

ROS BASED WASTE SEGREGATOR
AN INTERNSHIP REPORT

Submitted by

TRUSHANT PRAFULBHAI ADESHARA

180050131001

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

Computer Science and Engineering

Babaria Institute of Technology, Vadodara



Gujarat Technological University, Ahmedabad

April, 2022



Babaria Institute of Technology and Science

Vernama, Vadodara, Gujarat

CERTIFICATE

This is to certify that the internship report submitted along with the project entitled **ROS based Waste Segregator** has been carried out by **Trushant Prafulbhai Adeshara** under my guidance in partial fulfillment for the degree of Bachelor of Engineering in Computer Science and Engineering, 8th Semester of Gujarat Technological University, Ahmedabad during the academic year 2021-22.

Dr. Avani R. Vasant

Internal Guide

Dr. Avani R. Vasant

Head of the Department



GUJARAT TECHNOLOGICAL UNIVERSITY

CERTIFICATE FOR COMPLETION OF ALL ACTIVITIES AT ONLINE PROJECT PORTAL

B.E. SEMESTER VIII, ACADEMIC YEAR 2021-2022

Date of certificate generation : 04 May 2022 (11:05:01)

This is to certify that, *Adeshara Trushant Prafulbhai* (Enrolment Number - 180050131001) working on project entitled with *ROS based Waste Segregator* from *Computer Science & Engineering* department of *BABARIA INSTITUTE OF TECHNOLOGY, VARNAMA* had submitted following details at online project portal.

Internship Project Report	Completed
---------------------------	------------------

Name of Student : Adeshara Trushant
Prafulbhai

Name of Guide : HOD_005_07

Signature of Student :

TRUSHANT

*Signature of Guide :

Adeshara

Disclaimer :

This is a computer generated copy and does not indicate that your data has been evaluated. This is the receipt that GTU has received a copy of the data that you have uploaded and submitted as your project work.

*Guide has to sign the certificate, Only if all above activities has been Completed.

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Trushant Adeshara, a student of Babaria Institute of Technology has successfully completed his internship in the field of Robotics and Artificial Intelligence from 3rd Jan 2022 to 28th Apr 2022 (Total number of weeks: 12) under the guidance of the Co-Founder Mr. Rishabh Shah.

His internship activities included designing the architecture for the Robotics Operating System, computer vision software optimization and embedded integration.

During his course of internship, he has been exposed to different processes and was found diligent, hardworking and inquisitive.

We wish him every success in life and career.

For Wastefull Insights Private Limited
Manali Agarwal
Co-Founder





BITS
Edu Campus
Vadodara



Babaria Institute of Technology and Science

Vernama, Vadodara, Gujarat

DECLARATION

We hereby declare that the Internship report submitted along with the Project entitled **ROS Based Waste Segregator** submitted in partial fulfillment for the degree of Bachelor of Engineering in Computer Science and Engineering to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me at Wastefull Insights under the supervision of Dr. Avani R. Vasant and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of the Student

TRUSHANT ADESHARA

Sign of Student

TRUSHANT

Acknowledgement

First I would like to thank Ms. Manali Agarwal, Co-Founder, of Wastefull Insights for giving me the opportunity to take up an internship within the organization.

I wish to express my sincere gratitude to my external guide Mr. Rishabh Agarwal, Co-Founder, of Wastefull Insights for continuously guiding me at the company and helping me through all my doubts.

I also would like to thank all the people that worked along with me at Wastefull Insights. With their patience and openness they created an enjoyable working environment. It is indeed with a great sense of pleasure and immense sense of gratitude that I acknowledge the help of these individuals.

I am highly indebted to my internal guide and HOD Dr. Avani R. Vasant for helping me through my internship by giving me the necessary suggestions and advice along with her valuable coordination in completing this internship.

I am extremely grateful to my department, staff members, family and friends who helped me in successful completion of this internship.

Trushant Adeshara
180050131001

Abstract

Internships are considered to be windows into the real world for university students as they provide hands-on experience in a given field. With the boom in the robotics and AI industry it becomes very essential to understand the market, client and practical use-cases of skills learned at the university.

At Wastefull Insights, students from across India can experience what it is like to be part of the movement "Zero Waste World" with the help of robotics and AI. My project emphasized on optimization of pick and place waste segregation robots. With the increase in population, more and more plastic waste is generated every day. It is not practically feasible or environmentally viable to place this waste in landfills. Hence, recycling and upcycling are the only solutions. With the increased throughput of the system, more waste can be segregated in less amount of time, leading to lower installation and maintenance cost of the recycling company with improved performance of the robot.

During my internship period I restructured underlying ROS architecture to make it more flexible, modular and robust. Also, developed an Operator's UI with which one can control both robot and camera detection. In addition to these tasks, I developed a velocity estimation algorithm to increase overall throughput of the robot.

While developing optimization algorithms for pick and place sequence selection, I gained exposure on how to study research papers, extract different approaches from them and implement the one which is the best suited solution. I got in-depth insights on not only theoretical development of algorithms but also programming them in the most efficient manner with constrained resources.

In conclusion, this was an opportunity to develop and enhance skills and competencies in my career field which I actually achieved.

List of Figures

Fig 1.1	Organization Chart
Fig 2.1	Per Capita Plastic Waste Generation
Fig 2.2	Per Capita State/UT Wise Plastic Waste Generation
Fig 2.3	Gantt Chart (Jan 2022)
Fig 2.4	Gantt Chart (Feb 2022)
Fig 2.5	Gantt Chart (Mar 2022)
Fig 2.6	Kanban Board
Fig 3.1	ROS1 and ROS2 Architecture
Fig 3.2	ROS Publisher Subscriber Mechanism
Fig 3.3	ROS Service Mechanism
Fig 3.4	ROS Action Mechanism
Fig 3.5	ROS CvBridge Mechanism
Fig 3.6	ROS Bridge Architecture
Fig 3.7	ROS Serial Architecture
Fig 3.8	CUDA Architecture
Fig 3.9	Jetson Nano and Jetson NX SBC
Fig 3.10	TM4C1294 Microcontroller by TI
Fig 3.11	Energia IDE Interface
Fig 3.12	Cross Compiler
Fig 3.13	ArUco Marker Of Different Dictionary Type
Fig 3.14	Pose Estimation With ArUco Marker
Fig 4.1	PnP Time Profile
Fig 4.2	OPTSEQ Algorithm
Fig 4.3	OPTSEQDP Algorithm
Fig 4.4	SUBOPTDP Algorithm
Fig 4.5	Comparison Of Different PnP Algorithms
Fig 5.1	Initial System Design
Fig 5.2	Restructured System Design
Fig 5.3	System Design Flowchart
Fig 5.4	Initial State Transition Diagram
Fig 5.5	Optimized State Transition Diagram
Fig 5.6	Operator's UI Design

Fig 6.1	Jetson Nano Home Interface After Boot
Fig 6.2	CMAKE GUI Console
Fig 6.3	ROS Melodic Logo
Fig 6.4	Pylon SDK Console
Fig 6.5	Pylon IP Configuration Tool
Fig 6.6	ROS Talker And Listener Node
Fig 6.7	Pub Sub Computational Graph in rqt_graph
Fig 6.8	ROS Service For Adding Two Integers
Fig 6.9	ROS Core Execution
Fig 6.10	ROS Fibonacci Action Server
Fig 6.11	ROS Fibonacci Action Client
Fig 6.12	ROS Fibonacci Action Server Feedback
Fig 6.13	ROS TurtleSim And ROS Core
Fig 6.14	ROS Teleop
Fig 6.15	Differential Drive Robot (RVIZ)
Fig 6.16	Differential Drive Robot (Gazebo Environment)
Fig 6.17	Apt Source List for Jetson
Fig 6.18	Workspace Structure For ROS
Fig 6.19	Package Structure For ROS
Fig 6.20	Setup.py File
Fig 6.21	Check Installation Of GCC And Cross Compiler
Fig 6.22	Bash Script For Tiva Setup
Fig 6.23	Tivac ROS Publisher
Fig 6.24	Camera Calibration Procedure
Fig 6.25	ArUco Marker Generation Script
Fig 6.26	ArUco Marker Detection Script Directory
Fig 6.27	ArUco Marker Pose Estimation
Fig 6.28	Conveyor Velocity Estimation With ArUco Marker
Fig 6.29	Operator's UI

List of Tables

Table 4.1 Comparison Between Different Pick And Place Algorithms.....

List of Abbreviations

3D	3 Dimensional
ADC	Analog to Digital Converter
AI	Artificial Intelligence
AMR	Autonomous Mobile Robot
API	Application Programming Interface
ARM	Advanced RISC Machine
ArUco	Augmented Reality University of Cordoba
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network Library
DB	Data Base
DDS	Data Distribution Service
DNS	Domain Name System
EOL	End Of Life
FIFO	First In First Out
FPU	Floating Point Unit
GNU	GNU's not Unix
GPIO	General Purpose Input / Output
GPU	Graphics Processing Unit
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
I/O	Input / Output
IP	Internet Protocol
JS	JavaScript

JSON	JavaScript Object Notation
LiDAR	Light Detection and Ranging
MATLAB	Matrix Laboratory
ML	Machine Learning
MSP	Mixed Signal Microcontroller
OpenCV	Open Source Computer Vision Library
OPTSEQ	Optimal Sequence
OPTSEQDP	Optimal Sequence Dynamic Programming
PnP	Pick n Place
PPA	Personal Package Archive
RISC	Reduced Instruction Set Computer
ROI	Region Of Interest
ROS	Robot Operating System
RT	Real Time
RVIZ	ROS Visualization
SBC	Single Board Computer
SCARA	Selective Compliance Assembly Robot Arm
SDK	Software Development Kit
SPT	Shortest Processing Time
TCP	Transmission Control Protocol
TI	Texas Instruments
TF	Transform Frame
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
UI	User Interface

USB	Universal Serial Bus
UT	Union Territory
UUID	Universally Unique Identifier
VCS	Version Control System
XML	eXtensible Markup Language

Table of Contents

Acknowledgement	i
Abstract	ii
List of Figures	iii
List of Tables	v
List of Abbreviations	vi
Table of Contents	ix
Chapter 1 Overview of Company	1
1.1 Introduction	1
1.2 History.....	1
1.3 Scope of Work.....	1
1.4 Organization Chart.....	2
Chapter 2 Introduction to Internship and Internship Management.....	3
2.1 Internship Summary	3
2.2 Purpose	4
2.3 Objective	4
2.4 Scope	5
2.5 Internship Planning	5
2.5.1 Internship Development Approach and Justification	5
2.5.2 Internship Effort and Time	7
2.5.3 Roles and Responsibilities	8
2.5.4 Group Dependencies	8
2.6 Internship Scheduling	8
Chapter 3 Technology Stack and Hardware Platform	10
3.1 Programming Languages	10
3.1.1 C Language	10
3.1.2 C++ Language	10
3.1.3 Python	10

3.1.4	Javascript	11
3.2	Robotics Software Development Framework (ROS)	11
3.2.1	ROS Architecture	12
3.2.2	ROS Filesystem Level	13
3.2.3	ROS Computation Graph Level	14
3.2.4	ROS Publisher Subscriber Mechanism	15
3.2.5	ROS Service Mechanism	16
3.2.6	ROS Action Mechanism	16
3.2.7	ROS CvBridge	17
3.2.8	ROS Bridge	18
3.2.9	ROS Library JS	19
3.2.10	ROS Serial Library	19
3.3	Build System and Build Tool	20
3.3.1	Build System	20
3.3.2	Build Tool	21
3.4	Git Version Control System	21
3.5	CUDA Architecture	22
3.6	NumPy Library	23
3.7	Python UUIDs Library	23
3.8	OpenCV Library	23
3.9	Machine Learning	24
3.10	Kalman Filters	24
3.11	Edge Computing Device (SBC)	24
3.12	Embedded Board	26
3.13	Energia IDE	26
3.14	Cross Compiler	27
3.15	ArUCo Marker	28
Chapter 4	System Analysis	30
4.1	Study of Current System	30

