

Contents

Abstract	i
Acknowledgement	iii
List of Figures	vii
1 Introduction	1
2 Motivation	2
3 Literature Review	3
4 Methodology	5
4.1 System Design and Requirements Analysis:	5
4.2 Hardware Selection and Integration:	6
4.2.1 Locomotion and Chassis Design:	6
4.2.2 Power Supply and Distribution:	6
4.2.3 Sensor and Camera Integration;	6
4.2.4 Wired Communication Implementation:	7
4.3 2D Design and Initial Conceptualization:	7
4.4 3D Modeling and Component Design:	8
4.5 Software Development	9
4.5.1 Programming and Implementation:	9
4.5.2 Graphical User Interface (GUI) at the Base Station:	9
4.6 Hardware Testing:	10
4.7 Testing and Performance Evaluation	10
4.7.1 Environmental Sensor Validation:	10

4.7.2	Rover Navigation Testing:	11
4.7.3	Performance Analysis and Optimization	11
5	Implementation	12
5.1	System Architecture(Conceptual Design)	12
5.1.1	BeagleY-AI Board (Central Processing Unit)	12
5.1.2	Environmental sensors	13
5.1.3	Microphone and Speaker	13
5.1.4	IR Camera	13
5.1.5	Motor Driver and Motors	13
5.1.6	Wired Communication with Base Station	13
5.2	Hardware Wiring and Circuit Connections	14
5.2.1	Power Input	14
5.2.2	Base Station Connection	14
5.2.3	Motor Driver and Motors (M1 , M2)	15
5.3	Hardware Assembly and Integration:	15
5.4	Software Deployment and System Integration:	15
5.5	Communication System Design	16
5.6	Testing and Refinements	16
6	Hardware and Softwares	17
6.1	Hardware Components	17
6.1.1	Beagle Y AI Board	17
6.1.2	Arducam 1MX477	18
6.1.3	MQ135 Gas Sensor	18
6.1.4	DHT11 Temperature and Humidity Sensor	19
6.1.5	Slim Mini Audio Speaker (80 Ohm, 24x40mm)	19
6.1.6	InvenSense ICS43434	20
6.2	Software Components	20
6.2.1	Visual Studio Code	20
6.2.2	ROS Construct	20
6.2.3	SolidWorks	21

7	Program	22
7.1	Code for Base station	22
7.2	Code for Rover	33
8	Result	38
8.1	3D model of the rover	38
8.2	Hardware integration in rover	38
8.3	Graphical User Interface at Base station	39
8.4	Relay pipe guided rover	40
9	Scope for future applications	41
9.1	Technological Advancements	41
9.2	Application Expansion	42
9.3	Integration with Other Technology	42
10	Conclusion	43
	References	45

List of Figures

4.1	2D design of rover	7
4.2	Model of lower chassis	8
4.3	Model of upper chassis	8
4.4	Model of rover	8
4.5	GUI	9
4.6	Simple program in Beagle Y AI board	10
5.1	Block diagram	12
5.2	Hardware circuit diagram	14
6.1	Beagle Y AI board	17
6.2	Arducam 1MX477	18
6.3	MQ135-gas sensor	18
6.4	DHT11-temperature and humidity sensor	19
6.5	Slim Mini Audio Speaker (80 Ohm, 24x40mm)	19
6.6	InvenSense ICS43434-microphone	20
8.1	Final 3D model of rover	38
8.2	Hardware integration	39
8.3	GUI at base station	39
8.4	rover moving through pipe	40
8.5	output from beagle board	40

Chapter 1

Introduction

The increasing frequency of mining accidents and natural disasters has necessitated the development of innovative solutions for effective search and rescue operations. In situations where traditional rescue methods may be hindered by debris or confined spaces, robotic technologies offer a viable alternative to enhance safety and efficiency. This project introduces an advanced rover specifically designed to navigate through strategically drilled pipes, providing access to areas that are otherwise challenging for human rescuers.

Equipped with high-resolution cameras and sophisticated environmental sensors, the rover captures real-time images and monitors environmental conditions. Utilizing the BeagleY-AI board for onboard processing, the rover employs advanced image processing algorithms to enhance visibility in low-light and dusty environments, which is critical for assessing the conditions of trapped individuals.

To facilitate effective communication, the rover features a two-way audio system, allowing trapped individuals to interact directly with rescuers. This interaction not only provides emotional support but also enables rescuers to gather vital information, improving the overall effectiveness of the rescue effort.

This project aims to demonstrate the potential of robotics in enhancing the safety and effectiveness of rescue operations. By integrating real-time video transmission, environmental sensing, and communication technologies, the rover serves as a vital resource in improving outcomes in emergency situations, ultimately contributing to the successful recovery of trapped individuals.

Chapter 2

Motivation

The motivation behind this project arises from the significant safety challenges and operational hazards associated with coal mining, particularly in the aftermath of landslides. Landslides in mining areas can create unstable environments, obstruct pathways, and release hazardous gases, making it dangerous and often impossible for personnel to conduct direct inspections. Such conditions call for advanced solutions that can mitigate risks while ensuring effective monitoring and data collection.

Traditional inspection methods are either labor-intensive or unsuitable for confined spaces filled with toxic gases, low visibility, and unstable terrain. Robotic systems offer a promising alternative by enabling remote, real-time assessment without endangering human lives. By deploying a specialized rover that integrates navigation aids, multi-sensor capabilities, and reliable communication systems, this project aims to enhance post-landslide safety protocols.

Moreover, the rising demand for more autonomous and efficient mining solutions emphasizes the need for innovation in this field. The rover's ability to navigate through strategically drilled pipes and perform detailed environmental monitoring could lead to broader applications in mining, rescue operations, and even other high-risk industries. The project is, therefore, driven by a commitment to improving mine safety, increasing efficiency, and minimizing the risks associated with underground inspections.

Chapter 3

Literature Review

The Role of Robotics in Hazardous Environments[1,3]: Robotics has gained significant attention for applications in hazardous and inaccessible areas, where human presence may be unsafe or impractical. Many researchers highlights that autonomous and semi-autonomous robotic systems have been successfully deployed in high-risk areas, such as nuclear plants, deep-sea exploration, and disaster-stricken zones. These robots can perform inspections, monitor environmental conditions, and relay critical information back to human operators. This capability is particularly important in post-landslide scenarios within coal mines, where unstable structures and toxic gases create unsafe conditions for humans. Robotic inspection systems provide a viable solution by reducing risk while allowing continuous data collection to assess stability and safety.

Guidance and Navigation in Confined Space [4]: Navigation in confined and debris-filled spaces, such as those found in coal mines, requires specialized techniques to ensure stability and accuracy, which emphasize the importance of guided systems—rails, tethers, or relay pipes—that allow robots to move safely through narrow and hazardous areas. Relay pipes, specifically, provide a path that helps maintain the stability and orientation of the rover, even in environments with poor visibility or challenging terrain. This technology has been applied in urban search-and-rescue missions, showing success in guiding robotic platforms through narrow spaces while preventing deviations that could lead to navigation errors or loss of control.

Environmental Monitoring and Gas Detection in Mining Applications [2,7,8]: One of the most critical safety concerns in coal mines is the presence of toxic or explo-

sive gases, including methane and carbon monoxide. The mining industry frequently encounters these gases, especially following landslides, as they can accumulate in newly created cavities or pockets. Gas sensors, such as the MQ-series (e.g., MQ-7 for carbon monoxide and MQ-4 for methane), have become standard for real-time monitoring in underground environments, which shows that these sensors are effective in identifying dangerous gas concentrations and can be integrated into mobile robotic systems for continuous monitoring. Additionally, infrared (IR) gas sensors have been used for detecting methane due to their accuracy and sensitivity. Integrating such sensors into an inspection rover provides a reliable means of assessing air quality and warning operators of dangerous levels of toxic gases.

