Government College of Engineering (GCOEJ), Jalgaon

(An Autonomous Institute of Government of Maharashtra)

(Academic Year 2023-24)

DEPARTMENT OF COMPUTER ENGINEERING

PROJECT REPORT ON

REAL TIME AUDIO ANALYSIS OF ACOUSTIC SIGNALS

Submitted by:

1. Bhangale Diksha Shalik (2041007)
2. Bhargav Shamuvel Gurav (2041009)
3. Ghodke Ajay Satish (2041025)
4. Siddhi Kakpure (2041032)

Guardinger Technologies(OPC) Pvt Ltd

Industrial Guide

Prof. S. D.Cheke

PROJECT GUIDE

Government College of Engineering (GCOEJ), Jalgaon

(An Autonomous Institute of Govt. of Maharashtra)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the *Project* report, "Real time analysis of underwater acoustic signals", which is being submitted here with for the award of *LY* Computer Engineering (7th Semester) is the result of the work completed by Bhangale Diksha (2041007), Bhargav Gurav (2041009), Ghodke Ajay (2041025), Kakpure Siddhi (2041032) under my supervision and guidance within offline mode of classes of the institute, in the academic year 2023-24

Prof. S. D. Cheke
Project Guide

Prof. D. V. Chaudhari
Head of Department

DECLARATION

We hereby declare that the Project entitled **"Real time analysis of underwater acoustic signals"** was carried out and written by us under the guidance of **Prof. S. D. Cheke and Head of Department**, department of Computer Engineering, Government College of Engineering, Jalgaon. This work has not been previously formed for the award of any degree or diploma or certificate nor has been submitted elsewhere for the award of any degree.

Place: Jalgaon
Date:

Bhangale Diksha Shalik	2041007
Bhargav Shamuvel Gurav	2041009
Ghodke Ajay Satish	2041025
Kakpure Siddhi Narhari	2041032

INDEX

ABSTRACT

This final year project presents a cutting-edge solution for real-time analysis of acoustic waves, leveraging the power of NVIDIA Jetson Nano and edge computing technologies. Acoustic wave analysis plays a crucial role in various applications such as industrial monitoring, environmental sensing, and security systems. Traditional approaches often suffer from latency and scalability issues, prompting the need for efficient edge computing solutions.

The project integrates a sophisticated array of microphones with the Jetson Nano development kit to capture and process acoustic signals in real-time. The Jetson Nano, a high-performance embedded computing platform, provides the necessary computational power for on-the-fly signal processing and analysis. The use of edge computing enhances the system's responsiveness by minimizing the reliance on cloud-based processing, making it suitable for applications requiring low-latency and offline capabilities.

Key components of the project include signal preprocessing, feature extraction, and algorithms to interpret useful data. The system's architecture prioritizes efficiency and real-time responsiveness, making it adaptable to diverse environments and scenarios. The use of the Jetson Nano as an edge computing platform enables deployment in resource-constrained environments without compromising on computational capabilities.

This research contributes to the growing field of edge computing and real-time signal analysis, providing a versatile and robust solution for acoustic wave applications.

CHAPTER 1

INTRODUCTION

1.1 PROJECT DESCRIPTION

The vast and dynamic nature of the maritime environment presents significant challenges for effective security and surveillance. Underwater acoustic signals hold immense potential for gathering crucial information about underwater activities, contributing to enhanced maritime safety and security. This project aims to develop a real-time system for acquiring and analyzing underwater acoustic signals using the NVIDIA Jetson Nano kit, specifically dedicated to supporting the efforts of maritime security forces.

1.2 MOTIVATION

Real-time analysis of underwater acoustic signals offers several advantages for maritime security:

- **Enhanced Situational Awareness:** Detecting, classifying, and tracking underwater objects and activities, such as unauthorized vessels, divers, and underwater drones, provides critical information for improved decision-making and resource allocation.
- **Improved Border Security:** Monitoring underwater activity can assist in detecting and preventing illegal border crossings and smuggling operations, enhancing national security.
- **Protection of Critical Infrastructure:** Analyzing acoustic signals can help identify threats to underwater infrastructure like pipelines and cables, ensuring their safe operation and preventing potential environmental damage.

- **Search and Rescue Operations:** Real-time detection of underwater distress signals can significantly increase the efficiency and success rate of search and rescue operations, saving lives.

1.3 PROBLEM STATEMENT

Traditional methods for underwater acoustic analysis often rely on expensive and complex equipment, limiting their accessibility and deployment in real-world scenarios. Additionally, many systems lack real-time capabilities, hindering their ability to respond effectively to evolving maritime threats

1.4 OBJECTIVE

This project addresses these limitations by developing a cost-effective, real-time system for underwater acoustic analysis, specifically tailored to enhance maritime security. The system will aim to:

- **Increase Accessibility**: Utilize readily available hardware and open-source software for broader deployment and ease of use by maritime security personnel.
- **Enable Real-time Monitoring**: Acquire, process, and visualize data continuously, allowing for immediate response to critical underwater events.
- **Achieve High Accuracy**: Employ advanced signal processing and machine learning techniques for accurate detection and classification of underwater objects and activities relevant to maritime security.
- **Develop Security-specific Features:** Integrate functionalities like target tracking, threat assessment, and information sharing to support effective security operations.

1.5 PROJECT MODULES

The project will be divided into the following modules:

1.5.1 Hardware Setup: Select and configure the Jetson Nano, microphone, and any necessary data acquisition system for robust deployment in maritime environments.

1.5.2 Software Development: Develop software for signal acquisition, feature extraction, classification, and visualization, utilizing Python libraries and C language for implementation.

1.5.3 Machine Learning Model Training: Train and optimize machine learning models on relevant datasets for accurate underwater object and activity detection in maritime security contexts.

1.5.4 User Interface Development: Design and implement a user-friendly interface for displaying results, including spectrograms, waveforms, and target identification labels.

1.5.5 Integration with Communication Systems: Develop interfaces for integrating the system with existing communication systems used by maritime security forces for seamless data sharing and collaboration.

1.5.6 Testing and Evaluation: Rigorously test the system in simulated and real-world maritime scenarios to evaluate its performance in terms of accuracy, real-time capabilities, and effectiveness in supporting maritime security operations.

1.6 PROJECT REQUIREMENTS

1.6.1 Hardware

Specifications of Jetson nano kit :

Specifications	Details
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included yet)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30(H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I2C, I2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector.

1.7 ORGANIZATION OF PROJECT

This report is currently divided into chapters, each dealing with different aspects of the project. Each chapter has a short introduction, explaining the subject of each chapter, and then the details, each module is explained separately. The following is a short overview of each of the chapters:

Chapter 2, Outlines some of the research made on the project in the beginning. More research was made as this project report was being developed, as new areas had to be investigated. This research is summarized in the various chapters according to the different modules.

Chapter 3, will specify the approaches and methodologies used in the project.

Chapter 4, will specify implementation design used in the project.

Chapter 5, will define the general architectural flow of the system.

Chapter 6, will shows the Results of the project.

Chapter 7, Conclusion and the future scope of the project is discussed here

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

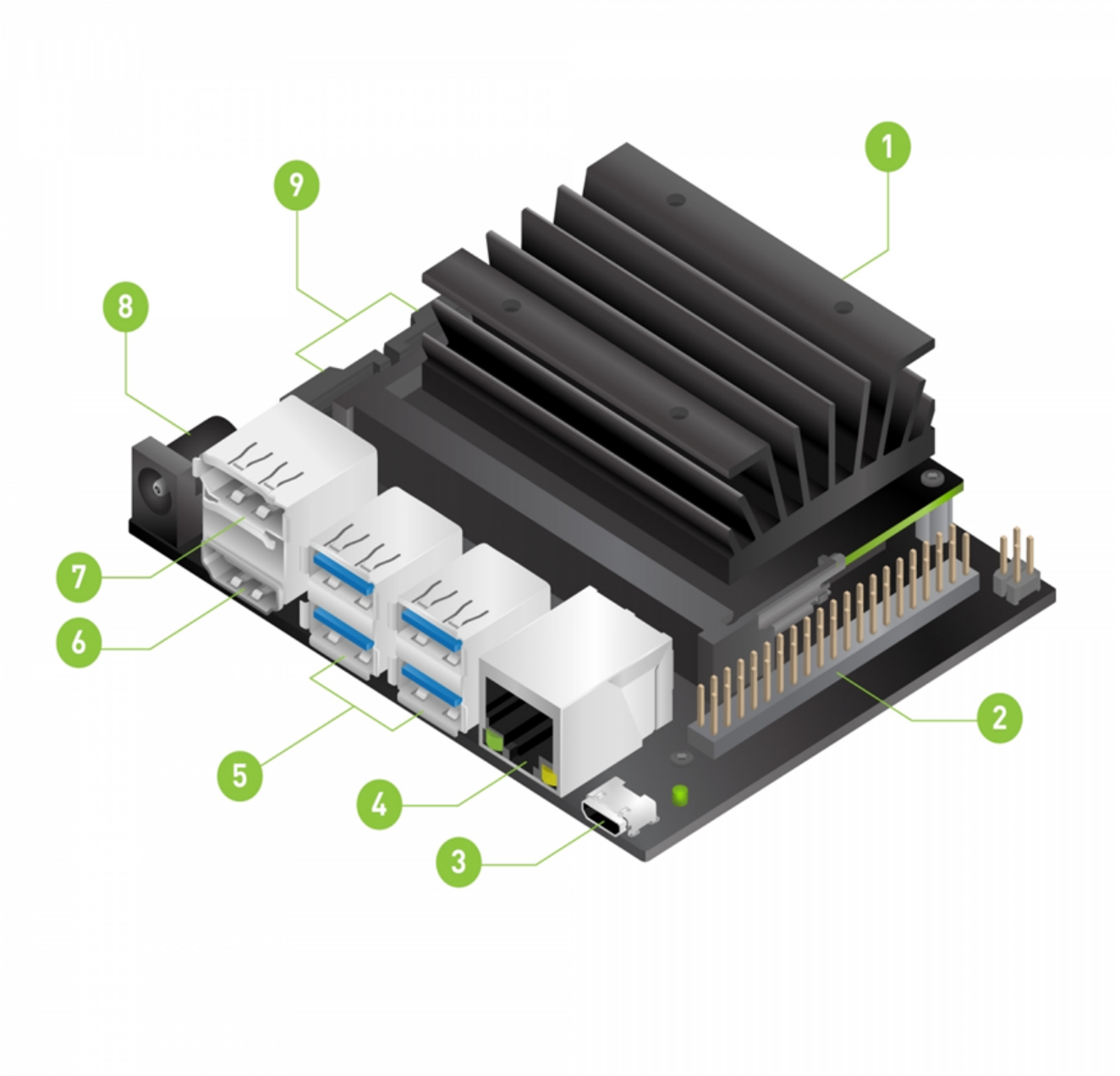
The vast and dynamic nature of the maritime environment presents significant challenges for effective security and surveillance. Underwater acoustic signals hold immense potential for gathering crucial information about underwater activities, contributing to enhanced maritime safety and security. This literature survey will explore recent advancements in real-time underwater acoustic signal analysis, focusing on techniques relevant to enhancing maritime security and safety.

The domain analysis that we have done for the project mainly involved data generation from three different channels, studying Mosquitto MQTT protocol and implementing it, installation and working with NVIDIA Jetson Nano Kit, and generating results from the useful data.

2.1.1 NVIDIA Jetson Nano Kit

What is Jetson Nano Kit?

The NVIDIA Jetson Nano™ Developer Kit is a mini powerhouse designed to bring the power of artificial intelligence (AI) to the edge. It can be described as a tiny computer specifically built for running AI applications, not just browsing the web or playing games.



Below is the labeled diagram of Jetson Nano Kit

where the labels are denoted as:

1. microSD card slot for main storage
2. 40-pin expansion header
3. Micro-USB port for 5V power input, or for Device Mode
4. Gigabit Ethernet port

5. USB 3.0 ports (x4)
6. HDMI output port
7. DisplayPort connector
8. DC Barrel jack for 5V power input
9. MIPI CSI-2 camera connectors

Below are the features which makes the Jetson Nano Kit special:

Small & Mighty:

- **Pocket-sized Portability:** Weighing just 45 grams and measuring 70mm x 45mm, the Jetson Nano fits comfortably in your palm. Its compact size makes it ideal for embedded projects like smart cameras and robots, where space is a constraint.
- **Performance Packed:** Don't be fooled by its size. The Jetson Nano boasts a quad-core ARM Cortex-A57 CPU and a powerful 128-core NVIDIA Maxwell GPU with 472 GFLOPs. This means it can handle complex AI models with ease, delivering smooth performance even for demanding tasks.
- **Memory & Storage:** 4GB LPDDR4 RAM ensures smooth multitasking, while 16GB eMMC storage provides ample space for your operating system and applications. Expand your storage further with a microSD card for larger datasets and models.

AI Ready:

- **JetPack™ SDK:** Forget the hassle of setting up your AI development environment. The JetPack SDK comes pre-loaded on the Jetson Nano, providing a comprehensive toolkit for building and deploying AI applications.
- **Deep Learning Frameworks:** TensorFlow, PyTorch, Caffe – the Jetson Nano supports all the popular deep learning frameworks, giving you the flexibility to choose the one that best suits your project's needs.

- **Beyond Deep Learning:** The JetPack SDK goes beyond just deep learning. It includes libraries for computer vision (OpenCV, CUDA), robotics (ROS), and multimedia (FFmpeg), enabling you to tackle diverse AI challenges.

Open & Easy to Use:

- **Familiar Operating System:** The Jetson Nano runs on Linux, an operating system familiar to many developers, making it easy to get started.
- **Extensive Resources:** Worried about getting stuck? Don't be! NVIDIA provides comprehensive documentation, tutorials, and community forums to guide you through every step of your AI journey.
- **Open-Source Platform:** The Jetson Nano is not just a tool, it's a community. Its open-source nature encourages collaboration and customization, allowing you to build upon existing projects and contribute your own innovations.

Affordable & Efficient:

- **Budget-Friendly:** The Jetson Nano is one of the most affordable AI development platforms available. This makes it accessible to individual developers and makers who want to experiment with AI without breaking the bank.
- **Low Power Consumption:** Unlike bulky AI servers, the Jetson Nano operates efficiently, consuming minimal power. This not only saves you money on electricity but also makes it suitable for battery-powered projects.

Technical Specification:

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz

Why

Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

NVIDIA Jetson Nano kit only?

Using the Jetson Nano kit for underwater acoustic wave detection can be advantageous for several reasons:

- **GPU Acceleration:** The Jetson Nano comes equipped with a 128-core Maxwell GPU, providing significant parallel processing power. This is particularly beneficial for computationally intensive tasks such as signal processing and machine learning, which are crucial for analyzing acoustic wave data.

- **AI and Deep Learning Capabilities:** The Jetson Nano supports popular AI frameworks like TensorFlow and PyTorch. This allows you to implement and deploy deep learning models for acoustic wave detection. Neural networks can be trained to recognize patterns and features in underwater acoustic signals, aiding in the identification of specific events or anomalies.

- **Compact Size:** The Jetson Nano is a compact and energy-efficient device, making it suitable for deployment in various environments, including underwater applications where space and power constraints may be significant considerations.

- **GPIO Pins for Sensor Integration:** The Jetson Nano provides GPIO pins that allow you to interface with various sensors and peripherals. This is essential for connecting and integrating underwater acoustic sensors or transducers.

- **Community and Support:** The Jetson Nano has a growing community of developers, and NVIDIA provides comprehensive documentation and support. Access to a community can be valuable when facing challenges or seeking advice on specific applications, such as underwater acoustic wave detection.

- **Compatibility with JetPack SDK:** The Jetson Nano is part of the NVIDIA JetPack SDK ecosystem, which includes software libraries and APIs optimized for AI and GPU-accelerated tasks. This SDK streamlines the development and deployment of applications, including those focused on signal processing and machine learning.

2.1.2 Mosquitto

What is Mosquitto?

Mosquitto is an open-source message broker or server that implements the MQTT (Message Queuing Telemetry Transport) protocol. MQTT is a lightweight and widely used messaging protocol designed for low-bandwidth, high-latency, or unreliable networks, making it suitable for various IoT (Internet of Things) and messaging applications. It acts as a central hub for managing the communication between MQTT clients. It enables devices or applications to publish messages to specific topics and subscribe to receive messages from those topics. It plays a key role in facilitating communication in MQTT-based systems.

Synopsis:

mosquitto [-c *config file*] [-d | --daemon] [-p *port number*] [-v]

Options:

- -c,
  --config-file

Load configuration from a file. If not given, then the broker will listen on port 1883 bound to the loopback interface, and the default values as described in mosquitto.conf(5) are used.

- -d,
  --daemon

Run **mosquitto** in the background as a daemon. All other behaviour remains the same.

- -p,
  --port

Listen on the port specified. May be specified up to 10 times to open multiple sockets listening on different ports.

- -v,
  --verbose

Use verbose logging. This is equivalent to setting log_type to all in the configuration file. This overrides and logging options given in the configuration file.

Mosquitto.conf

Mosquitto.conf is the configuration file for mosquitto. This file can reside anywhere as long as mosquitto can read it. By default, mosquitto does not need a configuration file and will use the default values listed below.

Synopsis:

Mosquitto.conf

General Options:

acl_file *file path*

Set the path to an access control list file. If defined, the contents of the file are used to control client access to topics on the broker.

If this parameter is defined then only the topics listed will have access. Topic access is added with lines of the format:

topic [read|write|readwrite|deny] <topic>

The access type is controlled using "read", "write", "readwrite" or "deny". This parameter is optional (unless <topic> includes a space character) - if not given then the access is read/write. <topic> can contain the + or # wildcards as in subscriptions. The "deny" option can used to explicitly deny access to a topic that would otherwise be granted by a broader read/write/readwrite statement. Any "deny" topics are handled before topics that grant read/write access.

The first set of topics are applied to anonymous clients, assuming allow_anonymous is true. User specific topic ACLs are added after a user line as follows:

```
user <username>
```

The username referred to here is the same as in password_file. It is not the clientid.

It is also possible to define ACLs based on pattern substitution within the topic. The form is the same as for the topic keyword, but using pattern as the keyword.

```
pattern [read|write|readwrite|deny] <topic>
```

The patterns available for substition are:

- %c to match the client id of the client
- %u to match the username of the client

The substitution pattern must be the only text for that level of hierarchy. Pattern ACLs apply to all users even if the "user" keyword has previously been given.

Example:

```
pattern write sensor/%u/data
```

Allow access for bridge connection messages:

```
pattern write $SYS/broker/connection/%c/state
```

If the first character of a line of the ACL file is a # it is treated as a comment.

If per_listener_settings is *true*, this option applies to the current listener being configured only. If per_listener_settings is *false*, this option applies to all listeners.

Reloaded on reload signal. The currently loaded ACLs will be freed and reloaded. Existing subscriptions will be affected after the reload.

See also https://mosquitto.org/documentation/dynamic-security/

allow_anonymous [true | false]

Boolean value that determines whether clients that connect without providing a username are allowed to connect. If set to *false* then another means of connection should be created to control authenticated client access.

Defaults to *false*, unless no listeners are defined in the configuration file, in which case it is set to *true*, but connections are only allowed from the local machine.

If per_listener_settings is *true*, this option applies to the current listener being configured only. If per_listener_settings is *false*, this option applies to all listeners.

Important

In version 1.6.x and earlier, this option defaulted to *true* unless there was another security option set.

Reloaded on reload signal.

allow_duplicate_messages [true | false]

This option is deprecated and will be removed in a future version. The behavior will default to true.

If a client is subscribed to multiple subscriptions that overlap, e.g. foo/# and foo/+/baz , then MQTT expects that when the broker receives a message on a topic that matches both subscriptions, such as foo/bar/baz, then the client should only receive the message once.

Mosquitto keeps track of which clients a message has been sent to in order to meet this requirement. This option allows this behaviour to be disabled, which may be useful if you have a large number of clients subscribed to the same set of topics and want to minimise memory usage.

It can be safely set to *true* if you know in advance that your clients will never have overlapping subscriptions, otherwise your clients must be able to correctly deal with duplicate messages even when then have QoS=2.

Defaults to *true*.

This option applies globally.

Reloaded on reload signal.

allow_zero_length_clientid [true | false]

MQTT 3.1.1 and MQTT 5 allow clients to connect with a zero length client id and have the broker generate a client id for them. Use this option to allow/disallow this behavior. Defaults to true.

See also the auto_id_prefix option.

If per_listener_settings is *true*, this option applies to the current listener being configured only. If per_listener_settings is *false*, this option applies to all listeners.

Reloaded on reload signal.

auth_plugin_deny_special_chars [true | false]

If *true* then before an ACL check is made, the username/client id of the client needing the check is searched for the presence of either a '+' or '#' character. If either of these characters is found in either the username or client id, then the ACL check is denied before it is sent to the plugin.

This check prevents the case where a malicious user could circumvent an ACL check by using one of these characters as their username or client id. This is the same issue as was reported with mosquitto itself as CVE-2017-7650.

If you are entirely sure that the plugin you are using is not vulnerable to this attack (i.e. if you never use usernames or client ids in topics) then you can disable this extra check and hence have all ACL checks delivered to your plugin by setting this option to *false*.

Defaults to *true*.

Applies to the current authentication plugin being configured.

Not currently reloaded on reload signal.

auto_id_prefix *prefix*

If allow_zero_length_clientid is *true*, this option allows you to set a string that will be prefixed to the automatically generated client ids to aid visibility in logs. Defaults to auto-.

If per_listener_settings is *true*, this option applies to the current listener being configured only. If per_listener_settings is *false*, this option applies to all listeners.

Reloaded on reload signal.

autosave_interval *seconds*

The number of seconds that mosquitto will wait between each time it saves the in-memory database to disk. If set to 0, the in-memory database will only be saved when mosquitto exits or when receiving the SIGUSR1 signal. Note that this setting only has an effect if persistence is enabled. Defaults to 1800 seconds (30 minutes).

This option applies globally.

Reloaded on reload signal.

autosave_on_changes [true | false]

If *true*, mosquitto will count the number of subscription changes, retained messages received and queued messages and if the total exceeds autosave_interval then the in-memory database will be saved to disk. If *false*, mosquitto will save the in-memory database to disk by treating autosave_interval as a time in seconds.

This option applies globally.

Reloaded on reload signal.

check_retain_source [true | false]

This option affects the scenario when a client subscribes to a topic that has retained messages. It is possible that the client that published the retained message to the topic had access at the time they published, but that access has been subsequently removed. If check_retain_source is set to true, the default, the source of a retained message will be checked for access rights before it is republished. When set to false, no check will be made and the retained message will always be published.

This option applies globally, regardless of the per_listener_settings option.

clientid_prefixes *prefix*

This option is deprecated and will be removed in a future version.

If defined, only clients that have a clientid with a prefix that matches clientid_prefixes will be allowed to connect to the broker. For example, setting "secure-" here would mean a client "secure-client" could connect but another with clientid "mqtt" couldn't. By default, all client ids are valid.

This option applies globally.

Reloaded on reload signal. Note that currently connected clients will be unaffected by any changes.

`connection_messages [ true | false ]`

If set to *true*, the log will include entries when clients connect and disconnect. If set to *false*, these entries will not appear.

This option applies globally.

Reloaded on reload signal.

CHAPTER 3

TOOLS SPECIFICATION

Tools to be used : Jetson nano developer kit, SD card, Desktop, DisplayPort cable, 5V DC 4 Pin Cooling fan, Audio card, Mic, 5V 2.5A Power Adaptor

Jetson nano Developer kit :-

1. **Specifications of Jetson nano kit :**

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

2.Installation :

Steps :-

1. Get an SD card at least 16GB, on which libraries and all will be downloaded.
2. Get it to the laptop to flash it (plug in the card to the laptop).
3. Go to nvidia website and download latest JetPack nano developer kit SD card image
4. For windows download SD card formatter from the same site, and also etcher.
5. First open the sd card formatter and format the sd card and then using etcher use downloaded image zip file and flash sd card.
6. Insert SD card to nano .
7. connect mouse, keyboard, desktop to it and boot the system on desktop.
8. open terminal in that system (linux) and check swap area (it should be 971 i.e 1GB)
9. run these commands to make it upto 4GB to make better space for our projects

    9.1) sudo systemctl disable nvzramconfig
    9.2) . sudo fallocate -l 4G /mnt/4GB.swap
    9.3) sudo chmod 600 /mnt/4GB.swap
    9.4) sudo mkswap /mnt/4GB.swap
    9.5) sudo vi /etc/fstab
    make it --INSERT--
    /mnt/4GB.swap swap swap defaults 0 0
    then close using :wq
    Then reboot and thus 4GB swap will be available.

Here is the Video for installation process click here.

CHAPTER 4

CALCULATION

Calculating flush rate for storage :-

Given, No. of Channels (for sample collection) = 3
No. of samples/channel sec = 12000
Size for one sample = 32 bits
(4 bytes float32)

Thus, Total samples / sec = 3 × 12000
= 36000 samples/sec

Total bits required / sec = 36000 × 32
= 1152000 bits/sec

Total bytes / sec = 11520000 ÷ 8
= 144000 bytes / sec

i.e. **we will be having 144000 bytes of data per sec getting stored**

If we take 64 GB SD card, and out of it let's take 32 GB is taken for sample data storage,

Then,

Flush rate = 32000000000 ÷ 144000
= 222222.222222 sec

= 222222.222222 ÷ 60
= 3703.703 minutes

= 3703.703 ÷ 60
≈ 61.7283 hrs

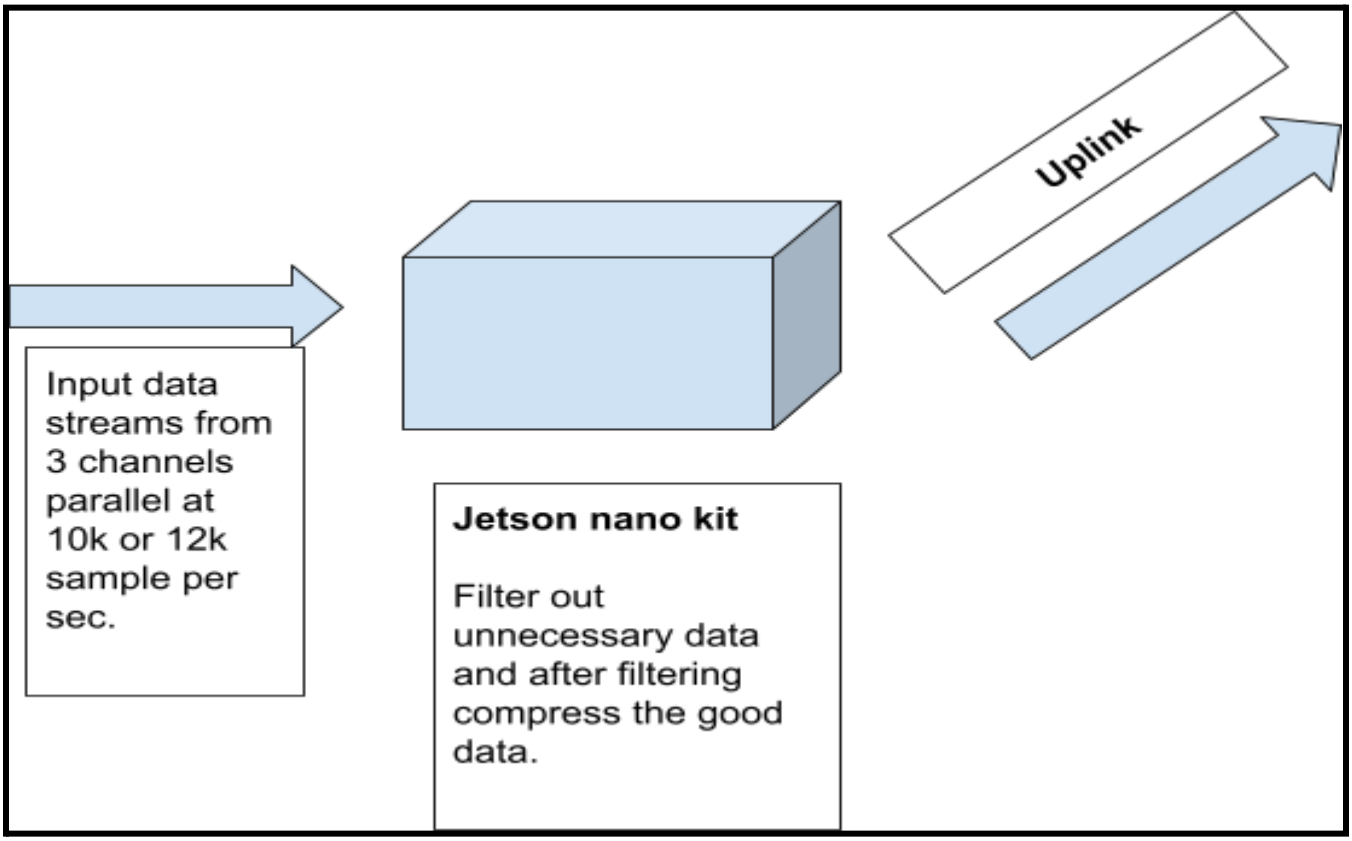
i.e. **The flush rate for the storage will be nearly after each 61.7283 hrs.**

CHAPTER 5

ARCHITECTURE

Action :- You will be getting 3 streams of input parallel from 3 channels at a rate of approximately 10k or 12k samples per second.

Workflow :-
1. Acquisition of data
2. Filtering out unnecessary data input
3. After filtering compress the good data
4. uplink the good data for storage



Date : 10/11/2023
Minutes of Meeting :
1. Data acquisition
2. Downsampling

3. Store data

CHAPTER 6
CHALLENGES FACED

1. Getting the exact match of jetson to work according to problem statements and other requirements.
2. Calculating the accurate flush rate and space requirements and accordingly ordering the best match of SD card to work on.
3. Burning the jetson nano system image on SD card.
4. Getting the system ready to work by installing the system on it by SD card.
5. Interfacing of audio input device with jetson nano.
6. Creating a pipeline for data acquisition in the correct format and with the correct number of channels.
7. Writing script for creating a pipeline for data transferring between publisher and subscriber for MQTT protocol.

CHAPTER 7
PROJECT FLOW

Action	Status
1. Project scope definition	Completed
2. Requirement gathering	Completed
3. Data acquisition (11 KHz sampling rate)	Completed
4. Data compressing (5.5 KHz after downsampling)	Completed
5. Data filtering	In Progress
6. Mosquitto Installation	Completed
7. Pushing data to mosquitto topic	Completed
8. Subscriber script	Completed
9. Installation at required site	Not Started
10. Identifying the potential tasks to integrate the solution with	Not Started

CHAPTER 8

CONCLUSION

Our project harnessing the power of edge computing through Jetson Nano, coupled with Mosquitto.conf for underwater acoustic signal analysis, has yielded remarkable results so far. The Jetson Nano's edge computing capabilities, combined with Mosquitto.conf optimizations, facilitated real-time and precise signal processing at the source. This edge computing approach enhances the system's efficiency, making it well-suited for applications such as real time analysis of data signals and saving bandwidth issues which include corruption of signal while transmitting which is overcomed by Edge Computing , also the power consumption power of previous technology. Despite encountered challenges, addressing them and gathering insights during the testing phase will guide future enhancements. This project establishes a robust framework for edge computing-driven advancements in underwater signal processing.