4.2 Problem and Weaknesses of Current System	30
4.3 Requirements of New System	31
4.4 Features of New System	33
4.5 Literature Review of Optimization Algorithms	33
4.5.1 Exhaustive Search Method	34
4.5.2 Local Augmentation Method	35
4.5.3 Experimental Results by Researcher	36
4.6 Selection of Optimization Approaches and Justification.....	37
Chapter 5 System Design.....	39
5.1 System Design and Methodology.....	39
5.2 Input / Output and Interface Design	42
5.2.1 State Transition Diagram.....	42
5.2.2 Operator’s UI Interface Design.....	43
Chapter 6 Implementation.....	44
6.1 Implementation Platform and Environment Setup.....	44
6.1.1 Jetson SBC Setup.....	44
6.1.2 OpenCV 3 Installation	45
6.1.3 ROS Melodic Installation	46
6.1.4 Pylon SDK Installation and Setup	46
6.2 ROS Basics Examples	48
6.2.1 ROS Publisher Subscriber	48
6.2.2 ROS Service	49
6.2.3 ROS Action	50
6.2.4 ROS TurtleSim	52
6.2.5 ROS RVIZ AMR	53
6.2.6 ROS Gazebo AMR	53
6.3 Updating apt-source list	54
6.4 ROS Workspace and Package Configuration	55
6.5 TM4C1294 Integration with ROS Serial	57

6.5.1 TivaWare SDK and Cross Compiler Setup	57
6.5.2 Automation Script for Tiva Setup	58
6.5.3 TM4C1294 Setup and Test	58
6.6 Conveyor Velocity Estimation with ArUco Marker	59
6.6.1 Camera Calibration	59
6.6.2 ArUco Marker Generation and Detection	60
6.6.3 Conveyor Velocity Estimation	62
6.7 Operator’s UI Integration with ROS	62
6.8 Modified ROS Software Architecture	64
Chapter 7 Testing.....	65
7.1 Testing Plan	65
7.2 Test Result and Analysis.....	65
7.2.1 Test Cases.....	65
Chapter 8 Conclusion and Discussion.....	67
8.1 Result and Analysis of Project Viabilities	67
8.2 Workflow Problems Encountered and Possible Solutions.....	67
8.3 Summary / Conclusion of Internship	68
8.4 Limitation and Future Enhancement	69
References	70

Chapter 1

Overview of the Company

1.1 Introduction

Wastefull Insights is a deep tech startup based out of Vadodara focusing on developing AI driven solutions for waste segregation. It focuses on a simple vision to make a zero waste world. It has an accompanying e-commerce Platform “Achhe Dinn” which is India’s first Platform to encourage recycled, upcycled and sustainable products. Its main objectives include promoting Source Segregation, Circular Economy and automating Dry Waste Segregation in the country.

1.2 History

Wastefull Insights was founded by two Co-Founders Ms. Manali Agarwal and Mr. Rishabh Shah in the year 2020. It began with AI driven waste segregation solution and later on an e-commerce platform “Achhe Dinn” which is India’s first platform to encourage recycled, upcycled and sustainable products emerged.

1.3 Scope of Work

All the segments from AI to Robotics but not limited to it are under scope of Wastefull Insights as long as they are aiming to zero waste world. The current product being delivered is a conveyor based pick and place waste segregator which uses AI for waste segregation. The robotic automation system comes with hardware and associated software which can be used out of box with no or minimum modification according to client’s requirement.

1.4 Organization Chart

Organization chart of the company is as shown in Fig 1.1

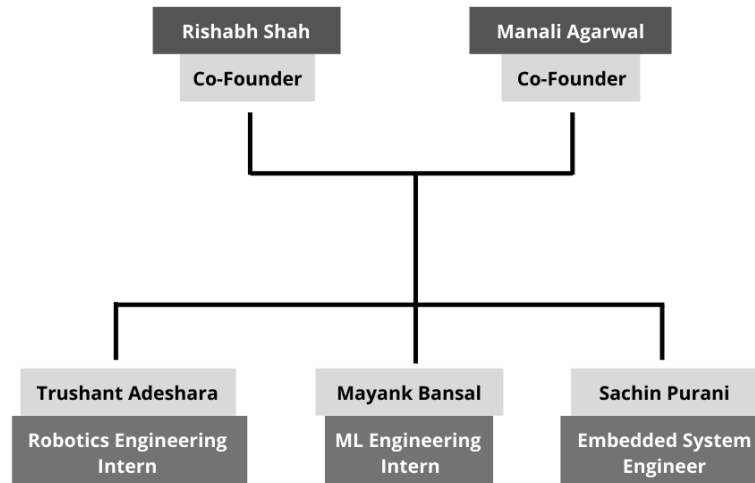


Fig 1.1 Organization Chart

Chapter 2

Introduction to Internship and Internship Management

2.1 Internship Summary

I always knew that plastic waste management is an issue but never really understood the gravity of the situation until joining Wastefull Insights. In an effort to assess the magnitude of plastic waste being generated, I came across some statistics as shown in Fig 2.1 and 2.2.

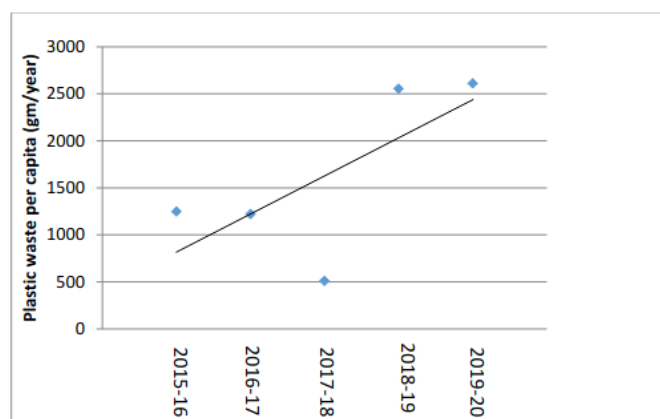


Fig 2.1 Per Capita Plastic Waste Generation

“Courtesy of (Central Pollution Control Board 2021)”

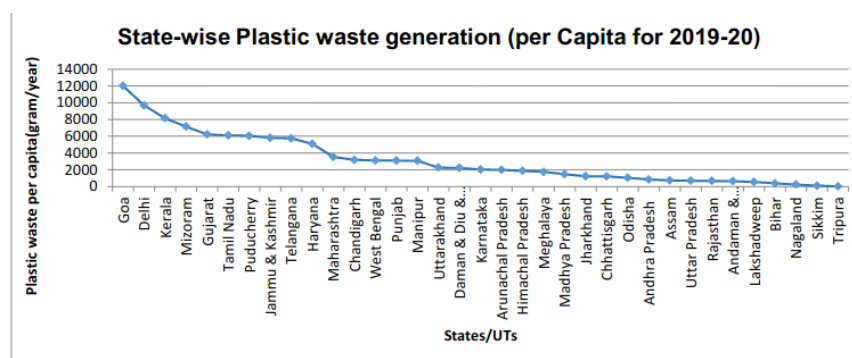


Fig 2.2 Per Capita State/UT Wise Plastic Waste Generation

“Courtesy of (Central Pollution Control Board 2021)”

This convinced me about the severity of the problem of plastic waste and the necessity of finding a solution before it's too late. Automation is the only way out.

With the help of AI and robotics it is now possible to recycle waste by segregation. This will help in sustaining life on planet earth. I joined Wastefull Insights to contribute to this cause.

Developing a pick and place optimization algorithm to improve throughput of the system was my primary goal. But considering the startup environment where one is not bound to a specific task or role, I was assigned tasks from different domains. Entire robot works on the ROS framework and I had to ROSify already existing functionality including integrating the embedded system.

For optimization I worked on restructuring software architecture of the robot in ROS and also customized computer vision software. One of the minor tasks was to integrate the UI interface with ROS for receiving live system updates and controlling the operator of the robot.

2.2 Purpose

To optimize performance of pick and place waste segregation robots so that maximum segregation can be ensured in least amount of time.

2.3 Objective

Internships are considered to be windows into the real world for university students as they provide hands-on experience in a given field. With the boom in the robotics and AI industry it becomes very essential to understand the market, client and practical use-cases of skills learned in university.

At Wastefull Insights, students from across India can experience what it is like to be part of the movement "Zero Waste World" with the help of robotics and AI. Hence, this internship enables students to broaden their horizons and look forward to pursuing a career in the field of robotics and AI.

2.4 Scope

Considering Wastefull Insights is a startup there were limitless opportunities to learn and grow, not only in the area for which one is interning but in the overall development of the product. As aforesaid, considering the startup environment all the decisions were made as a group and the team was always open for suggestions and criticism for improving performance of the robot.

Apart from designing software architecture of pick and place optimization, I was able to learn object detection with machine learning using computer vision, embedded system integration with ROS and Front-end UI integration with ROS.

2.5 Internship Planning

2.5.1 Internship Development Approach and Justification

As Wastefull Insights is a startup, tasks to be accomplished are dynamic in nature and depending on the current requirement major tasks went in the background and were resumed once foreground tasks were providing satisfactory results with passing all the test cases.

Though the marked period for the internship which is recorded in this report is for 12 weeks. My tenure as an intern at Wastefull Insights sums up after 21 weeks. Tasks below are overviews for those 12 weeks.

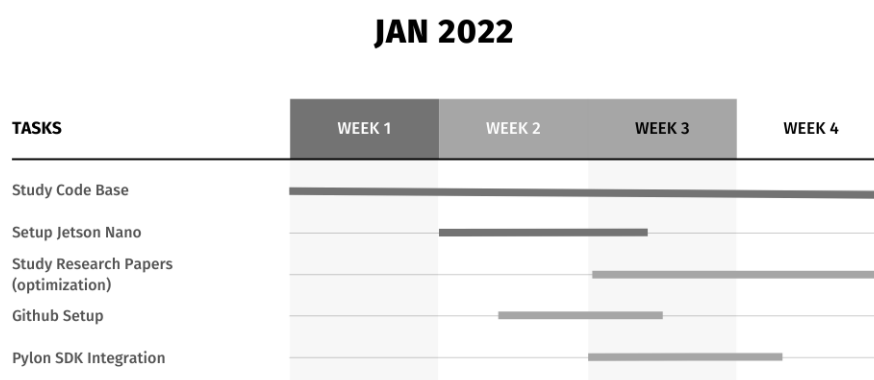


Fig 2.3 Gantt Chart (Jan 2022)

Major task of the first month, that is January, was to understand the code base of the company. Wastefull Insights has been working on the development of a waste segregator robot for about 2 years now and in that time there has been rapid development in the software architecture of the system. Considering the density of the code base, emphasis would be to go through the flow of the system and understand existing software architecture for the robot.

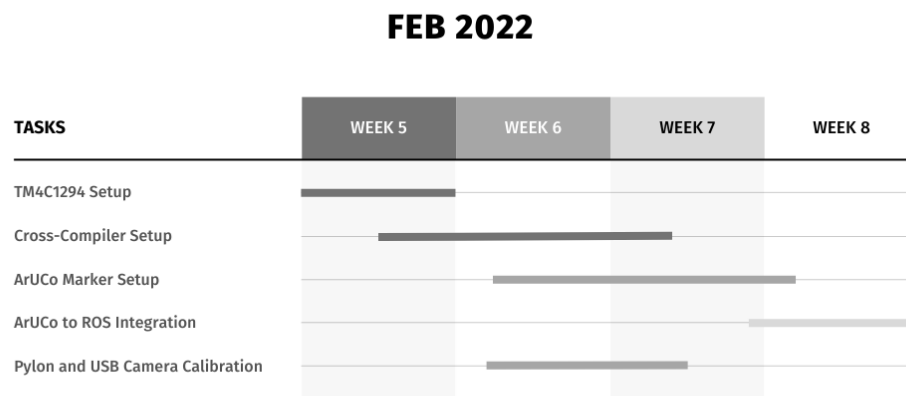


Fig 2.4 Gantt Chart (Feb 2022)

In the month of February after understanding the existing system, my next task was to test different optimization algorithms and implement them into the existing system while simultaneously studying its pros and cons from one which is already implemented. One of the major considerations is the number of picks that the robot can perform and avoid sub-optimally due to local maxima problem. Also, there were some minor tasks to accomplish this month which included integration of the embedded system with ROS serial instead of relying on pure UART communication. This will help in providing more flexibility and control on the robot.

Another task which was a bit impromptu was maker based velocity estimation of the conveyor belt. In the vision based system frame rate of camera and conveyor speed are two important parameters which

drastically impact the performance of the robot. Hence, I had to tie up all the loose ends and make the system as closed loop as possible.