Ethernet Communication in Robotics [9,14,10]: Ethernet technology has become a cornerstone in industrial robotics, providing a standardized, high-bandwidth medium for data exchange between system components. The adoption of Ethernet-based protocols, enables seamless integration with existing network infrastructures, enhancing interoperability and simplifying system architecture. This integration allows for centralized control and data collection, improving efficiency and reducing latency in robotic operations.

Chapter 4

Methodology

4.1 System Design and Requirements Analysis:

The development of the rover began with an in-depth analysis of the operational requirements needed for effective deployment in confined underground environments. The primary objective was to create a compact yet efficient system capable of maneuvering through relay pipes drilled in collapsed coal mines. Key considerations included designing a structure that ensures seamless mobility, integrating a robust wired communication system for uninterrupted data transmission, and incorporating environmental sensors for real-time monitoring of hazardous gases, temperature, and humidity levels. The rover was also required to support real time video transmission and bidirectional audio communication, enabling real-time interaction between trapped workers and rescue operators. To enhance usability, a graphical user interface (GUI) was developed at the base station, providing a centralized platform for monitoring and controlling the rover. This interface displays the real-time video feed, environmental sensor readings, and system diagnostics, ensuring operators have all necessary data at a glance. Additionally, the GUI includes interactive controls, allowing operators to navigate the rover remotely and make crucial decisions based on live feedback. These requirements guided the selection of appropriate hardware components, communication technologies, and software development strategies to ensure the rover's reliability in rescue operations.

4.2 Hardware Selection and Integration:

4.2.1 Locomotion and Chassis Design:

The chassis design focused on creating a lightweight yet structurally stable frame that could navigate through confined spaces while withstanding underground conditions. The rover was designed to move within a relay pipe, requiring a compact and durable body that would prevent obstruction while maintaining balance. High-torque DC motors were selected to drive the wheels, ensuring sufficient force for movement while operating efficiently within the power constraints .

4.2.2 Power Supply and Distribution:

The rover is powered by a battery-based power supply, ensuring mobility without dependence on external power sources. A lithium-polymer (LiPo) battery was selected for its high energy density, lightweight characteristics, and ability to supply a stable voltage output. A power management circuit was designed to regulate voltage levels, preventing over voltage or under voltage conditions that could impact the performance of motors and sensors. The distribution system was optimized to efficiently supply power to all components, ensuring prolonged operational capability while maintaining safety measures such as short-circuit protection and thermal management.

4.2.3 Sensor and Camera Integration;

To enhance situational awareness in the underground environment, the rover was equipped with environmental monitoring sensors and a high-resolution camera system. Gas sensors were integrated to detect hazardous gases. Additionally, temperature and humidity sensors were added to provide real-time environmental data to rescue operators. A high-resolution camera was mounted on the rover to capture live video footage, which was transmitted to the base station.

4.2.4 Wired Communication Implementation:

A critical aspect of the rover's design was the implementation of wired communication technology to enable real-time data transmission. This system facilitated real-time control of the rover, bidirectional audio communication, and live video streaming. Control signals from the base station were transmitted through the wired network, allowing precise maneuverability and real-time response from the rover. The wired approach eliminated concerns regarding wireless signal interference or loss of connectivity in underground environments.

4.3 2D Design and Initial Conceptualization:

Following the component selection, a hand-drawn 2D design was created to visualize the layout and arrangement of the components within the available space. The hand-drawn design (Figure 4.1) provided an understanding of the positioning of each component, ensuring efficient space usage and proper weight distribution for balanced navigation. This initial conceptualization allowed for quick revisions and adjustments, ensuring flexibility in the design process and ensured that all components would fit within the confined space of the pipe, with an emphasis on stability to minimize tipping risks.

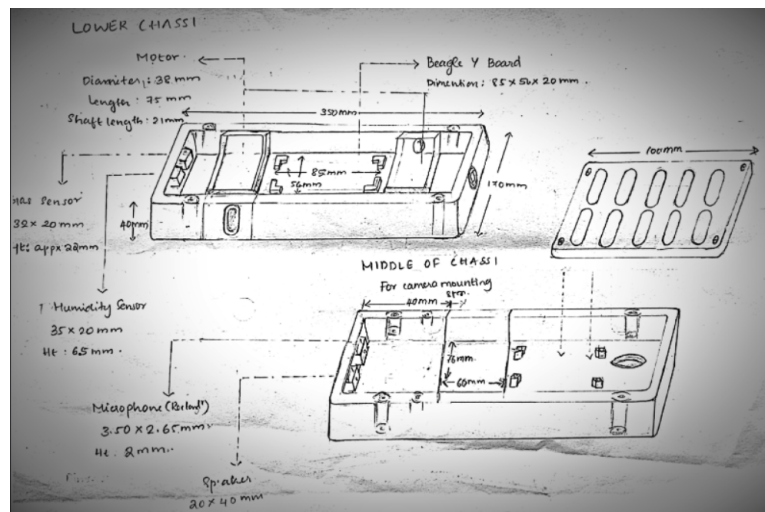


Figure 4.1: 2D design of rover

4.4 3D Modeling and Component Design:

After the 2D design, the rover was transitioned into a 3D model using CAD software, SolidWorks. The 3D modeling phase provided a more detailed visualization of the rover's components in three dimensions, ensuring their fit and arrangement as shown in Figure 4.2, Figure 4.3 and Figure 4.4. This step helped to identify potential issues such as component overlap or misalignment. Adjustments were made to optimize component sizes and positions to ensure effective navigation within the pipe while maintaining the rover's structural integrity and balance. The 3D model allowed for accurate representation of the rover's physical design, aiding in the evaluation of space utilization and structural efficiency.

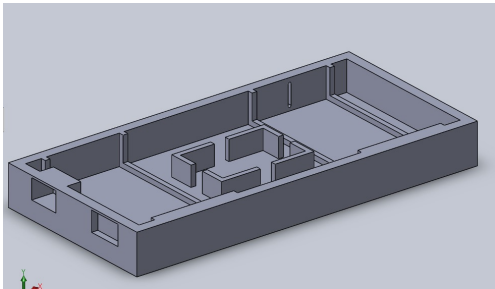


Figure 4.2: Model of lower chassis

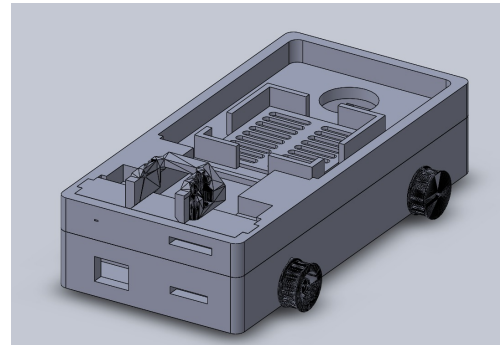


Figure 4.3: Model of upper chassis

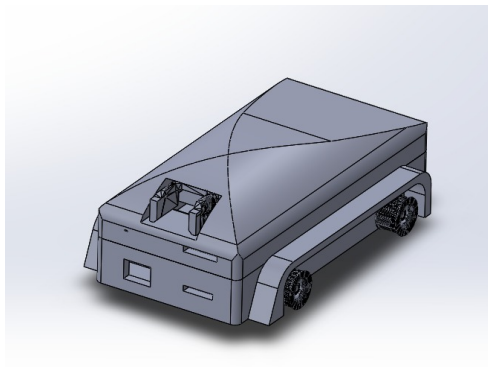


Figure 4.4: Model of rover

4.5 Software Development

4.5.1 Programming and Implementation:

The software architecture was developed to handle real-time video processing, sensor data acquisition, and remote control functionalities. Python is used to program the system, with multi-threading techniques employed to manage simultaneous data transmission, motor control, and environmental monitoring. The rover's software was designed to interface with the base station, where a Graphical User Interface (GUI) was developed to display live video feeds, sensor readings, and control parameters.

