**A**

**PROJECT REPORT ON**

# "OBJECT DETECTION AND COLLISION AVOIDANCE ALGORITHM "

SUBMITTED TO SAVITRIBAI PHULE PUNE UNIVERISTY FOR THE FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

## BACHELOR OF ENGINEERING
## IN
## ELECTRONICS AND TELECOMMUNICATION

Submitted by

**Prajwal Mahanawar (Exam No.B190073092 )**
**Akhilesh Shinde (Exam No.B190073130)**
**Aaditya Tiwari (Exam No.B190073001)**

Under the Guidance of

**Prof. Dr. P.G. Shete**


**Department of Electronics and Telecommunication Engineering**
**Pune Vidyarthi Griha's College of Engineering &Technology &G. K. Pate(Wani) Institute of Management, Pune -411009**

Academic Year 2023-2024

# Undertaking

We, the students of Department of Electronics and Telecommunication Engineering, PVG's College of Engineering and Technology & G. K. Pate (Wani) Institute of Management, Parvati, Pune-411009

| | | | |
|---|---|---|---|
| 1. | Prajwal Mahanawar | Exam Seat No.B190073092 | Signature |
| 2. | Akhilesh Shinde | Exam Seat No. B190073130 | Signature |
| 3. | Aaditya Tiwari | Exam Seat No. B190073001 | Signature |

do hereby undertake on 10.04.2024 the following:

1. We are aware of the **University Grants Commission (Promotion Of Academic Integrity and Prevention of Plagiarism In Higher Educational Institutions) Regulations, 2018 and Plagiarism Policy of Savitribai Phule Pune University Pune,**
2. We are also aware that **Department of Electronics and Telecommunication Engineering, PVG's College of Engineering and Technology & GKPIM , Parvati, Pune** has established **Departmental Academic Integrity Panel (DAIP)** as per **UGC regulations 2018** as mentioned above in (1),
3. We are aware of the consequences if found guilty of doing any act of plagiarism defined in the UGC regulations 2018 as mentioned above in (1),
4. We shall abide by the rules, regulations and code of conducts for the students of UGC, SPPU and Department of E&TC,
5. We further undertake and declare that the thesis submitted by us is scanned using anti-plagiarism software as decided by the department and report of the same has been submitted and is free from any kind of plagiarism mentioned above (1) in **UGC REGULATIONS, 2018,**
6. I understand that non-compliance of the **Academic Integrity and Prevention of Plagiarism** may results in **disciplinary action on us as per the University Grants Commission (Promotion of Academic Integrity and Prevention of Plagiarism in Higher Educational Institutions) Regulations, 2018.**


**Department of Electronics and Telecommunication Engineering**
**Pune Vidyarthi Griha's College of Engineering &Technology &G. K. Pate(Wani) Institute of Management,**
**Pune -411009**

# CERTIFICATE

This is to certify that the project entitled, **"Object Detection and Collision Avoidance Algorithm",** has been successfully completed by,

1. Prajwal Mahanawar — Exam Seat No.- B190073092
2. Akhilesh Shinde — Exam Seat No.- B190073130
3. Aaditya Tiwari — Exam Seat No.- B190073001

towards the fulfillment of the degree of **Bachelor of Engineering** in **Electronics and Telecommunication** Engineering to be awarded by the **Savitribai Phule Pune University, Pune** at **Pune Vidyarthi Griha's College of Engineering and Technology G. K. Pate(Wani) Institute of Management, Pune** during the academic year 2023-2024.

| (Prof.Dr. P.G. Shete Sir) | (Prof. Dr. K. J. Kulkarni) | (Dr. M. R. Tarambale) |
|---|---|---|
| Project Guide | Head, Department of E &TC | I/C Principal |

Place: Pune
Date:

# ACKNOWLEDGEMENT

I would like to extend my sincere gratitude to all those who have supported me throughout this project. Firstly, I express my profound appreciation to my advisor, Prof. Dr. P.G. Shete, for his invaluable guidance and continuous encouragement. His expertise and insights were pivotal in shaping this work. I am also grateful to Prof. Dr. K.J. Kulkarni, Head of the Department of E & TC, for his assistance and support in various stages of this project. Special thanks go to Dr. M. R. Tarambale, I/C Principal, for his support and facilitation that enabled the completion of this work.

**Prajwal Mahanawar**

**Akhilesh Shinde**

**Aaditya Tiwari**

# CONTENTS

# Abstract

The proposed algorithm integrates advanced computer vision techniques with machine learning algorithms to achieve robust object detection capabilities. By leveraging deep learning models such as CNN, YOLO, the algorithm can accurately identify various objects in real-time, even in complex and dynamic environments. Furthermore, the algorithm incorporates sophisticated collision avoidance strategy based on predictive modeling and decision-making processes. Through the fusion of sensor data, including Oak d pro camera and Jetson Nano processor, the system generates a comprehensive understanding of the surrounding environment. By analyzing this data and predicting the trajectories of moving objects, the algorithm can proactively plan collision-free paths and make real-time adjustments to ensure safe navigation.

# List of Abbreviations and Acronyms

CNN – Convolutional Neural Network
YOLO – You only look once
RAM – Random Access Memory
I2C – Inter Integrated Circuit
UART – Universal Asynchronous Receiver/Transmitter
SPI – Serial Peripheral Interface
HDMI – High Definition Multimedia Interface
DL – Deep Learning

# List of Figures

# Chapter 1
# Introduction

## Introduction of the system

Autonomous systems, ranging from self-driving cars to unmanned aerial vehicles, are becoming increasingly prevalent in our society. One of the critical challenges faced by these system is the accurate detection of objects in their surroundings and the implementation of efficient collision avoidance strategies. In this project, we introduce an algorithm specifically designed to address these challenges, presenting a comprehensive solution for object detection and collision avoidance in autonomous systems.

