Capstone Project Report

Embedded System for Intruder Detection

Ms. Pattaratorn Soranathavornkul

Ms. Puhnyanuj Smizdhanond

Mr. Thaninrath Thiraphotiwat

Robotics and AI Engineering

Faculty of Engineering

King Mongkut's Institute of Technology Ladkrabang

Academic Year 2022

Project Title	Embedded System for Intruder Detection
Student name	Ms. Pattaratorn Soranathavornkul (62011191), Ms. Puhnyanuj Smizdhanond (62011225), and Mr. Thaninrath Thiraphotiwat (62011276)
Degree	Bachelor of Engineering in Robotics and AI
Academic year	2022
Advisor name	Assoc. Prof. Somyot Kaitwanidvilai

ABSTRACT

Intruder detection is a critical aspect of security systems, and real-time embedded systems play a significant role in achieving efficient and timely responses. The project incorporates hardware and software components to identify unauthorized individuals rapidly and accurately. The primary objective is to enhance security and surveillance capabilities by developing an embedded system for real-time intruder detection targeted at red railways.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and appreciation to Assoc. Prof. Somyot Kaitwanidvilai for his invaluable guidance, support, and expertise throughout the duration of this project. His unwavering commitment to excellence and dedication to our success has been instrumental in shaping the development of the embedded system for intruder detection in red railways.

Assoc. Prof. Somyot Kaitwanidvilai's profound knowledge and vast experience in the field of embedded systems have provided us with invaluable insights and directions at every stage of the project. His mentorship and constructive feedback have greatly contributed to the refinement of our ideas, ensuring the effectiveness and feasibility of the proposed solution.

We would also like to extend our gratitude to the institution and department for providing the necessary resources and facilities to carry out this project. The support and encouragement received from the faculty and staff have been instrumental in our progress and achievements.

Lastly, we would like to acknowledge all individuals who have participated in this project by providing feedback, suggestions, and insights. Your contributions have played a vital role in shaping the outcome and ensuring the project's relevance and applicability.

TABLE OF CONTENTS

ABSTRACT I

ACKNOWLEDGEMENT II

TABLE OF CONTENTS III

LIST OF TABLES VI

LIST OF FIGURES VII

CHAPTER 1: INTRODUCTION 1

1.1 Background and Significance 1

1.2 Development Objectives 1

1.3 Scope of Development 2

1.4 Method of Conducting Research 3

1.5 Expecting Benefits 3

1.6 Project Timeline 5

CHAPTER 2: CONCEPT, THEORIES AND RELATED RESEARCH 6

2.1 Embedded System 6

2.2 Deep Learning 6

2.3 Object Detection 7

2.4 Human Detection 8

2.5 Human Tracking 9

2.6 COCO Dataset 10

2.7 MobileNet SSD 10

2.8 Centroid Tracking 11

2.9 Gstreamer 12

2.10 Line Notify 12
2.11 Ubuntu Screen 12
2.12 Remote VNC Access 13
2.13 Nvidia Jetson Nano 13
2.14 MicroSD Card 14
2.15 Dedicated Cooling Fan 14
2.16 Nano Electric Box IP65 15
2.17 HEPA Air Filter 15
CHAPTER 3: RESEARCH METHODS 16
3.1 Hardware Setup 16
3.1.1 MicroSD Card 17
3.1.2 Nvidia Jetson Nano 18
3.2 Software Setup 19
3.2.1 Create Swap File 19
3.2.2 OpenCV Installation 20
3.2.3 Jetson Inference 21
3.3 Software development 22
3.3.1 Gstreamer Pipeline Experimental Setup 22
3.3.2 Import Dependencies 23
3.3.3 Integrate Object Detection Model 24
3.3.4 Configuration and Initialization of Video capture 24
3.3.4.1 GStreamer Pipeline Configuration 24
3.3.4.2 Image Format Conversion 25

3.3.4.3 Video Capture 25
3.3.5 Human detection implementation 25
3.3.5.1 Frame Retrieval and Preprocessing 26
3.3.5.2 GPU Acceleration and Object Detection 26
3.3.5.3 Storage of Detected Human Bounding Boxes 27
3.3.5.4 Performance Tracking and Termination 27
3.3.6 Human Tracking with Centroid Tracker 27
3.3.6.1 Frame and Object Processing 28
3.3.6.2 Bounding Box 28
3.3.6.3 Processing and Updating Tracked Objects and Object ID List 28
3.3.6.4 Duration Tracking and Image Transmission for Detected Objects 29
3.3.7 Implementation of Non-Maximum Suppression 31
3.3.8 Display Frame 33
3.3.9 Centroid Tracker 33
CHAPTER 4: RESEARCH RESULT 39
CHAPTER 5: SUMMARY AND RECCOMMENDATIONS 40
REFERENCES 42

LIST OF TABLES

Table 1 Project timeline 5

LIST OF FIGURES

Figure 1 Scope of Development 2
Figure 2 Deep Learning 7
Figure 3 Object Detection 7
Figure 4 Centroid Tracking 11
Figure 5 Line Notify 12
Figure 6 VNC Viewer 13
Figure 7 Nvidia Jetson Nano 13
Figure 8 Micro SD Card 14
Figure 9 Cooling Fan 14
Figure 10 Nano Electric Box IP65 15
Figure 11 HEPA Air Filter 15
Figure 12 SD Card Formatter 17
Figure 13 Ether 17
Figure 14 Nvidia Jetson Nano Component 18
Figure 15 Nvidia Jetson Nano Connection 18
Figure 16 Script for Creating a Swap File 19
Figure 17 OpenCV Installation 20
Figure 18 Script for Jetson Inference 21
Figure 19 Download Pre-Trained DNN Models 21
Figure 20 Gstreamer Pipeline 22
Figure 21 Open Camera through Gstreamer 22
Figure 22 Import Dependencies 23

Figure 23 Define Model 24
Figure 24 GStreamer Pipeline Configuration 25
Figure 25 Video Capture 25
Figure 26 Human Detection Implementation 26
Figure 27 Human Tracking with Centroid Tracker 27
Figure 28 Update the Tracked Objects 29
Figure 29 Duration Tracking 30
Figure 30 PARINYA library 31
Figure 31 Line Token 31
Figure 32 Implementation of Non-Maximum Suppression 32
Figure 33 Display Frame 33
Figure 34 Centroid Tracker 33
Figure 35 Centroid Tracker script 1 34
Figure 36 Centroid Tracker script 2 35
Figure 37 Centroid Tracker script 3 36
Figure 38 Centroid Tracker script 4 37
Figure 39 Result from monitor 39
Figure 40 Result from Line Notify 40

CHAPTER 1: INTRODUCTION

1.1 Background and Significance

Intruders can pose a significant threat to the security of the establishment, resulting in financial losses and a damaged reputation. Camera systems have become an essential tool for surveillance and security in modern society. However, the effectiveness of these systems is heavily dependent on their ability to detect intruders in real-time. A delay in detecting an intruder can result in a gap in coverage, making it difficult to identify the culprit or recover stolen items.

Our project uses Jetson Nano, a powerful computer designed specifically for embedded applications and Raspberry Pi V2 camera which is a high-quality camera capable of capturing high-resolution images and video. Our purpose is to improve the security and surveillance capabilities of a variety of settings, such as homes, offices, and retail stores. By detecting intruders in real-time and sending images through the Line application, the system can alert users to potential security breaches and help them take appropriate action.

1.2 Development Objectives

This project's development objective is to create an embedded system for intruder detection, specifically targeted for red railways. The system will utilize hardware components such as Nvidia Jetson Nano, Raspberry Pi Camera v2, Micro SD (Standard Deviation) Card, Wi-Fi Module, Remote VNC access, and Ubuntu Screen for running terminal processes in the background.

Embedded systems are specifically designed for a particular task or application and are integrated into a larger system. In the case of intruder detection for red railways, an embedded system offers several advantages. Firstly, embedded systems are highly efficient in terms of power consumption, making them suitable for continuous operation in a railway environment. Secondly, embedded systems can perform real-time processing,

which is essential for the timely detection of intruders on railway premises. Additionally, embedded systems can be customized to meet specific requirements and can be designed to withstand harsh environmental conditions, making them robust and reliable for railway applications.

The main purpose of creating embedded systems for intruder detection in red railways is to enhance security measures and ensure railway premises' safety. Railway stations, depots, and tracks are critical infrastructures that require constant monitoring to prevent unauthorized access, theft, vandalism, and other security breaches. By deploying embedded systems for intruder detection, railway authorities can have a proactive and efficient security solution that can detect and alert the presence of intruders in real time.

1.3 Scope of Development

Our human detection system on Jetson Nano utilizes the SSD-MobileNet-v2 CNN (Convolutional Neural Networks) model as its backbone. The system utilizes a centroid tracking algorithm that tracks the centroid coordinate of the object in the frame. By comparing and calculating the distance between previous and current centroid coordinates and applying a predefined threshold value, the algorithm can determine whether to continue tracking the object or interpret it as a new object and begin tracking it as such. We employ a non-max suppression function to eliminate false detections that create multiple bounding boxes.

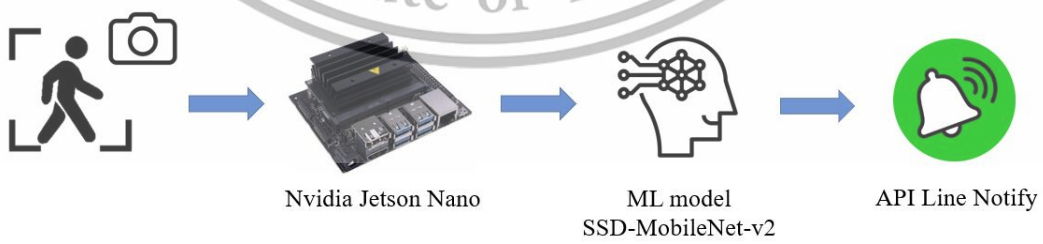

Figure 1 - Scope of development

1.4 Method of Conducting Research

1. Research related projects and compatible tools for the project
2. Hardware setup
3. Software setup
4. Coding and debugging
5. Testing the detection software with real world environment

1.5 Expecting Benefits

The development of an embedded system for intruder detection in red railways using the mentioned hardware components will bring several significant benefits to the railway infrastructure and its stakeholders.