4.5.2 Graphical User Interface (GUI) at the Base Station:

A dedicated Graphical User Interface (GUI) was developed at the base station to facilitate seamless interaction with the rover. The GUI serves as the central control hub where all data from the rover, including live video feeds, environmental sensor readings, and system status, are displayed in real-time. The interface allows rescue operators to monitor underground conditions, analyze images processed by the rover, and adjust control parameters as needed. Additionally, the GUI (Figure 4.5) provides interactive controls for operating the rover, such as adjusting its movement and sensor data.

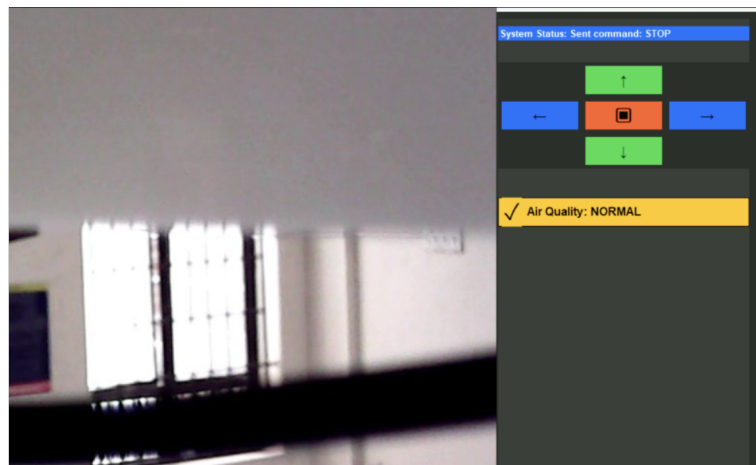
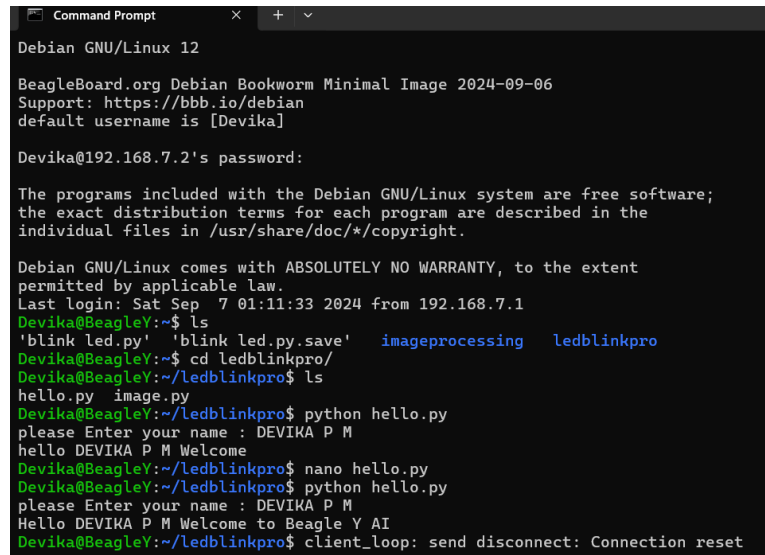


Figure 4.5: GUI

4.6 Hardware Testing:

Before full-scale deployment, individual hardware components were tested to ensure their functionality. The motors, sensors, and communication modules were evaluated under controlled conditions to verify their responsiveness and reliability. The Beagle Y AI board was used to execute simple test programs that assessed sensor data acquisition and motor operation. These preliminary tests confirmed that the rover could process environmental inputs and execute movement commands accurately. The wired communication system was tested separately to measure latency, and overall stability in data transmission. Control signals, bidirectional audio communication, and real-time video streaming were evaluated under simulated underground conditions. The results indicated that the system could handle continuous data flow without delays, ensuring effective remote operation.



```
Command Prompt
Debian GNU/Linux 12

BeagleBoard.org Debian Bookworm Minimal Image 2024-09-06
Support: https://bbb.io/debian
default username is [Devika]

Devika@192.168.7.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Sep  7 01:11:33 2024 from 192.168.7.1
Devika@BeagleY:~$ ls
'blink led.py'  'blink led.py.save'  imageprocessing  ledblinkpro
Devika@BeagleY:~$ cd ledblinkpro/
Devika@BeagleY:~/ledblinkpro$ ls
hello.py  image.py
Devika@BeagleY:~/ledblinkpro$ python hello.py
please Enter your name : DEVIKA P M
hello DEVIKA P M Welcome
Devika@BeagleY:~/ledblinkpro$ nano hello.py
Devika@BeagleY:~/ledblinkpro$ python hello.py
please Enter your name : DEVIKA P M
Hello DEVIKA P M Welcome to Beagle Y AI
Devika@BeagleY:~/ledblinkpro$ client_loop: send disconnect: Connection reset
```

Figure 4.6: Simple program in Beagle Y AI board

4.7 Testing and Performance Evaluation

4.7.1 Environmental Sensor Validation:

The accuracy of the environmental sensors was tested in a controlled setup by exposing them to varying conditions. Gas sensors were calibrated to detect different gases, under different environmental conditions to ensure precise readings.

4.7.2 Rover Navigation Testing:

A simulated test environment was created to evaluate the rover's movement through a relay pipe. The rover was tested for stability, maneuverability, and response to control signals. Adjustments were made to improve its movement and ensure efficient navigation within confined spaces. The rover's response to control signals was tested to measure real-time communication effectiveness, ensuring minimal latency in command execution.

4.7.3 Performance Analysis and Optimization

To ensure the rover's reliability and efficiency, a series of performance tests were conducted, focusing on communication latency, power efficiency, structural durability, and environmental adaptability. The rover's response time to control signals was analyzed to minimize latency, with software optimizations implemented for real-time communication. Its maneuverability was tested in confined spaces, leading to adjustments in motor control algorithms for smoother turns and better path optimization.

Chapter 5

Implementation

5.1 System Architecture(Conceptual Design)

The block diagram(Figure 5.1) represents the working architecture of the rover system, which is designed to navigate and communicate through a wired connection

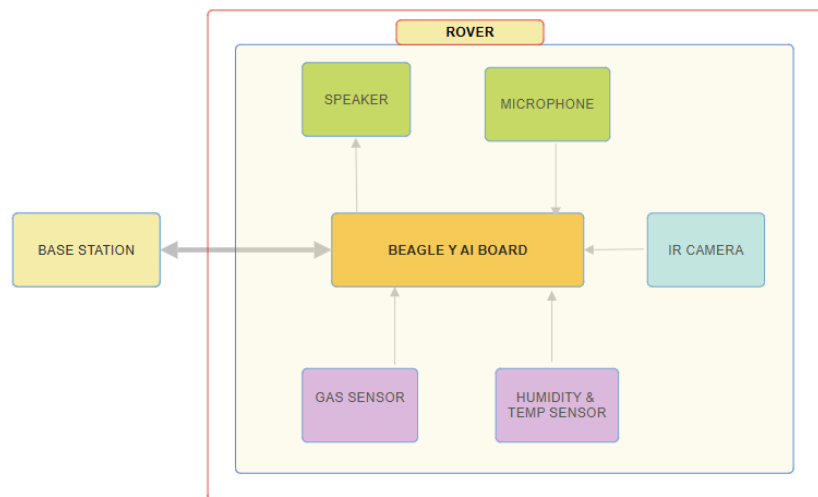


Figure 5.1: Block diagram

5.1.1 BeagleY-AI Board (Central Processing Unit)

The BeagleY-AI board is responsible for processing data, and controlling various hardware components. It receives input from sensors and the microphone, processes the data, and sends output commands to the motors, speaker, and base station. The board also facilitates real-time wired communication with the base station.

5.1.2 Environmental sensors

Gas Sensor Detects air quality and sends data to the BeagleY-AI board. Humidity and Temperature Sensor, Measures environmental conditions and provides necessary information for analysis.

