. The implemented system records without decoding and then re-encoding the video. It just reuses the already encoded H.264 stream and stores it into an MP4 container. This strategy decreases the computational load on the embedded platform and preserves the quality of the video, with the additional possibility of starting or stopping recording on demand. Moreover, recordings also preserve the same burned-in metadata overlay visible in the live stream because overlays are applied before encoding.

5.5.1 RTP Tap and Remuxing Strategy

Recording has an important implementation detail in that it must not introduce any increase in latency or stability issues into the live video stream. In this respect, the live video streaming and record video paths diverge immediately following the encoding module in the system design. This particular requirement in the Record module is enforced through the use of a `tee` module, which splits the output flow into two paths, each of which sends the produced video either to the media routing server for streaming, as well as to the record module. Such a split is not only rational but the most critical from the point of view of performance as well. As the system branching happens after the encoding step, the consequence is that the system does not have the two branches working in parallel on the same video feed from the camera. The most resource-intensive operation on embedded hardware is encoding, and therefore redundant encoding should be prevented if the system is

going to scale. In its own documentation, the need for each tee split to become asynchronous through the use of queues, which prevent one tee from blocking the other, is emphasized[13].

After this separation, the recording part is connected to an RTP tap mechanism. In its architecture, the encoded H.264 stream is packetized in RTP format by means of `rtph264pay` to be forwarded to a local UDP socket. In turn, these RTP packets are received at the recorder thread, which vouchers the video data and remuxes them into an MP4 container. The selection of RTP for tap protocol is attributed to its standard format that allows for easier transport of real-time data like audio or video. This RTP protocol is famously known for being appropriate for real-time transport applications for audio/video streaming. This tap is standardly implemented on the loopback interface `(127.0.0.1)` for localized data that is theoretical in overhead. The key advantage of this is that the recording subsystem can be started and stopped independently without changes in the main streaming pipeline. Instead of dynamically attaching and detaching file sinks inside a running pipeline-which may be fragile-the tap provides a clean boundary between the "publisher pipeline" and the "recorder pipeline." This follows best practices of the community in handling dynamic recording, where branching and dynamic reconnect behavior have to be treated with care in order to avoid instability of pipelines.

The solution allows for the independent control of all recorder operations for each camera feed. The need for this capability is critical for all UAV operations where only those streams that matter might have to be recorded or when the amount of buffer memory for storing the streams is very limited. The solution ensures this capability through the use of an optimal control plane that controls all start and stop operations for the recorders independently of the streaming operations. Each camera operates independently, including its own separate recorder, PID track, and log. On reception of the start recording trigger, the solution spawns a camera-wise recorder pipeline that listens on the relevant RTP tap port. The stop recording action only stops the recorder pipeline and does not affect the streaming graph.

```
t. ! queue max-size-buffers=0 max-size-bytes=0 max-size-time=0 ! \
        h264parse config-interval=1 ! rtph264pay pt=96 config-interval=1 mtu
   =1200 ! \
        udpsink host=127.0.0.1 port=${tap_port} sync=false async=false
```

5.5.2 File Naming and Storage Management

The recorded files are stored in an organized folder structure in the Jetson system. It provides a separate "records/" folder, which separates video artifacts, scripts, and configuration files. Organized file storage enhances system maintainability by enabling clean collection of data in an assessment environment. File naming is done in an organized manner, showing a date-formatted name along with the number of the respective camera. This ensures that there are at least three benefits, including no file overwrite, correlation between files of both cameras, and easy dataset

compilation, where each file name carries significant information by itself. Timestamps are encoded in the filename in a standard time format synchronized with the overlay clock to allow traceability from "what the operator saw" to "what was recorded." Additionally, logs are written in dedicated log directories for debugging and performance analysis ease. Furthermore, disk management also comprises various housekeeping activities regarding disk space utilization. For instance, because video files may account for massive amounts of disk space, the processes in the operational environment entail deletion of old files and retention of files needed for evaluation only. For experiment setups, the approach averts failures resulted by exhausted disk space.

```
uav_streamlab_updated/
  records/
    cam0_YYYYMMDD_HHMMSS.mp4
    cam1_YYYYMMDD_HHMMSS.mp4
  run/
    rec-pids/
    rec-logs/
  scripts/
```

5.6 Metadata Integration and Overlay Design

Metadata proves to be an essential component in real-time UAV video streaming applications because it carries information that cannot be derived from merely analyzing video content. In this dual-stream system, metadata will be directly incorporated into the video processing chain to facilitate real-time monitoring and analysis tasks. Instead of viewing metadata as another data source to handle, the system directly embeds relevant metadata from the video transmission for greater exposure and retention in streaming and file-capturing. The approach of handling this metadata takes into consideration system simplicity, robustness, and independence in using data protocols. The generation of all metadata takes place in the sending entity, with no in-built pre-encoding modifications.

5.6.1

Metadata integration begins with determining the relevance of the data considered essential and how it matters. Metadata in the video systems of a UAV has been vital in this regard, as it relates to situational awareness, validation, and performance. The details included in this aspect, and displayed, include:

- **Timestamp :**validates freshness of the stream and helps with reasoning about latency.
- **Camera ID :**Prevents confusion if both camera streams are shown side-by-side.

- **Resolution :**The resolution function confirms the quality level and says whether the configuration is correct.
- **FPS :** verifies planned motion smoothness and pipe layout.
- **Bitrate :**verifies the configuration of the encoding rate as well as the bandwidth settings.
- **IP address :**facilitates operational debugging and remote access validation.
- **Encoder label :**Encoder label provides transparency about which encoder is used.

In this case, these metadata elements are calculated on the sending side, since it is the Jetson that has the knowledge about the settings (resolution, fps, and bitrate) and the system details (IP address). The remaining metadata elements, such as jitter and RTT, cannot be included in the overlay since they cannot be calculated on the sending side, as they rely on network and browser playback patterns.

5.6.2 Configuration-Driven metadata

To ensure flexibility and maintainability, it has been ensured that the content of the metadata is not hardcoded in the pipeline logic itself. Rather, a configuration-driven paradigm has been incorporated into the system where the value of the metadata is specified through a variety of external configuration files and environment variables. Additionally, this paradigm helps keep the definition of metadata and the logic of the pipeline separate. Through this, it becomes easy to test different configurations of experiments without having to make any changes to the logic of the pipeline. Resolutions or bitrates of experiments can be easily modified centrally, thus affecting the logic of the pipeline as well as the value of the metadata presented in the system. In a research system, it is often a common phenomenon for experiments to be carried out several times with different parameters. On system start-up, the system reads the configuration value and dynamically creates strings representing the value of the metadata, thus passing it to the overlay elements of the video pipeline. Additionally, this paradigm incorporates customization for each of the cameras individually where each stream can showcase its own set of configuration variables.

The config file contains the encoder label, which is the name of the hardware encoder being used. It also contains the format for data and time which is given using `clockoverlay` later in the code. The placeholders for taking the values are also given in the config file. The overlay font, style, alignment are also provided along with.

The following snippet shows an example of the values used in the project:

```
# ========================================================
# Burned-in Overlay Metadata Configuration (Sender-side)
# ========================================================

# Encoder label shown in overlay (purely informational)
ENC_LABEL="H264(nvv4l2h264enc)"

# Date/time format for clockoverlay
CLOCK_FORMAT="%Y-%m-%d %H:%M:%S"

# Text template for metadata overlay.
# Supported:
#   {cam} {w} {h} {fps} {br} {ip} {enc}
OVERLAY_TEMPLATE="CAM={cam} RES={w}x{h} FPS={fps} BR={br} IP={ip} ENC={enc}"

# Overlay styling (clockoverlay and textoverlay)
OVERLAY_FONT="Sans, 22"
OVERLAY_SHADED="true"

# Position of the metadata text overlay
TEXT_VALIGN="top"
TEXT_HALIGN="left"

# Position of the clock overlay
CLOCK_VALIGN="top"
CLOCK_HALIGN="right"
```

5.6.3 Burned-In Video Overlay Implementation

A major design choice in metadata integration is to support burned-in video overlays, meaning that metadata is overlaid directly into the video imagery instead of being sent along as a data stream. This design choice is intentional, based on redundancy and ease of design for redundancy reasons. These overlays are implemented through standard overlay components that are part of the GStreamer library, which works with raw video images before encoding them. Two different layers are applied: for time metadata and for text metadata. "Burned-in overlays" ensure that metadata is always visible irrespective of the streaming protocol employed. Whether it's live streaming via WebRTC or an MP4 file, or accessed offline, metadata will always be visible. This ensures that there's no need to rely on protocols for transporting this metadata, which might not necessarily be supported on RTMP, WebRTC, or in file containers. A benefit of burned-in overlays in this respect is that it ensures synchronization between data and video content. This becomes extremely important in instances where Timestamp analysis is required. Even the smallest discrepancy in timing might result in different interpretations of data. The following snippet is taken from the pipeline implementation and this represents the overlay introduction to the video frames.

```
clockoverlay time-format="${CLOCK_FORMAT}" valignment=${CLOCK_VALIGN} halignment=
    ${CLOCK_HALIGN} font-desc="${OVERLAY_FONT}" shaded-background=${
    OVERLAY_SHADED} ! \
  textoverlay text="${overlay_text}" valignment=${TEXT_VALIGN} halignment=${
    TEXT_HALIGN} font-desc="${OVERLAY_FONT}" shaded-background=${OVERLAY_SHADED}
    ! \
```

The overlay is applied before encoding because once the encoding has happened, the video frames has to be again decoded so as to add the overlay again. There are mainly two types of overlays which are the `textoverlay` and the `clockoverlay`. With real-time video systems, temporal information is one of the most important metadata elements. In this framework, at the sender side, timestamps are generated and rendered directly onto each frame using an overlay element for the clock,`clockoverlay`. The stream identification is rendered through `textoverlay` by inserting the `cam0/cam1` label into the overlay text template. This is particularly important when both streams are rendered side by side within the ground station display. Otherwise, analysis and interaction with the streams will become error-prone.

5.7 Receiver side Webpage(GCS) Implementation

The ground station acts as the human-system interface layer in the proposed dual stream video transmission system. The principal goal of the ground station is to enable real-time viewing of videos, show contextual data embedded in the video stream, and enable the operator to control the recording feature. Unlike the traditionally desktop-based ground control stations, the system implemented has a minimalist web development focus for optimal functionality and suitability for an embedded system. The ground station does not engage in any active media processing or stream handling. It uses native WebRTC functionality in browsers for playing videos and simple HTTP functionality for handling interaction with users. The strategy ensures that the system is lightweight with an easy setup and does not require any platform-specific client software.

5.7.1 Web-Based User Interface Design

The GCS user interface is implemented as a static Web page served from an in-container lightweight Web server on the Jetson platform. The interface is written in standard HTML and CSS only; there is no client-side scripting of logic or data processing. This minimizes complexity and avoids reliance on JavaScript-based control logic, which can introduce extra failure modes and debugging overhead. While most conventional ground station applications require software running on a desktop system, the proposed interface is purely browser-based. This immediately negates any need for client-side installation burdens and guarantees operating system and device compatibility. The interface is served from the Jetson platform itself by means of a

lightweight web server running inside a Docker container, hence guaranteeing that UI is always co-deployed with the streaming back-end and exactly the same across deployments.