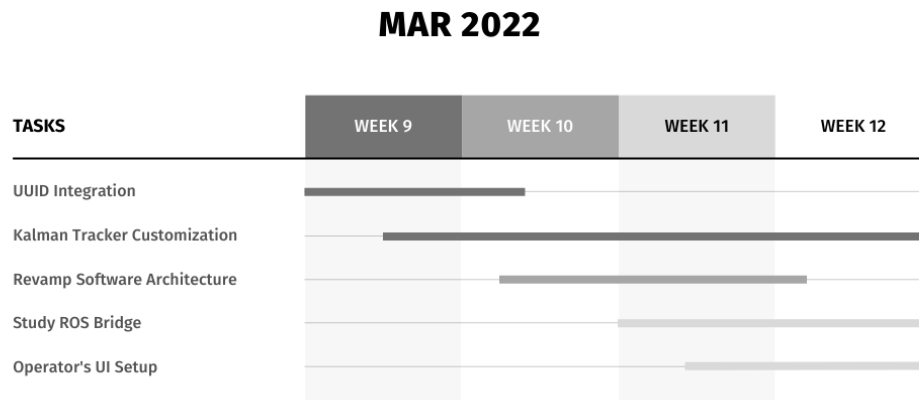


Fig 2.5 Gantt Chart (Mar 2022)

Month of March was about implementation of scheduler algorithms and revamping entire software architecture to make it more flexible and modular. While this was being carried out, I also worked on integrating operator's UI frontend with ROS backend.

2.5.2 Internship Effort and Time

Duration for the internship was about 8-10 hours per day. It would start around 9.30 a.m. and some days might even go till 8.00 p.m. Sundays were off unless some important tasks were pending from the previous week. I was averaging near to 60 hours per week.

Initially the workload was less while I was understanding the system, but later on I had to push myself to explore different options for the same task because the robot was implemented in a resource constrained system which gave me exposure to real-world situations.

2.5.3 Roles and Responsibilities

My major role throughout the internship was as a Robotics Engineering Intern in which I not only had the opportunity to explore the software side of robots, but other areas like Machine Learning, Embedded Systems and a little bit of Web Development. Basically any kind of integration that is supposed to be done with ROS would come under my responsibilities.

2.5.4 Group Dependencies

I was working with people from different parts of the world and as I was integrating modules from different sub-systems, my task was group dependent. For ML related tasks I was working with another intern from Kolkata and for Embedded System I was working with a Full Time Developer. In all those tasks I was continuously guided by Mr. Rishabh Shah and we would have brainstorming sessions on how to optimize the system.

2.6 Internship Scheduling

At the start of every week on Monday we would have a sync up meeting in which everyone would explain what they did last week and what their tasks are for the next week. So, at the start of the day there was a general idea about what will be the major task of the day. Now the next task was to break it down into smaller sub-tasks and generate tickets for them on Github Kanban board. Fig 2.6 is a snapshot of one such instance of Kanban Board.

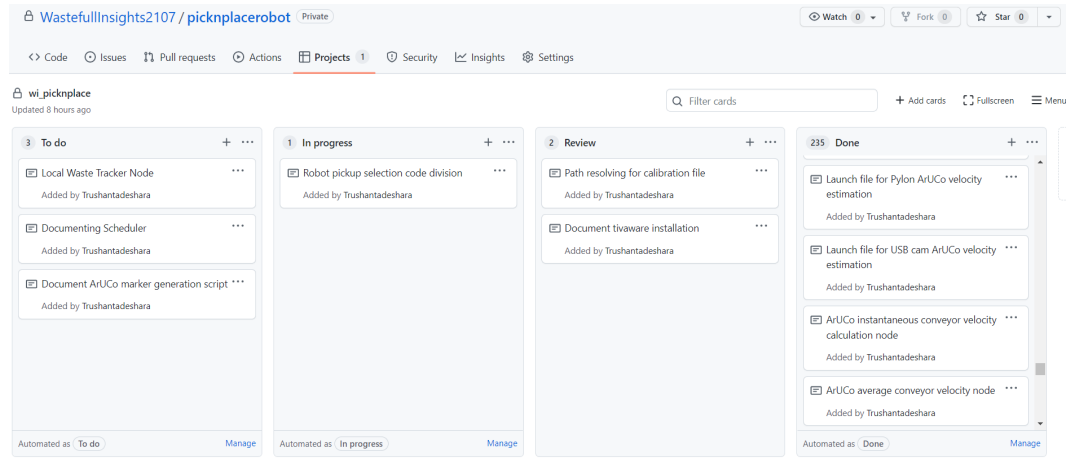


Fig 2.6 Kanban Board

Initially the tasks were in “To do”, from there they headed on to “In progress”. Once completed they would go into “Review” from where Mr. Rishabh Shah would look into it and provide suggestions and necessary modification. At the end that ticket would go to “Done”.

Chapter 3

Technology Stack and Hardware Platform

3.1 Programming Languages

3.1.1 C Language

C is a powerful general-purpose programming language. It can be used to develop software like operating systems, databases, compilers, and so on. One of the major uses for C is to develop firmware for embedded systems (Programiz, n.d.).

ROS framework is entirely built on C in the backend with a library called rcl. It is very essential to have knowledge of C to understand the working of ROS.

3.1.2 C++

C++ is a cross-platform language that can be used to create high-performance applications. It is an extension to the C language. C++ gives programmers a high level of control over system resources and memory. Additionally, C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs (“C++ Introduction”, n.d.).

When developing in ROS C++ provides roscpp API with an abstraction to implement Publishers, Subscribers, Services and Actions which are basic building blocks of ROS based systems. It also allows optimizing AI algorithms and increasing their performance by using GPU.

3.1.3 Python

Python is a high-level, general-purpose programming language that is interpreted. The use of considerable indentation in its design philosophy emphasizes code readability. Its language elements and object-oriented

approach are aimed at assisting programmers in writing clear, logical code for both small and large-scale projects (van Rossum, n.d.).

Python is garbage-collected and dynamically typed. It supports a variety of programming paradigms, including structured (especially procedural) programming, object-oriented programming, and functional programming. Because of its extensive standard library, it is often referred to as a “batteries included” language.

One of the major advantages of using python is that it is supported in ROS with rospy API which makes implementation of nodes in ROS really simple and with python development time for robotics software reduces drastically. Also, implementing AI models is much easier in python than C.

3.1.4 Javascript

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat (“JavaScript | MDN” 2022).

JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative styles. In ROS there is rosbridge which performs serialization of ROS messages from C type to corresponding JSON. After the message is converted to JSON it can be sent to Web UI, database or backend server via web socket.

3.2 Robotics Software Development Framework (ROS)

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms.

With powerful developer tools, ROS is open source (Open Robotoc, n.d.). A detailed preview of ROS with its capabilities are explained in upcoming chapters.

3.2.1 ROS Architecture

There are basically 2 versions of ROS, ROS1 and ROS2. ROS1 mainly supports Linux-based operating systems while on the other hand ROS2 provides more flexibility by supporting Linux, Windows, Mac and RTOS. Transport protocol used in ROS1 is TCPROS/UDPROS, and the communication is highly dependent on the operation of the Master node. Hence, one can say that ROS1 is centralized in nature and there is a single point of failure. In the case of ROS2 DDS is used for communication which enhances fault tolerance capabilities (Vankeirsbilck 2020).

In my internship I am using ROS1 because Jetson Compute Boards use Ubuntu 18.04 as their base operating system and ROS2 is not fully mature for this OS.

Individual entities in ROS are called Nodes and on the basis of the nature of those nodes, it is possible to bundle them in the form of a package. There are even multiple tools provided by ROS which help in debugging these nodes and as ROS is open source there is a huge community to help developers. Owing to the above reason ROS is becoming the new industry standard.

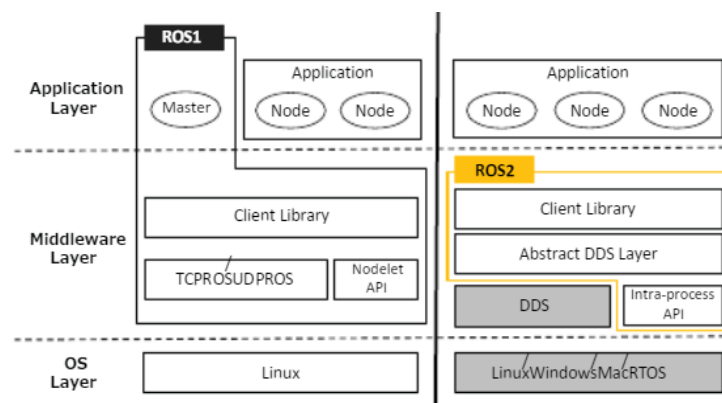


Fig 3.1 ROS1 And ROS2 Architecture

“Courtesy of (Y. Maruyama, S. Kato, and T. Azumi 2016)”

3.2.2 ROS Filesystem Level

The filesystem level (“ROS/Concepts - ROS” 2014) mainly covers those ROS resources that are encountered on disks of the device running ROS, such as:

- **Packages:**
Packages are the main unit for organizing software in ROS. A package may contain ROS runtime process (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages can be considered as the most atomic build item and release item in ROS.
- **Metapackages:**
Metapackages are specialized packages which only serve to represent a group of related other packages. These packages are used as a backward compatible placeholder for converted rosbuilt Stacks.
- **Package Manifests:**
Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.
- **Repositories:**
A collection of packages which share a common VCS system. Packages which share a VCS share the same version and can be released together using the catkin release automation tool bloom.
- **Message (msg) types:**
Message descriptions, stored in my_package/msg/type.msg, define the data structure for messages sent in ROS.
- **Service (srv) types:**
Service descriptions, stored in my_package/srv/type.srv, define the request and response data structure for services in ROS.

3.2.3 ROS Computation Graph Level

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags (“ROS/Concepts - ROS” 2014), all of which provide data to the Graph in different ways.

- **Nodes:**
Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. A ROS node is written with the use of ROS client libraries, such as roscpp or rospy.
- **Master:**
The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Server:**
The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.
- **Messages:**
Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays and other primitive types.
- **Topics:**
Messages are routed via a transport system with publish / subscribe semantics. The topic is a name that is used to identify the content of the message.

- Services:

The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transform is not appropriate for request / reply interactions, which are often required in distributed systems. This request / reply is carried out with the help of service.

- Bags:

Bags are a format for saving and playing back ROS message data.

3.2.4 ROS Publisher Subscriber Mechanism

A Publisher node is a program that publishes data like a camera captures frames and then transmits it. A Subscriber is a program that subscribes to published data. Nodes communicate with each other by passing messages via named topics and those nodes can find each other with the help of ROS Master (Sears 2020).

There can be multiple publishers to the same topic like velocity commands given to a robot and there can also be multiple subscribers to the same topic like an image frame from a camera is used by different ML models running at the same time.

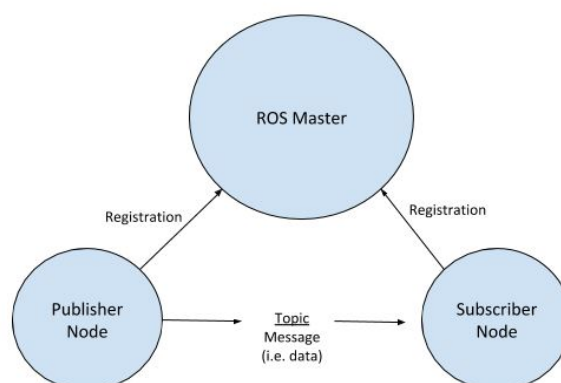


Fig 3.2 ROS Publisher Subscriber Mechanism

“Courtesy of (Sears 2020)”

3.2.5 ROS Service Mechanism

A ROS Service consists of a pair of messages: one for the request and one for the reply. A service-providing ROS node (i.e. Service Server) offers services like reading sensor data).

A client node (i.e. Service Client) calls the service by sending a request message to the service provider. The client node then awaits the reply. In ROS, a service is defined using .srv files (Sears 2020).

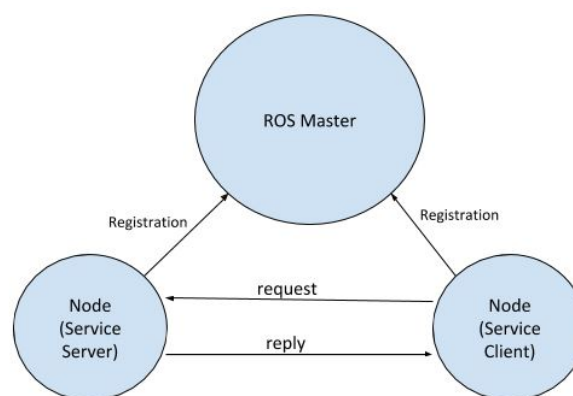


Fig 3.3 ROS Service Mechanism
“Courtesy of (Sears 2020)”

3.2.6 ROS Actions Mechanism

ROS Actions have a client-to-server communication relationship with a specified protocol. The actions use ROS topic to send goal messages from a client to the server. One can cancel goals using the action client. After receiving a goal, the server processes it and can give information back to the client. This information includes the status of the server, the state of the current goal, feedback on that goal during operation, and finally a result message when the goal is complete (“ROS Actions Overview - MATLAB & Simulink” 2022).

