Design of Autonomous Rover for Firefighter Rescue: Integrating Deep Learning with ROS2

Saad A. Rafiq[1,*], Erich S. Ellsworth[2,*], Oscar Resendiz[2], Susmitha H. Varanasi[3], Adenrele A. Ishola[2]
Rachel Koldenhoven[4], John W. Farrell III[4], Yumeng Li[4], Maria D. Resendiz[5], Semih Aslan[2], Ting Liu[6],
Damian Valles[2]
[1]*Department of Computer Science, Texas State University, San Marcos, TX 78666, USA*
[2]*Ingram School of Engineering, Texas State University, San Marcos, TX 78666, USA*
[3]*Department of Information Systems and Analytics, Texas State University, San Marcos, TX 78666, USA*
[4]*Department of Health and Human Performance Texas State University, San Marcos, TX 78666, USA*
[5]*Department of Communication Disorders, Texas State University, San Marcos, TX 78666, USA*
[6]*Department of Counseling, Health and Kinesiology, Texas A&M University-San Antonio San Antonio, TX 78224, USA*
[*]ers131@txstate.edu

2024 IEEE World AI IoT Congress (AIIoT) | 979-8-3503-8780-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/AIIoT61789.2024.10578950

***Abstract*—Firefighters often face the dangerous task of navigating burning structures to rescue endangered individuals, with challenges like intense heat, toxic fumes, and debris. We introduce a rover concept to aid these rescue missions through autonomous data collection. This rover can climb stairs, detect distressed sounds, locate individuals, and gather environmental data. The paper details the design of its power and hardware components for data operations and movement. Using the Robotic Operating System (ROS2), telemetry data is automated in real-time, integrating three NVIDIA Jetson Nanos and two deep learning models. These models identify individuals via infrared (IR) video and detect human screams. We discuss the models' performance, latency, and integration with the ROS2 framework. We aim to expedite rescue operations, enhancing safety and survival rates for firefighters and civilians.**

***Index Terms*—autonomous, firefighting, ros2, deep learning, rover**

I. Introduction

This paper proposes using autonomous units powered by the Robot Operating System 2 (ROS2) and advanced machine learning algorithms to assist in fire rescue operations. By using autonomous scouting units, the time needed to locate victims trapped in a burning building could be reduced, potentially saving the lives of civilians and reducing firefighters' overall exposure to dangerous situations. According to the United States Fire Administration, 2,840 lives were lost in residential fires [1]. This paper presents a rocker-bogie-based robotic vehicle that utilizes machine learning algorithms to detect people inside burning buildings. The overall design focuses on mobility in harsh environments, such as burning buildings. ROS2 is a framework to interface the scream and person detection algorithms to the rover's other software subsystems.

A. Bacground

Robot-assisted firefighting is familiar, with several designs exploring tank-style tracked vehicles for firefighting applications. Counterweight-assisted tracked vehicles have shown promise, with excellent performance when scaling steep obstacles such as stairs and debris [2]. Furthermore, the use of a rocker-bogie style design is promising due to the relative mechanical simplicity of the design while maintaining sufficient performance in scaling objects [3].

Additionally, the complex functionality of modern robotics has led to ROS2, which implements standard robotics tooling into an easy-to-use framework [4]. The use of combining ROS2 with machine learning models has been well explored, including the use of traditional CNN models [5]. The use of ROS2 with firefighting-specific machine learning models still needs to be well explored, as the use of firefighting-specific machine learning models is still being explored.

II. Materials and Methods

A. Power

The rover is powered by a 13Ah Nickel-metal hydride (NiMH) battery consisting of ten F-cell batteries. The battery's total rated capacity is 156.0Wh, with a nominal cell voltage of 1.2V, translating to a nominal pack voltage of 12.0V. However, when fully charged, the peak cell voltage may reach 1.4V. The decision to use a NiMH-based battery was based on the requirements for long service life and high resistance to damage caused by over-discharging. The battery pack is rated for a continuous output of 13A, with a 20A fuse connected inline before reaching a master cutoff switch. Power is then distributed via WAGO 221 lever-nuts. The two motor controllers and the DC-DC converter are connected directly to the 12V output of the battery. The DC-DC converter provides a 5 V supply for the three onboard NVIDIA Jetson Nano single-board computers. The rover power system is demonstrated in Fig. 1.