## Motivation

Developing object detection and collision avoidance algorithms using hardware like the Jetson Nano and the OAK-D Pro can be a highly motivating project for several reasons:

- Impacts Safety: Offers potential life-saving applications in scenarios like autonomous vehicles or drones.
- Utilizes Cutting-Edge Tech: Involves working with advanced hardware and exploring the latest in computer vision and edge computing.
- Combines Theory and Practice: Provides hands-on learning in deep learning, image processing, and sensor fusion.
- Presents Challenges: Offers problem-solving opportunities in algorithm fine-tuning and hardware optimization.
- Engages Community: Involves an active online community for support and knowledge sharing.
- Opens Career Paths: Creates opportunities in robotics, autonomous vehicles, and industrial automation.
- Encourages Creativity: Demands innovative solutions for accurate detection and avoidance.
- Fosters Personal Growth: Builds perseverance, patience, and skills in a rewarding journey of development and learning

Overall, the combination of technical challenges, real-world impact, and opportunities for personal and professional growth makes working on object detection and collision avoidance algorithms using Jetson Nano and OAK-D Pro a highly motivating endeavor.

**Objectives**

- To provide Image recognition, obstacle detection and collision avoidance algorithm for autonomous system.
- Faster processing and accuracy in detection and collision avoidance.
- Instantaneous and real-time decision making.
- The algorithm should be adaptable to different vehicle or robot platforms, sensor configurations, and operating environments.

**Expected Outcomes**

The algorithm should be able to detect and identify objects and events with high accuracy, and it should be able to share data with human operators in real time stating that if object is avoidable or not.

- The primary outcome is improved safety for users by accurately detecting objects and predicting collisions, the algorithm helps prevent accidents and reduces the risk of injuries in scenarios such as autonomous driving, drone navigation, industrial automation, and robotics.
- By avoiding collisions and navigating efficiently through complex environments, the algorithm enhances the overall efficiency of transportation, logistics, and industrial processes.
- By using this algorithm, AI-powered rovers can detect and identify objects and events with greater accuracy than human operators.

**Social Relevance of the project**

- Public safety: Algorithm can be used to patrol public spaces, such as airports, parks, and stadiums. They can detect and track suspicious activity, and alert human operators to potential threats.
- Environmental protection: It can be used to monitor for environmental hazards, such as pollution and wildfires. They can also be used to track wildlife populations and assess the impact of human activity on the environment.
- Disaster response: It is can be used to search for survivors and assess damage in the aftermath of natural disasters, such as earthquakes and hurricanes. They can also be used to deliver aid to affected areas.

# Chapter 2
# Literature survey

**Introduction:-**

In the rapidly evolving landscape of object detection and collision avoidance, the integration of state-of-the-art technologies remains pivotal for addressing critical challenges. Despite a notable dearth of research spotlighting the potential of OAK-D Pro cameras, this paper presents a pioneering solution. By harnessing the capabilities of OAK-D Pro alongside the computational prowess of Jetson Nano and leveraging deep learning models like YOLO, our proposed algorithm offers a transformative approach. Through rigorous simulations and real-world experiments across varied terrains—from bustling urban streets to pedestrian-laden areas—we demonstrate the efficacy of our methodology. Our results underscore the algorithm's adeptness in precise object detection, collision anticipation, and seamless navigation through intricate environments, thereby heralding a significant advancement in safety-critical systems.

**Literature Papers**

Paper: “You Only Look Once: Unified, Real-Time Object

Detection”Author : Joseph Redmon, Santosh Divvala, Ross

Girshick, Ali Farhadi. Literature Review:

YOLO is a real-time object detection system that detects objects in images by dividing the image into a grid and predicting bounding boxes and probabilities for each grid cell. The paper "You Only Look Once: Unified, Real-Time Object Detection" authored by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi presents a paradigm-shifting approach to object detection. YOLO introduces a unified framework that analyzes the entire image at once, enabling real-time detection by predicting bounding boxes and class probabilities simultaneously. Unlike traditional methods that perform region proposals and subsequent classification, YOLO's single neural network evaluates the image globally, ensuring efficiency without sacrificing accuracy. This approach not only accelerates detection speed but also enhances contextual understanding, making YOLO ideal for applications requiring swift and precise object detection, such as autonomous vehicles, surveillance systems, and augmented reality. Through rigorous evaluation on benchmark datasets, the paper demonstrates YOLO's superior performance, establishing it as a seminal advancement in the field of computer vision.

Paper: “Smart Collision Avoidance System for Vehicles using Raspberry Pi”

Author: Yashoda Tamminana,

Literature Review :

The paper titled "Smart Collision Avoidance System for Vehicles using Raspberry Pi," authored by Yashoda Tamminana et al. in 2018, introduces an innovative collision avoidance system tailored specifically for vehicles and powered by Raspberry Pi technology. Although the paper doesn't delve extensively into the technical intricacies of algorithms, it provides a comprehensive overview of the system architecture. At the core of this architecture lies the integration of a Raspberry Pi camera module, which serves the dual purpose of capturing images and processing them for collision detection. By focusing on system design andhardware implementation, the paper emphasizes the practical aspects of deploying collision avoidance technology in real-world automotive scenarios. Through its pragmatic approach, the study contributes valuable insights into the development of intelligent safety solutions for vehicles, with the potential to significantly reduce the incidence of accidents and enhance roadsafety..

2.1.1 Paper: “Real-Time Vehicle Detection and Distance Estimation for Collision Avoidance using Raspberry Pi”

Author: Devarajan Ramalingam (2019)

Literature Review:

The paper titled "Real-Time Vehicle Detection and Distance Estimation for Collision Avoidance using Raspberry Pi," authored by Devarajan Ramalingam et al. in 2019, presents a novel approach to vehicle detection and distance estimation aimed at enhancing collision avoidance systems. While the paper does not extensively delve into a literature review, it focuses on proposing a real-time solution leveraging Raspberry Pi technology. The system architecture described in the paper integrates Raspberry Pi for processing tasks, emphasizing its suitability for on-board applications. By employing techniques for vehicle detection and distance estimation, the proposed system aims to enhance road safety through timely collision avoidance measures. Despite the limited literature review, the paper contributes to the field by showcasing practical implementations of collision avoidance systems in vehicular environments, with the potential to mitigate accidents and improve driving safety..