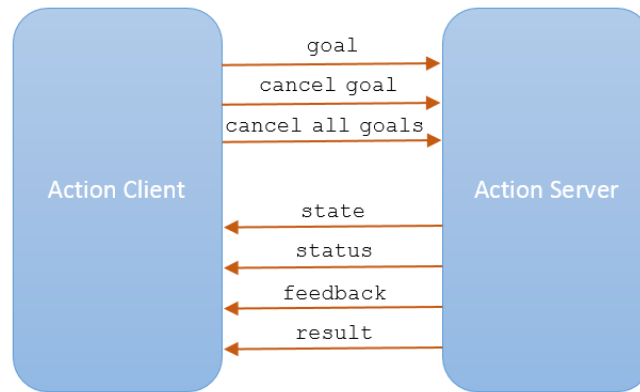


Fig 3.4 ROS Action Mechanism

“Courtesy of (“ROS Actions Overview - MATLAB & Simulink” 2022)”

3.2.7 ROS CvBridge

ROS passes around images in its own `sensor_msgs/Image` message format, but many users will want to use images in conjunction with OpenCV. CvBridge is a ROS library that provides an interface between ROS and OpenCV (Bowman and Mihelich 2010).

While converting ROS image messages to OpenCV images, the input is the image message, as well as an optional encoding. The encoding refers to the destination `cv::Mat` Image. If the default value of “passthrough” is given, the destination image encoding will be the same as the image message encoding.

CvBridge will optionally do color or pixel depth conversions as necessary. To use this feature, one needs to specify the encoding to be one of the following strings:

- `mono8`: CV_8UC1, grayscale image
- `mono16`: CV_16UC1, 16-bit grayscale image
- `brg8`: CV_8UC3, color image with blue-green-red color order
- `rgb8`: CV_8UC3, color image with red-green-blue color order
- `brga8`: CV_8UC4, BGR color image with an alpha channel
- `rgba8`: CV_8UC4, RGB color image with an alpha channel

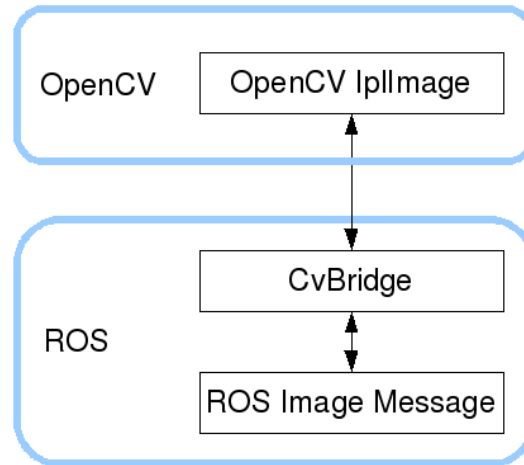


Fig 3.5 ROS CvBridge Mechanism

“Courtesy of (Bowman and Mihelich 2010)”

3.2.8 ROS Bridge

There are a variety of front ends that interface with rosbriidge, including a WebSocket server for web browsers to interact with.

The rosbriidge_suite package is a collection of packages that implement the rosbriidge protocol and provides a WebSocket transport layer (Mace 2017).

The packages include:

- **rosbriidge_library:**
It contains the core rosbriidge package which are responsible for taking the JSON string and sending the commands to ROS and vice versa.
- **rosapi:**
It makes certain ROS actions accessible via service calls that are normally reserved for ROS client libraries. This includes getting and setting params, getting topics list, and more.
- **rosbriidge_server:**

While `rosbridge_library` provides the JSON - ROS conversion, it leaves the transport layer to others. `Rosbridge_erver` provides a WebSocket connection so browsers can “talk rosbridge”.

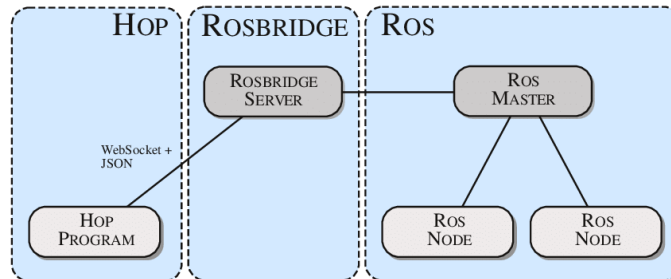


Fig 3.6 ROS Bridge

“Courtesy of (Marcin et al. 2015)“

3.2.9 ROS Library JS

The `roslibjs` is the core JavaScript library for interacting with ROS from the browser. It uses WebSockets to connect with `rosbridge` and provides publishing, subscribing, service calls, `actionlib`, TF, URDF parsing, and other essential ROS functionality. `RoSlibjs` is developed as part of the Robot Web Tools (“`roslibjs`” 2019).

3.2.10 ROS Serial Library

The `rosserial` is a protocol for wrapping standard ROS serialized messages and multiple topics and services over a character device such as a serial port or network socket (“`rosserial`” 2018).

Client libraries allow users to easily get ROS nodes up and running on various systems. These clients are ports of the general ANSI C++ `rosserial_client` library. Currently, these packages are included:

- `rosserial_arduino`
- `rosserial_embeddedlinux`
- `rosserial_windows`
- `rosserial_mbed`

- roserial_tivac
- roserial_stm32
- ros-teensy

Devices running roserial code require a node on the host machine to bridge the connection from the serial protocol to the more general ROS network. Following are two such interfaces:

- roserial_python:
A Python-based implementation (for PC usage)
- roserial_server:
A C++ implementation based on the ShapeShifter message, some limitations compared to roserial_python but recommended for high-performance applications.

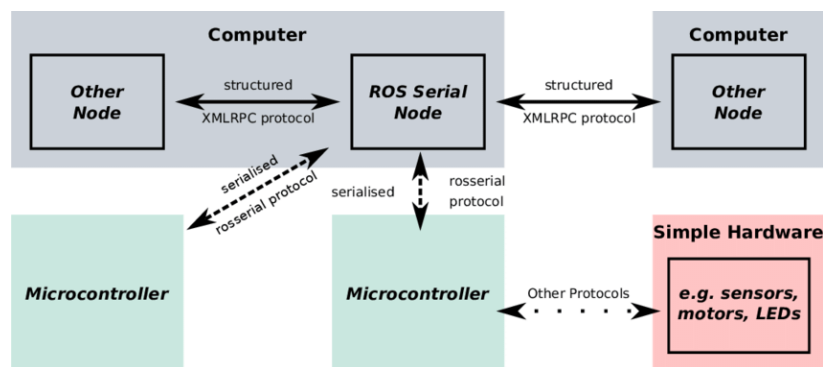


Fig 3.7 ROS Serial Architecture

“Courtesy of (Andrew et al. 2018)”

3.3 Build System and Build Tool

ROS uses both build system and build tool to manage, find and build packages in a catkinized manner. Thus, it is important to know about them.

3.3.1 Build System

Build automation involves scripting or automating the process of compiling computer source code into binary code. ROS uses CMake as a build system which is built on top of Make, a classic Unix build tool (“List of build automation software”, n.d.).

3.3.2 Build Tool

Build tools are programs that automate the creation of executable applications from source code. Building incorporates compiling, linking and packaging the code into a usable or executable form.

In small projects, developers will often manually invoke the build process. But it is not practical for large projects, as it is very hard to keep track of what needs to be built, in what sequence and what dependencies there are in the building process. Using an automation tool allows the build process to be more consistent (“What is Build Tool? - Definition from Techopedia” 2011).

In ROS catkin is the most widely used build tool. Catkin provides `catkin_make` to build workspace. Another widely used build tool is `catkin build` which makes more compartmentalized environments.

3.4 Git Version Control System

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency (Git, n.d.).

When developing with ROS every sub-system development takes place in the form of packages. And when more developers and features are there in the system it is very critical to manage all the packages up to date. To resolve this issue, the git system helps in distributing all the packages in the form git repositories.

3.5 CUDA Architecture

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units [GPUs]. With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.

In GPU-accelerated applications, the sequential part of the workload runs on the CPU - which is optimized for single-threaded performance - while the compute intensive portion of the application runs on thousands of FPU cores in parallel.

We are using Jetson lineup boards which allow us to use CUDA on Nvidia's Tegra GPU for accelerating vision algorithms. PyCUDA gives easy, Pythonic access to Nvidia's CUDA parallel computation API (Nvidia, n.d.).

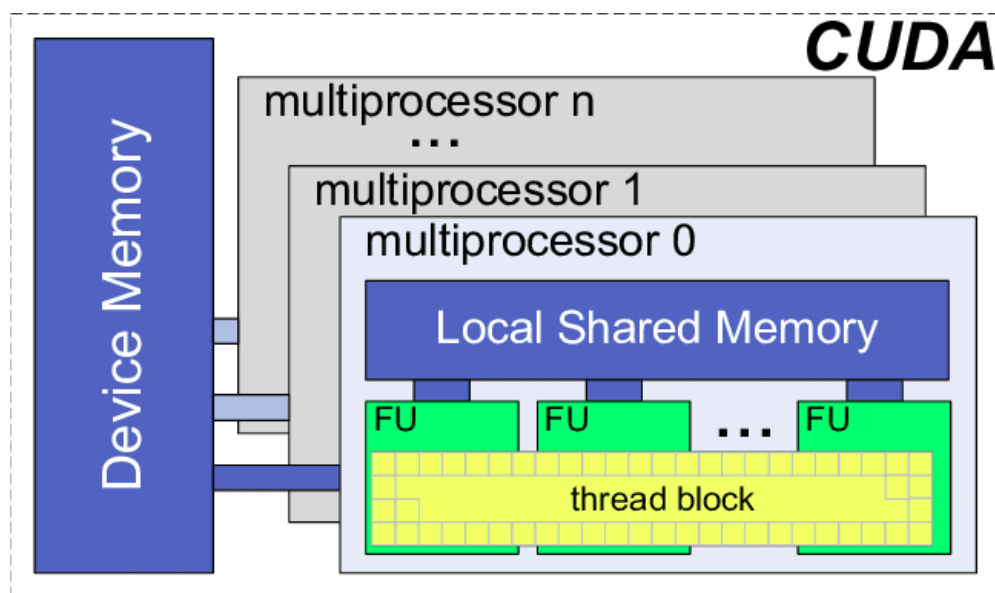


Fig 3.8 CUDA Architecture

“Courtesy of (Debapriya, Andrew, and Valeria 2009)“

3.6 NumPy Library

NumPy (Numerical Python) is an open source Python Library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems ("NumPy Library", n.d.).

NumPy is the fundamental package for scientific computing in Python. It is a Python Library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

3.7 Python UUIDs Module

This module provides immutable UUID objects and the functions `uuid1()`, `uuid3()`, `uuid4()` and `uuid5()` are used for generating version 1, 3, 4, and 5 UUID. If all one wishes for is a unique ID then either `uuid1()` or `uuid4()` are better options. But one thing to consider is that `uuid1()` may compromise privacy since it creates a UUID containing the computer's network address. `uuid4()` creates a random UUID.

UUID are used for uniquely identifying each object in the vision system. ROS provides a package called `Unique_id` which performs serialization and deserialization of UUID from hexadecimal to list ("uuid — UUID objects according to RFC 4122 — Python 3.10.4 documentation", n.d.).

3.8 OpenCV Library

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perceptions in the commercial products ("OpenCV Library", n.d.).

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

3.9 Machine Learning

Machine Learning is a branch of artificial intelligence (AI) that allows computers to learn and improve on their own without being explicitly programmed. In vision based systems machine learning is used in case scenarios where there is need for object detection, and later on classifying it on the basis of certain parameters (“Machine learning”, n.d.).

3.10 Kalman Filters

In certain cases it is very hard to directly calculate certain parameters of an AI based system. In such cases we use Kalman Filters. It provides us a probabilistic way to get predictions of the output that we were not able to infer directly. Kalman filters when used for tracking objects are known as kalman trackers and the algorithm that they use is known as Hungarian algorithm (“Kalman filter”, n.d.).

3.11 Edge Computing Device (SBC)

The NVIDIA Jetson lineup includes high-performance, power-efficient modules in compact form-factor for developing advanced robots and other autonomous machine products. Two of the most widely used computing boards are Jetson Nano and Jetson AGX series (“Embedded Systems Developer Kits & Modules from NVIDIA Jetson”, n.d.).

Benefit of using Jetson boards is that it comes with the NVIDIA JetPack SDK which is one of the most comprehensive solutions for building AI applications. One just needs to flash the board with the latest OS image, install developer tools for both the host computer and developer kit, and install the libraries and APIs, samples, and documentation to jumpstart the development environment.