The user interface implements the use of standard HTML structures and CSS styling for layout purposes. No client-side scripting has been applied in the interface's logic and handling of the media files. The interface makes use of the browser's WebRTC abilities for video playback, in addition to the use of HTTP interactions for the interface's control functionalities. This approach makes the interface less complex, an important aspect in the case of an embedded system environment, especially when working with systems that operate in a real-time environment.

Figure 5.2: User Interface Design for Dual Streaming

User interaction with the system is deliberately minimized to only necessary control actions. Control recording functionality is introduced in a simplified way with hyperlinks or form elements that send HTTP GET requests to backend CGI scripts. As a result of a user clicking a control element, a request is sent by the browser to the related control endpoint that triggers the appropriate action on the Jetson system. On the usability side, the design of the user interface emphasizes clarity rather than visual interest. The absence of dynamic behaviour on the client-side ensures determinism and predictability for user input events. The latter is extremely advantageous for the evaluation of user interfaces, particularly when it turns to experimentation.

5.7.2 Dual-Streaming Visualization Layout

Functions as a dual-stream visualization layout, which is capable of showing two live video feeds within a single window of a browser, reflecting the dual-camera component architecture of the system itself (described as `cam0` and `cam1`). Primarily, such a layout has been created to aid real-time awareness of operators to monitor both streams simultaneously rather than having to switch between them. This becomes more crucial in cases of UAV operations where each of these streams could function with a different purpose in mind, such as having one optimized for responsiveness, referred to as `cam0`, while others are optimized for detail, referred to as `cam1`. The layout of the implemented ground station is also made to be simple

and deterministic. All streams are put inside their visual panels that are made of the title label, the `<video>` element of playback and a small control area of recording actions. This design will make sure that both streams are distinctly separated and avoid confusion to the operators when the two streams are operating. The layout is not required to have any dynamic overlays or scripting to show metadata based on the sender (camera ID, resolution, FPS, bitrate, and timestamp and IP address), as the metadata is burned directly into the video frames on the Jetson side, and the required context is immediately available in each stream. The design is also based on a separation-of-concerns principle: streams are simply arranged and rendered by the browser, with media transport being done by WebRTC and routing logic being done by MediaMTX. This implies that the dual-stream layout can be kept constant despite dynamism in streaming backends provided the WHEP endpoints of both streams are present.

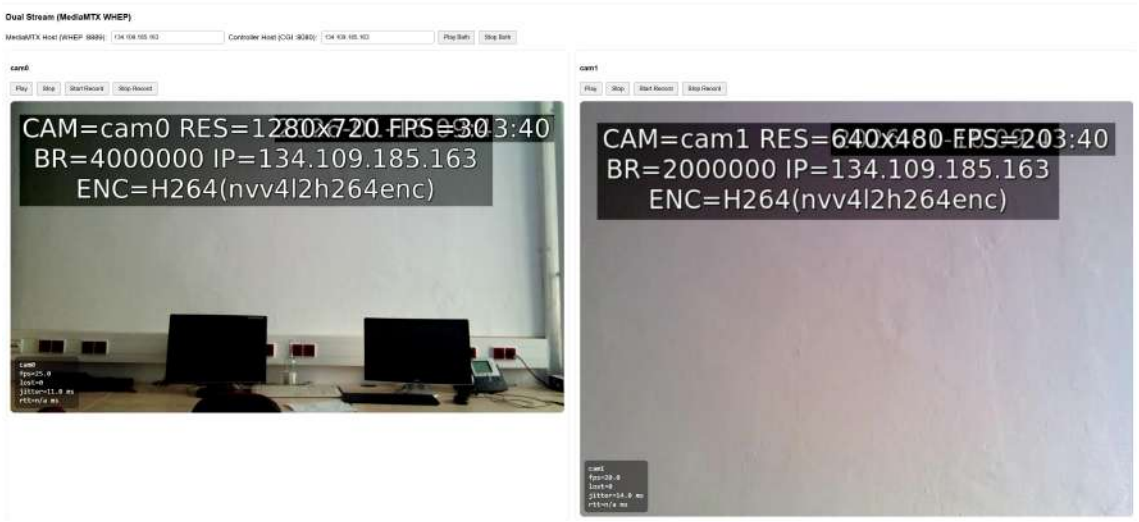

Figure 5.3: Dual- Streaming in the Interface

5.7.3 Receiver-Side Performance Statistics

The receiver-side performance statistics are the description of the actual delivery, decoding, and rendering of the video streams in the ground station after passing through the network and media routing elements. These figures indicate the quality of quality experienced at the stream and not what the sender has intended. In the deployed system, the ground station interface is not explicitly calculating or presenting any numerical measures of performance since it is intended to be light-weight and not be carrying client-side processing logic. In lieu, the receiver-side performance is measured by visual observation, sender-side metadata implanted, and biased use of browser-side diagnostics when experimenting. Primary metric of performance of the receiver side is the playback smoothness, i.e. how well the video can be shown without apparent stalling or jitter. Smooth playback means that the

browser is receiving Frames at a reasonable rate and the Decoding is processing at a reasonable rate keeping up with the received. Wherever the freezing and stuttering is noticeable, this will indicate that the network is saturated, or it is employing a high bitrate or that the receiver is being overloaded by the process of decoding.

The other statistic that is important is the end-to-end latency which is the time difference between the capture of the video at the sender and the display at the receiver. Latency is estimated qualitatively in this system based on the sender-side timestamp written into each frame of video. The operator can also estimate the delay incurred by coding, transmission, routing and decoding by comparing the shown timestamp with the local system time at the ground station. Though this method is not able to give accurate numerical values it gives a sound comparative value between streams. The other metrics that comes along with the performance statistics are:

- **Frame rate :** Frame rate of the receiver displays the number of frames per second actually rendered by the browser. The real receiver frame rate may be lower than the configured frame rate but the configured frame rate is shown in the burned-in metadata. The actuality that there exists a significant reduction in the smoothness of motion/ intermittent updating of frames implies that the receiver is not able to sustain the intended frame rate.

- **Packet loss :**The video data packet loss is called packet loss, and its cause is due to network conditions. The packet loss can be graphically indicated as short artifacts, empty frames or temporary stutters during playback.

- **Jitter :** Jitter is the difference between the time when the reception of packets by the receiver takes place. Irregular playback may be experienced even with average bandwidth and high jitter. The browser WebRTC jitter buffer is used to handle jitter in the system, but jitter may result in playing that is visible to be irregular even with the buffering system.

All these receiver-side statistics are tracked by the WebRTC stack that is available internally to the browser and can be assessed using browser diagnostic tools. The fact that these metrics are not integrated into the ground station interface enables the system to be simple and portable yet it still offers a means to achieve considerable evaluation of the performance of streams. The approach is also viable in the face of the experimental nature of the project through which the qualitative observation and comparative analysis of streams may prove more beneficial than raw numerical measures in the majority of situations.

5.7.4 Stream and Recording Control Integration

Stream and recording control integration The ground station enables the operator to do more than just observe the live video system by integrating stream and recording control. This integration in the implemented framework is a deliberate design that is

straightforward, declared, and dependable, in general, the philosophy of maintaining a lightweight ground station and good control over the key system processes. Live streaming and recording are considered two features that are not interdependent and the ground station only controls the recording subsystem but allows streaming to run on-demand. To the ground station, streaming can always be allowed provided the system is operational. This design eliminates the necessity to use start/stop controls in the user interface to stream in and cannot interrupt the real-time video delivery accidentally. That is why the ground station is only concerned with the recording control that is operator-driven activity and might not be necessary at all at certain points of a mission or experiment.

The recording control mechanism is one of the very simplest based on HTTP mechanisms. The proclaimed streams displayed in the user interface by the way of hyperlinks or buttons possess certain means to start and stop recording. When an operator makes a control, the browser sends an HTTP GET request to a CGI program that is actively running on the Jetson platform. A request consists of the camera ID of the camera (cam0 or cam1) and the action which is required (start or stop). This control system is request based, therefore explicit control plane which is totally dis-integrated with the media plane. One of the strengths of this design is that it is very robust. Since recording is dealt with by different processes and is not managed internally, any faults or delays by the recording subsystem will not be fed back to the live streaming pipeline. Equally, pausing a recording does not involve reconfiguring a running GStreamer pipeline, which can be complex in dynamically reconfiguring a pipeline, and can also minimize the risk of instability. Manual recording of actions can be enabled by the use of a browser or command-line tools and therefore makes the system easy to test, debug, and extend in terms of functionality as well.

5.8 System Control and Automation

Automation and system control This is necessary in ensuring that there is reliability in the operation of the proposed dual stream video transmission framework, particularly in an embedded UAV-based system where manual intervention would pivotal to a minimum. Automation in the system implemented is done via a series of organized shell scripts and configuration files that handle service startup, shutdown, process life-cycle and runtime parameters. In this way, it is possible to repeat experiments, it is easy to deploy, and the chances of human error in its work are minimized.

5.8.1 Startup and Stop Automation Scripts

The system relies on special startup and shutdown scripts to put all the necessary components in known and consistent state. Instead of having to start the media server, streaming pipelines, and control interfaces one at a time, the startup script

has the duty of coordinating the entire process of bringing the entire system online.

The automation script does a clean start during the startup process by first ensuring that there are no stale processes or containers of the last runs. It then introduces containerized services like the media routing server and the web interface as the next level and then the control server which deals with recording commands. After the core services have been started, the script will check whether there are any camera devices by keeping track of them and launching the relevant streaming pipelines to each device. This sequential startup order will make sure that the services that are dependent are ready before the process of video publishing commences.

