

VISVESVARAYA TECHNOLOGICAL UNIVERSITY BELAGAVI



Project Report on

“Autonomous Voice Controlled Arm Robot - AVCAR”

Submitted in the partial fulfillment for the requirements of the degree of

**BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING**

Submitted By

Chaitra Sri Naidu 1BY20CS044

S Guru Prasad 1BY20CS154

Sai Harshit B 1BY20CS162

Vigneshwaran P S 1BY20CS215

Under the guidance of

Mrs. Ambika G N

Assistant Professor
Department of CSE,
BMSIT&M



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.

2023-2024

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI**

**BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA, BENGALURU – 560064**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the Project work entitled “**Autonomous Voice Controlled Arm Robot - AVCAR**” is a bonafide work carried out by **Chaitra Sri Naidu Chenaboina (1BY20CS044)**, **S Guru Prasad (1BY20CS154)**, **Sai Harshit B (1BY20CS162)**, **Vigneshwaran P S (1BY20CS215)**, in partial fulfillment for the award of **Bachelor of Engineering Degree in Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2023-2024. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report. The project report has been approved as it satisfies the academic requirements in respect of project work for B.E. Degree.

Signature of the Guide

Mrs. Ambika G N
Assistant Professor
Dept. of CSE, BMSIT&M

Signature of the HOD

Dr. Thippeswamy G
Professor & HOD,
Dept. of CSE, BMSIT&M

Signature of Principal

Dr. Sanjay H. A
Principal, BMSIT&M

External VIVA-VOCE

Name of the Examiners

- 1.
- 2.

Signature with Date

ACKNOWLEDGEMENT

We are pleased to present this project report upon its successful completion. This project would not have been possible without the guidance, assistance, and suggestions of many individuals. We would like to express our deep sense of gratitude to each and every person who has contributed to making this project a success.

First and foremost, we extend our heartfelt thanks to **Dr. Sanjay H. A, Principal, BMS Institute of Technology & Management**, for his constant encouragement and inspiration in undertaking this project.

We also express our sincere gratitude to **Dr. Thippeswamy G, Head of the Department, Computer Science and Engineering, BMS Institute of Technology & Management**, for his unwavering support and motivation throughout this endeavor.

We are grateful to our project guide, **Prof. Ambika G N, Assistant Professor, Department of Computer Science and Engineering**, for their invaluable encouragement and guidance throughout the project.

We extend our heartfelt thanks to our project coordinators, **Dr. Vidya R Pai and Dr. Arunakumari B. N, Assistant Professor, Department of Computer Science and Engineering**, for their constant support and valuable advice during the project.

Special thanks are due to all the staff members of the Computer Science and Engineering Department for their help and cooperation.

Finally, we would like to express our gratitude to our parents and friends for their unwavering encouragement and support throughout the duration of this project.

By,
Chaitra Sri Naidu
S Guru Prasad
Sai Harshit B
Vigneshwaran P S

DECLARATION

We **Chaitra Sri Naidu (1BY20CS044)**, **S Guru Prasad (1BY20CS154)**, **Sai Harshit (1BY20CS162)**, **Vigneshwaran P S (1BY20CS215)** students of Eight semester B. E, in the Department of Computer Science and Engineering, BMS Institute of Technology and Management, Bengaluru declare that the project work entitled “**Autonomous Voice Controlled Arm Robot - AVCAR**” has been carried out by us and submitted in partial fulfilment of the course requirements for the award of degree in **Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belagavi** during the academic year 2023 - 2024. The matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Chaitra Sri Naidu (1BY20CS044)

S Guru Prasad (1BY20CS154)

Sai Harshit B (1BY20CS162)

Vigneshwaran P S (1BY20CS154)

ABSTRACT

Introducing an assistive robot arm, operated via voice commands and powered by Plansys2 within the ROS2 framework. This innovative integration revolutionizes assistive applications, enabling seamless control through natural language interaction. Plansys2 enhances capabilities with minimal adjustments, ensuring adaptability and ease of use. Users can effortlessly customize functionalities through intuitive voice commands, empowering diverse applications from healthcare to manufacturing. Optimizing resource allocation, this system promises improved performance and ease of integration, marking a significant advancement in assistive robotics technology.

TABLE OF CONTENTS

Chapter 1:Introduction.....	1
1.1 Background.....	2
1.2 Literature Survey.....	3
1.3 Motivation.....	5
1.4 Problem Statement.....	6
1.5 Aim and Objective.....	7
1.6 Scope.....	7
1.7 Challenges.....	8
1.8 Organization of thesis.....	9
Chapter 2: Overview.....	10
2.1 Input.....	10
2.2 Tools.....	10
2.3 Procedure.....	10
2.4 Output.....	11
Chapter 3: Requirement Specification.....	12
3.1 Functional Requirements.....	12
3.2 Non-Functional Requirements.....	13
3.3 Software Requirements.....	13
3.4 Hardware Requirements.....	14
Chapter 4: Detailed Design.....	15
4.1 System Architecture.....	15
4.2 Data Flow Design.....	17
4.3 Sequence Diagram.....	19

4.4 Use Case Diagram.....	20
Chapter 5: Implementation.....	21
5.1 Voice Control	21
5.2 Perception Module.....	26
5.3 Task Planning Module	31
5.4 Arm Control and Simulation Module.....	33
5.5 ROS Controllers and plugins	42
Chapter 6: Testing.....	51
6.1 Testing Robotic Arm Functionality	52
6.2 Object Detection Testing with YOLOv8.....	53
6.3 Testing Speech Recognition.....	54
6.4 Integration Testing	55
Chapter 7: Results and Discussions	56
7.1 YOLOv8 object detection results	56
7.2 ROS arm controllers grip and pick results	57
Chapter 8: Conclusion and Future Enhancement.....	60
8.1 Conclusion	60
8.2 Future Enhancement.....	60
References.....	62

LIST OF FIGURES

Figure 4.1.1 Structure of the Backend Logic of the System.....	15
Figure 4.2.1.1 High level control flow of the system	17
Figure 4.2.2.1 Complete control flow of the system.	18
Figure 4.3.1 Sequence diagram.....	19
Figure 4.4.1 Use Case Diagram	20
Figure 5.1.1 Voice Control Subsystem	21
Figure 5.1.4.1 Speechmatics Sub Process	23
Figure 5.1.4.2 Speechmatics Audio Transcription.....	24
Figure 5.1.4.3 Saving Transcription Results	25
Figure 5.2.1 YOLO v8 Lite Architecture	26
Figure 5.2.2 High-Level Architecture of Perception Subsystem	27
Figure 5.2.2.1 YOLO Detection	29
Figure 5.3.1 High-Level Diagram of Plansys2 Subsystem	31
Figure 5.4.1.1 Robot Link and Joints.....	35
Figure 5.4.1.2 Franka Emika Panda Arm Robot	38
Figure 5.4.1.3 Simulated Environment	38
Figure 5.4.2.1 High-Level Architecture of the Arm Controller Subsystem	39
Figure 5.5.3.1 Code snippet of openni_kinect plugin	48
Figure 5.5.3.2 Code snippet of gazebo_grasp plugin	49
Figure 5.5.3.3 Code snippet of force-torque sensor plugin	50
Figure 7.1.1 YOLO training loss curve.....	56
Figure 7.1.2 YOLO coke detection	57
Figure 7.2.1 Plan Generated by PlanSys2.....	57
Figure 7.2.2 Initial Position of the Arm	58
Figure 7.2.3 Attaching Object	58

Figure 7.2.4 Picking Object from Position.....59

Figure 7.2.5 Placing Object to Position.....59

LIST OF TABLES

Table 1.2 Literature Survey	4
Table 6.1.1 Testing Robotic Arm Functionality	52
Table 6.2.1 Object Detection Testing with YOLOv8	53
Table 6.3.1 Testing Speech Recognition	54
Table 6.4.1 Integration Testing	55

CHAPTER 1

INTRODUCTION

PlanSys2, a ROS2-based planning system framework, can greatly enhance the capabilities of assistive robots, making them more effective in supporting elderly and bedridden patients who lack caretakers. By leveraging PlanSys2, developers can more easily create and integrate advanced planning features into their robotic systems, enabling them to better assist with daily tasks and improve the quality of life for these individuals.

One of the key challenges faced by elderly and handicapped individuals without caretakers is difficulty performing daily tasks such as cooking, cleaning, and personal care

This can lead to an unsafe living environment and negatively impact their overall health. For example, bedridden patients may struggle to fetch objects that are out of reach. PlanSys2 can help address these issues by allowing developers to create robots with advanced manipulation and planning capabilities. These robots can assist with tasks like feeding, drinking, and adjusting blankets or pillows, reducing the physical strain on patients and caregivers alike.

Another significant challenge for this population is limited mobility within their homes due to physical disabilities or age-related conditions. PlanSys2-enabled robots can be designed to provide mobility assistance, helping individuals move around their homes safely and maintain their independence.

Furthermore, these robots can be programmed to monitor the environment and alert caregivers or emergency services if necessary, providing an additional layer of safety and support.

Financial challenges are also common among elderly and handicapped individuals without caretakers, as they may struggle to pay for housekeepers and access healthcare services. Assistive robots powered by PlanSys2 can help alleviate some of these financial burdens by reducing the need for constant human assistance. Additionally, these robots can be designed to facilitate remote communication with healthcare providers, ensuring that individuals receive the medical attention they need.

In conclusion, PlanSys2 is a powerful tool for enhancing the capabilities of assistive robots, particularly in the context of supporting elderly and bedridden patients without caretakers. By enabling developers to more easily create robots with advanced manipulation, planning, and monitoring features, PlanSys2 can help address the daily challenges faced by this vulnerable population, ultimately improving their quality of life and overall well-being.

1.1 Background

The implementation of an Autonomous Robotic Arm with Voice Control holds immense promise for enhancing the quality of life for elders and bedridden patients, particularly in scenarios where caregivers are unavailable. The primary aim is to develop a robotic arm capable of autonomously assisting with essential daily activities, such as retrieving items from nearby surfaces. By integrating a planning system, this technology seeks to empower patients to maintain their independence and reduce their dependence on human assistance, a crucial aspect for those living in solitary conditions or lacking regular caregiving support.

The core functionality of the robotic arm hinges on a sophisticated combination of speech recognition, object detection, and ROS2 integration, facilitating precise and efficient movements. Leveraging AI and machine learning algorithms, the arm can dynamically adapt and learn new tasks over time, offering a tailored and personalized caregiving experience to patients. This innovative approach has the potential to revolutionize elderly and bedridden patient care, affording them greater autonomy and significantly improving their overall quality of life.

1.2 Literature Survey

Sl. No	Year	Authors	Title	Objective	Key contribution
1.	2016	Syed Faiz Ahmed, Mohd Norzali Haji Mohd, Mahfuzah Mustafa, Salmiah Ahmad	<i>“Modelling and Simulation of a Moving Robotic Arm Mounted on Wheelchair”</i>	This work is to implement a robotic arm that slides over a track mounted over a wheelchair is proposed to attain much wider workspace, at no extra complexity of arm design	This work represents the Kinematic equations, Dh-parameters and dynamic modeling & simulation of a robotic arm moving along a path built onto a wheelchair including the 3D workspace of the system
2.	2017	J. A. Palacios-Hernandez, E. D. Saenz-Ibarra, J. C. Avila-Vilchis, A. L. Hernandez-Hernandez	<i>“Design and Implementation of a Robotic Arm using ROS and MoveIt”</i>	The main purpose of this work, is to manipulate and grip objects to transport them from one location to another using Robot Operating System (ROS).	Interfacing and manipulation of a robotic arm with a motion planner (MoveIt) and a 3D viewer (RViz) is implemented.
3.	2021	Yuki Nakayama, Yusuke Kajiwarra, Kazuto Yokoyama, Hiroyuki Hamada	<i>“A Self-Feeding Assistive Robotic Arm for People with Physical Disabilities of the Extremities”</i>	To develop a self-feeding assistive robotic arm for people with severe disabilities	Capable of detecting the target object and localize it in the scene. Can move towards the target.
4.	2019	R Melo, TP de Araújo, AA Saraiva, JVM Sousa, NMF Ferreira	<i>“Computer Vision System with Deep Learning for Robotic Arm Control”</i>	To provide a pattern recognition system, which can be used in classification applications for hand gestures for control of robotic arms.	The system uses feature matching for extracting objects from a scene, edge detector and deep learning. The use of extraction of the region of interest and edges segmentation reduces the amount of processing required to recognize signals, thus speeding up the recognition process.

5.	2021	S Du, F Wang, G Zhou, J Li, L Yang, D Wang	<i>“Vision-Based Robotic Manipulation of Intelligent Wheelchair with Human Computer Shared Control”</i>	To provide a novel robotic manipulation fashion combining computer vision and brain-computer interface (BCI) based on human-computer shared control.	This system uses a pipeline of asynchronous brain-computer interface based on steady-state visual evoked potential (SSVEP), vision detection with deep network and robotic manipulation of URS robot. electroencephalography (EEG) signals to find the pattern and judge them.
6.	2020	RA John, S Varghese, ST Shaji, KM Sagayam	<i>“Assistive Device for Physically Challenged Persons Using Voice Controlled Intelligent Robotic Arm”</i>	The prototype is aimed to integrate the voice recognition system into the robotic arm and move the robotic arm accordingly with the voice commands given, hold the object and leave it.	The system uses Google speech recognition API to understand human voice commands. And it is interfaced with raspberry pi to process the input and move the arm accordingly.
7.	2021	R Natharani, F Liri, J Samawi, H Lin, K Lee, N Ruppert, K George, A Panangadan	<i>“Voice Controlled Object Grasping Robotic Arm for Visually Impaired Disabled Veterans”</i>	Implemented a low-cost robotic arm, DOBOT Magician, to help visually impaired individuals access essential items from a refrigerator.	The system takes in audio input using natural language, i.e., the system can identify the object using input voice commands and depth sensing camera and a robotic arm that retrieves the item based on the audio and visual input.

Table 1.2: Literature Survey

1.2.1 Limitations of existing system

- Existing solutions rely on more physical interaction because of controllers, buttons etc.. Hence implementing voice inputs make the system easier for physically weak.

- No easy and better interaction for elderly and handicapped/injured.
- Lack of intelligence to act on self.
- Poor detection and interaction system, making it tough for elderly who often forget about names of the objects that they need to pick.
- AI methodologies like object detection, recognition and localization along with usage of state-of-the-art path planning algorithms and planner packages to avoid collisions and effective movement of arms.
- Currently, similar solutions are very tedious to develop and enhance as they require the developer to explicitly code in the sequence for every possible command.