Also, Jetson boards come with Tegra GPU from NVIDIA which are based on MAXWELL architecture. It supports usage of CUDA for optimizing vision systems on GPU. Also, with the NVIDIA GPU one gets the opportunity to use TensorRT which is an inference engine developed by NVIDIA to increase performance of AI models.

Owing to the above reasons, Jetson boards are the best choice for robotics application and I was able to use them extensively while working at Wastefull Insights. For small tests and development we used Jetson Nano and once the results were satisfactory we migrated that code block to Jetson NX.

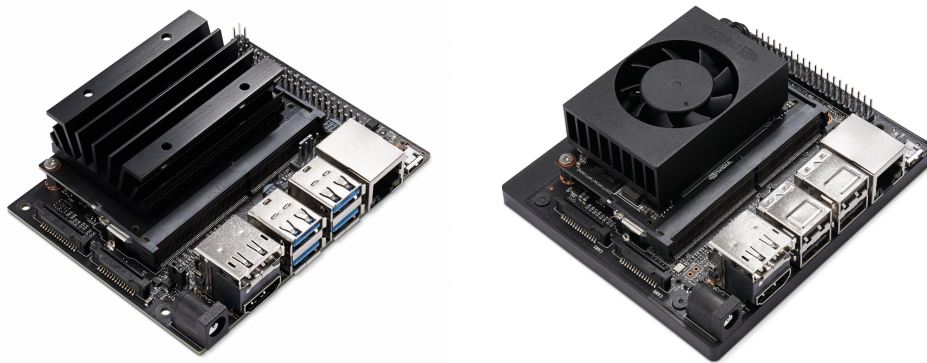


Fig 3.9 Jetson Nano and Jetson NX SBC

“Courtesy of (“NVIDIA Jetson Nano Developer Kit”, n.d.) and (“Jetson Xavier NX for Embedded & Edge Systems | NVIDIA”, n.d.)”

3.12 Embedded Board

Jetson Boards do provide GPIO pins for interfacing different peripherals but they have one limitation. They do not have internal ADC converters and also it is not ideal to put the entire system's load on a single device which is going to use an extensive GPU. Also, due to fluctuation in the current there are chances of the system clock being a bit inaccurate or having less precision. Considering this is essential to select an embedded board in our case it was TM4C1294XL or rather TivaC Launchpad board.

The TM4C1294XL Connected LaunchPad Evaluation Kit is a low cost development platform for ARM Cortex-M4F based microcontrollers. It is developed by Texas Instruments (“EK-TM4C1294XL Evaluation board | TI.com” 2014).

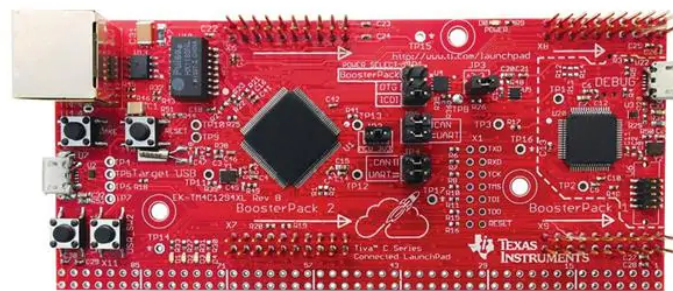


Fig 3.10 TM4C129 Microcontroller by TI

“Courtesy of (“EK-TM4C1294XL Texas Instruments”, n.d.)”

3.13 Energia IDE

Energia is an open-source electronics prototyping platform with the goal to bring the Wiring and Arduino framework to the Texas Instruments MSP430 based LaunchPad.

The Energia IDE is cross platform and supported on Mac OS, Windows, and Linux. Energia uses the mspgcc compiler by Peter Bigot. Together with Energia, LaunchPad can be used to develop interactive objects, taking inputs from a variety

of switches or sensors, and controlling a variety of lights, motors, and other physical outputs.

The framework is thoughtfully created with designers and artists in mind to encourage a community where both beginners and experts from around the world share ideas, knowledge and their collective experience. Professional engineers, entrepreneurs, makers, and students can all benefit from the ease of use Energia brings to the microcontroller (“Energia IDE”, n.d.).

In order to integrate TM4C1294 with ROS Serial there are two options, one of which is with Energia IDE. The first `rosserial_tivac` library generates ROS specific header files which are then imported in Energia IDE and that way publisher, subscriber or service can be implemented on microcontroller.



```
1 #define LED_RED_LED
2
3 // the setup routine runs once when you press reset:
4 void setup() {
5   // initialize the digital pin as an output.
6   pinMode(LED, OUTPUT);
7 }
8
9 // the loop routine runs over and over again forever:
10 void loop() {
11   digitalWrite(LED, HIGH); // turn the LED on (HIGH is the voltage level)
12   delay(1000); // wait for a second
13   digitalWrite(LED, LOW); // turn the LED off by making the voltage LOW
14   delay(1000); // wait for a second
15 }
```

Done Saving.

15 RED LaunchPad w/ msp432 EMT (48MHz) on /dev/cu.usbmodemM4321001

Fig 3.11 Energia IDE Interface
“Courtesy of (“Energia IDE”, n.d.)”

3.14 Cross Compiler

In order to program a TM4C1294 microcontroller from an x86 architecture system, a cross-compiler is essential. A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running (“Cross compiler”, n.d.).

A cross compiler is necessary to compile code for multiple platforms from one development host. Direct compilation on the target platform might be infeasible. GNU Toolchain provides arm-none-eabi cross-compiler which will be used to integrate TM4C1294 with ROS.

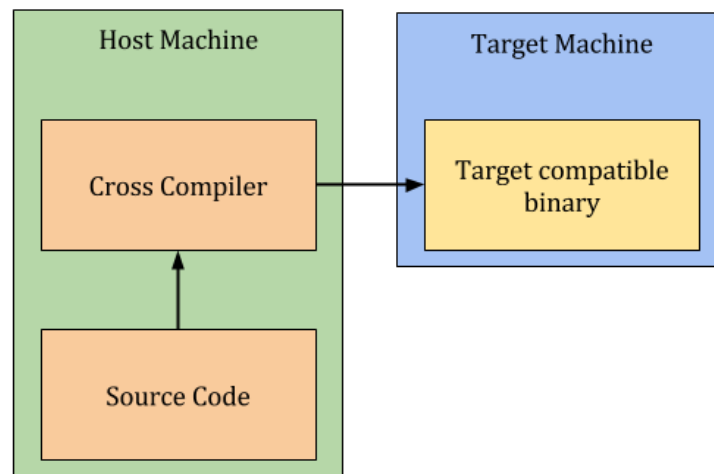


Fig 3.12 Cross Compiler

“Courtesy of (“Cross Compilation Toolchain for ARM - Example with Raspberry Pi”, n.d.)”

3.15 ArUco Marker

ArUco marker is a synthetic square marker composed by a wide black border and inner binary matrix which determines its identifiers. The black border facilitates its fast detection in the image and binary codification allows its identification. There are other markers available like April Tags but ArUco has one of the fastest detection (“OpenCV: Detection of ArUco Markers”, n.d.).

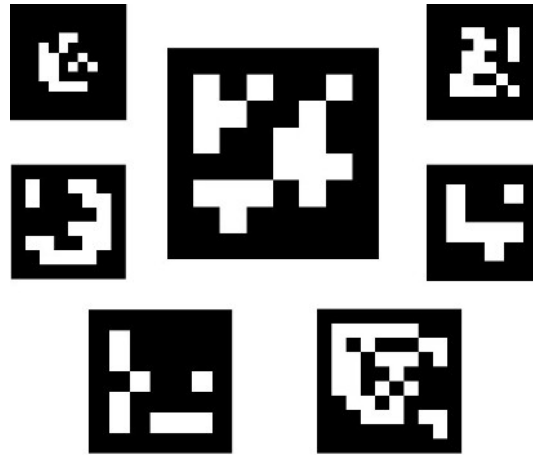


Fig 3.13 ArUco Marker Of Different Dictionary Type
“Courtesy of (“OpenCV: Detection of ArUco Markers”, n.d.)”

It is possible to fetch the location of the marker but in order to do that first camera needs to be calibrated and from its distortion matrix we will be able to locate it.

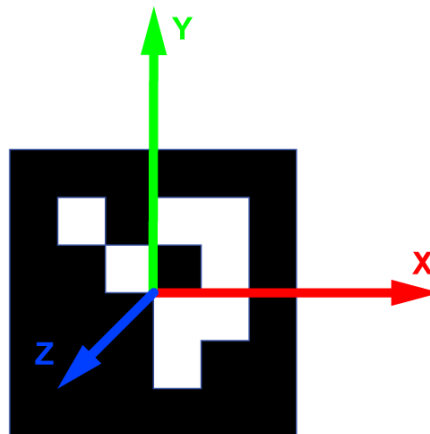


Fig 3.14 Pose Estimation With ArUco Marker
“Courtesy of (“OpenCV: Detection of ArUco Markers”, n.d.)”

Chapter 4

System Analysis

4.1 Study of Current System

In the first month of the internship my major task was to understand the company's existing code base. Entire code base was in a single Git repository and there were several packages ranging from feed capturing packages, ML implementation packages, Waste pickup selection packages, and Embedded system packages.

We were having a conveyor based system on which plastic waste will go from camera's ROI to robot's ROI and on the basis of software architecture and implemented algorithm it will pick up waste and put it in respective bins. Instead of going with delta bots which have low coverage area and complete joint calculations, the system developed by Wastefull Insights was gantry based which provided fast and efficient pick drop of the waste.

4.2 Problem and Weaknesses of Current System

Current system was developed as Proof of Concept in mind and hence there were many scope for improvements and optimization for the system. One major issue was that the robot was using a greedy approach to pick up the waste which might lead to issue of local maxima in the long run and thus will result in sub-optimal performance.

In the waste segregation system one of the important parameters is how fast the robot can pick objects and how many it can pick in a certain duration. While considering in the long run optimal performance will result in less number of robots deployed at specific plants and this will reduce client's cost of installation and maintenance.

Flexibility and Modularity go hand in hand, it is crucial that the software system running the robot can be modified with any given parameters or considerations given either by developer or client. For this reason the software architecture should be accommodative in nature which was not the case at the movement.

Most of the logic and algorithms on which the entire system was running was in 2-3 ROS packages with one node in each. Due to this there were 8-9 levels of abstraction in the code which were completely dependent on each other which made the entire system rigid and prone to errors even if slight modification was made in one node.

Also, ROS provides plenty of amazing features which will reduce system's overhead and there are multiple ways to perform a single task. Hence, it is essential to use that one approach which provides best results with modularity and robustness. I was able to see that many of those functionalities were not included and that was emulating hard coded behavior for the software architecture.

4.3 Requirements of New System

For the new system to be more efficient, flexible and robots following modifications are essential:

- First and foremost is the structure of ROS packages with singular package structure. All the python executables should be placed in the scripts folder of respective packages. Also, Setup.py should be configured with include directory as python package initializer. All the 3rd party or user files, libraries and dependencies should be included in src directory with package name abstraction layer. Configuration files should be included in the config directory of respective folders.
- ROS system software should be modified and include services and actions wherever necessary instead of just relying on publisher subscriber

functionality. This is make a significant different to message transmission overhead of the entire system

- Currently every package builds their own respective ROS messages. This is beneficial to customize a specific set of messages but it will be redundant if two nodes of packages are building similar message types. Also, if one needs to modify one specific message then he/she needs to find all the copies and also all of those packages need to declare message building dependencies. The best approach is to make a separate ROS message package from which all the nodes will include message definitions.
- In vision based system cameras are of prime importance and with that comes frame rate. When using ROS based image processing packages there might occur some processing delay. Not only in image processing, but in all message transmission if the message size is larger then it is likely that there will be some transmission delay. So, the new system would have to take that into consideration because if not then the robot will not be able to pick drop waste.
- To optimize the system it is important to revamp the software architecture in ROS to provide sufficient flexibility with which optimization algorithms can be integrated and tested. Also, for the end user there needs to be an operator's UI interface with which the robot and vision system can be controlled.
- Another import parameter is the security of the system. As ROS uses TCP/UDP protocol based on IP it is very important that none of the ports are left open and the UI created should be internally routed so that no one even in the same network can access it other than the device on which it is running.

- Unit testing framework integrated with ROS and development of structured unit tests to identify system fault in real-time on validating passed and failed tests.