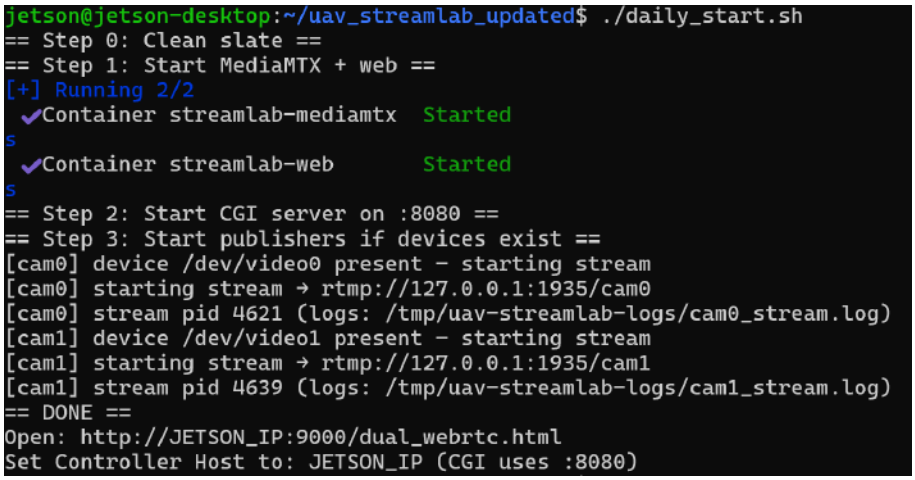

Figure 5.4: Executing the starting bash script

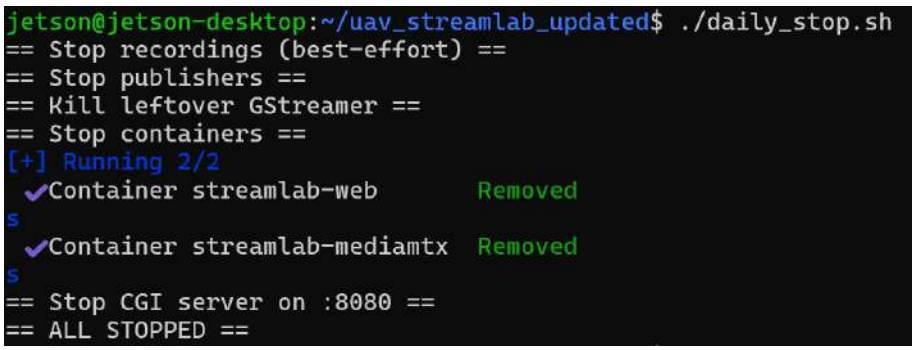

Figure 5.5: Executing the stopping bash script

Shutdown automation has a similar format. The first thing the shutdown script will do is to halt any processes that are in active recording so that recorded video files can be finalized correctly. It also halts streaming pipelines, halts control services and eventually closes containerized components. The system can prevent corrupt recordings, orphaned processes, and locked network ports, which are some of the issues that would arise when an orderly shutdown is not followed.

5.8.2 Process Management and Fault Handling

The management of processes which is adopted in the implemented system is to be explicit and transparent. The key components (streaming pipelines, recording processes and control services) are launched as individual processes. PIDs are not only registered in special runtime directories, but can be used to monitor the running components. The system can prevent- or restart any part in a safe way and will not interfere with others as it maintains clear PID files. For example, Recording process might be aborted regardless of the live streaming pipeline or re-initiating one camera stream might happen without shutting down the entire system. Fine grained control can be easily applied during the process of debugging and evaluation.

Fault handling is done in a best-effort approach as opposed to elaborate recovery logic. Each time a new process is initiated, the system will ensure that no previous process of the same role is already in operation and ends it in case of necessity. The scripts during the shutdown will work to ensure that all known processes are gracefully brought to a halt and cleanup of any left over artifacts is done. This can easily be understood from the Figures 5.4 and 5.5. This method will guarantee that a temporary failure or sudden shutdown will not place the system in an irregular state. In terms of design, this approach is biased towards sturdiness and simplicity. Instead of trying to automatically recover all possible failure conditions, the system guarantees the ability to get the system in a clean state always by means of controlled shutdown and start-up. This more effectively suits experimental systems, in which predictability and repetitions are more critical than continuous autonomous recovery.

6 Results and Evaluation

The chapter is a report on the findings and analysis of the proposed real-time dual stream video transmission system that will be utilized by UAVs. This testing is meant to ensure that the system is working correctly in all the circumstances and its operation is effective, and that its consumption of resources on the platform integrated in the system is analyzed. It is tested on both streaming and recording capabilities and functionality of the system as many video streams with different settings pass through it simultaneously. The test is intended to be functional validation, quantitative resource testing and stress testing under various configurations. The functional evaluation proves the following system features, and they are as follows: the system is configured to work in two streams independently, metadata overlay, and recording control are performed in accordance with the expectations. Quantitative analysis is used to determine parameters of the system that can be measured such as the CPU and the utilization of the GPU, utilization of memory and the calculation of metrics like FPS, latency and the number of dropped frames. Finally; there is stress testing which is carried out to test the stability and limits of the system when it is running at full load, e.g. when there are more resolutions, higher frame rates and recording at the same time.

6.1 Functional and Scenario-Based Evaluation

Here the section analyses the functionality of the proposed system in the various operating conditions. This is not meant to quantify the performance in any way, but to ensure that everything that is supposed to be a core functionality of the system, such as dual streaming, independent configuration, recording, and metadata display, is functioning as intended. The subsections concentrate on one particular functional element on the system as well as confirm it using controlled test situations.

6.1.1 Dual-Stream Validation

This sub section authenticates that the system supports the capability of streaming two independent camera sources of video in real time. This test aims to ensure that the proposed framework can manage parallel video pipelines without conflict, instability, and interruption, which is one of the basic requirements of a multi-camera UAV application.

In this assessment, the two cameras (cam0 and cam1) were attached to the system and identified when the system was initially started. The automated startup script

was activated, which started the media routing server, control services and single pipelines of GStreamer each to each camera. The pipelines sent their encoded video streams to their own endpoint and the media routing server revealed both the video streams to the ground station in separate WebRTC sessions. A successful operation in dual-stream was confirmed at the ground station, by showing the two video streams simultaneously on the browser-based interface. All streams were updated in real time and they stayed stable during the test period. The fact that two independent video panels were present proved that the system could support multiple simultaneous media sessions and not to contend with resources and not to interfere with the streams. Notably, the working of a stream did not influence availability or behavior of the other indicating an appropriate isolation between pipelines.

This test shows that the architecture of the system can indeed be dual-stream, and there is a distinct camera capture, encoding, publishing, and delivery path per camera. The outcome provides a practical basis of further assessments of independent configuration, recording, and performance analysis.

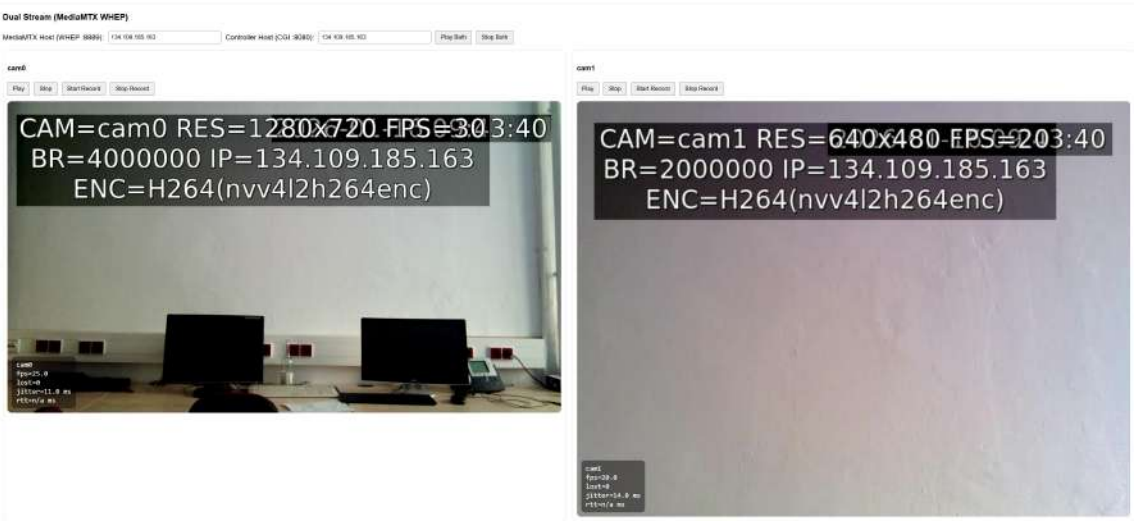

Figure 6.1: Dual-stream visualization-in side by side

6.1.2 Independent-Stream Configuration

Independent configuration of camera streams is one of the most crucial needs of multi-camera UAV systems since various camera streams can be used to serve different functional purposes and, therefore, can have varying quality and performance demands.

The camera specific streaming parameters of the implemented system are to be set in the configuration file and applied to each GStreamer pipeline separately. In this test, cam0 and cam1 were assigned to various settings. The quality based stream was represented by one of the streams with high resolution and frame rate and the other stream was represented by low settings that represent the resource

efficient stream. This was succeeded by automated startup script that initiated the system and system was checked at the ground station by watching both of the live streams. The active configuration parameters (resolution, frame rate and bitrate) were immediately verified by the burned in metadata overlay on each video frame. The values shown were in compliments with the values of each camera given and apparent divergence in quality of the images and motion also served to indicate that streams were operating in different settings. The individual streams getting started can be visualized in Figure. 6.2.

The fact that the system can appropriately introduce camera specific settings and can be independently controlling all the streams testifies to such an evaluation. Streams can also be configured independently in the performance, which enables an adjustment to different mission requirements to be flexible and is the basis of performance trade-offs analysis in the later sections.

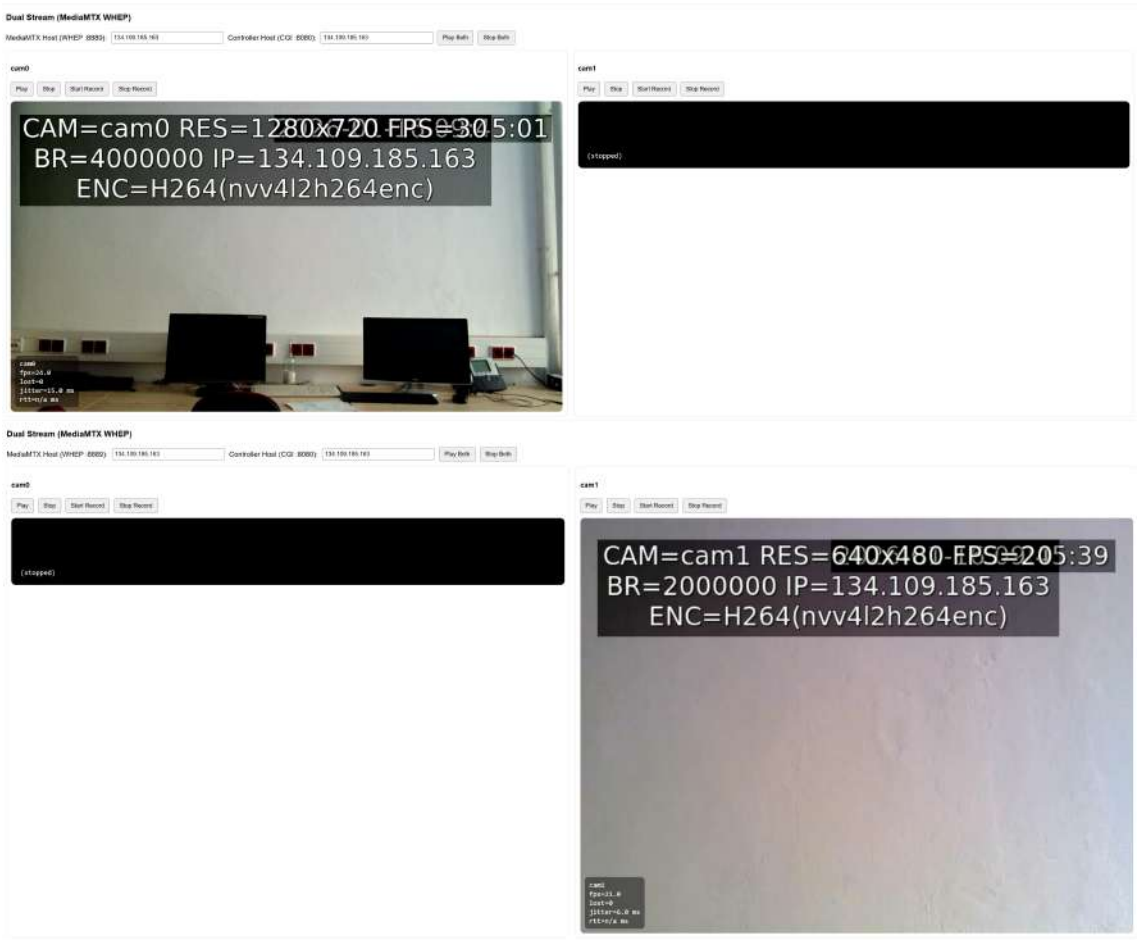

Figure 6.2: Independent Streaming of both Cameras

The system since integrated with the GCS developed, We could see the same view in the main GCS webpage implemented. The port number of the WebRTC channel has to be entered in the webpage so as to get the view from the cameras. It provides

the option for selecting the camera and thus could possible see the view from the selected camera as in Figure 6.3.

