VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi - 590 018

PROJECT REPORT ON

NeuroHex: Artificial Neural Network (ANN) Controlled Robot for Autonomous Navigation and Environment Perception

Submitted in partial fulfillment for the Award of Degree of
Bachelor of Engineering
in
Electronics and Communication Engineering

Submitted by

ULLAS RAJ S	1RN21EC154
SAHANA T	1RN21EC119
VISHWAS M	1RN21EC172
NANDINI R	1RN22EC418

Under the Guidance of
Mr. PRAVEEN G
Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RNS INSTITUTE OF TECHNOLOGY
Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE (NAAC 'A+ Grade' Accredited, NBA Accredited UG - CSE, ECE, ISE, EIE and EEE) Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560098
2024-2025

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi - 590 018

PROJECT REPORT ON

NeuroHex: Artificial Neural Network (ANN) Controlled Robot for Autonomous Navigation and Environment Perception

Submitted in partial fulfillment for the Award of Degree of
Bachelor of Engineering
in
Electronics and Communication Engineering

Submitted by

ULLAS RAJ S	1RN21EC154
SAHANA T	1RN21EC119
VISHWAS M	1RN21EC172
NANDINI R	1RN22EC418

Under the Guidance of
Mr. PRAVEEN G
Assistant Professor

ESTD 2001

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RNS INSTITUTE OF TECHNOLOGY
Autonomous Institution Affiliated to VTU
Approved by AICTE (NAAC 'A+ Grade' Accredited, NBA Accredited UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560098
2024-2025

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE (NAAC 'A+ Grade' Accredited, NBA Accredited UG - CSE, ECE, ISE, EIE and EEE)

Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560098

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ESTD 2001

CERTIFICATE

Certified that the Final Year Project work entitled **"NeuroHex: Artificial Neural Network (ANN) Controlled Robot for Autonomous Navigation and Environment Perception "** is carried out by **Ullas Raj S (1RN21EC154), Sahana T(1RN21EC119) , Vishwas M (1RN21EC172) and Nandini R (1RN22EC418)** in partial fulfillment for the award of degree of Bachelor of Engineering in **Electronics and Communication Engineering** of **Visvesvaraya Technological University**, Belgavi, during the year 2023-2024. It is certified that all corrections/suggestions indicated during internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in aspect of the project work prescribed for the award of degree of **Bachelor of Engineering**.

....	....	....
Mr. Praveen G	Dr. Rajini V Honnungar	Dr. Ramesh Babu H S
Assistant Professor	Head of the Department	Principal

External Viva

Name of the Examiners	**Signature with date**
1	....
2	....

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK,
Approved by AICTE NAAC ‘A+’ Accredited, NBA Accredited
(UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560098

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ESTD 2001

DECLARATION

We hereby declare that the entire work embodied in this project report titled, **“NeuroHex: Artificial Neural Network (ANN) Controlled Robot for Autonomous Navigation and Environment Perception”** submitted to **Visvesvaraya Technological University**, Belagavi, is carried out at the department of **Electronics and Communication Engineering, RNS Institute of Technology, Bengaluru** under the guidance of **Mr. Praveen G**, Assistant Professor. This report has not been submitted for the award of any Diploma or Degree of this or any other University.

Name	USN	Signature
1. ULLAS RAJ S	1RN21EC154	....
2. SAHANA T	1RN21EC119	....
3. VISHWAS M	1RN21EC172	....
3. NANDINI R	1RN22EC418	....

Acknowledgement

The joy and satisfaction that accompany the successful completion of any task would be incomplete without thanking those who made it possible. We consider ourselves proud to be a part of RNS Institute of Technology, the institution which moulded us in all our endeavors.

We are grateful to our Founder Chairman, RNS Group of Institutions, **Late. Dr. R N Shetty**, for providing state of art facilities. We would like to express our gratitude to our honourable **Mr. Satish R Shetty**, Managing Director, RNS Groups of Institutions, for providing state of art facilities.

We would like to express our sincere thanks to **Dr. Ramesh Babu H S**, Principal and **Dr. Rajini Honnangur**, Professor and HOD, Department of ECE, for their valuable guidance and encouragement throughout our program.

We express our profound gratitude to the coordinators who have given valuable suggestions and guidance throughout the project. We would like to express our sincere gratitude to our guide **Mr. Praveen G**, Assistant Professor, for his guidance , continuous support and motivation in completing the project successfully.

Finally, we take this opportunity to extend our earnest gratitude and respect to our parents, teaching and non-teaching staff of the department, the library staff and all our friends who have directly or indirectly supported us.

ULLAS RAJ S

SAHANA T

VISHWAS M

NANDINI R

Abstract

This study explores the work on autonomous robot capable of identifying objects and navigating independently. The system uses six motorized L joints, allowing biologically inspired locomotion for enhanced stability and adaptability on uneven terrain. The robust computer vision system of NeuroHex, powered by a Raspberry Pi camera module, facilitates obstacle detection and recognition in real time using machine learning (ML) algorithms. This fusion of ANN-based control and visual perception allows the the robot to learn and adapt dynamically to changing environments, optimizing decision-making and movement efficiency. The robot offers both autonomous and user-driven control modes, balancing flexibility and performance based on situational demands. Its innovative design includes a differential shock-absorber system, ensuring stable and efficient mobility even on challenging terrains. The project targets critical applications such as search and rescue missions, environmental monitoring, and exploration in difficult-to-access areas, demonstrating a novel approach to intelligent robotics. Furthermore, and also it provides environment-generated textual feedback, enabling users to fine-tune performance parameters in real time. This project showcases the potential of combining ANN-based decision-making with adaptive locomotion and ML-driven perception to create versatile, autonomous robotic systems.

Table of Contents

Acronyms

ML : Machine Learning

CV : Compuuter Vision

DNN : Deep Neural Network

RPi : Raspberry Pi

USB : Universal Serial Bus

GPIO : General purpose Input Output

SSD : Single Shot MultiBox Detector

SOM : Self Organizing Map

TFLite : TensorFlow Lite

ANN : Artificial Neural Network

CSI : Camera Serial Interface

COCO : Common Object in Context

YOLO : You Look Only Once

CAD : Computed Aided Design

SBC : Single Board Computer

FPS : Frames per Second

Chapter 1

Introduction

The rapid advancements in robotics and artificial intelligence have revolutionized the development of autonomous systems capable of perceiving and interacting with complex environments. NeuroHex, an Artificial Neural Network (ANN)-controlled robot designed for autonomous navigation and environmental perception. Equipped with six L-joints powered by motors and a sophisticated control architecture, and also mimics biological locomotion and neural decision-making processes, offering exceptional mobility across diverse terrains.

The robot integrates motorized leg joints for versatile movement, combined with a robust computer vision system that allows it to detect, recognize, and navigate through various obstacles. The use of an ANN enables the robot to learn and adapt to changing environments in real time, improving its decision-making efficiency and autonomy. This project represents a novel approach to intelligent robotics, targeting applications in search and rescue, environmental monitoring, and exploration of difficult-to-reach areas.

1.1 Motivation

- The primary motivation is to apply theoretical knowledge in practical scenarios.
- Hands-on approach in electronics and mainly in embedded systems solidifies understanding and ignites passion for learning.
- The aim to push technological boundaries, using projects to explore cutting-edge solutions and refine skills.
- The goal is to make meaningful contributions to robotics, inspiring others, and paving the way for future advancements. Through dedication and teamwork, we believe we can achieve remarkable feats.

1.2 Objectives

The primary objective of this project is

1. To Design and build a six-wheeled robotic vehicle with differential shock absorber capable of stable and efficient movement across uneven terrain.

2. To incorporate Raspberry Pi Camera Module to detect and recognize obstacles and objects in real-time with the help of ML.

3. . To provide autonomous as well as user driven controls and navigation for best performance.

4. To perceive the environment and generate relevant text so that the user can fine-tune performance.

1.3 Methodology

Methodology involves various approaches to enable the robot to navigate through its environment efficiently. The methodology of this project are:

1. Develop a detailed design plan outlining the mechanical, electrical, and software components of the robot. Specify the dimensions and materials for the robot chassis, select appropriate sensors and actuators, and outline the architecture of the control system and algorithms to be implemented.

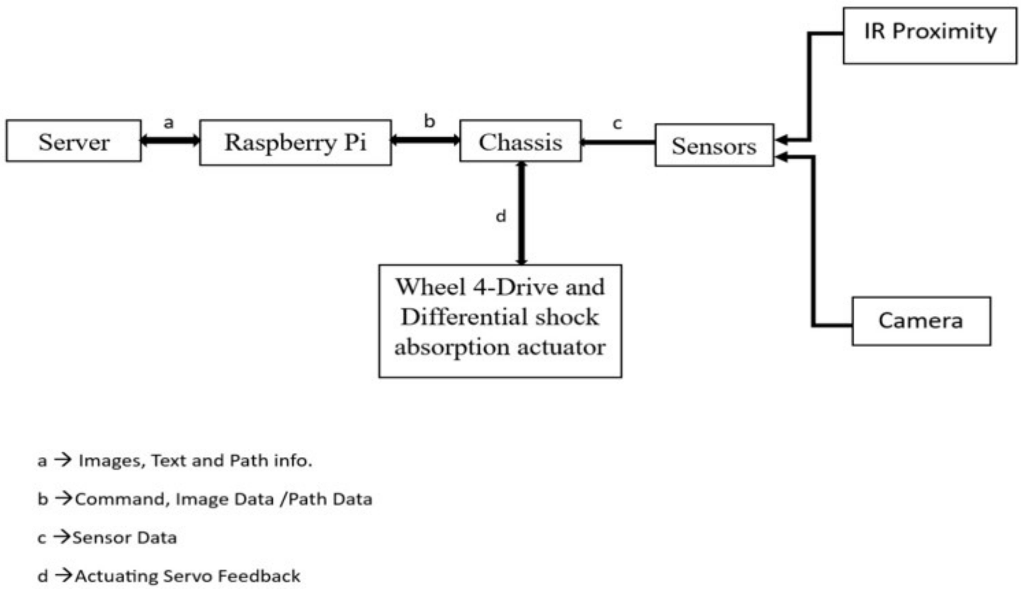
2. Design and Fabrication Proceed with the mechanical design and fabrication of the robot, utilizing computer-aided design (CAD) software to create precise schematics and 3D models of the robot components. Fabricate the chassis, wheels, and other mechanical parts using prototyping tools such as 3D printing or CNC machining.

3. Integrate the electronic components into the robot, which is interfacing the sensors, motor drivers, and power supply units to Raspberry Pi 3 Model B+. Calibrate the onboard sensors, including infrared proximity sensors and Raspberry Pi Camera Module 3, to accurately detect obstacles and robot orientation. Conduct rigorous testing to validate sensor performance under various lighting conditions, maze configurations, and environmental factors.