4.4 Features of New System

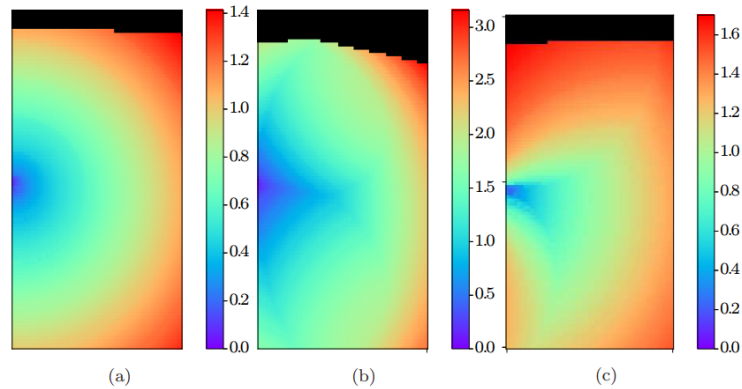
After implementation of the above suggested requirements following will be the features of the new system:

- Modular in nature
- Non-redundant builds
- Robust
- Structured
- Secured
- Fault-Tolerant
- Efficient
- Maintainable
- Portable

4.5 Literature Review of Optimization Algorithms

One of the important research papers that helped in designing optimization algorithms was “Toward Fast and Optimal Robotic Pick-and-Place on a Moving Conveyor” by Shuai D. Han, Si Wei Feng and Jingjin Yu. According to this paper when a greedy approach is considered like SPT or FIFO, it will work with a very short horizon and will generally end up in sub-optimality (Han, Feng, and Yu 2019).

A significant challenge in the design and implementation of object picking sequence selection algorithm is how to deal with the geometry and dynamics of the robots that are involved. This was not that much of a concern to use due to the gantry based system. When visualizing typical PnP Time Profiles, following results were achieved by the researchers.



(a) Simplified telescoping robot. (b) Delta robot (c) SCARA robot

Fig 4.1 PnP Time Profile

“Courtesy of (Han, Feng, and Yu 2019)”

After these results it is essential to implement optimization algorithms for pick and place robots. There are basically two options available, one is Exact and another one is Approximate algorithm for selecting the best picking sequence.

4.5.1 Exhaustive Search Methods

With the GetPnPTime routine, a baseline exhaustive search routine is straightforward to obtain. We call such a routine OPTSEQ, which computes the optimal object picking sequence for a given horizon. After this dynamic programming is applied to speed up OPTSEQ, yielding the routine OPTSEQ DP, which is significantly faster yet without any loss of optimality (Han, Feng, and Yu 2019).

Algorithm 1: OPTSEQ

Input: objects' initial location $(x_1, y_1), \dots, (x_n, y_n)$

Output: S^* : a time-optimal PNP sequence

```

1  $t^* \leftarrow \infty, S^* \leftarrow \text{none}$ 
2 for  $P \in \text{ALLPERMUTATIONS}(\{1, \dots, n\})$  do
3    $t \leftarrow 0$ 
4   for  $i \in P$  do  $t \leftarrow t + \text{GETPNPTIME}(x_i - v_b t, y_i)$ 
5   if  $t < t^*$  then  $t^* \leftarrow t, S^* \leftarrow P$ 
6 return  $S^*$ 

```

Fig 4.2 OPTSEQ Algorithm

OPTSEQ Algorithm prioritizes the best sequence from all that are available. The issue with this algorithm is that it will perform redundant calculation as it is not storing details about previous sequence performance.

Algorithm 2: OPTSEQDP

Input: objects' initial location $(x_1, y_1), \dots, (x_n, y_n)$
Output: a time-optimal PNP sequence

```

1  $T = \{\emptyset : 0\}, S = \{\emptyset : ()\}$ 
2 for  $1 \leq k \leq n$  do
3   for  $U \leftarrow \text{ALLCOMBINATIONS}(\{1, \dots, n\}, k)$  do
4     UPDATE( $T, S, U$ )
5 return  $S[\{1, \dots, n\}]$ 

```

Fig 4.3 OPTSEQDP Algorithm

OPTSEQDP Algorithm uses OPTSEQ with dynamic programming. Hence, it will store detail about previous sequences and that way be non-redundant. Also, it is dynamic in nature and hence will not search entire problem space and this saves time and computation.

4.5.2 Local Augmentation Method

In consideration with OPTSEQDP, many different heuristics were used to further boost its efficiency. But there is another method which achieves optimality close to OPTSEQDP but scales are much better. This method is called the local augmentation-based method SUBOPTDP, which uses OPTSEQDP as a subroutine.

One consideration was that the conveyor will run without stopping for extended periods of time, for the continuous setting, OPTSEQDP or SUBOPTDP are invoked repeatedly with real-time locations of all the pick-able objects in the workspace.

Algorithm 3: SUBOPTDP

Input: objects' initial location $(x_1, y_1), \dots, (x_n, y_n)$
Output: a near-optimal PNP sequence

```

1  $S \leftarrow \text{GETINITIALPICKINGSEQUENCE}()$ 
2 for  $m_1$  times do
3    $t \leftarrow 0$ 
4   for  $1 \leq k \leq n - m_2$  do
5      $O \leftarrow \emptyset$ 
6     for  $i \in S[k : k + m_2]$  do
7        $O \leftarrow O \cup \{(x_i - v_b t, y_i)\}$ 
8      $S[k : k + m_2] \leftarrow \text{OPTSEQDP}(O)$ 
9      $t \leftarrow t + \text{GETPNPTIME}(x_{S[k]} - v_b t, y_{S[k]})$ 
10 return  $S$ 

```

Fig 4.4 SUBOPTDP Algorithm

This is one of the most optimized algorithms. It uses OPTDP algorithm but instead of applying it on sequence, it will select a sub-sequence. This way the solution state space will reduce drastically and achieve optimum solution in the fastest possible manner.

4.5.3 Experimental Results by Researcher

After evaluating all the three algorithms, the observation made by the researchers was that the active workspace in a conveyor PnP system contains a few to low tens of objects. From the following figure, one can observe that both OPTSEQDP and SUBOPTDP can complete a single sequence computation for ten objects within 10-4 seconds and fifteen objects with 10-2 seconds. Because, Delta and SCARA-based PnP systems generally do not pick more than a single digit number of objects per second, OPTSEQDP and SUBOPTDP impose negligible time overhead. As such, they are sufficiently fast for industrial applications.

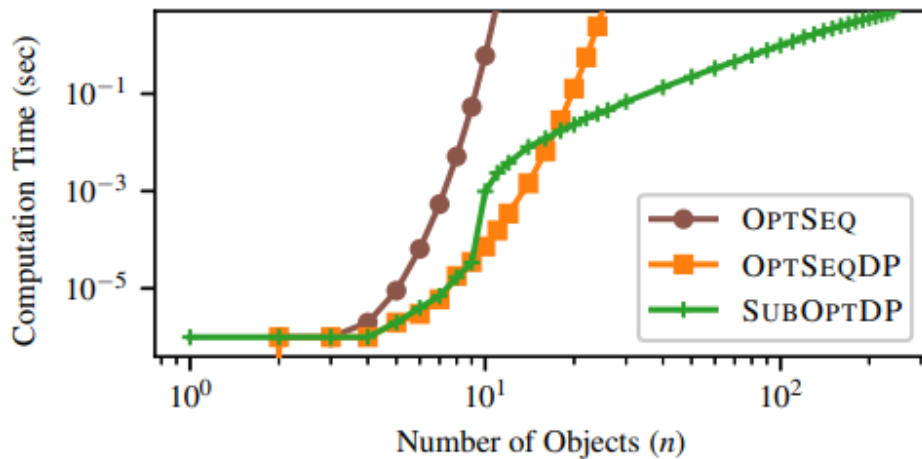


Fig 4.5 Comparison Of Different PnP Algorithms

“Courtesy of (Han, Feng, and Yu 2019)”

4.6 Selection of Optimization Approaches and Justification

After having studied the above tested algorithms by the researchers, they seemed viable options which could be tested on the Wastefull Insights’ robot but, there was one major difference. All the above three stated algorithms were tested and experimented on robotics arms and not gantry systems. So, considering this execution plan included re-structuring ROS software architecture in such a manner that all three algorithms could be tested and from them the best suited one with optimal performance was finalized.

Starting with Optimum sequence generation and then using dynamic programming with it should increase the efficiency of the robot. One more approach would be to use concepts of scheduler in conjunction with these algorithms and observer performance differences.

	Length Scaling on x-axis / Performance				
Algorithms	6000	8000	10000	12000	14000
SUBOPTDP	<60%	<80%	<90%	<95%	~100%
SPT	~60%	<80%	<85%	<90%	<95%
EUCLIDEAN	<65%	<75%	<85%	<90%	<95%
FIFO	<40%	<60%	<80%	<90%	<100%

Table 4.1 Comparison Between Different Pick And Place Algorithm
“Referenced from (Han, Feng, and Yu 2019)”

Chapter 5

System Design

5.1 System Design and Methodology

System Design according to current architecture is as follow:

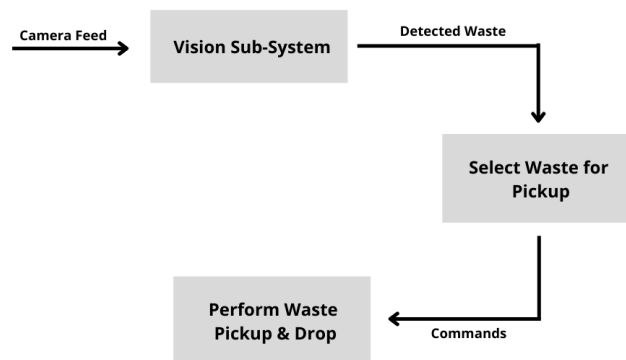


Fig 5.1 Initial System Design

Major problem with this is modularity and maintainability. Also, whenever someone who has not written the system tries to understand it, it might take him/her a while to understand the flow of functions and level of abstractions.

Restructured System Design with the new architecture will be as follow:

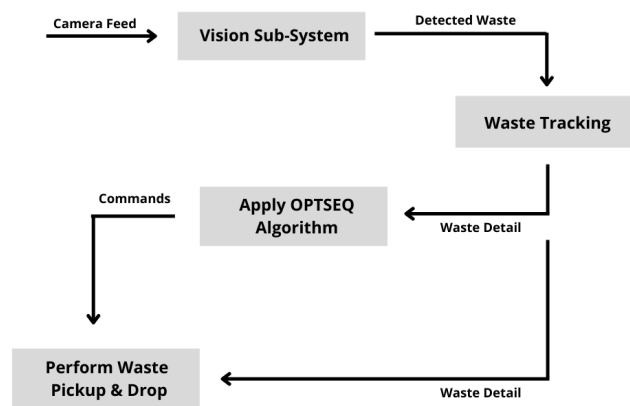


Fig 5.2 Restructured System Design

This gives more flexibility for modifications that can take place later on depending on the client's need or upgradation of the optimization algorithm.