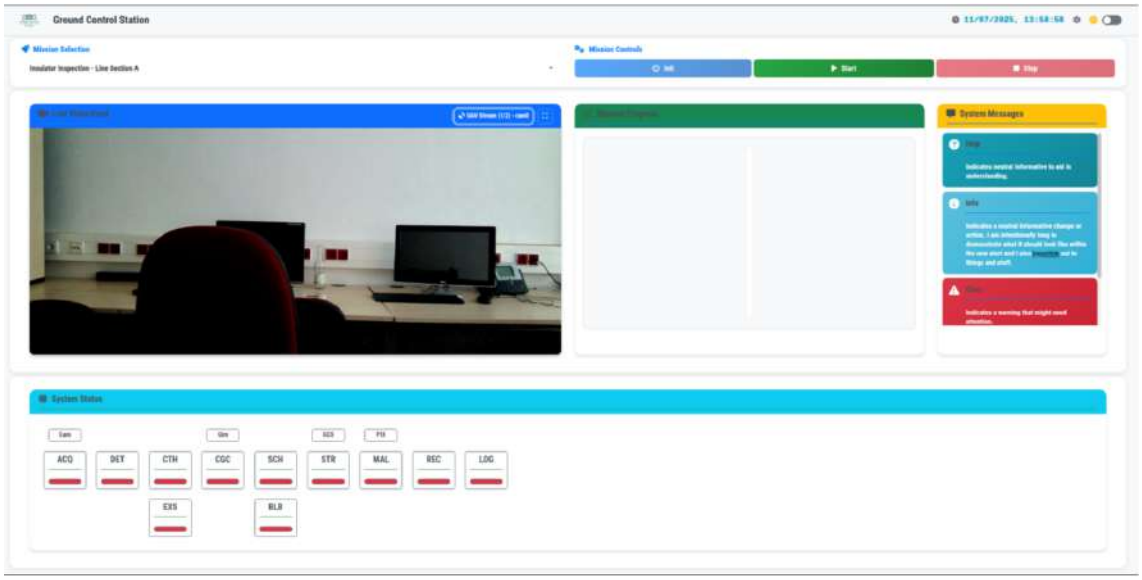

Figure 6.3: Selected Camera view in the main GCS

6.1.3 Streaming and Recording Coexistence

The concept of streaming and recording coexistence is the capability of the system to provide the live video streams to the ground station and at the same time store the same video locally without disturbing either of the functions. It is a necessity of UAV applications where real-time monitoring and post-mission analysis can frequently be needed simultaneously.

In this assessment, camera streams had been initiated and confirmed at the ground station to maintain live play back. During the streaming, the recording was activated with the help of recording control interface. During the recording time, the live video streams kept on updating live meaning that the recording activation did not affect the pipelines of streaming. There were no apparent interruptions, freezes or restarts of streams witnessed in this process. Once the recording was put on hold, the system was examined to ensure that the video files recorded were made and finalized properly. The fact that there were legitimate video files proved the fact that the recording subsystem was functioning properly as long as live streaming was in progress. Notably, the streaming did not cause any issues prior to and after the recording was activated, indicating that the streaming and recording paths remained separated.

This analysis proves the fact that concurrent streaming and recording are supported by the system architecture. The system avoids the real time video delivery

of the video being compromised by other processing related to recording because the recording path and live streaming path are separated after encoding.

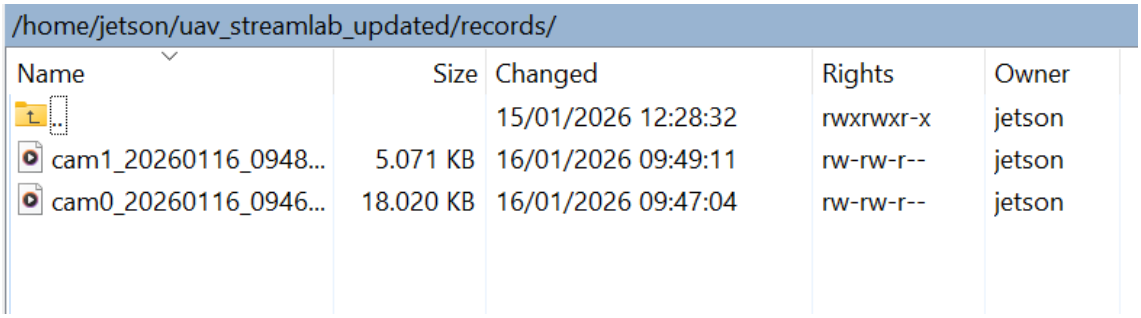

Figure 6.4: The files being recorded getting stored in the records folder

The recording start and stop buttons too provides an alert about the functions being called and the files and data being stored at respective locations as shown in Figure 6.5. The quality of this system, which makes it stand out, is the ability to start and stop recording each video stream without disturbing the work of other streams. To test this capability, both streams of the cameras were first switched on and tested in the ground station in order to test the stability of the live streaming. The recording functions on the webpage acted effectively even when the cameras were streaming and not streaming. This was monitored to make sure that the system behavior was as desired in which the recording action was not permitted to introduce any change to the streaming behavior of the stream as well. Moreover, the files getting stored and the respective size of the files can be seen from Figure 6.4. This ensures the recording check which includes ensuring the files getting played and the proper size being captured.

6.2 Quantitative Resource and Performance Analysis

In this section, a quantitative analysis of the system in respect of its resource consumption and performance under various operating conditions will be given. The aim is to gauge the CPU, GPU, and memory resources, latency, Output FPS and dropped frames in various single-stream and dual-stream mode, and recording-on. Through careful parameter changes by altering parameters like resolution, frame rate and bitrate, the analysis will be able to offer an insight into how the system would scale as workload increases and which points of the system might act as bottlenecks. The findings in this segment are grounded on the data gathered personally at the Jetson platform utilizing system monitoring applications and provided in the form of tables, graphs, and supportive screenshots.

The test cases were all maintained with fixed frame rate of 30 FPS and the different uses cases are mentioned below:

- Single streaming at 720p resolution

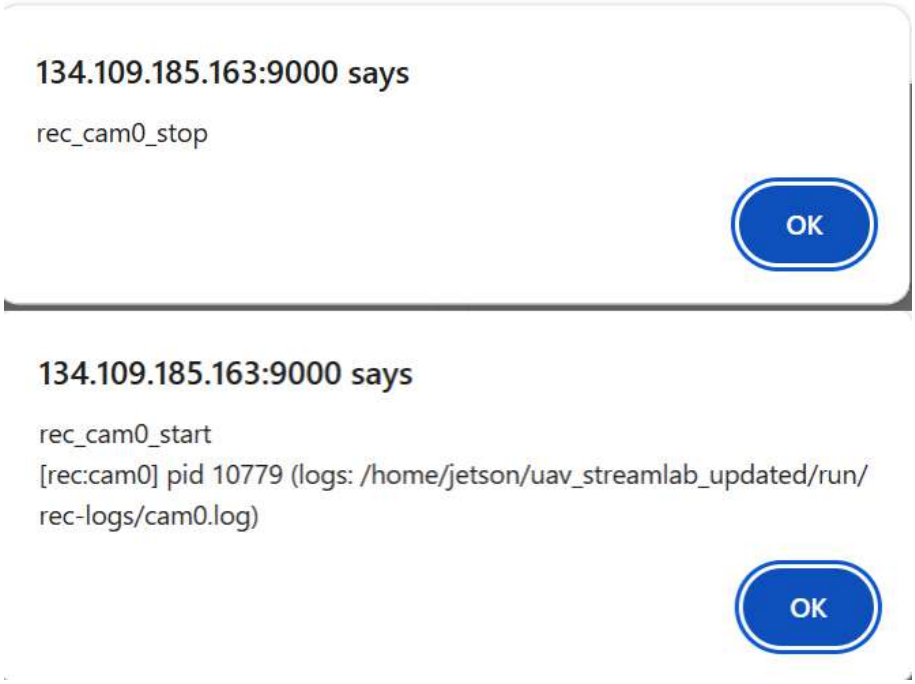

Figure 6.5: Recording function dialogue box

- Single streaming at 1080p resolution
- Dual Streaming with both cameras at 720p
- Dual Streaming with one camera at 1080p and other at 720p
- One camera at 1080p and recording enabled
- Dual Streaming with one camera at 1080p and other at 720p and recording enabled

For each of the test cases the following metrics were calculated and studied which are explained in detail in further subsections.

- **CPU Usage :** It is expressed in percentage and is a measure of the average, and the maximum workload on the system when streaming occurs.
- **GPU utilization:** Displays the hardware acceleration being utilized as well as available space.
- **RAM usage:** This will provide an indication of the average and the maximum usage of the memory that will be used to buffer and process on average and peak.
- **Latency :** end-to-end latency is the duration required to have a video frame traversing between the point at which it is created on the transmitting side and when it is received on the receiving side.

- **Average FPS :** It may be said as the efficient frames per second that have been utilized and are usually how the real time may be flowing effortlessly and freely.
- **Dropped frames :** Frame drops refer to how many frames of the transmission process or processing process that describes the pipeline strength were lost.

6.2.1 Sytem Load Analysis(CPU Usage, GPU Usage and RAM Usage)

This sub section highlights and discusses what system level metrics should be used in assessing and analyzing the level of computational load brought by various video streaming settings. Essentially, through theory and practical analysis, there has been a need to learn and analyze how feasible it would be to comprehend CPU consumption, GPU consumption, and RAM consumption with a view to learning how the proposed system would react with a varied amount of work. Essentially and specifically, just these three aspects were analyzed on all existing test circumstances of single and coupled video streaming with recording. Table 6.1 highlights and shows how each case was analyzed. However, there has been a need to comprehend how well the system would react and execute required performances within limits, and this has often been associated with embedded UAV systems.