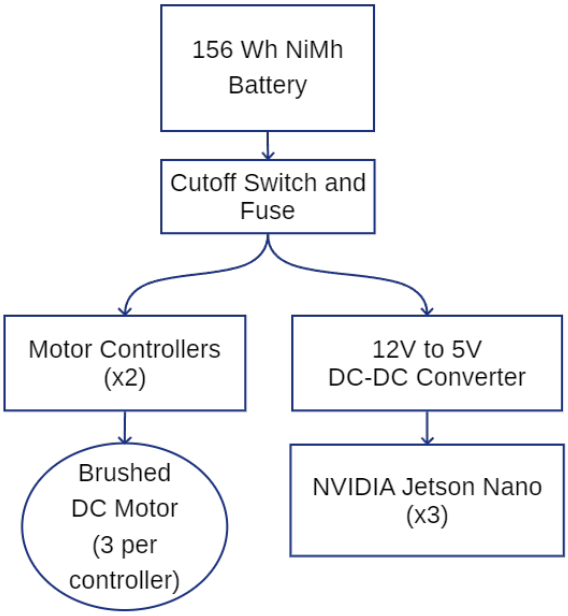

Fig. 1. A generalized diagram of the rover's power system.

B. Compute

Three NVIDIA Jetson Nano single-board computers control the rover. The Jetson Nanos are responsible for data collection, processing, and controlling the rover's systems. The Jetson Nano features a quad-core ARM Cortex-A57 CPU running at 1.43 GHz with an integrated NVIDIA Maxwell GPU. The CPU and GPU share 4 GB of system RAM. Storage is provided by a 32 GB MicroSD card plugged directly into the Jetson Nano. Each Jetson receives power from its onboard 5V input barrel jack. NVIDIA provides software tooling to limit the power consumption for each Jetson to 5 W compared to the default of 10 W. This power rating does not consider any additional peripherals, so the rover's power budget guarantees 15W of power to be available to each Jetson Nano. Communication is provided via a standard 2.4 GHz Wi-Fi USB adapter, which connects to an off-the-shelf wireless router.

C. Sensors

The rover is equipped with a Bosch BME680 environmental data sensor that monitors temperature, humidity, pressure, and the presence of volatile organic compounds (VOCs) in the air. The sensor is connected directly to the I2C bus that is exposed on Jetson 2's General Purpose Input/Output (GPIO) header. This sensor enables the rover to collect and analyze critical environmental data in real-time, providing valuable insights into the surrounding environment. The rover also features a front-facing active illumination infrared camera connected to Jetson 3 via a flexible ribbon cable. The Jetson Nano uses the Camera Serial Interface (CSI) protocol to communicate with the camera. The camera is equipped with an array of infrared LEDs controlled by a photo-resistor, which turns on the LEDs in dark conditions, enabling the camera to operate effectively in low-light conditions. Lastly, Jetson 1 has a standard USB microphone for scream detection. The microphone is a general-purpose design with a focus on unidirectional reception.

III. Software Stack

The robot software stack uses Xubuntu 20.04 to run the Robot Operating System 2 (ROS) Foxy Edition. Xubuntu is a lightweight Linux distribution; the advantage is that it leaves resources for running Artificial Intelligence (AI) models on the Jetson Nano. ROS, which supports C++ and Python 3.x, provides an array of software libraries and tools for robotics applications. In particular, ROS allows us to transfer data between the Jetson Nanos using a publisher/subscriber server and send commands from the end user to the robot using an action/client-server. All three Jetson Nanos and the end user computer are connected to the same WiFi network, allowing communication. The data that is being transmitted can be described as follows and can be visualized in Fig. 2.

- Jetson 1
  - Publishes true or false if a scream is detected
- Jetson 2
  - Publishes environmental data using the BME680 sensor (heat, humidity, air pressure, air quality)
  - Executes action input from the user and commands the motor controllers
- Jetson 3
  - Publishes true or false if a person is detected
- Client Computer
  - Sends user input to Jetson 2
  - Subscribes and displays data from all of the Jetson Nanos

1) Publisher/Subscriber Function: A publisher is run and subscribed on the client computer, *publisher_master_function.py*. The purpose of this function is to synchronize the data coming in. Each Jetson publishes data at different rates, which is time-stamped on receipt. The *subscriber_function.py* will hold on to the received data until it receives a publication from *publisher_master_function.py* and will then output all of the data the subscriber holds to the end user. Old data will be used if no new data updates the subscriber. Without this *master_publisher*, the code depends on one of the three Jetsons for when it can output data, potentially creating a bottleneck. Another advantage to the *master_publisher* is that the end-user has one place locally to change the data output frequency.