5.1.3 Microphone and Speaker

Microphone Captures real-time audio from the rover's surroundings and sends it to the BeagleY-AI board. Speaker Plays back audio signals received from the base station or generated by the system. These components enable bi-directional voice communication.

5.1.4 IR Camera

The IR Camera captures real-time video of the rover's surroundings. The video feed is processed by the BeagleY-AI board and transmitted to the base station via a wired connection.

5.1.5 Motor Driver and Motors

The BeagleY-AI board controls the motors via the motor driver module. The motor driver converts the control signals into high-power signals that can drive the motors (M1 and M2). This enables the rover to move forward, backward, left, and right based on commands from the base station or programmed logic.

5.1.6 Wired Communication with Base Station

The BeagleY-AI board sends sensor data, video feed, and audio to the base station. It also receives control commands from the base station, such as motor movement instructions and audio playback. This ensures real-time bi-directional communication between the rover and rescue operators.

5.2 Hardware Wiring and Circuit Connections

The Figure 5.2, provides a detailed wiring layout of how the BeagleY-AI board connects to various hardware components in the rover system.

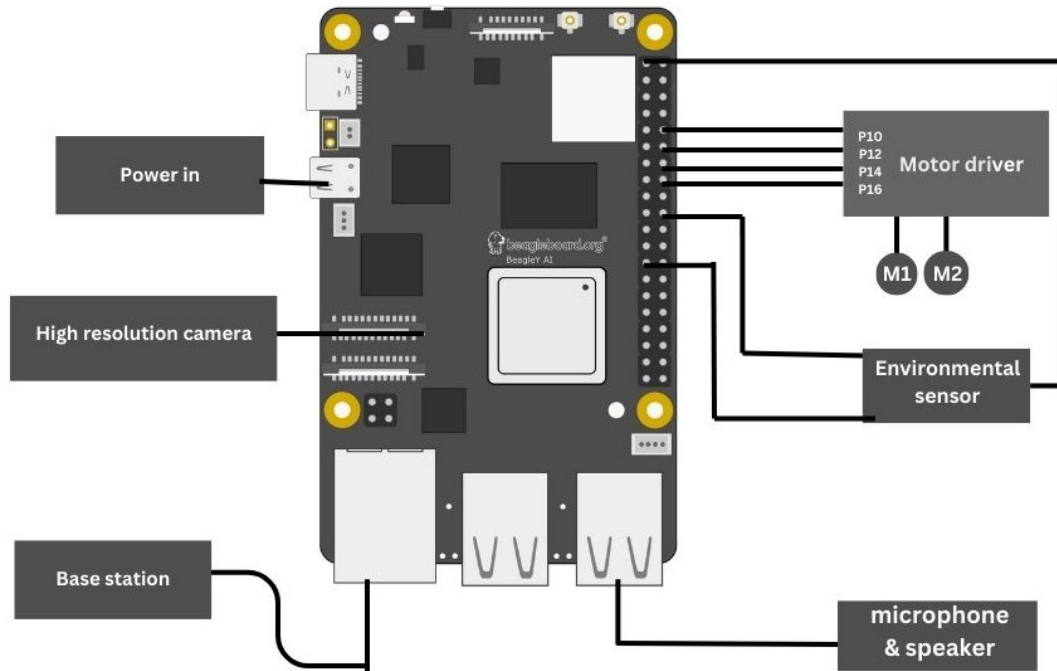


Figure 5.2: Hardware circuit diagram

5.2.1 Power Input

The BeagleY-AI board is powered through a USB or external power supply. This ensures that all components, including the sensors, motors, and communication modules, receive the necessary power.

5.2.2 Base Station Connection

The wired connection between the BeagleY-AI board and the base station allows for real-time data transmission. This includes Bidirectional audio transmission, video and sensor data transmission from rover to base station and control signal transmission from base station to rover.

5.2.3 Motor Driver and Motors (M1 , M2)

The motor driver is connected to the BeagleY-AI board via GPIO pins (P10, P12, P14, and P16). The BeagleY-AI board sends control signals to the motor driver. The motor driver then amplifies the signals and drives two motors (M1 and M2). This allows the rover to move forward, backward, left, or right based on base station commands.

5.3 Hardware Assembly and Integration:

The physical assembly of the rover was carried out by integrating the selected hardware components into the designed chassis. The Beagle Y AI board was securely mounted within the rover's structure, ensuring proper connectivity with other essential components. The high-resolution infrared camera was positioned strategically to capture optimal video feeds for real-time processing.

The environmental sensors, including temperature, humidity, and gas sensors, were installed at appropriate locations to ensure accurate monitoring of underground conditions. Special care was taken to organize the wiring layout efficiently to minimize interference and potential damage during movement.

The rover's mobility system, consisting of motors and chains, was tested for smooth operation. The chassis underwent multiple refinements to ensure optimal balance and stability while navigating confined spaces, such as a relay pipe in a collapsed mine.

5.4 Software Deployment and System Integration:

The software developed in the earlier phase was implemented on the Beagle Y AI board, ensuring real-time control and data management. Multi-threading techniques were used to handle concurrent tasks such as video streaming, sensor data acquisition, and motor control without latency issues. The processed data, including live video feeds and environmental sensor readings, was transmitted to the base station through the wired communication system.

5.5 Communication System Design

The rover is equipped with a wired communication system to enable uninterrupted data transmission. This system established a direct connection between the rover and the base station, ensuring high-speed, interference-free data transfer. It facilitated real-time transmission of video, sensor data, and control signals for precise operation.

5.6 Testing and Refinements

After hardware and software integration, the system was tested in a controlled environment. The rover's movement and response to control signals were evaluated to ensure smooth navigation through confined spaces. Sensor readings were verified, and necessary adjustments were made to improve performance. The wired communication system was also tested for stability, confirming real-time bidirectional data exchange.

Chapter 6

Hardwares and Softwares

6.1 Hardware Components

6.1.1 Beagle Y AI Board

The Beagle Y AI Board is a powerful single-board computer optimized for AI and machine learning tasks, featuring advanced processing capabilities and a variety of I/O interfaces, shown in Figure 6.1 . Equipped with an AI-optimized processing unit, it supports real-time data processing, multimedia functions, and edge computing, making it ideal for applications in robotics, automation, and IoT. The board's high-speed communication capabilities enhance its versatility, allowing it to handle complex tasks like data analysis, sensor integration, and multimedia processing efficiently.

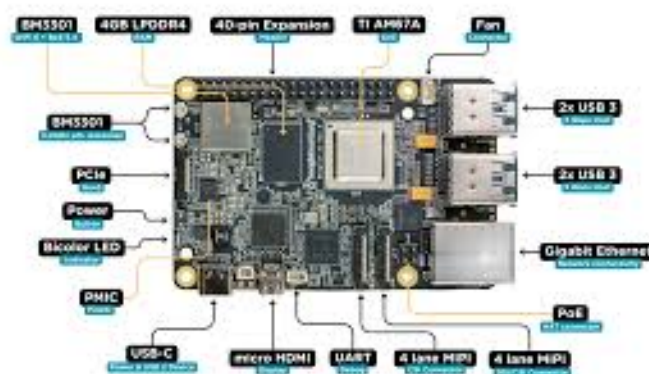


Figure 6.1: Beagle Y AI board

6.1.2 Arducam 1MX477

The Arducam 1MX477 is a high-quality camera module based on the Sony IMX477 image sensor, known for its exceptional low-light performance, high resolution, and adaptability to a range of imaging tasks. Designed primarily for applications needing detailed visuals in challenging conditions, this camera is widely used in robotics, drones, surveillance, and industrial inspections as shown in Figure 6.3. Its capabilities make it ideal for environments where clarity and precision are paramount, such as the interior of a mine.



Figure 6.2: Arducam 1MX477

6.1.3 MQ135 Gas Sensor

The MQ135 Gas Sensor is a sensor module commonly used for air quality monitoring, capable of detecting a range of hazardous gases(Figue 6.3). It is specifically designed to detect gases such as ammonia (NH_3), sulfur, benzene, smoke, Carbon dioxide, and other potentially toxic gases.