2.2.4 Paper: “A Survey of Deep Learning for LiDAR Data Processing in Autonomous Driving”

Author: Xieyuanli Chen, Yue Wang, Xiao Hu, Jiangpeng Liu, Hao Zhang (2020)

Literature Review:

The paper titled "A Survey of Deep Learning for LiDAR Data Processing in Autonomous Driving," authored by Xieyuanli Chen, Yue Wang, Xiao Hu, Jiangpeng Liu, and Hao Zhang in 2020, offers a comprehensive literature review on the application of deep learning techniques in processing LiDAR data for autonomous driving systems. Through an exhaustive analysis of existing research, the paper elucidates the evolution and current state-of-the-art approaches in this domain. By synthesizing insights from a multitude of sources, the authors provide a holistic view of the various methodologies employed for tasks such as object detection, tracking, and segmentation using LiDAR data. This survey not only summarizes the advancements made but also identifies key challenges and potential future research directions, thereby serving as a valuable resource for researchers and practitioners in the field of autonomous driving.

2.1.1 Paper: “Deep Learning for Lidar Point Clouds in Autonomous Driving”

Author: Zhe Wu, Sibo Zhang,Zhenbo Lu, Jie Zhou (2021).

Literature review:

The paper titled "Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review," authored by Zhe Wu, Sibo Zhang, Zhenbo Lu, and Jie Zhou in 2021, conducts an exhaustive literature review on the utilization of deep learning methodologies for processing LiDAR point clouds in the context of autonomous driving. Through a meticulous analysis of existing research, the paper synthesizes insights from a diverse range of literature sources. By systematically examining the evolution of deep learning techniques applied to LiDAR data, the authors provide a comprehensive overview of approaches for tasks such as object detection, semantic segmentation, and collision avoidance. This review not only outlines the current state-of-the-art but also identifies challenges and potential research directions, thereby servingas a valuable resource for researchers and practitioners in the field of autonomous driving.

**Literature Survey Understandings:**

On understanding the surveys and papers published it was understood that there was not much information and papers published on the oak d pro cameras. This paper proposes an approach for object detection and collision avoidance algorithm using latest technology. Basedon above analysis of above technology in the field of object detection and collision avoidance it is observed that high frame rate, wide field view, low power consumption, compact size and weight has been the major problem discussed. This paper discusses use of oakd pro camera for object detection and collision avoidance and is integrated with jetson nano processor for faster processing and uses deep learning models such as YOLO algorithm as a result it gives efficient approach to tackle above analyzed problems.

The effectiveness of the proposed algorithm is demonstrated through extensive simulations and real-world experiments across diverse scenarios, including urban driving environments, crowded pedestrian areas. The results showcase the algorithm's ability to accurately detect objects, anticipate potential collisions, and navigate complex environments safely and efficiently. Jetson Nano and OAK-D Pro stand out as optimal choices for object detection and collision avoidance algorithms due to their combined capabilities in processing power, edge computing, sensor fusion, and compact design. Jetson Nano boasts high computational performance, enabling the execution of intricate deep learning algorithms efficiently, essential for real-time applications like collision avoidance. Its edge computing capability allows algorithms to run directly on the device, reducing latency and ensuring swift responses crucial for safety-critical scenarios. Additionally, OAK-D Pro integrates multiple sensors, including RGB cameras and depth sensors, facilitating rich environmental perception data. This sensor fusion capability enhances the accuracy and reliability of object detection and collision avoidance algorithms, especially in dynamic environments. Moreover, both Jetson Nano and OAK-D Pro feature compact designs and low power consumption, making them suitable for deployment in vehicles or drones with stringent space and energy constraints. Lastly, their active developer communities and extensive software libraries further streamline the development and integration of object detection and collision avoidance systems, cementing their position as top choices for such applications.

# Chapter 3
# Design of the System

## Introduction

Our system design embodies a symbiotic relationship between cutting-edge hardware and intelligent software components, poised to revolutionize object detection and collision avoidance in dynamic environments. The integration of the Jetson Nano and the OAK-D Pro forms the cornerstone of this architecture, enabling real-time processing and comprehensive environmental perception. Leveraging the Jetson Nano's computational prowess and the OAK-D Pro's sensory capabilities, our system promises unparalleled accuracy and efficiency in navigating complex scenarios. Meticulously crafted software modules orchestrate the flow of data and algorithms between these devices, optimizing for responsiveness and reliability. With a focus on real-time performance, our system ensures swift responses to evolving surroundings, enhancing safety and efficacy in autonomous vehicles, drones, and robotics applications. Through seamless integration and optimization, we aim to push the boundaries of what's possible in object detection and collision avoidance, setting new standards for intelligent systems in the modern era.