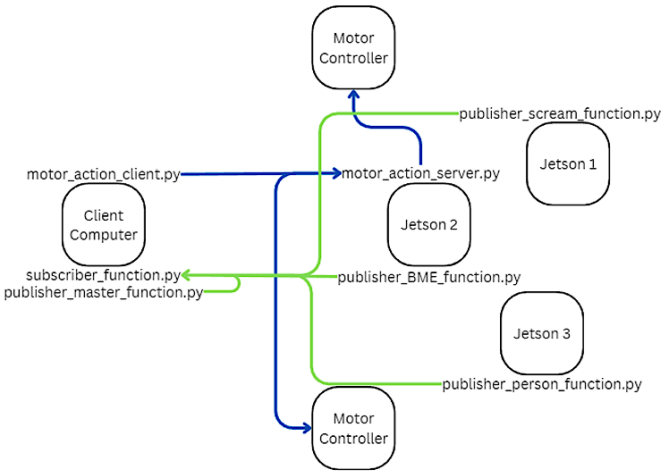

Fig. 2. ROS Data Transmission

By default, publishers and subscribers in ROS can transmit nineteen data types, and the type of data being published/subscribed must be defined using messages. Messages are data types the user can create in ROS; messages can be compared to C++ structs. An example of a *.msg* for transferring BME680 data can be shown in **??**. Furthermore, in this message, we are telling ROS that in a single publishing event, we will transfer several *float64* type variables called *temp*, *humidity*, *pressure*, *gas_iaq*, and an *int64* type variable called *gas_ohm*. This *.msg* file is imported for both the publisher and the subscriber.

In Algorithm 1, we can see a pseudo implementation of a publisher in Python. This example shows how *.msg* files are imported and used. In this example, once the code starts, an instance of the *MinimalPublisher* Node is created, which then starts running by the *rclpy.spin* function (not shown). On instantiation, the *MinimalPublisher* instance will create itself as a publisher, passing parameters signifying what kind of data will be outputted, in our case, the BME message type. Once the object initialization is complete, it will run in the *timer_callback* function, which populates the variables defined in the message file with data from the board. This is then published every 5 seconds until the program is ended.

On the other side, shown in Algorithm 2, will similarly create an instance of *MinimalSubscriber* and starts running with the same *rclpy.spin* function. The subscriber will define itself as a subscriber and subscribe to the BME and master. This means that the *BME_callback* function runs once a BME message is received., as seen in Algorithm 3. Similarly, the *master_callback* function is run.

2) Action/Client Server: The action/client-server works similarly to the publisher/subscriber in concept. In our implementation, the client (run on the client computer) will

Algorithm 1 BME Publisher Python Implementation

```
Require: import rclpy, board
Require: import adafruit_bme680
Require: from rclpy.node import Node
Require: from pub_package.msg import BME
class MinimalPublisher(Node):
    def __init__(self):
        super().__init__('BME_publisher')
        self.publisher_ = self.create_publisher(BME,
                          'topic', 10)
        self.timer = self.create_timer(.5,
                     self.timer_callback)
    def timer_callback(self):
        msg = BME()
        msg.temp = get_temperature()
        msg.pressure = get_pressure()
        msg.humidity = get_humidity()
        msg.gas_ohm = get_gas_ohm()
        msg.gas_iaq = get_gas_iaq()
        self.publisher_.publish(msg)
        self.get_logger().info("Publishing...")
```