Figure 6.3: MQ135-gas sensor

6.1.4 DHT11 Temperature and Humidity Sensor

The DHT11 is a low-cost, digital sensor used to measure temperature and humidity(Figure 6.4). It operates on a supply voltage between 3.5V and 5.5V and communicates with a microcontroller through a single-wire digital signal. The sensor contains a humidity sensing component and a thermistor, converting analog data into a digital signal, which is then processed by the microcontroller.

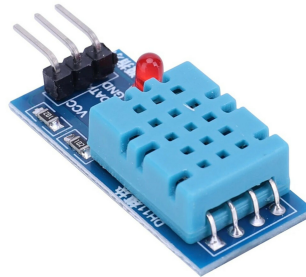


Figure 6.4: DHT11-temperature and humidity sensor

6.1.5 Slim Mini Audio Speaker (80 Ohm, 24x40mm)

The Slim Mini Audio Speaker (80 Ohm, 24x40mm)is a compact speaker designed for space-constrained electronic projects (Figure 6.5). With an impedance of 80 ohms, it is suitable for low-power devices and produces clear sound for basic audio applications, such as voice prompts, alerts, and notifications.



Figure 6.5: Slim Mini Audio Speaker (80 Ohm, 24x40mm)

6.1.6 InvenSense ICS43434

The InvenSense ICS43434 is a high-performance digital MEMS microphone designed for use in various audio applications, including consumer electronics, smartphones, smart speakers, and voice-controlled devices as shown in Figure 6.6. It features a digital I2S output interface, which simplifies the integration of the microphone with digital processing systems, such as microcontrollers or digital signal processors (DSPs).

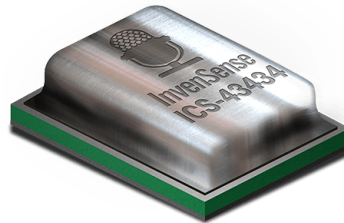


Figure 6.6: InvenSense ICS43434-microphone

6.2 Software Components

6.2.1 Visual Studio Code

VS Code is a popular and lightweight code editor that supports a wide range of programming languages and offers numerous extensions, making it an ideal tool for embedded systems development. It provides features such as syntax highlighting, debugging, version control integration, and a customizable environment, helping developers streamline the coding process. With its support for various extensions and its ability to handle large projects, VS Code is widely used in the development of both simple and complex software systems.

6.2.2 ROS Construct

ROS Construct is a framework designed to simplify the use of the Robot Operating System (ROS) by providing tools for simulation, testing, and integration of robotic systems. It offers a user-friendly interface for configuring and managing ROS packages, hardware interfaces, and simulation environments, making it easier for developers to

design, test, and deploy robots. ROS Construct enhances the development process by allowing for easy simulation and debugging, particularly in the context of robotic applications and autonomous systems.

6.2.3 SolidWorks

SolidWorks is a leading 3D CAD (Computer-Aided Design) software used for designing, modeling, and simulating mechanical parts and assemblies. It is widely used in engineering to create detailed 3D models that can be tested and visualized before actual manufacturing. SolidWorks offers a range of tools for designing complex mechanical systems, performing simulations (such as stress analysis), and generating technical drawings. Its intuitive interface and powerful design capabilities make it a key tool for prototyping and product development across various industries.

Chapter 7

Program

7.1 Code for Base station

```
# final
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import cv2
import numpy as np
import socket
import struct
import threading
import queue
import pyaudio

# Configuration
HOST = '192.168.1.10' # BeagleBone AI IP
VIDEO_PORT = 5000
AUDIO_PORT = 5001
SENSOR_PORT = 5002
MOTOR_AUDIO_PORT = 5003

# UI Configuration
```

```

COLORS = {
    "dark_bg": "#2a2a2a",
    "light_bg": "#3a3a3a",
    "accent": "#2196f3",
    "success": "#4caf50",
    "danger": "#f44336",
    "warning": "#ff9800",
    "text": "#ffffff"
}

class RoverClient:
    def _init_(self, root):
        self.root = root
        self.running = True

        # Video and audio queues
        self.video_queue = queue.Queue(maxsize=2)
        self.audio_queue = queue.Queue(maxsize=10)

        # Network connections
        self.motor_audio_socket = None

        # Initialize GUI
        self.setup_gui()

        # Audio setup
        self.audio = pyaudio.PyAudio()
        self.audio_stream = self.audio.open(
            format=pyaudio.paInt16,
            channels=2,
            rate=44100,
            output=True
        )
        self.input_stream = self.audio.open(

```

```

        format=pyaudio.paInt16,
        channels=2,
        rate=44100,
        input=True,
        frames_per_buffer=1024
    )

    # Start network connections
    self.connect_services()

    # Start processing threads
    threading.Thread(target=self.receive_video, daemon=
        True).start()
    threading.Thread(target=self.receive_audio, daemon=
        True).start()
    threading.Thread(target=self.receive_sensor, daemon=
        True).start()
    threading.Thread(target=self.play_audio, daemon=True).
        start()
    threading.Thread(target=self.send_audio, daemon=True).
        start()

    # Start UI updates
    self.update_video()

def setup_gui(self):
    self.root.title("Rover Control Center")
    self.root.geometry("1280x720")
    self.root.configure(bg=COLORS["dark_bg"])

    # Main container
    main_frame = ttk.Frame(self.root)
    main_frame.pack(expand=True, fill=tk.BOTH, padx=10,
        pady=10)

```

```

# Video panel
video_frame = ttk.Frame(main_frame)
video_frame.pack(side=tk.LEFT, expand=True, fill=tk.
    BOTH)

self.video_label = ttk.Label(video_frame, background="
    black")
self.video_label.pack(expand=True, fill=tk.BOTH)

# Control panel
control_frame = ttk.Frame(main_frame, width=300, style
    ="Control.TFrame")
control_frame.pack(side=tk.RIGHT, fill=tk.Y, padx=10)

# Status display
self.status_label = ttk.Label(
    control_frame,
    text="System Status: Connecting...",
    style="Status.TLabel"
)
self.status_label.pack(pady=10, fill=tk.X)

# Motor controls
self.create_motor_controls(control_frame)

# Sensor display
self.sensor_frame = ttk.Frame(control_frame, style="
    Sensor.TFrame")
self.sensor_frame.pack(pady=20, fill=tk.X)

self.sensor_icon = ttk.Label(
    self.sensor_frame,
    text="    ",

```

```

        font=("Arial", 24),
        style="Sensor.TLabel"
    )
    self.sensor_icon.pack(side=tk.LEFT, padx=5)

    self.sensor_label = ttk.Label(
        self.sensor_frame,
        text="Air Quality: Initializing...",
        style="Sensor.TLabel"
    )
    self.sensor_label.pack(side=tk.LEFT)

# Configure styles
self.configure_styles()

def configure_styles(self):
    style = ttk.Style()
    style.theme_create("custom", settings={
        "TFrame": {"configure": {"background": COLORS["dark_bg"]}},
        "TLabel": {"configure": {
            "background": COLORS["dark_bg"],
            "foreground": COLORS["text"],
            "font": ("Arial", 10)
        }},
        "Control.TFrame": {"configure": {"background":
            COLORS["light_bg"]}},
        "Sensor.TFrame": {"configure": {
            "background": COLORS["warning"],
            "borderwidth": 1,
            "relief": "solid"
        }},
        "Sensor.TLabel": {"configure": {
            "background": COLORS["warning"],