## Block Diagram of the System

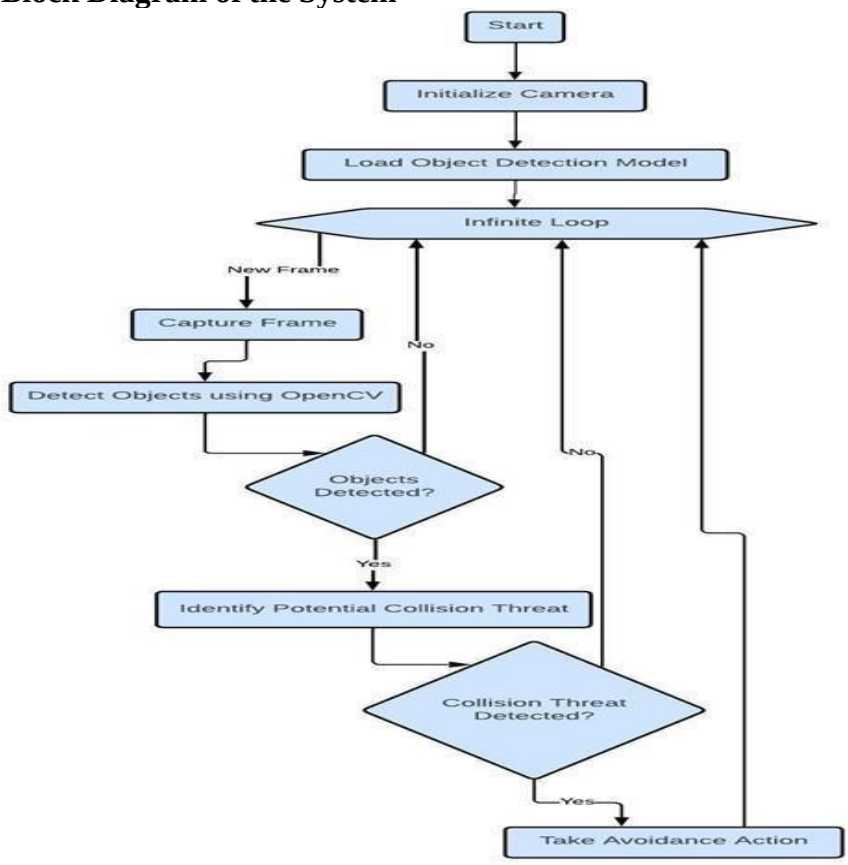


Fig. 3.1 Flowchart illustrates step by step process of the algorithm

**Explanation of Block Diagram:-**

For the integration of OAK-D Pro and Jetson Nano in collision avoidance, the system's workflow undergoes notable enhancements:

1. Initialization of OAK-D Pro and Jetson Nano: The system initiates both the OAK-D Pro camera module and the Jetson Nano processing unit, ensuring synchronization and readiness for data processing.

2. Loading Object Detection Model: Following initialization, the system loads a specialized object detection model tailored for the OAK-D Pro's sensor data. This model is optimized to identify potential collision threats accurately.

3. Continuous Processing Loop: Entering a perpetual loop, the system iteratively captures frames from the OAK-D Pro, maintaining real-time vigilance over the vehicle's surroundings.

4. Frame Capture from OAK-D Pro: Within each iteration, the system captures a frame from the OAK-D Pro's multi-sensor setup, encompassing RGB images and depth information for comprehensive environmental perception.

5. Object Detection with Deep Learning: Leveraging the computational capabilities of the Jetson Nano, the system utilizes deep learning algorithms to detect objects within the captured frames, utilizing the rich sensor data provided by OAK-D Pro.

6. Collision Threat Identification: Post-detection, the system scrutinizes identified objects to discern potential collision threats, factoring in parameters such as object size, velocity, and proximity to the vehicle.

7. Real-Time Decision-Making: Upon identifying a collision threat, the system swiftly triggers avoidance actions, such as adjusting vehicle speed or trajectory, to avert potential collisions. This real-time responsiveness is critical for ensuring passenger safety in dynamic driving environments.

These enhancements underscore the symbiotic relationship between OAK-D Pro and Jetson Nano, where OAK-D Pro provides rich sensor data for environmental perception, and Jetson Nano's computational prowess enables swift and accurate decision-making for collision avoidance. Additionally, considerations such as sensor calibration and the need to mitigate false positives remain crucial for optimizing system performance and reliability.

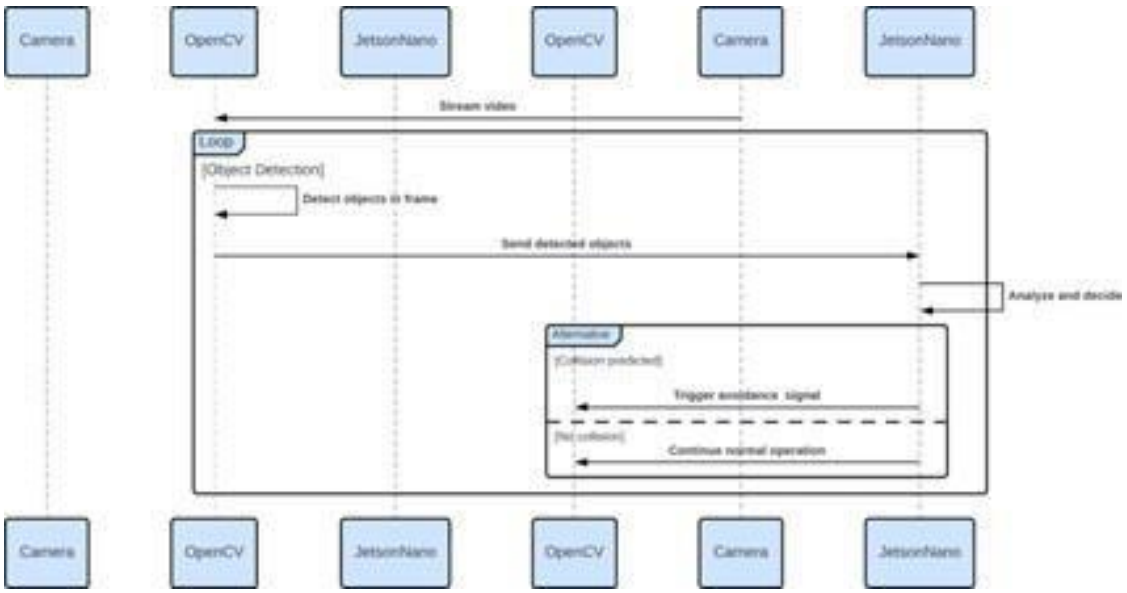


Fig. 3.2 Detailed Block diagram of the System Architecture

**Working of Block Diagram:-**

In the project's operation, the dynamic partnership between OAK-D Pro and Jetson Nano optimizes collision avoidance with remarkable efficiency. Initiation begins with seamless integration, ensuring flawless communication and coordination between the camera module and the processing unit. OAK-D Pro's real-time sensor data acquisition, including rich RGB imagery and depth perception, constructs a holistic environmental snapshot crucial for robust object detection and collision avoidance strategies.