1. Enhanced Security: The primary benefit of implementing this embedded system is the enhancement of security measures in red railways. By utilizing advanced hardware components such as the Nvidia Jetson Nano and Raspberry Pi Camera v2, the system can effectively detect intruders in real-time. This timely detection enables prompt response measures to prevent unauthorized access, theft, vandalism, and other security breaches, ensuring the safety of railway premises and passengers.
2. Proactive Intruder Detection: The embedded system's real-time processing capabilities, facilitated by the powerful Nvidia Jetson Nano, enable proactive intruder detection. The system can continuously analyze data from the Raspberry Pi Camera v2 and other sensors, quickly identifying potential threats and raising alerts. This proactive approach helps prevent security incidents from escalating and allows for swift intervention to mitigate risks.
3. Power Efficiency: Embedded systems, including the Nvidia Jetson Nano, are known for their power efficiency. By utilizing these energy-efficient hardware components, the intruder detection system can operate continuously without significant power consumption. This ensures a reliable

and uninterrupted surveillance solution that does not burden the existing power infrastructure of red railways.

4. Customizability: Embedded systems offer the advantage of customization to meet specific requirements. Railway authorities can tailor the embedded system to address the unique security challenges of their red railway infrastructure. Whether it is adjusting detection parameters, integrating additional sensors, or incorporating specific software features, the system can be customized to match the railway's security needs effectively.
5. Durability in Harsh Environments: The embedded system, including the Nano Electric BOX IP65 enclosure and the HEPA air filter, is designed to withstand the harsh environmental conditions commonly encountered in railway settings. The IP65 enclosure protects the system from dust, water, and other external elements, ensuring its durability and reliability even in challenging operational environments. The HEPA air filter prevents the ingress of particulates, maintaining a clean internal environment and protecting the hardware components from potential damage.
6. Remote Monitoring and Control: The inclusion of Remote VNC access in the embedded system allows for convenient remote monitoring and control. This feature enables administrators and security personnel to access and manage the system from a centralized location. They can monitor intruder detection data, receive real-time notifications and alerts, and take necessary actions promptly. This remote accessibility significantly enhances operational efficiency and facilitates timely response measures.
7. Long-term Cost Savings: By implementing an embedded system for intruder detection, red railways can potentially achieve long-term cost savings. The system's power efficiency minimizes energy consumption, reducing operational costs. Additionally, the system's proactive detection capabilities help prevent security incidents, mitigating potential financial losses associated with theft, vandalism, or unauthorized access.

1.6 Project Timeline

	February					March				April			
Task	week 1	week 2	week 3	week 4	week 5	week 6	week 7	week 8	week 9	week 10	week 11	week 12	week 13
Planning and research													
Hardware and software design													
Development													
Testing and debugging													
Implementation													

Table 1 - Project Timeline

CHAPTER 2: CONCEPT, THEORIES AND RELATED RESEARCH

2.1 Embedded System

Embedded system is combination of integrated components of hardware and software to perform tasks operate in real time environment within large system or device. Microcontrollers or microprocessors, memory, and input/output interfaces. The software is designed particularly to operate and manage the hardware components as well as to perform the desired functions of the embedded system.

The embedded systems area is constantly changing due to advances in hardware technology, software development methodologies, and the growing desire for smart and networked products. This involves incorporating wireless communication, Internet of Things (IoT) capabilities, artificial intelligence, and machine learning into embedded systems, allowing for more advanced features and connection across several domains.

2.2 Deep Learning

Deep learning are mathematical foundations and ideas that govern the operation of deep neural networks, a subset of machine learning models, which referred to as deep learning theory. Deep learning theory spans a wide range of concepts and strategies for training and optimizing deep neural networks for complicated pattern recognition tasks. Deep learning algorithms consist of interconnected layers, including an input layer, hidden layers, and an output layer. These layers form deep neural networks, which process input data by propagating it through a series of interconnected nodes called neurons. The neurons in each layer are connected to neurons in subsequent layers, enabling the flow of information through the network.

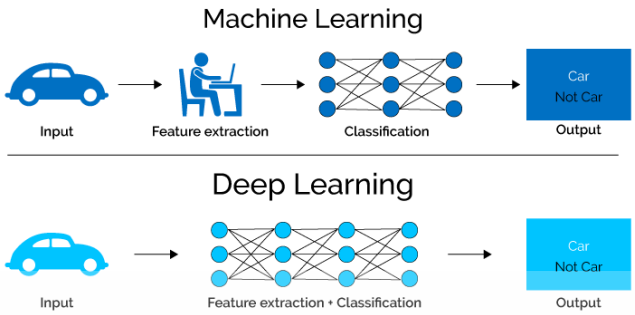

Figure 2 - Deep Learning Model

Overall, deep learning leverages the structure of interconnected layers and the iterative adjustment of weights and biases to learn from data and make predictions. It has proven to be a powerful approach in various domains, enabling the development of sophisticated models capable of handling complex tasks.

2.3 Object Detection

Object detection is a computer vision task that involves recognizing and localizing various items in an image or video. The goal is to offer precise bounding box coordinates that outline the positions of objects in addition to recognizing their existence. The algorithm of object detection must detect and classify multiple objects simultaneously and precisely locate them within the visual data in the form of images or videos.

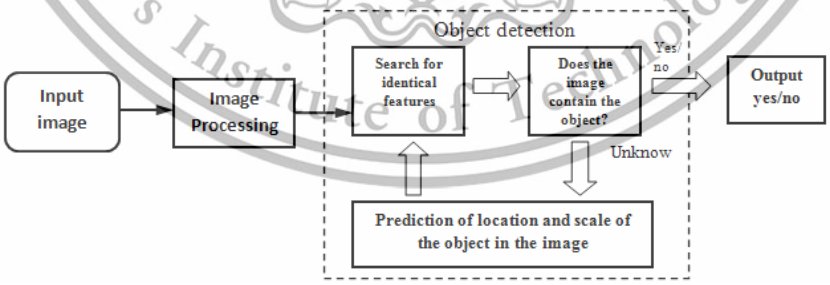

Figure 3 – Object Detection Flow

2.4 Human Detection

Human detection is an algorithm that aims to accurately detect and localize human bodies or body parts in images or video frames. These algorithms typically leverage machine learning techniques, particularly deep learning, to learn discriminative features that separates humans from the background or other objects in the frame.

The process of human detection involves training labeled data in the form of images or video frames with bounding boxes around the desired object. The training session uses machine learning model, such as deep neural network, to train to learn the discriminative features that differentiate humans from other objects. The model optimizes its parameters based on a defined objective function, aiming to minimize detection errors.

The following are examples of the key concepts in human detection.

2.4.1 Feature and Representation

Human detection algorithms rely on the extraction of relevant features or representations from the input data. These traits capture distinguishing human characteristics such as shape, texture, color, and motion. Haar-like features, Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and deep learning-based features produced from Convolutional Neural Networks (CNNs) are examples of commonly used features.

2.4.2 Machine learning

Machine learning plays a crucial role in human detection. To learn discriminative patterns that distinguish humans from non-human objects or backgrounds, supervised learning techniques such as Support Vector Machines (SVM), Random Forests, and Neural Networks are often used. These algorithms are trained on labeled datasets in which people are categorized as positive and non-humans as negative.

2.4.3 Classifier cascades

Classifier cascades are used to efficiently exclude non-human areas early in the detecting process. Cascades are made up of several stages, each with a classifier that filters out non-human areas based on increasing complexity. This hierarchical strategy speeds up the detection process by reducing the number of locations that must be thoroughly analyzed.

2.4.4 Metrics of Evaluation

Metrics of evaluation are used to measure the performance of human detection algorithms. Precision, recall, and the F1 score are common measures that assess the algorithm's capacity to recognize humans while reducing false positives and false negatives. To assess the accuracy of bounding box localization, additional metrics such as mean Average Precision and Intersection over Union are utilized.

2.5 Human Tracking

Human tracking is the process of monitoring and predicting the movements of individuals within a scene over time. Its goal is to accurately locate individuals within a video sequence to generate persistent paths, or trajectories, of the people. This process plays a significant role in applications such as video surveillance, activity recognition, human-computer interaction, and autonomous systems.

The initial operation in human tracking involves detecting individuals within each frame by using different tracking algorithms, including deep learning and other computer vision techniques. After that, tracking initialization occurs to assign a unique identification or label to each observed human and to construct an initial bounding box or region of interest around them.

The scenarios where individuals may be partially or completely obscured from view may happen which is one of the challenges of human tracking. Techniques such as multi-object tracking, and data association are employed to maintain tracking continuity and recover lost tracks.

2.6 COCO Dataset

The COCO (Common Objects in Context) dataset is an image recognition dataset to perform various tasks, including object detection, instance segmentation, and image captioning, serving as a dependable resource for training and evaluating various models.

Each image within the dataset is annotated with bounding boxes that accurately enclose instances of the object categories shown in the frame. In addition, the dataset includes segmentation masks for each object instance as well as descriptive label that explain the image content.

The COCO dataset offers comprehensive annotations, rendering it invaluable or advancing computer vision research.

2.7 SSD MobileNet V2

SSD MobileNetV2 is a prevalent object detection framework that combines the Single Shot Multi-Box Detector (SSD) architecture with the MobileNetV2 convolutional neural network backbone which is built to find a balance between high accuracy and computational efficiency.