The CPU consumption was calculated using both the CPU average and the CPU peaks. The CPU average usage is a measure of constant work done or processing power carried out by the system. The CPU peaks are a measure of increased work done or increased periods of enhanced processing power carried out in short periods or short time through either exposure to intense activities of either encoding, buffering, or concurrent activities of both video and sound stream and/or recording. The use of a GPU was also measured in this regard to show us what form of hardware acceleration was used within this system and its processes of execution. The RAM used in this system is measured similarly in terms of average and peak

These tests were done by logging the tetgrastats data for every one second using the `tetgrastats`command and then later running the pipeline. The pipeline was made to run straight for 10 minutes and then the metrics were calculated from the terminal itself. The log file was used by bash scripts tpo calculate all the values for system load.For example, Each of the line of the log file would look like :

```
RAM 2394/3956MB (lfb 11x2MB) SWAP 923/1978MB (cached 12MB) CPU [66%@1479,81%
@1479,73%@1479,54%@1479] EMC_FREQ 0% GR3D_FREQ 0% PLL@47.5C CPU@51C PMIC@50C
GPU@47.5C AO@53C thermal@49C
```

Since, the pipeline was made to run for 10 minutes, we could get approximately 600 lines similar to this: From this the values along with RAM corresponds to the average usage of RAM in each case, the values with CPU gives the CPU usage among all the four cores and from this the average for each sample can be calculated. The

Test case	CPU Usage (%)		GPU Usage (%)	RAM Usage (MB)	
	Average	Peak		Average	Peak
720p30	51.46	95	4.65	2522.5	2999.7
1080p30	39.75	81	3.34	2650.8	3202
720p30 + 720p30	68.43	75.75	3.35	2571.8	2662
1080p30 + 720p30	76.5	84.5	1.92	2361.5	2488
1080p30 – Stream and record	61.09	81.7	3.22	2029.4	2142
1080p30 + 720p30 – Dual stream and record	59.23	96	3.95	2137.3	2339

Table 6.1: Average and peak CPU/GPU/RAM usage for different test cases.

maximum of which is the peak usage in each case. `GR3D FREQ 0` gives the average GPU usage in each sample. From this data, we calculate the average of all the metrics using simple bash script so that we get the metrics values in ease within the terminal itself.

The quantitative data reported in Table 6.1 indicate an increase in the computational demand as video resolution, streams in operation, and the presence of the record functionality increases. The use of CPU is clearly dependent on stream resolution and concurrency. When using single stream at 720p at 30 frames per second the average CPU usage of the system is 51.46% and the maximum usage is 95%, meaning that there are some higher events in the CPU usage when it is processing an encoder or buffer. Conversely, single-stream 1080p30 operation results in a lower average CPU utilization of 39.75% and a peak of 81% and possibly more efficient operation since the encoder is optimized, although with the high resolution. Despite having a higher resolution, the average CPU load of 1080p streaming is lower than 720p streaming because of variations in the behaviour of the encoders and pipeline optimisation, and there are less overheads in CPU scheduling as the streaming process approaches sustained operation. With the introduction of dual streaming with homogeneous resolutions (720p30 + 720p30) the average CPU use is high at 68.43% and the peak is comparatively low at 75.75% which is a sign of balanced workload distribution. The case of heterogeneous dual-stream (1080p 30 + 720p 30) is the highest sustained load on the CPU with the mean of 76.5 percent and the peak of 84.5 percent showing the extra processing load incurred due to mixed-resolution handling. The aspect of recording also has its effect on CPU. In 1080p streaming at 30 with simultaneous recording, the CPU load is 61.09% on average with the peak load of 81.7% which reflects the extra expense associated with simultaneous recording and disk I/O. In the most taxing system, dual streaming and simultaneous recording (1080p30 + 720p30), average CPU usage is moderate, at 59.23% percent but maximum CPU use is 96 percent, which suggests that there are brief stressful

situations in parallel processing. Even with these peaks, the system is able to stabilize without frame drops and this proves that the CPU resources are adequate to support real time performance.

The use of GPU is still low in all the considered configurations. Single-stream 720p30 and 1080p30 conditions display 4.65 and 3.34 percent utilization of GPU respectively. Dual-stream systems have equal or even less GPU utilization with values of 3.35 and 1.92 percent respectively. In streaming and recording, the use of a GPU is less than 4, and in the most demanding case, with two streams and record, the usage is 3.95 percent. These findings suggest that the video transmission pipeline is CPU bound and that the resources of a GPU are underutilized, which offers a lot of spare bandwidth to integrate vision based tasks like object detection or tracking in the future. The video encoder is implemented on the special NVENC units, which are not captured in the overall metrics of using the GPUs. Hence, a low GPU utilization means an efficient offloading and not underutilization.

There is stable and controlled behavior of RAM usage in all cases of the test. In single-stream cases, the average RAM use is between 2522.5 MB (720p30) and 2650.8 MB (1080p30) with the highest values of 2999.7 MB and 3202 MB, respectively. Dual-stream configurations exhibit the middle memory used, as 720p dual streaming has an average RAM consumption of 2571.8 MB and mixed-resolution case is 2361.5 MB. With streaming and recording on-demand, the average RAM usage will drop to 2029.4 MB, with the peak usage range being in 2142 MB, which reveals that the memory is used wisely. The peak RAM usage of 2339 MB is registered in the dual-stream and record setup, still being quite within the memory capacity of the system.

All these metrics are better visualized in the graph below which incorporates all the system load values. The memory usage even though calculated in terms of size (MB) in the table, it has been defined in % in the graph. The total memory available is 4096 MB and thereby the value in % is calculated.

In jetson nano the direct GPU Utilization in terms of a value cannot be said, because this is not evident in terms. We can only check whether the encoder `NVENC` status, whether it is ON or OFF. More over, from the data collected from the `jtop` command shows less CPU offload eventhough the hardware encoder is active. This is shown in Figure 6.7.

In general, the analysis of the system load indicates that the CPU usage is predictably high with the increase in the complexity of the workload, the use of the GPU does not exceed the minimum in all settings, and the use of the RAM does not leave the safety limits of its operation. These findings validate that the suggested multi-stream video delivery system may be used to facilitate real-time performance in challenging conditions and preserve the effective and consistent usage of resources, which is why it can be implemented on an embedded UAV platform.

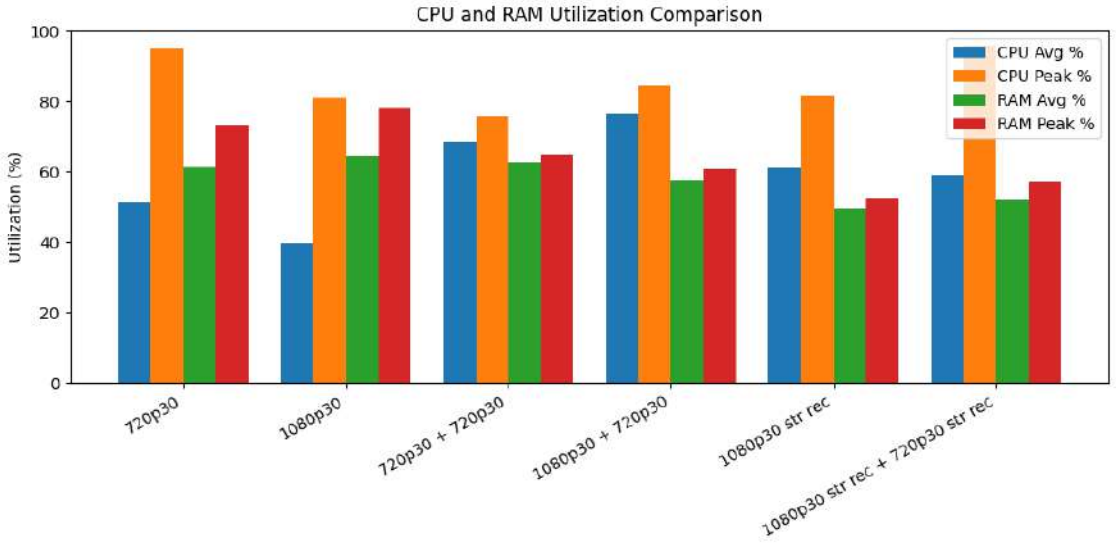

Figure 6.6: System load vs Test case graph

6.2.2 Streaming Performance Assessment (Latency, Frame rate, Dropped frames)

In this section, the performance of the suggested multi-stream video transmission framework in relation to three important metrics, namely end-to-end latency, average frame rate (FPS), and dropped frames will be assessed. Such measures are necessary to evaluate the live video transmission of UAV schemes where the delay, smooth playback, and consistent frame transmission have a direct effect on the effectiveness of operations. Table 6.2 shows the results in terms of the measured results in all test cases. Similar to the system load file, we make a log file for performance assessment as well for each of the test cases. It was made by making use of the `fpsdisplaysink` for the needed branches in the pipeline. For example, if there are 2 streaming and recording, we make use of three `fpsdisplaysink` at the end of each branch. After running the code for 10 minutes, we get a log file similar to the one below:

```
/GstPipeline:pipeline0/GstFPSDisplaySink:fps_cam0: last-message = rendered:
24, dropped: 0, current: 23,73, average: 23,73
```

From this data, current and average gives the FPS and dropped gives the number of dropped frames. Similar to System load, we use simple bash code for taking the average of all these values. Latency is measured by logging the sender time in the overlay and the real time from the system into a csv file and then taking the average of this. For each of the test case, 4-5 samples were recorded and then the average was made. For example, each of the test case would have a latency file similar to the one below:

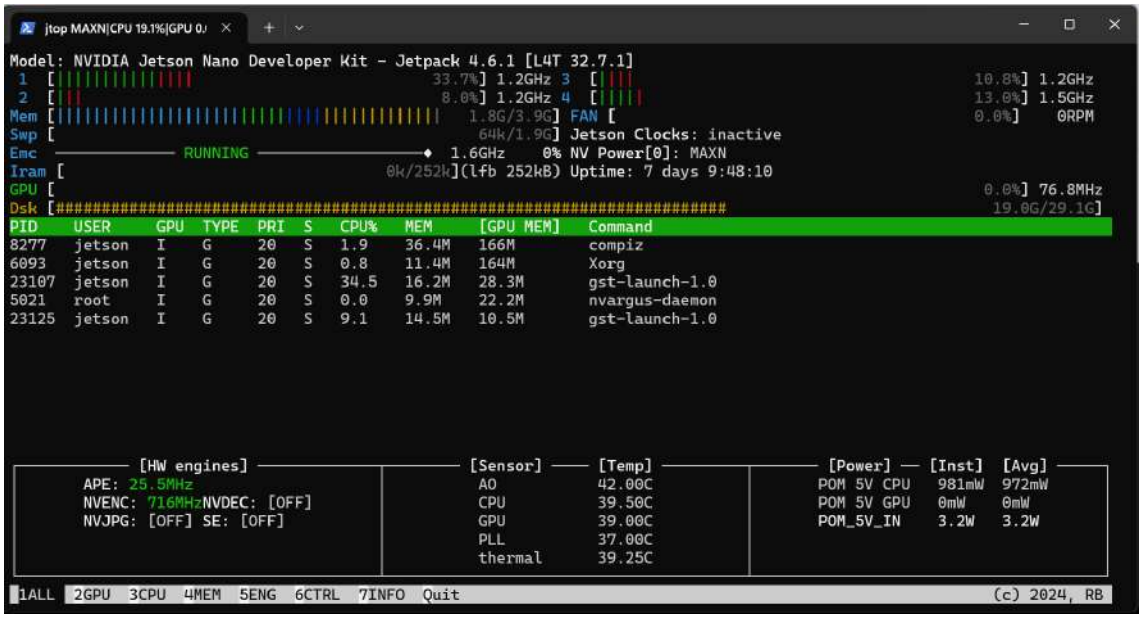

Figure 6.7: Encoder ON and less CPU usage in high payload

```
sample,receiver_time,sender_overlay_time,latency_ms,notes
1,2026-01-24 12:30:10.080,2026-01-24 12:30:10.225,145,ok
2,2026-01-24 12:31:42.080,2026-01-24 12:31:42.225,145,ok
3,2026-01-24 12:33:05.080,2026-01-24 12:33:05.225,145,ok
4,2026-01-24 12:34:28.080,2026-01-24 12:34:28.225,145,ok
5,2026-01-24 12:36:01.080,2026-01-24 12:36:01.225,145,ok
```

The end to end latency is experiencing an increase with the complexity of streaming and workload on the system. Latency is relatively small in single-stream cases (110 ms at 720p30 and 135ms at 1080p30) with real-time responsiveness being observed. This is because the latency experienced with 1080p streaming is higher because it requires more encoding and processing. Latency in the dual-stream application enhances even more since several video streams are handled at once. The homogeneous dual-stream scenario (720p30 + 720p30) is associated with a 145ms latency whereas the heterogeneous situation (1080p30 + 720p30) has a 170ms latency of the 1080p stream and a 155ms latency of the 720p stream due to the imbalance in computational needs between mixed-resolution streaming. The maximum latency values are the dual-stream and recording set up, as the latency is up to 185 ms and 170 ms because the overhead of parallel encoding, transmission, and disk I/O is combined. Although this has increased, the latency is still within acceptable real time UAV video applications.

The metrics can be better visualized from the graphs generated below using the data collected and using python.

The findings of the frame rate results indicate that the system is stable in terms of

Test cases	Latency (ms)		FPS (average)				Frames dropped			
	cam0	cam1	cam0		cam1		cam0		cam1	
			Stream	Record	Stream	Record	Stream	Record	Stream	Record
720p30 - Single	110	–	30.00	–	–	–	0	–	–	–
1080p30 - Single	135	–	23.61	–	–	–	0	–	–	–
720p30 + 720p30	145	150	30.00	–	29.97	–	0	–	0	–
1080p30 + 720p30	170	155	22.78	–	29.97	–	0	–	0	–
1080p30 - Stream and record	160	–	23.29	22.79	–	–	0	0	–	–
1080p30 + 720p30 – Dual stream + record	185	170	23.21	21.89	29.97	30.16	0	0	0	0

Table 6.2: Latency, average FPS, and dropped frames values in different test cases.

video playback in any of the considered scenarios. Single-stream 720p30 can deliver the target 30 FPS, so it can be stated that it works well in real-time. Single-stream 1080p@30 has a resultant average of 23.61 FPS which is only a little lower yet is still high enough to perceive smooth video in higher resolution. In dual-streams, the 720p streams will always have near perfect frame rates of about 29.97 -30 FPS whereas the 1080p streams are usually around 22-23 FPS. A slight loss in FPS is noticed when streaming and recording are conducted at the same time, especially 1080p streams, to which streaming and recording FPS of 23.29 FPS and 22.79 FPS are recorded. Both streams share resources and are robust in the pipeline even in the toughest dual-stream and record setting, showing that the frame rates are stable.

In all test cases, the number of dropped frames is zero, in both the dual-stream and simultaneous recording cases. The fact that the pipeline does not experience frame drops proves that the pipeline is capable of controlling buffering, scheduling, and transmission of data without excessive utilization of system resources. This becomes a crucial need with UAVs operations as it is always required to have reliable and constant video feedback to be used to navigate and gain situational awareness.

Generally, the results of the streaming performance indicate that the proposed framework provides a balanced trade-off between the latency, frame rate, and reliability. Although higher resolution, streaming, and recording features add to the latency and slight decrease in FPS, the system has been able to provide stable video without loss of frames. These findings affirm the appropriateness of the suggested

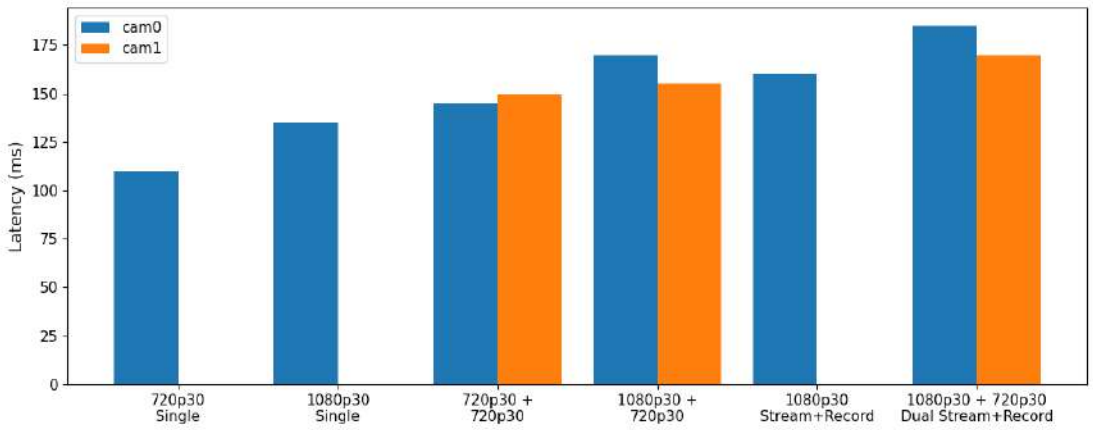

Figure 6.8: Latency vs Test Case graph

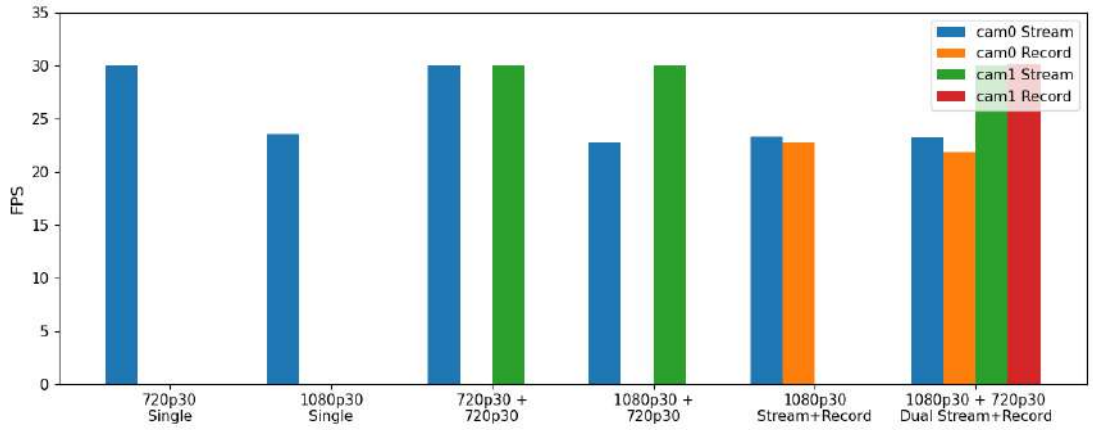

Figure 6.9: FPS vs Test Case graph

multi-stream video transmission architecture in real-time UAV implementations in different contexts of operation.

6.3 Stress Testing and System Limits

The stress testing is done to test the resiliency, scaling and functionality of the proposed multi-stream video transmission system when it is subject to more and more challenging conditions. In contrast to normal performance evaluation, stress testing is designed to test the behavior of the system around both the computational and thermal limits, which is important in the deployment of UAVs when resource requirements are constrained and operations are prolonged. The subsections that follow examine how the scaling of the resolution, frame rate, bit rate and combined

long-duration workload affects the work performance of the system.

Four baseline stress tests were performed: CPU-only, memory-focused, and a combined stress workload representing worst-case contention across compute and memory subsystems. Each of the tests were carried for 120 seconds and the effective or average CPU Utilization was calculated.The (Table 6.3) show how resource utilization varies under each stress profile. It also details the commands used for conducting the tests as well. The combined stress test produced the highest sustained load and serves as an upper bound for system capacity under non-streaming conditions. To put the real limits in life, the combined stress work was exercised together with the dual-stream transmission pipeline. This stress + streaming model is a simulation of an environment that includes a UAV, which sends video and does compute operations on board. The system logs have been confirmed to be stable throughout the period of the test and the Functional Validation of the UAV working was also tested In total, the stress-ng experiments provide a consistent evidence of the boundaries of the system stability and prove the fact that the provided streaming architecture may be used even under the condition of the increased computational pressure.

Test Type	**stress-ng Command**	**Duration (s)**	**Avg CPU (%)**
CPU Stress	stress-ng –cpu 4	120	64:2
Memory Stress	stress-ng –vm 2 –vm-bytes 60%	120	40.1
Combined Stress	stress-ng –cpu 4 –vm 2	180	78.8

Table 6.3: Stress-ng Based System Stress Test Results

7 Discussion and Future Scope

The chapter talks about the findings on the implementation and testing of the proposed real-time multi-stream video transmission system on autonomous UAV platforms. The discussion ponders on the performance realized, design choices and limitations witnessed. Moreover, the possible enhancements and future research opportunities are provided to make the system more applicable and robust.

7.0.1 Discussion

The main aim of this thesis was to architect, implement and test a real time two stream video transmission system that can work within resource limited UAV platforms. The findings discussed in the two chapters above show that this has been effectively accomplished. The proposed system was demonstrated to be efficient in transmitting a number of video streams taking into consideration real-time constraints and the stable functioning of the system through systematic performance analysis.

The effectiveness of hardware-accelerated video encoding with NVENC by NVIDIA on the Jetson Nano platform is one of the main results of this work. In all the experimented conditions such as single-stream, dual-stream, recording-enabled, and high-load, the CPU usage was kept within acceptable bounds. The enough headroom in computational power was maintained even in worst-case conditions, a key factor in the UAV platforms that will be required to perform the flight control, navigation, and perception system simultaneously. This proves that the encoding of video to specific hardware has been very beneficial in efficiency of the system as opposed to the software based encoding methods. It was also found that the use of GPUs can be predicted to go up as the amount of video workload like higher resolutions, frame rates, and bitrates is increased. This elastic scaling is desirable since it enables the system designers to provide estimates of the resources needed by the system and also the ability to set up the system according to the mission-specific constraints. Notably, the stress test did not show any sudden deterioration of performance or encoder instability, which means the system architecture is sound across a wide range of operation conditions. The other critical observation is related to the stability of the system when it is operated for a long time. The stress tests (over time) indicated that the system would not exhibit any noticeable performance variability over time but showed no signs of memory leakage, thermal throttling or the slowing down of performance. The latter is especially applicable to UAV missions which need to be streamlined with video over long durations, like surveillance, inspection, or search-and-rescue missions.