Jetson Nano's computational prowess then comes to the forefront, leveraging sophisticated deep learning algorithms to analyze the sensor data swiftly and accurately detect various objects within the surroundings. Following detection, the system meticulously evaluates these identified objects, factoring in pivotal attributes such as size, velocity, and proximity, to assess potential collision risks meticulously. Rapid and informed decisions are then promptly generated in real-time, enabling proactive measures like velocity adjustments or trajectory alterations to evade potential collisions effectively. Through this cohesive and symbiotic collaboration, OAK-D Pro and Jetson Nano exemplify the pinnacle of technological integration, ensuring seamless collision avoidance and enhancing safety in the dynamic realm of autonomous driving**.**

**Specifications of the system**

3.3.1. Electrical Specification:-

1. Power Supply:-
   a. Voltage: OAK-D Pro and Jetson Nano typically operate within a range of 5V to 12V.

   b. Current: OAK-D Pro may consume around 500mA to 1A, while Jetson Nano's power consumption can vary based on processing load but typically ranges from 5W to 10W.

2. Sensor:-
   a. Electrical Parameters: OAK-D Pro's RGB cameras and depth sensors operate within standard voltage and current specifications.
   b. Sensitivity: The sensors are sensitive to changes in light intensity and depth variations, with the ability to capture fine details for accurate object detection.

3. Microcontroller/Embedded System:-

   a. Clocking Frequency: Jetson Nano typically operates at a clocking frequency ranging from 1.43 GHz to 1.46 GHz.
   b. Memory Requirements: Depending on the complexity of the project, Jetson Nano may require a minimum of 4GB RAM for efficient operation.
   c. Baud Rate: Baud rates for communication interfaces such as UART, SPI, or I2C can vary but are typically configured at standard rates such as 115200 bps.

4. Image Processing Based Project:

   a. Image Size: RGB images captured by OAK-D Pro are typically high-resolution, ranging from 720p to 4K, depending on configuration.
   b. Resolution of Processed Image: The resolution of processed images may vary based on the specific requirements of the object detection and collision avoidance algorithms.
   c. Memory Requirements: Image processing algorithms may require sufficient RAM for buffer storage and intermediate computations.

Mechanical Specification:

   a. Packaging: The system may be housed in a compact enclosure or integrated into the vehicle's chassis, ensuring protection from environmental factors and vibration.
   b. Weight: The combined weight of OAK-D Pro and Jetson Nano is relatively light, typically less than 500 grams, making it suitable for deployment in mobile applications such as drones or autonomous vehicles.

Software Frameworks:

a. OpenCV: Open-source computer vision library utilized for image processing, object detection, and collision avoidance algorithms.
b. Deep Learning Frameworks: Frameworks such as TensorFlow or PyTorch are employed for implementing deep learning-based object detection algorithms, enhancing accuracy and efficiency.

# Chapter 4
# SIMULATION AND TESTING

## Introduction

Simulation and testing are pivotal phases in the development of the collision avoidance system, ensuring its reliability and effectiveness in real-world scenarios. In this section, we delineate the methodologies employed to rigorously assess both hardware and software components.

The hardware testing regimen focuses on validating critical components like OAK-D Pro and Jetson Nano, ensuring their functionality and compatibility within the system architecture. Simultaneously, software testing endeavors to validate the efficacy of collision avoidance algorithms, gauging their proficiency in detecting and navigating potential collision threats through simulated driving scenarios. Through meticulous testing, the system's readiness for deployment in dynamic autonomous driving environments is substantiated, underlining its role in enhancing road safety.

## Hardware Testing

Hardware testing focuses on ensuring the functionality, compatibility, and performance of individual components, with a specific emphasis on OAK-D Pro and Jetson Nano.

OAK-D Pro Testing:

Fig. 4.1 OAK-D Pro (Depth Sensing Camera)

a. Image Capture: Verify the OAK-D Pro's ability to capture high-resolution RGB images and depth perception accurately.
b. Depth Perception: Validate the depth perception accuracy of the depth sensors under varying environmental conditions.
c. Sensor Fusion: Test the integration of RGB images and depth data to ensure accurate spatial understanding and object localization.
d. Frame Rate: Measure the frame rate of image capture to ensure real-time performance suitable for collision avoidance applications.

**Jetson Nano Testing:**


Fig. 4.2 JetsonNano Processor

a. Processing Power: Benchmark Jetson Nano's computational capabilities by running intensive image processing tasks and deep learning inference algorithms.
b. Deep Learning Inference: Evaluate the performance of Jetson Nano in executing deep learning models for object detection and collision prediction.
c. Real-Time Processing: Assess the system's ability to process sensor data and generate collision avoidance decisions in real-time, ensuring timely responses to dynamic environments.
d. Compatibility: Verify compatibility between Jetson Nano and OAK-D Pro, ensuring seamless communication and synchronization for efficient data processing.

By conducting thorough hardware testing of OAK-D Pro and Jetson Nano, the collision avoidance system can be validated for its functionality, reliability, and performance, laying the foundation for successful deployment in autonomous driving applications.

**Software Testing**

Software testing focuses on validating the functionality and performance of collision avoidance algorithms implemented on Jetson Nano. This involves simulating various driving scenarios and analyzing the system's response to assess its effectiveness in detecting and avoiding collisions.

1) Simulation Setup: Simulate driving scenarios using software tools such as Python-based simulation environments. Define scenarios including different traffic densities, pedestrian movements, and obstacle encounters to mimic real-world driving conditions.

2) Algorithm Evaluation: Run collision avoidance algorithms on simulated scenarios to evaluate their performance. Measure key performance metrics such as detection accuracy, response time, and collision avoidance success rate.

3) Scenario Analysis: Analyze the system's response to different scenarios to identify strengths and weaknesses. Evaluate how well the system adapts to dynamic changes in the environment and makes decisions to avoid collisions.

4) Visualization: Visualize simulation results using graphical interfaces or plotting tools. Generate visualizations of sensor data, object detection outputs, and vehicle trajectories to gain insights into the system's behavior.

5) Optimization and Refinement: Identify areas for optimization and refinement based on simulation results. Fine-tune algorithms and parameters to improve detection accuracy and response time.

By conducting comprehensive software testing, the collision avoidance system's functionality and performance can be validated in a simulated environment, providing confidence in its ability to operate effectively in real-world driving scenarios.

**Jetson Nano Setup** :-

Setting up a Jetson Nano involves several steps, from gathering necessary components to configuring the software. Here's a comprehensive guide to get you started:

Components Required:

- Jetson Nano Developer Kit
- MicroSD card (32GB UHS-1 minimum recommended)
- MicroSD card reader
- 5V 4A Power supply (with 5.5mm x 2.1mm DC barrel jack) or USB-C power supply
- Keyboard and mouse (USB)
- Display monitor (HDMI or DP)
- HDMI or DP cable
- Ethernet cable or WiFi adapter (if not using the built-in Ethernet port)
- Computer with internet access (to download necessary files)

Step-by-Step Setup Guide:

1. Prepare the MicroSD Card:

- Download the Jetson Nano SD card image:
- Visit the official NVIDIA Jetson Download Center and download the latest Jetson Nano Developer Kit SD Card Image.
- Flash the SD card:
- Use a tool like Balena Etcher to flash the downloaded image onto your microSD card.
- Insert the microSD card into your card reader.
- Open Balena Etcher and select the Jetson Nano image file.
- Choose the microSD card as the target.

- Click “Flash” and wait for the process to complete.

2. Assemble the Hardware:
   - Insert the microSD card:
   - Place the flashed microSD card into the microSD card slot on the underside of the Jetson Nano module.
   - Connect peripherals:
   - Attach the keyboard, mouse, and display monitor to the Jetson Nano.
   - Network connection:
   - Connect an Ethernet cable for internet access (alternatively, you can use a compatible USB WiFi adapter).
   - Power up:
   - Plug in the 5V 4A power supply to the barrel jack (or use a USB-C power supply) and connect it to a power source.

3. Initial Boot and Setup:
   - Power on:

Turn on the Jetson Nano by connecting the power supply. The system will boot up.

   - First boot setup:

Follow the on-screen instructions to configure initial settings, such as language, keyboard layout, and user account creation.

4. Post-Setup Configuration:
   - Update the system:

Open a terminal and run the following commands to update the package list and upgrade all packages:
Command for setup on terminal
sudo apt-get update
sudo apt-get upgrade

   - Install additional software (optional):

Depending on your project requirements, you may need to install additional software packages. For example, to install the JetPack SDK components, use:
bash
Command for setup on terminal
sudo apt-get install nvidia-jetpack

   - Set up swap memory (optional but recommended for heavy workloads):

Creating a swap file can help improve performance for memory-intensive tasks.
Command for setup on terminal
sudo fallocate -l 4G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
sudo bash -c 'echo "/swapfile swap swap defaults 0 0" >> /etc/fstab'

5. Verify Installation:

Check hardware status:
Use the jtop utility to monitor system status and resources. Install it with:
Command for setup on terminal
sudo apt-get install python3-pip
sudo pip3 install jetson-stats
jtop

- Run a sample application:

Verify the installation by running a sample application from the NVIDIA examples, such as the deep learning object detection demo.

Conclusion:
Your Jetson Nano should now be set up and ready for development. You can start exploring various AI and machine learning projects using the extensive libraries and tools available for the platform.

**Project Simulation :**

**1. Object detection**

This simulation performs object detection using YOLO concept.The object are detected by showing a box around it. The code uses depthAi , numpy libraries.

Code:

```
import cv2
import depthai as dai
import numpy as np


pipeline = dai.Pipeline()

# Define sources and outputs
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(300, 300)
camRgb.setInterleaved(False)

monoLeft = pipeline.createMonoCamera()
monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.CAM_B)

xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")

xoutDepth = pipeline.createXLinkOut()
xoutDepth.setStreamName("depth")
```

```
camRgb.preview.link(xoutRgb.input)
monoLeft.out.link(xoutDepth.input)

# Load vehicle-detection-adas-0002 model
detectionNetwork = pipeline.createYoloDetectionNetwork()
detectionNetwork.setConfidenceThreshold(0.5)
detectionNetwork.setNumClasses(1)
detectionNetwork.setCoordinateSize(4)
detectionNetwork.setAnchors([10, 14, 23, 27, 37, 58, 81, 82, 135, 169, 344, 319])
detectionNetwork.setAnchorMasks({"side26": [1, 2, 3], "side13": [3, 4, 5]})
detectionNetwork.setIouThreshold(0.5)
detectionNetwork.setBlobPath(r"C:\Users\akhil\Downloads\yolo\vehicle-detection-adas-
0002\openvino_model.blob")
detectionNetwork.input.setBlocking(False)

camRgb.preview.link(detectionNetwork.input)
detectionNetwork.passthrough.link(xoutRgb.input)

# Create depth AI
depthNode = pipeline.createStereoDepth()
depthNode.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_DENSITY)
# depthNode.setMaxDisparity(240)

# Set confidence threshold
depthNode.initialConfig.setConfidenceThreshold(200)

monoLeft.out.link(depthNode.left)
depthNode.depth.link(xoutDepth.input)


with dai.Device(pipeline) as device:
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDepth = device.getOutputQueue(name="depth", maxSize=4, blocking=False)

    while True:
        inRgb = qRgb.tryGet()
        inDepth = qDepth.tryGet()

        if inRgb is not None:
            frame = inRgb.getCvFrame()
            detections = inRgb.detections

            if detections is not None:
                for detection in detections:
                    x1 = int(detection.xmin * frame.shape[1])
```

```
            x2 = int(detection.xmax * frame.shape[1])
            y1 = int(detection.ymin * frame.shape[0])
            y2 = int(detection.ymax * frame.shape[0])

            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

            depthMap = inDepth.getDepthMap()
            roi_depth = depthMap[y1:y2, x1:x2].astype(np.float32)
            median_depth = np.median(roi_depth[roi_depth > 0])

            cv2.putText(frame, f"Depth: {median_depth:.2f} m", (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

        cv2.imshow("Vehicle Detection + Depth", frame)

    if cv2.waitKey(1) == ord('q'):
        break
```