Algorithm 2 BME Subscriber Python Implementation

```
Require: import rclpy, time
Require: from rclpy.node import Node
Require: from BME_package.msg import BME
Require: from master_package.msg import
  Master
class MinimalSubscriber(Node):
    def __init__(self):
        super().__init__('subscriber')
        self.subscription = self.create_subscription(BME,
                            'topic', self.BME_callback, 10)

        self.subscription = self.create_subscription(Master,
                            'topic', self.master_callback, 10)

        self.subscription
        # Set Default Values

    def BME_callback(self, msg):
        self.temp = msg.temp
        self.humidity = msg.humidity
        self.pressure = msg.pressure
        self.gas_ohm = msg.gas_ohm
        self.gas_iaq = msg.gas_iaq

    def master_callback(self, msg):
        self.print_data()
```

Algorithm 3 BME.msg

```
float64 temp
float64 humidity
float64 pressure
float64 gas_iaq
int64 gas_ohm
```

ask the server (run on Jetson 2) to execute a key command. The rover's keyboard inputs are as follows:

- W - Forwards
- A - Turn Left
- S - Backwards
- D - Turn Right
- E - Stop
- Q - Quit

Once the server receives one of these inputs, a function corresponding to each key will run, setting the motor PWM parameters—the *RPi.GPIO* library is used to interface with the motor controllers via PWM, which are changed in the corresponding functions for forwards, backward, left, right, and stop. The purpose of having *Q* as an option is to allow a graceful exit for the motors. It is a standard error that abruptly turning off the PWM output can cause issues in the following use of PWM.

Unlike the publisher/subscriber server, the action/client-server uses actions consisting of three parts: a goal, feedback, and result. What ROS calls "goal" is the request sent by the client to the server to execute. In our basic implementation of the rover, we created the Motor action. This uses a *char* data type to transmit the input (the goal) and then receives a *String* confirmation log indicating whether the goal was executed correctly (the feedback/result).

The action/client-server implementation runs similarly to the publisher/subscriber code implementation with ROS. Both of the servers are dependent on receiving data from another computer. What is done with the data once received separates the two servers. In the action/client-server, two-way communication ensures that the goal is executed correctly. The code also behaves similarly; the client node starts, and a goal is sent to the server (keyboard input), which will then execute its callback function and finally return a confirmation result. The exact process can be done using a publisher/subscriber function; however, we chose to go with the action/client-server to improve scalability for our basic WASD implementation. For example, there are plans to add autonomous navigation in the future, which can take advantage of the feedback function to execute more commands.

A. Deep Learning Models

1) Introduction to Improved AI Model Pipeline: While prior research in this project led to the development of two AI Models, the CNN – Person Detection Model and the LSTM – Scream detection model, these models proved too slow for our specific use case. As a result, the team decided to redevelop the AI model pipeline with an edge computing endpoint in mind. The team opted to utilize two pre-trained edge computing-based models: YAMNet and MobileNetv2.
2) Rationale for Model Selection: We selected these models based on their demonstrated efficacy and efficiency in performing various audio classification and object detection tasks. This choice proved advantageous for our pipeline, given the constraints of computational power and GPU memory compared to our primary workstations, initially employed to develop the preliminary model set.
3) MobileNetV2: Person Detection: For our person detection, we used MobileVNet2 to detect objects and classify human presence from the live video stream provided by the infrared camera. MobileNetV2 is also a pre-trained neural network optimized for mobile and embedded devices. Google researchers developed an optimized version of the original MobileVNet architecture.

MobileNetV2 was trained on the ImageNet dataset, a large collection of images with more than one million examples spanning over a thousand categories. It has achieved state-of-the-art performance on several image classification benchmarks, including the ImageNet Classification challenge.
4) YAMNet: Scream Detection: For scream detection, we used YAMNet to classify the recorded audio samples from Jetson Nano 1 as screams or non-screams. YAMNet is a pre-trained deep neural network designed and developed by Google researchers for Audio analysis. The model is a variant of the MobileNet architecture.

YAMNet was trained on the AudioSet dataset, a large-scale collection of labeled audio events developed by Google. The dataset contains over two million 10-second sound clips that cover multiple audio categories, including musical instruments, animal sounds, environmental sounds, and, for our use case, human speech.
5) Model Integration in the Current System: During the integration process, the models had to be developed to allow seamless integration with the current data pipeline. Specifically, we had to ensure that the CIS Camera and USB Microphone were still appropriately utilized with no issues.