1.3 Motivation

The motivation for the project "Autonomous Voice Controlled Arm Robot" lies in the need to aid bedridden patients and elders to assist in the absence of caretaker. There is often need for a caretaker with an injured/elder/disabled to assist them with daily needs. However, in the absence of the caretaker people try to overdo things and get themselves more injured.

In case of elders, at the very old age their body and mind coordination will not be great. Hence, they find it difficult to perform easy task and be dependent on the caretaker completely.

This has increased the work for a caretaker which affects both mentally and physically for both caretaker and the elders. In hospitals, management won't/won't often allow families/family to be with the patient to assist them, instead appoints people from the hospital which increase the maintenance cost for both the parties.

Middle class people, who live away from their family and elders who are physically weak and disabled cannot afford to hire a caretaker. Of course, the taker should be the one who is completely trusted. This problem has given rise to smart robotic arms which can be fixed on the railing or even to the chair handle which assist the to pick or fetch the objects near them.

The advancements in AI and robotics have led to the development of such Autonomous Voice Controlled Arm Robots that can assist disabled and bedridden patients in performing their daily tasks such as fetching desired objects. With the growing aging population and an increasing number of people with disabilities, there is a need for innovative solutions to support these individuals' independence and quality of life. A smart robotic arm can provide the necessary

assistance to patients with limited mobility, enabling them to perform tasks such as eating, drinking, and taking medications without the need for a caregiver's constant presence.

The use of ROS, speech recognition, and object detection technologies in the development of the robotic arm allows for precise and accurate movements, increasing its reliability and efficiency.

Moreover, robotic arms can perform tasks that are repetitive, heavy, or hazardous for human caregivers, reducing their workload and risk of injury. They can also operate 24/7, providing patients with continuous care and reducing the need for human assistance.

All these development in AI and robotics provides motivation to develop a simple yet effective prototype of the robotic arm that can assist people in the absence of the caretaker.

1.4 Problem Statement

The project "Autonomous Voice Controlled Arm Robot" aims to address the inherent limitations in task execution and sequencing within current robotic systems designed for general assistive purposes, with elder care being one notable application. Presently, these systems require meticulous programming for each task and its variations, resulting in cumbersome development processes and inflexible functionality. This rigidity restricts the adaptability of the robotic arm to cater to diverse user needs and scenarios effectively. By integrating a planning system, the project seeks to transform this paradigm by empowering the robot to analyze commands and autonomously determine optimal task sequences across various assistive contexts. This innovative approach not only enhances the efficiency of the robotic arm but also simplifies its development and facilitates future enhancements. The planning system enables the robot to dynamically adjust its actions based on real-time environmental cues and user preferences, offering a more intuitive and responsive assistive experience. While elder care stands as a significant use case, this technology holds promise in a wide range of applications, including disability support, rehabilitation assistance, and household tasks for individuals with varying needs and abilities.

1.5 Aim and Objectives

The objectives of the project titled "Autonomous Voice Controlled Arm Robot (A.V.C.A.R)" are:

- To design and simulate a robotic arm that can move in 3D space of simulated world.
- To integrate Plansys2, a PDDL based planning tool for autonomous task planning.
- To be make the robot capable of generating the appropriate
- Train the state-of-the art object detection model to detect and localize objects in the scene.
- Implement speech recognition algorithms/models to take user voice commands as input to the system.
- Develop controller scripts to move the arm and grip, pick and place objects in the scene.
- Develop and program scripts to transform coordinates between different frames.
- Tune the controllers to make the arm precise enough to pick correct object and get it without dropping and collision.
- Finally building a pipeline to integrate all the individual components into one system.

1.6 Scope

The scope of the project for Autonomous Voice Controlled Arm Robot includes the following:

- To develop a simulated 2-fingered robotic arm which can move in the 3D space.
- To device, compare and tune the controllers to ensure the proper movement of the arm's joints.
- Implementation and integration of task planning systems and path planners to move the arm to the goal state.
- Training, tuning and integration of object detection and speech recognition models to the system.
- Establishing and maintaining communication among different nodes.
- Ensuring proper communication between ROS nodes and correct transformations of the coordinate frames.

The project will primarily focus on to create a Autonomous Voice Controlled Arm Robot that employs speech recognition and YOLO technology to give people with impairments or limited mobility and weak elders a safe and customized experience. In order to facilitate seamless

operation and communication, the device's functionality should be simple to use, affordable, and able to interact with the user with a simple yet effective interface.

1.7 Challenges

Simulating and manipulation of a robotic arm comes with a greatwith great number of problems and challenges, below are some of the challenges that we faced:

- First and foremost, the design of the robotic arm. To us it took some good amount of time to get a working model of the arm with the camera sensor. We ran into many problems and got stuck at some time. However, we were able to modify a few open source models for the project.
- Motion planner sometimes failed to plan, due to constraints in the scene.
- Controllers were often failing to execute the published commands.
- Coordinate transformation was quite challenging when transforming pose of the object and arm joints between target and source frames.
- The Point Cloud data was corrupted and disoriented because of the bugs in the plugin which affected the frame transformation.
- To train any ML algorithm, we need extensive amount of data. In this project we trained YOLO object detection model for which we had to manually take object images from the simulator, locate and label the images, augment them to maintain model generalization.
- Choosing speech recognition model was challenging task since we need to transcribe the audio input asynchronously. Most of the models out there does transcription in synchronous mode. After finding model, it was challenging to fine tune the recognizer to specific words.

It took us some time to overcome these obstacles, which necessitated a detailed understanding of the underlying technology, a strict design and development methodology, and extensive testing and validation. Additionally, it required cooperation amongst specialists in several fields, including robotics (ROS2), Planning Systems, PDDL, ROS Client libraries, open source models etc. to ensure that all facets of the system are effectively addressed.

1.8 Organization of thesis

The organization of the thesis for this project on Autonomous Voice Controlled Arm Robot will follow a standard structure, which includes the following sections:

- Chapter 1 – Introduction: An outline of the project's goals, drivers, and scope is mentioned in this chapter. The literature survey and the shortcomings in the current system are also included. The difficulties encountered during project construction are also mentioned.
- Chapter 2 – Overview: This chapter discusses the project's overall process, its input and output, and the steps taken. It also includes the tools used to build the project.
- Chapter 3 – Requirements Specification: The project's functional and non-functional needs are discussed in this chapter. Additionally, it covers the software and hardware needed to develop the project.
- Chapter 4 – Detailed Design: The system architecture is a prominent topic of discussion in this chapter. It also discusses the various UML (Unified Modelling Language) diagrams, including use case diagrams and sequence diagrams.
- Chapter 5 – Implementation: In this chapter, we talk about the programming language and the framework used to build the project. Additionally, we go over the algorithms we utilize and why.
- Chapter 6 – Testing: The main areas of discussion in this chapter are the many sorts of testing carried out, such as bounding box validation and object localization for object detection, speech detection and recognition for different audio models and testing controllers on various simulated environments.
- Chapter 7 – Experimental Results: Training and testing results of various AI models with the error graph are depicted. We also go over the outcomes we've gotten, along with screen shots of each input and the resulting output.
- Chapter 8 – Conclusion: This chapter will summarize the key findings of the project and provide a conclusion on the project's success in meeting its objectives.
- References: This section will provide a list of all the references cited in the thesis.

The thesis will be structured in a logical and organized manner to ensure that the reader can follow the development process, understand the rationale behind the choices made, and evaluate the effectiveness of the final system.

CHAPTER 2

OVERVIEW

2.1 Input:

The following is supplied as input to the project.

- User input as speech
- Image from the RGBD camera mounted on the arm
- User choice of object as the input (speech)

2.2 Tools:

The following tools are used to build the project:

- Python - rclpy,
- C++ - rclcpp,
- OpenCV,
- YOLO v8,
- ROS2 Humble,
- Gazebo,
- Rviz,
- TF2_ROS,
- Speechmatics,
- Gemini API,
- Pyttsx.

2.3 Procedure:

- The nodes are launched via launch files.
- The Robot simulation and controller nodes are first set up, followed by the Plansys2 nodes and then the voice and perception nodes.

- The user is allowed to give commands once all the nodes are online and have verified that they are working smoothly.
- If the command is wrong the robot goes to standby position
- Else the command is then parsed into the goal in PDDL syntax
- The robot scans the nearby table for the objects. The information is stored in the environment server and it provides the predicates for the problem file.
- Once the goal and the predicates have been obtained, Plansys2 starts generating the sequence to reach the goal state.
- Plansys2 then executes the sequence via a group of nodes called the action nodes that carry out the step and return back the status of the step, whether it succeeded or failed.
- If the plan succeeds, the arm gets back to its ready position.
- If any of the steps fail, Plansys2 again scans the environment again and replans the sequence of steps to the targeted goal state.
- This procedure can be looped in a command to manipulate different objects but one object can be interacted with only once in a single command. It can be interacted with again in the next command.

2.4 Output:

The Robot properly analyses the command given and generates an appropriate plan to carry out the task, and in case of failure it corrects itself and ultimately achieves its task.

CHAPTER 3

REQUIREMENT SPECIFICATION

3.1 Functional Requirements:

Functional requirements are the specific features and capabilities that the system should have to fulfill its purpose.

Here are some functional requirements for a smart robotic arm in ROS (Robot Operating System) using YOLO (You Only Look Once) object detection and speech recognition:

- **Object Detection:** The robotic arm should be able to detect objects in its environment using YOLO object detection. The arm should be able to identify the object's size, and location accurately in the camera frame.
- **Speech Recognition:** The robotic arm should be able to recognize voice commands given by the user using speech recognition. It should be able to interpret the user's commands accurately and transcribe it to native language (English).
- **Plan Generation:** The robotic arm must be able to generate a sequence of known simple steps to accomplish the goal state of the given task, and execute it.
- **Re-planning and correction:** The robotic arm should restart in the case of failure and replan to cover up its mistake and accomplish the task.
- **Arm Movement:** The robotic arm should be able to move its arm to a specific location accurately based on the object's location and the user's commands.
- **Grasping:** The robotic arm should be able to grasp the object accurately using its gripper or end-effector based on the object's size and shape.
- **Manipulation:** The robotic arm should be able to manipulate the object by moving it to a specific location or performing a specific task such as placing the object in a specific location.
- **Communication:** The robotic arm should be able to communicate with other nodes and systems using ROS messaging protocols. It should be able to send and receive messages to and from other devices in the system such as camera and other sensors.
- **Integration:** The robotic arm should be able to integrate with other systems and devices such as cameras, sensors, and other robotic systems to enhance its functionality and capabilities.

3.2 Non-Functional Requirements:

Non-functional requirements are those requirements that describe the performance, quality, and operational characteristics of the system. Some key non-functional requirements that should be considered include:

- **Accuracy and reliability:** The robotic arm should provide accurate and reliable performance in object detection, tracking, and interpretation of voice commands.
- **Performance:** The system should meet performance requirements such as speed, scalability, and responsiveness. It should be able to process information and perform tasks within an acceptable time frame and handle a growing number of requests without compromising the system's performance.
- **Safety:** The robotic arm should have safety measures in place to prevent any harm to the user or the surrounding environment. It should be able to detect any potential hazards and take appropriate measures to prevent accidents such collisions etc.
- **User Interface:** The robotic arm should have an intuitive and user-friendly interface for the user to interact with. It should be able to display relevant information about the object and the arm's movements to the user.
- **Maintainability:** The system should be easy to maintain, with modular and well-documented code and easy-to-replace components.
- **Compatibility:** The system should be compatible with different hardware configurations, operating systems, and versions of ROS.
- **Security:** The system should ensure secure data transmission and storage, protecting against unauthorized access, data breaches, and cyber-attacks.
- **Adaptability:** The system should be adaptable to changing environments, with the ability to learn from new data and adjust its performance accordingly.
- **Cost-effectiveness:** The system should be cost-effective, with affordable components and low maintenance costs, while still providing high-quality performance.

3.3 Software Requirements:

The Software requirement Specification is the official statement of what is required of the system developers. This requirement document includes the requirement definition and the

requirement specification. It should set out what the system should do without specifying how it should be done. The requirement set out in this document is complete and consistent.

Software required to develop this project:

- Operating System: Ubuntu 22.04 operating system
- Gazebo 9
- Rviz2
- ROS2 Humble
- C++
- Python 3.10 or above
- MoveIt2
- Python libraries:
 - Tensorflow
 - OpenCV
 - rclpy
- C++ libraries:
 - rclcpp
 - move_group_interface

3.4 Hardware Requirements:

Hardware required to develop the project:

- Processor: Intel i7 or above
- RAM: 16GB and above.
- Hard Disk: 30 GB SSD and Above.

Hardware required to run the application:

- Any embedded device with Ubuntu running on it for ROS2 with all the required dependencies installed.

CHAPTER 4

DETAILED DESIGN

4.1 System Architecture

ROS allows us to work on the different subsystems separately and provides the base for integrating the different subsystems. A.V.C.A.R consists of the following subsystems:

1. Task Planning Subsystem - Uses Plansys2
2. Arm Control Subsystem
3. Camera Perception Subsystem
4. Voice Input Subsystem.

In addition to the above subsystems we also have a subsystem for software simulation which would be replaced with driver code for hardware implementation. Fig.4.1.1 shows a simplified structure of the core logic software of the system.