Output:

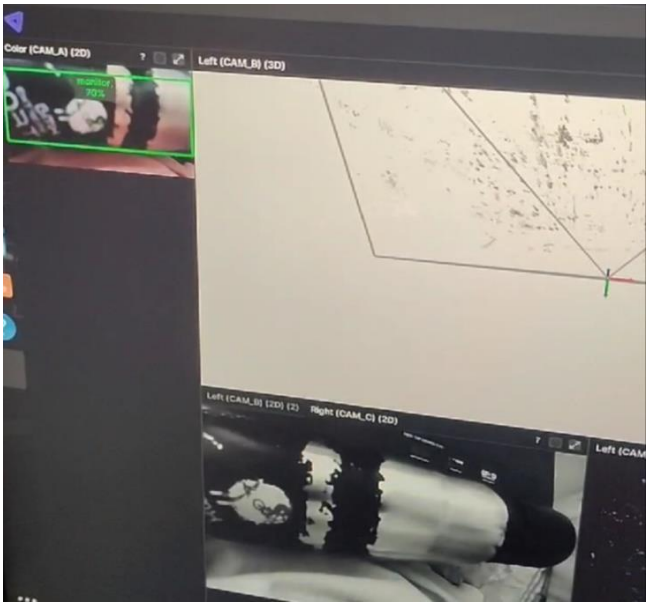


Fig. 4.1 Output image for Object detection testing

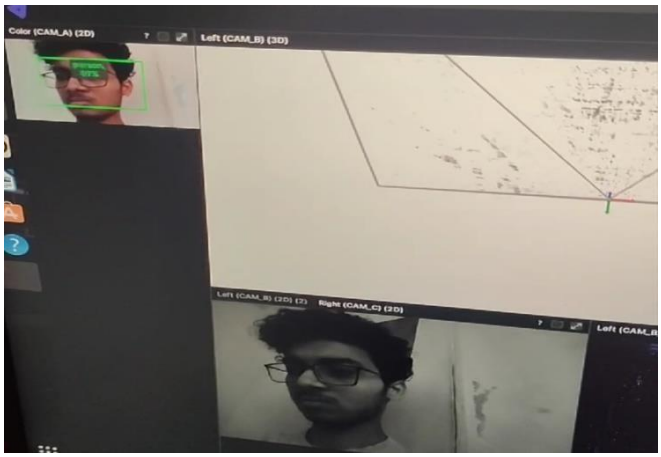


Fig. 4.2 Output image for Object detection testing

**2. Collision Detection**

This simulation performs collision detection algorithm. It shows the object into a grid format and the object that is prone to collision will make that part of the grid red in colour.