For the MobileNetv2 Model, this proved to be a simple procedure as the team could utilize NVIDIA's boilerplate code for Object Detection found in their online tutorials [6]. This code allowed TensorRT to power the MobileVNet2 model in the background, allow faster inference time, and, more importantly, allow seamless integration of our CIS Camera.

After a few modifications to the code, we can extract Person Detection in a Boolean value and by an actual image of the person detected.

The YAMNet model was integrated by building upon the code provided by Google [7]. Instead of calling from an API to provide inference, the model was downloaded from the TensorFlow Hub as a TF2.0 Saved model .pb file. From here, it was loaded into the model using TensorFlow's built-in load function. From here, we modified our code to create 16khz Wav files from the USB microphone's live recordings and used these recordings to provide the most probable prediction.

6) AI Model pipeline ROS2: In the finalization stage of the AI Model Pipeline, the team recognized the need for an effective communication method to relay the variety of positive values generated from the inference of both pipelines. The team decided to implement ROS2 directly into the system. In particular, the real-time publisher-subscriber model allowed constant surveillance of the AI model's positive output. This ensures timely identification and communication of positive classification across all the Rovers systems.

IV. Physical Construction

The primary design goal of the rover is the ability to navigate harsh environments. The rover should be able to climb stairs and maneuver around obstacles in tight spaces. Additionally, several design decisions were made to improve the ability to redesign and repair the rover as the design phase progressed.

1) Rocker Bogie: The rocker-bogie design was chosen due to it is versatility in rugged terrain. A functional requirement of the rover is the ability to climb stairs, which is essential when navigating multi-story buildings. One of the most considerable benefits of the rocket-bogie design is that it is mechanically simple to implement, requires no additional actuators, and allows the rover to scale objects higher than the wheels' diameter. The rover can climb various obstacles in our testing, including stairs and shipping boxes. The rocker-bogie system allows the front and middle wheels to articulate together on a shared pivot while the rear wheel maintains a fixed position, providing support as the rover scales an object. Fig. 3 shows the side view of the rover with the rocker-bogie point at the front of the rover. Fig. 4 shows the side view of the actual rover climbing a monitor box with the rocker-bogie adapting to the climbing as it moves forward.

2) Parts: Below defines a list of parts used for the rover in Table I. The structural frame of the rover consists of 1.5 in PVC diameter pipes connected together via friction fit and secured together with a bolt. PVC pipe was used due to it's relatively low cost,high availability. The PVC sections were cut using a standard miter saw. Once the PVC is cut, it was then sanded down with a belt sander and hammered together to form the complete section.

The base plate of the rover is made from 0.5 cm acrylic, which was cut from a vertical band saw. Once the piece had been cut, the spacers were sanded down to remove imperfections. The electronics sitting on the base plate are mounted using brass standoffs. After drilling holes with a standard hand drill, the standoffs were screwed into the acrylic using a wrench. Finally, combining the frame and the base plate, we can add the motors and wire the rover for testing and software stack implementation. Fig. 5 shows the view of the Jetson Nanos and other peripherals on top of the acrylic base.

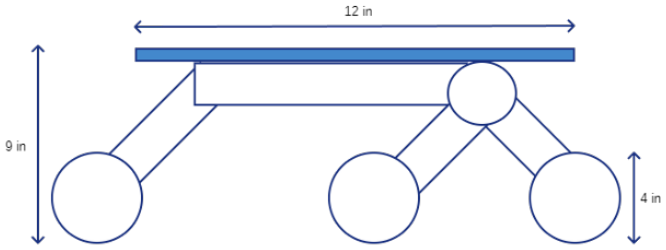

Fig. 3. A side view of the rover showing the rocker bogie pivot point.

Fig. 4. A side view of the rover as it scales a an obstacle.

V. Initial Results

1) AI Model pipeline speedup: Below, we have outlined the inference time the current pipeline averages over three runs compared to the previous implementation. The results reveal a substantial overall reduction in the inference time that can also improve the real-time processing capabilities shown in Table II. The original LSTM-based Scream Detection model averaged a Total AI Pipeline time of 215.7 seconds. We can then compare it to the new 3.1-second average that the YAMNet model provides. One thing to note in the new YAMNet model is that the most time-consuming process is the 3-second data recording. This showcases that the YAMNet

TABLE I
A PARTS LISTING FOR THE ROVER

Category	Components
Physical Components	4x 7”x1.5” PVC 2x 6”x1.5” PVC 2x 5”x1.5” PVC 2x 4”x1.5” PVC 2x 2”x1.5” PVC 2x 1.5” PVC Caps 6x 1.5” 90degree PVC Elbows 4x 1.5” 45degree PVC Elbows 1x 12”x18”x0.2 Acrylic 4x 3”x0.5” Acrylic Spacers 6x Adjustable Pipe Clamps 6x 5” Rubber RC Tires
Electrical Components	6x 12v 20RPM Motors 2x 20A Motor Controllers 3x NVIDIA 4GB Jetson Nanos 1x IR Ribbon Camera 1x USB Microphone 1x 12V to 5V DC-DC Converter 1x 12V 13Ah NiMH Battery
Miscellaneous	Nuts and Bolts 16 AWG Wire Power Switch M&F XT60 Connectors Wago Wire Nuts Standoffs Velcro

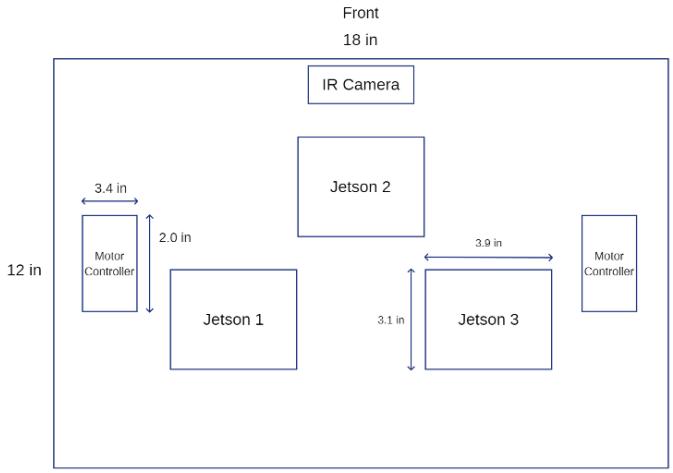

Fig. 5. The view of the deck of the rover, where the Jetson Nanos, motor controllers, and IR camera is located.

pipeline is mainly constrained by how much data the team chose to gather before using the data for inference. This new speedup allows the model to be better suited for a time-sensitive application that needs to detect screams as soon as they occur.

TABLE II
TABLE OF AI MODEL PIPELINE SPEEDUP.

AI Pipeline	Pre-Optimization	Post-Optimization
Scream Detection	215.7	3.1
Person Detection	30.9	8.05e-5

Similarly, the Person Detection model exhibits a reduction from 30.9 seconds to 8.05e-5 seconds using the MobileVNet2 architecture. This reduction will allow the ability to scale the AI models utilizing the increased speedup in classifying humans in a time-critical environment.

2) AI Model Training and Testing: The LSTM and the YAMNet networks were trained from the same dataset found in [11]. The LSTM model showed stable training and validation learning curves, as seen in Fig. 6. The model converges just below 96% accuracy in its validation at 50 epochs. Its classification performance can be analyzed in its confusion matrix in Fig. 7. Some misclassifications are due to similar frequency components and signal-to-noise (SNR) ratio variations presented in the dataset samples. The correlation of the alarms to conversations can be complex in distinguishing low SNR from the alarm and mimicking frequency components in the high SNR conversation samples.

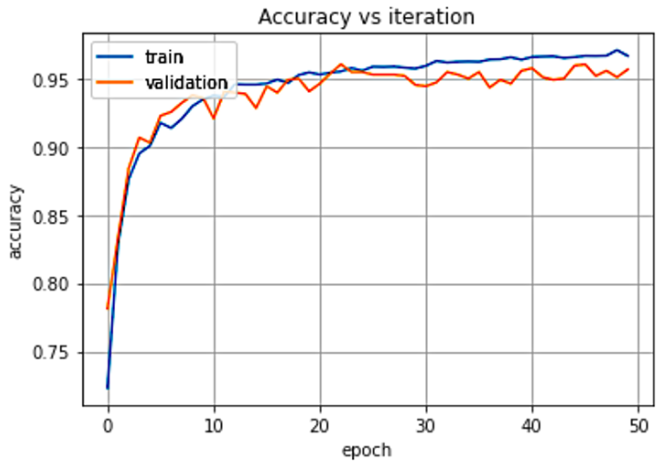

Fig. 6. The Accuracy vs. Epoch graph for the LSTM model.

The YAMNet network learning curves are shown in Fig. 8. The curves show a lower performance of the validation curve at 70 epochs with an accuracy of 91%. The overall classification performance of the YAMNet network is seen in Fig. 9. The large misclassification of the alarms to the conversation improved from the LSTM network; however, other misclassifications in other labels increased and showed why the validation curve is at a lower performing curve. These initial training and testing performance metrics show that the results in Table II show that the networks require more time to analyze the complexity of the screaming-detection feature.

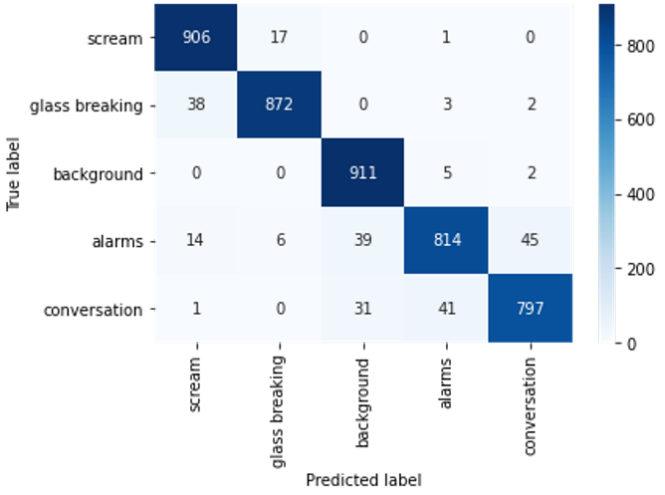

Fig. 7. The confusion matrix of the LSTM model.

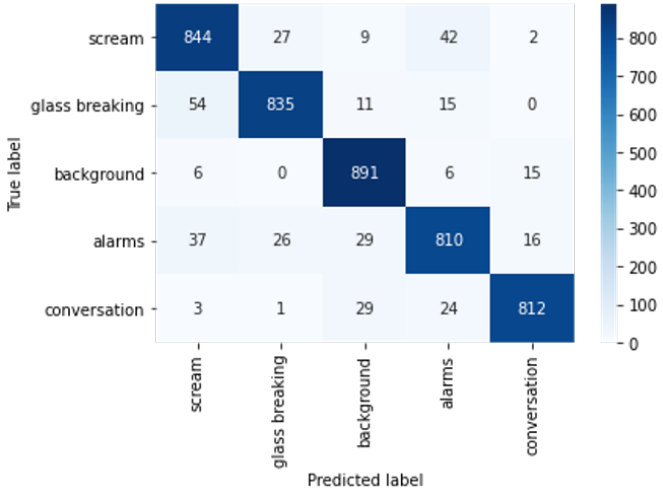

Fig. 9. The confusion matrix of the YAMNet model.

The whole is to continue improving people's critical detection at its lowest latency computation determination.

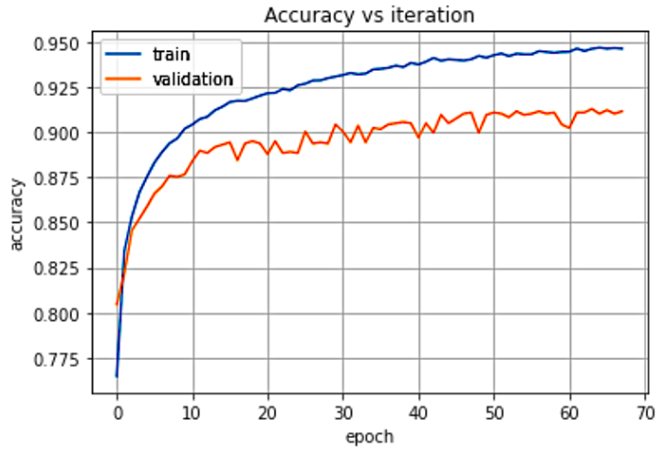

Fig. 8. The Accuracy vs. Epoch graph for the YAMNET model.

VI. Conclusions

In this research, we have presented a study on integrating the Robot Operating System (ROS) with advanced machine learning models to enhance fire rescue operations. Our work has culminated in several significant findings and contributions that can substantially impact emergency response. The development and optimization of the person recognition and scream detection models can accurately recognize victims in a fire, with excellent performance on power-limited systems. We have demonstrated that by leveraging the flexibility and adaptability of ROS, coupled with the power of machine learning algorithms, it is possible to develop intelligent and autonomous systems capable of effectively addressing complex and dynamic fire rescue scenarios. The Rocker-bogie suspension design shows promise to provide a high degree of mobility without using sophisticated actuators or mechanical legs. The rocker-bogie suspension design has a long history in aerospace and is easily adapted to the harsh conditions that may be experienced when scouting a burning building.

A. Future Work

One crucial aspect of our future work involves integrating a 3D LiDAR system for mapping the vehicle's surroundings as it navigates a building. This will allow firefighters to better pinpoint the location of victims and the current position of the vehicle. The addition of a sound localization system would improve the rover's ability to locate a victim trapped in a burning building. By combining the localization system with the existing scream detection model, the approximate position of a person could be relayed back to firefighters to assist in rescue operations.

The current rover uses 2.4 Ghz WiFI to communicate between the rover and firefighters. Although WiFI allows for real-time video and audio to be streamed to first responders, a secondary, LoRa-based communication link would allow for the commands and telemetry to be exchanged between the rover and the fire truck. Lastly, we are exploring the incorporation of reinforcement learning-based algorithms to allow for coordination between multiple autonomous units. Using many drone and rover bases, autonomous units could increase the speed at which victims are located by utilizing a divide-and-conquer approach to navigating the building.

REFERENCES

[1] U.S. Fire Administration, "Residential Fires Statistics," *U.S. Fire Administration (USFA)*. [Online]. Available: https://www.usfa.fema.gov/statistics/residential-fires/.

[2] S. Hoang et al., "Designing and Control Track-Belt Robot for Firefighting Task," in *Proc. of the 2023 7th International Conference on Robotics and Automation Sciences (ICRAS)*, Wuhan, China, 2023, pp. 53-57, doi: 10.1109/ICRAS57898.2023.10221617.

[3] K. Desarda et al., "RBFF: Rocker Bogie Fire-Fighter," in *Proc. of the 2022 International Conference on Signal and Information Processing (IConSIP)*, Pune, India, 2022, pp. 1-5, doi: 10.1109/ICoNSIP49665.2022.10007490.

[4] S. Macenski et al., "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[5] Y. -H. Chang et al., "Deep learning for object identification in ROS-based mobile robots," in *Proc. of the 2018 IEEE International Conference on Applied System Invention (ICASI)*, Chiba, Japan, 2018, pp. 66-69, doi: 10.1109/ICASI.2018.8394348.

[6] D. Franklin, "GitHub - dusty-nv/jetson-inference: Hello AI World guide to deploying deep-learning inference networks and deep vision primitives with TensorRT and NVIDIA Jetson," *GitHub*. [Online]. Available: https://github.com/dusty-nv/jetson-inference/tree/master.

[7] "Transfer learning with YAMNet for environmental sound classification," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/audio/transfer_learning_audio.

[8] "How to Make a Mars Rover/Rocker Bogie Robot - Stair Climbing," *Youtube*, 2017. [Online]. Available: https://www.youtube.com/watch?v=3Zx7tGtwF5g.

[9] H. Nayar et al., "Design optimization of a lightweight rocker–bogie rover for ocean worlds applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 6, p. 172988141988569, 2019.

[10] P. McWhorter, "AI ON THE JETSON NANO LESSON 59: PWM ON THE GPIO PINS OF THE JETSON NANO," *Technology tutorials*, Sep. 12, 2020. [Online]. Available: https://toptechboy.com/ai-on-the-jetson-nano-lesson-59-pwm-on-the-gpio-pins-of-the-jetson-nano/.

[11] F. S. Saeed, A. Al Bashit, V. Viswanathan, and D. Valles, "An Initial Machine Learning-Based Victim's Scream Detection Analysis for Burning Sites," *Appl. Sci.*, vol. 11, no. 18, p. 8425, Sep. 2021. [Online]. Available: https://doi.org/10.3390/app11188425.