4. The camera module is set to suitable FPS which is sent to SBC and with the help of pre-existing ML libraries it will do the image processing for Object detection and identification, finally analyzing the type of environment. The type of environment and other data are generated in text and image.

5. The traversed path and the distance is then mapped using miniSOM which takes account of the computed data and sends it through wireless communication to a server.

1.4 Block diagram



IR Proximity
Server
a
Raspberry Pi
b
Chassis
c
Sensors
d
Wheel 4-Drive and Differential shock absorption actuator
Camera
a → Images, Text and Path info.
b →Command, Image Data /Path Data
c →Sensor Data
d →Actuating Servo Feedback

Figure 1.1: Block Diagram

The block diagram in Figure 1.1 illustrates how a robotic system in a server interacts with a Raspberry Pi, which monitors and manages the robot's operations. The Raspberry Pi collects and processes data from various onboard sensors, including infrared (IR) proximity sensors and a camera, to achieve environmental awareness. These sensors detect obstacles and capture visual information, enabling the Raspberry Pi to make informed navigation decisions.

The robot chassis houses the Raspberry Pi and sensors and integrates a four-wheel drive system with a differential shock absorption actuator. This configuration allows the robot to traverse diverse terrains effectively. Communication between the server and the Raspberry Pi, likely over a Wi-Fi connection, facilitates the transmission of high-level commands.

The Raspberry Pi is programmed in Python, employing libraries such as RPi.GPIO for motor control and OpenCV for image processing. The system is designed to adapt dynamically to real-time environmental changes, making it well-suited for autonomous applications. The shock absorption actuator enhances stability, while the sensors provide essential data for safe and efficient navigation.

1.5 Applications

This project has a wide range of applications in various industries. These robots are equipped with sensors and embedded systems to gather information about their surroundings, such as mapping, navigation, and obstacle detection. Obstacle avoidance plays a crucial role in the operation of autonomous robots, enabling them to navigate their environment efficiently and safely.

1.**Industrial Inspection and Maintenance**: The robot can perform inspections of industrial equipment, pipelines, and machinery, identifying faults or maintenance needs in real time. Real-Time Fault Detection using sensors and cameras to detect wear, leaks, or structural issues.
2. **Autonomous Navigation in Dynamic Environments**: This robot can navigate complex and dynamic environments, such as crowded urban areas or shifting terrains, in real time. Adaptive Locomotion Real-time adjustment of gait and movement to adapt to terrain changes.
3. **Mapping and Exploration**: It can generate real-time maps of uncharted or hazardous areas, such as caves, tunnels, or disaster zones, for exploration and situational awareness.
4. **Urban Infrastructure Inspection** : It also can inspect bridges, tunnels, and other critical infrastructure in real time, identifying structural issues and ensuring public safety. Continuous Monitoring Real-time data collection during inspections. .
5. **Agricultural Monitoring and Precision Farming** : It can be used in agricultural fields to monitor crop health, detect pests, and optimize irrigation in real time.Real-Time Crop Analysis Use of cameras and sensors to assess plant health and detect anomalies.

1.6 Advantages and Disadvantages

1.6.1 Advantages

- This robot can independently navigate through complex, unpredictable terrains, such as rubble, uneven ground, and confined spaces, without human intervention.

- The hexapod's six motorized legs, equipped with differential shock absorbers, mimic biological locomotion, allowing for stable and adaptable movement across diverse terrains.

- The ANN allows robot to learn from its environment and improve its performance over time through machine learning algorithms.
- The robot can transmit real-time data, including visual feeds and sensor readings, to a remote server or control center.

1.6.2 Disadvantages

- The ANN-based control system and real-time image processing require significant computational power, which can strain the onboard processor (Raspberry Pi 3 B+).
- The power demands of the motors, sensors, and onboard processing units can result in limited battery life, restricting the robot's operational duration.
- Calibration of sensors (e.g., infrared sensors and camera module) and maintaining the mechanical components (e.g., motors, joints, and shock absorbers) can be complex and time-consuming.
- The robot's performance may be affected by harsh environmental conditions, such as extreme temperatures, dust, water, or low lighting.
- In the event of system malfunctions, such as sensor failures or control errors, the robot may behave unpredictably, posing safety risks.

1.7 Organisation of Report

Chapter 1: Introduction This chapter just provides the introduction to our project and discusses the motivation and the different objectives of our project and just gives a glimpse of the methodology we used and what were the challenges that we faced and the solution to those challenges.

Chapter 2: Literature Survey This chapter gives a summary of the paper we used for reference and explains the methodology behind it and how it contributed to the making of our project and the ideas we took from the different papers.

Chapter 3: Hardware and Software Requirements This chapter gives us insight into the technical details of our project such as the software requirement specification, hardware requirement specifications, high-level design, etc.

Chapter 4: Working Model This chapter elaborates the whole process and the methodology behind the implementation of the project in a step-by-step process and provides insight into programming coding guidelines used during the making of our project.

Chapter 5: Result Analysis In this chapter, we discuss the outputs rather than the results obtained after completion of our project with the efficiency of the model or the accuracy of making correct predictions by our model.

Chapter 6: Conclusion and Future Scope Here we provide the conclusion obtained by our project with the future scope which tries to cover the probable loopholes in our project that may occur in the future as the scenario of the future may be different as compared to now.

Chapter 2

Literature survey

The development of an object detection system using Raspberry pi 3b+ Model integrates multiple fields including embedded systems,and machine learning. This literature survey highlights relevant research and developments that provide a foundation for this project
R. Szabó and A. Gontean,[1] ”Industrial Robotic Automation with Raspberry PI using Image Processing,” explores a cost-effective solution for industrial automation by utilizing Raspberry Pi as the control unit. It integrates a robotic arm, USB cameras, and end-effectors to perform automated tasks with image-based object detection and manipulation.The system employs image processing techniques to enhance robotic vision, enabling real-time identification of objects and guiding precise movements. The Raspberry Pi, as a low-cost yet efficient platform, processes visual data and controls actuators for object handling.This research demonstrates the feasibility of using affordable hardware for industrial tasks like pick-and-place, quality control, and assembly line automation. It highlights the potential of combining robotic vision and automation in production environments, offering scalable and flexible solutions for small to medium-scale industries.

Chai, X., Gao, F., Qi, C., et al.,[2] ”Obstacle avoidance for a hexapod robot in unknown environment,” focuses on developing an advanced obstacle avoidance system for hexapod robots navigating in unstructured and dynamic environments. The study proposes a combination of sensor-based perception and motion planning algorithms to enable real-time obstacle detection and avoidance. It incorporates a hierarchical control strategy where the robot’s gait is adjusted dynamically to navigate around obstacles while maintaining stability and efficiency. Key contributions include the development of algorithms to process environmental data from sensors and the integration of path-planning techniques to ensure smooth and collision-free movement. The research demonstrates the system’s effectiveness through experiments, highlighting its robustness in adapting to unknown terrains.This work advances hexapod robotics by addressing challenges in autonomous navigation, contributing to applications in exploration, rescue missions, and industrial automation.

Dhruv Patel ,et al,[3]”Development and Design of Autonomous Navigation Robot Using Raspberry Pi and Computer Vision,” presents a cost-efficient and effective solution for building an autonomous navigation robot. The system is powered by Raspberry Pi and employs computer vision algorithms for environmental perception and decision-making.The robot utilizes cameras and image processing techniques to identify obstacles, detect paths, and navigate autonomously in real-world scenarios. A combination of motion control algorithms and sensor integration ensures efficient obstacle avoidance and smooth navigation.This research emphasizes the scalability and versatility of using Raspberry Pi as a low-cost platform for autonomous systems, highlighting applications in indoor mapping, industrial automation, and surveillance. The study demonstrates the system’s effectiveness through experiments, showcasing its ability to navigate dynamic and unstructured environments.By integrating computer vision with Raspberry Pi, the paper contributes to advancements in robotics, enabling affordable solutions for complex navigation challenges.

Maideen and Mohanarathinam,[4] ”Computer Vision-Assisted Object Detection and Handling Framework for Robotic Arm Design Using YOLOV5,” introduces a robust framework for robotic arm control using advanced computer vision techniques. The study employs YOLOv5, a state-of-the-art deep learning model for real-time object detection, to identify and localize objects with high accuracy.The proposed system integrates YOLOv5 with a robotic arm, enabling automated object handling tasks such as picking, sorting, and placement. The framework emphasizes efficient object recognition and precise control, ensuring seamless operation in dynamic environments.The authors validate the system through experiments, demonstrating its ability to handle multiple objects in real-time scenarios with minimal errors. The research highlights the applicability of such frameworks in industries like manufacturing, logistics, and warehousing. By combining computer vision and robotics, this study contributes to advancements in intelligent automation, showcasing a scalable and adaptable solution for real-world robotic applications.

The study proposed by De Gouvêa et al.,[5] ”IntraSOM: A Comprehensive Python Library for Self-Organizing Maps with Hexagonal Toroidal Maps Training and Missing Data Handling,” introduces IntraSOM, an innovative Python library designed to streamline the implementation of Self-Organizing Maps (SOMs). The library provides advanced features, including support for hexagonal toroidal map structures, which improve the representation of data and minimize edge effects.A key highlight of the framework is its ability to handle datasets with missing values, making it highly versatile for real-world applications where incomplete data is common. IntraSOM offers customizable parameters and intuitive interfaces for users to train and visualize SOMs effectively. The authors validate the library's performance through experiments, showcasing its computational efficiency and accuracy across diverse datasets. By integrating advanced SOM capabilities with Python's ecosystem, IntraSOM serves as a valuable tool for researchers and practitioners in data science and machine learning.This work significantly contributes to the accessibility and usability of SOMs, promoting their application in clustering, visualization, and data preprocessing tasks.

Greco et al.,[6] "Computer Vision Algorithms on a Raspberry Pi 4 for Automated Depalletizing," explores the implementation of a low-cost and efficient automated depalletizing system using computer vision algorithms. The study utilizes the computational capabilities of a Raspberry Pi 4 to identify, classify, and handle objects in a palletizing setup.The system employs advanced image processing and object detection techniques to locate and pick objects accurately, even in cluttered or variable environments. The lightweight algorithms are optimized for the Raspberry Pi 4, ensuring real-time performance without requiring high-power hardware.This research highlights the potential of combining computer vision with affordable hardware for industrial automation tasks like warehouse management and logistics. Through experiments, the authors demonstrate the system's effectiveness in handling a variety of objects, showcasing its adaptability and scalability. The study contributes to advancing cost-efficient automation technologies, making them accessible for small to medium-scale industries looking to streamline their operations.

Chapter 3

Hardware and Software Requirements

This chapter presents the design and architecture of the Object Detection System for the ANN-controlled robot, deployed on the Raspberry Pi 3 B+ Model. The system integrates hardware and software components to enable real-time, efficient, and accurate path detection, essential for autonomous navigation and environment perception. The hardware includes a processing unit (Raspberry Pi 3 B+), camera for visual input, optional distance sensors for obstacle detection, and actuators for movement control, all powered by a portable energy source (12V battrery) to ensure uninterrupted operation. On the software side, an Artificial Neural Network (ANN) processes path recognition and environmental adaptation, with pretrained DL model Tensorflow lite which provide robust object detection capabilities. Additionally, miniSOM is utilized for real-time mapping and localization to enhance navigation precision. The system is designed to be portable, low-power, and adaptable to various environments, ensuring versatile functionality across diverse applications such as robotics and logistics. By combining advanced hardware and software technologies, this chapter lays the foundation for a scalable solution in autonomous navigation systems.

3.1 Hardware Components

Many components were required to build the hardware model of the robot. The main requirement for the circuit is a computational power which is obtained by Raspberry pi 3B+ Module.

3.1.1 Raspberry pi 3B+ Model

The Raspberry Pi 3B+ is a compact, powerful single-board computer featuring a 64-bit quad-core ARM Cortex-A53 processor running at 1.4GHz and 1GB LPDDR2 SDRAM. It includes dual-band Wi-Fi (2.4GHz and 5GHz) and Bluetooth 4.2/BLE for wireless connectivity.
Its 40-pin GPIO header supports hardware interfacing, while the MIPI DSI and CSI ports enable connection to compatible displays and cameras. The board features four

USB 2.0 ports and a full-size HDMI port for peripherals and video output. Gigabit Ethernet (limited to 300Mbps) provides faster networking. A PoE header supports Power-over-Ethernet when used with a PoE HAT. It also has a 4-pole stereo/composite port for audio and older video outputs. Powered via a 5V/2.5A Micro USB, it's ideal for IoT, robotics, and multimedia projects.

1. BCM2837BO SoC (64-bit Quad-Core @ 1.4GHz): The Raspberry Pi 3B+ features a Broadcom BCM2837BO system-on-chip. This is a 64-bit quad-core ARM Cortex-A53 processor running at 1.4 GHz, providing better performance compared to earlier models.

2. 1GB LPDDR2 SDRAM: The module has 1GB of LPDDR2 memory, which is shared between the CPU and GPU for efficient multitasking and video processing.

3. Wireless LAN (2.4GHz and 5GHz): Supports dual-band Wi-Fi (IEEE 802.11 b/g/n/ac), offering improved network performance. Ideal for IoT and other wireless applications.

4. Bluetooth 4.2/ BLE: Comes with integrated Bluetooth 4.2 and Bluetooth Low Energy (BLE) support for connecting peripherals like keyboards, mice, and IoT devices.

5. 40-pin GPIO Header: General-Purpose Input/Output (GPIO) pins provide the ability to connect and control hardware components like LEDs, sensors, and motors. It includes 28 GPIO pins and supports various communication protocols such as I2C, SPI, and UART.

6. Power-over-Ethernet (PoE) Header: A PoE header is present, allowing the Raspberry Pi 3B+ to be powered through Ethernet when paired with a PoE HAT (Hardware Attached on Top).

7. 4 x USB 2.0 Ports: Four USB 2.0 ports enable connection to peripherals like keyboards, mice, flash drives, and external hard drives.

8. Faster Ethernet (300Mbps over USB 2.0): The device includes Gigabit Ethernet, which is limited to 300 Mbps due to its connection through a USB 2.0 bus.

9. Full-Size HDMI Port: A full-sized HDMI port allows direct connection to monitors and TVs for video output. Supports resolutions up to 1080p.

10. MIPI DSI Display Port: MIPI (Mobile Industry Processor Interface) DSI (Display Serial Interface) is for connecting official Raspberry Pi touch displays or other compatible screens.

11. MIPI CSI Camera Port: The MIPI CSI port allows for the connection of the official Raspberry Pi camera module or other compatible camera devices.

12. 4-pole Stereo Output and Composite Video Port: Provides analog stereo audio output. Also offers a composite video signal for older displays through a 3.5mm jack.

13. 5V/2.5A DC Input via Micro USB Connector: Power is supplied via a Micro USB connector with a 5V, 2.5A power supply to ensure stable operation with peripherals.

3.1.1.1 Pin Functions

The pins typically used for connecting input (sensors) and outputs(actuators) are referred to as GPIO. The GPIO range of pins we typically use as shown in Figure 3.1 is described below:

- Power Pins:
- Pin 1 (3.3V): Provides 3.3V power output.
- Pin 2 (5V): Provides 5V power directly from the power source.
- Pin 4 (5V): Additional 5V power pin.
- Pin 6 (GND): Ground.
- Pin 9 (GND): Additional ground pin.
- Pin 14, 20, 25, 30, 34, 39 (GND): More ground pins.
- GPIO Pins (Configurable):
- Pins 3 and 5: GPIO 2 (SDA1) and GPIO 3 (SCL1) for I2C communication.
- Pins 7, 11, 13, 15, 19, 21, 23, 29, 31, 33, 35, 37: General-purpose GPIOs for custom input/output.
- Pins 8 and 10: GPIO 14 (TXD) and GPIO 15 (RXD) for UART communication.
- SPI Interface:
- Pin 19 (GPIO 10 - MOSI): SPI data out.

- Pin 21 (GPIO 9 - MISO): SPI data in.
- Pin 23 (GPIO 11 - SCLK): SPI clock.
- Pin 24 (GPIO 8 - CE0): Chip enable 0.
- Pin 26 (GPIO 7 - CE1): Chip enable 1.
- PWM (Pulse Width Modulation):
- Pin 12 (GPIO 18) and Pin 32 (GPIO 12): PWM-capable GPIO pins. Often used for motor control or dimming LEDs.
- I2C Communication:
- Pin 3 (GPIO 2 - SDA) and Pin 5 (GPIO 3 - SCL): Primary I2C interface.
- UART (Universal Asynchronous Receiver-Transmitter):
- Pin 8 (GPIO 14 - TXD) and Pin 10 (GPIO 15 - RXD): Used for serial communication.
- Pin 27 (GPIO 0 - ID_SD) and Pin 28 (GPIO 1 - ID_SC): Reserved for ID EEPROM (e.g., HAT auto-detection).[7]

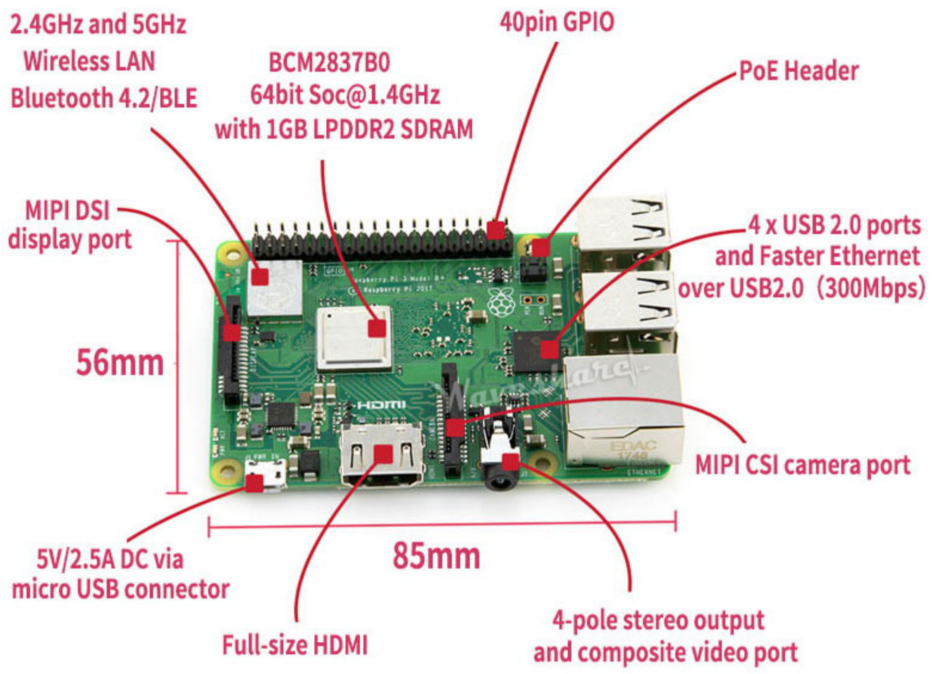


Figure 3.1: Raspberry pi 3B+ Module

3.1.2 Raspberry Pi Camera Module 3

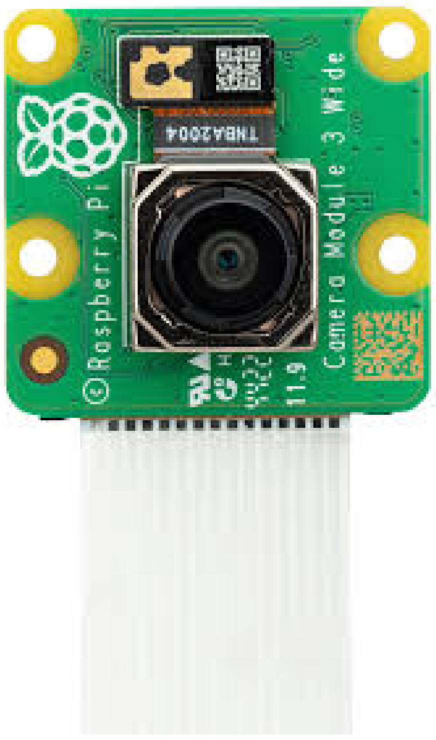


Figure 3.2: Raspberry pi Camera Module 3

The Raspberry Pi Camera Module 3 shown in Fig 3.2 is an advanced imaging device designed for use with Raspberry Pi boards, offering significant improvements over its predecessors. It features a 12MP Sony IMX708 sensor, delivering high-resolution images and video. The inclusion of Phase Detection Autofocus (PDAF) ensures precise and sharp focusing, making it suitable for a wide range of applications. The module is available in two variants: Standard (75° field of view) and Wide (120° field of view), catering to different project requirements. Additionally, NoIR (No InfraRed) versions are available, allowing for infrared photography and night vision applications. It supports High Dynamic Range (HDR), which enhances image quality in scenes with both bright and dark areas. The improved low-light sensitivity makes it ideal for dim environments. For video recording, it supports 1080p at 50fps and 720p at 100fps, offering smooth and high-quality video output.

The Camera Module 3 connects to Raspberry Pi boards via the MIPI CSI-2 interface and is compatible with all models featuring a CSI port. It comes with a flex ribbon cable for simple connection and draws power directly from the Raspberry Pi board. The module integrates seamlessly with the Raspberry Pi's libcamera stack and Python libraries like OpenCV, enabling advanced functionalities such as machine vision and AI-based image processing. Its compact design and lightweight construction make it suitable for embedded projects, including robotics, surveillance, and creative photography. The autofocus feature allows flexibility in capturing subjects at varying distances, and the HDR support enhances video quality in complex lighting conditions. Overall, the Raspberry Pi Camera Module 3 is a versatile and powerful tool for both beginners and advanced users working on imaging and computer vision projects.

3.1.3 2WD Motor driver module - L298N

The L298N is a dual H-Bridge motor driver which allows speed and direction control of four DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A. The pin-out of L298N motor driver module can be viewed in Figure 3.3 and working can be understood in the section below.

The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output. This depends on the voltage used at the motors VCC. The module have an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V, the 5V regulator can be enabled and the 5V pin can be used as output, for example for powering the Arduino board. Usually for lower VCC current produced for the regulated voltage is low and hence cannot operate ESP32. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need connect it to a 5V power supply in order the IC to work properly.

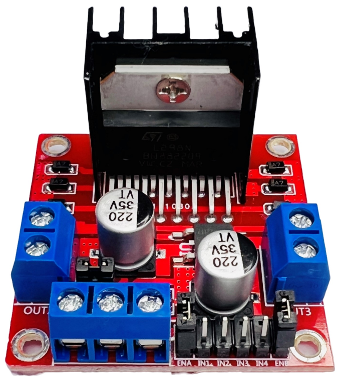


Figure 3.3: L298N Motor Driver Module

3.1.4 10 RPM Geared DC Motor

A 10 RPM geared DC motor is a type of direct current motor that has a reduction gear system, which reduces the speed of the motor's output shaft. In this case, the motor rotates at 10 revolutions per minute (RPM), which means that it completes 10 full rotations in one minute. These types of motor are typically used when low-speed, high-torque operation is required, such as in robotics, conveyor belts, or small automation systems. The motor can be seen in Figure 3.4.

DC Motor Functionality :

- Converts electrical energy into mechanical energy using electromagnetic principles.
- When a voltage is applied, the current flows through the motor windings, creating a magnetic field.

Gearbox Mechanism:

- Attached to the motor shaft, the gearbox reduces the speed of rotation while increasing torque.
- This provides precise and low-speed motion ideal for robotic applications.

Speed Control:

- The speed of the motor can be adjusted by varying the input voltage or using a Pulse Width Modulation (PWM) signal.
- This ensures precise movement and adaptability.

High Torque: Provides sufficient force to lift and move robotic legs.
Low Speed: Ensures stable and precise movements.
Energy Efficiency: Optimized for low power consumption.

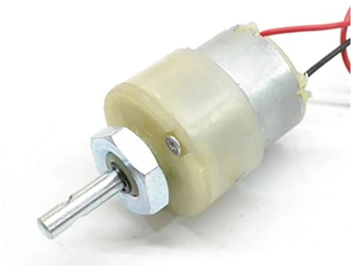


Figure 3.4: 10 RPM Geared DC Moto

3.1.5 HC-SR04(Ultrasonic sensor)

Ultrasonic sensor, which consists of a transmitter and receiver. The sensor measures how far things are without touching them, and it uses sound waves to get the measurements right. It can work well when things are between 2-4 cms away.

As the name indicates, ultrasonic sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception. An optical sensor has a transmitter and receiver, whereas an ultrasonic sensor uses a single ultrasonic element for both emission and reception. In a reflective model ultrasonic sensor, a single oscillator emits and receives ultrasonic waves alternately. This enables miniaturization of the sensor head.

The ultrasonic sensor uses SONAR to determine the distance to an object. Here's what happens as shown by Figure 3.5 the ultrasound transmitter (trig pin) emits a high-frequency sound (40 kHz). The sound travels through the air. If it finds an object, it bounces back to the module. The ultrasound receiver (echo pin) receives the reflected sound (echo).

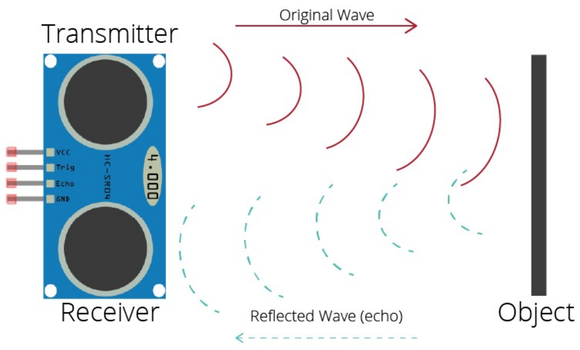


Figure 3.5: HC-SR04 Ultrasonic sensor

The HC-SR04 Ultrasonic Sensor features

- Distance Measurement The HC-SR04 measures the distance between the sensor and an object using ultrasonic waves. It sends out a high-frequency sound wave (40 kHz) and measures the time it takes for the wave to bounce back from the object.

- Trigger and Echo Pins : The HC-SR04 has two main pins - Trigger (Trig) Pin is used to send a signal to the sensor to start measuring the distance. Echo Pin outputs a signal that indicates the time it took for the ultrasonic wave to bounce back from the object.

- Distance Calculation : The distance is calculated using the following formula: distance = (speed of sound x time) / 2 where speed of sound is approximately 343 m/s at room temperature and atmospheric pressure. +
- Output Signal : The HC-SR04 outputs a pulse width modulation (PWM) signal on the Echo Pin, where the pulse width is proportional to the distance measured.
- Range and Accuracy : The HC-SR04 has a range of approximately 2 cm to 400 cm (0.8 in to 157 in) and an accuracy of ±3 mm (±0.12 in) The HC-SR04 Ultrasonic Sensor uses SONAR to determine the distance to an object. This sensor reads from 2cm to 400cm (0.8inch to 157inch) with an accuracy of 0.3cm (0.1inches), which is good for most hobbyist projects. In addition, this particular module comes with ultrasonic transmitter and receiver modules.
- Power Supply :+5V DC
- Quiescent Current : >2mA
- Working Current: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1 – 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS TTL pulse
- Echo Output Signal: TTL pulse proportional to the distance range Dimension: 45mm x 20mm x 15mm

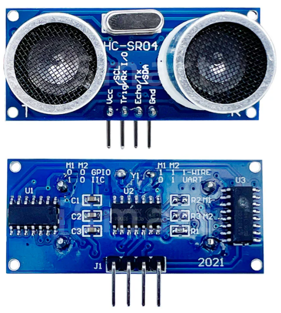


Figure 3.6: HC-SR04(Ultrasonic sensor)

The ultrasound transmitter (trig pin) emits a high-frequency sound (40 kHz). The sound travels through the air; if it finds an object it bounces back to the module. The ultrasound receiver (echo pin) receives the reflected sound (echo)[12]. The two pins trigger pin and echo pin can be seen in Figure 3.6.

3.2 Software Requirements

The autonomous robot with the help of Raspberry Pi 3B+ model uses mainly python programming, as python programming is versatile for coding even for embedded work to the object recognition which is mainly the AIML applications. Due to the fact that many pretrained models are widely available on Github by various comapnies. One such made used in the project is the Tensorflow lite model which is the tensorflow lite runtime. The full version of tensorflow package can't be managed by the Raspberry Pi processor. Hence, the lighter version is TFlite runtime for the robot. Along with object detection, additional packages are made used for obstacle avoidance, these packages too are ML libraries for path detection and mapping. intraSom and miniSOM are few ML libraries made used to train the robot with the ultrasonic sensor data.

3.2.1 Python3

Python3 has powerful libraries such as TensorFlow or PyTorch for neural network development, and other ML libraries for robotic navigation and control. This project would involve designing an ANN to process sensory input (e.g., from ultrasonic and camera sensors) and generate control signals for the robot's actuators. You could use Python libraries like numpy and pandas for data handling, matplotlib for visualization and OpenCV for processing visual input. Sensor data could be fed into the ANN for training, with the goal of learning to recognize obstacles, map the environment, and make navigation decisions autonomously.

Some of the features of python3 implemented to this project are:

- Simulation Initialization: A hexapod model is loaded into a PyBullet simulation.
- Sensor Data Acquisition: Sensor readings are simulated using distance sensors.
- ANN Design: A neural network is defined to map sensor inputs to joint movements.
- Data Generation: Training data is generated by simulating random movements.
- Training and Deployment: The ANN is trained with the generated data and deployed for navigation.

- ANN Architecture: The neural network has an input layer (sensory inputs), a hidden layer, and an output layer (movement commands for hexapod legs).
- Hexapod Movement: Sensory inputs control leg movements via ANN outputs.

3.2.2 TensorFlow Lite (Neural network framework)

TensorFlow Lite (TFLite Runtime) is a lightweight and efficient framework designed to deploy machine learning models on edge devices, making it an ideal choice for this project. By converting complex ANN models trained in TensorFlow into compact, deployable versions, TensorFlow Lite ensures that the robot's onboard hardware can efficiently process environmental data in real-time. This is particularly crucial in resource-constrained embedded systems, where low power consumption and minimal memory usage are paramount. TensorFlow Lite's model quantization further optimizes resource usage without compromising accuracy, enabling the hexapod to perform tasks such as path planning, obstacle avoidance, and terrain adaptation effectively. Additionally, TensorFlow Lite supports hardware acceleration, which can leverage edge-specific processors like Coral TPU or NVIDIA Jetson Nano to enhance performance.

TensorFlow Lite also integrates an ANN to enable real-time decision-making and environmental understanding in resource-constrained environments and interested amateurs over other systems:

1. The robot employs ANN models for path planning, obstacle avoidance, and terrain adaptation, all powered by TensorFlow Lite.
2. TensorFlow Lite's pre-compiled libraries reduce deployment complexity for robot's robotics platform..
3. Model optimization with TensorFlow Lite ensures faster inference, allowing the hexapod to react to environmental changes promptly.
4. TensorFlow Lite provides tools for debugging and profiling to ensure the ANN operates efficiently in real-world conditions.
5. TensorFlow Lite Micro is utilized for running models on low-power microcontrollers used in the hexapod.
6. TensorFlow Lite allows integration of transfer learning techniques, enabling NeuroHex to adapt to diverse environments.

7. ANN models in NeuroHex benefit from TensorFlow Lite's support for TensorFlow operations critical for robotics tasks.

8. The portability of TensorFlow Lite enables cross-platform compatibility, simplifying development and deployment.

9. TensorFlow Lite's flexibility ensures easy integration with the hexapod's real-time control systems.

10. The ANN-powered hexapod achieves efficient, autonomous exploration of dynamic terrains using TensorFlow Lite.

11. TensorFlow Lite's tools for model conversion streamline the pipeline from ANN development to hexapod deployment.

12. The overall system efficiency and autonomy of NeuroHex are significantly enhanced by TensorFlow Lite's tailored optimizations.

3.2.3 cv2 (OpenCV)

OpenCV (Open Source Computer Vision Library) is a comprehensive open-source library designed for real-time computer vision tasks and image processing. It provides a wide range of functionalities, including image transformations, feature detection, object recognition, video analysis, and more. OpenCV supports multiple programming languages and is widely used in fields like robotics, surveillance, and augmented reality. It also includes tools for drawing shapes, text annotations, and converting between color spaces, making it ideal for applications involving image-based input and output.

In this project, cv2 is employed for preprocessing video frames to ensure compatibility with the TensorFlow Lite model. It resizes frames, adjusts color channels, and performs any required normalization. Additionally, cv2 is used to annotate the frames by drawing bounding boxes and labels around detected objects, providing visual feedback on the object detection process.

Some of the Key Features of openCV:

1. Image Processing: Functions for filtering, edge detection, transformations, and histogram analysis.

2. Computer Vision: Face detection, object recognition, feature detection, and tracking.

3. Video Analysis: Motion detection, video stabilization, and background subtraction.

4. Machine Learning: Pre-built models for object classification, clustering, and feature extraction. Support: Works with images and video from files, webcams, and other devices.

3.2.4 picamera2

picamera2 is a Python library designed specifically for interfacing with Raspberry Pi cameras. It provides a modern and user-friendly API for capturing images and videos, offering advanced features such as configuration management, preview capabilities, and various output formats. The library supports flexible control over resolution, frame rates, and other camera parameters, making it an essential tool for projects requiring real-time video input on Raspberry Pi devices.

In this application, Picamera2 initializes and configures the camera for capturing real-time video frames. These frames are then passed through the object detection pipeline. The library's ability to provide consistent and high-quality frame capture ensures seamless integration into the real-time object detection and video streaming workflow.
Some of the Key Features of picamera2:

1. Simplified API: Makes capturing images and videos easier for developers.

2. Integration: Provides better support for the Raspberry Pi's libcamera stack.

3. Customization: Allows fine-tuning of camera settings like resolution, framerate, and exposure.

4. Real-Time Applications: Supports previewing and capturing simultaneously, suitable for robotics or IoT projects.

3.2.5 Flask

Flask is a lightweight Python web framework designed to simplify the development of web applications and APIs. Known for its minimalistic and modular approach, Flask provides essential features like routing, templating, and request handling without enforcing a rigid structure. This flexibility makes it a popular choice for building small to medium-sized web services or embedding web interfaces into larger applications.

In this project, Flask serves as the foundation for the web application that streams real-time video with object detection results. It provides an HTTP endpoint that users can access via a web browser to view the processed video stream. Flask's capabilities enable seamless integration of real-time computer vision output into an easily accessible web-based interface.
Some of the Key Features of Flask:

1. Minimalist Design: Provides the essentials needed for web development without unnecessary bloat.

2. Extensibility: Easily integrates with libraries for databases, authentication, and APIs.

3. Routing: Simple URL routing to manage application endpoints.

4. Templates: Uses Jinja2 templating for rendering dynamic HTML pages.

5. Scalability: Suitable for small projects and can scale up with extensions for larger applications.

3.2.5.1 SSD MobileNet v2

SSD (Single Shot Detector) MobileNet v2 is a variant of the MobileNet architecture, specifically designed for object detection tasks. It combines the strengths of MobileNet v2 seen in Figure 3.7, a lightweight and efficient convolutional neural network (CNN), with the SSD object detection framework.

- Lightweight: SSD MobileNet v2 has fewer parameters and computations compared to other object detection models, making it suitable for resource-constrained devices like smartphones and embedded systems.

- Efficient: The model uses depthwise separable convolutions (DSConv) and inverted residual blocks, which reduce computational complexity and memory usage.

- Accurate: SSD MobileNet v2 achieves competitive accuracy with larger models, such as YOLO and Faster R-CNN, on popular object detection benchmarks like COCO (Common Objects in Context). Where YOLO v5 library is not used in this project .

- Flexible: The model can be trained on various datasets and tasks, including object detection, instance segmentation, and semantic segmentation.

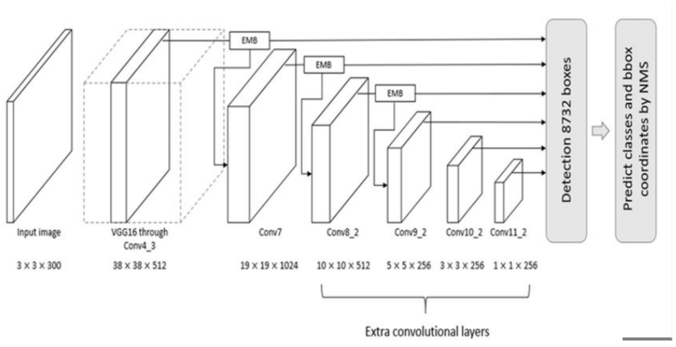


Figure 3.7: SSD MobileNet v2 layers

3.2.5.2 Applications of SSD MobileNet v2

- Mobile devices: SSD MobileNet v2 can be used for object detection in mobile apps, such as image recognition, augmented reality, and autonomous vehicles.
- Embedded systems: The model can be deployed on resource-constrained devices, like smart home appliances, robots, and drones, for object detection and tracking.
- Computer vision: SSD MobileNet v2 can be used for various computer vision tasks, including object detection, segmentation, and tracking, in industries like healthcare, retail, and manufacturing.

3.2.6 miniSOM

MiniSom can play a crucial role in unsupervised learning and pattern recognition. miniSOM is a Python-based implementation of Self-Organizing Maps (SOMs), which are neural networks capable of reducing the dimensionality of high-dimensional sensor data collected by the hexapod. By using MiniSom, NeuroHex can cluster environmental data, such as terrain characteristics or obstacles, into meaningful patterns while preserving the topological relationships within the data. This allows the hexapod to autonomously recognize and adapt to different environmental conditions, optimize its navigation strategy, and make real-time decisions. MiniSom's adaptability and efficiency in processing multi-dimensional data make it an excellent tool for enhancing the hexapod's perception and learning capabilities without requiring explicit supervision or labeled datasets.

miniSOM is a lightweight Python library designed to implement Self-Organizing Maps (SOMs), a type of unsupervised neural network used for clustering and dimensionality reduction. One of its key features is its simplicity and ease of use, making it accessible for both beginners and experienced users. MiniSom supports customizable hyperparameters, such as learning rates and neighborhood functions, which can be fine-tuned to suit specific datasets and tasks. It is highly efficient, particularly with small to medium-sized datasets, and can handle multi-dimensional data while preserving topological relationships. Another notable feature is its flexibility, allowing users to define grid shapes and sizes for the SOM, enabling tailored visualization and clustering. Additionally, MiniSom includes built-in tools for visualizing the results, such as distance maps , which aid in interpreting the patterns and clusters formed during training. These features make miniSOM a powerful and user-friendly tool for exploratory data analysis and pattern recognition.

Chapter 4

Methodology

The construction of the model requires the working principle, robot architecture and algorithms. The way of implementing the pre-trained model TFLite Runtime in the model is explained in the section giving briefly about each step.

4.1 Block diagram

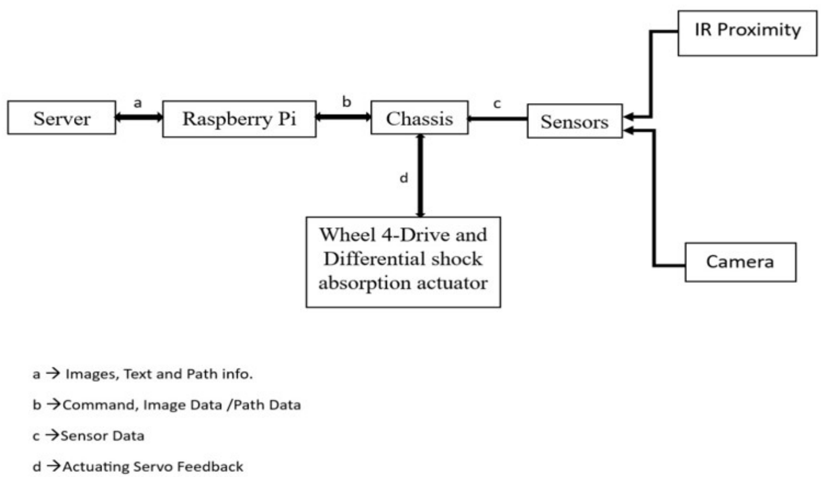

IR Proximity
Server
a
Raspberry Pi
b
Chassis
c
Sensors
d
Wheel 4-Drive and Differential shock absorption actuator
Camera
a → Images, Text and Path info.
b → Command, Image Data /Path Data
c → Sensor Data
d → Actuating Servo Feedback

Figure 4.1: Block Diagram

The block diagram in Figure 4.1 represents a robotic system where a server communicates with a Raspberry Pi, which observes the robot's operations. The Raspberry Pi processes data from various sensors, including IR proximity sensors and a camera, which provide environmental awareness. The sensors detect obstacles and capture visual data, which the Raspberry Pi uses to make navigation decisions.

The chassis houses the Raspberry Pi and sensors and connects to a wheel 4-drive and differential shock absorption actuator, enabling the robot to move across different terrains. The server communicates with the Raspberry Pi, possibly via Wi-Fi, to send higher-level commands. The implementation involves using Python to program the Raspberry Pi, utilizing libraries like RPi.GPIO for motor control and OpenCV for image processing. The system can adapt to real-time changes in its environment, making it suitable for autonomous applications. The shock absorption actuator ensures stability, and the sensors provide crucial data for safe navigation.

4.2 Flowchart

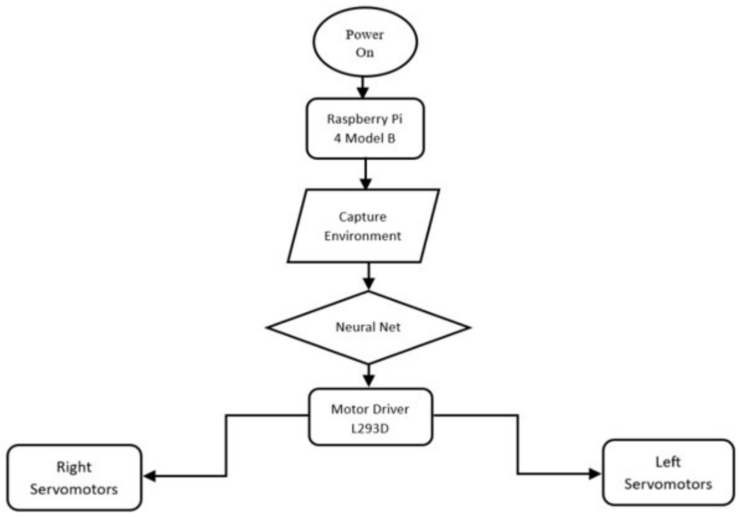


Figure 4.2: Flow chart

The flow diagram referring to Figure 4.2 represents the operational process of a robotic or automated system. The system starts when power is supplied, activating all components. The Raspberry Pi 3 Model B+ serves as the central controller, initializing and managing the system's operations. It collects environmental data through sensors, such as cameras or other input devices, to understand the surroundings. This data is then processed by a neural network, which analyzes the information and makes decisions based on pre-trained models. The decisions are passed to the motor driver which acts as an interface to control the servomotors. Finally, the motor driver directs the right and left servomotors to perform specific actions, such as movement or steering, enabling the system to respond dynamically to its environment. This process is commonly used in applications like robotics, autonomous vehicles, and smart systems for real-time decision-making and action.

4.3 Chassis CAD Modeling

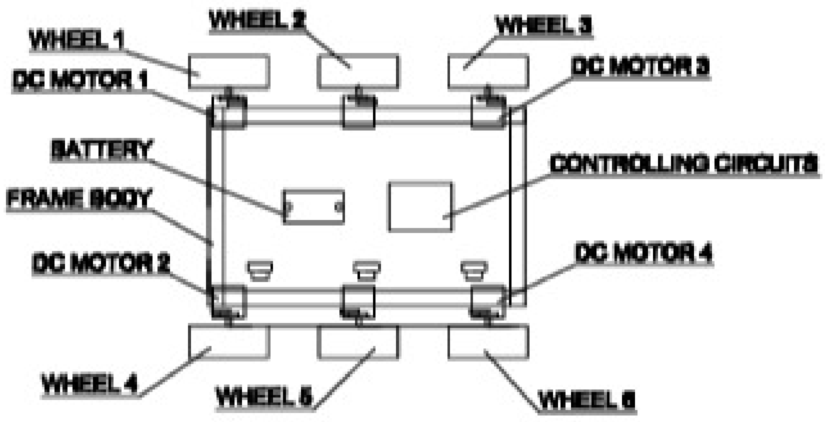


Figure 4.3: CAD Modeling of Robot's Chassis

The modeling of the robot's chassis as defined by the project was done using SolidEdge Mechanical CAD software can be seen in Figure 4.3. The robot chassis is a 6-wheeled robot , at four corner wheels we have DC geared motors which run at 10 RPM. The use of DC geared motor is to stop the robot from moving down an inclined surface while maneuvering on the real-world. The robot chassis then houses all the other hardware components the distance sensors which are the ultrasonic sensors, the visual sensor (Raspberry pi camera module) and the 12V 1.3 Amp LiPo Battery and mainly the Raspberry Pi 3B+ SBC.

4.4 Object detection using TFLite Runtime framework

The object detection methodology leverages a pre-trained TensorFlow Lite (TFLite) model optimized for real-time inference on embedded systems. The Raspberry Pi 3 B+ is used as the primary processing unit to execute object detection tasks, supported by a camera module for capturing live video frames.

The detection pipeline begins with frame acquisition, where images are captured at a resolution of 640x480 pixels using the Picamera2 library. These frames are preprocessed to meet the input requirements of the TFLite model. Preprocessing involves resizing the frames to the model's expected dimensions, converting the color format to RGB, and normalizing pixel intensity values for efficient feature extraction. The processed frames are then fed into the TFLite model, which generates three key outputs:

1. Class Predictions: Indicating the detected object category.
2. Bounding Boxes: Representing the coordinates of detected objects.
3. Confidence Scores: Reflecting the likelihood of correct detection.

Post-detection, the bounding boxes and class labels as seen in Figure 4.4 are rendered on the frames using OpenCV, providing visual feedback for real-time analysis. This robust approach ensures accurate detection and classification while maintaining low latency, suitable for autonomous navigation and environment perception tasks.

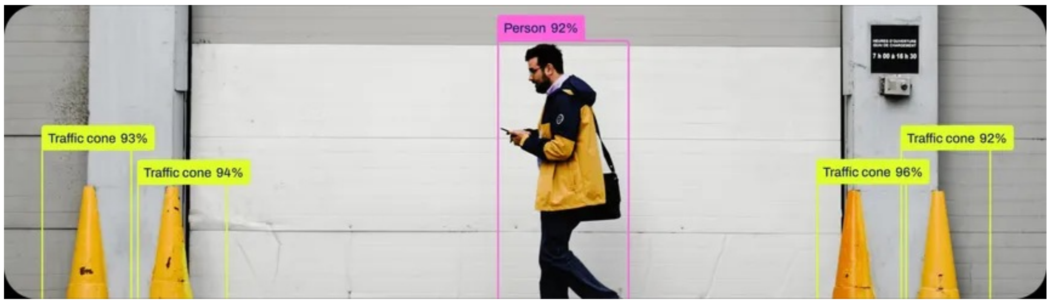


Figure 4.4: Object Detection

4.5 Navigating using miniSOM

The robot's navigation system utilizes a Self-Organizing Map (SOM) to autonomously navigate in an environment, aided by distance measurements from ultrasonic sensors. The methodology integrates the miniSOM algorithm with a system for logging movements and timestamps to track the robot's actions when a map cannot be created. The following steps outline the approach:

1. Sensor Data Collection The robot is equipped with four ultrasonic sensors: two positioned at the front (left and right) and two diagonally (left and right). These sensors continuously measure the distance to obstacles in the robot's surroundings. The data from the sensors are collected in real-time to inform navigation decisions.
   - Left Front: Measures the distance to obstacles on the left side.
   - Right Front: Measures the distance to obstacles on the right side.

- Left Diagonal: Measures the distance to obstacles in the left diagonal direction.
- Right Diagonal: Measures the distance to obstacles in the right diagonal direction.

2. Initialization of the Self-Organizing Map (miniSOM) The miniSOM algorithm is initialized with a 3x3 grid, where each node in the grid corresponds to a learned state based on the distance inputs. The SOM is trained using predefined training data that represents various navigation scenarios:

   - Move Forward: All sensors show similar distances indicating an open path.
   - Turn Left/Right: One side of the robot has a shorter distance, suggesting the need to turn.
   - Stop: Both front and diagonal sensors report short distances, indicating an obstacle ahead.

   The SOM is trained by feeding it example distance data, which allows the robot to classify its environment and make navigation decisions accordingly.

3. Navigation Logic After collecting sensor data, the robot calculates its position within the SOM grid using the 'winner' function. The SOM output determines the robot's movement:

   - Move Forward: The robot moves straight if the SOM classifies the state as safe (open space).
   - Turn Left/Right: If the SOM detects an obstacle on one side, the robot turns to avoid it.
   - Stop: If obstacles are detected in all directions, the robot stops to avoid collision.

4. Movement Logging and Time Recording Each movement action is logged with a timestamp to track the robot's behavior and time spent on each movement. This is done using a matrix, where each entry consists of:

   - Action: The movement executed (e.g., move forward, turn left, turn right, stop).
   - Timestamp: The time at which the action occurred, recorded using Python's `time.time()` function.The `movement_matrix` is updated every time a movement is performed, providing a log of the robot's journey. This matrix is printed at the end of the program to provide insight into the robot's behavior during the run.

5. Error Handling: No Map Creation If, for any reason, the robot is unable to create a map or the SOM fails to provide actionable outputs, the movement matrix serves as a fallback system. This ensures that the robot continues to operate based on sensor data, logging its actions and movement times for analysis, even without map creation.

6. Control and Motor Functions Motor control is implemented through GPIO pins, allowing the robot to perform basic movements like moving forward, turning left, turning right, and stopping. The control functions are activated based on the SOM output, which is influenced by real-time sensor data.

7. System Setup

   - GPIO Configuration: The Raspberry Pi's GPIO pins are configured for motor drivers and ultrasonic sensors.
   - Motor Driver Setup: Motor drivers control the movement of the robot based on the GPIO outputs.
   - Sensor Setup: Ultrasonic sensors are set up to continuously measure distances in four directions.

8. Continuous Operation The robot operates in a loop where sensor data is continuously collected, the SOM is queried for the best movement action, and the robot is directed to move. This loop runs indefinitely, allowing the robot to navigate autonomously. The process is interrupted by a keyboard command (Ctrl+C), which stops the robot and prints the movement log.

This methodology ensures that the robot can navigate using the SOM and log its movements, even when a map cannot be created. The combination of autonomous navigation and movement logging allows for adaptable behavior in a variety of environments.

Chapter 5

Result and Discussions

5.1 Final Hardware Model

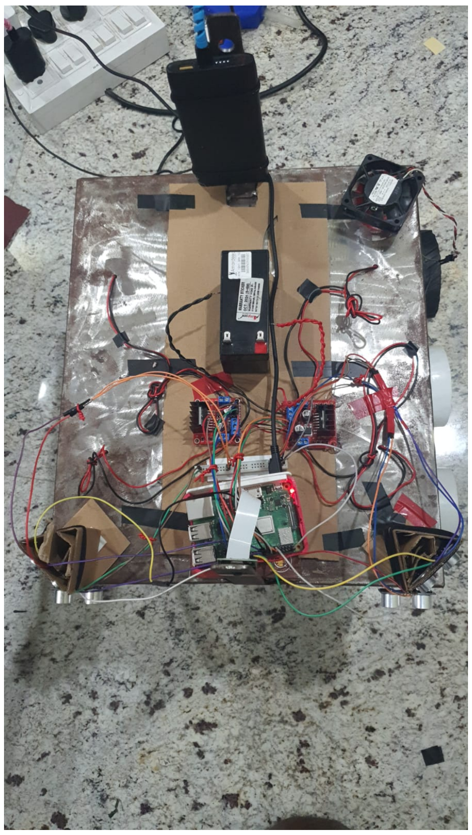


Figure 5.1: Hardware Model

The final hardware model shown in Figure 5.1 integrates two core applications: object detection for environment perception and an embedded application for robot movement. These functionalities are seamlessly combined into a compact, efficient, and autonomous robotic system.

The hardware comprises a Raspberry Pi 3B+ SBC, ultrasonic sensors, motor drivers, motors, and a robust chassis. The hardware connections can be seen in Figure 5.2. For environment perception, a computer vision system powered by a camera module operates alongside the object detection algorithm, enabling the robot to identify objects in its surroundings. This capability is critical for understanding environmental elements and distinguishing obstacles. The object detection module

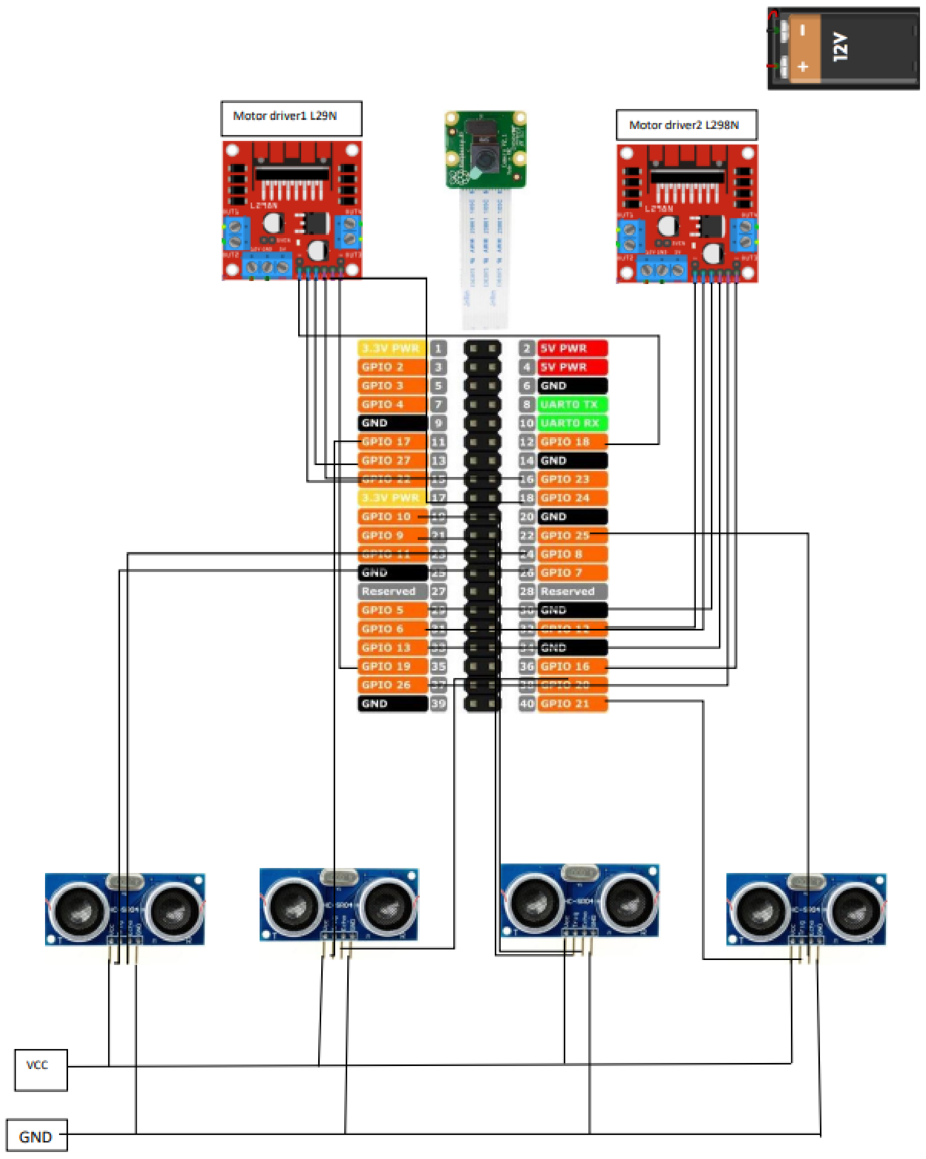


Figure 5.2: Circuit Interconnection

utilizes lighter version of TensorFlow, which is TFLite runtime, providing real-time insights into the robot's operational area.

The embedded application governs the robot's movement using ultrasonic sensors to measure distances and a self-organizing map (miniSOM) for navigation. The system dynamically determines optimal actions, such as moving forward, turning, or stopping, based on real-time sensor input. Motor drivers control wheel movement, ensuring smooth and precise operations. A logging mechanism records every movement and timestamp, which is stored in a movement matrix for later analysis.

5.2 Working of the Autonomous Robot

The dual-application hardware model demonstrates the integration of perception and action, with the object detection system feeding environmental data to inform the robot's navigation decisions. The design is versatile, enabling the robot to adapt to different environments and tasks, thereby showcasing the potential of combining vision-based perception and embedded movement systems.

5.2.1 Working of Object Detection Part

The object detection system utilizes TensorFlow Lite (TFLite) to enable efficient real-time object detection on the Raspberry Pi. The PiCamera captures live video frames at a resolution of 640x480 pixels, providing input to the detection algorithm. The pretrained TFLite model, optimized for edge devices, processes the frames to identify and classify objects, with a corresponding label file mapping class indices to human-readable names. Each frame is resized and preprocessed to meet the model input specifications, including adjustments to the color format and normalization. The processed frame is fed into the model, which outputs bounding boxes, class indices, and confidence scores for detected objects as shown in Figure 5.3. Post-processing scales the bounding boxes back to the original frame dimensions and filters detections based on a confidence threshold. Valid detections are visualized with bounding boxes and labels indicating object classes and confidence levels. The processed frames are streamed in real time via a Flask-based web application, allowing users to view live detection results through a network-connected device. This system combines TFLite's lightweight inference capabilities with real-time video processing, making it ideal for embedded AI applications.
The built robot model is kept at the entrance (as seen in Figure 5.4) of the obstacle environment. The robot is programmed in the way that with the help of 4 Ultrasonic sensors which are placed at an angle to each other of 45° detects the object distance readings kept in its path and takes required turning based on the Algorithm decision.

5.2.2 Working of Robot Navigation

The navigation system of the autonomous robot operates through an embedded control mechanism using ultrasonic sensors and the miniSOM algorithm for real-time decision-making. The robot is equipped with four ultrasonic sensors strategically positioned to measure distances to obstacles from the front-left, front-right, left-diagonal, and right-diagonal directions. These distance readings are continuously collected and fed into the Raspberry Pi microcontroller, which preprocesses the data before passing it to the miniSOM algorithm. The miniSOM, trained with pre-defined environmental scenarios, classifies the data into specific actions such as “Move Forward,” “Turn Left,” “Turn Right,” or “Stop,” depending on the proximity of obstacles. The Raspberry Pi then translates these decisions into motor control commands, using GPIO pins to drive the motors for precise movements. The system also logs each movement and its timestamp in a matrix, providing a detailed record of the robot’s actions for post-analysis. Operating in a loop, the navigation system ensures that the robot dynamically perceives its surroundings, adapts to changing environments, and navigates autonomously while maintaining safety and efficiency.

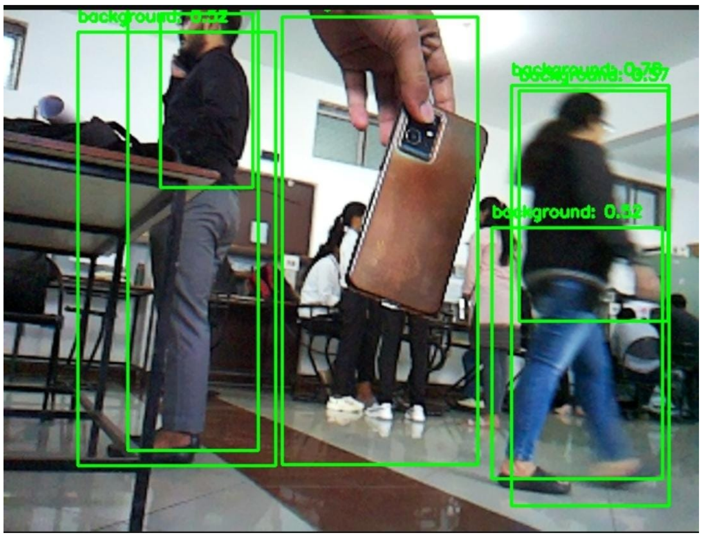


Figure 5.3: At the entrance

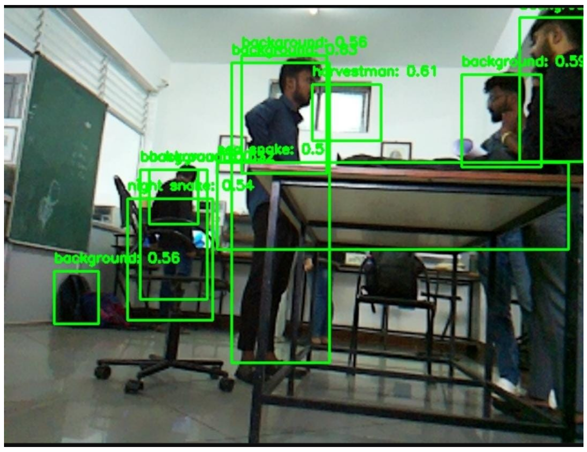


Figure 5.4: Moving in the Obstacle environment

Chapter 6

Conclusion and Future scope

6.1 Conclusion

In conclusion, the NeuroHex project successfully demonstrates the potential of integrating Artificial Neural Networks (ANN) with 6 Legged robotics to achieve autonomous navigation and enhanced environmental perception. The system utilizes the flexibility and adaptability of ANN to process sensor data, enabling real-time decision-making and obstacle avoidance. The hexapod's six-legged design ensures superior stability and mobility across various terrains, making it ideal for applications in unpredictable and complex environments. Key outcomes of the project include.

The project's achievements include:

1. The modular design of both hardware and software facilitates future upgrades and adaptations for diverse applications.
2. The implementation of ANN algorithms allows the hexapod to navigate autonomously, minimizing the need for human intervention.
3. The ANN-based control system enables seamless navigation in complex and dynamic environments.
4. The hexapod's biomimetic design ensures superior stability over uneven and challenging terrains.
5. NeuroHex excels in environmental perception through effective sensor fusion.
6. ANN algorithms facilitate real-time decision-making for obstacle avoidance and path planning.
7. The modular architecture of NeuroHex allows easy customization and scalability for diverse applications.
8. The project demonstrates utility in search and rescue, industrial inspections, and remote explorations.

9. The system proves capable of operating without human intervention in various scenarios.

6.2 Future Scope

The future scope of this project is quite extensive, as there are numerous ways to expand and improve upon the initial goals and challenges.

Here are some potential directions for future work:

- **Use of more enhanced Mapping libraries:** Due to computational incompetency of the Raspberry Pi 3B+ advance mapping libraries such as ORB-SLAM2 was not deployed. The higher versions of Raspberry Pi can made used to do so.
- **Custom trained models for specific purposes:** We can train the model and create a new dataset and labelmap for the sepcified application of use.
- **Integration of Advanced Sensors**: Incorporating thermal imaging, or other high-resolution sensors to enhance the robot's environmental understanding.
- **Real-world Applications**: Expanding the system for specific applications such as search and rescue missions, industrial inspection, agricultural automation, and planetary exploration.
- **Testing in Diverse Environments**: Conducting extensive testing in varied terrains and environmental conditions to ensure robustness and reliability.
- **Miniaturization**: Designing smaller versions for reconnaissance or medical applications in confined spaces.
- **Extraterrestrial Exploration**: Testing NeuroHex for planetary exploration and data collection in extraterrestrial environments.

References

[1] R. Szabó and A. Gontean, ”Industrial robotic automation with Raspberry PI using image processing,” 2016 International Conference on Applied Electronics (AE), Pilsen, Czech Republic, 2016, pp. 265-268, doi: 10.1109/AE.2016.7577287. keywords: Manipulators;Service robots;Universal Serial Bus;Cameras;Robot vision systems;Production;actuator;automation;camera;end effectors;image processing;industrial control;Raspberry PI;robotic arm.

[2] Chai, X., Gao, F., Qi, C. et al. Obstacle avoidance for a hexapod robot in unknown environment. Sci. China Technol. Sci. 60, 818–831 (2017). https://doi.org/10.1007/s11431-016-9017-6

[3] Patel, Dhruv, Priyam Dalwadi, Shubham Dhumal, and Abhishek Bhalodiya. ”Development and Design of Autonomous Navigation Robot Using Raspberry Pi and Computer Vision.” International Research Journal of Engineering and Technology (IRJET) 7 (2020): 864-869.

[4] Maideen, A., Mohanarathinam, A. (2023). Computer Vision-Assisted Object Detection and Handling Framework for Robotic Arm Design Using YOLOV5. Adcij Advances in Distributed Computing and Artificial Intelligence Journal, 12, e31586. https://doi.org/10.14201/adcaij.31586

[5] De Gouvêa, R. C. T., Gioria, R. D. S., Marques, G. R., De Carvalho Carneiro, C. (2023b). IntraSOM: A comprehensive Python library for Self-Organizing Maps with hexagonal toroidal maps training and missing data handling. Software Impacts, 17, 100570. https://doi.org/10.1016/j.simpa.2023.100570

[6] Greco, D., Fasihiany, M., Ranjbar, A. V., Masulli, F., Rovetta, S., Cabri, A. (2024). Computer Vision Algorithms on a Raspberry Pi 4 for Automated Depalletizing. Algorithms, 17(8), 363. https://doi.org/10.3390/a17080363

[7] https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf

[8] https://quartzcomponents.com/products/dual-shaft-dc-geared-motor-300-rpm.

[9] https://components101.com/sensors/ultrasonic-sensor-working-pinout-datasheet.

Appendix

A.Working Code for Object Detection

```python
import tflite_runtime.interpreter as tflite
import numpy as np
from flask import Flask, Response
from picamera2 import Picamera2
import cv2

# Create the webApp instance
app = Flask(__name__)

# Initialize and configure the camera
camera = Picamera2()
camera.configure(camera.create_preview_configuration(main={"format": 'XRGB8888', "size": (640, 480)}))
camera.start()

# Load the TFLite model
model_path = '/home/ullas/Ullas/tflite_model/detect.tflite'  # u can use MobileNet_V2.py too
labels_path = '/home/ullas/Ullas/tflite_model/labelmap.txt'

interpreter = tflite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()

# Load labels
with open(labels_path, "r") as f:
    labels = {i: line.strip() for i, line in enumerate(f.readlines())}

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Debug input details
print("Input Details:", input_details)

# Preprocess frames for detection
def preprocess_frame(frame):
    # Resize frame to model input size
    input_shape = input_details[0]['shape'][1:3]  # Expected width and height
    resized_frame = cv2.resize(frame, tuple(input_shape))

    # Remove alpha channel if present
    if resized_frame.shape[2] == 4:
        resized_frame = cv2.cvtColor(resized_frame, cv2.COLOR_BGRA2RGB)  # Convert BGRA to RGB

    # Convert to UINT8 if required
    if input_details[0]['dtype'] == np.uint8:
        input_data = np.expand_dims(resized_frame, axis=0).astype(np.uint8)
    else:
        input_data = np.expand_dims(resized_frame, axis=0).astype(np.float32)
        input_data = input_data / 255.0  # Normalize if expected dtype is FLOAT32

    # Debug preprocessed frame
    print("Preprocessed Frame Shape:", input_data.shape, "Dtype:", input_data.dtype)
    return input_data

# Perform Object Detection
def detect_objects(frame):
    input_data = preprocess_frame(frame)

    try:
        # Set input tensor and invoke the model
        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()
    except ValueError as e:
        print(f"Error setting tensor or invoking model: {e}")
        return [], [], []

    # Get output tensors (bounding boxes, classes, and scores)
    boxes = interpreter.get_tensor(output_details[0]['index'])[0]
    classes = interpreter.get_tensor(output_details[1]['index'])[0]
    scores = interpreter.get_tensor(output_details[2]['index'])[0]

    # Debug detected objects
    print(f"Detected Objects - Boxes: {boxes}, Classes: {classes}, Scores: {scores}")
    return boxes, classes, scores

# Draw Detected Objects
def draw_boxes(frame, boxes, classes, scores, threshold=0.5):
    height, width, _ = frame.shape
    for i, score in enumerate(scores):
        if score >= threshold:
            box = boxes[i]
            y_min = int(box[0] * height)
            x_min = int(box[1] * width)
            y_max = int(box[2] * height)
            x_max = int(box[3] * width)
            class_id = int(classes[i])
            label = labels.get(class_id, "Unknown")

            # Draw bounding box
            cv2.rectangle(frame, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)

            # Add label
            cv2.putText(frame, f"{label}: {score:.2f}", (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0))

    return frame

# Integrate into the Flask application
def generate_frames():
    while True:
        frame = camera.capture_array()

        # Perform object detection
        boxes, classes, scores = detect_objects(frame)

        # Draw bounding boxes on the frame
        frame = draw_boxes(frame, boxes, classes, scores)

        # Encode the frame
        ret, buffer = cv2.imencode('.jpg', frame)
        if not ret:
            print("Failed to encode frame")
            continue
        frame = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Listing 6.1: Object Detection Code using TFLite

B. Working Code for Navigation with SOM and ultrasonic Sensors

```
import RPi.GPIO as GPIO
import time
from minisom import MiniSom
import numpy as np

# GPIO Setup
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Motor Driver 1 (left motors) Pin Assignments
ena1, enb1 = 18, 19
in1_1, in2_1, in3_1, in4_1 = 22, 27, 23, 24
# Motor Driver 2 (right motors) Pin Assignments
ena2, enb2 = 12, 16
in1_2, in2_2, in3_2, in4_2 = 6, 5, 13, 26
# Ultrasonic Sensor Pins
trigger_left, echo_left = 17, 20
trigger_right, echo_right = 21, 25
trigger_left_diag, echo_left_diag = 7, 8
trigger_right_diag, echo_right_diag = 9, 10

# Movement Matrix for storing actions and time
movement_matrix = []

# Motor Setup
def motor_setup():
    motor_pins = [ena1, enb1, in1_1, in2_1, in3_1, in4_1, ena2, enb2,
   in1_2, in2_2, in3_2, in4_2]
    for pin in motor_pins:
        GPIO.setup(pin, GPIO.OUT)

# Motor Control Functions
def move_forward():
    GPIO.output(in1_1, GPIO.HIGH)
    GPIO.output(in2_1, GPIO.LOW)
    GPIO.output(in3_1, GPIO.HIGH)
    GPIO.output(in4_1, GPIO.LOW)
    GPIO.output(in1_2, GPIO.HIGH)
    GPIO.output(in2_2, GPIO.LOW)
    GPIO.output(in3_2, GPIO.HIGH)
    GPIO.output(in4_2, GPIO.LOW)
    record_movement('move_forward')

def turn_left():
    GPIO.output(in1_1, GPIO.LOW)
    GPIO.output(in2_1, GPIO.HIGH)
    GPIO.output(in3_1, GPIO.HIGH)
    GPIO.output(in4_1, GPIO.LOW)
    GPIO.output(in1_2, GPIO.HIGH)
    GPIO.output(in2_2, GPIO.LOW)
    GPIO.output(in3_2, GPIO.LOW)
    GPIO.output(in4_2, GPIO.HIGH)
    record_movement('turn_left')

def turn_right():
    GPIO.output(in1_1, GPIO.HIGH)
    GPIO.output(in2_1, GPIO.LOW)
    GPIO.output(in3_1, GPIO.LOW)
    GPIO.output(in4_1, GPIO.HIGH)
    GPIO.output(in1_2, GPIO.LOW)
    GPIO.output(in2_2, GPIO.HIGH)
    GPIO.output(in3_2, GPIO.HIGH)
    GPIO.output(in4_2, GPIO.LOW)
    record_movement('turn_right')

def stop():
    GPIO.output(in1_1, GPIO.LOW)
    GPIO.output(in2_1, GPIO.LOW)
    GPIO.output(in3_1, GPIO.LOW)
    GPIO.output(in4_1, GPIO.LOW)
    GPIO.output(in1_2, GPIO.LOW)
    GPIO.output(in2_2, GPIO.LOW)
    GPIO.output(in3_2, GPIO.LOW)
    GPIO.output(in4_2, GPIO.LOW)
    record_movement('stop')

# Record the movement and timestamp in the matrix
def record_movement(action):
    current_time = time.time()
    movement_matrix.append([action, current_time])

# Ultrasonic Sensor Functions
def setup_ultrasonic(trigger, echo):
    GPIO.setup(trigger, GPIO.OUT)
    GPIO.setup(echo, GPIO.IN)
```

```python

def get_distance(trigger, echo):
    GPIO.output(trigger, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(trigger, GPIO.LOW)

    start_time = time.time()
    stop_time = time.time()

    while GPIO.input(echo) == 0:
        start_time = time.time()

    while GPIO.input(echo) == 1:
        stop_time = time.time()

    time_elapsed = stop_time - start_time
    distance = (time_elapsed * 34300) / 2
    return distance

# MiniSOM Initialization
som = MiniSom(x=3, y=3, input_len=4, sigma=1.0, learning_rate=0.5)
training_data = []  # Placeholder for collected ultrasonic data

# Training the SOM (replace with actual data collection and training)
training_data = [
    [30, 30, 50, 50],  # Move forward
    [15, 30, 50, 50],  # Turn right
    [30, 15, 50, 50],  # Turn left
    [10, 10, 20, 20],  # Obstacle ahead
]
som.random_weights_init(training_data)
som.train_random(training_data, 100)  # 100 iterations

# Navigation Logic
def navigate():
    dist_left = get_distance(trigger_left, echo_left)
    dist_right = get_distance(trigger_right, echo_right)
    dist_left_diag = get_distance(trigger_left_diag, echo_left_diag)
    dist_right_diag = get_distance(trigger_right_diag, echo_right_diag)

    input_data = np.array([dist_left, dist_right, dist_left_diag,
   dist_right_diag])
    winner = som.winner(input_data)

    # Control logic based on SOM output
    if winner == (0, 0):
        move_forward()
    elif winner == (0, 1):
        turn_right()
    elif winner == (1, 0):
        turn_left()
    else:
        stop()

# Main Loop
if __name__ == "__main__":
    try:
        motor_setup()
        setup_ultrasonic(trigger_left, echo_left)
        setup_ultrasonic(trigger_right, echo_right)
        setup_ultrasonic(trigger_left_diag, echo_left_diag)
        setup_ultrasonic(trigger_right_diag, echo_right_diag)

        while True:
            navigate()
            time.sleep(0.1)  # Delay between cycles
    except KeyboardInterrupt:
        stop()
        GPIO.cleanup()
        print("Program terminated.")
        print("Movement Log: ", movement_matrix)
```

Listing 6.2: Navigation Code with MiniSOM and Ultrasonic Sensors