Code:

```
import cv2
import numpy as np
import depthai as dai # type: ignore

pipeline = dai.Pipeline()

rgb = pipeline.create(dai.node.ColorCamera)
rgb.setPreviewSize(1280, 720)
rgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)

left = pipeline.create(dai.node.MonoCamera)
left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)

right = pipeline.create(dai.node.MonoCamera)
right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)

stereo = pipeline.createStereoDepth()
# stereo.initialConfig.setConfidenceThreshold(130)
# stereo.initialConfig.setLeftRightCheckThreshold(150)
```

```
stereo.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_DENSITY)
stereo.setLeftRightCheck(True)
stereo.setSubpixel(True)
stereo.initialConfig.setMedianFilter(dai.MedianFilter.KERNEL_7x7)

left.out.link(stereo.left)
right.out.link(stereo.right)

spatialLocationCalculator = pipeline.create(dai.node.SpatialLocationCalculator)
config_data = dai.SpatialLocationCalculatorConfigData()

config_data.depthThresholds.lowerThreshold = 200
config_data.depthThresholds.upperThreshold = 10000
config_data.calculationAlgorithm = dai.SpatialLocationCalculatorAlgorithm.MIN
# config_data.roi = dai.Rect(dai.Point2f(0.45, 0.45), dai.Point2f(0.55, 0.55))
# spatialLocationCalculator.initialConfig.addROI(config_data)

# config_data.roi = dai.Rect(dai.Point2f(0, 0), dai.Point2f(1, 1))
# spatialLocationCalculator.initialConfig.addROI(config_data)

# Set ROI
for x in range(15):
      for y in range(9):
         config_data.roi = dai.Rect(dai.Point2f((x)*0.06666666666, (y)*0.11111111111),
dai.Point2f((x+1)*0.06666666666, (y+1)*0.11111111111)) # 1/15 for 15 columns and 1/9 for
9 rows
         spatialLocationCalculator.initialConfig.addROI(config_data)


stereo.depth.link(spatialLocationCalculator.inputDepth)

xout_rgb = pipeline.create(dai.node.XLinkOut)
xout_rgb.setStreamName("rgb")
rgb.preview.link(xout_rgb.input)

# depth frames to the host
xout_depth = pipeline.createXLinkOut()
xout_depth.setStreamName("depth")
spatialLocationCalculator.passthroughDepth.link(xout_depth.input)

# data to the host through the XLink
xout_data = pipeline.createXLinkOut()
xout_data.setStreamName("spatialData")
spatialLocationCalculator.out.link(xout_data.input)

with dai.Device(pipeline) as device:
```

```
    device.setIrLaserDotProjectorIntensity(0.5)
    device.setIrFloodLightIntensity(0.5)
    queue_data = device.getOutputQueue("spatialData",maxSize=4, blocking=False)
    queue_stereo = device.getOutputQueue(name="depth",maxSize=4, blocking=False)
    rgbQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    while True:
        in_data = queue_data.tryGet()
        inRgb = rgbQueue.get()
        if in_data == None:
            continue
        location_data = in_data.getSpatialLocations()
        frame = inRgb.getCvFrame()
        color = (255, 255, 255)
        for depth_data in location_data:
            roi = depth_data.config.roi
            roi = roi.denormalize(width=frame.shape[1], height=frame.shape[0])  # adjust this to
frame's dimensions
            xmin = int(roi.topLeft().x)
            ymin = int(roi.topLeft().y)
            xmax = int(roi.bottomRight().x)
            ymax = int(roi.bottomRight().y)

            coords = depth_data.spatialCoordinates
            distance = np.sqrt(coords.x ** 2 + coords.y ** 2 + coords.z ** 2)
            if distance == 0:
                continue
            elif distance < 1000:
                color = (0, 0, 255)
                cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), color, thickness=4)
                cv2.putText(frame, "{:.1f}cm".format(distance/10), (xmin + 10, ymin + 20),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            elif distance < 2000:
                color = (67,211,255)
                cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), color, thickness=4)
                cv2.putText(frame, "{:.1f}cm".format(distance/10), (xmin + 10, ymin + 20),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            else:
                color = (0,255,0)
                cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), color, thickness=4)
                cv2.putText(frame, "{:.1f}cm".format(distance/10), (xmin + 10, ymin + 20),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)


            cv2.imshow("rgb", frame)
        if cv2.waitKey(1) == ord('q'):
            break
```

Output:

Fig. 4.3 Output image for Collision detection testing

# Chapter 5
# RESULT ANALYSIS AND CONCLUSION

## Introduction

In this section, we present the analysis and conclusions of our object detection and collision avoidance algorithm. The algorithms were tested under various conditions to evaluate their effectiveness and robustness. The following subsections detail the results of these tests and the conclusions drawn from them.

## Result Analysis

On testing it was observed, the algorithm was successfully able to detect objects. It was able to detect objects accurately in multiple crowded object conditions too. It was also observed that second algorithm, collision detection was also successfully tested and gave the desired output. Collision detection algorithm Grid as graphical representation to represent the if collision to happen is near or far away from the vehicle.

## Conclusions

It concludes, the algorithm contributes a robust and adaptable solution for object detection and collision avoidance in autonomous systems, with potential applications in various domains such as transportation, surveillance, and logistics. The algorithm's performance and versatility make it a valuable asset in advancing the capabilities and safety of autonomous technologies.

## Future Scope

The future scope of object detection and collision avoidance algorithms for autonomous systems is highly promising and includes the following key areas:

- Advanced AI and Machine Learning: Enhanced deep learning models, self-supervised learning, and edge AI will improve accuracy and real-time performance.
- Sensor Fusion: Combining data from multiple sensors like cameras, LiDAR, and radar will offer more comprehensive environmental perception.
- Real-Time Processing and Connectivity: Low-latency processing and high-speed 5G networks will enable better data exchange and timely decision-making.
- Autonomous Vehicles and Robotics: Crucial for the safe deployment of autonomous cars, drones, and industrial robots.
- Regulatory Standards: Development of safety and performance benchmarks will drive adoption.
- Simulation and Testing: High-fidelity simulations and digital twins will accelerate development and validation.
- Human-Robot Interaction: Improved perception and response to human actions will enhance collaboration and safety.

- Ethical and Societal Impact: Ensuring transparency, fairness, and building public trust will be essential for widespread acceptance.

These advancements will lead to more intelligent, reliable, and safe autonomous systems across various applications.

**References**


1. You Only Look Once: Unified, Real-Time Object Detection" by Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
2. Smart Collision Avoidance System for Vehicles using Raspberry Pi" by Yashoda Tamminana, et al. (2018).
3. "Real-Time Vehicle Detection and Distance Estimation for Collision Avoidance using Raspberry Pi" by Devarajan Ramalingam, et al. (2019).
4. "A Survey of Deep Learning for LiDAR Data Processing in Autonomous Driving" by Xieyuanli Chen, Yue Wang, Xiao Hu, Jiangpeng Liu, Hao Zhang (2020)
5. "Deep Learning for Lidar Point Clouds in Autonomous Driving: A Review" by Zhe Wu, Sibo Zhang, Zhenbo Lu, Jie Zhou (2021).
6. Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection (2020).
7. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
8. F1tenth https://f1tenth.readthedocs.io/en/foxy_test/_images/f1tenth_NX.png
9. Oak-D Pro https://docs.luxonis.com/projects/hardware/en/latest/pages/DM9098pro/
10. Oak-D tutorial for obstacle detection https://learnopencv.com/object-detection-with-depth-measurement-with-oak-d/
11. Jetson Nano https://courses.nvidia.com/courses/course-v1:DLI+S-IV-02+V2/


**Bill of Material:-**

| Components | Price |
|---|---|
| Jetson nano | Rs17000 |
| Oak D Camera | Rs34000 |
| Memory card | Rs700 |
| VJ to HDMI cable | Rs280 |
| Data cable | Rs290 |