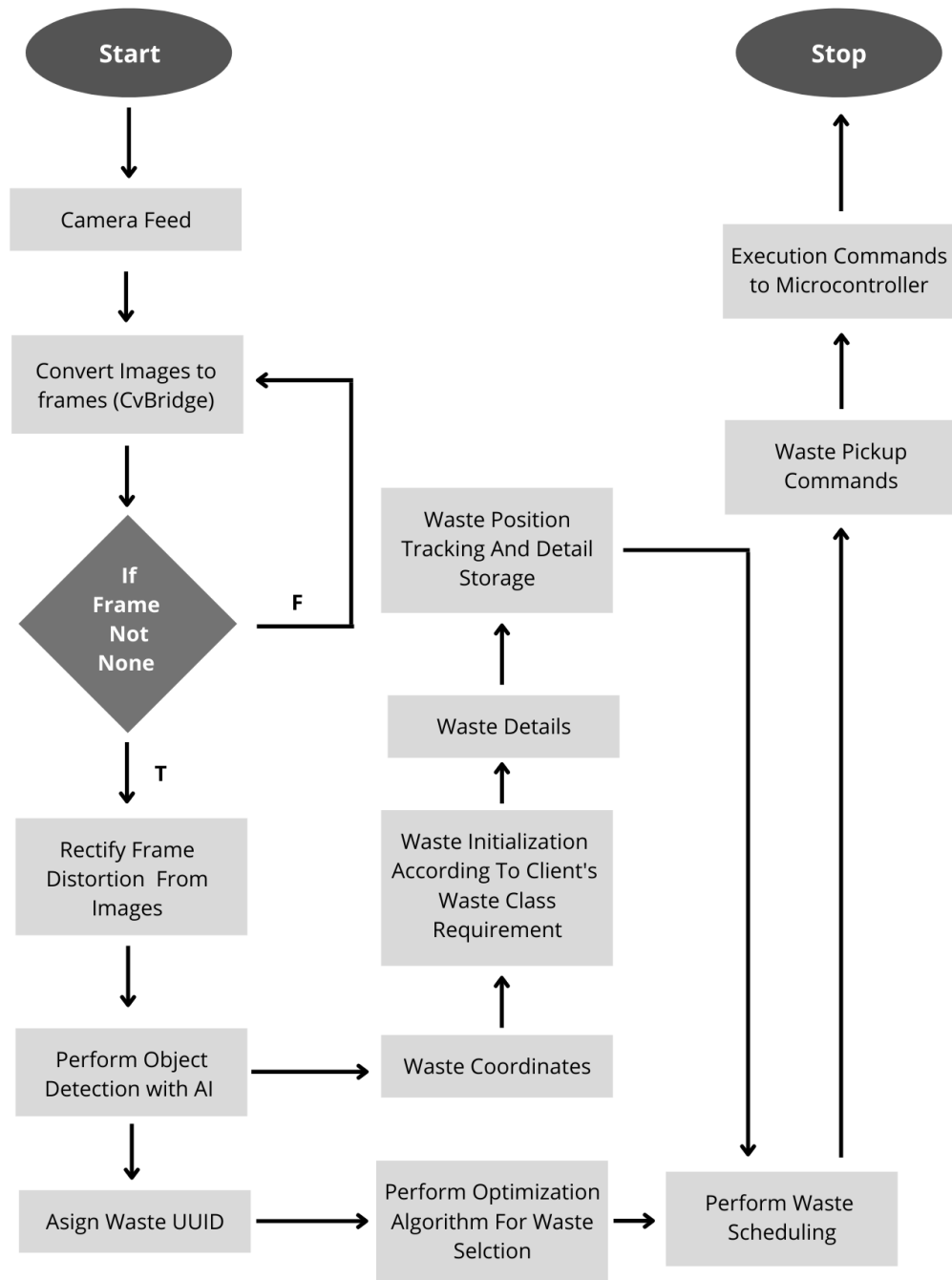


Fig 5.3 System Design Flowchart

Images from the camera feed are taken as input and converted to ROS Image message types from CV Images using CvBridge. If the image frame captured is empty then it is skipped. After this using image processing, frames are rectified from any distortions. From there images are sent to an object detection algorithm which will classify waste.

From here detected waste will go on two different paths. First it will go to the initialization node where waste is categorized according to their class and bin locations are fed into the system. After this waste details will be stored in the waste tracker node and its coordinate on the conveyor will be continuously updated.

And the second path is where waste UUIDs are transferred to the optimization algorithm node where the best sequence of execution is selected and fed to the scheduler. Scheduler will generate necessary delay between pick and place of two consecutive wastes and send final commands to the microcontroller node which will then control the robot.

This entire process will continuously run in a loop until the system is either stopped, or it enters into any error state. One of the major advantages of this architecture is that it is modular in nature and one part of the system is not dependent or has adverse effects on other part of the system. It is kind of a black box system where only input and output parameters are taken into consideration.

Still even after this there will be some limitations but they will be because ROS1 has a centralized approach wherein ROS Master is taking care of everything. In the later iteration of the system when Nvidia launches support for ROS2 we would be able to migrate this into complete decentralized architecture.

5.2 Input / Output and Interface Design

5.2.1 State Transition Diagram

Following is the state transition diagram of the system before optimization and in this design everything is taking place in linear fashion due to which there are many possible loopholes in which system might get stuck with the problem of local maxima.:

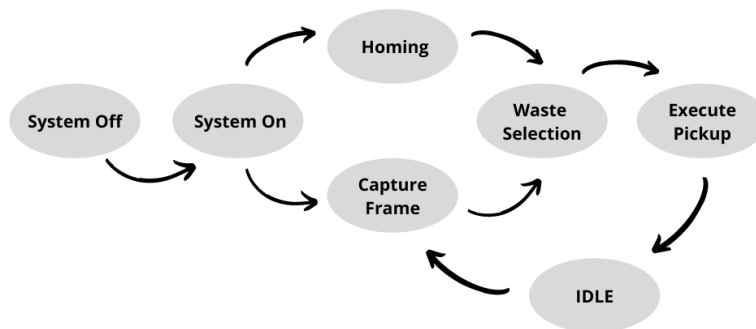


Fig 5.4 Initial State Transition Diagram

Following is the state transition diagram of the system after optimization:

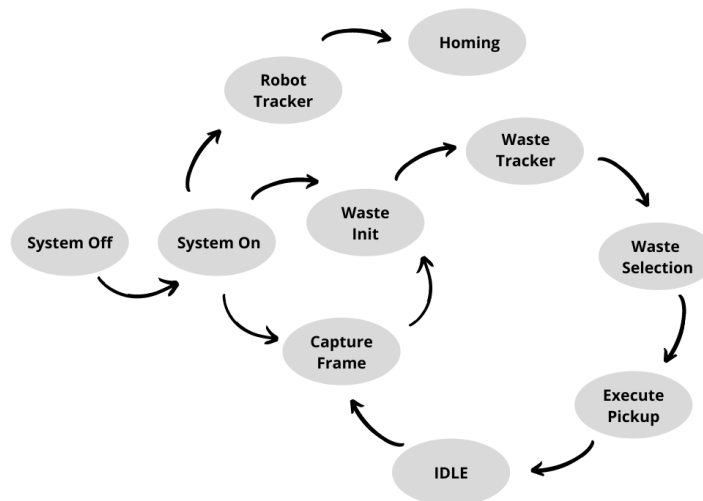


Fig 5.5 Optimized State Transition Diagram

5.2.2 Operator's UI Interface Design

Following is the rough format of the frontend interface which will be used by the operator to control and monitor the robot.

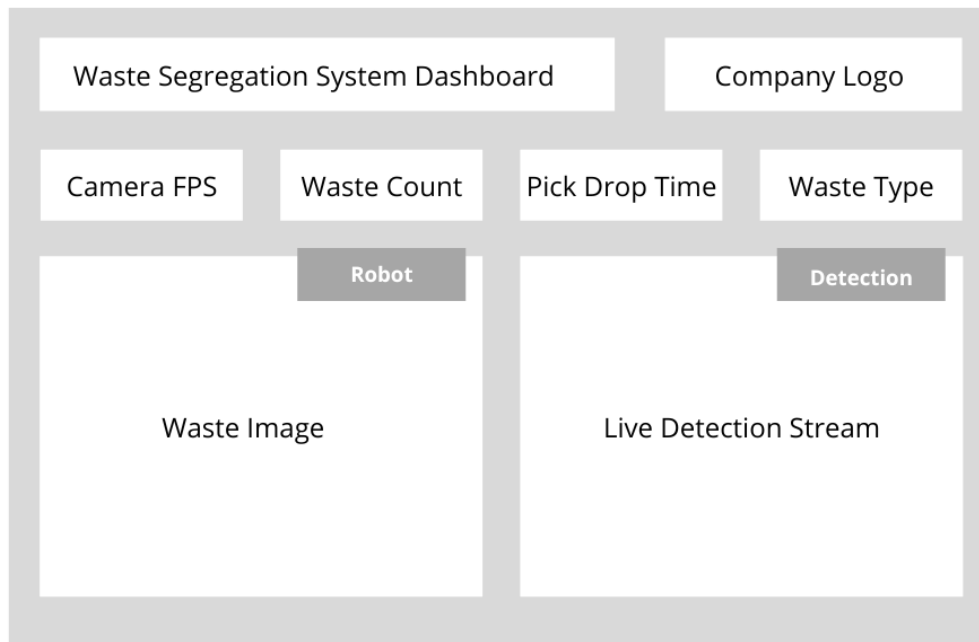


Fig 5.6 Operator's UI Design

Chapter 6

Implementation

6.1 Implementation Platform and Environment Setup

6.1.1 Jetson Nano Setup

One of the major considerations while setting up Jetson Nano is the version of the JetPack SDK. Each JetPack SDK comes with a specific set of dependencies. Consider JetPack 4.6.1, It comes with following:

- OS - Ubuntu 18.04
- TensorRT - Version 8.2.1
- cuDNN - Version 8.2.1
- CUDA - Version 10.2

Hence, it becomes very important to consider the version of all the included dependencies because version compatibility is very important. The JetPack SDK version that I am targeting is 4.5.1.

Another important consideration is the OpenCV version. With the JetPack SDK 4.5.1 we receive OpenCV 4.1.1 but Wastefull Insights is using OpenCV 3.4.6 which runs efficiently on Jetson Nano and NX.

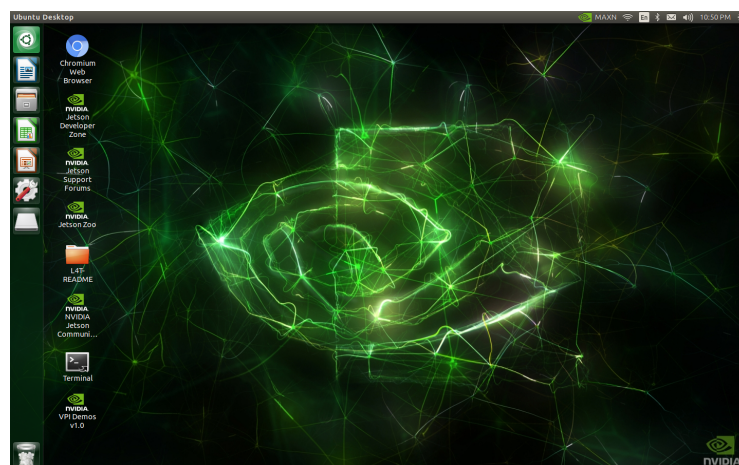


Fig 6.1 Jetson Nano Home Interface After Boot

6.1.2 OpenCV 3 Installation

Installing OpenCV 3 or 4 on an x86 architecture computer is pretty straightforward but that is not the case with Jetson compute boards. Jetson boards are edge computing devices because of which they are resource constrained and when installing OpenCV on Jetson Nano with 4GB of RAM one needs to increase swap memory. Even after this entire installation process will take nearly 2-3 hours considering only core packages are being installed.

OpenCV comes with contrib modules which are developed and maintained by the community. But one needs to add them separately. ArUCo is one such contrib module and hence I had to include contrib modules in installation.

Directly prompting make and cmake can be tedious and prone to errors. To avoid that there is a GUI based utility called cmake-gui which provides terminal as follow:

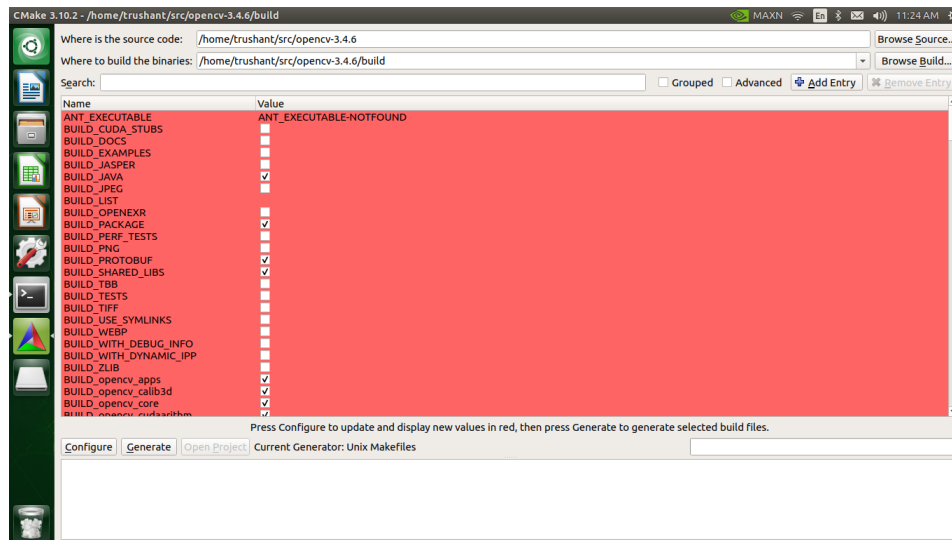


Fig 6.2 CMAKE GUI Console

6.1.3 ROS Melodic Installation

There are various versions of ROS and it depends on the underlying operating system which one to install. As Jetson Nano supports only Ubuntu 18.04, with it ROS Melodic can be installed.



Fig 6.3 ROS Melodic Logo
“Courtesy of (Foote 2018)”

There are two options available for ROS installation, one is building from source and another one is installing pre-build binaries. The issue is that ROS melodic when built from binaries comes with python 2.7 support which reached its EOL. Hence, another consideration while installing ROS Melodic is to build it manually from source by replacing python2.7 headers with that of python3.

Another important note is that though ROS Melodic will be built with python3 support explicitly, one needs to build cv_bridge and image_proc packages explicitly in the workspace from which it will be used.