The primary components of SSD MobileNetV2 are the backbone network and the detection head. The backbone network acts as a feature extractor, extracting relevant features from the input image.

Another component, SSD MobileNetV2's detection head, predicts object bounding boxes and their class probabilities. It consists of a set of prediction layers followed by a sequence of convolutional layers with diminishing spatial resolution. These prediction layers provide a set of default bounding boxes with varied aspect ratios and sizes.

The advantages of this model lie in its high detection accuracy and compatibility with mobile and embedded devices.

SSD MobileNetV2 has made significant inroads into real-time object identification applications such as autonomous driving, mobile robots, and intelligent surveillance systems. Its efficient architecture enables real-time object detection on devices with limited processing resources, making it ideal for deployment in scenarios requiring low latency and high accuracy.

2.8 Centroid Tracking

Centroid tracking is an effective algorithm for tracking moving objects, including humans. It works by assigning a unique ID to each object detected in a video frame, and then using the object's centroid to track its movement across subsequent frames. The algorithm consists of the following steps:

1. Detection: Objects are detected in the first frame of the video using a computer vision algorithm, such as YOLO or OpenCV. Each object is assigned a unique ID.
2. Initialization: bounding box is created around each detected object and calculates the centroid of the box.
3. Tracking: In each subsequent frame, the object is tracked by calculating the distance between its current centroid and the centroids of all previously detected objects. The object with the closest centroid is the same object and its ID is updated. If no object is found within a certain threshold, a new object is created with new have not been detected for a certain number of frames are deleted from the tracking list.

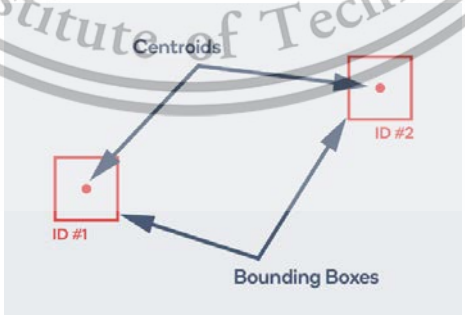

Figure 4 – Centroid Tracking

2.9 Gstreamer

GStreamer is an open-source multimedia framework that enables programmers to develop programs to handle multimedia tasks. The pipeline-based architecture of GStreamer makes it a modular and extensible multimedia framework. It provides a set of software components, known as elements, which can be combined to create complex multimedia processing workflows. Each element in the processing pipeline performs a specific task, such as capturing media data, decoding, encoding, filtering, or rendering.

2.10 Line Notify

Line Notify is a web service that allows users to send messages or notifications from various applications, services, or websites to their Line messaging app. It provides a convenient way to receive notifications from numerous services or applications directly to the user's Line account. It can be used for personal alerts, reminders, or integrating with other services to send updates.

Figure 5 – Line Notify

2.11 Ubuntu Screen

To efficiently run terminal processes in the background and enable the processing and analysis of intruder detection data, the project will utilize Ubuntu Screen. This software utility creates multiple virtual terminal sessions within a single terminal window, allowing for concurrent execution of various processes. Ubuntu Screen enhances the system's performance by enabling efficient multitasking, facilitating real-time data analysis, and streamlining the management of the intruder detection system.

2.12 Remote VNC Access

Remote VNC (Virtual Network Computing) access is a crucial feature that allows authorized users to access and control the embedded system from a remote location. With this capability, security personnel or administrators can monitor the system's status, review captured data, and make necessary adjustments or configurations as needed. Remote VNC access enhances the system's flexibility and enables centralized management of multiple embedded systems.

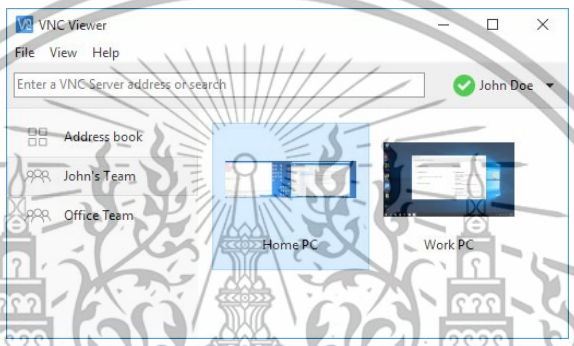

Figure 6 – VNC Viewer

2.13 Nvidia Jetson Nano

The Nvidia Jetson Nano is a powerful and energy-efficient single-board computer. It possesses impressive computational capabilities, making it suitable for real-time processing and machine-learning applications. With its onboard GPU, the Jetson Nano can efficiently perform tasks like object detection, classification, and tracking, making it an ideal choice for intruder detection in red railways.

Figure 7 - Nvidia Jetson Nano

2.14 MicroSD Card

The MicroSD card serves as a storage medium for the embedded system. It stores the operating system, software, and data required for the system's proper functioning. The high capacity and small form factor of Micro SD Cards make them an ideal choice for storing and accessing essential system components.

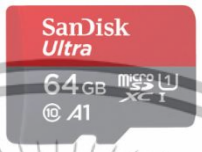

Figure 8 – MicroSD Card

2.15 Dedicated Cooling Fan for Jetson Nano

Given the high-performance nature of the Nvidia Jetson Nano, a dedicated cooling fan will be incorporated into the design. This fan ensures optimal thermal management, dissipating heat generated during operation and preventing potential overheating issues. By maintaining a stable operating temperature, the system can operate reliably and efficiently for prolonged periods, minimizing the risk of malfunctions or performance degradation.

Figure 9 - Cooling Fan

2.16 Nano Electric Box IP65

To ensure the embedded system's durability and protection against environmental factors, a Nano Electric BOX IP65 enclosure can be utilized. This enclosure is designed to provide excellent ingress protection, safeguarding the embedded system from dust, water, and other potential hazards. It enhances the system's resilience to challenging railway environments, allowing it to function reliably in diverse weather conditions and operational scenarios.

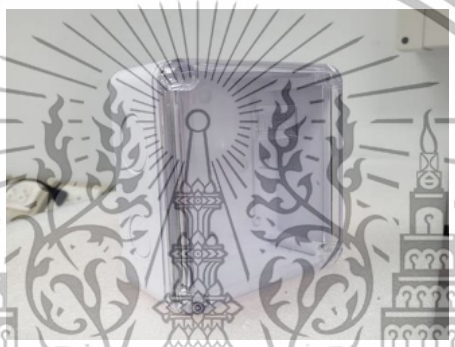

Figure 10 - Nano Electric Box IP65

2.17 HEPA Air Filter

In railway environments, dust and other particles can accumulate and potentially affect the performance and lifespan of the embedded system. Integrating a HEPA (High-Efficiency Particulate Air) filter into the system's design can help maintain a clean and dust-free environment within the enclosure, ensuring optimal functioning and longevity of the hardware components.

Figure 11 – HEPA Air Filter

CHAPTER 3: RESEARCH METHODS

3.1 Hardware Setup

This project's development objective is to create an advanced and robust embedded system for intruder detection in red railways. The system will utilize various hardware components to ensure effective surveillance and timely response to potential security threats. The list of hardware used in this project is shown below:

- NVIDIA Jetson Nano Developer Kit (4GB)
- Raspberry Pi NoIR Camera Board v2 (8MP)
- MicroSD Card 128GB
- Micro SD Card Reader
- Power Supply Applicable for Jetson Nano 5V/4A
- Cooling Fan
- Monitor with an HDMI connector
- USB keyboard
- USB mouse

By integrating these hardware components and software utilities, the developed embedded system will provide effective intruder detection capabilities in red railways. It will leverage the power of Nvidia Jetson Nano for real-time processing, utilize the Raspberry Pi Camera v2 for capturing visual data, rely on Micro SD Cards for storage, and enable remote access through VNC. This comprehensive solution will enhance railway security, ensuring timely identification and response to potential security threats.

3.1.1 MicroSD Card

Jetson Nano uses a microSD card for storing the operating system. We need a MicroSD card reader to connect the card and laptop in the setup process.

1. Download the 'Jetson Nano Developer Kit SD Card Image' file to our PC.
2. Write the image to our microSD card. The instructions will be different according to the PC operating system. For Windows, download and launch SD Card Formatter from the SD Association.

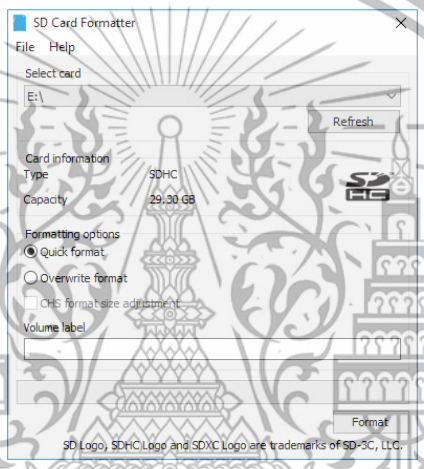

Figure 12 - SD Card Formatter

3. Install and launch Etcher for writing images to the microSD card. Then choose the zipped image file downloaded in step 1, choose a drive, and click 'Flash!'