Nevertheless, some weaknesses were also found in the assessment. The experiments were performed under controlled conditions, with constant network conditions, which might not be entirely applicable to actual wireless communication situation in the real world. Issues like the loss of packets, fluctuating latency, and changes in bandwidth may cause a considerable effect on the performance of video streaming within an outdoor UAV deployment. Also, the test has been done regarding one embedded platform, and the outcomes can change when implemented on other hardware-based configurations or highly developed Jetson modules. On the whole, the results discussion proves that the suggested dual-stream video transmission model is efficient and stable within the limits that were investigated. The results confirm the design decisions of this thesis and give a good basis of future improvements.

7.0.2 Improvements

Despite the fact that the adopted system can be observed to work well, there are a number of improvements that can be made that will make the system to be more flexible, resilient, and applicable in real life situation. One of the potential areas of improvement is adaptive streaming mechanisms. It is currently configured in static terms the parameters of streaming, such as resolution, frame rate, and bitrate. The adaptive control can be introduced to the performance at different bandwidth or computer load to improve performance under real-time network and system conditions. The other aspect that we require to relax is our network resilience. The current implementation assumes rather a stable communication connection. Some mechanisms such as forward error correction, or retransmission plans or dynamic bit rate control would be a significant enhancement in achieving robustness in the unreliable wireless world. Such developments would be of particular assistance to the UAV missions that are long range. Another opportunity that can be improved is the system level optimization. The overhead can also be reduced by the intellectualization of the buffer handling, pipeline management as well as the memory allocation though the CPU usage remains at a safe level. Additionally, by being more synchronized with the flight control stack of the UAV, the video streaming can probably be synchronized with other operations on the board. Software engineering The streaming pipe might be decoupled and abstracted to allow it to become more maintainable and extensible.

7.0.3 Future Scope

The research can be expanded in scope even further than the prototype that was created in the future, and offers several research and development opportunities. One of the extensions that would be important would be to implement and test the presented framework on more powerful embedded platforms, e.g., more recent Jetson modules. It would allow trying out more sophisticated multi-camera setups and more complex processing pipelines, including real-time computer vision and inference with

deep learning besides streaming. The other orientation that should be tackled is a combination of adaptive and intelligent streaming approaches. Machine learning used as decision mechanisms can be used to optimize streaming parameters based on the mission goal, the network environment, and the system status. The strategies can be used to make things efficient and reliable in volatile situations. The future work could also be found in the analysis and optimization of end to end latency. Though the utilization of resources and stability was the primary consideration that was taken into account by this thesis, there exist smaller measures of latency that can assist with the realization of whether the system is suitable during time sensitive UAV operations. This would entail the capture-to-display latency measurements and influence of different encoding and transport settings.

Furthermore, it is an interesting research issue to be investigated by extending the framework to make multi-UAV scenarios possible. New challenges would arise in the area of synchronization and scale with the video streams being synchronized and bandwidth controlled on many aerial platforms communicating with a ground station or a cloud infrastructure. Finally, field testing is a highly significant aspect in practice in the future. The fact that the system was applied to a real UAV and the industrial cameras were implemented and the efficiency of the implementation was tested at the real flight conditions, including movements, environment conditions, and wireless distraction, would also prove the applicability of the proposed solution to the real world.

8 Conclusion

This thesis was the design, implementation and analysis of a real time multi stream video transmission model of autonomous unmanned aerial vehicles (UAVs). The overall objective was to develop an efficient and scalable architecture that can be utilized to transmit a few video streams in real-time and at the same time achieve the computational and power limitations of an embedded system. The proposed system can simultaneously stream two videos with a possibility of record as well and real-time performance is consistent under a range of operating conditions. As the results of the experiment say, the CPU use does not change significantly in the normal dual-stream mode but only by a significant margin once the recording and the high-load modes are enabled. This confirms that encoding video to dedicated devices takes significantly less processing hardware and CPU resources remain to be utilized by other UAV capabilities, such as navigation, control and perception. It was also established that the hardware encoder could be used successfully without saturation, and the exhaustion of the software encoder to a higher resolution, frame rate as well as bit rate meant that the software encoder was used accordingly.

Extensive tests of the performance of the system including stress testing, and long duration tests were done to ensure the robustness and reliability of the system. The system remained stable when there was heavy load and in heavy load conditions and performance was not noticed to decrease, memory leaks or thermal throttling was not observed. These findings show that the proposed architecture is appropriate in the scenario where there is a long UAV operation and the transmission of video should be available and predictable all the time.Other notable design ideas of embedded UAV systems that were also highlighted in this thesis are the unified memory usage, hardware-software co-design, and trade-offs between video quality and system resource utilization. The results provide an insight into the importance of using platform-based hardware acceleration in achieving the effective real-time execution using resource-restricted platforms.

This paper has generally offered a practical and experimental solution to real-time multi-stream video transmission on UAVs. The planned framework provides a solid foundation to the future extensions including the adaptive streaming, the combination to the onboard perception algorithms and real flight scenario application. This thesis helps in advancing the state of the art of the UAV communication systems since it demonstrates that the high-quality multi-camera video streaming is a proven solution to an embedded platform and can be utilized in the development of more flexible and independent aerial platforms.

Bibliography

[1] Alvestrand, H.: Webrtc 1.0: Real-time communication between browsers. RFC 8825 (2021), standards Track

[2] Battseren, B., Harradi, R., Kilic, F., Hardt, W.: Automated Power Line Inspection. Technische Universität Chemnitz, Fakultät für Informatik (2020)

[3] Bluenviron: Mediamtx documentation: Real-time media router. `https://github.com/bluenviron/mediamtx` (2026), accessed: Jan. 14, 2026

[4] Burks, S.D., Doe, J.M.: Gstreamer as a framework for image processing applications in image fusion. In: Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2011. vol. 8064, pp. 180–186. SPIE (2011)

[5] Calhoun, G.L., Draper, M.H., Nelson, J.T., Ruff, H.A.: Advanced display concepts for uav sensor operations: Landmark cues and picture-in-picture. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting. vol. 50, pp. 121–125. SAGE Publications Sage CA: Los Angeles, CA (2006)

[6] Chodorek, A., Chodorek, R.R., Sitek, P.: Uav-based and webrtc-based open universal framework to monitor urban and industrial areas. Sensors 21(12), 4061 (2021)

[7] Colomina, I., Molina, P.: Unmanned aerial systems for photogrammetry and remote sensing: A review. ISPRS Journal of Photogrammetry and Remote Sensing 92, 79–97 (2014)

[8] Cramer, M., Przybilla, H.J., Zurhorst, A.: Uav cameras: Overview and geometric calibration benchmark. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 42, 85–92 (2017)

[9] Cui, H., Zhao, Z., Zhang, F.: Research on panorama generation from a multi-camera system by object-distance estimation. Applied Sciences 13(22), 12309 (2023)

[10] Fotouhi, A., Qiang, H., Ding, M., Hassan, M., Giordano, L.G., Garcia-Rodriguez, A., Yuan, J.: Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges. IEEE Communications Surveys & Tutorials 21(4), 3417–3442 (2019)

[11] Genie Doc: Gstreamer. `https://geniedoc.blogspot.com/p/gstreamer.html` (2024), accessed: 2026-01-02

[12] Goldberger, L., Gonzalez-Hirshfeld, I., Nelson, K., Metha, H., Mei, F., Tomlinson, J., Schmid, B., Tagestad, J.D.: High resolution image products acquired from mid-sized uncrewed aerial systems: Addressing challenges of flying higher, faster, and longer (2023)

[13] GStreamer Project: GStreamer Application Development Manual. The GStreamer Team (2024), accessed: 2026-01-02

[14] Guo, H., Tian, B., Yang, Z., Chen, B., Zhou, Q., Liu, S., Nahrstedt, K., Danilov, C.: Deepstream: Bandwidth efficient multi-camera video streaming for deep learning analytics. arXiv preprint arXiv:2306.15129 (2023)

[15] Guo, H., Yao, S., Yang, Z., Zhou, Q., Nahrstedt, K.: Crossroi: Cross-camera region of interest optimization for efficient real time video analytics at scale. In: Proceedings of the 12th ACM Multimedia Systems Conference. pp. 186–199 (2021)

[16] Hayat, S., Yanmaz, E., Muzaffar, R.: Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. IEEE Communications Surveys & Tutorials 18(4), 2624–2661 (2016)

[17] Heller, A., Harradi, R., Hardt, W.: Hangar system for unmanned aerial vehicle autonomous missions. In: 2024 International Symposium ELMAR. pp. 291–294. IEEE (2024)

[18] ITU-T: H.264: Advanced video coding for generic audiovisual services. Recommendation H.264 (2019), accessed: Jan. 14, 2026

[19] Jacobsen, K.: Development of digital aerial cameras. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives 38 (2010), Nr. 1W17 38(1W17) (2010)

[20] Javed, S., Hassan, A., Ahmad, R., Ahmed, W., Ahmed, R., Saadat, A., Guizani, M.: State-of-the-art and future research challenges in uav swarms. IEEE Internet of Things Journal 11(11), 19023–19045 (2024)

[21] Kacianka, S., Auton, T.: Adaptive video streaming for uav networks. `https://dblp.org/rec/conf/mmsys/KaciankaH15` (2015), presented at MoVid@MMSys2015, ACM Transactions on Cyber-Physical Systems

[22] Kazanzides, P., Vagvolgyi, B.P., Pryor, W., Deguet, A., Leonard, S., Whitcomb, L.L.: Teleoperation and visualization interfaces for remote intervention in space. Frontiers in Robotics and AI 8, 747917 (2021)

[23] Kilic, F., Hassan, M., Hardt, W.: Prototype for multi-uav monitoring–control system using webrtc. Drones 8(10), 551 (2024)

[24] Lee, D.R., Kim, S., Yoon, N., Seo, W., Kim, H.: A multi-stage approach to uav detection, identification, and tracking using region-of-interest management and rate-adaptive video coding. Applied Sciences 14(13), 5559 (2024)

[25] Meng, X., Wang, W., Leong, B.: Skystitch: A cooperative multi-uav-based real-time video surveillance system with stitching. In: Proceedings of the 23rd ACM international conference on Multimedia. pp. 261–270 (2015)

[26] Meuel, H., Schmidt, J., Munderloh, M., Ostermann, J.: Region of interest coding for aerial video sequences using landscape models. Advanced Video Coding for Next-Generation Multimedia Services p. 51 (2012)

[27] Mittal, S.: A survey on optimized implementation of deep learning models on the nvidia jetson platform. Journal of Systems Architecture 97, 428–442 (2019)