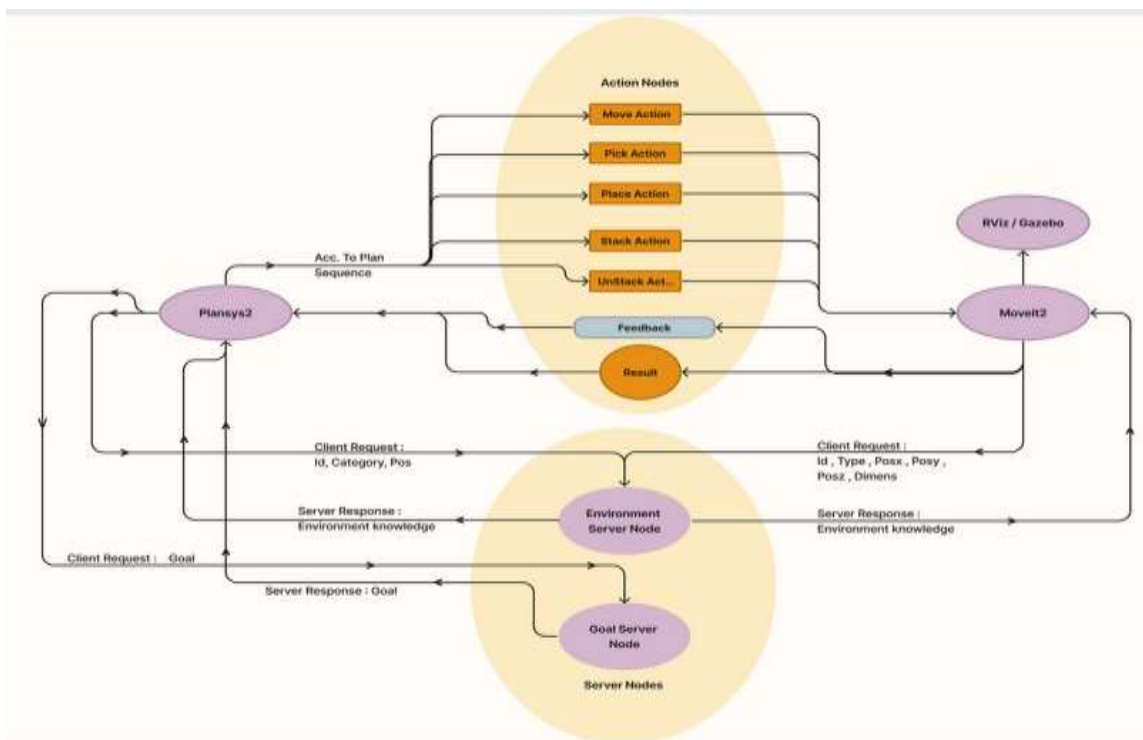


Fig. 4.1.1 Structure of the Backend Logic of the System

Individual programs within the system run independently and ROS merely acts as a communication service between the different programs, referred to as nodes when working with ROS. Fig.4.1 explains how the different nodes communicate with each other.

The Environment Server node is the Perception module which gets its input from the camera of the robot and then extracts information of the environment. The information extracted is used by both the planning system as well as the Motion Planning Node shown in the diagram as MoveIt2.

The Goal Server node is where the system gets the commands from the user via voice commands. The command is fed to the Planning System shown in the diagram as the Plansys2 Node.

The Plansys2 node is the planning sequence which uses the information from the Environment Server and the command from the Goal Node to generate a sequence of steps to carry out the task. The output of this node is fed to the Motion Planning Node.

In between the Plansys2 node and the MoveIt node we have Action nodes which act as feedback for the Plansys2 and as the nodes that send the function calls to the MoveIt node.

The MoveIt2 node is the Motion Planning subsystem. It uses the information from the Environment Server and acts on the command of the Plansys2 node to plan the trajectory and motion of the arm. The output of this node results in the motion of the arm which is either shown using Gazebo/RViz for software simulation or the hardware arm itself if it is in the hardware implementation.

ROS2 Humble is the middleware framework through which the subsystems communicate with each other. In the Architecture as shown in Fig.9.1 uses Actions and Services extensively. They are as follows:

- Environment Service is used to deliver information of the environment from the perception module.
- Goal Service operates between the Goal Server node (Speech Recognition) and Plansys2 node. Commands are passed through this service.
- Plansys2 and Action Nodes are connected using actions. Each action node has a separate action. Action nodes use services to request the MoveIt2 node for execution.

4.2 Data flow design

4.2.1 level 0 data flow design:

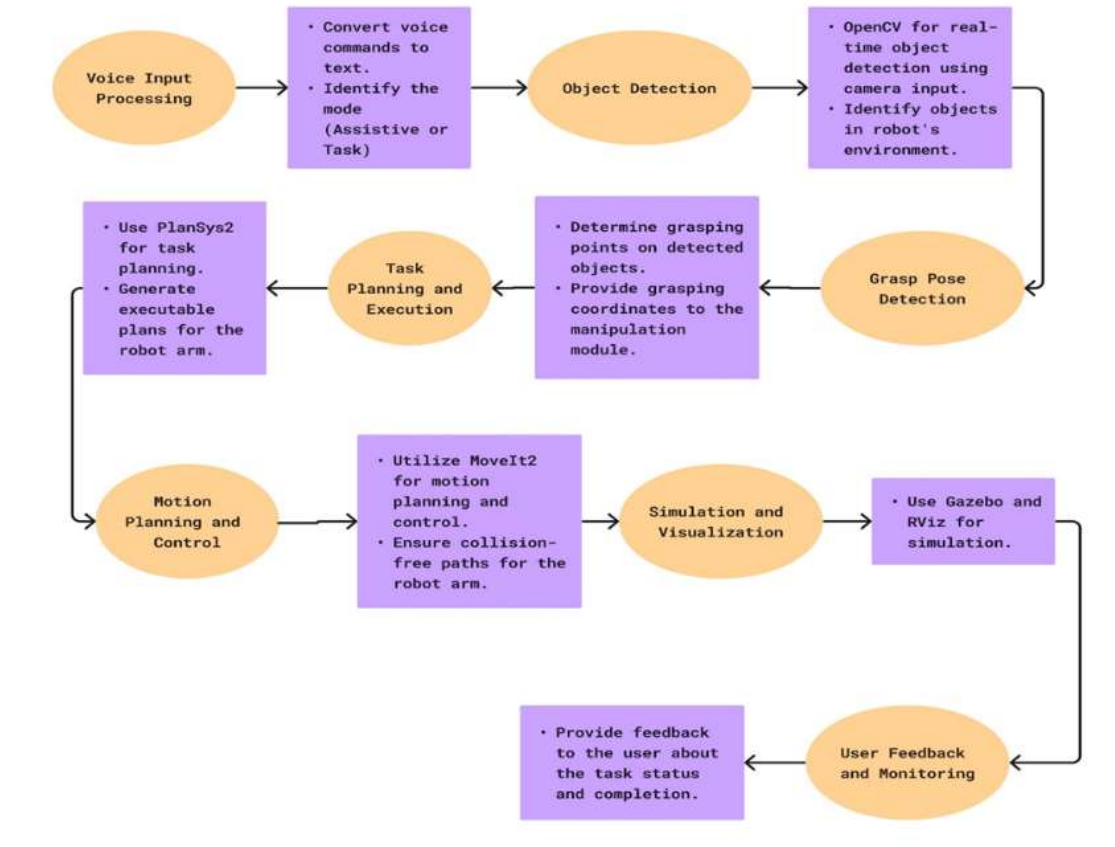


Fig 4.2.1.1 High level control flow of the system

1. The system waits for the user speech input, this is defaulted to 10 seconds.
2. The input speech signal is then sent to the Voice Control Module to recognize the user input.
 - a. The Speechmatics model converts the user input to text
 - b. The text is sent to Gemini API which converts the text to PDDL.
3. Upon matching the speech input with the predefined keywords, the YOLO is initialized and image from RGBD is fed to YOLO to detect and localize objects in the scene.
4. After getting the object coordinates, the objects with bounding box and number is displayed to the user for his desired choice.
5. The object pose location is then sent to python script to command the arm to move to the target position.

4.2.2 level 1 data flow design

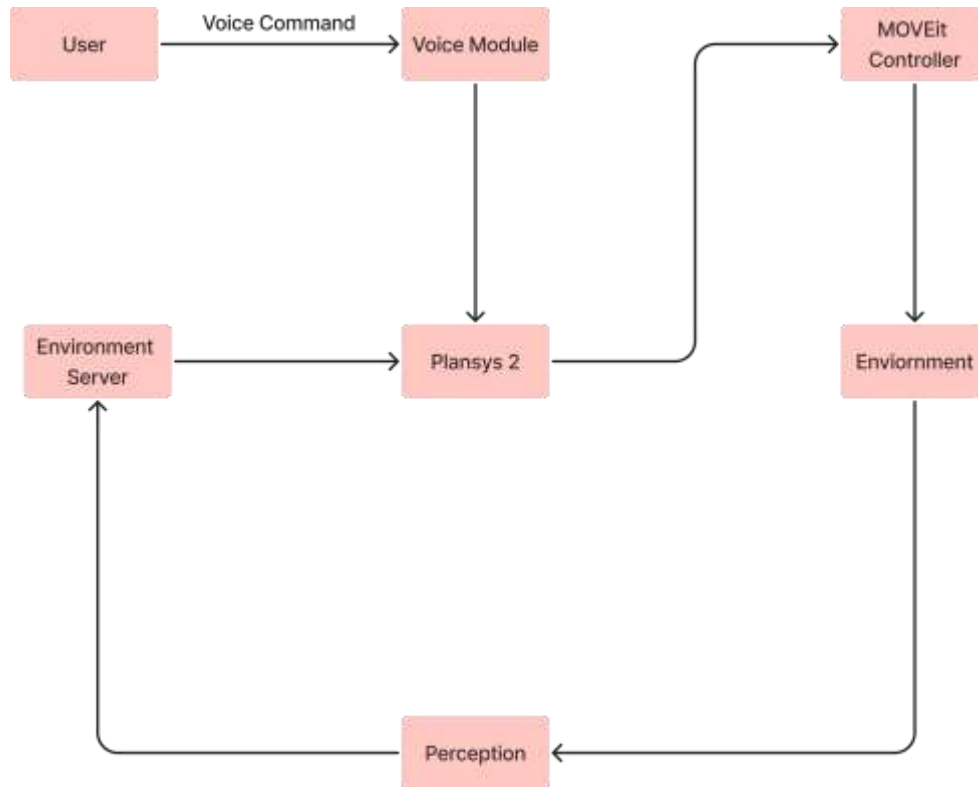


Fig 4.2.2.1 Complete control flow of the system.

1. Microphone is initialized to get the user audio input.
2. The parameters for the speech recognition model i.e Speechmatics is initialized, and the stream object is initialized.
3. Get the recognized/transcribed output from the model.
4. The output is matched against the predefined keywords using regular expression and if conditions.
5. If the match is “AMIGO”, again the system waits for further user voice input.
6. If not, a voice output is given saying that the activation command is wrong.
7. If not, a voice output is given saying that the command is wrong.
8. On successful recognition and matching of the output, YOLO model is initialized and the objects and the environment are noted down into the environment server.
9. The YOLO detects and localize the object by displaying bounding boxes and numbering on it.
10. Again, user will choose an object from the displayed objects, by specifying the object name as voice input.

11. The specified object's coordinates are then transformed to the target coordinate frame, which is then sent to the controllers of the arm.
12. Through ROS, MoveIt commander moves the end effector to the specified target pose.

4.3 Sequence Diagram

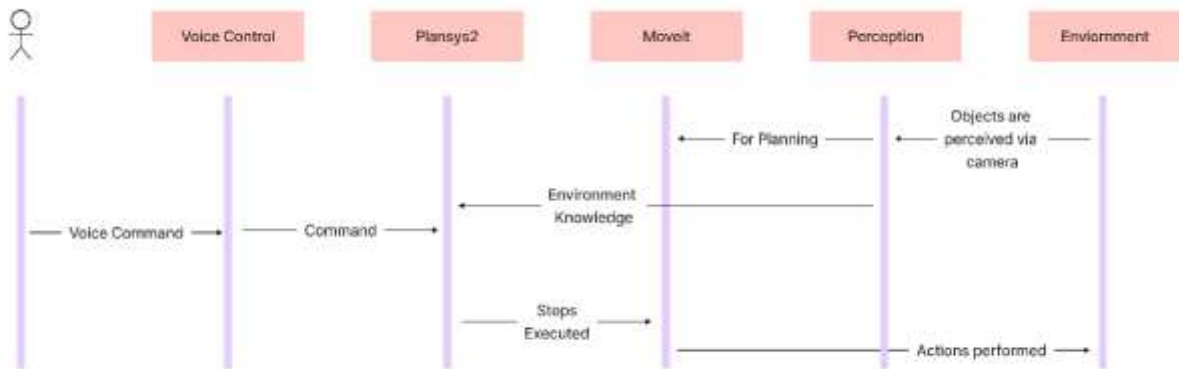


Fig 4.3.1 Sequence diagram

1. The system waits for the user to speak; the default wait time is five seconds.
2. The Speechmatics recognition model is then given the input voice signal to identify the user input and convert to PDDL.
3. The YOLO is initialized after the speech input is compared to the predetermined keywords, and an image from RGBD is provided to it to help it locate and identify things in the picture.
4. After obtaining the object coordinates, the user is shown the items along with their bounding boxes and numbers so he can make his selection.
5. The Python script with ROS MoveIt commander is then instructed to move the arm to the target position using the object's pose location.

4.4 Use case Diagram

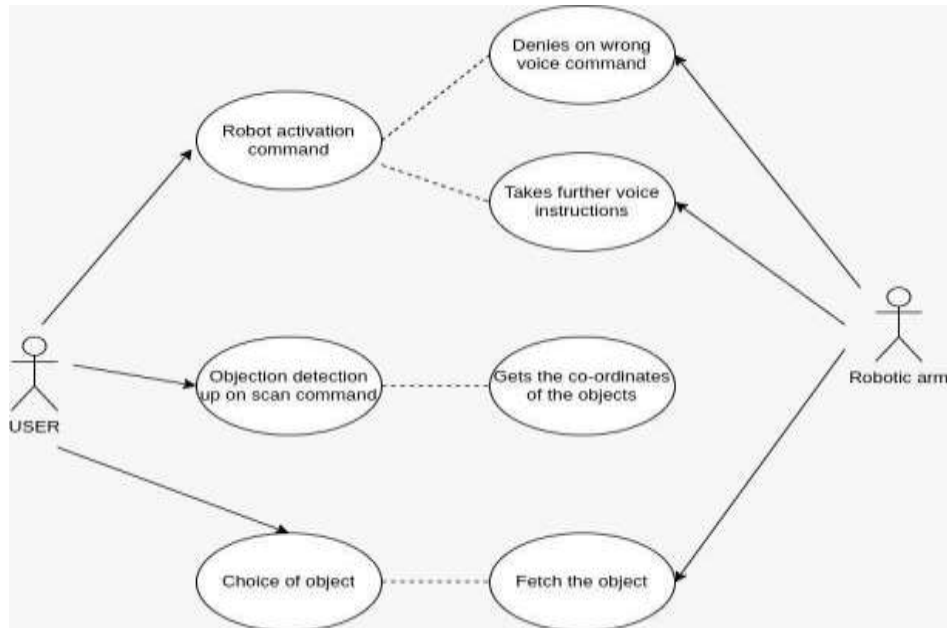


Fig 4.4.1 Use case diagram

The above use case diagram in fig 4.3 shows how the user and robotic arm will interact with each other.

1. There are two participants in the entire process:
 - a. User
 - b. Robotic arm
2. User can perform the following actions:
 - a. Give wakeup/activation command.
 - b. Give scan command and user choice of object as voice input.
3. Robotic arm perform the following actions:
 - a. Listens to the user's voice commands.
 - b. Runs the YOLO to perform object detection on the fed image stream.
 - c. Performs the coordinate transformation.
 - d. Execute the planned motion to grab and fetch the object

CHAPTER 5

IMPLEMENTATION

The entire implementation of the project can be divided into 4 major modules.

1. Voice Control Module
2. Perception module.
3. Plansys2 module
4. Arm Control and Simulation Module.

5.1 Voice Control

Our project incorporates a voice module for natural language user interaction. This module relies on two key components: the Speechmatics API for speech recognition and the Gemini API for natural language understanding and PDDL generation.

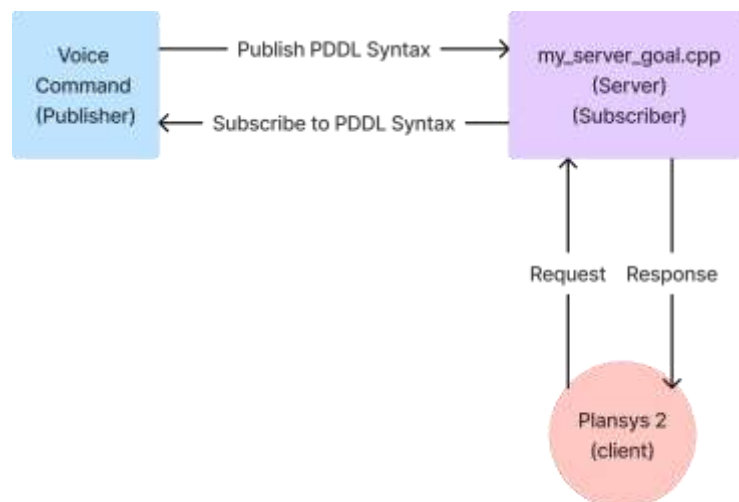


Fig 5.1.1 Voice Control Subsystem

5.1.1 Speech Recognition with Speechmatics:

We selected the Speechmatics API for speech-to-text conversion due to its several advantages:

1. **High Accuracy:** Speechmatics boasts a well-deserved reputation for delivering exceptional accuracy in speech recognition, especially when dealing with diverse

accents and challenging audio environments. This is crucial for our project, as robust user interaction is paramount.

2. **Scalability:** The Speechmatics platform offers various pricing tiers to accommodate different project needs. We opted for the free tier during initial development and testing phases, as it provides a generous quota for our usage. This allowed us to explore the functionality and integrate it into the project without initial financial investment.
3. **Ease of Use:** Speechmatics offers a user-friendly API with clear documentation and readily available support resources. This streamlined the integration process and facilitated efficient development.

5.1.2 Natural Language Processing with Gemini:

The transcribed user commands from Speechmatics are subsequently passed to the Gemini API. Here's why Gemini was the ideal choice for this critical step:

1. **PDDL Generation:** Gemini excels at converting natural language into PDDL syntax, which is the language understood by the PlanSys2 planner we utilize. This seamless integration between user input and planning logic was a key factor in our selection.
2. **Flexibility:** The Gemini API offers a wide range of functionalities, including intent recognition, entity extraction, and dialogue management. While we leverage its PDDL generation capabilities in this project, the API's versatility allows for potential future enhancements in user interaction.
3. **Free Tier Exploration:** Similar to Speechmatics, Gemini provides a free tier that proved valuable for initial development. This free tier allowed us to explore the core functionality within the project's scope and assess its suitability. Upgrading to a paid plan with more advanced features remains an option for future project iterations.

5.1.3 Module Operation

The voice module operates in a well-defined sequence:

1. **Trigger Word Detection:** The module constantly listens for the predefined trigger word, "Amigo," which initiates the user command capture process.
2. **Command Recording:** Upon detecting "Amigo," the module remains active for ten seconds, allowing the user to complete their utterance. This duration ensures that even complex commands are captured in their entirety.
3. **Speech-to-Text Conversion:** The recorded audio is then sent to the Speechmatics API for real-time speech recognition. The transcribed text, representing the user's command, is received by the voice module.
4. **Command Processing:** The transcribed command is forwarded to the Gemini API, where it is processed to understand the user's intent and extract relevant entities. Based on this interpretation, Gemini generates the corresponding PDDL syntax.
5. **Plan Generation:** The generated PDDL code is then utilized by the PlanSys2 planner to formulate an action plan based on the user's command and the project's current state.

5.1.4 Implementation of Speechmatics

```
import subprocess
import os

class dsp:
    def __init__(self):
        # activate virtual environment
        self.venv_path = "/home/shashank/dsp1"
        self.bin_path = os.path.join(self.venv_path, "bin")
        self.env = os.environ.copy()
        self.env["PYTHONPATH"] = self.venv_path
        # self.env["PATH"] = f"{self.bin_path}:{self.env['PATH']}"
        self.env["PATH"] = "{}:{}".format(self.bin_path, self.env['PATH'])

        # run command in subprocess with virtual environment activated
    def start(self):
        subprocess.call(["python3", "/home/shashank/catkin_ws/src/amigos/src/dsp-original.py"], env=self.env)
```

Fig 5.1.4.1 Speechmatics sub process

This code above code block defines a class `dsp` that activates a virtual environment and runs a Python script in a subprocess with the virtual environment activated since our ROS needs python 2.7 and Speechmatics needs python 3.6 or above.

Here's how it works:

- In the `__init__` method, the virtual environment path is defined.
- The ``env`` variable is set to a copy of the current environment variables using ``os.environ.copy()``. This will allow us to modify the environment variables for the subprocess.
- The `PYTHONPATH` environment variable is set to the virtual environment path.
- The `PATH` environment variable is modified to include the virtual environment bin path. This will ensure that any executables installed in the virtual environment can be found by the subprocess.
- The ``start`` method calls ``subprocess.call`` to run the Python script `'dsp-original.py'` in a subprocess. The ``env`` parameter is passed to the subprocess to activate the virtual environment.

```
# Encapsulate DeepSpeech audio feeding into a callback for PyAudio
text_so_far = ''
def process_audio(in_data, frame_count, time_info, status):
    global text_so_far
    data16 = np.frombuffer(in_data, dtype=np.int16)
    stt_stream.feedAudioContent(data16)
    text = stt_stream.intermediateDecode()
    if text != text_so_far:
        # print('Interim text = {}'.format(text[len(text_so_far)+1:]))
        text_so_far = text

    return (in_data, pyaudio.paContinue)

FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
CHUNK_SIZE = 1024

# Feed audio to deepspeech in a callback to PyAudio
audio = pyaudio.PyAudio()
stream = audio.open(
    format=FORMAT,
    channels=CHANNELS,
    rate=RATE,
    input=True,
    frames_per_buffer=CHUNK_SIZE,
    stream_callback=process_audio
)

print('Please start speaking...')
print("you have 5 seconds to complete....")
stream.start_stream()
```

Fig 5.1.4.2 Speechmatics audio transcription

We use PyAudio library to stream audio data from a microphone and feed it to a Speechmatics model for real-time speech-to-text transcription.

Here's how it works:

- The ``process_audio`` function is defined to take in audio data in the form of a byte string (``in_data``), convert it to a NumPy array of 16-bit integers (``data16``), and feed it to a Speechmatics streaming transcription object (``stt_stream``) using its ``feedAudioContent`` method. The ``intermediateDecode`` method is then called to get the intermediate transcription result, which is stored in the ``text`` variable.
- The PyAudio library is used to open an audio input stream from the default microphone using the ``audio.open`` method. The ``format``, ``channels``, ``rate``, and ``frames_per_buffer`` parameters are set to 16-bit PCM, mono, 16 kHz, and 1024 samples, respectively. The ``stream_callback`` parameter is set to ``process_audio``, which will be called every time a new chunk of audio data is available from the microphone.
- The ``stream.start_stream()`` method is called to start the audio stream and wait for new data to arrive. This will call the ``process_audio`` function repeatedly with new audio data, which will be fed to the Speechmatics model for transcription.

```
try:
    while stream.is_active():
        time.sleep(0.1)
except KeyboardInterrupt:
    print(text_so_far)
    with open("/home/shashank/catkin_ws/src/amigos/src/output.txt", "w") as f:
        print("writing speech detection to file...")
        f.write(text_so_far)
    stream.stop_stream()
    stream.close()
    audio.terminate()
    print('Finished recording.')
    # exit(0)
```

Fig 5.1.4.3 Saving transcription results

- The transcription function is called in a loop for 5 seconds, then the result which is stored in ``text_so_far`` is written to a text file.

- The stream channel is stopped.
- The audio listening is closed.

5.2 Perception Module

We have used deep learning-based YOLO object detection algorithm to detect objects on the table.

YOLO (You Only Look Once) is a popular real-time object detection algorithm that uses deep learning techniques to detect objects in images and video streams. The YOLO algorithm processes images in a single pass and outputs the detected objects along with their bounding boxes and confidence scores.

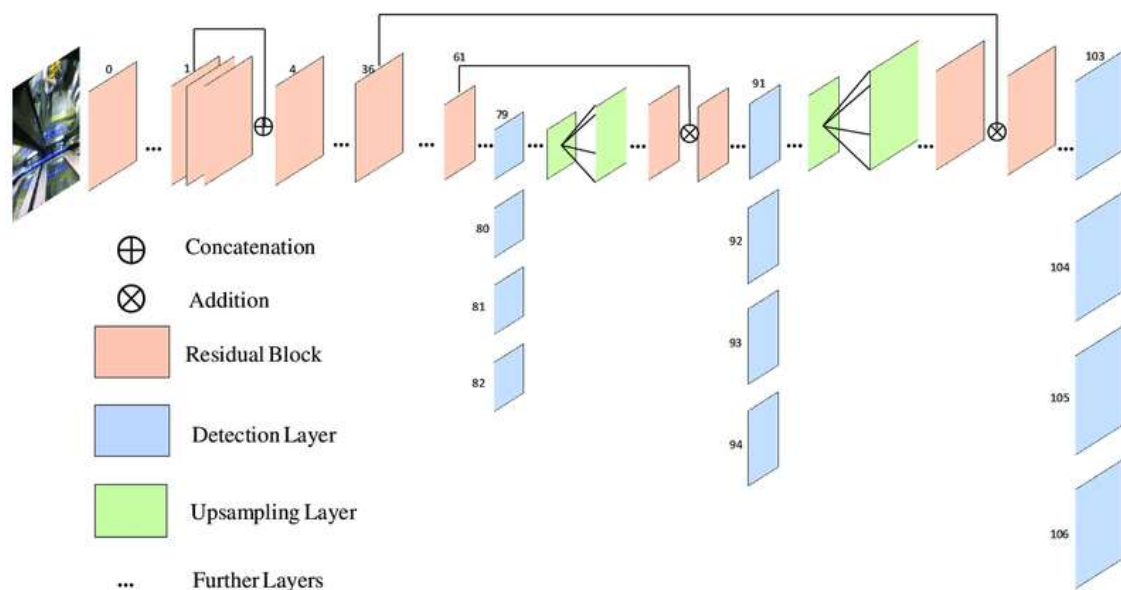


Fig 5.2.1 YOLO v8 architecture

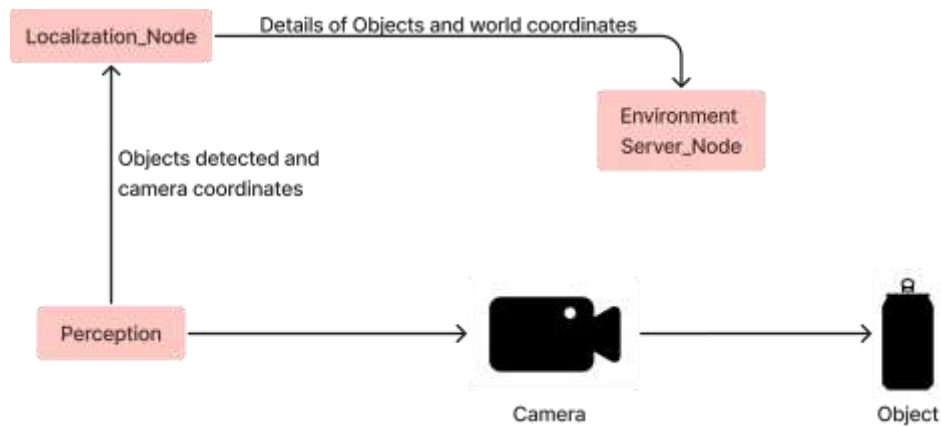


Fig 5.2.2 High-Level Architecture of Perception Subsystem

The YOLOv8 network is based on the Darknet-53 architecture, which is a deep neural network that consists of 53 convolutional layers. The Darknet-53 architecture is used as a backbone network to extract features from the input image. The features are then passed through several additional layers to generate the output predictions.

The YOLOv8 lite network predicts the coordinates of the bounding boxes that surround the detected objects, the class probabilities for each object, and the confidence scores for each detection. The network also uses anchor boxes to improve the localization of objects and incorporate skip connections to preserve spatial information.

It also includes several improvements over its predecessor, including a feature pyramid network that detects objects at different scales and improves the accuracy of small object detection, and a new technique called SPP (Spatial Pyramid Pooling) that allows the network to process input images of different sizes.

YOLO divides the input image into a grid and predicts the bounding boxes and class probabilities for each grid cell. Each grid cell can predict multiple bounding boxes, each with its own confidence score. The final set of predictions is obtained by selecting the boxes with the highest confidence scores and performing non-maximum suppression to remove overlapping boxes.

YOLO also incorporates several techniques to improve the accuracy of object detection, such as using anchor boxes to improve the localization of objects and incorporating skip connections to preserve spatial information. YOLO v3, the latest version of the algorithm, also uses a feature

pyramid network to detect objects at different scales and a new loss function to improve the training process.

5.2.1 Why YOLO?

Below are the reasons why we chose YOLO (You Only Look Once) over other object detection algorithms such as Fast R-CNN, Faster R-CNN, and SSD (Single Shot Detector):

- **Speed:** YOLO is significantly faster than other object detection algorithms, making it suitable for quick detection when arm is moving. YOLO processes images in a single pass and predicts object bounding boxes and class probabilities simultaneously.
- **Simplicity:** YOLO has a simple architecture compared to other algorithms, which makes it easier to train and deploy. The model architecture is also more straightforward, which allows for faster training and better optimization.
- **Accuracy:** Despite its speed and simplicity, YOLO has shown to be competitive with other state-of-the-art algorithms in terms of accuracy. The latest versions of YOLO, have achieved state-of-the-art performance on several object detection benchmarks.
- **Generalization:** YOLO is better at detecting small objects and objects in cluttered scenes where there are multiple objects on the table compared to other algorithms. It achieves this by using anchor boxes and multi-scale feature maps to detect objects at different scales and aspect ratios.

5.2.2 Implementation of YOLOv8

```

if predict(self):
    while(self.image is None):
        rospy.sleep(1)
    #Getting the image dimensions
    (self.H, self.W) = self.image.shape[:2]

    # self.center = (319.5, 239.5) # (self.W/2, self.H/2)
    self.center = (self.W/2, self.H/2)
    # self.focal_length = 525.0 # self.center[0] / np.tan(60/2 * np.pi / 180)
    self.focal_length = self.center[0] / np.tan(60/2 * np.pi / 180)
    self.camera_matrix = np.array([[self.focal_length, 0, self.center[0]],
                                    [0, self.focal_length, self.center[1]],
                                    [0, 0, 1]], dtype = "double")

    # determine only the *output* layer names that we need from YOLO
    ln = self.net.getLayerNames()
    ln = [ln[i][0] - 1] for i in self.net.getUnconnectedOutLayers()
    #Creating Blob from images
    blob = cv2.dnn.blobFromImage(self.image, 1 / 255.0, (416, 416), swapRB=True, crop=False)

    #Feeding the blob as input
    self.net.setInput(blob)
    #Forward pass
    layerOutputs = self.net.forward(ln)

    #Initializing lists of Detected Bounding Boxes, Confidences, and Class IDs
    boxes = []
    confidences = []
    classIDs = []

    #Looping over each of the layer outputs
    for output in layerOutputs:
        #Looping over each of the detections
        for detection in output:
            #Extracting the class ID and confidence
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            #Filtering out weak predictions
            if confidence > 0.2:
                #Scale the Bounding Box Coordinates
                box = detection[0:4] * np.array([self.W, self.H, self.W, self.H])
                (centerX, centerY, width, height) = box.astype("int")
                # use the center (x, y)-coordinates to derive the top and
                # and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))
                #Updating the Lists
                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)

```

Fig 5.2.2.1 YOLO detection

- Cv2.dnn.readFromDarknet function is used to load the custom trained model.
- The 'predict' function first waits for an image to be available, and then extracts the height and width of the image.
- It calculates the center of the image and the focal length of the camera, and creates a camera matrix based on these values. The camera matrix is used to convert pixel coordinates of objects into real-world coordinates.
- Next, the function determines the output layers of the YOLOv3 model, creates a blob from the input image, and feeds it through the model.
- The resulting output consists of a list of bounding boxes, each with an associated confidence score and class ID. The function then loops over each of these bounding boxes and filters out weak predictions (those with a confidence score below 0.2).
- For each remaining box, the function calculates the top-left corner coordinates and appends the box coordinates, confidence score, and class ID to the respective lists.

5.3 Task Planning Module

Plansys2 is the core component of AVCAR, without which there is nothing to show in the project. This subsystem is what generates the plan, which is a sequence of steps that help in achieving the goal state. We know that we can break down any task into a sequence of simpler steps, and a lot of actions are just variations and different combination of same simple steps. Instead of having to code every possible sequence, what if someone could generate the appropriate sequence based on context? Also what to do if due to some mechanical disturbance the robot is not able to do its job? It should be able to correct itself right? That's exactly what is being achieved here.

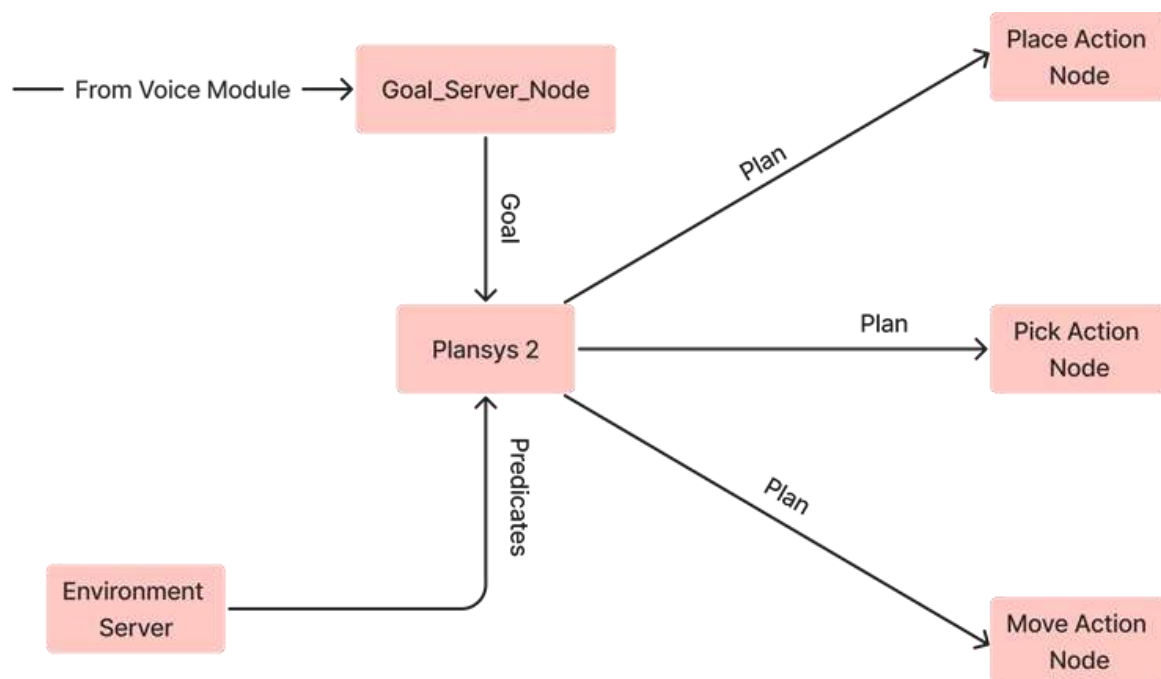


Fig 5.3.1 High-Level Diagram of Plansys2 Subsystem

The tools and technologies used here are discussed below as follows:

5.3.1 Plansys2

Plansys2 is a flexible and powerful robotic planning system designed to facilitate the development of autonomous systems. It serves as a middleware for robotic applications, enabling efficient task planning and execution. One of its key features is the incorporation of

the ROS (Robot Operating System) 2 framework, which enhances compatibility and interoperability with various robotic platforms and components.

Plansys2 leverages the Planning Domain Definition Language (PDDL) as a formalism for expressing planning problems. PDDL offers a declarative method to articulate high-level goals and constraints, enabling Plansys2 to autonomously generate executable plans suitable for robotic systems.

Moreover, Plansys2 provides a user-friendly interface for developers to interact with and monitor the planning process. This includes visualization tools to inspect the generated plans, debug potential issues, and analyze the system's behavior. Additionally, Plansys2 supports ROS 2's communication mechanisms, facilitating seamless integration with other ROS 2 nodes and components.

The open-source nature of Plansys2 encourages a collaborative and community-driven approach to development. This fosters the sharing of knowledge, improvements, and extensions among the robotics community. With its emphasis on flexibility, adaptability, and compatibility, Plansys2 plays a crucial role in advancing the capabilities of autonomous robotic systems in diverse applications, ranging from industrial automation to service robotics.

Fig.5.3 shows how Plansys2 has been integrated into the system for the project.

5.3.2 STRIPS Algorithm

The STRIPS (Stanford Research Institute Problem Solver) algorithm is a classical planning algorithm used for automated planning in artificial intelligence. In the context of PlanSys2, STRIPS can be used for generating plans to achieve goals in a dynamic environment. It is designed to generate plans to achieve specified goals. The core of the algorithm lies in predicate logic and state space search techniques to generate a plan i.e. a sequence of steps to achieve the desired goal.

The steps are as follows:

- First define the Domain : The actions that can be performed, the required preconditions and the resulting change in the state of the environment due to the action.
- Set the Predicates: Information of the environment. Set the Goal: We specify a goal state. The initial predicates form the initial state.
- Now the algorithm uses the result conditions of the first step as input to the next and finds a path to the goal state. The path, i.e. the sequence of the actions is the required plan.
- This plan is executed step by step as simple function calls in the Motion Planning Subsystem.

5.3.3 Planning Domain Definition Language

PDDL was introduced to standardize the representation of planning problems in a machine-readable format. It serves as a domain-specific language for describing the essential components of a planning scenario, including objects, predicates, actions, and the initial and goal states. PDDL provides a systematic approach to encoding planning problems, allowing automated planners to generate solutions by identifying sequences of actions that transform an initial state into a desired goal state.

5.4 Arm Control and Simulation Module

5.4.1 Robotic arm design/modelling

- The robotic arm design which we have used in this project is inspired/taken from UR5 (Universal Robotics).
- For the gripper we have used Robotiq two fingered gripper design.
- The Robotic arm definition and design is stored in URDF (Unified Robot Description Format) files.
- URDF is an XML based standard file format for specifying the geometry and organization of robots in ROS.

- URDF alone make code complex and more repetitive, hence we have used xacro to optimize the code.
- Xacro is a macro language that is used to simplify the creation and maintenance of URDF files in ROS. Xacro stands for XML macro, and it allows users to define and reuse XML snippets in a modular and flexible way. Below are some key aspects of Xacro in URDF:
 - Macros: Xacro allows us to define macros, which are reusable XML snippets that can be used to simplify the creation of URDF files. These can be defined with parameters, which makes them more flexible and adaptable to different scenarios.
 - Inclusion: Xacro allows users to include other URDF files and Xacro macros, which makes it easy to reuse and combine existing models and components.
 - Conditional statements: Xacro allows users to use conditional statements, which makes it possible to define different robot models based on different conditions, such as different sensors, actuators, or configurations.
 - Property definition: Xacro provides a way to define properties, which can be used to simplify the creation and maintenance of URDF files. Properties can be used to define common values, such as mass, length, or position, and they can be easily modified and reused throughout the file.

The URDF for the arm mainly consists of links and joints with other nested properties/tags such as geometry to define its shape etc.

Components of our robotic arm URDF are:

1. Links: A link in URDF represents a physical component of the robotic arm, such as a link, joint, or sensor. Links are defined by their geometry, mass properties, and inertia.
2. Joints: Joints in URDF define the movement of links relative to each other. A robotic arm typically has multiple joints, such as:
 - a. fixed: This type of joint is used when two links are fixed relative to each other and cannot move.
 - b. revolute: This type of joint allows rotational movement around a single axis.
 - c. continuous: This is a type of revolute joint that has no limit on the range of motion.
 - d. prismatic: This type of joint allows linear movement along a single axis.

- e. floating: This type of joint allows 6 degrees of freedom (3 translations and 3 rotations) between two links.
- f. planar: This type of joint allows movement in a plane defined by two axes.
- g. fixed-axis: This is a special type of joint that allows rotational movement around a fixed axis that is not aligned with any of the coordinate axes.

The joint properties, such as limits and ranges, are also defined in the URDF file.

3. Visual and Collision Models: URDF allows for the definition of both visual and collision models for each link. The visual model describes the appearance of the link in the robot's graphical interface, while the collision model defines the shape of the link for collision detection.
4. Inertial Properties: URDF allows for the definition of the mass, center of mass, and inertia properties of each link. These properties are used for simulation and control of the robot's motion.

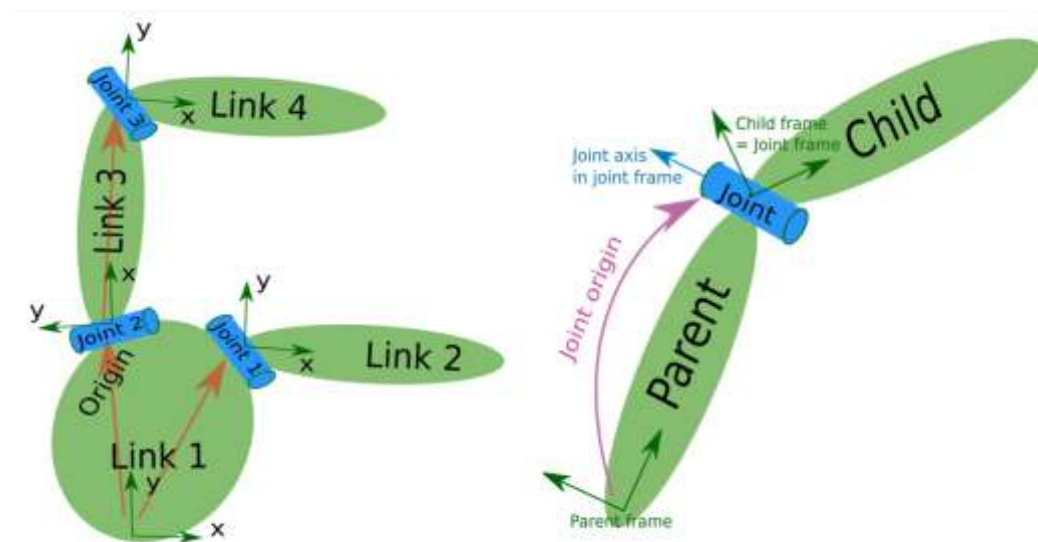


Fig 5.4.1.1 Robot Link and Joints

There are total 16 links defined for this robotic arm:

1. world
2. panda_base

UR5 arm Links:

3. panda_link0
4. panda_link1
5. panda_link2
6. panda_link3
7. panda_link4
8. panda_link5
9. panda_link6
10. panda_link7
11. panda_link8
12. panda_hand
13. camera_link
14. panda_leftfinger
15. panda_rightfinger
16. end_effector_frame

There are 15 joints which are connecting between the above 16 links as in the table below:

Parent link	Child link	Type	Joint name
Connecting Joints			
world	table	fixed	world_joint
panda_base	base_link	fixed	base_joint
UR5 arm Joints			
panda_link0	panda_link1	revolute	panda_joint1
panda_link1	panda_link2	revolute	panda_joint2
panda_link2	panda_link3	revolute	panda_joint3
panda_link3	panda_link4	revolute	panda_joint4
panda_link4	panda_link5	revolute	panda_joint5
panda_link5	panda_link6	revolute	panda_joint6
panda_link6	panda_link7	revolute	panda_joint7
panda_link7	panda_link8	revolute	panda_joint8
panda_link8	panda_hand	revolute	panda_hand_joint
panda_hand	camera_link	fixed	camera_joint
panda_hand	panda_leftfinger	prismatic	panda_finger_joint1
panda_hand	panda_rightfinger	prismatic	panda_finger_joint2
panda_rightfinger	end_effector_frame	fixed	end_effector_frame_fixed _joint

Table 5.4.1.1 Links and Joints for Panda Arm Robot

The below diagrams: Fig.5.4.1.2 and Fig. 5.4.1.3 show the Franka Emika Panda Robot arm, which was used for the project.



Fig 5.4.1.2 Franka Emika Panda Arm Robot



Fig 5.4.1.3 Simulated Environment

5.4.2 Arm Control Module

The core of the Arm Robot is the controller, which executes the commands from the Plansys2 planning system. For the simulation, we used models from IFRA (Intelligent Flexible Robotics Assembly) by Cranfield University. The models also included a package for controlling the models via action servers, where each server is a type of movement.

The strategy we applied to control the models was to create a node, which contains clients of the required servers and servers for the action nodes from plansys2, as seen in the diagram below:

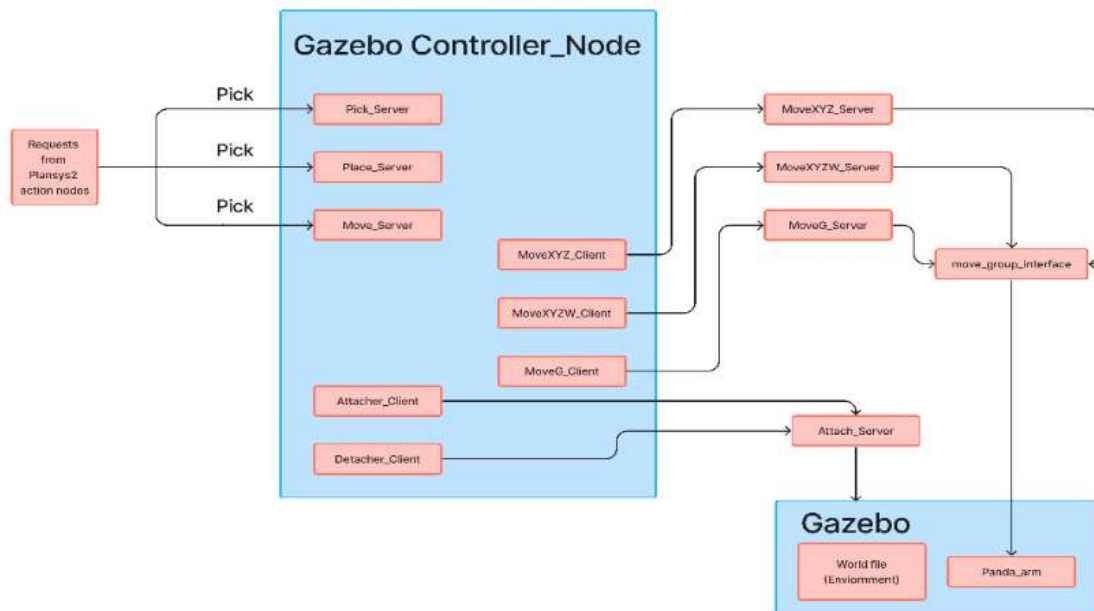


Fig 5.4.2.1 High-Level Architecture of the Arm Controller Subsystem

The action servers used in the project were:

1. **MoveXYZ:** this moves the arm to the desired coordinates (wrt to world coordinate frame) in a straight line with no change in the orientation of the gripper.

2. **MoveXYZW:** this moves the arm to the desired coordinates (wrt to world coordinate frame). However, here, we can set the orientation. We use this in areas where the end orientation of the gripper will be different.
3. **MoveG:** this moves only the gripper – open and close
4. **Attacher:** This is used to attach the objects to the arm and then detach. This is required as there is no concept of friction in the gazebo simulation and objects would slip out of the gripper if this is not done. This isn't required for the hardware implementation though.

The action nodes which carry out the steps from plansys2, act as clients to the controller node and so we also have a server function for each action node. Here, there are 3 of them – move, pick and place.

In the constructor of the controller node, we instantiated instances of the client classes for the action servers that connect to moveit and gazebo. Then, in the server functions, the appropriate actions are called to carry out the required tasks.

For example:- A simple pick and place action.

The steps would be as follows:

1. Move
2. Pick
3. Move
4. Place

For a Move step: We can just use the MoveXYZ action and send the appropriate coordinates as the arguments.

For a Pick step: We can arrange the sequence as follows:

- Move to the object using MoveXYZW
- Attach the object
- Close gripper using MoveG

- Lift the object using MoveXYZ to maintain orientation.

For a Place step: We can arrange the sequence as follows:

- Move to the desired location using MoveXYZW
- Place the object down using MoveXYZ to maintain orientation.
- Detach the object
- Open gripper using MoveG
- Lift the arm upward using MoveXYZW.

This module was implemented in C++ and rclcpp, as support for MoveIt2 is still under development for python and other languages. The algorithm that takes care of the trajectory planning underneath MoveIt2 in our project is RRT*

RRT* (RRT-star) is an extension of the basic RRT algorithm that aims to improve the efficiency and optimality of the generated paths. RRT* optimizes the tree structure by considering the cost of the paths and attempts to find a more optimal solution. This algorithm is particularly useful for robotic systems that require efficient and optimal paths in dynamic environments.

In the context of MoveIt2, which is an updated version of MoveIt with enhancements and new features, you can use RRT* for motion planning to generate more optimal paths for a robot to navigate from a start configuration to a goal configuration. Here's a general overview of how RRT* could be applied in the context of MoveIt2:

Configuration Space Representation:

Represent the robot and its surroundings in a configuration space.

- Initialization:
Define the starting and target configurations.
- RRT Tree Expansion:*
Grow a tree from the starting configuration by randomly exploring and extending towards new configurations. Consider path costs to minimize overall tree cost.
- Optimizing Paths:
Continuously optimize the tree by rewiring connections to reduce path costs.

- **Goal Check:**
Verify if the goal is reached within a set tolerance.
- **Path Execution:**
Execute the optimized path using MoveIt2's motion control interfaces.

5.5 ROS Controllers and plugins

To actuate a robot's actuators, such as motors or servos, controllers in ROS are software modules that compute control signals. As they translate high-level commands (such "move to this position" or "grab this object") into low-level motor commands that propel the robot's movements, controllers are an essential component of a robot's control system.

These controllers are defined in the `ros_controllers.yaml` file

5.4.1 ROS Controllers

ROS controllers plugins which we have used in this project are as follows:

1. **Joint Trajectory Controller:** A ROS controller plugin called the Joint Trajectory Controller creates smooth trajectories for joint locations, velocities, and accelerations. In order to give the robot's mobility accurate control, it is frequently utilized in robotic arms and other jointed robots.
Receiving a trajectory message that details the required joint locations, velocities, and accelerations over time allows the Joint Trajectory Controller to function. After that, it creates a smooth route for the robot to follow using this knowledge.
Additionally capable of managing numerous trajectories at once, the Joint Trajectory Controller enables sophisticated motion planning and execution. To generate the necessary motion profile, it can be combined with a variety of methods, including cubic splines and quintic polynomials.
2. **Joint State Controller:** A ROS controller plugin called the Joint State Controller is used to communicate joint state information to other nodes in the ROS network. Providing current data on the position, speed, and effort of each joint in the robot is a popular use for it in robotic systems.

The robot's hardware interface, such as an encoder or other sensor, provides the Joint State Controller with joint position and velocity data. Then, it makes this data available in a JointState message that other network nodes can subscribe to.

To offer closed-loop control of the robot's motion, the Joint State Controller can also be utilised in conjunction with other ROS controllers. For example, the Joint Trajectory Controller can use information from the Joint State Controller to generate smooth trajectories for the robot to follow.

3. **Joint Position Controller:** A ROS controller plugin called the Joint Position Controller is used to manage a robot's joints' positions. It generates control signals to move the joints to the specified locations after receiving commands in the form of target joint positions.

The robot's hardware interface is used by the Joint Position Controller to read the current joint positions, which are then compared to the target positions supplied in the command message. The appropriate control signals, such as torque or voltage, are subsequently calculated in order to move the joints towards the desired positions.

Here the Joint Position Controller used for pick-and-place operations, assembly lines, and mobility robots. Additionally, it can be used in conjunction with other ROS controllers, like the Joint Trajectory

4. **Joint Velocity Controller:** A ROS controller plugin called the Joint Velocity Controller is used to regulate a robot's joints' velocity. It creates control signals to move the joints at the target joint velocities in response to commands in the form of target joint velocities.

The current joint velocities are read from the hardware interface of the robot and compared to the goal velocities stated in the command message by the Joint Velocity Controller. Then, in order to move the joints at the specified speeds, the necessary control signals, such as torque or voltage, are calculated.

5. **Joint Effort Controller:** A ROS controller plugin called the Joint Effort Controller is used to regulate the effort or torque of a robot's joints. Target joint efforts are used as commands, and it generates control signals to move the joints in accordance with those efforts.

The current joint locations and velocities are read from the hardware interface of the robot and compared to the target joint efforts stated in the command message by the Joint Effort Controller. After that, it determines the control signals—such as voltage or current—that are required to move the joints with the required force.

The Joint Effort Controller is where the robot needs to apply a specific amount of force or torque, such as in robotic grasping or manipulation tasks. It can also be used in conjunction with other ROS controllers, such as the Joint Position Controller or the Joint Velocity Controller, to provide closed-loop control of the robot's motion.

6. **Gripper Controller:** A robot gripper, such as a robotic hand or end effector, can have its position and/or level of effort controlled using the Gripper Controller, a ROS controller plugin. Target gripper positions or efforts are input as commands, and the device creates control signals to move the gripper in accordance with those positions or efforts.

The robot's hardware interface is used by the Gripper Controller to read the gripper's current position and effort, which it then compares to the goal position or effort supplied in the command message. The gripper is then moved with the desired position or force after the system calculates the necessary control signals, such as voltage or current.

7. **Cartesian Controller:** The Cartesian controller is a ROS controller plugin that controls the end-effector of a robot in Cartesian space (i.e., XYZ coordinates and orientation) instead of controlling the individual joints. It takes in desired Cartesian poses and uses inverse kinematics to compute the joint positions required to achieve that pose. It can be used for controlling robotic arms or mobile robots with holonomic motion.
8. **Force/Torque Controller:** The Force/Torque controller is a ROS controller plugin that enables a robot to apply force/torque in a desired direction to its environment. This type of controller is commonly used in robotic applications such as grasping, manipulation, and assembly tasks. It takes in desired force/torque values and uses a feedback control loop to adjust the joint positions of the robot in order to maintain the desired force/torque. The Force/Torque controller can be used with a variety of sensors, including force/torque sensors and tactile sensors.

9. **Computed Torque Controller:** The Computed Torque controller is a ROS controller plugin that controls the motion of a robot by computing the required torques at each joint based on the desired trajectory, robot dynamics, and feedback from sensors. It is a model-based control method that uses a mathematical model of the robot dynamics to calculate the torques required to achieve the desired motion. The Computed Torque controller is often used in applications where high precision and accuracy are required, such as in industrial robotics or robotic surgery. It can also be used for trajectory tracking and path following tasks.

5.5.2 ROS plugins

A plugin in ROS is a modular piece of software that may be dynamically loaded and unloaded during operation. Plugins are made to add new features or increase the functioning of already installed software modules. They are frequently employed to add functionality to already-existing ROS nodes or to develop drivers for hardware devices.

A library named Pluginlib, which offers a framework for developing and maintaining plugins, is used to build plugins in ROS. A set of interfaces and base classes that plugins can extend are defined by Pluginlib. The functionality that plugins must offer in order to cooperate with other ROS software modules is specified by these interfaces and base classes.

Plugins are added by following the below mentioned steps:

1. Create a package for your plugin: Use `ros2 pkg create`` to create a new package for your plugin.
2. Create a plugin: Write your plugin code using the appropriate ROS plugin template, such as `nodelet::Nodelet`` for nodelets or `pluginlib::ClassLoader`` for libraries.
3. Add plugin dependencies: Declare any dependencies in the package manifest file (`package.xml``) or the build manifest file (`CMakeLists.txt``).
4. Build the plugin: Use `colcon build`` to build the package containing your plugin.
5. Use the plugin: Once the plugin is built, it can be used in other ROS nodes or packages by loading it using the appropriate ROS API, such as `nodelet::Loader`` for nodelets or `pluginlib::ClassLoader`` for libraries.

ROS plugins which are used in this project are:

1. Robot Model: This plugin displays a 3D model of a robot in RViz.
2. TF: This plugin provides the ability to visualize the transform tree in RViz.
3. Image View: This plugin allows you to view and save image data published on ROS topics.
4. RQT Console: This plugin provides a console for displaying ROS messages.
5. Robot Localization: This plugin provides a way to estimate the pose of a robot using sensor data.
6. MoveIt!: This plugin provides a way to plan and execute robot motion planning.
7. Gazebo: This plugin provides a way to simulate robots and environments in Gazebo.
8. URDF: This plugin provides a way to visualize and manipulate URDF models.
10. ROS Control: This plugin provides a way to control robot joints using various controllers.
11. RViz: This plugin provides a 3D visualization tool for robot data.

5.5.3 Gazebo plugins

Gazebo plugins are software components that can be added to Gazebo simulations to extend their functionality. They are written in C++ and can be used to simulate sensors, actuators, controllers, and other elements of a robot. Gazebo plugins provide a way to interface between a robot model and the simulation environment, allowing for more accurate and realistic simulations.

Below are few gazebo plugins used in this project:

1. Gazebo Ros Packages: This package provides ROS plugins for Gazebo. It includes plugins for publishing and subscribing to sensor data, controlling robot models, and more.
2. Gazebo Ros Control Plugin: This plugin provides an interface between ROS controllers and Gazebo's physics engine. It allows for simulation of robot models with ROS controllers.
3. Gazebo Grasp Plugin: This plugin enables simulation of grasping and manipulation tasks in Gazebo.
4. Gazebo IMU Plugin: This plugin simulates inertial measurement units (IMUs) in Gazebo. It generates simulated sensor data for use in ROS.
5. Gazebo Camera Plugin: This plugin simulates cameras in Gazebo. It generates simulated camera data for use in ROS.
6. Gazebo Laser Plugin: This plugin simulates laser range finders in Gazebo. It generates simulated laser scan data for use in ROS.
7. Gazebo Force-Torque Sensor Plugin: This plugin simulates force-torque sensors in Gazebo. It generates simulated force-torque data for use in ROS.

There are three plugins i.e oppeni_kinect for camera and gazebo_grasp_fix for grasping and force-torque are customized and manually added to the robot.

```

<gazebo reference="camera_link">
  <sensor name="kinectdepth" type="depth">
    <camera name="kinect">
      <horizontal_fov>1.0382</horizontal_fov>
      <image>
        <width>1280</width>
        <height>720</height>
        <format>B8G8R8</format>
      </image>
      <clip>
        <near>0.1</near>
        <far>100</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.100</stddev>
      </noise>
    </camera>
    <always_on>1</always_on>
    <update_rate>30</update_rate>
    <visualize>0</visualize>
    <plugin filename="libgazebo_ros_openni_kinect.so" name="kinect_controller">
      <baseline>0.2</baseline>
      <alwaysOn>true</alwaysOn>
      <updateRate>30.0</updateRate>
      <cameraName>kinect_ir</cameraName>
      <imageTopicName>/kinect/color/image_raw</imageTopicName>
      <cameraInfoTopicName>/kinect/color/camera_info</cameraInfoTopicName>
      <depthImageTopicName>/kinect/depth/image_raw</depthImageTopicName>
      <depthImageInfoTopicName>/kinect/depth/camera_info</depthImageInfoTopicName>
      <pointCloudTopicName>/kinect/depth/points</pointCloudTopicName>
      <frameName>camera_frame</frameName>
      <pointCloudCutoff>0.5</pointCloudCutoff>
      <pointCloudCutoffMax>3.0</pointCloudCutoffMax>
      <distortionK1>0.00000001</distortionK1>
      <distortionK2>0.00000001</distortionK2>
      <distortionK3>0.00000001</distortionK3>
      <distortionT1>0.00000001</distortionT1>
      <distortionT2>0.00000001</distortionT2>
      <CxPrime>0</CxPrime>
      <Cx>0</Cx>
      <Cy>0</Cy>
      <focalLength>0</focalLength>
      <hackBaseline>0</hackBaseline>
    </plugin>
  </sensor>
</gazebo>

```

Fig 5.5.3.1 Code snippet of openni_kinnect plugin

The above code is an XML configuration for a Gazebo simulation environment. It defines a sensor called "kinectdepth" which is of type "depth". The sensor has a camera component with the following properties:

- `horizontal_fov`: the horizontal field of view of the camera, measured in radians.

- image: the resolution and pixel format of the image output by the camera.
- clip: the near and far clip planes of the camera's frustum.
- noise: the type, mean, and standard deviation of any noise added to the camera's output.

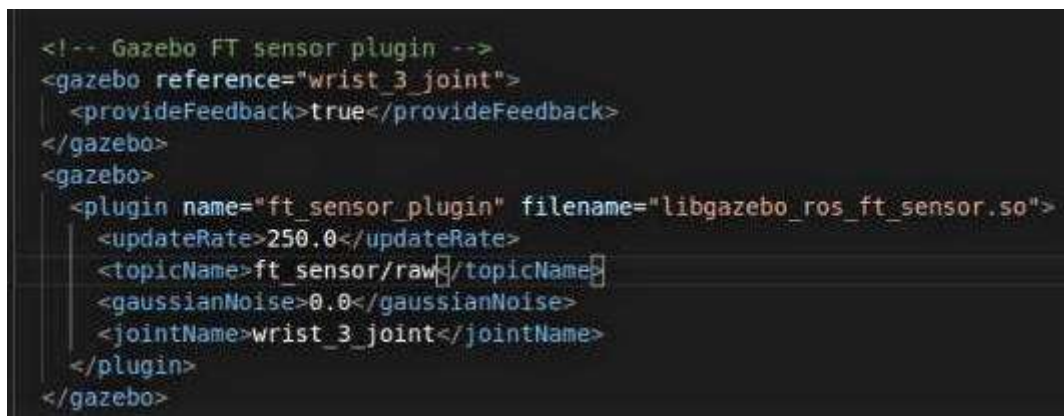
The sensor is set to "always_on" and has an "update_rate" of 30Hz. Visualization of the output is disabled ("visualize" set to 0). The sensor is implemented using the "libgazebo_ros_openni_kinect.so" plugin and is controlled by a component named "kinect_controller". The plugin provides various properties related to the behavior and configuration of the sensor, including baseline, update rate, camera name, image and depth topics, and camera frame name.

```
<!-- Gazebo grasping plugin -->
<gazebo>
  <gripper name="gazebo_gripper">
    <grasp_check>
      <attach_steps>2</attach_steps>    <!-- default: 20 -->
      <detach_steps>2</detach_steps>    <!-- default: 40 -->
      <min_contact_count>3</min_contact_count>
    </grasp_check>
    <gripper_link>robotiq_85_left_finger_tip_link</gripper_link>
    <gripper_link>robotiq_85_right_finger_tip_link</gripper_link>
    <palm_link>robotiq_85_base_link</palm_link>
  </gripper>
</gazebo>

<gazebo>
<plugin filename="libgazebo_grasp_fix.so" name="gazebo_grasp_fix">
  <arm>
    <arm name>ur5_arm</arm name>
    <palm_link>robotiq_85_left_inner_knuckle_link</palm_link>
    <palm_link>robotiq_85_right_inner_knuckle_link</palm_link>
    <gripper_link>robotiq_85_right_finger_tip_link</gripper_link>
    <gripper_link>robotiq_85_left_finger_tip_link</gripper_link>
  </arm>
  <forces_angle_tolerance>100</forces_angle_tolerance>
  <update_rate>10</update_rate>
  <grip_count_threshold>4</grip_count_threshold>
  <max_grip_count>8</max_grip_count>
  <release_tolerance>0.05</release_tolerance>
  <disable_collisions_on_attach>false</disable_collisions_on_attach>
  <contact_topic>__default_topic__</contact_topic>
</plugin>
</gazebo>
```

Fig 5.5.3.2 Code snippet of gazebo_grasp plugin

The above block of XML code defines a Gazebo plugin named "gazebo_grasp_fix" which is loaded as a plugin on an arm named "ur5_arm". The plugin is responsible for ensuring stable grasps by adjusting the contact forces applied to objects in the arm's gripper. The arm has a palm link named "robotiq_85_base_link" and two gripper links named "robotiq_85_left_finger_tip_link" and "robotiq_85_right_finger_tip_link". The plugin updates at a rate of 10 Hz, and will only attempt to grasp an object if at least 4 contact points are detected. The maximum number of contact points for a grasp is 8. The plugin will disable collisions on objects being grasped if the "disable_collisions_on_attach" parameter is set to true. Finally, the "contact_topic" parameter specifies the name of the ROS topic where contact information is published.



```
<!-- Gazebo FT sensor plugin -->
<gazebo reference="wrist_3_joint">
  <provideFeedback>true</provideFeedback>
</gazebo>
<gazebo>
  <plugin name="ft_sensor_plugin" filename="libgazebo_ros_ft_sensor.so">
    <updateRate>250.0</updateRate>
    <topicName>ft_sensor/raw</topicName>
    <gaussianNoise>0.0</gaussianNoise>
    <jointName>wrist_3_joint</jointName>
  </plugin>
</gazebo>
```

Fig 5.5.3.3 Code snippet of force-torque sensor plugin

The first block of the above code specifies that the joint with reference "wrist_3_joint" in the Gazebo simulation will provide feedback. This means that the state of this joint will be monitored and can be used by other components in the simulation.

The second block of code defines a Gazebo plugin that simulates a force-torque sensor. The plugin is named "ft_sensor_plugin" and its implementation is defined in the shared library file "libgazebo_ros_ft_sensor.so". The plugin is associated with the joint "wrist_3_joint", and it will publish force-torque readings on the ROS topic "ft_sensor/raw". The "updateRate" parameter specifies the frequency of the published readings. Finally, "gaussianNoise" is a parameter that allows adding Gaussian noise to the sensor readings to simulate measurement noise.

CHAPTER 6

TESTING

System testing constitutes a crucial phase in software assessment, aiming at evaluating the comprehensive and interconnected system to ascertain its conformity with intended functionality and specified requirements. This encompasses a thorough examination of the system in its entirety, encompassing all constituent elements, modules, interfaces, and interoperations, to validate its behavior, efficacy, and dependability.

In the domain of a autonomous voice control arm robot, system testing involves the meticulous evaluation of the complete robotic arm system, comprising the structural model, control software, and integrated components such as plugins. The objective is to ensure the smooth functioning and optimal performance of the entire robotic arm system, including its software components, ensuring seamless collaboration to achieve desired functionality and meet predefined criteria.

Within our project framework, system testing for the autonomous voice control arm robot leveraging YOLOv8 for object detection and speech recognition via Speechmatics is imperative. It encompasses scrutinizing the integrated functionality and performance of the robotic arm, object detection capabilities facilitated by YOLOv8, and the accuracy of speech recognition provided by Speechmatics in diverse scenarios.

Moreover, considering the utilization of Plansys2 and MoveIt2 within our system architecture, the system testing process extends to validating the cohesive operation of these components alongside YOLOv8 and Speechmatics, ensuring compatibility and efficacy across all integrated modules.

We test the entire system in below mentioned stages:

6.1. Testing Robotic Arm Functionality:

- Ensure the autonomous voice-controlled robotic arm operates as designed, assessing its movement precision and control mechanisms.
- Validate various arm configurations, joint movements, and end-effector functionalities to confirm proper operation.
- Assess the arm's capability to engage with objects and execute desired tasks like grasping, positioning, or manipulating items.

Test id	Input to the system	Environment	Expected result	Testing outcome
1	Command to move to a position on table	MoveIt (Interface), Plansys2 Gazebo (simulator)	Robotic Arm The robotic arm should move to the specified position accurately	Robotic arm moves to the target position within acceptable tolerance.
2	Initiate joint movement with specified angles	MoveIt (Interface), Gazebo (simulator)	Robotic Arm The robotic arm joints should adjust to the specified angles accurately	Robotic arm joints move to the desired angles as commanded.
3	Test collision detection with obstacles	MoveIt (Interface), Gazebo (simulator)	The robotic arm should halt movement upon detecting a collision with simulated obstacles	Robotic arm stops upon collision detection with simulated obstacles.
4	Command to grasp an object	MoveIt (Interface), Gazebo (simulator)	The robotic arm gripper should close around the specified object securely	Robotic arm gripper closes around the object firmly without slippage.
5	Command to release a grasped object	MoveIt (Interface), Gazebo (simulator)	The robotic arm gripper should release the grasped object smoothly	Robotic arm gripper releases the object Securely.

Table 6.1.1 Testing Robot Arm Functionality

6.2. Object Detection Testing with YOLOv8:

- Assess the object detection functionalities of YOLOv8 within the robotic arm system.
- Evaluate the accuracy and consistency of object detection by examining YOLOv8's ability to identify and locate objects in real-time.
- Test YOLOv8's performance across diverse scenarios, encompassing objects of varied sizes, occlusions, challenging lighting conditions, and intricate backgrounds.
- Quantify the object detection performance of YOLOv8 through metrics such as precision, recall, and average precision.

Test id	Input to the YOLO	Environment	Expected result	Testing outcome
1	Image containing a Coke can was input into the trained model	Ubuntu OS, Python 3.10.2	Bounding box outlining the Coke can with its center coordinates	The model accurately detected the Coke can, generating a bounding box with its center coordinates.
2	Image containing a syrup was input into the trained model	Ubuntu OS, Python 3.10.2	Bounding box outlining the Syrup with its center coordinates	The model accurately detected the Coke can, generating a bounding box with its center coordinates.
3	Image containing a syrup was input into the trained model	Ubuntu OS, Python 3.10.2	Bounding boxes for all the objects with object center coordinates	The model accurately detected the object generating a bounding box with its center coordinates.

Table 6.2.1 Object Detection Testing Through YOLO v8

6.3. Testing Speech Recognition:

- Check how well Speechmatics works with Voice Gemini, the module that recognizes object names and positions, then converts them into PDDL syntax for the robotic arm system.
- Make sure the system accurately understands spoken commands, especially for recognizing objects and giving positional instructions, then translates them into PDDL for actions.
- See if speech recognition works correctly in different situations, like with different accents, background noise, and speaking styles.

Test id	Input to Speechmatics model	Environment	Expected result	Testing outcome
1	Speech input "amigos" is spoken into the microphone.	Ubuntu OS, Python 3.10.2	The system should recognize the spoken word and accurately transcribe it into English.	The model loads successfully, detects the audio input, and accurately transcribes it into English.
2	Spoken input is provided, with background noise present.	Ubuntu OS, Python 3.10.2	The system should separate relevant words from background noise and accurately transcribe them.	The model accurately distinguishes spoken words from background noise and transcribes them correctly. It matches the spoken words with their intended meanings, even with noise present.
3	Speech input with varied accents is recorded.	Ubuntu OS, Python 3.10.2	The system should precisely transcribe words spoken with different accents, especially recognizing object names and positional instructions for PDDL syntax generation.	The model accurately transcribes speech input with a US accent. However, it struggles to accurately transcribe speech with more complex accents into English.

Table 6.3.1 Testing Speech Recognition

6.4. Integration Testing:

Ensure the seamless integration of the robotic arm, YOLOv8, and speech recognition components.

Test the system's capacity to take input from the object detection module (YOLOv8) and utilize it to direct the actions of the robotic arm.

Evaluate the system's proficiency in recognizing spoken commands, converting them into actions, and accomplishing tasks using the robotic arm.

Assess the system's responsiveness, synchronization, and overall performance while handling simultaneous tasks of object detection and speech recognition.

Test id	Input to the integrated system	Environment	Expected result	Testing outcome
1	Speech input is given through the microphone.	Ubuntu OS, Python 3.10.2	The system should detect the audio input and accurately transcribe it into English text.	The model loads successfully, detects the audio input, and transcribes it accurately into English.
2	Command to pick up an object is issued	Ubuntu OS, Python 3.10.2	The system should perform the action of moving and grabbing the object.	The robotic arm moves to the target coordinates and successfully grasps the object by closing the arm gripper.

Table 6.4.1 Integration Testing

CHAPTER 7

RESULTS AND DISCUSSIONS

In this chapter, we showcase the outcomes attained through the implementation of the project's diverse modules and assess the collective performance of the integrated system. We conducted experiments on individual modules as well as on the integrated system, and subsequently analyzed the results.

7.1 YOLOv8 object detection results:

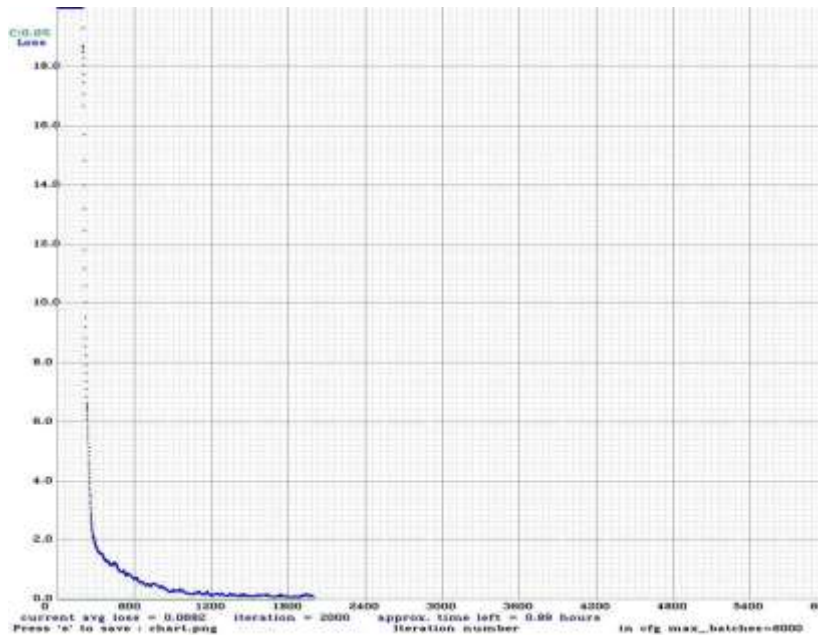


Fig 7.1.1 YOLO training loss curve

The YOLO model was trained for 2000 iterations, focusing solely on the "coke can" class. This class comprised 30 images, each subject to various augmentations including shear, random crop, brightness adjustment, and Gaussian noise. Throughout the training process, the model achieved an overall loss of 0.0682.

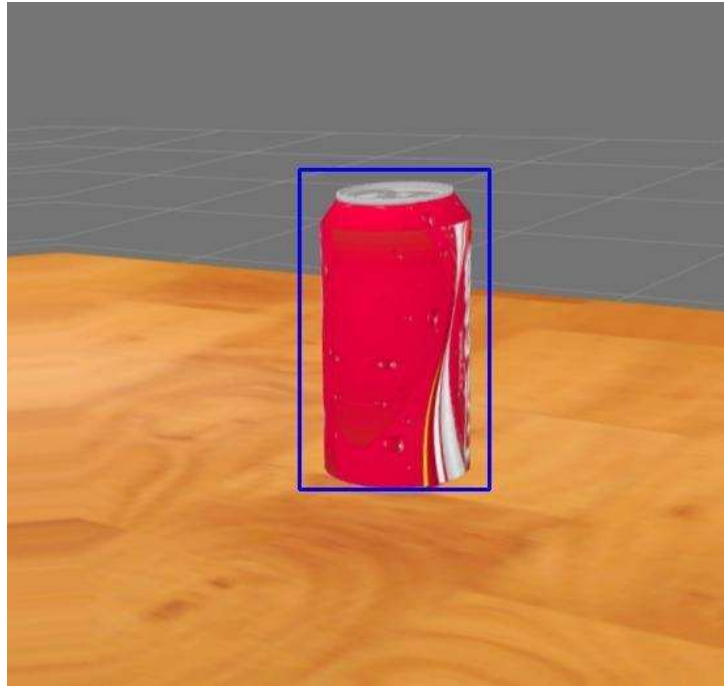


Fig 7.1.2 YOLO coke detection

7.2 ROS arm controllers grip and pick results:

The below image displays the plan generated by PlanSys2, outlining the sequence of actions for the robotic arm:

The workspace was divided into 9 parts, each part having its own coordinates and being treated as a location. The object in question was named as “beer” at the time of the trial run as it had been downloaded from open source models just a few minutes prior. The name has been changed for it now.

The command given was: “Amigos: move beer from position1 to position 5”

```

plan
/tmp
my_server.cpp x domain.pddl x problem.pddl x plan x *Untitled Document 1 x
1 Constructing lookup tables: [10%] [20%] [30%] [40%] [50%] [60%] [70%] [80%] [90%] [100%]
2 Post filtering unreachable actions: [10%] [20%] [30%] [40%] [50%] [60%] [70%] [80%] [90%] [100%]
3 72% of the ground temporal actions in this problem are compression-safe
4 b (5.000 | 5.000)b (4.000 | 10.001)b (3.000 | 10.002)b (2.000 | 10.002)b (1.000 | 15.003);;; Solution
   Found
5 ; Time 0.01
6 0.000: (move arm pos0 pos1) [5.000]
7 5.001: (pick_up arm beer pos1) [5.000]
8 5.002: (move arm pos1 pos5) [5.000]
9 10.003: (place arm beer pos5) [5.000]
  
```

Fig 7.2.1 Plan Generated by PlanSys2

Fig 7.2.2 depicts the initial position of the robotic arm before executing any actions.

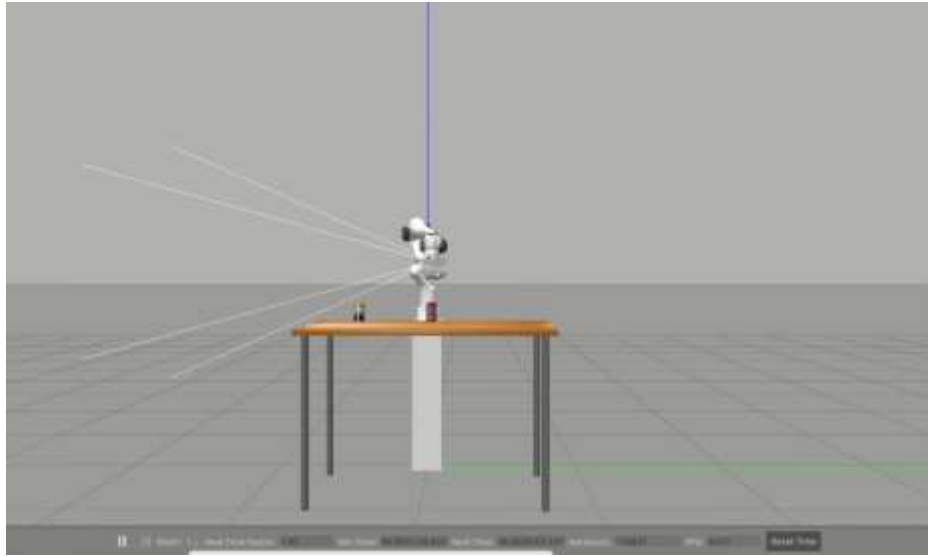


Fig 7.2.2 Initial Position of the Arm

The arm after generating the plan, starts executing it. First step was to Move to pick the object, then the second step was to Pick the object. To Pick the object, the object must be attached to the robot in the simulator as the gazebo virtual world has no concept of friction and hence the object would slip away or get knocked away, treating the contact with the arm as a forbidden collision.

The below image illustrates the process of attaching an object to the end-effector of the robotic arm.



Fig 7.2.3 Attaching Object

Once the object is attached to the robot, the next step in the Pick action is to lift it off the table. Fig 7.2.4 showcases the robotic arm picking up an object from a specified position.



Fig 7.2.4 Picking Object from Position

The third step is to then Move it to the desired location. We keep move operations at a height, so as to avoid colliding with other objects on the table that might come in the way. Once the arm is hovering above the desired location, the last step – Place action starts. Here, the object is gently placed on the table, then detached from the arm and the arm lifts itself back up, leaving the object behind.

Fig 7.2.5 exhibits the robotic arm placing the object it picked up to the designated position.



Fig 7.2.5 Placing Object to Position

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 Conclusion

The AVCAR (Autonomous Voice Controlled Arm Robot) project has achieved its goal of creating a system that can find, pick up, and put down objects based on what users say. We used a bunch of technologies like ROS2 Humble, PlanSys2, MoveIt2, Gazebo, RViz, Speechmatics with Gemini, and YOLOv8 for object detection.

We designed the system carefully, making sure everything works well together. ROS2 Humble helped different parts of the system talk to each other smoothly. PlanSys2 helped with planning, while MoveIt2 made sure the robot arm moved accurately.

We used YOLOv8 to help the robot see and recognize objects, and Speechmatics with Gemini so people could tell the robot what to do. With these tools, AVCAR can understand what users want and do tasks on its own.

After lots of testing, AVCAR can now find objects, pick them up, and put them down just like we wanted. This project opens up possibilities for making robots that can help people in different ways. In the future, we could make AVCAR even better by improving how it finds objects, how it grabs things, and how it understands people when they talk to it.

8.2 Future Enhancement:

The AVCAR (Autonomous Voice Controlled Arm Robot) project has laid a strong foundation, but there's still room for growth and improvement. Here are some areas we can focus on in the future:

1. **Enhanced Object Detection:** We can work on improving how AVCAR sees and recognizes objects. This could involve using more advanced object detection algorithms or fine-tuning YOLOv8 to make it even better at identifying different objects in various environments.
2. **Advanced Grasping Techniques:** We can explore better ways for AVCAR to pick up and hold objects. This might include developing smarter grasping algorithms that adapt to different object shapes and sizes, improving AVCAR's ability to handle delicate or irregular objects.
3. **Improved Natural Language Understanding:** AVCAR can become even more user-friendly by enhancing its understanding of spoken commands. By incorporating advanced natural language processing techniques, we can make AVCAR better at interpreting user instructions and carrying out tasks accurately.
4. **Autonomous Learning and Adaptation:** We can enable AVCAR to learn from its experiences and adapt to new situations. By implementing machine learning algorithms like reinforced learning, AVCAR can become more autonomous over time, refining its performance based on feedback from interactions with users and its environment.
5. **Expanded Applications:** AVCAR's capabilities can be extended to serve a wider range of applications. For example, we could explore how AVCAR can assist in industrial automation tasks, household chores, or healthcare settings. Customizing AVCAR's functionalities to suit different contexts will enhance its versatility and utility.
6. **Enhanced Capabilities:** AVCAR's capabilities can be extended to serve a wider range of applications and do more complex tasks like for example, handling doors, opening and closing water bottles, etc.

REFERENCES

- [1] S. Tasnim, M. Z. Hasan, T. Ahmed, M. T. Hasan, F. J. Uddin and T. Farah, "A ROS-based Voice Controlled Robotic Arm for Automatic Segregation of Medical Waste Using YOLOv3," 2022 2nd International Conference on Computer, Control and Robotics (ICCCR), Shanghai, China, 2022, pp. 81-85, doi: 10.1109/ICCCR54399.2022.9790152.
- [2] S. Gushi, Y. Shimabukuro and H. Higa, "A Self-Feeding Assistive Robotic Arm for People with Physical Disabilities of the Extremities," 2020 5th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Okinawa, Japan, 2020, pp. 61-64, doi: 10.1109/ICIIBMS50712.2020.9336390.
- [3] I. P. Ktistakis and N. G. Bourbakis, "A survey on robotic wheelchairs mounted with robotic arms," 2015 National Aerospace and Electronics Conference (NAECON), Dayton, OH, USA, 2015, pp. 258-262, doi: 10.1109/NAECON.2015.7443079.
- [4] R. A. John, S. Varghese, S. T. Shaji and K. M. Sagayam, "Assistive Device for Physically Challenged Persons Using Voice Controlled Intelligent Robotic Arm," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 806-810, doi: 10.1109/ICACCS48705.2020.9074236.
- [5] D. Calaver, L. Floroian and S. M. Grigorescu, "Assistive Rehabilitation Using a 7-DoF Robotic Arm with Self-Collision and Obstacle Avoidance System," 2019 E-Health and Bioengineering Conference (EHB), Iasi, Romania, 2019, pp. 1-4, doi: 10.1109/EHB47216.2019.8970082.
- [6] R. Melo, T. Pontes de Araújo, A. Andrade Saraiva, J. V. Moura Sousa and N. M. Fonseca Ferreira, "Computer Vision System with Deep Learning for Robotic Arm Control," 2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE), João Pessoa, Brazil, 2018, pp. 357-362, doi: 10.1109/LARS/SBR/WRE.2018.00071.
- [7] S. S. Pradhan, "Design and Development of a Multi-Control Gesture-Recognition based Robotic Arm," 2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET), Shergaon, India, 2019, pp. 100-106, doi: 10.1109/ICITAET47105.2019.9170215.
- [8] S. Hernandez-Mendez, C. Maldonado-Mendez, A. Marin-Hernandez, H. V. Rios-Figueroa, H. Vazquez-Leal and E. R. Palacios-Hernandez, "Design and implementation of a robotic

- arm using ROS and MoveIt!," 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), Ixtapa, Mexico, 2017, pp. 1-6, doi: 10.1109/ROPEC.2017.8261666.
- [9] R. Vermelho and L. A. Alexandre, "Grasping and Sorting Cutlery in an Unconstrained Environment with a 6 DoF Robotic Arm and an RGB+D Camera," 2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Santa Maria da Feira, Portugal, 2022, pp. 3-8, doi: 10.1109/ICARSC55462.2022.9784780.
- [10] R. Yenorkar and U. M. Chaskar, "GUI Based Pick and Place Robotic Arm for Multipurpose Industrial Applications," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 200-203, doi: 10.1109/ICCONS.2018.8663079.
- [11] R. K. Megalingam, N. Katta, R. Geesala, P. K. Yadav and R. C. Rangaiah, "Keyboard-Based Control and Simulation of 6-DOF Robotic Arm Using ROS," 2018 4th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 2018, pp. 1-5, doi: 10.1109/CCAA.2018.8777568.
- [12] Z. Huang, F. Li and L. Xu, "Modeling and Simulation of 6 DOF Robotic Arm Based on Gazebo," 2020 6th International Conference on Control, Automation and Robotics (ICCAR), Singapore, 2020, pp. 319-323, doi: 10.1109/ICCAR49639.2020.9107989.
- [13] L. Sawaged, A. I. Al-Ali, K. S. Hatamleh and M. A. Jaradat, "Modeling and simulation of a moving robotic arm mounted on wheelchair," 2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), Sharjah, United Arab Emirates, 2017, pp. 1-5, doi: 10.1109/ICMSAO.2017.7934876.
- [14] A. R. F. Quiros, A. C. Abad and E. P. Dadios, "Object locator and collector robotic arm using artificial neural networks," 2015 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), Cebu, Philippines, 2015, pp. 1-6, doi: 10.1109/HNICEM.2015.7393209.
- [15] H. Lou, H. Xu and Y. Zhang, "Path Planning of Six-Axis Manipulator Based on ROS System," 2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA), Chengdu, China, 2021, pp. 1063-1066, doi: 10.1109/ICIEA51954.2021.9516087.
- [16] G. -S. Huang, H. -C. Lin and P. -C. Chen, "Robotic arm grasping and placing using edge visual detection system," 2011 8th Asian Control Conference (ASCC), Kaohsiung, Taiwan, 2011, pp. 960-964.

- [17] R. J. Moreno, "Tracking of human operator arms oriented to the control of two robotic arms," 2014 XIX Symposium on Image, Signal Processing and Artificial Vision, Armenia, Colombia, 2014, pp. 1-4, doi: 10.1109/STSIVA.2014.7010125.