```

```

        "foreground": "#000000",
        "font": ("Arial", 12, "bold")
    }},
    "Status.TLabel": {"configure": {
        "background": COLORS["accent"],
        "foreground": COLORS["text"],
        "font": ("Arial", 10, "bold")
    }}
})
style.theme_use("custom")

def create_motor_controls(self, parent):
    control_frame = ttk.Frame(parent)
    control_frame.pack(pady=20)

    btn_style = {
        "font": ("Arial", 14, "bold"),
        "width": 8,
        "relief": "flat"
    }

    # Forward button
    self.btn_forward = tk.Button(
        control_frame,
        text="    ",
        bg=COLORS["success"],
        command=lambda: self.send_motor_command("forward")
        ,
        **btn_style
    )
    self.btn_forward.grid(row=0, column=1, padx=5, pady=5)

    # Left button
    self.btn_left = tk.Button(

```

```

        control_frame,
        text="    ",
        bg=COLORS["accent"],
        command=lambda: self.send_motor_command("left"),
        **btn_style
    )
    self.btn_left.grid(row=1, column=0, padx=5, pady=5)

# Stop button
self.btn_stop = tk.Button(
    control_frame,
    text="    ",
    bg=COLORS["danger"],
    command=lambda: self.send_motor_command("stop"),
    **btn_style
)
self.btn_stop.grid(row=1, column=1, padx=5, pady=5)

# Right button
self.btn_right = tk.Button(
    control_frame,
    text="    ",
    bg=COLORS["accent"],
    command=lambda: self.send_motor_command("right"),
    **btn_style
)
self.btn_right.grid(row=1, column=2, padx=5, pady=5)

# Reverse button
self.btn_reverse = tk.Button(
    control_frame,
    text="    ",
    bg=COLORS["success"],

```



```

        command=lambda: self.send_motor_command("reverse")
        ,
        **btn_style
    )
    self.btn_reverse.grid(row=2, column=1, padx=5, pady=5)

def connect_services(self):
    try:
        self.motor_audio_socket = socket.socket(socket.
            AF_INET, socket.SOCK_STREAM)
        self.motor_audio_socket.connect((HOST,
            MOTOR_AUDIO_PORT))
        self.update_status("Connected to motor/audio
            controller")
    except Exception as e:
        self.update_status(f"Connection failed: {str(e)}")

def send_motor_command(self, command):
    try:
        if self.motor_audio_socket:
            # Send motor command with header
            header = struct.pack('!BH', 0, len(command)) #
                0 for motor command
            self.motor_audio_socket.sendall(header +
                command.encode())
            self.update_status(f"Sent command: {command.
                upper()}")
    except Exception as e:
        self.update_status(f"Command failed: {str(e)}")

def send_audio(self):
    while self.running:
        try:
            audio_data = self.input_stream.read(1024)

```

```

        if self.motor_audio_socket:
            # Send audio with header (1 for audio data
            # ) and size
            header = struct.pack('!BI', 1, len(
                audio_data))
            self.motor_audio_socket.sendall(header +
                audio_data)
        except Exception as e:
            print(f"Audio send error: {str(e)}")
            break

def receive_video(self):
    sock = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    try:
        sock.connect((HOST, VIDEO_PORT))
        while self.running:
            size_data = sock.recv(4)
            if len(size_data) != 4:
                break
            size = struct.unpack('!I', size_data)[0]
            data = b''
            while len(data) < size:
                data += sock.recv(size - len(data))
            self.video_queue.put(data)
    finally:
        sock.close()

def receive_audio(self):
    sock = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    try:
        sock.connect((HOST, AUDIO_PORT))
        while self.running:

```

```

        size_data = sock.recv(4)
        if len(size_data) != 4:
            break
        size = struct.unpack('!I', size_data)[0]
        data = b''
        while len(data) < size:
            data += sock.recv(size - len(data))
        #print(data)
        self.audio_queue.put(data)
    finally:
        sock.close()

def receive_sensor(self):
    sock = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    try:
        sock.connect((HOST, SENSOR_PORT))
        while self.running:
            size_data = sock.recv(4)
            if len(size_data) != 4:
                break
            size = struct.unpack('!I', size_data)[0]
            data = b''
            while len(data) < size:
                data += sock.recv(size - len(data))
            #print(data)
            self.process_sensor_data(data)
    finally:
        sock.close()

def play_audio(self):
    while self.running:
        try:
            data = self.audio_queue.get(timeout=0.1)

```

```

        self.audio_stream.write(data)
    except queue.Empty:
        pass

def update_video(self):
    try:
        data = self.video_queue.get_nowait()
        frame = cv2.imdecode(np.frombuffer(data, np.uint8)
                               , cv2.IMREAD_COLOR)
        if frame is not None:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(frame)
            imgtk = ImageTk.PhotoImage(image=img)
            self.video_label.config(image=imgtk)
            self.video_label.image = imgtk
        except queue.Empty:
            pass
    self.root.after(10, self.update_video)

def process_sensor_data(self, data):
    try:
        value = data.decode()
        #print(value)
        if value == '0':
            self.sensor_label.config(text="Air Quality:
                                         NORMAL")
            self.sensor_frame.config(style="Sensor.TFrame"
                                     )
            self.sensor_icon.config(text="  ", foreground
                                     ="#000000")
        else:
            self.sensor_label.config(text="Air Quality:
                                         TOXIC!")

```

```

        self.sensor_frame.config(style="Danger.TFrame"
                                   )
        self.sensor_icon.config(text="  ", foreground
                                   = "#ff0000")
    except:
        self.sensor_label.config(text="Sensor Error")
        self.sensor_frame.config(style="Danger.TFrame")

def update_status(self, message):
    self.status_label.config(text=f"System Status: {
        message}")

def on_close(self):
    self.running = False
    try:
        self.motor_socket.close()
        self.audio_stream.stop_stream()
        self.audio_stream.close()
        self.audio.terminate()
    except:
        pass
    self.root.destroy()

if __name__ == '__main__':
    root = tk.Tk()
    client = RoverClient(root)
    root.protocol("WM_DELETE_WINDOW", client.on_close)
    root.mainloop()

```

7.2 Code for Rover

```

import socket
import threading

```

```

import queue
import cv2
import pyaudio
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.ADC as ADC
import time

# Motor Configuration
LEFT_MOTOR_IN1 = "P8_10"
LEFT_MOTOR_IN2 = "P8_12"
RIGHT_MOTOR_IN3 = "P8_14"
RIGHT_MOTOR_IN4 = "P8_16"

GPIO.setup(LEFT_MOTOR_IN1, GPIO.OUT)
GPIO.setup(LEFT_MOTOR_IN2, GPIO.OUT)
GPIO.setup(RIGHT_MOTOR_IN3, GPIO.OUT)
GPIO.setup(RIGHT_MOTOR_IN4, GPIO.OUT)

def motor_control(cmd):
    if cmd == 'forward':
        GPIO.output(LEFT_MOTOR_IN1, GPIO.HIGH)
        GPIO.output(LEFT_MOTOR_IN2, GPIO.LOW)
        GPIO.output(RIGHT_MOTOR_IN3, GPIO.HIGH)
        GPIO.output(RIGHT_MOTOR_IN4, GPIO.LOW)
    elif cmd == 'backward':
        GPIO.output(LEFT_MOTOR_IN1, GPIO.LOW)
        GPIO.output(LEFT_MOTOR_IN2, GPIO.HIGH)
        GPIO.output(RIGHT_MOTOR_IN3, GPIO.LOW)
        GPIO.output(RIGHT_MOTOR_IN4, GPIO.HIGH)
    elif cmd == 'left':
        GPIO.output(LEFT_MOTOR_IN1, GPIO.LOW)
        GPIO.output(LEFT_MOTOR_IN2, GPIO.HIGH)
        GPIO.output(RIGHT_MOTOR_IN3, GPIO.HIGH)
        GPIO.output(RIGHT_MOTOR_IN4, GPIO.LOW)

```

```

elif cmd == 'right':
    GPIO.output(LEFT_MOTOR_IN1, GPIO.HIGH)
    GPIO.output(LEFT_MOTOR_IN2, GPIO.LOW)
    GPIO.output(RIGHT_MOTOR_IN3, GPIO.LOW)
    GPIO.output(RIGHT_MOTOR_IN4, GPIO.HIGH)
elif cmd == 'stop':
    GPIO.output(LEFT_MOTOR_IN1, GPIO.LOW)
    GPIO.output(LEFT_MOTOR_IN2, GPIO.LOW)
    GPIO.output(RIGHT_MOTOR_IN3, GPIO.LOW)
    GPIO.output(RIGHT_MOTOR_IN4, GPIO.LOW)

# Air Quality Sensor Setup
ADC.setup()
AIR_QUALITY_PIN = "AIN0"

# Network Configuration
DATA_QUEUE = queue.Queue()
LAPTOP_IP = "192.168.1.100" # Update this
DATA_PORT = 5000

# Initialize Data Socket
data_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM
    )
data_socket.connect((LAPTOP_IP, DATA_PORT))

def data_sender():
    while True:
        data_type, payload = DATA_QUEUE.get()
        header = data_type.ljust(5).encode()
        size = len(payload).to_bytes(4, 'big')
        data_socket.sendall(header + size + payload)

sender_thread = threading.Thread(target=data_sender, daemon=
    True)

```

```

sender_thread.start()

# Video Capture
def video_stream():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if ret:
            _, jpeg = cv2.imencode('.jpg', frame)
            DATA_QUEUE.put(('VIDEO', jpeg.tobytes()))
            time.sleep(0.05)

video_thread = threading.Thread(target=video_stream, daemon=
    True)
video_thread.start()

# Audio Capture
audio = pyaudio.PyAudio()
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
CHUNK = 1024

def audio_stream():
    stream = audio.open(format=FORMAT, channels=CHANNELS,
                        rate=RATE, input=True,
                        frames_per_buffer=CHUNK)
    while True:
        DATA_QUEUE.put(('AUDIO', stream.read(CHUNK)))
    stream.stop_stream()
    stream.close()

audio_thread = threading.Thread(target=audio_stream, daemon=
    True)

```



```

audio_thread.start()

# Sensor Data
def sensor_data():
    while True:
        value = ADC.read(AIR_QUALITY_PIN)
        DATA_QUEUE.put(('SENSOR', str(value).encode()))
        time.sleep(1)

sensor_thread = threading.Thread(target=sensor_data, daemon=
    True)
sensor_thread.start()
def command_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', 5001))
    server.listen(1)
    while True:
        conn, _ = server.accept()
        while True:
            cmd = conn.recv(1024).decode().strip()
            if cmd:
                motor_control(cmd)
        conn.close()

cmd_thread = threading.Thread(target=command_server, daemon=
    True)
cmd_thread.start()

while True:
    time.sleep(1)

```

Chapter 8

Result

8.1 3D model of the rover

The Figure 8.1 presents the 3D model of the rover. This model provides a comprehensive view of the rover's structural design, highlighting components specifically tailored for our project's unique requirements.



Figure 8.1: Final 3D model of rover

8.2 Hardware integration in rover

The integration of hardware components, as depicted in Figure 8.2, consists of several key elements that work together to ensure the efficient functioning of the system. These components include a high-resolution camera for capturing a live

video feed, enabling real-time visual monitoring. A microphone is incorporated for audio communication, allowing seamless interaction and transmission of voice signals. Additionally, environmental sensors are deployed to collect crucial data such as temperature, humidity, gas concentration, and other parameters essential for situational awareness.

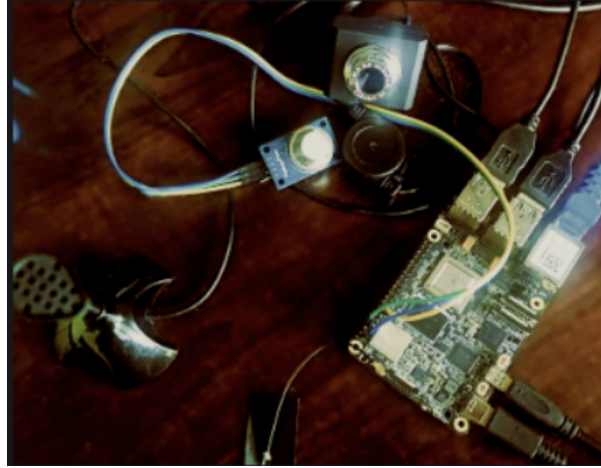


Figure 8.2: Hardware integration

8.3 Graphical User Interface at Base station

The GUI (Figure 8.3) at the base station provides an intuitive interface for monitoring and controlling the rover. It displays the real-time video feed, sensor data, and system status while offering on-screen controls for navigation and communication.

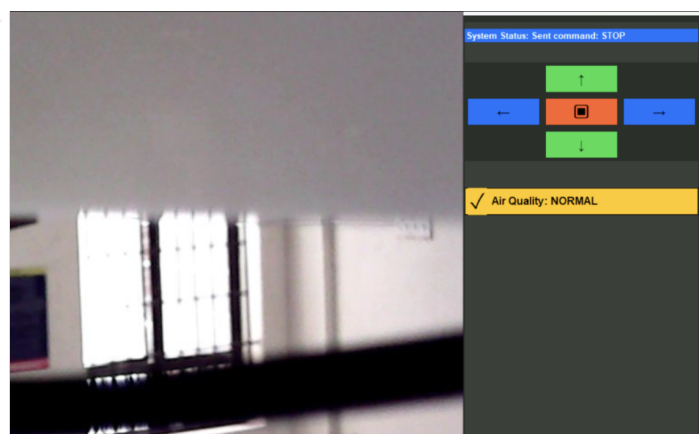


Figure 8.3: GUI at base station

8.4 Relay pipe guided rover

Figures 8.3 and 8.4 illustrate the final working model of the rover, showcasing the integration of all hardware and software components into a fully functional system. This model represents the culmination of extensive design, prototyping, and testing phases, ensuring that all features operate seamlessly. The rover is equipped with a robust chassis designed to navigate challenging terrains, including rocky surfaces, narrow tunnels, and unstable environments such as collapsed coal mines.



Figure 8.4: rover moving through pipe

```
Motor/Audio client connected: ('192.168.1.1', 49575)
Video client connected: ('192.168.1.1', 49576)
Audio client connected: ('192.168.1.1', 49577)
forward
reverse
reverse
```

Figure 8.5: output from beagle board

Chapter 9

Scope for future applications

By incorporating future advancements, the relay pipe guided rover can become an even more powerful tool for mine rescue operations, saving more lives and mitigating the impact of disasters.

9.1 Technological Advancements

1. **AI-Powered Autonomous Navigation:** Develop advanced AI algorithms to enable the rover to autonomously navigate through complex and changing environments within the pipe. Utilize machine learning techniques to analyze sensor data and adapt to unforeseen obstacles or pipe configurations.
2. **Enhanced Sensor Capabilities:** Incorporate more sophisticated sensors like LiDAR or thermal cameras to provide detailed 3D mapping and detect heat signatures of trapped miners. Integrate gas sensors with higher sensitivity and specificity to detect a wider range of hazardous gases.
3. **Improved Communication Systems:** Explore higher-bandwidth optical fiber technologies to transmit larger volumes of data, including high-resolution video and real-time sensor readings. Develop robust error correction and redundancy techniques to ensure reliable communication in challenging conditions.
4. **Hybrid system:** It enhances the rover's reliability by integrating wired and wireless communication for uninterrupted data transmission. Optical fiber pro-

vides high-speed, low-latency communication, while wireless fallback ensures connectivity when fiber deployment is obstructed.

9.2 Application Expansion

1. **Real-time Health Monitoring:** Equip the rover with biometric sensors to monitor the vital signs of trapped miners, such as heart rate, blood pressure, and oxygen levels. Transmit this data to medical experts for real-time assessment and guidance on medical interventions.
2. **Structural Assessment:** Utilize advanced imaging techniques and AI algorithms to assess the structural integrity of the mine, identify potential hazards, and inform rescue strategies.
3. **Environmental Monitoring:** Monitor environmental parameters like temperature, humidity, and radiation levels to assess the impact of the disaster and guide rescue efforts.
4. **Remote Control and Tele operation:** Develop intuitive user interfaces and advanced tele operation techniques to enable remote operators to control the rover with precision and efficiency.

9.3 Integration with Other Technology

1. **Drone Integration:** Collaborate with drones to provide aerial reconnaissance and mapping of the disaster site, complementing the rover's underground capabilities.
2. **IoT and Big Data:** Connect the rover to the Internet of Things (IoT) to enable remote monitoring, data analysis, and predictive maintenance. Utilize big data analytics to identify patterns and trends in sensor data, improving decision-making and resource allocation.

Chapter 10

Conclusion

This project focuses on developing a cutting-edge mine rescue rover designed specifically for post-collapse scenarios, where traditional communication and rescue methods are ineffective. The rover is equipped with a robust wired communication system, ensuring a stable, interference-free, and real-time data link between the rover and the base station. This instantaneous transmission is critical in underground environments, where wireless communication is unreliable due to physical obstructions and signal loss. High-resolution imaging and environmental sensors are integral to the rover's functionality, allowing it to capture clear visuals and monitor hazardous conditions such as toxic gases, temperature variations, and humidity levels. By combining these features, the rover provides crucial real-time data to rescue teams, enabling them to accurately assess the situation and respond effectively, even in confined and challenging underground spaces.

Enhanced Rescue Capabilities: The rover is designed to navigate through a strategically placed relay pipe, ensuring safe movement within collapsed or obstructed passages. This pipe serves as a dedicated pathway, allowing the rover to reach areas where miners may be trapped. A key innovation is its support for bidirectional communication, enabling two-way audio and data transfer between rescue teams and trapped individuals. This capability is essential for gathering first-hand information from those inside the collapsed area and providing them with updates and support. Additionally, the rover's environmental monitoring system continuously measures gas levels, temperature, and other parameters, offering a comprehensive real-time overview

of underground conditions. This continuous monitoring aids in formulating effective rescue strategies, reducing risks, and adapting to changing conditions—significantly improving the chances of a successful rescue mission.

Potential to Save Lives: By integrating state-of-the-art technology, this rover sets a new benchmark in mine safety and rescue operations. Its wired communication system ensures uninterrupted contact, overcoming one of the biggest challenges in underground rescues—the loss of communication due to debris and structural obstructions. The rover’s high-quality imaging and real-time environmental monitoring bridge the communication gap that often hampers traditional rescue methods. This constant flow of data empowers rescue teams to make rapid, informed decisions, reducing the time required to locate and assist trapped miners. Ultimately, the rover’s ability to maintain real-time communication, monitor hazardous conditions, and navigate through challenging environments is aimed at saving lives by accelerating response times, improving situational awareness, and enhancing the overall effectiveness of mine rescue operations.

References

1. G. Junyao, G.Xueshan, Z. Wei, Z. Jianguo and W. Boyu, "Coal Mine Detect and Rescue Robot Design and Research", 2008 *IEEE International Conference on Networking, Sensing and Control*, Sanya, China, 2008, pp. 780-785, doi: 10.1109/ICNSC.2008.4525321
2. Hemanth Reddy A, Balla Kalyan, Ch. S. N. Murthy, "Mine Rescue Robot System A Review", 2015 *Global Challenges, Policy Framework Sustainable Development for Mining of Mineral and Fossil Energy Resources (GCPF2015)*
3. Yutan Li, MenggangLi, Hua Zhu, Eryi Hu, Chaoquan Tang,Peng Li,Shaoze You,"Development and applications of rescue robots for explosion accidents in coal mines", *J Field Robotics*. 2019;1–24
4. S.g. Roh, D. W. Kim, J. -S. Lee, H. Moon and H. R. Choi, "Modularized in-pipe robot capable of selective navigation inside of pipelines", 2008 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 2008, pp. 1724-1729, doi: 10.1109/IROS.2008.4650968
5. Ray, D. N., Das, R., Sebastian, B., Roy, B., & Majumder, S. (2016). Design and Analysis Towards Successful Development of a Tele-Operated Mobile Robot for Underground Coal Mines. In **CAD/CAM, Robotics and Factories of the Future** (pp. 589–602). Springer, New Delhi.
6. BeagleBoard.org. (2024). BeagleY-AI — BeagleBoard Documentation. Retrieved from <https://docs.beagleboard.org/boards/beagle-y-ai/index.html>.

7. Components101. (n.d.). MQ-135 Gas Sensor Pinout, Features, Alternatives, Datasheet & Uses Guide. Retrieved from <https://components101.com/sensors/mq135-gas-sensor-for-air-quality>
8. Ariff, M. F. M., Majid, Z., Setan, H., & Chong, A. K. (2010). Near-infrared camera for night surveillance applications. *Geoinformation Science Journal*, 10(1), 38–48.
9. Kostenko, K. V. (2019). Real-time communication between robotic PLC and PC via Ethernet-based protocols.
10. Eskafi, F., Ross, B., & Xiao, L. (2002). Internet Controlled Rover.
11. A. M. C. Rezende et al., "Indoor Localization and Navigation Control Strategies for a Mobile Robot Designed to Inspect Confined Environments," 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), Hong Kong, China, 2020, pp. 1427-1433, doi: 10.1109/CASE48305.2020.9217005.
12. Krishnamoorthy, S., & Kesava Panikkar, P. P. (2023). A comprehensive review of different electric motors for electric vehicles application. *International Journal of Power Electronics and Drive Systems (IJPEDS)*, 15(1), 74-90. Retrieved from
13. Kandasamy, N. K., Ramachandran, K. I., & Rajendran, P. (2022). A review of Li-ion batteries for autonomous mobile robots: Perspectives and outlook for the future. *Journal of Power Sources*, 541, 231766.
14. Kumar, L., & Jain, A. (2021). 3D printing – A review of processes, materials and applications in industry 4.0. *Materials Today: Proceedings*, 45, 2080-2086. Retrieved from <https://www.sciencedirect.com/science/article/pii/S2666412721000441>
15. Wary, N., & Mandal, P. (2015). High-speed energy-efficient bi-directional transceiver for on-chip global interconnects. *IET Circuits, Devices & Systems*, 9(5), 319-327.

16. Beniz, D. B., & Espindola, A. M. (2016). Using Tkinter of Python to Create Graphical User Interface (GUI) for Scripts in LNLS. *Proceedings of the 11th International Workshop on Personal Computers and Particle Accelerator Controls*, (pp. 1–4).
17. L. Zhou, Y. Dou, H. Liu, W. Zhang, J. Hu and C. Yang, "Shared Control Method for Coal Mine Rescue Robots," 2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI), Tianjin, China, 2021, pp. 1-6, doi: 10.1109/CVCI54083.2021.9661251.
18. R.R.Murphy, J. Kravitz, S. L. Stover and R. Shoureshi, "Mobile robots in mine rescue and recovery," in *IEEE Robotics Automation Magazine*, vol. 16, no. 2, pp. 91-103, June 2009, doi: 10.1109/MRA.2009.932521.
19. Yong Guo, Fuqiang Yang, Mining safety research in China: Understanding safety research trends and future demands for sustainable mining industry, *Resources Policy*, 10.1016/j.resourpol.2023.103632, 83, (103632), (2023).
20. A. Akbari, P. S. Chhabra, U. Bhandari and S. Bernardini, "Intelligent Exploration and Autonomous Navigation in Confined Spaces," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 2157-2164, doi: 10.1109/IROS45743.2020.9341525.
21. OpenAI. (2023). Introducing ChatGPT. Accessed: May 26, 2023. [Online]. Available: <https://openai.com/blog/chatgpt>