[28] Murillo, S.G., Chen, C., Jenkins, D.: Webrtc-http egress protocol (whep). `https://datatracker.ietf.org/doc/draft-ietf-wish-whep/03/` (2025), iETF Internet-Draft, Work in Progress. Accessed: Jan. 14, 2026

[29] Nex, F., Remondino, F.: Uav for 3d mapping applications: a review. Applied Geomatics 6(1), 1–15 (2014)

[30] Nuñez, L., Toasa, R.M.: Performance evaluation of rtmp, rtsp and hls protocols for iptv in mobile networks. In: 2020 15th Iberian conference on information systems and technologies (CISTI). pp. 1–5. IEEE (2020)

[31] NVIDIA: Getting started with jetson nano developer kit. `https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit` (2024), accessed: 2026-01-02

[32] Olson, D.O., Anderson, J.V.: Review on unmanned aerial vehicles, remote sensors, imagery processing, and their applications in agriculture. Agronomy Journal 113(2), 971–992 (2021)

[33] Ortega, L.D., Loyaga, E.S., Cruz, P.J., Lema, H.P., Abad, J., Valencia, E.A.: Low-cost computer-vision-based embedded systems for uavs. Robotics 12(6), 145 (2023)

[34] Pahl, C.: Containerization and the paas cloud. IEEE Cloud Computing 2(3), 24–31 (2015)

[35] Rea, D.J., Seo, S.H.: Still not solved: A call for renewed focus on user-centered teleoperation interfaces. Frontiers in Robotics and AI 9, 704225 (2022)

[36] Recchiuto, C.T., Sgorbissa, A., Zaccaria, R.: Visual feedback with multiple cameras in a uavs human–swarm interface. Robotics and Autonomous Systems 80, 43–54 (2016)

[37] Reolink: Rtsp vs rtmp: What's the difference? `https://reolink.com/blog/rtsp-vs-rtmp/` (2024), accessed: 2025-01-30

[38] ResearchGate: Rtmp streaming session (figure 1). `https://www.researchgate.net/figure/RTMP-Streaming-Session_fig1_272376917` (2015), accessed: 2025-12-04

[39] Sacoto-Martins, R., Madeira, J., Matos-Carvalho, J.P., Azevedo, F., Campos, L.M.: Multi-purpose low latency streaming using unmanned aerial vehicles. In: 2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP). pp. 1–6. IEEE (2020)

[40] Schulzrinne, H.: Rtp: A transport protocol for real-time applications. RFC1889 (1996)

[41] Seo, S.H., Rea, D.J., Wiebe, J., Young, J.E.: Monocle: Interactive detail-in-context using two pan-and-tilt cameras to improve teleoperation effectiveness. In: 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). pp. 962–967 (2017)

[42] Siemiatkowska, B., Stecz, W.: A framework for planning and execution of drone swarm missions in a hostile environment. Sensors 21(12), 4150 (2021)

[43] Stephan, M., Battseren, B., Tudevdagva, U.: Autonomous unmanned aerial vehicle development: Mavlink abstraction layer. In: International Symposium on Computer Science, Computer Engineering and Educational Technology (ISCSET-2020). pp. 45–49 (2020)

[44] Torres-Rua, A., Nieto, H., Parry, C., Elarab, M., Collatz, W., Coopmans, C., McKee, L., McKee, M., Kustas, W.: Inter-comparison of thermal measurements using ground-based sensors, uav thermal cameras, and eddy covariance radiometers. In: Proceedings of SPIE - The International Society for Optical Engineering. vol. 10664, p. 106640E (2018), epub 2018 Jul 16

[45] Tudevdagva, U., Battseren, B., Hardt, W., Blokzyl, S., Lippmann, M.: Unmanned aerial vehicle-based fully automated inspection system for high voltage transmission line. In: Proceedings on the 12th International Forum on Strategic Technology IEEE conference, IFOST 2017. pp. 300–305 (2017)

[46] Ullah, H., Zia, O., Kim, J.H., Han, K., Lee, J.W.: Automatic 360 mono-stereo panorama generation using a cost-effective multi-camera system. Sensors 20(11), 3097 (2020)

[47] Van Beeck, K., Tuytelaars, T., Scarramuza, D., Goedemé, T.: Real-time embedded computer vision on uavs: Uavision2018 workshop summary. In: European Conference on Computer Vision. pp. 3–10. Springer (2018)

[48] Wierzbicki, D.: Multi-camera imaging system for uav photogrammetry. Sensors 18(8), 2433 (2018)

[49] Xie, S., Deng, G., Lin, B., Jing, W., Li, Y., Zhao, X.: Real-time object detection from uav inspection videos by combining yolov5s and deepstream. Sensors 24(12), 3862 (2024)

[50] Yang, S., Scherer, S.A., Yi, X., Zell, A.: Multi-camera visual slam for autonomous navigation of micro aerial vehicles. Robotics and Autonomous Systems 93, 116–134 (2017)

[51] Zarándy, Á., Nemeth, M., Nagy, Z., Kiss, A., Santha, L., Zsedrovits, T.: A real-time multi-camera vision system for uav collision warning and navigation. Journal of Real-Time Image Processing 12(4), 709–724 (2016)

[52] Zhang, Z., Zhu, L.: A review on unmanned aerial vehicle remote sensing: Platforms, sensors, data processing methods, and applications. Drones 7(6), 398 (2023)

9 References: Professorship of Computer Engineering

2 Battseren, B., Harradi, R., Kilic, F., Hardt, W.: Automated Power Line Inspection. Technische Universit¨at Chemnitz, Fakult¨at f¨ur Informatik (2020)

17 Heller, A., Harradi, R., Hardt, W.: Hangar system for unmanned aerial vehicle autonomous missions. In: 2024 International Symposium ELMAR. pp. 291–294. IEEE (2024)

23 Kilic, F., Hassan, M., Hardt, W.: Prototype for multi-uav monitoring–control system using webrtc. Drones 8(10), 551 (2024)

43 Stephan, M., Battseren, B., Tudevdagva, U.: Autonomous unmanned aerial vehicle development: Mavlink abstraction layer. In: International Symposium on Computer Science, Computer Engineering and Educational Technology (ISCSET-2020). pp. 45–49 (2020)

45 Tudevdagva, U., Battseren, B., Hardt, W., Blokzyl, S., Lippmann, M.: Unmanned aerial vehicle-based fully automated inspection system for high voltage transmission line. In: Proceedings on the 12th International Forum on Strategic Technology IEEE conference, IFOST 2017. pp. 300–305 (2017)

TECHNISCHE UNIVERSITÄT CHEMNITZ

Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Sibi Vorname: Josey Mol geb. am: 01.08.1997 Matr.-Nr.: 726634	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 29.01.2026 Unterschrift: ……

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

CSR-24-01 Seyhmus Akaslan, Ariane Heller, Wolfram Hardt, Hardware-Supported Test Environment Analysis for CAN Message Communication, Juni 2024, Chemnitz

CSR-24-02 S. M. Rizwanur Rahman, Wolfram Hardt, Image Classification for Drone Propeller Inspection using Deep Learning, August 2024, Chemnitz

CSR-24-03 Sebastian Pettke, Wolfram Hardt, Ariane Heller, Comparison of maximum weight clique algorithms, August 2024, Chemnitz

CSR-24-04 Md Shoriful Islam, Ummay Ubaida Shegupta, Wolfram Hardt, Design and Development of a Predictive Learning Analytics System, August 2024, Chemnitz

CSR-24-05 Sopuluchukwu Divine Obi, Ummay Ubaida Shegupta, Wolfram Hardt, Development of a Frontend for Agents in a Virtual Tutoring System, August 2024, Chemnitz

CSR-24-06 Saddaf Afrin Khan, Ummay Ubaida Shegupta, Wolfram Hardt, Design and Development of a Diagnostic Learning Analytics System, August 2024, Chemnitz

CSR-24-07 Túlio Gomes Pereira, Wolfram Hardt, Ariane Heller, Development of a Material Classification Model for Multispectral LiDAR Data, August 2024, Chemnitz

CSR-24-08 Sumanth Anugandula, Ummay Ubaida Shegupta, Wolfram Hardt, Design and Development of a Virtual Agent for Interactive Learning Scenarios, September 2024, Chemnitz

CSR-25-01 Md. Ali Awlad, Hasan Saadi Jaber Aljzaere, Wolfram Hardt, AUTOSAR Software Component for Atomic Straight Driving Patterns, März 2025, Chemnitz

CSR-25-02 Billava Vasantha Monisha, Hasan Saadi Jaber Aljzaere, Wolfram Hardt, Automotive Software Component for QT Based Car Status Visualization, März 2025, Chemnitz

CSR-25-03 Zahra Khadivi, Batbayar Battseren, Wolfram Hardt, Acoustic-Based MAV Propeller Inspection, Mai 2025, Chemnitz

Chemnitzer Informatik-Berichte

CSR-25-04 Tripti Kumari Shukla, Ummay Ubaida Shegupta, Wolfram Hardt, Time Management Tool Development to Support Self-regulated Learning, August 2025, Chemnitz

CSR-25-05 Ambu Babu, Ummay Ubaida Shegupta, Wolfram Hardt, Development of a Retrieval Model based Backend of a Tutoring Agent, August 2025, Chemnitz

CSR-25-06 Shahid Ismail, Ummay Ubaida Shegupta, Wolfram Hardt, Development of a Generative Model based Backend of Tutoring Agent, August 2025, Chemnitz

CSR-25-07 Chaitanya Sravanthi Akula, Ummay Ubaida Shegupta, Wolfram Hardt, Integration of Learning Analytics into the ARC-Tutoring Workbench, August 2025, Chemnitz

CSR-25-08 Jörn Roth, Reda Harradi, Wolfram Hardt, Implementation of a Path Planning Algorithm for UAV Navigation, Dezember 2025, Chemnitz

CSR-25-09 Alhassan Khalil, Reda Harradi, Stephan Rupf, Wolfram Hardt, Development of an Automation Framework for 1D Measurement, Dezember 2025, Chemnitz

CSR-26-01 Vismay Gunda, Shadi Saleh, Wolfram Hardt, Cloud-Based AI Solutions for Ensuring Data Quality in Predictive Models, Februar 2026, Chemnitz

CSR-26-02 Sami Mansoor Alavi, Shadi Saleh, Wolfram Hardt, Continuous Integration for Cloud-Based Swarm Farming Applications, Februar 2026, Chemnitz

CSR-26-03 Sarah Onyinyechi Obasi, Shadi Saleh, Wolfram Hardt, Cloud-Based AI for Data Completeness Analysis and Improvement in Predictive Modeling, März 2026, Chemnitz

CSR-26-04 Atul Chandra Nath, Reda Harradi, Wolfram Hardt, GPS-based UAV Precision Landing, April 2026, Chemnitz

CSR-26-05 Josey Mol Sibi, Batbayar Battseren, Wolfram Hardt, Real Time Multi Stream Video Transmission in Autonomous UAV, Mai 2026, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz