A Project Report on

ROS-BASED VOICE CONTROLLED MOBILE ROBOTIC ARM FOR AUTOMATIC SEGREGATION OF MEDICAL WASTE

Submitted to

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY , Hyderabad

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

A Sai Saketh	**(21BD1A0501)**
Gunaabhi Ram D	**(21BD1A050B)**
L Ajit Reddy	**(21BD1A050N)**
Sreeja Reddy K	**(21BD1A051Q)**
Anjali Dulam	**(21BD1A0565)**

Under the guidance of

Mr. Sai Krishna
Department of R & D

Department of Computer Science and Engineering

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Approved by AICTE, Affiliated to JNTUH

3-5-1206, Narayanaguda, Hyderabad, TS- 500029

2020 – 2021

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Accredited by NBA & NAAC, Approved By A.I.C.T.E., Reg by Govt of Telangana State & Affiliated to JNTU, Hyderabad)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled being entitled "**ROS-BASED VOICE CONTROLLED MOBILE ROBOTIC ARM FOR AUTOMATIC SEGREGATION OF MEDICAL WASTE**" being submitted by

A Sai Saketh	**21BD1A0501**
Gunaabhi Ram D	**21BD1A050B**
L Ajit Reddy	**21BD1A050N**
Sreeja Reddy K	**21BD1A051Q**
Anjali Dulam	**21BD1A0565**

InternalGuide

(MR. SAI KRISHNA)

HeadoftheDepartment

(MRS. RUPA)

Submitted for Viva Voce Examination held on _____

External Examiner

Unit of Keshav Memorial Educational Society

#: 3-5-1026 Narayanaguda Hyderabad 500029.

040-3261407 www.kmit.in e-mail:principal@kmit

CONTENTS

CHAPTER - 4

CHAPTER - 5

CHAPTER-6

CHAPTER - 7

CHAPTER - 8 **72-74**

LIST OF FIGURES

ABSTRACT

This project presents a system for controlling a robotic arm mounted on a 6WD rover using voice commands to achieve automatic segregation of medical waste. The system incorporates a voice recognition system to convert voice commands into actionable instructions. Perception capabilities are integrated, employing sensors such as cameras for object detection and segregation of medical waste. The robotic arm's control software facilitates precise manipulation and segregation of waste based on its classification. The system operates in a feedback loop, continuously processing voice commands, detecting and classifying medical waste, and performing the necessary actions. The proposed system has the potential to improve waste management processes in healthcare facilities, offering increased efficiency, accuracy, and safety in the segregation of medical waste.

CHAPTER - 1

1. INTRODUCTION

1.1 PROBLEM STATEMENT:

Develop a system to detect and segregate medical waste from other wastes using a mobile robotic arm based on (ROS) platform to pick up and drop the waste object into color-coded bins, using a suitable deep learning-based algorithm to detect the waste. Use a voice recognition system to control the mobile robotic arm to perform the task.

1.2 PURPOSE OF THE PROJECT:

The outbreak of the Covid-19 pandemic has resulted in a surge in the generation of medical waste. Due to the transmissible nature of the Virus and the lack of effort at proper disposal, the safety of the front-line health workers, as well as the disposer, is at risk. Hence, to mitigate the spread of infectious diseases, a system is proposed that uses a robotic arm for segregating medical waste automatically. The robotic arm is operable through voice commands and is placed on a 6 wd robotic rover. The system uses the YOLOv5 (You Only Look Once) algorithm to detect and classify medical waste and then uses the Robot Operating System (ROS) platform to pick up and drop the waste object into color-coded bins. For this research, the medical waste has been categorized into 4 types, and for each type, a color-coded bin has been used for segregation. Our system has achieved 94% training accuracy for the YOLOv5 model on a custom dataset.

1.3 SYSTEM DESIGN

Designing a ROS-based voice-controlled robotic arm system for automatic segregation of medical waste using a 6WD rover and robotic arm involves multiple components and interactions.Here is the system design that outlines the major components and their interactions:

1. **Hardware Components:**

- DC motors
- Jetson nano
- Jumper wires
- UBEC
- Pixhawk
- servo motor
- Power module
- Raspberry pi camera module V2
- Sabertooth
- PCA9685
- Lithium ion battery

2. **Software Components:**

- ROS (Robot Operating System)
- Voice Recognition.
- Object Detection and Segmentation
- Path Planning and Navigation
- Robotic Arm Control
- Waste Segregation Logic

3. **System Workflow:**

1. Voice Input: The microphone captures voice commands from the user.
2. Voice Recognition: The voice recognition system processes the captured audio and converts it into text commands.
3. Command Interpretation: The text commands are interpreted to identify the intended action
4. Object Detection: The perception system uses sensors (e.g., cameras) to detect and identify medical waste objects in the surroundings.
5. Waste Classification: The detected objects are classified based on their characteristics (e.g., color, shape, labels) to determine the type of medical waste.
6. Path Planning: The system plans a safe and efficient path for the rover to reach the location of the identified waste object.
7. Rover Navigation: The rover moves using the 6WD system and navigates to the desired location.
8. Robotic Arm Control: Once the rover reaches the waste object, the robotic arm is commanded to pick it up or perform the required action for segregation.
9. Waste Segregation: Based on the waste object classification, the robotic arm performs the appropriate actions
10. Feedback and Loop: The system repeats the above steps to process additional voice commands and continue with the segregation process until completed.

CHAPTER - 2

2. ARCHITECTURE AND CONSTRUCTION OF ROVER

2.1 Hardware Requirements

- DC MOTORS :

A DC (Direct Current) motor is an electrical device that converts electrical energy into mechanical energy. It operates using direct current, where the flow of electric current is unidirectional. DC motors are widely used in various applications, including robotics, industrial machinery, automotive systems, and home appliances.

DC motors offer several advantages, such as high torque at low speeds, easy speed and direction control, and relatively simple construction. However, they also have limitations, including the need for regular maintenance (e.g., brush replacement in brushed DC motors) and the generation of electrical noise.

Overall, DC motors are widely used due to their versatility and ability to provide precise control over motor speed and torque, making them suitable for a wide range of applications.

Fig 2.1 : Dc motor

-PIXHAWK :

Pixhawk is an open-source hardware and software platform designed for autonomous vehicles, particularly unmanned aerial vehicles (UAVs) or drones. It provides a complete flight control system that includes a flight controller board, sensors, and software, enabling the control and navigation of autonomous vehicles.

The Pixhawk project was initiated by the open-source community and is currently maintained by the PX4 open-source community. The Pixhawk flight controller board is based on the ARM Cortex-M microcontroller architecture and features various sensors such as accelerometers, gyroscopes, magnetometers, barometers, and GPS receivers.

The Pixhawk flight controller board is connected to the various sensors and actuators of the vehicle, allowing it to collect data and make decisions based on the autopilot software running on the board. The software provides features like stabilization, autonomous flight, waypoint navigation, and support for different flight modes.

One of the notable features of Pixhawk is its compatibility with different types of vehicles, including multirotors, fixed-wing aircraft, helicopters, and even ground vehicles. It has a wide range of inputs and outputs, making it highly versatile and suitable for a variety of applications.

Fig 2.2 : Pixhawk

- SABERTOOTH 2x32 MOTOR DRIVER :

The Sabertooth 2x32 motor driver is a dual-channel motor controller designed for controlling two brushed DC motors. It is produced by Dimension Engineering, a company known for its motor control solutions.

The Sabertooth 2x32 motor driver is capable of delivering up to 32 amps of continuous current per channel, making it suitable for driving high-power motors. It supports a wide voltage range, typically from 6V to 30V, allowing it to work with various battery configurations.

Here are some key features of the Sabertooth 2x32 motor driver:

1. Dual-channel control: It can independently control two DC motors, allowing differential drive or independent control of two separate motors.
2. Motor current sensing: The motor driver provides real-time feedback on the current being drawn by each motor, which can be useful for monitoring and protection purposes.
3. Simplified control interface: The Sabertooth 2x32 motor driver utilizes simplified serial and analog inputs for controlling the motors, making it easy to interface with microcontrollers, Arduino boards, or other control systems.

Fig 2.3 : 2X32 Sabertooth

- JETSON NANO :

The Jetson Nano is a small, power-efficient computer module developed by NVIDIA specifically for AI and robotics applications. It is part of the Jetson family of embedded computing platforms, which are designed to accelerate artificial intelligence (AI) workloads at the edge.

Here are some key features of the Jetson Nano:

1. GPU Acceleration: The Jetson Nano integrates a powerful NVIDIA Maxwell architecture GPU with 128 CUDA cores, enabling high-performance computing for AI inference and image processing tasks.
2. CPU and Memory: It is equipped with a quad-core ARM Cortex-A57 CPU running at 1.43 GHz and 4GB of LPDDR4 RAM, providing a balance of CPU and GPU processing power.
3. AI Performance: The Jetson Nano is capable of running popular AI frameworks and libraries such as TensorFlow, PyTorch, and Caffe, enabling developers to deploy and optimize deep learning models directly on the device.
4. Connectivity: It offers various connectivity options, including Gigabit Ethernet, USB 3.0, USB 2.0, and HDMI. It also has an M.2 Key E connector for Wi-Fi and Bluetooth modules.
5. Expansion and IO: The Jetson Nano features a 40-pin GPIO header that provides interfaces for connecting peripherals and expansion boards. This allows for custom hardware integrations and prototyping.

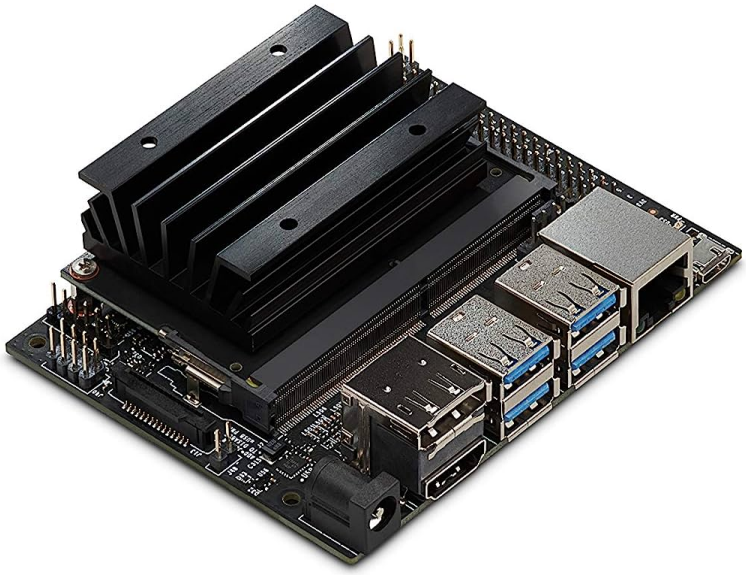


Fig 2.4 : Jetson Nano

- SERVO MOTORS (ROBOTIC ARM):

Servo motors are a type of rotary actuator that provides precise control over angular position. They are commonly used in various applications that require accurate and controlled movement, such as robotics, industrial automation, RC vehicles, and aerospace.

Here are some key features and characteristics of servo motors:

1. Control Mechanism: Servo motors are typically controlled using a closed-loop feedback system. They have built-in position feedback devices, such as potentiometers or encoders, that provide information about the motor's current position. This feedback is used to adjust the motor's control signals and ensure accurate positioning.
2. Position Control: Servo motors excel at achieving and maintaining specific positions within a defined range of motion. They can rotate to a desired angle and hold that position with minimal or no drift. This makes them ideal for applications that require precise positioning.
3. Torque and Speed: Servo motors are available in various sizes and power ratings, providing a wide range of torque and speed options. The torque output of a servo motor is directly proportional to its physical size and the current supplied to it. Speed is determined by the motor's design and control signals.
4. PWM Control: Servo motors are commonly controlled using Pulse Width Modulation (PWM) signals. By varying the width of the PWM signal, the control system can determine the desired position for the motor. The servo motor's built-in controller interprets the PWM signal and adjusts the motor's position accordingly.

Fig 2.5 : Servo Motor

- PCA9685SERVO CONTROLLER:

The PCA9685 is a popular 16-channel PWM (Pulse Width Modulation) driver module. It is commonly used to control multiple servo motors, LEDs, or other devices that require precise and adjustable control of their output signals**.**

Here are some key features and characteristics of the PCA9685:

1. PWM Control: The PCA9685 can generate PWM signals on up to 16 individual channels. Each channel can be independently controlled with a programmable duty cycle (0-100%) and frequency. This allows for precise control of devices connected to each channel.
2. I2C Communication: The PCA9685 module communicates with a microcontroller or other devices using the I2C (Inter-Integrated Circuit) protocol. It has a 7-bit I2C address, which can be configured using address pins on the module, allowing multiple PCA9685 modules to be used simultaneously on the same I2C bus.
3. Adjustable Frequency: The PCA9685 supports a programmable PWM frequency from 24 Hz to 1526 Hz, which provides flexibility in choosing the appropriate frequency for different applications.
4. Output Resolution: The PCA9685 has a resolution of 12 bits per channel, allowing for fine-grained control over the duty cycle. This high resolution results in smooth and precise control of connected devices.

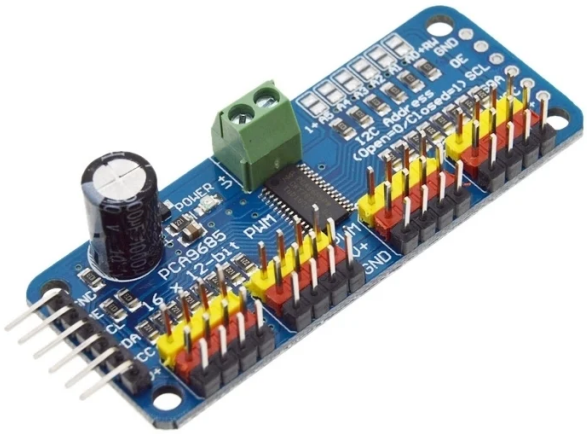


Fig 2.6 : PCA9685

- JUMPER WIRES :

Jumper wires, also known as jumper cables or DuPont wires, are simple electrical wires with connectors at both ends. They are widely used in electronics and prototyping to create temporary or permanent connections between components on breadboards, circuit boards, or other electronic modules. Here are some key features and characteristics of jumper wires:

1. Connectors: Jumper wires typically have male pins or connectors, such as male header pins or banana plugs, at each end. The connectors allow for easy insertion and removal from various electronic components, such as headers, sockets, or pins.
2. Length and Color Coding: Jumper wires come in different lengths, typically ranging from a few centimeters to several inches. They are often color-coded, with different colors representing different functions or signals. This makes it easier to identify and organize connections in a circuit or prototype.
3. Wire Gauge: The gauge or thickness of the wire used in jumper wires can vary. Thinner wires with higher gauges are suitable for low-power and signal-level connections, while thicker wires with lower gauges can handle higher currents.
4. Flexibility and Insulation: Jumper wires are usually made of stranded copper wire, which offers flexibility and durability. The wires are often insulated with materials such as PVC or silicone to protect against short circuits and provide electrical isolation between adjacent connections.

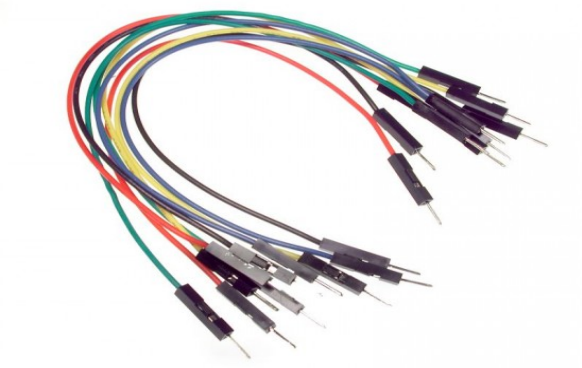


Fig 2.7 : Jumper wires

-POWER MODULE:

A power module is an integrated circuit (IC) or module that combines power semiconductors, control circuitry, and other components to provide specific power functions. It is designed to handle high currents or high voltages and often serves as a building block for power electronics applications.

Power modules come in various forms and serve different purposes. Here are a few examples:

1. Motor Drive Modules: Motor drive modules are power modules specifically designed for controlling electric motors. They integrate power transistors or insulated gate bipolar transistors (IGBTs), along with the necessary control circuitry, to drive motors efficiently and protect against overcurrent or overheating.
2. Power Amplifier Modules: Power amplifier modules are used in audio systems, wireless communication devices, or other applications requiring high-power amplification. They integrate power transistors or RF power amplifiers along with control circuitry to amplify signals while maintaining high efficiency.
3. Switching Power Supply Modules: Switching power supply modules are used to convert electrical power efficiently from one voltage level to another. They typically include power semiconductors such as MOSFETs or IGBTs, along with control circuitry for regulating the output voltage and current. These modules are commonly found in computer power supplies, industrial equipment, and consumer electronics.

Fig 2.8 : Power module

-LITHIUM ION BATTERY :

A lithium-ion battery, often abbreviated as Li-ion battery, is a type of rechargeable battery that uses lithium ions as the primary component of its electrochemical system. It has become one of the most popular types of batteries due to its high energy density, long cycle life, and relatively low self-discharge rate.

Here are some key features and characteristics of lithium-ion batteries:

1. High Energy Density: Lithium-ion batteries offer a high energy density, meaning they can store a significant amount of energy in a relatively compact and lightweight package. This makes them ideal for portable electronics such as smartphones, laptops, and electric vehicles.
2. Rechargeable: Lithium-ion batteries are rechargeable, allowing them to be used multiple times before needing replacement. They can undergo hundreds to thousands of charge-discharge cycles, depending on the specific battery chemistry and usage conditions.
3. Low Self-Discharge Rate: Compared to other rechargeable batteries, lithium-ion batteries have a relatively low self-discharge rate. This means they can retain their charge for longer periods when not in use, making them convenient and reliable power sources.
4. No Memory Effect: Lithium-ion batteries do not suffer from memory effect, which is a phenomenon where batteries lose their maximum capacity if not fully discharged before recharging. This allows users to recharge lithium-ion batteries at any point in their discharge cycle without affecting their overall capacity.

Fig 2.9 : Lithium ion battery

- UBEC:

A UBEC (Universal Battery Elimination Circuit) is a type of voltage regulator used in radio-controlled (RC) vehicles and other electronics applications. It is specifically designed to convert a higher voltage from a battery or power source into a regulated lower voltage suitable for powering electronic components.

Here are some key features and characteristics of a UBEC:

1. Voltage Regulation: The primary function of a UBEC is to regulate the voltage supplied to connected electronic components. It typically takes a higher input voltage, such as from a LiPo battery or other power source, and steps it down to a stable and regulated lower voltage, commonly 5V or 6V. This ensures that the connected electronics receive a consistent and appropriate voltage level.
2. Current Output: UBECs are rated for a maximum current output that they can provide to the connected devices. This rating determines the amount of current that can be drawn by the electronic components without exceeding the UBEC's capabilities. It is important to select a UBEC with a current rating suitable for the power requirements of the connected devices.
3. Low Noise and Ripple: A good UBEC minimizes noise and ripple in the output voltage, ensuring clean and stable power supply to the connected electronics. This is crucial for sensitive components such as receivers, flight controllers, or other electronic devices that require a reliable and noise-free power source.

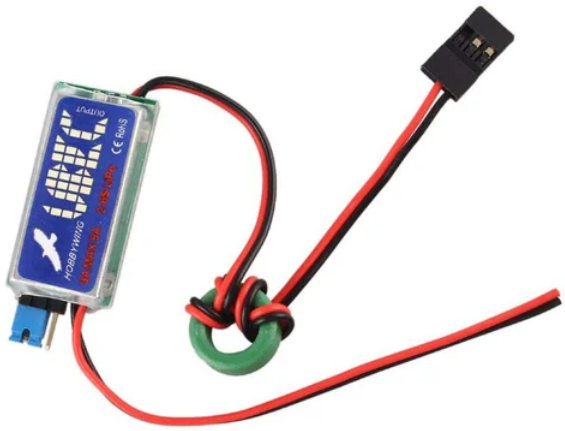


Fig 2.10 : UBEC

-RASPBERRY PI CAMERA MODULE V2:

The Raspberry Pi Camera Module V2 is a small camera board designed specifically for use with Raspberry Pi single-board computers. It provides a convenient way to capture still images and record video directly from a Raspberry Pi.

Here are some key features and characteristics of the Raspberry Pi Camera Module V2:

1. Image Quality: The camera module features an 8-megapixel Sony IMX219 image sensor, capable of capturing high-resolution still images and video.
2. Video Recording: It supports video recording at various resolutions, including 1080p Full HD at 30 frames per second (fps) and 720p HD at 60fps. It can also record at lower resolutions and frame rates for specific requirements.
3. Still Image Capture: The camera module allows for capturing still images with a resolution of up to 3280 x 2464 pixels. It offers support for various image formats, including JPEG, PNG, and RAW formats.
4. Interchangeable Lenses: The camera module has a removable lens assembly, allowing for the attachment of different lenses or lens adapters. This enables users to experiment with different focal lengths and perspectives, depending on their specific needs.

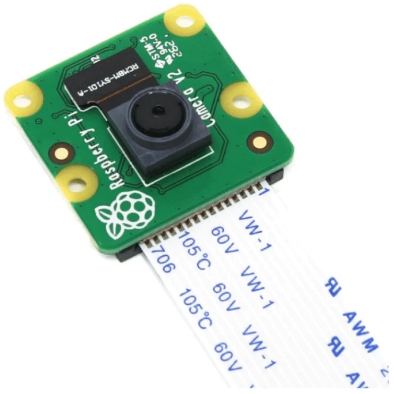


Fig 2.11 : Raspberry pi camera module V2

2.2 Construction of 6 WD rover

The rover's frame serves as the foundation, consisting of two 30x2 cm and two 10x2 cm rods assembled using L-shaped clamps. The symmetrical design ensures stability and proper weight distribution. On top of this frame, we place a durable 30x14 cm aluminum sheet with a 1 mm thickness. This sheet acts as the main platform, providing ample space for mounting various components. Using T nuts and screws, we securely attach the aluminum sheet to the frame. We create a 2x1 cm slit in the sheet to accommodate the motor wires, allowing for seamless connections to the motor driver.

The rover's middle plate is designated for the Jetson Nano, a powerful onboard computer. The Jetson Nano facilitates advanced processing and decision-making capabilities, essential for autonomous functionalities. The upper plate, on the other hand, houses the Pixhawk flight controller and a robotic arm. This arrangement ensures effective management of the robotic arm's movements and the Pixhawk's control over the rover's navigation.

For propulsion, we equip the rover with twelve 12V DC motors with a high rotational speed of 620 revolutions per minute(rpm). We securely solder connecting wires to the motor terminals and insulate them to ensure safety. These motors are connected to the 2x32 Sabertooth motor controller, offering precise speed and direction control. We use 2, 5, and 6 dip switches on the Sabertooth to manage the rover's movement effectively. The power module is connected to the B+ and B- terminals of the motor controller.

The rover relies on a total of six wheels for optimal mobility and stability. These wheels are strategically positioned and provide the necessary traction for various terrains and challenging environments. The battery is positioned at the bottom plate, and we connect it to a reliable power supply module. This setup ensures sufficient power for extended rover operations and allows for easy battery replacement when needed.

To enable precise navigation and positioning, we integrate a GPS module onto the rover. The GPS module connects to the Pixhawk flight controller, providing accurate location data for autonomous navigation and mission planning.

Pixhawk, it is equipped with essential components such as a safety switch and buzzer for smooth operation. The telemetry system connects to TELEML1, enabling real-time data transmission and monitoring. The GPS module interfaces with the Pixhawk through a 4-pin connector, and the I2C connection is established through a 2-pin connector.

Additionally, the motor driver signals S1 and S2 are directed to Main Out 1 and 3 on the Pixhawk. Proper grounding of the Pixhawk is maintained by connecting it to the motor driver's ground. The power module is linked to the Pixhawk's power through a 6-pin connector.

To connect a 6-degree-of-freedom (6-DOF) robotic arm to the PCA9685, which is then connected to a Jetson Nano, you'll need to follow the steps.

1. Connect the Robotic Arm Servo Motors to PCA9685: Identify the 6 servo motors used for the 6-DOF robotic arm. Each servo motor will have three wires: power (usually red), ground (usually black or brown), and control signal (usually yellow or white). Connect the power wires of all the servo motors to a suitable power supply that provides the required voltage and current for the servos. Be cautious to avoid overloading the PCA9685 or the power supply. Connect all the ground wires of the servo motors to the ground (GND) pin of the PCA9685.
2. Connect PCA9685 to Jetson Nano: Connect the PCA9685 module to the Jetson Nano via the I2C bus. The PCA9685 uses I2C communication for control, which requires connecting the SDA (data) and SCL (clock) pins of the PCA9685 to the D2 and D3 pins on the Jetson Nano.

After fixing of the robotic arm, we will place the arm on the second platform of the rover.

Fig 2.12 : 6WD Rover

CHAPTER - 3

3. SOFTWARE REQUIREMENTS SPECIFICATIONS AND NAVIGATION

3.1 What is Ardupilot

ArduPilot is an open-source autopilot software suite for controlling autonomous vehicles, including drones (unmanned aerial vehicles or UAVs), ground rovers, and boats. It provides a comprehensive set of features for autonomous navigation, mission planning, and control of these vehicles.

ArduPilot is developed and maintained by a community of volunteers and has gained significant popularity in the hobbyist, academic, and professional UAV industries. It is based on the Arduino framework, making it compatible with a wide range of hardware platforms.

Some key features of ArduPilot include:

1. **Flight modes**: ArduPilot supports various flight modes, such as manual control, stabilize, altitude hold, position hold, loiter, auto, and many more. These modes allow the vehicle to perform different types of tasks, from simple manual control to fully autonomous missions.

2. **Waypoint navigation**: ArduPilot enables the definition of GPS waypoints to create autonomous missions. Vehicles can be programmed to follow specific paths, perform actions at waypoints, and execute complex missions.

3. **Sensor integration**: ArduPilot integrates with a variety of sensors, including GPS, inertial measurement units (IMUs), barometers, compasses, and more. These sensors provide crucial data for accurate positioning, attitude estimation, and control.

4. **Telemetry and ground control**: ArduPilot supports real-time telemetry transmission, allowing the vehicle to send data back to a ground control station. Ground control software, such as Mission Planner or QGroundControl, provides a user interface to monitor and control the vehicle during missions.

5. **Fail-safe and safety features**: ArduPilot includes robust fail-safe mechanisms to handle unexpected situations or loss of control signals. It supports features like return to launch (RTL), geofencing, battery monitoring, and emergency procedures to ensure the safe operation of autonomous vehicles.

ArduPilot is compatible with a wide range of hardware platforms, including popular flight controllers like Pixhawk, Cube, and Navio. It supports various communication protocols such as MAVLink, which allows integration with other software and systems.

3.2 What is DroneKit

DroneKit is an open-source software development kit (SDK) that facilitates the creation of applications for unmanned aerial vehicles (UAVs) or drones. It offers a comprehensive set of tools, libraries, and APIs that simplify communication with drones, access to telemetry data, control of flight operations, and the creation of autonomous missions. With support for multiple programming languages and connectivity options, DroneKit enables developers to build robust and efficient drone applications. By abstracting low-level commands and providing high-level abstractions, DroneKit streamlines the development process and empowers developers to unlock the full potential of drones for various use cases.

1.Features

- **Vehicle State:** DroneKit enables you to access various attributes of the drone's state, such as its GPS position, altitude, battery level, and attitude.
- **Vehicle Control:** You can send commands to control the drone's behavior, such as changing flight modes, setting waypoints, or adjusting its speed and direction.
- **Mission Planning:** DroneKit allows you to define missions by specifying a sequence of waypoints that the drone should follow. You can also define complex behaviors using mission commands.
- **Telemetry Streaming:** You can receive real-time telemetry data from the drone, including sensor readings, GPS information, and battery status.
- **Events and Callbacks:** DroneKit provides event-driven programming capabilities, allowing you to define callbacks that get triggered when specific events occur, such as changes in the drone's state or incoming messages.

2.Installation

Install DroneKit-Python, using pip, the Python package installer, by running the command:

pip install dronekit

3.Communication and Connectivity

- DroneKit supports various connection options, including serial, UDP, TCP, and MAVProxy.
- **Serial:** For direct serial connection, you need to specify the serial port and baud rate.
- **UDP:** To connect over a network, you need to provide the IP address and port number of the drone.
- **TCP:** Similar to UDP, you need to provide the IP address and port number, but using TCP as the transport protocol.
- **MAVProxy:** MAVProxy is a command-line ground control station that can act as a bridge between DroneKit and the drone. It allows you to connect to the drone using MAVLink over serial, UDP, or TCP.

4.Vehicle Attributes and Telemetry

- DroneKit provides access to a wide range of vehicle attributes and telemetry data.
- You can access information such as GPS position, altitude, attitude (roll, pitch, and yaw), battery level, speed, and more.
- These attributes are updated in real-time, allowing you to monitor the drone's state during flight.

5.Vehicle Control

- DroneKit allows developers to create autonomous missions for drones, specifying waypoints, commands, and actions.
- Mission planning involves defining a sequence of waypoints that the drone should follow, as well as specifying other parameters and actions at each waypoint.
- Developers can create complex mission behaviors by incorporating various mission commands, such as changing speed, triggering camera capture, or executing custom actions.
- DroneKit provides functions and methods to upload, execute, and monitor the progress of missions, enabling developers to create sophisticated autonomous flight behaviors.

6.Mission Planning and Execution

- DroneKit allows you to plan and execute autonomous missions for your drone.
- Missions can be defined using waypoints, which specify the GPS coordinates and other parameters for the drone to follow.
- DroneKit provides functions and methods to create, upload, and execute missions, as well as monitor their progress and handle mission events.

7.Events and Callbacks

- DroneKit supports event-driven programming, allowing you to define callbacks that get triggered when specific events occur.
- You can register callbacks to handle events such as changes in vehicle state, reception of specific messages, mission updates, and more.
- These callbacks provide a way to respond to events in real-time and perform actions accordingly.

8.Error Handling and Safety

- Implement appropriate error handling and safety measures to ensure the safe operation of the rover.
- Handle exceptions, timeouts, and connection failures in your code.
- Incorporate safety checks, such as obstacle detection, collision avoidance, and emergency stop mechanisms, specific to your rover's hardware and control system.

9.Testing and Debugging

- Use MAVProxy as a ground control station to test and debug your rover's integration with DroneKit.
- MAVProxy provides a command-line interface and visualization tools to monitor and control the rover's behavior.

3.3 Mission Planner and its key features

Mission Planner is an open-source ground control station (GCS) software developed primarily for ArduPilot-based autonomous vehicles, including drones (UAVs), rovers, and boats. It provides a comprehensive set of tools and features to plan, monitor, and control autonomous missions.
Here are some key features and functionalities of Mission Planner:
1. **Mission Planning**: Mission Planner allows you to plan and design autonomous missions for your vehicle. You can define waypoints on a map, set altitude and speed parameters, specify actions at waypoints, and create complex mission sequences.

2. **Real-time Telemetry**: Mission Planner provides real-time telemetry data from your vehicle, including GPS position, altitude, attitude, battery voltage, sensor readings, and more. This information is displayed in customizable gauges, graphs, and maps.

3. **Parameter Configuration**: You can access and modify various parameters and settings of your vehicle's autopilot system through Mission Planner. This includes motor and servo configuration, flight modes, failsafe settings, sensor calibration, and more.

4. **Vehicle Control**: Mission Planner allows you to take manual control of your vehicle, overriding the autonomous functions if needed. You can control the vehicle's throttle, yaw, pitch, and roll using joystick input or keyboard commands.

6. **Simulation**: Mission Planner features a simulation mode that allows you to test and simulate missions without flying your vehicle. You can simulate the vehicle's behavior, verify mission plans, and assess the feasibility of your mission before executing it in the real world.

Fig 3.1 : Mission planner

3.4 Working on Mission planner

To use Mission Planner, we need to download and install it. Once installed, connect the rover vehicle to the computer using a telemetry module or USB cable. The vehicle is powered on and the connection is established. Before starting a mission, we need to configure certain parameters specific to the rover, such as motor and steering settings, sensor calibrations, and navigation modes. Mission Planner provides a graphical interface to access and modify these configurations.

Mission planning: With Mission Planner, we can plan and design autonomous missions for rover. This includes setting waypoints, defining actions at waypoints (such as stopping, taking pictures, or performing certain tasks), and specifying mission parameters. Once your mission is planned, we can upload it to the rover through Mission Planner. The mission is then stored on the rover's onboard autopilot system.

Monitoring and control: During the mission execution, Mission Planner allows to monitor the rover's real-time telemetry data, such as GPS position, speed, battery status, and sensor readings. We can visualize the rover's path on a map and monitor its progress.

Mission execution: Once the mission is uploaded and the rover is ready, we initiate the autonomous mission through Mission Planner. The rover follows the predefined waypoints and execute actions as specified in the mission plan.

3.5 Simulation of Rover

Mission Planner is used to simulate the behavior of a rover vehicle before executing missions in the real world. The simulation feature in Mission Planner allows you to test and validate mission plans, assess the performance of your rover, and make necessary adjustments without the need for physical deployment. Here's how we simulate a rover using Mission Planner:

1. **Connect to Simulation**: Launch Mission Planner and establish a connection to the simulation environment. In the top right corner of the Mission Planner window, click on the "Connect" button and select "Connect to Simulation." This will initiate the simulation mode.

2. **Set Up Simulation Vehicle**: In the simulation mode, configure a simulated vehicle that represents rover. Go to the "Initial Setup" tab and select "Simulator" from the left-hand menu. Here specify the characteristics of the simulated rover, including size, weight, motor parameters, and sensor information.

3. **Define Mission Plan**: Go to the "Flight Plan" tab and use the map interface to define waypoints, set altitudes, and configure actions at waypoints. This mission plan will be executed by the simulated rover.

4. **Start Simulation**: After defining the mission plan, click on the "Write WPs" button to upload the mission to the simulated rover. Once uploaded, click on the "Auto" button to start the simulation. The simulated rover will follow the defined waypoints and execute the specified actions.

5. **Monitor Telemetry and Performance**: During the simulation, Mission Planner provides real-time telemetry data and performance information about the simulated rover. We can monitor its position on the map, track altitude, speed, battery voltage, sensor readings, and other relevant parameters.

6. **Analyze Simulation Results**: After the simulation is completed, review the simulation logs and data collected by Mission Planner. These logs provide insights into the rover's behavior, mission execution, and any anomalies that occurred during the simulation.

Fig 3.2 : Simulation of Rover

3.6 Navigation of Rover with GPS

```
from dronekit import connect, VehicleMode,LocationGlobalRelative,APIException
import time
import math
import argparse

def connectMyCopter():



    vehicle = connect('com11',baud=57600,wait_ready=True)

    return vehicle

def arm():
    while vehicle.is_armable!=True:
        print("Waiting for vehicle to become armable.")
        time.sleep(1)
    print("Vehicle is now armable")

    vehicle.mode = VehicleMode("GUIDED")

    while vehicle.mode!='GUIDED':
        print("Waiting for drone to enter GUIDED flight mode")
        time.sleep(1)
    print("Vehicle now in GUIDED MODE. Have fun!!")

    vehicle.armed = True
    while vehicle.armed==False:
        print("Waiting for vehicle to become armed.")
        time.sleep(1)
    print("Vehicle is now armed.")

    return None

def get_distance_meters(targetLocation,currentLocation):
    dLat=targetLocation.lat - currentLocation.lat
    dLon=targetLocation.lon - currentLocation.lon

    return math.sqrt((dLon*dLon)+(dLat*dLat))*1.113195e5

def goto(targetLocation):
```

```
    distanceToTargetLocation =
get_distance_meters(targetLocation,vehicle.location.global_relative_frame)

    vehicle.simple_goto(targetLocation)

    while vehicle.mode.name=="GUIDED":
        currentDistance =
get_distance_meters(targetLocation,vehicle.location.global_relative_frame)
        if currentDistance<distanceToTargetLocation*.05:
            print("Reached target waypoint.")
            time.sleep(2)
            break
        time.sleep(1)
    return None
##########MAIN EXECUTABLE###########

wp1 = LocationGlobalRelative(17.3971356,78.4899206,0)


vehicle = connectMyCopter()

vehicle.parameters['WP_SPEED']=2

arm()


goto(wp1)

vehicle.mode = VehicleMode("RTL")


while vehicle.mode!='RTL':
    print("Waiting for drone to enter RTL flight mode")
    time.sleep(1)
print("Vehicle now in RTL mode. Driving home.")
```

CHAPTER - 4

4. ROS AND MAVROS PLATFORM SETUP

4.1 Robot Operating System(ROS)

Robot Operating System or simply ROS is a framework that is used by hundreds of Companies and techies of various fields all across the globe in the field of Robotics and Automation. It provides a painless entry point for nonprofessionals in the field of programming Robots.ROS stands for Robot Operating System. It is an open-source framework and set of tools widely used in the field of robotics for developing robot applications.

ROS provides a flexible and modular architecture that helps in building software for controlling robots, simulating robot behavior, and integrating different hardware components. Although ROS is not an operating system (OS) but a set of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post, and multiplex sensor data, control, state, planning, actuator, and other messages. Despite the importance of reactivity and low latency in robot control, ROS is not a real-time operating system (RTOS)

Overall, ROS provides a powerful set of tools and libraries that simplify the development of robot software. It enables modular, distributed, and collaborative development, making it a popular choice for robotics applications.ROS offers a standard software platform to developers across industries that will carry them from research

4.2 ROS and underlying operating system

ROS depends on the underlying Operating System. ROS demands a lot of functionality from the operating system. On top of that ROS must be freely available to a large population, otherwise, a large population may not be able to access it. Much of the popularity of ROS is due to its open nature and easy availability to the mass population. It also needs an operating system that is open source so the operating system and ROS can be modified as per the requirements of the application.

Proprietary Operating Systems such as Windows 10 and Mac OS X may put certain limitations on how we can use them. This may lead to rigidity in the development process, which will not be ideal for an industry standard like ROS. Hence, most people prefer to run ROS on Linux particularly Debian and Ubuntu since ROS has very good support with Debian-based operating systems, especially Ubuntu. That doesn't mean that ROS can't be run with Mac OS X or Windows 10 for that matter. But the support is limited and people may find themselves in a tough situation with little help from the community.

Once we have all the code ready and running, we need to test our code so that we can make changes if necessary. Doing this on a real robot will be costly and may lead to a waste of time in setting up the robot every time. Hence we use robotic simulations for that. The most popular simulator to work with ROS is Gazebo. It has good community support, it is open source and it is easier to deploy robots on it. One is bound to run into issues with Linux especially when working with ROS, and a good knowledge of Linux will be helpful to avert/fix these issues.

- A Meta Operating system has a huge amount of functionality, so much that it cannot be classified as a framework or a cluster of libraries but not so much that it can be categorized as an operating system either. It provides functionalities of both Operating Systems as well as frameworks but not fully hence, it cannot be classified as either e.g, it does not provide the core functionalities that an operating system is supposed to provide but provides APIs.

- RViz is a 3D Visualization tool for ROS. It is one of the most popular tools for visualization. It takes in a topic as input and visualizes that based on the message type being published. It lets us see the environment from the perspective of the robot.

4.3 Installation of ROS

- Settingup sources.list

  Setup your computer to accept software from packages.ros.org.

  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

- Setting up keys

  sudo apt install curl # if you haven't already installed curl

  curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -

- Installation

  First, make sure Debian package index is up-to-date:

  sudo apt update

- Desktop-Full Install: (Recommended) :

  sudo apt install ros-melodic-desktop-full

- Environment setup

  It's convenient if the ROS environment variables are automatically added to bash session every time a new shell is launched:

  echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

  source ~/.bashrc

- To change the environment of the current shell, instead of the above :

  source /opt/ros/melodic/setup.bash

- If zsh is used instead of bash run the following commands to set up shell:

  echo "source /opt/ros/melodic/setup.zsh" >> ~/.zshrc

  source ~/.zshrc

- Dependencies for building packages

  sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential

- Initialize rosdep

  Before you can use many ROS tools, you will need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS. If you have not yet installed rosdep, do so as follows.

  sudo apt install python-rosdep

- With the following, initialize rosdep.

  sudo rosdep init
  rosdep update

4.4 MAVROS

MAVROS stands for "MAVLink to ROS" and is a popular middleware for integrating the MAVLink communication protocol with the Robot Operating System (ROS). MAVLink is a lightweight communication protocol specifically designed for unmanned systems, such as drones and ground vehicles, to exchange messages and commands. ROS, on the other hand, is a flexible framework for writing robot software. It provides a collection of tools, libraries, and conventions to facilitate the development of robotic systems. ROS allows you to build modular and distributed robot applications, making it easier to integrate various components and sensors.

MAVROS acts as a bridge between MAVLink and ROS, allowing ROS nodes to communicate with MAVLink-enabled vehicles. It provides a set of ROS packages that establish the communication link and enable interaction with the vehicle, such as sending commands, receiving telemetry data, and accessing sensor information. With MAVROS, you can develop ROS-based applications for controlling drones or other MAVLink-enabled vehicles. It simplifies the process of interfacing with the vehicle by abstracting the MAVLink protocol and providing high-level ROS interfaces and APIs. This allows you to focus on developing higher-level algorithms and behaviors using the familiar ROS ecosystem. Overall, MAVROS is a powerful tool for integrating MAVLink-enabled vehicles with ROS, enabling developers to leverage the extensive capabilities of both platforms in building robust and sophisticated robotic systems.

Mavros handles the translation of ROS messages to MAVLink messages and vice versa, facilitating communication between the two systems. It supports features like vehicle control, telemetry, mission ROS - planning, and sensor data access. Developers can leverage Mavros to build complex UAV applications using ROS, taking advantage of ROS's extensive library of packages and tools for tasks such as perception, planning, and control. It simplifies the integration of ROS and MAVLink-based autopilots, enabling rapid prototyping and development of advanced robotic systems.Mavros enables the development of autonomous flight capabilities using the ROS ecosystem. It allows you to send high-level commands, waypoints, and mission plans to the autopilot system, making it easier to develop and test autonomous UAV behaviors.

4.5 Installation of MAVROS

Currently we are working with ROS "Melodic on Ubuntu 18.04:

1. First we need to download the ubuntu_sim_ros_melodic.sh script in a bash shell:htt

wget https://raw.githubusercontent.com/PX4/Devguide/master/build_scripts/ubuntu_sim_ros_melodic.sh

2.To run the script we need to execute following command

bash ubuntu_sim_ros_melodic.sh

We installed MAVROS from source or binary. We used the Source installation.

1. First we created Catkin workspace

   mkdir -p ~/catkin_ws/src

   cd ~/catkin_ws

   catkin init

   wstool init src

   wstool init src
2. We have installed python tools like rosinstall and catkin tools

   sudo apt-get install python-catkin-tools python-rosinstall-generator -y

3 .We initialised source space as we are using wstool for first time

wstool init ~/catkin_ws/src

4.Next we installed MAVLINK

We use the Kinetic reference for all ROS distros as it's not distro-specific and up to date

rosinstall_generator --rosdistro kinetic mavlink | tee /tmp/mavros.rosinstall

5.We installed MAVROS from latest source

rosinstall_generator --upstream-development mavros | tee -a /tmp/mavros.rosinstall

6.Then we created Workspace and deps

wstool merge -t src /tmp/mavros.rosinstall

wstool update -t src -j4

rosdep install --from-paths src --ignore-src -y

7.Geographical Libraries have been installed

./src/mavros/mavros/scripts/install_geographiclib_datasets.sh

8.We built the source

catkin build

9.Setup.bash has been used from workspace

source devel/setup.bash

CHAPTER - 5

5. 6 DOF MOBILE ROBOTIC ARM CONTROL

5.1 What is 6DOF Robotic Arm

A robotic arm is a type of mechanical arm, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robot. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement.[1][2] The links of the manipulator can be considered to form a kinematic chain. The terminus of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand.

We are using 6DOF Robotic Arm.A 6-DOF robotic arm refers to a robotic arm that has six degrees of freedom. "DOF" stands for "degrees of freedom," which represents the number of independent movements or axes that a robotic arm can achieve. In the case of a 6-DOF robotic arm, it means that the arm can move in six different ways or along six different axes. These axes are typically referred to as pitch, yaw, roll, and translation along the X, Y, and Z axes.

With a 6-DOF robotic arm, it becomes possible to achieve intricate movements and perform complex operations. The arm can rotate, bend, and extend in multiple ways, simulating human-like motions. This level of dexterity allows the robotic arm to handle delicate objects, perform intricate assembly tasks, and adapt to changing environments.In industries where precision is paramount, a 6-DOF robotic arm shines. It can execute movements with accuracy, ensuring consistent and reliable results. Whether it's performing surgical procedures, welding components with precise alignment, or handling fragile materials, the arm's ability to precisely control its six degrees of freedom allows for meticulous and controlled operations.

Hence, the utilization of a 6-DOF robotic arm offers a wide range of benefits. Its six degrees of freedom enable it to perform complex movements, achieve high precision, and adapt to various tasks and environments. With its versatility, accuracy, and dexterity, the 6-DOF robotic arm plays a crucial role in enhancing productivity, improving efficiency, and expanding automation capabilities across industries.

5.2 Configuration of 6 DOF Robotic Arm

The configuration of a 6-DOF (six degrees of freedom) robotic arm refers to the arrangement and placement of its joints and links, allowing it to achieve a wide range of movements and orientations. The specific configuration of a 6-DOF robotic arm can vary depending on the application and design requirements.

A common configuration for a 6-DOF robotic arm is a serial or chain-like structure, consisting of several interconnected links and joints. The arm typically comprises a series of revolute and prismatic joints, which provide rotational and linear motion, respectively. These joints are strategically positioned along the arm to enable the desired degrees of freedom.

Starting from the base, the first joint allows the arm to rotate horizontally, providing the base rotation or azimuth movement. This joint is responsible for orienting the arm's entire structure in the desired direction.

The subsequent joints, often referred to as the shoulder, elbow, and wrist joints, contribute to the remaining degrees of freedom. The shoulder joint allows the arm to move vertically or perform pitch motion. The elbow joint facilitates the bending or rotation of the arm, enabling yaw motion. The wrist joint is responsible for additional flexibility, providing pitch, yaw, and roll movements to the arm.

The links connecting these joints determine the arm's reach and flexibility. Each link has a specific length and geometry, affecting the overall range of motion. The lengths of the links are carefully chosen to optimize the arm's workspace and ensure that it can access various positions and orientations.

To achieve precise and coordinated movements, the robotic arm's configuration requires careful planning and kinematic analysis. The control system must accurately calculate the joint angles and positions required to achieve a specific end effector position or trajectory.

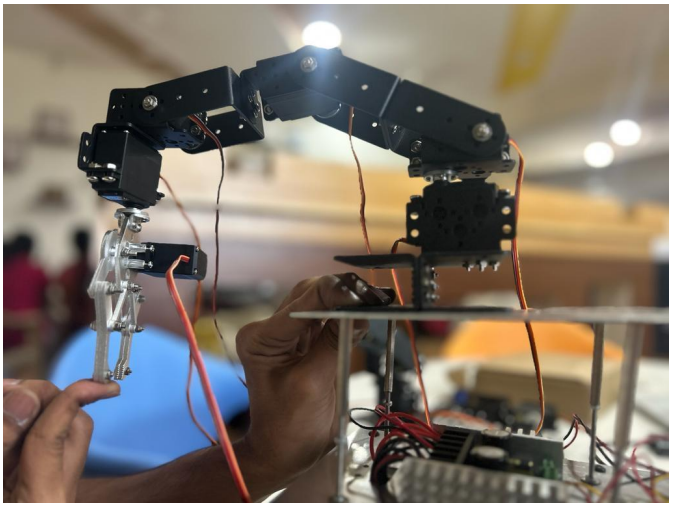


Fig 5.1 : 6DOF Robotic arm

5.3 MOVEIT

MoveIt is a powerful open-source software framework that revolutionizes the field of robotics by providing advanced motion planning and manipulation capabilities. With its extensive libraries and tools, MoveIt enables developers to tackle complex robotic tasks with ease.

One of the key features of MoveIt is its motion planning capability. Robots equipped with MoveIt can generate collision-free trajectories to reach desired positions and orientations in their environment. By taking into account the robot's kinematics, environmental obstacles, and constraints, MoveIt ensures that robots can navigate safely and efficiently.

MoveIt also excels in manipulation tasks. Whether it's picking and placing objects, grasping, or manipulating complex items, MoveIt offers comprehensive support. Through intuitive interfaces, developers can define end-effectors, specify grasps, and perform manipulation planning. This allows robots to interact with their surroundings and perform intricate tasks with precision.

Another noteworthy aspect of MoveIt is its integration with perception systems. By combining perception and planning, robots can perceive their environment, understand object properties, and make informed decisions. MoveIt seamlessly interfaces with various perception technologies, empowering robots to leverage visual information and enhance their capabilities.

In summary, MoveIt is a game-changer in the robotics field. Its motion planning, manipulation, and perception integration capabilities empower robots to perform complex tasks autonomously and interact with their environment effectively. With its open-source nature and thriving community, MoveIt paves the way for the future of robotics, revolutionizing industries and pushing the boundaries of what robots can achieve.

5.4 Installation of MOVEIT

- After installing ROS we made sure all packages are updated
  rosdep update
  sudo apt-get update
  sudo apt-get dist-upgrade

- Install cantkin ROS build system
  sudo apt-get install ros-melodic-catkin python-catkin-tools

- We installed MOVEIT from pre-built libraries
  sudo apt install ros-melodic-moveit

- Then we need to setup Catkin workspace
  mkdir -p ~/ws_moveit/src

- We installed from Debian any package dependencies not already in your workspace:
  cd ~/ws_moveit/src

  rosdep install -y --from-paths . --ignore-src --rosdistro melodic

- Next command will configure your catkin workspace
  cd ~/ws_moveit
  catkin config --extend /opt/ros/${ROS_DISTRO} --cmake-args
  -DCMAKE_BUILD_TYPE=Release
  catkin build

- Source the catkin workspace
  source ~/ws_moveit/devel/setup.bash

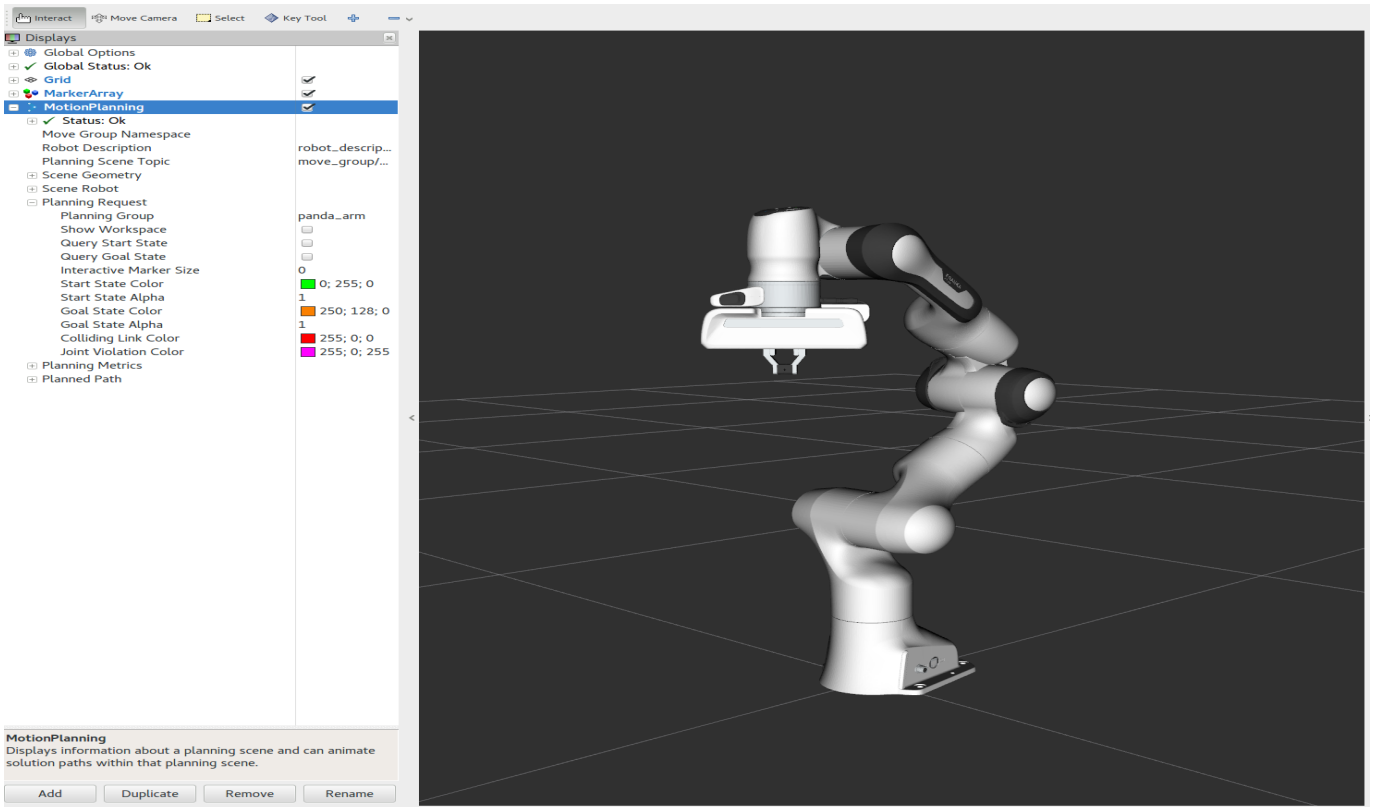


Fig 5.2 : MoveIt

5.5 Code Implementation

The below code is implemented by using PyQt5-based application that controls servos with the PCA9685 PWM driver. It includes a custom QThread subclass called ServoControlThread that handles servo control based on key presses, and a MainWindow class that sets up the GUI window and handles key events.

To run the code, we need to have the required dependencies installed, including pyqt5 and smbus.The I2C connection is given to the PCA9685 board with the correct address (0x40 in this case). The PCA9685 should be connected to the servo motors.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout
from PyQt5.QtCore import Qt, QThread, QTimer
```

```
import time
import smbus


# PCA9685 Registers
MODE1 = 0x00
PRESCALE = 0xFE
LED0_ON_L = 0x06

# PCA9685 Constants
I2C_BUS = 1  # I2C bus number
PCA9685_ADDRESS = 0x40  # I2C address of PCA9685
FREQUENCY = 50  # PWM frequency (Hz)
SERVO_MIN = 500  # Min pulse length for servos (0 degrees)
SERVO_MAX = 2500  # Max pulse length for servos (180 degrees)
SERVO_RANGE = SERVO_MAX - SERVO_MIN  # Pulse length range for servos

class ServoControlThread(QThread):
    u_angle = 40
    d_angle = 135
    l_angle = 80
    r_angle = 80
    o_angle = 60

    def __init__(self):
        super().__init__()
        self.bus = smbus.SMBus(I2C_BUS)
        self.set_pwm_frequency(self.bus, PCA9685_ADDRESS, FREQUENCY)
        self.moving_up = False
        self.moving_down = False
        self.moving_left = False
        self.moving_right = False
        self.moving_default = False
        self.open_grip = False
        self.close_grip = False

    def set_pwm_frequency(self, bus, address, frequency):
        prescale_val = int((25000000 / (4096 * frequency)) - 1)
        bus.write_byte_data(address, MODE1, 0x10)  # Sleep mode
        bus.write_byte_data(address, PRESCALE, prescale_val)
        bus.write_byte_data(address, MODE1, 0x00)  # Wake up

    def set_servo_angle(self, bus, address, channel, angle):
        pulse_width = SERVO_MIN + (float(angle) / 180.0) * SERVO_RANGE
```

```
        pulse_width_value = int(pulse_width * 4096 / 20000)  # Convert to 12-bit
value
        # Set the PWM pulse for the servo motor
        bus.write_byte_data(address, LED0_ON_L + 4 * channel, 0x00)
        bus.write_byte_data(address, LED0_ON_L + 4 * channel + 1, 0x00)
        bus.write_byte_data(address, LED0_ON_L + 4 * channel + 2,
pulse_width_value & 0xFF)
        bus.write_byte_data(address, LED0_ON_L + 4 * channel + 3,
pulse_width_value >> 8)

    def default_pos(self):
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 0, 60)
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 5, 80)
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 4, 40)
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 3, 120)
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 2, 40)
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 1, 90)

    def move_Up(self):
        self.u_angle += 1
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 4, self.u_angle)  # Move
up

    def move_Down(self):
        self.u_angle -= 1
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 4, self.u_angle)  # Move
down

    def move_Left(self):
        self.l_angle += 1
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 5, self.l_angle)  # Move
left

    def move_Right(self):
        self.l_angle -= 1
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 5, self.l_angle)  # Move
right

    def gripper_Open(self):
        self.set_servo_angle(self.bus, PCA9685_ADDRESS, 0, 60)
        self.o_angle = 60  # Open gripper

    def gripper_Close(self):
        if self.o_angle < 115:
            self.o_angle += 5
```

```
            self.set_servo_angle(self.bus, PCA9685_ADDRESS, 0, self.o_angle)  #
Close gripper

    def run(self):
        while True:
            if self.moving_up:
                self.move_Up()
            if self.moving_down:
                self.move_Down()
            if self.moving_left:
                self.move_Left()
            if self.moving_right:
                self.move_Right()
            if self.moving_default:
                self.default_pos()
            if self.open_grip:
                self.gripper_Open()
            if self.close_grip:
                self.gripper_Close()

            time.sleep(0.01)


class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.servo_thread = ServoControlThread()
        self.init_ui()

    def init_ui(self):
        self.setWindowTitle("Servo Control")
        self.setGeometry(100, 100, 300, 200)

        # Set layout
        layout = QVBoxLayout()
        self.setLayout(layout)

    def keyPressEvent(self, event):
        if event.key() == Qt.Key_W:
            self.servo_thread.moving_up = True
        elif event.key() == Qt.Key_S:
            self.servo_thread.moving_down = True
        elif event.key() == Qt.Key_A:
            self.servo_thread.moving_left = True
        elif event.key() == Qt.Key_D:
```

```
            self.servo_thread.moving_right = True
        elif event.key() == Qt.Key_P:
            self.servo_thread.moving_default = True
        elif event.key() == Qt.Key_O:
            self.servo_thread.open_grip = True
        elif event.key() == Qt.Key_C:
            self.servo_thread.close_grip = True

    def keyReleaseEvent(self, event):
        if event.key() == Qt.Key_W:
            self.servo_thread.moving_up = False
        elif event.key() == Qt.Key_S:
            self.servo_thread.moving_down = False
        elif event.key() == Qt.Key_A:
            self.servo_thread.moving_left = False
        elif event.key() == Qt.Key_D:
            self.servo_thread.moving_right = False
        elif event.key() == Qt.Key_P:
            self.servo_thread.moving_default = False
        elif event.key() == Qt.Key_O:
            self.servo_thread.open_grip = False
        elif event.key() == Qt.Key_C:
            self.servo_thread.close_grip = False

    def closeEvent(self, event):
        self.servo_thread.quit()
        self.servo_thread.wait()
        event.accept()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    window.servo_thread.start()
    sys.exit(app.exec_())
```

On running the code, a window will open showing the GUI. Use the following keys to control the servos:

W: Move the servo up

S: Move the servo down

A: Move the servo left

D: Move the servo right

P: Move the servos to their default positions

O: Open the gripper

C: Close the gripper

Press and hold the keys to control the corresponding servo movement. Release the keys to stop the movement.

CHAPTER - 6

6. YOLOv5 BASED OBJECT DETECTION

6.1 What is YOLOV5

YOLOv5 is a state-of-the-art real-time object detection algorithm. It is an evolution of the YOLO (You Only Look Once) family of models and was developed by Ultralytics. YOLOv5 was introduced in May 2020 and aims to improve upon the previous versions by offering better accuracy and efficiency.

The main idea behind YOLOv5 is to detect objects in an image by dividing the image into a grid and predicting bounding boxes and class labels for objects within each grid cell. YOLOv5 follows a single-shot detection approach, meaning it performs both object localization and classification in a single pass through the network.

6.2 Why YOLOv5

Running YOLOv5 on the NVIDIA Jetson Nano can be beneficial for several reasons:

1. On-device inference: The Jetson Nano is a powerful embedded AI computing device that provides the capability to perform real-time inference directly on the device. By running YOLOv5 on the Jetson Nano, you can achieve low-latency object detection without relying on a remote server or cloud infrastructure.

2. Optimized for AI workloads: The Jetson Nano is specifically designed for AI and deep learning applications. It features an NVIDIA GPU with CUDA cores, which can accelerate the computations required for running YOLOv5. The Jetson Nano also comes with libraries like cuDNN and TensorRT, which further optimize neural network operations for enhanced performance.

3. Energy-efficient inference: The Jetson Nano is built with power efficiency in mind, allowing for efficient inference on edge devices. YOLOv5 on the Jetson Nano leverages the device's hardware acceleration capabilities to perform energy-efficient object detection.

4. Compact and portable: The Jetson Nano is a compact and portable device, making it suitable for deployment in various scenarios, including robotics, drones, and IoT applications. By running YOLOv5 on the Jetson Nano, you can enable real-time object detection in these edge computing environments.

5. Development and prototyping: The Jetson Nano provides a convenient platform for developing and prototyping AI applications. By running YOLOv5 on the Jetson Nano, you can experiment with object detection on a small-scale device and iterate quickly on your models and algorithms.

Overall, running YOLOv5 on the Jetson Nano enables efficient, on-device object detection for a range of AI applications, particularly in resource-constrained and edge computing environments.

6.3 Custom Dataset

Step 1: Installing requirements

Here we clone the YOLOv5 repository from GitHub then change the current working directory to the cloned repository. Install the required dependencies and then install the Roboflow library. It also imports necessary modules such as torch and IPython.display for displaying images.

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images

print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else
'CPU'})")
```

Step 2: Assemble the dataset

In order to train our custom model, we used Roboflow and assembled a dataset of representative images with bounding box annotations around the objects that we want to detect. We need our dataset to be in YOLOv5 format.

The dataset used for this project consists of 922 images trained with their labels. 708 images were used for training, 144 for validation and 70 for testing purposes.

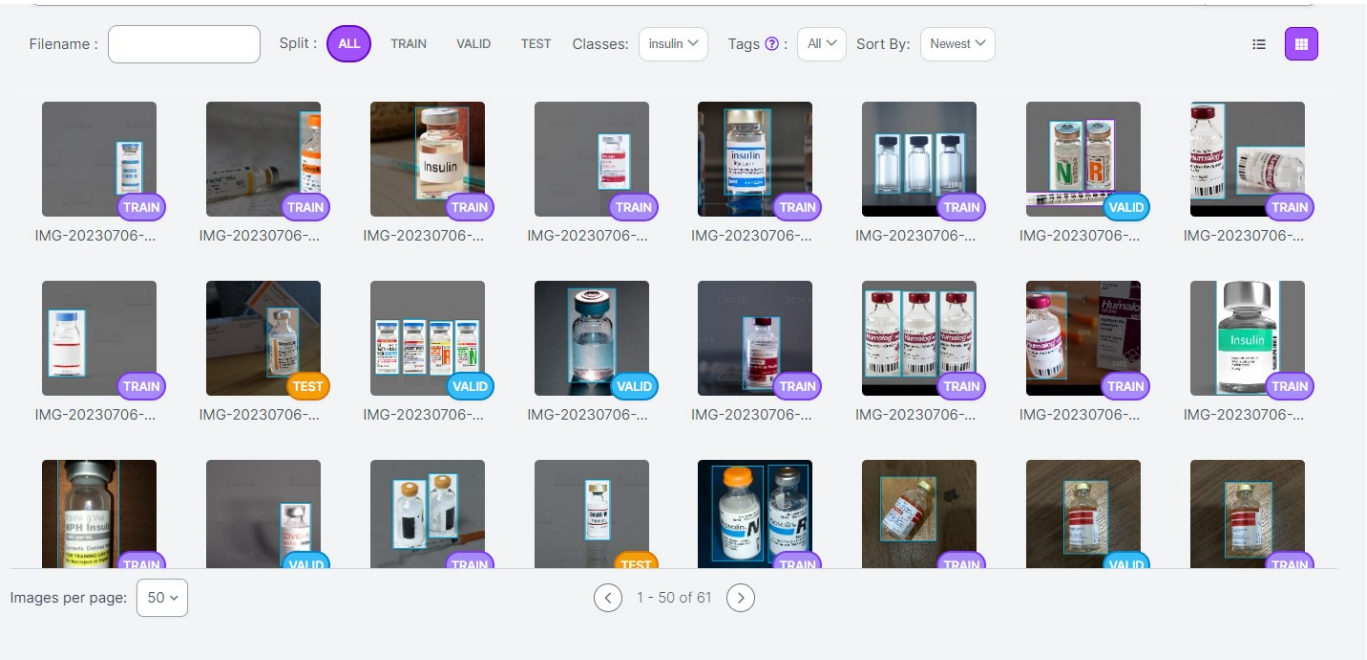


Fig 6.1 : Custom dataset on Roboflow

Step 3: Setting up the environment and importing the dataset from Roboflow

Set the environment variable DATASET_DIRECTORY to the directory path "/content/datasets". It then installs the Roboflow library using pip.

The next few lines import the Roboflow class from the Roboflow library, create an instance of Roboflow by providing an API key, and retrieve a specific project and dataset version from the Roboflow workspace using the project and version IDs.

Finally, the code downloads the dataset associated with the specified project and version using the YOLOv5 format.

```
# set up environment
os.environ["DATASET_DIRECTORY"] = "/content/datasets"
```

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="5OqL0lwRMWaNf2EiQy4X")
project = rf.workspace("anjali-dulam-gwekj").project("waste-detection-ps")
dataset = project.version(3).download("yolov5")
```

Step 4: Train Our Custom YOLOv5 model

Here, we are pass a number of arguments:

img: define input image size

batch: determine batch size

epochs: define the number of training epochs. (Note: often, 3000+ are common here!)

data: Our dataset locaiton is saved in the dataset.location

weights: specify a path to weights to start transfer learning from. Here we choose the generic COCO pretrained checkpoint.

cache: cache images for faster training

```
!python train.py --img 416 --batch 16 --epochs 150 --data
dataset.location}/data.yaml --weights yolov5s.pt --cache
```

The output of the following:

```
train: weights=yolov5s.pt, cfg=, data=/content/datasets/waste-detection-ps-3/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=150, batch_size=16, imgsz=416, rect=False, resume=
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v7.0-192-g459dd49 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 🚀 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 25.5MB/s]
Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
100% 14.1M/14.1M [00:00<00:00, 18.3MB/s]

Overriding model.yaml nc=80 with nc=7

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12          [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16          [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
```

```
Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
0/149      1.7G     0.1119    0.02142    0.06059         29        416: 100% 18/18 [00:07<00:00,  2.32it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:01<00:00,  1.54it/s]
             all         81         95    0.00291      0.648    0.00725    0.00184

Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
1/149      1.85G    0.08624    0.02565    0.05639         46        416: 100% 18/18 [00:02<00:00,  8.11it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:00<00:00,  4.99it/s]
             all         81         95    0.00373       0.84     0.0627     0.0183

Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
2/149      1.85G    0.06902    0.02487    0.05333         31        416: 100% 18/18 [00:03<00:00,  5.27it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:00<00:00,  4.94it/s]
             all         81         95      0.668      0.215      0.252      0.103

Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
3/149      1.85G    0.06582    0.02306    0.04951         37        416: 100% 18/18 [00:02<00:00,  7.77it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:00<00:00,  5.32it/s]
             all         81         95      0.445      0.323      0.235     0.0759

Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
4/149      1.85G    0.05983    0.02101    0.04661         32        416: 100% 18/18 [00:02<00:00,  8.46it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:00<00:00,  5.22it/s]
             all         81         95      0.237      0.488       0.28      0.128

Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
5/149      1.85G    0.06254    0.01987    0.04281         26        416: 100% 18/18 [00:02<00:00,  8.16it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:00<00:00,  4.28it/s]
             all         81         95      0.449      0.482      0.335      0.164

Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
6/149      1.85G    0.05541    0.01883    0.03948         33        416: 100% 18/18 [00:03<00:00,  5.10it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:00<00:00,  5.84it/s]
```

```
Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:01<00:00,  1.84it/s]
                 all         81         95       0.93      0.898      0.926      0.694
              bottle         81         14          1      0.947      0.995      0.784
              cotton         81         15      0.776      0.733      0.681       0.43
              gloves         81         13      0.963          1      0.995      0.871
             insulin         81         15        0.9          1      0.987      0.751
                mask         81         15          1      0.788      0.953      0.722
             syringe         81         14          1      0.818      0.875      0.519
             tablets         81          9      0.873          1      0.995      0.782
Results saved to runs/train/exp
```

Fig 6.2: output of training with custom dataset

Step 5: Evaluate Custom YOLOv5 Detector Performance

The code sets up and launches TensorBoard to visualize logs stored in the "runs" directory, which typically contains training and validation logs for machine learning models.

```
# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```

Step 6: Run Inference With Trained Weights

Run inference with a pre trained checkpoint on contents of test/images folder downloaded from Roboflow.

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf 0.1
--source {dataset.location}/test/images
```

The below code displays the inference results on all test images in a directory using the YOLOv5 model. It utilizes the glob module to retrieve all JPEG images within the specified directory and the IPython.display module to display the images in the Jupyter Notebook or IPython environment

```
#display inference on ALL test images

import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/runs/detect/exp/*.jpg'): #assuming
JPG
    display(Image(filename=imageName))
    print("\n")
```

Here we see the inference results (bounding boxes, labels, etc.) of the YOLOv5 model overlaid on each test image in the specified directory.

Fig 6.3 : Inference results of YOLO v5

Step 7: Conclusion and Downloading best.pt file

By executing this code, the best.pt file will be downloaded to your local machine, allowing you to save it for future use or further analysis.

```
#export your model's weights for future use
from google.colab import files
files.download('./runs/train/exp/weights/best.pt')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

6.4 Code implementation

```
import torch
import numpy as np
import cv2
import pafy
import time
class ObjectDetection:
    """Class implements Yolo5 model to make inferences on a youtube video using
OpenCV """
    cap = cv2.VideoCapture(0)
    def _init_(self):
        """
        Initializes the class with youtube url and output file.
        :param url: Has to be as youtube URL,on which prediction is made.
        :param out_file: A valid output file name.
        """
        self.model = self.load_model()
        self.classes = self.model.names
        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
        print("\n\nDevice Used:", self.device)

    def load_model(self):
        """
        Loads Yolo5 model from pytorch hub.
        :return: Trained Pytorch model.
        """
        model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
        return model

    def score_frame(self, frame):
```

```
        """
        Takes a single frame as input, and scores the frame using yolo5 model.
        :param frame: input frame in numpy/list/tuple format.
        :return: Labels and Coordinates of objects detected by model in the
frame."""
        self.model.to(self.device)
        frame = [frame]
        results = self.model(frame)
        labels, cord = results.xyxyn[0][:, -1], results.xyxyn[0][:, :-1]
        return labels, cord
    def class_to_label(self, x):
        """
        For a given label value, return corresponding string label.
        :param x: numeric label
        :return: corresponding string label
        """"
        return self.classes[int(x)]
    def plot_boxes(self, results, frame):


        labels, cord = results
        n = len(labels)
        x_shape, y_shape = frame.shape[1], frame.shape[0]
        area = 0  # Initialize area variable
        center = None  # Initialize center point variable
        for i in range(n):
            row = cord[i]
            if row[4] >= 0.2:
                x1, y1, x2, y2 = int(row[0] * x_shape), int(row[1] * y_shape),
int(row[2] * x_shape), int(row[3] * y_shape)
                bgr = (0, 255, 0)
                cv2.rectangle(frame, (x1, y1), (x2, y2), bgr, 2)
                cv2.putText(frame, self.class_to_label(labels[i]), (x1, y1),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, bgr, 2)

                if self.class_to_label(labels[i]) == 'person':
                    l = x2 - x1
                    b = y2 - y1
                    area = l * b  # Calculate area for the 'person' class
                    center = (x1 + l // 2, y1 + b // 2)  # Calculate the center
point of the bounding box
                    # Display the center of the bounding box
                    cv2.circle(frame, center, 3, (0, 0, 255), -1)

        return frame, area, center
```

```
    def _call_(self):
        """
        This function is called when class is executed, it runs the loop to read
the video frame by frame,
        and write the output into a new file.
        :return: void
        """
        while cap.isOpened():
            start_time = time.perf_counter()
            ret, frame = cap.read()
            if not ret:
                break
            results = self.score_frame(frame)
            frame, area, center = self.plot_boxes(results, frame)  # Receive the
area value and center point
            end_time = time.perf_counter()
            fps = 1 / np.round(end_time - start_time, 3)
            cv2.putText(frame, f'FPS: {int(fps)}', (20, 70),
cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 255, 0), 2)
            # Calculate the center of the screen
            height, width, _ = frame.shape
            center_x = width // 2
            center_y = height // 2
            cv2.circle(frame, (center_x, center_y), 3, (255, 0, 0), -1)
            # print(center_x, center_y)
            cv2.imshow("img", frame)
            if area != 0:  # Check if an area value is available
                return area, center
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
# Create a new object and execute.
detection = ObjectDetection()
# area, center = detection()
while True:
    area, center = detection()
    print(area)
    print(center)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
```

The code implements a class named `ObjectDetection` that utilizes the YOLOv5 model for object detection on a video stream captured from a camera using OpenCV.

- The class has a class variable named `cap`, which is an instance of `cv2.VideoCapture(0)`. This captures video from the default camera (index 0) . The class has an initialization method named `__init__`, which initializes the class.

- The method loads the YOLOv5 model by calling the `load_model` method and assigns the loaded model to the `model` attribute.

- It sets the `classes` attribute to the names of the classes in the model.

- The `device` attribute is set to 'cuda' if a CUDA-enabled GPU is available, otherwise 'cpu'.

- The `load_model` method loads the YOLOv5 model from the "ultralytics/yolov5" repository on PyTorch Hub. It specifically loads the "yolov5s" variant of the model, which is a smaller version optimized for speed.

- The `score_frame` method takes a single frame as input and passes it through the YOLOv5 model for inference. It returns the labels and coordinates of the detected objects in the frame.

- The `class_to_label` method maps a numeric label to its corresponding string label using the `classes` attribute.

- The `plot_boxes` method receives the results from the YOLOv5 inference and the input frame. It iterates over the detected objects, draws bounding boxes and labels on the frame, and performs additional operations for the 'person' class. It calculates the area of the 'person' bounding box and the center point.

- The `__call__` method is called when the class is executed. It runs a loop to read frames from the video stream captured by `cap`, performs object detection on each frame, and displays the output with bounding boxes, labels, and FPS information. It also calculates the center of the screen and displays it as a circle. The loop breaks if 'q' is pressed.

- The code then creates an instance of the `ObjectDetection` class named `detection` and continuously calls the `__call__` method to perform object detection on the video stream. It prints the area and center of the detected 'person' object. The loop breaks if 'q' is pressed.

Finally, `cv2.destroyAllWindows()` is called to close all OpenCV windows when the program ends.

CHAPTER - 7

7. VOICE RECOGNITION MODULE

7.1 Seed Studio XIAO nRF52840(Sense)

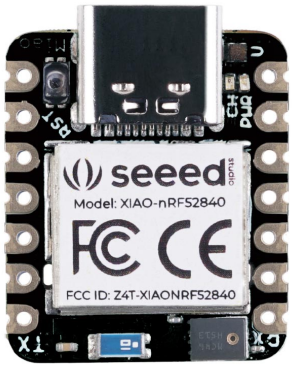


Fig 7.1 : Seeed Studio XIAO nRF52840

Seeed Studio XIAO nRF52840 is equipped with a powerful Nordic nRF52840 MCU which integrates Bluetooth 5.0 connectivity. Meanwhile, it has a small and exquisite form-factor which can be used for wearable devices and Internet of Things(IoT) projects. The single-sided surface-mountable design and the onboard Bluetooth antenna can greatly facilitate the rapid deployment of IoT projects.

In addition, there is an advanced version of this board, Seeed Studio XIAO nRF52840 Sense. It is integrated with two extra onboard sensors. One of them is a Pulse Density Modulation (PDM) Digital Microphone. It can receive audio data in real-time which allows it to be used for audio recognition. The other one is a 6-axis Inertial Measurement Unit (IMU), this IMU can be very useful in TinyML projects like gesture recognition. These onboard sensors provide a great convenience for users while the board is ultra-small.

The first thing to note is that the Near Field Communication (NFC) interface is functional on the board. Secondly, there is a tiny reset button on the side of the Type-C interface. On the other side, there is a 3-in-one LED (User LED) along with a Charge LED to indicate the charging status when a battery is connected. There are 11 digital I/O that can be used as PWM pins and 6 analog I/O that can be used as ADC pins. It supports all three common serial interfaces such as UART, I2C, and SPI.

7.2 Libraries required for Seeed Studio XIAO nRF52840(Sense)

Seeed Studio XIAO nRF52840 assembles many functions in one tiny board and sometimes may not perform the best of them. Hence, Seeed has published two Arduino libraries to let it maximum the power of each function.

- It is recommanded to use the Seeed nRF52 Boards library if you want to apply Bluetooth function and "Low Energy Cost Function".
- It is recommanded to use the Seeed nRF52 mbed-enabled Boards library if you want to use it in embedded Machine Learning Applications or apply "IMU & PDM advanced function".
- Both libraries support very well when it comes to the basic usage, such as LED, Digital, Analog, Serial, I2C, SPI.

7.3 Hardware overview

The hardware overview of the Seeed Studio Xiao NRF52840 development board includes the following key components and features:

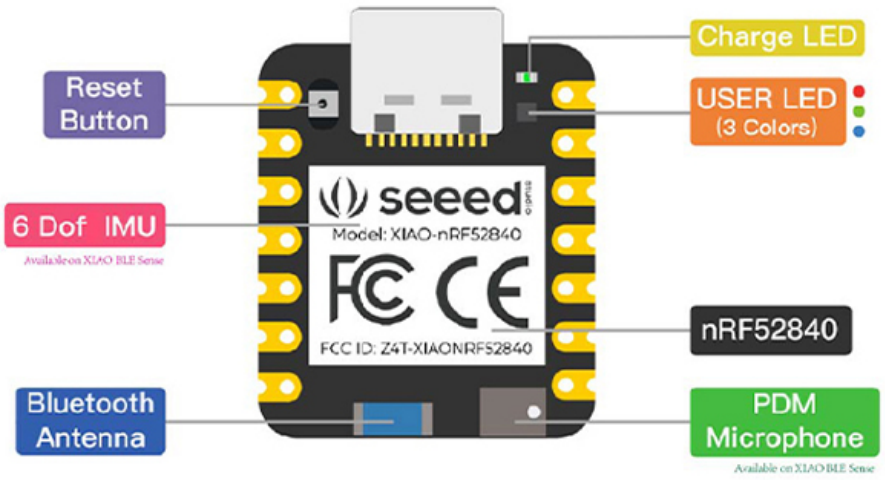


Fig 7.2 : Hardware Overview

1.Nordic nRF52840 SoC: The board is based on the Nordic Semiconductor nRF52840 system-on-chip (SoC). It features an Arm Cortex-M4F processor running at 64MHz, 1MB flash memory, and 256KB RAM. The SoC provides Bluetooth Low Energy (BLE) connectivity and supports other wireless protocols like Zigbee, Thread, and ANT.

2.GPIO Pins: The Xiao NRF52840 offers a variety of GPIO (General-Purpose Input/Output) pins for connecting external components. It provides digital pins, analog pins, and PWM (Pulse Width Modulation) pins for flexible interfacing.

3.Communication Interfaces:

- SPI: The board features Serial Peripheral Interface (SPI) for high-speed data transfer with other SPI-compatible devices.
- I2C: It provides an Inter-Integrated Circuit (I2C) interface for communication with I2C-compatible sensors, modules, and peripherals.
- UART: The board has Universal Asynchronous Receiver-Transmitter (UART) pins for serial communication with other devices.
- USB: It supports USB connectivity for data transfer and power supply.

4.Built-in PCB Antenna: The Xiao NRF52840 includes a built-in PCB antenna for wireless communication. It ensures reliable and convenient Bluetooth and other wireless connections without the need for an external antenna.

5.Power Supply:

Micro USB Port: The board can be powered through the micro USB port, allowing it to draw power from a computer or USB power source.

External Power Source: It also supports powering via an external power source, allowing flexibility in different application scenarios.

Power Management: The board incorporates power management features to optimize power consumption and extend battery life.

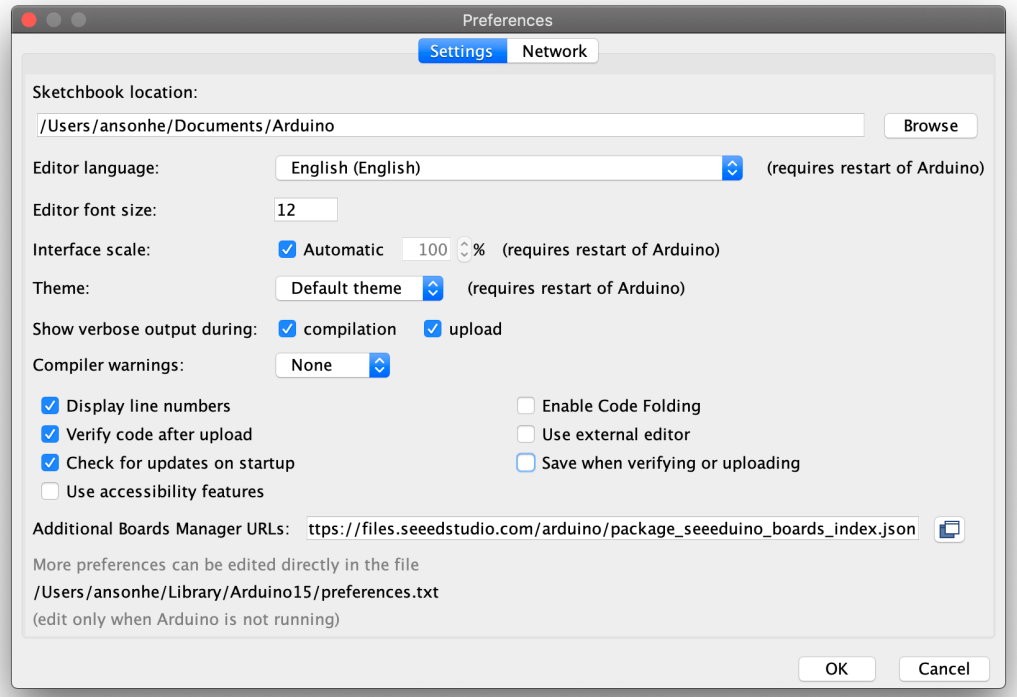
7.4 Software Setup

Step 1: Download and Install the latest version of Arduino IDE according to your operating system

Step 2: Launch the Arduino application

Step 3: Add Seeed Studio XIAO nRF52840 (Sense) board package to your Arduino IDE

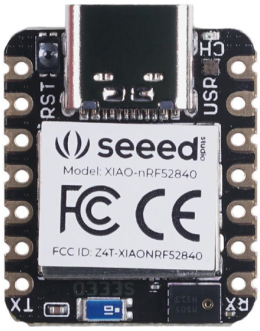
Navigate to File > Preferences, and fill "Additional Boards Manager URLs" with the url below: https://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json



Navigate to Tools > Board > Boards Manager..., type the keyword "seeed nrf52" in the search box, select the latest version of the board you want, and install it. You can install both.

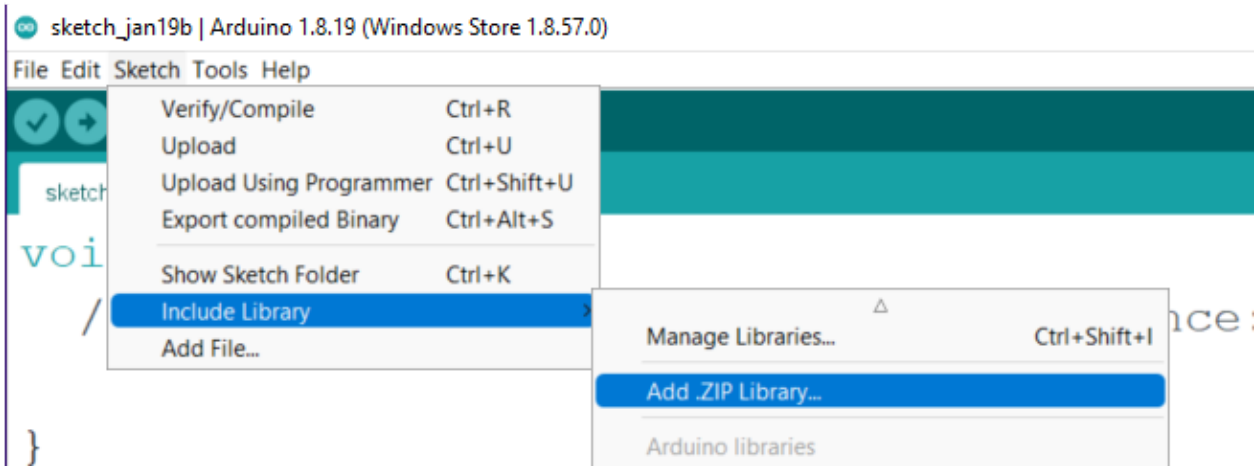
Step 4: Select your board and port

7.5 Speech Recognition on Seeed Studio XIAO nRF52840 Sense



Software Setup

- **Step 1:**Download tflite-micro-arduino-examples library as a zip file
- **Step 2:**Open Arduino IDE, navigate to Sketch > Include Library > Add .ZIP Library... and open the downloaded zip file



7.6 Train data and generate TensorFlow Lite models

- **Step 1:**Open this Python notebook

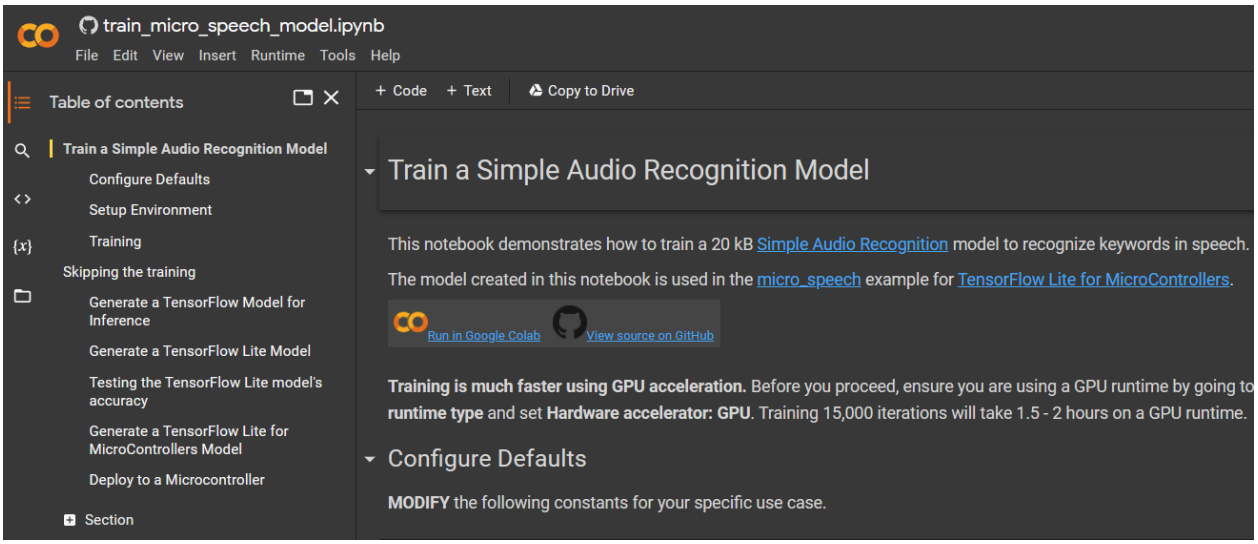


Fig 7.3 : Train_micro_Speech_model python notebook

By default, it will load this dataset which can recognize the words: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "bed", "bird", "cat", "dog", "happy", "house", "marvin", "sheila", "tree", "wow"

- **Step 2:**Under Configure Defaults column, change the WANTED_WORDS parameter according to the words that you want the model to recognize. You can choose from: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "bed", "bird", "cat", "dog", "happy", "house", "marvin", "sheila", "tree", "wow"

- **Step 3:**Navigate to Runtime > Run all to run all the code cells

- **Step 4:**Once all the code cells are executed, navigate to the files tab on the left corner and you will find a new model.cc file generated under the models folder

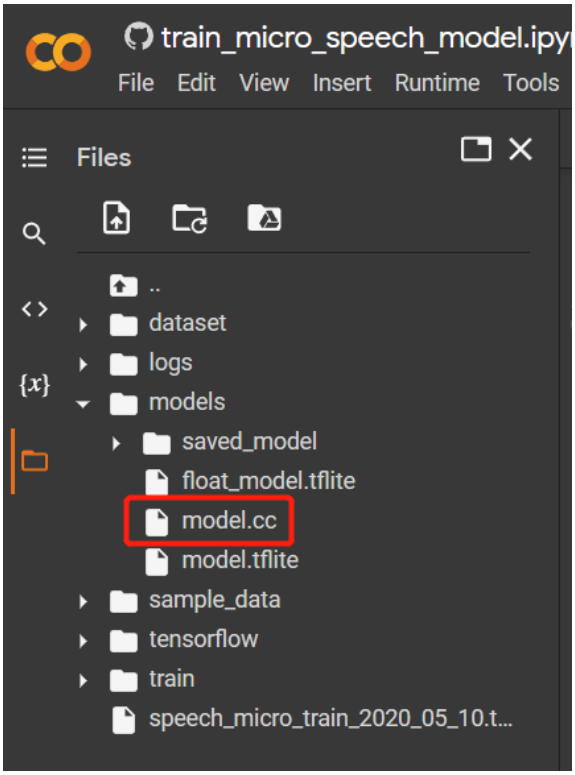


Fig 7.4 : model.cc file

- **Step 6:** click Download to download the file to your PC

Inference

Downloaded TensorFlow Lite model file (model.cc) to recognize different words using the microphone on the Seeed Studio XIAO nRF52840 Sense.

- **Step 1:**Navigate to the library path of tflite-micro-arduino-examples Library (normally under Documents > Arduino > libraries > tflite-micro-arduino-examples), visit examples > micro_speech and open micro_features_model.cpp
- **Step 2:**Replace the values under const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = { with the new values from the model.cc file

```
const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
    0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
    0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
    0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
    0x0b, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f,
    0x72, 0x75, 0x6e, 0x74, 0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73,
    0x69, 0x6f, 0x6e, 0x00, 0x0c, 0x00, 0x00, 0x00, 0xd4, 0x41, 0x00, 0x00,
    0xb4, 0x41, 0x00, 0x00, 0x24, 0x03, 0x00, 0x00, 0xf4, 0x02, 0x00, 0x00,
    0xec, 0x02, 0x00, 0x00, 0xe4, 0x02, 0x00, 0x00, 0xc4, 0x02, 0x00, 0x00,
    0xbc, 0x02, 0x00, 0x00, 0x2c, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00,
    0x1c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x16, 0xbd, 0xff, 0xff,
    0x04, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e,
    0x30, 0x00, 0x00, 0x00, 0x94, 0xba, 0xff, 0xff, 0x98, 0xba, 0xff, 0xff,
    0x32, 0xbd, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x80, 0x02, 0x00, 0x00,
    0xfa, 0xee, 0x28, 0xc4, 0xee, 0xfe, 0xcf, 0x0f, 0x1e, 0xf7, 0x1f, 0x06,
    0x0d, 0xed, 0xe9, 0x83, 0x5c, 0xc9, 0x18, 0xe3, 0xf9, 0x14, 0x28, 0x2a,
    0x09, 0xf2, 0x18, 0x34, 0x62, 0xea, 0xef, 0xd6, 0x36, 0xb7, 0x1e, 0xf7,
    0x3b, 0x22, 0x28, 0x39, 0xc2, 0x9d, 0xf1, 0x07, 0x5e, 0x0b, 0x1e, 0x2c,
```

Fig 7.5 : trained models

- **Step 3:**Change g_model_len according to the value from model.cc.
- **Step 4:** Open micro_features_micro_model_settings.cpp inside micro_speech folder and add all the words that we defined in the training process
- **Step 5:**Open micro_features_micro_model_settings.h inside micro_speech folder and change constexpr int kCategoryCount according to the number of categories defined.
- **Step 6:**Open micro_speech.ino inside micro_speech folder and upload the codes to the Seeed Studio XIAO nRF52840 Sense
- **Step 7:** Open serial monitor window and say out loud the words that we defined before. You will see the serial monitor output the correct words spoken after recognition.

```
Initialization complete
Heard yes (205) @22400ms
Heard yes (212) @29920ms
Heard up (211) @34032ms
Heard yes (203) @45088ms
Heard yes (202) @49792ms
Heard yes (202) @59248ms
Heard up (202) @66912ms
Heard up (204) @69872ms
Heard down (202) @73568ms
Heard no (202) @84400ms
Heard up (201) @89104ms
```

Fig 7.6 : output on serial monitor

7.7 Code implementation

The below Python script uses the DroneKit library, tkinter, and speech_recognition to control a Pixhawk-based rover using a 6WD rover and voice commands. The script connects to the Pixhawk, arms the vehicle, and listens for keyboard and voice commands to control the rover's movement. It also performs object detection using a custom object detection model. It defines movement functions for the rover, handles keyboard and voice control events, and processes voice commands to perform the corresponding actions. The tkinter library is used to create a GUI window, and the script enters a main loop to continuously process events. It's essential to ensure proper hardware setup and possess the required knowledge and safety precautions when operating drones.

```
from dronekit import connect, VehicleMode
import time
import tkinter as tk
import speech_recognition as sr
from custom_object_detection import ObjectDetection

connection_string = 'COM9' # Replace with the serial port of your Pixhawk
baud_rate = 57600 # Replace with the baud rate that your Pixhawk is
configured to use
vehicle = connect(connection_string, baud=baud_rate, wait_ready=False)

# Set the mode of the vehicle to MANUAL
vehicle.mode = VehicleMode("MANUAL")

# Arm the vehicle
vehicle.armed = True

detection = ObjectDetection("C:/Users/chotu/Desktop/ps/best (6).pt")


print("q")

def forward():
   vehicle.channels.overrides['3'] = 1300

def backward():
  vehicle.channels.overrides['3'] = 1800
  time.sleep(1)
```

```
  vehicle.channels.overrides['3'] = 1500
  vehicle.channels.overrides['1'] = 1500


# Turn right
def right():
  vehicle.channels.overrides['3'] = 1500
  vehicle.channels.overrides['1'] = 1800

#left
def left():
  vehicle.channels.overrides['3'] = 1500
  vehicle.channels.overrides['1'] = 1100


def move():
  while True:
    areas , center = detection()
    if center[0] < 315 :
      left()
      vehicle.channels.overrides['3'] = 1500
      vehicle.channels.overrides['1'] = 1500
    elif center[0] > 325 :
      right()
      vehicle.channels.overrides['3'] = 1500
      vehicle.channels.overrides['1'] = 1500
    else:
      forward()
      if areas >= 274000 and areas <= 280000 :
        vehicle.channels.overrides['3'] = 1500
        vehicle.channels.overrides['1'] = 1500
        break

def key(event):
    if event.char == event.keysym:  # Standard keys
        if event.keysym == 'q':
            print("QUIT")
            vehicle.armed = False
            vehicle.close()
            exit()
        elif event.keysym == 'r':
            # Add RTL mode implementation
            pass
    else:  # Non-standard keys
        if event.keysym == 'Up':
            forward()
```

```
        elif event.keysym == 'Down':
            move()
        elif event.keysym == 'Right':
            right()
        elif event.keysym == 'Left':
            left()
        elif event.keysym == 'space':
            print('hi')
            recognize_live_voice_command()


def recognize_live_voice_command():
    # Create a recognizer object
    r = sr.Recognizer()

    # with sr.Microphone(device_index=1) as source:
    # Use the default microphone as the audio source
    with sr.Microphone() as source:
        print("Listening...")

        # Continuously listen for voice commands in real-time
        while True:
            try:
                # Adjust for ambient noise
                r.adjust_for_ambient_noise(source)

                # Listen for the user's voice command
                audio = r.listen(source)
                # Recognize speech using Google Speech Recognition
                command = r.recognize_google(audio)
                print("Voice command:", command)

                # Process the recognized command
                process_voice_command(command)
            except sr.UnknownValueError:
                print("Speech recognition could not understand audio")
            except sr.RequestError as e:
                print("Could not request results from Google Speech
Recognition service; {0}".format(e))

def process_voice_command(command):
    if 'forward' in command:
        forward()
    elif 'backward' in command:
        backward()
    elif 'right' in command:
```

```
        right()
    elif 'left' in command:
        left()
    # Add more voice commands and corresponding actions as needed
    elif 'move' in command:
        move()
    else:
        print("Unrecognized voice command")


print("q")

root = tk.Tk()
print(">> Control the drone with the arrow keys. Press r for RTL mode")
root.bind_all('<Key>', key)
root.mainloop()
```

CHAPTER - 8

8.1 TESTING AND RESULT

During each trial, different types of medical waste objects and multiple instances of the same waste object were placed in front of the robotic arm. In addition, after conducting each trial, new waste objects were then placed in a different position and orientation to test the robustness of the system's waste detection and classification module. Accuracy metrics and performance metrics were calculated to analyze the efficiency of the system.

//IMAGES

8.2 CONCLUSION

Our project successfully developed a comprehensive system aimed at detecting and segregating medical waste from other types of waste using a mobile robotic arm.By leveraging a deep learning-based algorithm, we achieved accurate and efficient waste detection, ensuring precise identification of medical waste items. The mobile robotic arm demonstrated exceptional capabilities in picking up waste object and reducing the risk of cross-contamination.The dedication and collaborative efforts of our team members were instrumental in successfully addressing these challenges and achieving the desired outcomes.The developed system holds significant promise in addressing the critical issue of medical waste management, improving efficiency, and minimizing health hazards associated with improper waste disposal. It has the potential to greatly benefit healthcare facilities, ensuring compliance with waste management regulations while promoting a cleaner and safer environment.With this innovative solution, healthcare facilities can streamline their waste management operations, adhere to regulatory guidelines, and contribute to a safer and cleaner environment. The system's potential for expansion and integration with existing waste management infrastructure opens up opportunities for scalability and wider adoption, making it a promising solution for addressing the pressing issue of medical waste management in the healthcare sector.In conclusion, our project represents a significant step forward in medical waste management through the integration of robotics, deep learning, and voice recognition technologies. We are proud of the accomplishments achieved and believe that our system has the potential to make a positive impact in healthcare settings and beyond.

8.3 BIBLIOGRAPHY:

melodic/Installation/Ubuntu - ROS Wiki

ROS with MAVROS Installation Guide | PX4 User Guide

Getting Started — moveit_tutorials Melodic documentation (ros.org)

https://youtu.be/x0ThXHbtqCQ

https://www.researchgate.net/publication/361321595_A_ROS-based_Voice_Controlled_Robotic_Arm_for_Automatic_Segregation_of_Medical_Waste_Using_YOLOv3

https://youtu.be/SLDJAOEjVt4

https://wiki.seeedstudio.com/XIAO_BLE/