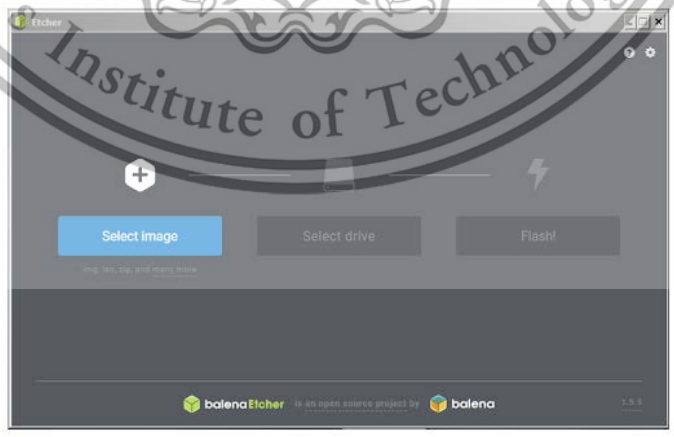

Figure 13 – Etcher

3.1.2 Nvidia Jetson Nano

After setup microSD card with system image already written to it. Insert the card into the slot on the underside of Jetson Nano module. We connect the power supply to the 5V/4A Power Jack. Then connect with monitor, keyboard, and mouse for initial setup (with display attached). Raspberry Pi NoIR Camera is connected through the MIPI CSI camera connector.

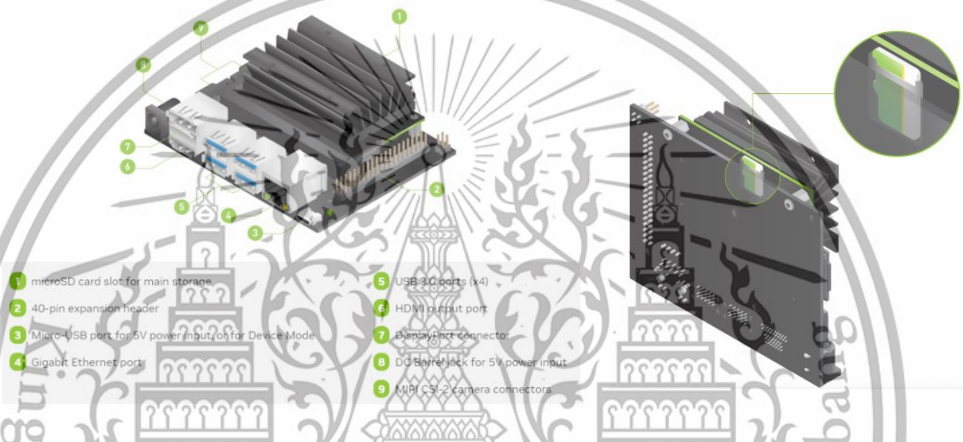

Figure 14 - Nvidia Jetson Nano Component

Figure 15 - Nvidia Jetson Nano Connection

For the first-time boost-up, the developer kit will take you through some initial setup, including:

- Review and accept the Nvidia Jetson Software End User License Agreement
- Select system language, keyboard layout, and time zone
- Create a username, password, and computer name
- Select APP partition size

Then it will automatically reboot to the Ubuntu desktop.

3.2 Software Setup

3.2.1 Create Swap File

Physical memory (RAM), primarily used for operational processes, cannot always handle the entire load. Therefore, we created a swap file, a virtual memory type, to prevent Jetson Nano from crashing. Memory swapping enhances the operating system's performance by efficiently switching between physical and virtual memory. In this project, we added a 4GB swap file to prevent Jetson Nano from running out of memory.

```
sudo fallocate -l 4G /var/swapfile
sudo chmod 600 /var/swapfile
sudo mkswap /var/swapfile
sudo swapon /var/swapfile
sudo bash -c 'echo "/var/swapfile swap swap defaults 0 0" >>
/etc/fstab'
```

Figure 16 - Script for Creating a Swap File

3.2.2 OpenCV Installation

The script installs OpenCV version 4.5.0 and requires dependencies for building OpenCV. These include libraries, utilities, and build tools such as cmake, python, and numpy. Then download and extract the OpenCV and OpenCV contrib source code from GitHub to the specified folder. The cmake command configures the build with various options, such as enabling CUDA and GStreamer support and specifying the location of the OpenCV contrib modules.

```
version="4.5.0"
folder="workspace"

echo "** Remove other OpenCV first"
sudo sudo apt-get purge *libopencv*


echo "** Install requirement"
sudo apt-get update
sudo apt-get install -y build-essential cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
sudo apt-get install -y python2.7-dev python3.6-dev python-dev python-numpy python3-numpy
sudo apt-get install -y libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libdc1394-22-dev
sudo apt-get install -y libv4l-dev v4l-utils qv4l2 v4l2ucp
sudo apt-get install -y curl


echo "** Download opencv-"${version}
mkdir $folder
cd ${folder}
curl -L https://github.com/opencv/opencv/archive/${version}.zip -o opencv-${version}.zip
curl -L https://github.com/opencv/opencv_contrib/archive/${version}.zip -o opencv_contrib-${version}.zip
unzip opencv-${version}.zip
unzip opencv_contrib-${version}.zip
cd opencv-${version}/


echo "** Building..."
mkdir release
cd release/
cmake -D WITH_CUDA=ON -D WITH_CUDNN=ON -D CUDA_ARCH_BIN="5.3,6.2,7.2" -D CUDA_ARCH_PTX="" -D OPENCV_GENERATE_PKGCONFIG=ON -D
make -j$(nproc)
sudo make install
```

Figure 17 - OpenCV Installation

3.2.3 Jetson Inference

The inferencing library (libjetson-inference) contains different deep-learning components intended to be run on Jetson with support for Python. It provides inference and real-time vision DNN libraries for NVIDIA using TensorRT to run optimized models on GPUs and PyTorch for training. Several pre-trained DNN (Deep Neural Networks) models are available for download and built in Jetson Nano.

Jetson inference requires some dependencies, including git, cmake, libpython3-dev, and numpy. After installing the prerequisites dependencies, we will clone the Jetson inference repository from GitHub. Then navigate to the jetson-inference directory and use the command to create and install Jetson inference.

```
sudo apt-get update
sudo apt-get install git cmake libpython3-dev python3-numpy
git clone --recursive --depth=1 https://github.com/dusty-nv/jetson-inference
cd jetson-inference
mkdir build
cd build
cmake ../
make -j$(nproc)
sudo make install
sudo ldconfig
```

Figure 18 - Script for Jetson Inference

In our project, we will download SSD Mobilenet-v2.

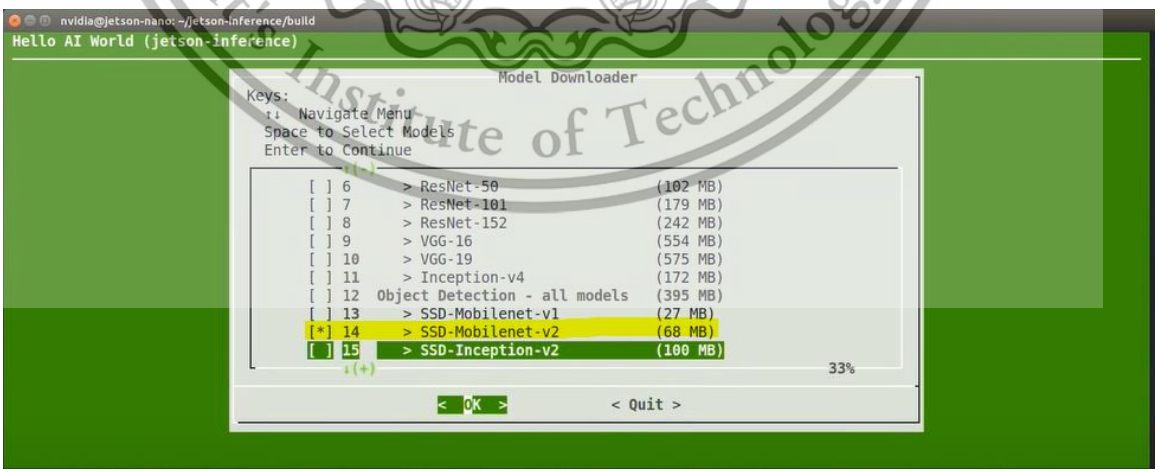

Figure 19 - Download Pre-Trained DNN Models

3.3 Software Development

3.3.1 Gstreamer Pipeline Experimental Setup

To evaluate the functionality of the Raspberry Pi v2 camera, an experimental setup was established using the Ubuntu terminal. The camera's capabilities were tested by constructing and executing gstreamer pipeline within the terminal environment. The terminal command, specifically gst-launch, was utilized along with various screen setting parameters to initialize the camera's operation.

```
gst-launch-1.0 nvarguscamerasrc ! nvoverlaysink
```

Figure 20 – Gstreamer Pipeline

Upon executing the command, the camera's video feed was successfully rendered on the connected monitor through the nvoverlaysink plugin.

Figure 21 – Open Camera through Gstreamer

3.3.2 Import Dependencies

The initial step in the script is to import all the necessary pre-installed dependencies to ensure the proper functioning of the environment. This includes importing the following libraries:

```
import numpy as np
import cv2
import jetson.inference
import jetson.utils
```

Figure 22 – Import Dependencies

1. NumPy: This library is used for numerical computations and provides efficient data structures and functions for handling arrays and matrices.
2. cv2 (OpenCV): This library is widely used for computer vision tasks and provides various functions and tools for image and video processing.
3. Jetson. Inference: This library is specific to the Jetson platform and provides high-level APIs for performing object detection, classification, and other deep learning-based inference tasks.
4. jetson.utils: This library complements Jetson. Inference and provides utility functions for image and video processing, such as loading images or capturing frames from a camera.

By importing these pre-installed dependencies, the script establishes the necessary environment and ensures that the required functionality and tools are available for subsequent steps, such as implementing human detection algorithms or performing inference tasks on the Jetson platform.

3.3.3 Integrate Object Detection Model

In this step, the object detection model is incorporated into the script, and the following code is utilized.

```
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
```

Figure 23 – Define Model

It creates an instance of the detectNet class from Jetson. Inference library. The chosen object detection model is ssd-mobilenet-v2. The threshold parameter specifies the confidence threshold. Objects with confidence above this threshold are considered as valid detections. In this case, a threshold value of 0.5 is set, indicating that objects must have a confidence score of at least 0.5 to be considered as detected.

The net variable serves as a reference to the initialized object detection network, which can be utilized in subsequent steps for processing images or video frames and obtaining object detection results.

3.3.4 Configuration and Initialization of Video capture

The pipeline configuration code is integrated into the script to configure the video capture functionality of the Raspberry Pi V2 camera. By doing this, the camera can be accessed, and video frames can be processed for subsequent tasks such as object detection or image analysis.

3.3.4.1 GStreamer Pipeline Configuration

The GSTREAMER_PIPELINE variable holds the GStreamer pipeline configuration. The pipeline begins with the nvarguscamerasrc element, which

accesses the camera source and captures frames in the NVMM memory format with a specified parameter.

```
GSTREAMER_PIPELINE = 'nvarguscamerasrc ! video/x-raw(memory:NVMM), width=3280, height=2464, format=(string)NV12, framerate=21/1 !
 nvvidconv flip-method=0 ! video/x-raw, width=960, height=616, format=(string)BGRx ! videoconvert ! video/x-raw, format=(string)BGR ! appsink'
```

Figure 24 – GStreamer Pipeline Configuration

3.3.4.2 Image Format Conversion

The pipeline then performs format conversion and manipulation operations on the captured frames. The nvvidconv element is used to flip the image, and videoconvert converts the image format from NVMM to BGR.

3.3.4.3 Video Capture

The cv2.VideoCapture function is utilized to create a video capture object. The GSTREAMER_PIPELINE is passed as the input argument which specifies that GStreamer is used as the video capture backend.

```
cap = cv2.VideoCapture(GSTREAMER_PIPELINE, cv2.CAP_GSTREAMER)
```

Figure 25 – Video Capture

3.3.5 Human Detection Implementation

In this step, the human detection script is implemented into the main script, which is designed to detect humans in real-time video streams. The script shown below uses the GStreamer pipeline setup and the selected object detection model, ssd-mobilenet-v2, to execute continuous frame processing.

```
while True:
        ret, frame = cap.read()
        img = frame[:,:,::-1]
        imgCuda = jetson.utils.cudaFromNumpy(frame)
        detector = net.Detect(imgCuda, overlay = "OVERLAY_NONE")
        total_frames = total_frames + 1
        rects = []
```

Figure 26 – Human Detection Implementation

3.3.5.1 Frame Retrieval and Preprocessing

Within the script, a while loop is employed to continuously retrieve frames from the video stream. The cap.read() function reads the next frame from the capture object cap. Each retrieved frame is then stored in the frame variable.

The color channels in the frame are reordered from BGR to RGB, and the resulting image is assigned to the 'img' variable.

3.3.5.2 GPU Acceleration and Object Detection

To increase the GPU acceleration, the jetson.utils.cudaFromNumpy() function is applied to convert the frame into a CUDA array format, represent as imgCuda variable. This format enhances the speed of object detection.

The object detection network or the net variable, is then applied to the CUDA array using the net.Detect() function. By specifying the overlay= "OVERLAY_NONE" parameter, the script ensures that no overlays are displayed on the frame, focusing solely on the human detection task. The results of detection are stored in the detector variable.

3.3.5.3 Storage of Detected Human Bounding Boxes

To track the detected humans in each frame, a list named rects is initialized. This list serves as a container to store the bounding boxes of the detected humans. The script appends the relevant bounding boxes to this list for further processing or visualization purposes.

3.3.5.4 Performance Tracking and Termination

By incrementing the 'total_frames' variable within each iteration of the loop, it tracks the total number of processed frames which then will be utilized for performance evaluation, such as calculating the frame rate or monitoring the overall efficiency of the human detection system.

3.3.6 Human Tracking with Centroid Tracker

The implementation incorporates the centroid tracker algorithm for tracking the detected humans. A list named rects is initialized to store the bounding boxes of the detected humans. Within a loop, each detected object is examined to determine if it corresponds to a person. If the object is classified as a person, the bounding box coordinates are extracted and appended to the rects list.

```
rects = []

for d in detector:
    className = net.GetClassDesc(d.ClassID)

    if className == "person":
        x1,y1,x2,y2 = int(d.Left), int(d.Top), int(d.Right), int(d.Bottom)
        rects.append([x1, y1, x2, y2])

objects = tracker.update(rects)
```

Figure 27 – Human Tracking with Centroid Tracker

The tracker.update(rects) function is then called to update the tracker's state and generate the tracked objects. The tracked objects are stored in the object variable, showing the tracked humans in the current frame.

3.3.6.1 Frame and Object Processing

The script continues to process frames and track humans in real-time, incrementing the total_frames variable to keep track of the processed frames. Further processing can be performed on the tracked objects, such as visualization, analysis, or integration with other systems or modules.

3.3.6.2 Bounding Box

In the initial steps, an empty list is initialized which will be used to store the bounding box coordinates of the detected humans. A loop is initiated to iterate over each object detected by the detector. The class name for each object is obtained using a function.

If the class name is identified as person, the bounding box coordinates (x1, y1, x2, y2) are extracted and converted to integer values, which are then applied to the list's variable as a four-element array [x1, y1, x2, y2]. Finally, the tracker is updated with the new set of bounding boxes using the "tracker.update(rects)" function.

3.3.6.3 Processing and Updating Tracked Objects and Object ID List

The code will update the tracked objects by extracting the bounding box coordinates and maintaining a list of unique object IDs. It ensures that the list remains up to date with the latest detected objects during the object tracking process.

```
for (objectId, bbox) in objects.items():
            x1, y1, x2, y2 = bbox
            #x1, y1, x2, y2, objectId = bbox
            x1 = int(x1)
            x2 = int(x2)
            y1 = int(y1)
            y2 = int(y2)

            if objectId not in obj_id_list:
                obj_id_list.append(objectId)
```

Figure 28 – Update the Tracked Objects

A loop is initiated to iterate over the items in the objects dictionary, which contains the tracked objects and their corresponding bounding box coordinates which the object ID (objectId) and the bounding box (bbox) in each item are extracted. The bounding box coordinates (x1, y1, x2, y2) are assigned to variables using tuple unpacking.

The code then checks if the objectId is present or not in the list. If the object is not on the list, it implies that it is a newly detected object. In this case, the objectId is appended to the list to keep track of the detected objects.

3.3.6.4 Duration Tracking and Image Transmission for Detected Objects

In this session, a time counter is implemented to measure the duration in seconds for objects belonging to the "person" class that appear in the frame.

```
if objectId not in obj_id_list:
                obj_id_list.append(objectId)
                dtime[objectId] = datetime.datetime.now()
                dwell_time[objectId] = 0

            else:
                current_time = datetime.datetime.now()
                old_time = dtime[objectId]
                time_diff = current_time - old_time
                dtime[objectId] = datetime.datetime.now()
                sec = time_diff.total_seconds()
                dwell_time[objectId] += sec


            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
            text = "{}|{}".format(objectId,int(dwell_time[objectId]))
            cv2.putText(frame, text, (x1, y1-5), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0, 255), 1)

            if int(dwell_time[objectId]) == 60:
                #line.sendimage(frame[:, :, ::-1])
                response = requests.post(url,headers=HEADERS,params={"imageFullsize": img})
                print(response)
```

Figure 29 – Duration Tracking

In this part the code will check if the object ID is not present in the object ID list, it indicates a newly detected object. In this case, the object ID is added to the list, and the current time is recorded as the initial time when object entered the camera view. Additionally, a dwell time counter for the object is initialized to zero.

For previously detected objects, the current time is obtained, and the recorded initial time for the object is retrieved from the time library. The time difference between the current time and the initial time is calculated to determine the elapsed duration in seconds. The current time is then updated as the new initial time for subsequent calculations. The elapsed duration is added to the dwell time counter for the object.

To indicate the tracked objects, a rectangle is drawn around the object's bounding box coordinates using the OpenCV library. Additionally, text is displayed near the top-left corner of the bounding box, indicating the object ID and the dwell time in seconds.

If the dwell time for an object reaches the predefined threshold of 60 seconds, a request is sent to transmit the image via the LINE Notify API which is imported from PARINYA library.

```
from parinya import LINE
```

Figure 30 – PARINYA library

To send the captured images through the Line notify platform, the Line token and Line Notify URL are defined as paths to facilitate the image transmission process. The Line token serves as an authentication mechanism, allowing us to access the Line API and send messages. The Line Notify URL specifies the endpoint to which the image will be sent.

```
line = LINE('iwjQcXhtNmrCYjLUc8G3TdIj1LNhO0UiNl4EyPGlVBf')
url = 'https://notify-api.line.me/api/notify'
```

Figure 31 – Line Token

3.3.7 Implementation of Non-Maximum Suppression

To enhance the accuracy of the detection algorithm, a non-maximum suppression technique is incorporated in the code. The purpose of non-maximum suppression is to eliminate redundant bounding boxes and retain only the most relevant and accurate bounding boxes for each detected object.

```python
def non_max_suppression_fast(boxes, overlapThresh):
    try:
        if len(boxes) == 0:
            return []
        if boxes.dtype.kind == "i":
            boxes = boxes.astype("float")
        pick = []
        x1 = boxes[:, 0]
        y1 = boxes[:, 1]
        x2 = boxes[:, 2]
        y2 = boxes[:, 3]
        area = (x2 - x1 + 1) * (y2 - y1 + 1)
        idxs = np.argsort(y2)
        while len(idxs) > 0:
            last = len(idxs) - 1
            i = idxs[last]
            pick.append(i)
            xx1 = np.maximum(x1[i], x1[idxs[:last]])
            yy1 = np.maximum(y1[i], y1[idxs[:last]])
            xx2 = np.minimum(x2[i], x2[idxs[:last]])
            yy2 = np.minimum(y2[i], y2[idxs[:last]])
            w = np.maximum(0, xx2 - xx1 + 1)
            h = np.maximum(0, yy2 - yy1 + 1)
            overlap = (w * h) / area[idxs[:last]]
            idxs = np.delete(idxs, np.concatenate(([last],
                                                   np.where(overlap > overlapThresh)[0])))
        return boxes[pick].astype("int")
    except Exception as e:
        print("Exception occurred in non_max_suppression : {}".format(e))
```

Figure 32 – Implementation of Non-Maximum Suppression

The non_max_suppression_fast function is designed to remove redundant bounding boxes and retain the most relevant ones based on a specified overlap threshold. It takes a list of bounding boxes and the overlapping threshold as input parameters. The algorithm starts by checking if the box list is empty and returns an empty list if it is. The bounding boxes are then sorted based on their y2 values in ascending order to prioritize the bottom-most boxes.

Within each iteration, the algorithm compares the coordinates of the selected bounding box with the remaining boxes to calculate the maximum and minimum coordinates of the overlapping region. It also computes the overlap ratio between the overlapping region and the area of the remaining boxes. Bounding boxes with an overlap ratio above the threshold are removed, ensuring that only non-overlapping boxes are retained.

The process continues until all the boxes have been processed, resulting in the selection of the final set of non-overlapping bounding boxes, which is returned as the output of the function. The implementation includes error handling to capture any exceptions that may occur during execution.

Overall, the non_max_suppression_fast function effectively applies non-maximum suppression to the provided bounding boxes, reducing redundancy, and improving the accuracy of object detection

3.3.8 Display Frame

In this part, the monitor will display a frame with additional information allowing for real-time monitoring and analysis of the video stream. The cv2.putText() function is used to draw text on the frame, specifically for displaying the frames per second (fps) information. The text is placed at the coordinates (5, 30) on the frame.

```
cv2.putText(frame, fps_text, (5, 30), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0, 255), 1)
cv2.imshow("Image", frame)
```

Figure 33 – Display Frame

After rendering the text on the frame, the updated frame is displayed using cv2.imshow(). The frame is displayed in a separate window.

3.3.9 Centroid Tracker

In this step centroid tracker code is integrated with the human detection code to allow for the automatic tracking of human objects in the frame. The tracker helps in maintaining the identity and continuity of the detected humans across frames, even when they temporarily disappear and reappear.

```
tracker = CentroidTracker(maxDisappeared=120, maxDistance=90)
```

Figure 34 – Centroid Tracker

The Centroid Tracker class is used to track objects in a video or image frame. The class has been instantiated with the parameters maxDisappeared and maxDistance to define the criteria for object tracking.

The maxDisappeared parameter determines the maximum number of consecutive frames an object can be marked as "disappeared" before it is considered as no longer being tracked. This parameter helps in handling situations where an object temporarily goes out of view or is occluded by other objects.

The maxDistance parameter sets the maximum distance between centroids of consecutive frames to associate them as the same object. If the distance between centroids exceeds this threshold, the object is considered as a new or different one. This parameter helps in handling object movements and tracking accuracy.

```
class CentroidTracker:
    def __init__(self, maxDisappeared=200, maxDistance=50):
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()
        self.bbox = OrderedDict()  # CHANGE

        self.maxDisappeared = maxDisappeared
        self.maxDistance = maxDistance

    def register(self, centroid, inputRect):
        self.objects[self.nextObjectID] = centroid
        self.bbox[self.nextObjectID] = inputRect  # CHANGE
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1

    def deregister(self, objectID):
        del self.objects[objectID]
        del self.disappeared[objectID]
        del self.bbox[objectID]  # CHANGE
```

Figure 35 – Centroid Tracker Script 1

The tracker compares the centroids of the detected objects with the existing tracked objects, determines the association based on proximity, and maintains the object tracks over time. The tracker also handles scenarios where objects disappear for a certain number of frames and deregisters them from tracking.

The register method takes a centroid and an input rectangle as parameters. It assigns a unique object ID to the centroid, adds it to the objects dictionary along with the corresponding bounding box and sets the disappearance count to 0. The object ID is then incremented for the next object.

The deregister method removes an object from the tracking system based on its object ID. It deletes the corresponding entries from the objects, disappeared, and bbox dictionaries.

In the next step, the list of bounding boxes is updated in function update. It takes list function takes a list of bounding box coordinates rects as input and performs an update on the object tracking algorithm.

```
if len(rects) == 0:
    for objectID in list(self.disappeared.keys()):
        self.disappeared[objectID] += 1

        if self.disappeared[objectID] > self.maxDisappeared:
            self.deregister(objectID)
    return self.bbox
```

Figure 36 – Centroid Tracker Script 2

The function checks if the list is empty. If it is, it indicates that no objects are detected in the current frame. In this case, the function iterates over the disappeared dictionary to increase the disappearance count for each tracked object. If an object has exceeded the maxDisappeared threshold, it is removed from the tracking algorithm. Finally, the function returns the bounding box dictionary (bbox).

If the list is not empty, the function proceeds to process the bounding box coordinates. It initializes two empty lists which are inputCentroids to store the centroid coordinates and inputRects to store the bounding box coordinates.

```
inputCentroids = np.zeros((len(rects), 2), dtype="int")
inputRects = []
for (i, (startX, startY, endX, endY)) in enumerate(rects):
    cX = int((startX + endX) / 2.0)
    cY = int((startY + endY) / 2.0)
    inputCentroids[i] = (cX, cY)
    inputRects.append(rects[i])  # CHANGE

if len(self.objects) == 0:
    for i in range(0, len(inputCentroids)):
        self.register(inputCentroids[i], inputRects[i])  # CHANGE
else:
    objectIDs = list(self.objects.keys())
    objectCentroids = list(self.objects.values())
    D = dist.cdist(np.array(objectCentroids), inputCentroids)
    rows = D.min(axis=1).argsort()
    cols = D.argmin(axis=1)[rows]
    usedRows = set()
    usedCols = set()
```

Figure 37 – Centroid Tracker Script 3

The function then iterates over each bounding box in the rects list. For each bounding box, it calculates the centroid coordinates by finding the midpoint of the top-left and bottom-right corners. The centroid coordinates (cX, cY) are appended to the inputCentroids list, and the original bounding box coordinates are appended to the inputRects list.

Next, the function checks if there are any existing objects in the objects dictionary. If the dictionary is empty, it means that no objects are currently being tracked. In this case, the function iterates over the inputCentroids and inputRects lists and registers each centroid and bounding box pair by calling the register function.

If there are existing objects in the objects dictionary, the function performs object matching between the existing objects and the newly detected centroids. The function then performs sorting based on the minimum distances (min(axis=1)) in the distance matrix which then sort results as array.

The function iterates over the sorted indices and matches each object centroid with the nearest input centroid. If it is, the object's centroid and bounding box coordinates are updated with the new values, and the disappearance count is reset to 0.

If the number of existing objects is greater than or equal to the number of input centroids, the function iterates over the unused rows, increments the disappearance count for each corresponding object, and deregisters objects that have exceeded the maxDisappeared threshold.

If the number of input centroids is greater than the number of existing objects, the function iterates over the unused columns and registers new objects by calling the register function with the corresponding input centroid and bounding box. Finally, the function returns the updated bounding box dictionary (bbox).

```
        for (row, col) in zip(rows, cols):
            if row in usedRows or col in usedCols:
                continue

            if D[row, col] > self.maxDistance:
                continue

            objectID = objectIDs[row]
            self.objects[objectID] = inputCentroids[col]
            self.bbox[objectID] = inputRects[col]  # CHANGE
            self.disappeared[objectID] = 0
            usedRows.add(row)
            usedCols.add(col)
        unusedRows = set(range(0, D.shape[0])).difference(usedRows)
        unusedCols = set(range(0, D.shape[1])).difference(usedCols)

        if D.shape[0] >= D.shape[1]:
            for row in unusedRows:
                objectID = objectIDs[row]
                self.disappeared[objectID] += 1

                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)
        else:
            for col in unusedCols:
                self.register(inputCentroids[col], inputRects[col])
    return self.bbox
```

Figure 38 – Centroid Tracker Script 4

If there are existing tracked objects, the code calculates the pairwise distance (D) between the centroid coordinates of the existing objects and the input objects using the cdist function. It then performs row and column sorting on D to find the minimum distances and their corresponding indices. The algorithm uses a set-based approach (usedRows and usedCols) to ensure each input and existing object is matched only once based on the minimum distance criterion.

For each matched pair of indices, the code checks if the distance between the objects exceeds the maximum allowed distance (maxDistance). If it does, the matching is skipped, and the algorithm moves on to the next pair.

If the distance is within the acceptable range, the algorithm updates the existing object's centroid and bounding box with the corresponding input object's information. The disappearance count for the object is reset to zero since it is detected in the current frame.

The code identifies the unused rows and columns, i.e., the unmatched existing objects and unmatched input objects, respectively. If the number of existing objects is greater than or equal to the number of input objects, the unused rows (existing objects) are iterated over, and their disappearance count is incremented. If an existing object has reached the maximum disappearance count, it is deregistered.

Conversely, if the number of input objects is greater than the number of existing objects, the unused columns (input objects) are iterated over, and the corresponding input objects are registered as new tracked objects. Finally, the code returns the updated of the bounding box information for each tracked object.

CHAPTER 4: RESEARCH RESULT

The night vision camera we use, the Raspberry Pi NoIR v2, is intended to improve visibility in low-light conditions, enabling efficient surveillance and monitoring. However, the output of these cameras frequently displays in pink, a characteristic feature of night vision technology.

When a human presence is detected in the camera's frame, the system initiates a tracking process and assigns a unique ID to each individual. This feature enables accurate identification and distinguishes between multiple individuals within the frame. Furthermore, it provides a confidence value, indicating the level of certainty in the accuracy of the identification process. This value helps the overall system effectiveness by providing an additional precise identification and monitoring indicator.

The system effectively monitors the duration of each individual's presence within the camera's field of view. It accurately tracks and records the time each person remains within the frame. We will implement the system at the red railway, where individuals are not typically present. The system has been designed to identify any person present within the frame over a predefined period as an intruder. It will capture an image of the suspecting individual as an intruder and promptly transmit it to the relevant authorities via line notification.

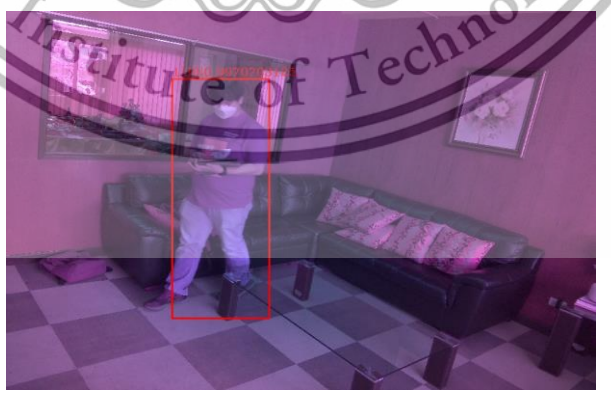

Figure 39 - Result from monitor

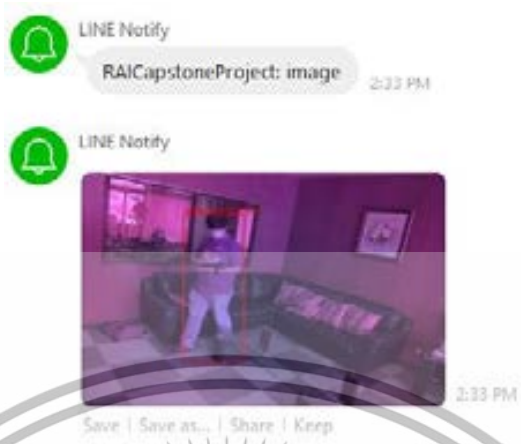

Figure 40 - Result from Line Notify

We conducted experiments to validate the embedded system's effectiveness for intruder detection in red railways. In the initial phase, we implement the embedded system for detecting and tracking the people in the room. The project is tested remotely using a VNC application to monitor its performance. We evaluated the system's performance based on the following metrics:

1. Detection Accuracy: The embedded system successfully identified people with high accuracy in various scenarios. With an overall detection accuracy of 78.796%, ensuring reliable intruder detection capabilities.

   Human tracking was tested from April 12th to May 2nd with 228 captured images from line notifications. The accuracy is calculated based on those images using the following equation:

   - True Positive (TP): The number of correctly detected human
   - False Positive (FP): The number of incorrectly detected human
   - False Negative (FN): The number of missing detected human
   - True Negative (TN): Not applicable for object detection/tracking

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{FP}}$$

$$= \frac{301}{301 + 59 + 22 + 0}$$

$$= \frac{301}{382} = 0.78795812\ (78.795\%)$$

2. Response Time: The response time was evaluated by measuring the time taken from intruder detection to generate an alert or trigger an appropriate response. With an average response time of 5 seconds, the system consistently displayed rapid response times. This quick reaction enables prompt action, averting potential security breaches.
3. Power Consumption: Power Consumption of the embedded system was measured to evaluate energy efficiency. The Nvidia Jetson Nano enhanced the system's efficiency because of its low power consumption.

CHAPTER 5: SUMMARY AND RECCOMMENDATIONS

In conclusion, our project successfully developed an embedded system using Nvidia Jetson Nano and Raspberry Pi Camera v2 for security and surveillance applications. Our testing and evaluation demonstrate that the system can detect and track humans accurately in real-world scenarios, with some errors related to the centroid tracking method.

The development of an embedded system for intruder detection in red railways using the mentioned hardware components brings benefits to ensuring the safety of railway premises, including enhanced security, proactive threat detection, power efficiency, customizability, durability in harsh environments, remote monitoring and control, and long-term cost savings.

Further research could include exploring alternative tracking methods to address the limitations of the centroid tracking algorithm and incorporating more advanced features such as facial recognition for enhanced security measures. Additionally, the system's camera could be upgraded to an IP camera to improve its overall performance and compatibility with other systems.

CHAPTER 6: REFERENCES

AC8265 Wireless NIC for Jetson Nano, WiFi / Bluetooth. (n.d.). Retrieved from WaveShare: https://www.waveshare.com/wireless-ac8265.htm

Cameras for Jetson. (n.d.). Retrieved from ArduCam: https://www.arducam.com/jetson-nano-xavier-nx-camera-imx219-imx477-arducam/

Dedicated Cooling Fan for Jetson Nano. (n.d.). Retrieved from Arduitronics: https://www.arduitronics.com/product/4232/dedicated-cooling-fan-for-jetson-nano-pwm-adjustment-%E0%B8%82%E0%B8%AD%E0%B8%87%E0%B9%81%E0%B8%97%E0%B9%89%E0%B8%88%E0%B8%B2%E0%B8%81-waveshare

Embedded Vision cameras. (n.d.). Retrieved from e-con Systems: https://www.e-consystems.com/embedded-vision-cameras.asp

Get Started With Jetson Nano Developer Kit. (n.d.). Retrieved from Nvidia Developer: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#:~:text=The%20Jetson%20Nano%20Developer%20Kit,with%20operating%20system%20and%20software.

HEPA FILTER. (n.d.). Retrieved from Aircontrol And Technology: http://www.aircontrol.co.th/Content.aspx?subheadmenu=true&subheadid=H000000003&id=M000000013

How to Set Up the NVIDIA Jetson Nano Developer Kit. (2021, April 2). Retrieved from Automatic Addison: https://automaticaddison.com/how-to-set-up-the-nvidia-jetson-nano-developer-kit/

Jetson Nano Developer Kit. (n.d.). Retrieved from Nvidia Developer: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

Memon, Z. (2020). *How to Use Screen Command on Ubuntu*. Retrieved from linuxhint: https://linuxhint.com/screen_command_ubuntu/

NANO Electric® NANO-202CW. (n.d.). Retrieved from Chaly shop: https://www.chalyshop.com/product/353/nano-electric%C2%AE-nano-202cw-%E0%B8%81%E0%B8%A5%E0%B9%88%E0%B8%AD%E0%B8%87%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B8%99%E0%B9%89%E0%B8%B3%E0%B8%9E%E0%

B8%A5%E0%B8%B2%E0%B8%AA%E0%B8%95%E0%B8%B4%E0%B8%81-%E0%B8%9D%E0%B8%B2%E0%B

RPi NoIR Camera V2, Supports Night Vision. (n.d.). Retrieved from WaveShare: https://www.waveshare.com/rpi-noir-camera-v2.htm

Shakhadri, S. A. (2021, December 10). *Complete Guide to People Counting and Tracking: End-to-end Deep Learning Project*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/11/complete-guide-to-people-counting-and-tracking-end-to-end-deep-learning-project/

VNC viewer. (n.d.). Retrieved from RealVNC: https://www.realvnc.com/en/connect/download/viewer/windows/

Waters, D. A. (n.d.). *Introduction to Embedded Systems*. Retrieved from SlidePlayer: https://slideplayer.com/slide/9324550/

Embedded System for Intruder Detection

Pattaratorn Soranathavornkul,
Puhnyanuj Smizdhanond,
Thaninrath Thiraphotiwat
Dept. of Robotics and AI Engineering
King Mongkut's Institute of Technology Ladkrabang
62011191@kmitl.ac.th,
62011225@kmitl.ac.th,
62011276@kmitl.ac.th

***Abstract*—Intruder detection is a critical aspect of security systems, and real-time embedded systems play a significant role in achieving efficient and timely responses. The project incorporates hardware and software components to identify unauthorized individuals rapidly and accurately. The primary objective is to enhance security and surveillance capabilities by developing an embedded system for real-time intruder detection targeted at red railways.**

Keywords— Embedded System, Intruder Detection

I. INTRODUCTION

Intruders can pose a significant threat to the security of the establishment, resulting in financial losses and a damaged reputation. Camera systems have become an essential tool for surveillance and security in modern society. However, the effectiveness of these systems is heavily dependent on their ability to detect intruders in real-time. A delay in detecting an intruder can result in a gap in coverage, making it difficult to identify the culprit or recover stolen items.

Our project uses Jetson Nano, a powerful computer designed specifically for embedded applications and Raspberry Pi V2 camera which is a high-quality camera capable of capturing high-resolution images and video. Our purpose is to improve the security and surveillance capabilities of a variety of settings, such as homes, offices, and retail stores. By detecting intruders in real-time and sending images through the Line application, the system can alert users to potential security breaches and help them take appropriate action

II. THEORIES

A. *Embedded System:*

Embedded systems encompass the integration of hardware and software components to perform real-time tasks within larger systems or devices. These systems consist of microcontrollers or microprocessors, memory, and input/output interfaces. As technology advances, embedded systems continuously evolve, incorporating wireless communication, artificial intelligence, and machine learning to enable more sophisticated features and connectivity across domains.

B. *Human Detection:*

Human detection algorithms aim to accurately detect and localize human bodies or body parts in images or video frames. These algorithms leverage machine learning techniques, particularly deep learning, to learn discriminative features that distinguish humans from the background or other objects in the scene.

1) Process: Human detection involves training labeled data, such as images or video frames, with bounding boxes around the desired object. Machine learning models, such as deep neural networks, are trained to learn discriminative features that differentiate humans from other objects. These models optimize their parameters based on a defined objective function to minimize detection errors.

2) Key concept:

a) Feature and Representation:

Human detection algorithms rely on the extraction of relevant features or representations from the input data. These features capture distinguishing human characteristics such as shape, texture, color, and motion. Commonly used features include Haar-like features, Histogram of Oriented Gradients (HOG) and deep learning-based features generated from Convolutional Neural Networks (CNNs).

b) Machine learning:

Machine learning techniques play a crucial role in human detection. Supervised learning algorithms such as Support Vector Machines (SVM), Random Forests, and Neural Networks are employed to learn discriminative patterns that separate humans from non-human objects or backgrounds.

C. *Human Tracking*

Human tracking involves monitoring and predicting the movements of individuals within a scene over time. The goal is to accurately locate individuals and generate persistent paths or trajectories. Human tracking plays a vital role in applications such as video surveillance and autonomous systems.

1) Process: Human tracking begins with the detection of individuals within each video frame using various tracking algorithms, including deep learning and computer vision techniques. Tracking initialization assigns a unique identification or label to each observed human and constructs an initial bounding box or region of interest around them.

2) Centroid tracker: Centroid tracking is an effective algorithm for tracking moving objects, including humans, in video sequences. This algorithm assigns a unique identification (ID) to each object detected in the initial frame and utilizes the object's centroid to track its movement across subsequent frames. The algorithm follows the following steps:

a) Detection: In the first frame of the video, a computer vision algorithm such as YOLO or OpenCV is used to detect objects. Each detected object is assigned a unique ID, enabling its individual tracking.

b) Initialization: A bounding box is created around each detected object, encompassing its spatial extent. The centroid of the bounding box is then calculated, representing the center of the object's position.

c) Tracking: In each subsequent frame, the algorithm tracks the objects by comparing the distance between the current centroid of each detected object and the centroids of all previously tracked objects. The algorithm determines the object with the closest centroid, indicating that it is the same object being tracked.

During the tracking process, if an object cannot be matched to any previously tracked object within a certain threshold, it is considered a new object.

D. SSD MobileNet V2:

SSD MobileNet V2 is a widely adopted object detection framework that combines the Single Shot Multi-Box Detector (SSD) architecture with the MobileNetV2 convolutional neural network backbone. This integration aims to achieve a balance between high accuracy and computational.

III. METHODOLOGY

This section describes the methodology followed for the implementation of the embedded system for intruder detection in red railways. The methodology encompasses both the hardware configuration and the software installation and configuration. The steps involved in setting up the system are outlined below:

A. System Setup

1) System Hardware: The hardware setup involved integrating the necessary components to ensure effective surveillance capabilities for intruder detection in red railways. The configuration included the following components:

- *NVIDIA Jetson Nano Developer Kit*: The Jetson Nano, a compact and powerful embedded system, served as the main processing unit for the intruder detection system.
- *Raspberry Pi NoIR Camera Board v2*: The NoIR camera board, connected to the Jetson Nano, provided the visual input for object detection and tracking.
- *MicroSD card*: A high-capacity MicroSD card was used for storing the operating system and software dependencies.
- *Cooling fan*: To prevent overheating, a cooling fan was installed to maintain optimal operating temperatures.

Figure 1 – Hardware component assembly

By assembling these hardware components, the system was prepared for the subsequent software installation and configuration.

2) Software Installation and Configuration: The software setup involved installing and configuring the necessary components for intruder detection. The following steps were performed:.

a) *System Optimization:*

To optimize system performance and prevent the Jetson Nano from running out of memory during resource-intensive tasks, a 4GB swap file was created. This ensured stability and efficient operation of the system.

Computer vision libraries were installed on the Jetson Nano to facilitate advanced vision processing capabilities.

b) Object Detection Framework:

The Jetson Inference library, specifically designed for NVIDIA Jetson platforms, was acquired, and installed on the Jetson Nano. This library provided efficient and accurate object detection functionalities. This library provided efficient and accurate object detection functionalities. The SSD Mobilenet-v2 model was selected within the Jetson Inference library for object detection tasks.

By completing these software installation and configuration steps, the system was ready for the subsequent stages of the experiment, evaluation, and implementation of object detection and tracking functionalities.

B. Object Detection Implementation

1) Importing Dependencies: All necessary dependencies were imported into the project. These libraries provided essential functionalities and utilities required for object detection and tracking tasks.

2) Integration of object detection model: The SSD Mobilenet-v2 object detection model was seamlessly integrated into the embedded system. Key parameters, such as the confidence threshold for object detection, were set to appropriate values. The object detection network was initialized, enabling the system to identify and locate objects of interest accurately.

3) Video Capture Configuration and Initialization: The GStreamer pipeline, a versatile multimedia framework, was configured to facilitate video capture from the Raspberry Pi NoIR Camera Board v2. This involved setting up the necessary format conversion and configuring video capture properties. The widely adopted OpenCV function, cv2.VideoCapture, was employed to initialize the video capture process and obtain the camera feed for subsequent analysis.

4) Human detection: The human detection module was implemented to identify and localize human subjects within the captured video frames. The process involved retrieving video frames from the camera feed, utilizing the power of the GPU for accelerated processing, performing object detection using the SSD Mobilenet-v2 model, storing the detected human bounding boxes for further analysis, and tracking the system's performance throughout the process.

5) Human Tracking with Centroid Tracker: To enable the continuous tracking of detected humans across consecutive frames, the centroid tracker algorithm was employed. This algorithm allowed for the association of object detections with corresponding tracks over time.

By updating the tracked objects and processing the bounding box coordinates using the centroid tracker, the system achieved robust and accurate human tracking capabilities.

IV. RESEARCH RESULT

The system employs human detection and tracking, assigning unique IDs to individuals in the camera's frame. The implemented system effectively monitors the duration of individuals within the camera's field of view and accurately records their presence. It is intended to be deployed at the red railway, where individuals are not typically present. The system is designed to identify any person detected within the frame over a predefined period as an intruder. In such cases, an image of the suspicious individual is captured as evidence of intrusion and promptly transmitted to the relevant authorities using line notification.

Figure 2– Experimental result

Experiments were conducted to validate the effectiveness of the embedded system in detecting intruders. The initial phase involved implementing the system for detecting and tracking people in a room, with remote monitoring via a VNC application. Performance evaluation was based on the following metrics:

1. *Detection Accuracy:* The embedded system achieved an overall detection accuracy of 78.796% in various scenarios, ensuring reliable intruder detection capabilities. To evaluate the system's performance, human tracking was tested from April 12th to May 2nd with 228 captured images from line notifications. The accuracy is calculated based on those images using the following equation:

- True Positive (TP): The number of correctly detected human
- False Positive (FP): The number of incorrectly detected human
- False Negative (FN): The number of missing detected human
- True Negative (TN): Not applicable for object detection/tracking

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{FP}}$$

$$= \frac{301}{301 + 59 + 22 + 0}$$

$$= \frac{301}{382} = 0.78795812\ (78.795\%)$$

2. *Response Time:* The system consistently displayed a rapid average response time of 5 seconds from intruder detection to generating an alert or triggering a response.
3. *Power Consumption:* Power Consumption of the embedded system was measured to evaluate energy efficiency. The Nvidia Jetson Nano enhanced the system's efficiency because of its low power consumption.

V. SUMMARY AND RECCOMMENDATIONS

In conclusion, our project successfully developed an embedded system using Nvidia Jetson Nano and Raspberry Pi Camera v2 for security and surveillance applications. Our testing and evaluation demonstrate that the system can detect and track humans accurately in real-world scenarios, with some errors related to the centroid tracking method.

The development of an embedded system for intruder detection in red railways using the mentioned hardware components brings benefits to ensuring the safety of railway premises, including enhanced security, proactive threat detection, power efficiency, customizability, durability in harsh environments, remote monitoring and control, and long-term cost savings.

Further research could include exploring alternative tracking methods to address the limitations of the centroid tracking algorithm and incorporating more advanced features such as facial recognition for enhanced security measures. Additionally, the system's camera could be upgraded to an IP camera to improve its overall performance and compatibility with other systems.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude and appreciation to Assoc. Prof. Somyot Kaitwanidvilai for his invaluable guidance, support, and expertise throughout the duration of this project. His expertise and commitment to excellence have been crucial in developing the embedded system for intruder detection in red railways.

Assoc. Prof. Somyot Kaitwanidvilai's profound knowledge and experience in embedded systems have provided us with invaluable insights at every stage of the project. His mentorship and feedback have greatly refined our ideas and ensured the effectiveness of the proposed solution.

We would like to extend our gratitude to the institution and department for providing the necessary resources and facilities to carry out this project. The support and encouragement received from the faculty and staff have been instrumental in our progress and achievements.

REFERENCES

[1] Boonchan, S. (2018, May 30). Line notify : ส่งภาพ สติ๊กเกอร์ และข้อความ ด้วย python. Retrieved from https://jackrobotics.me/line-notify/

[2] Cochard, D. (2021, May 25). MobilenetSSD: A Machine Learning Model for Fast Object Detection. Retrieved from Medium: https://medium.com/axinc-ai/mobilenetssd-a-machine-learning-model-for-fast-object-detection-37352ce6da7d

[3] How to Set Up the NVIDIA Jetson Nano Developer Kit. (2021, April 2). Retrieved from Automatic Addison: https://automaticaddison.com/how-to-set-up-the-nvidia-jetson-nano-developer-kit/

[4] Human Detection and Tracking. (n.d.). Retrieved from SpringerLink: https://link.springer.com/referenceworkentry/10.1007/978-0-387-73003-5_35

[5] Shakhadri, S. A. (2021, December 10). Complete Guide to People Counting and Tracking: End-to-end Deep Learning Project. Retrieved from AnalyticsVidhya: https://www.analyticsvidhya.com/blog/2021/11/complete-guide-to-people-counting-and-tracking-end-to-end-deep-learning-project/