6.1.4 Pylon SDK Installation and Setup

In order to reduce transmission delay and achieve more customizability, instead of using a USB camera, Wastefull Insights is using a GiGe camera. This camera is from Basler and it comes with its own SDK called Pylon. Basler people have provided a ros_pylon package to integrate Pylon camera with ROS system but it will require Pylon SDK.

ROS package for pylon camera provides all the features with camera_info_manager which tracks camera_calibration parameters and configurations.

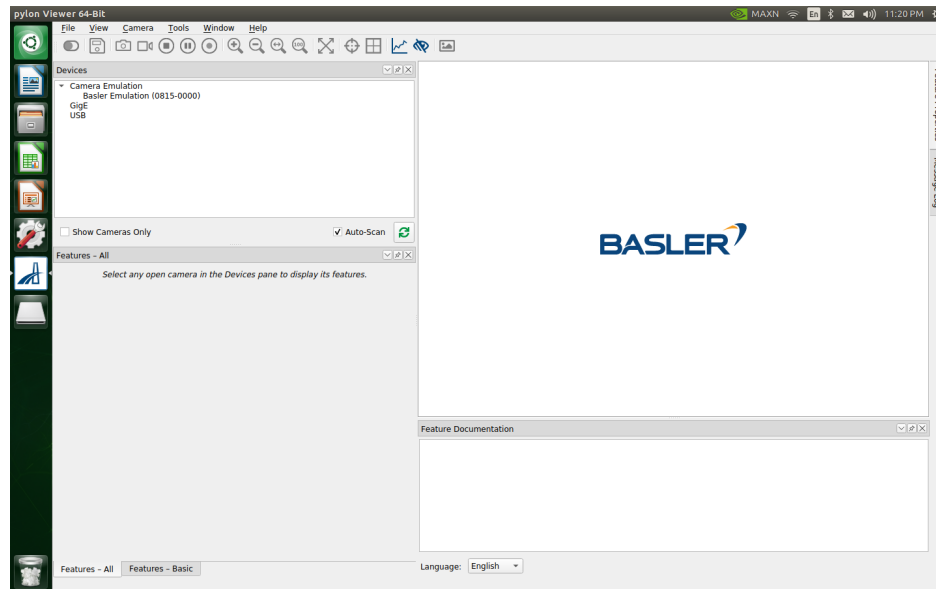


Fig 6.4 Pylon SDK Console

GiGe Camera works on IP so they require IP configuration setup and for that Pylon SDK provides pylon IP configuration tool.

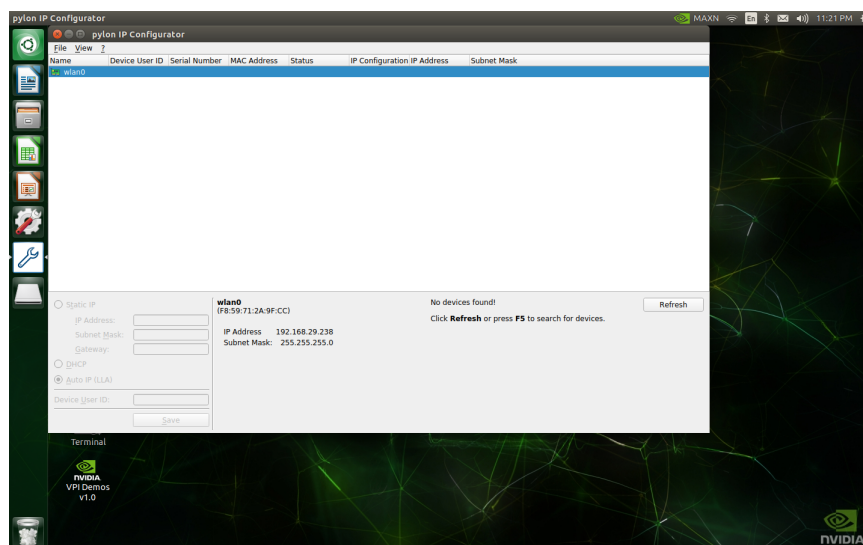
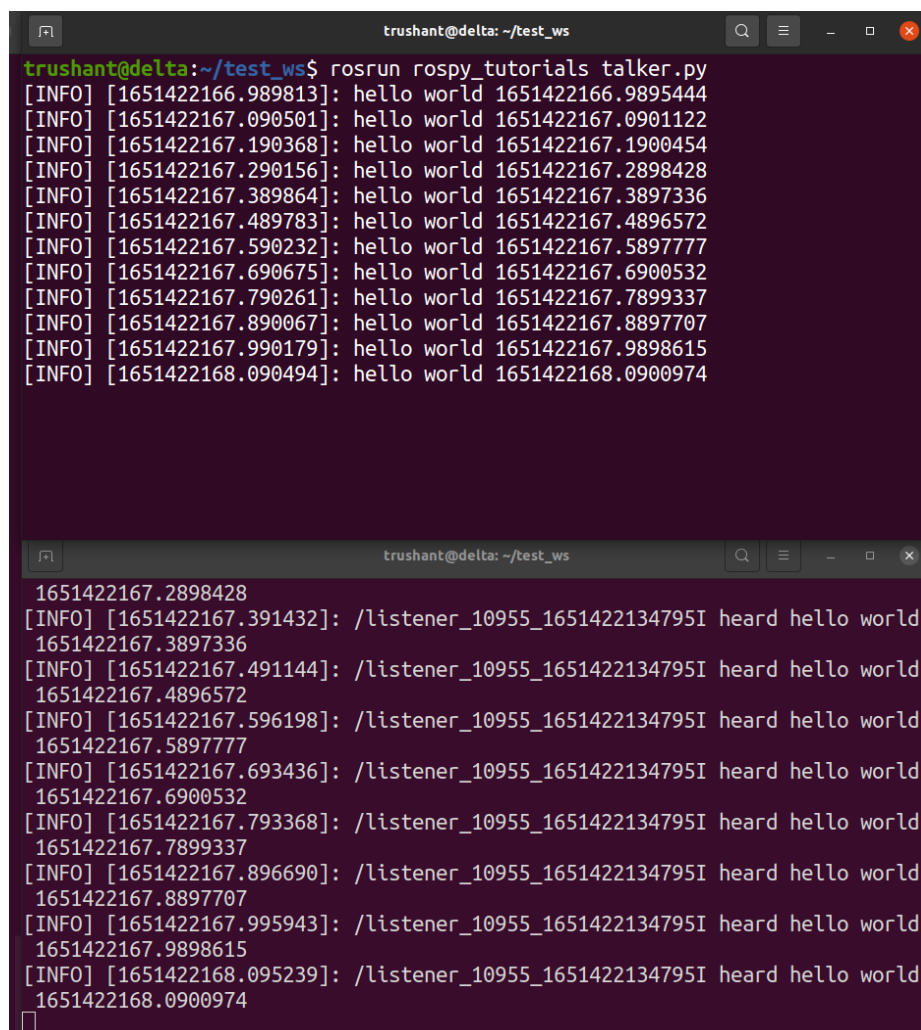


Fig 6.5 Pylon IP Configuration Tool

6.2 ROS Basics Examples

6.2.1 ROS Publisher Subscriber Implementation

For Publisher Subscriber, roscore is running in the background. In the figure below, the top terminal is a Talker Node which is Publisher “hello world” message with ROS clock on a /chatter topic. And in the bottom terminal a Listener Node which is a subscriber subscribing to /chatter topic and displaying ROS messages sent by Publisher Node.



```
trushant@delta: ~/test_ws
trushant@delta:~/test_ws$ rosrunc rospy_tutorials talker.py
[INFO] [1651422166.989813]: hello world 1651422166.9895444
[INFO] [1651422167.090501]: hello world 1651422167.0901122
[INFO] [1651422167.190368]: hello world 1651422167.1900454
[INFO] [1651422167.290156]: hello world 1651422167.2898428
[INFO] [1651422167.389864]: hello world 1651422167.3897336
[INFO] [1651422167.489783]: hello world 1651422167.4896572
[INFO] [1651422167.590232]: hello world 1651422167.5897777
[INFO] [1651422167.690675]: hello world 1651422167.6900532
[INFO] [1651422167.790261]: hello world 1651422167.7899337
[INFO] [1651422167.890067]: hello world 1651422167.8897707
[INFO] [1651422167.990179]: hello world 1651422167.9898615
[INFO] [1651422168.090494]: hello world 1651422168.0900974

1651422167.2898428
[INFO] [1651422167.391432]: /listener_10955_1651422134795I heard hello world
1651422167.3897336
[INFO] [1651422167.491144]: /listener_10955_1651422134795I heard hello world
1651422167.4896572
[INFO] [1651422167.596198]: /listener_10955_1651422134795I heard hello world
1651422167.5897777
[INFO] [1651422167.693436]: /listener_10955_1651422134795I heard hello world
1651422167.6900532
[INFO] [1651422167.793368]: /listener_10955_1651422134795I heard hello world
1651422167.7899337
[INFO] [1651422167.896690]: /listener_10955_1651422134795I heard hello world
1651422167.8897707
[INFO] [1651422167.995943]: /listener_10955_1651422134795I heard hello world
1651422167.9898615
[INFO] [1651422168.095239]: /listener_10955_1651422134795I heard hello world
1651422168.0900974
```

Fig 6.6 ROS Talker and Listener Node

Following output is from rqt_graph which is a GUI tool in ROS used to visualize running nodes and topics. According to this there are two nodes

called talker and listener who are using /chatter topic for communicating ROS messages. Rqt_graph is an essential tool for debugging ROS software systems.

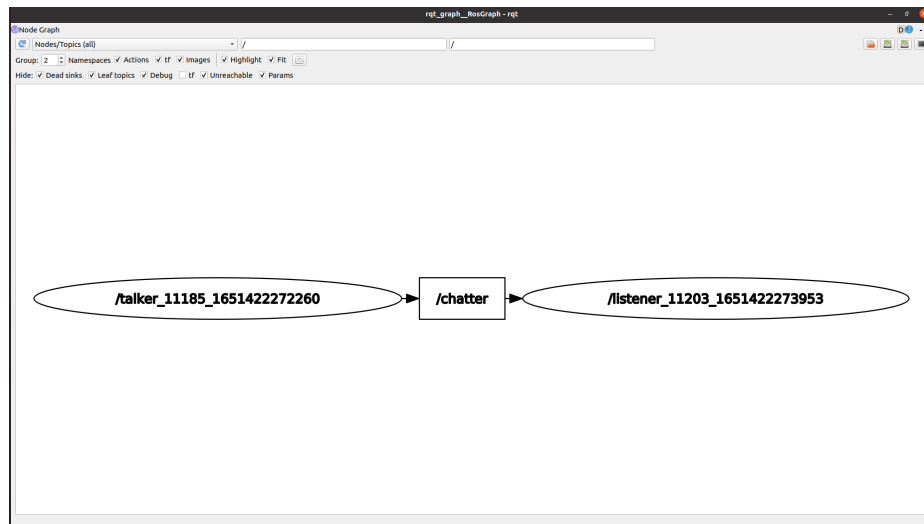


Fig 6.7 Pub Sub Computational Graph in rqt_graph

6.2.2 ROS Service Implementation

In the following snapshot a service server on the top terminal is implemented which takes 2 integers as input and returns their sum. In the bottom terminal a service client is executed which sends those 2 integers. Also, roscore is running in the background.

ROS services are following client server architecture and due to that reason while the service is being served by the server, the client will be paused. This kind of structure is very useful in ROS to start and stop certain parameters of a robot.

One down side to service is that they are synchronous in nature and you won't be able to debug it with rqt_graph.

```

trushant@delta: ~/ros_ws
trushant@delta:~/ros_ws$ roslaunch rospy_tutorials
add_two_ints_client          server_connection_header.py
add_two_ints_server         talker
advanced_publish.py         talker_connection_header.py
client_connection_header.py talker_header.py
listener                     talker_listener_test.py
listener_connection_header.py talker.py
listener_header.py          talker_timer.py
listener.py                  test_add_two_ints.py
listener_subscribe_notify.py test_client_connection_header.py
listener_with_user_data.py  test_listener_connection_header.py
param_talker.py             test_on_shutdown.py
publish_on_shutdown.py      test_peer_subscribe_notify.py
publish_on_shutdown_test_node.py test_server_connection_header.py
publish_pointcloud2.py

trushant@delta:~/ros_ws$ roslaunch rospy_tutorials add_two_ints_client
[roslaunch] You have chosen a non-unique executable, please pick one of the following:
1) /home/trushant/ros_ws/devel/share/rospy_tutorials/005_add_two_ints/add_two_ints_client
2) /home/trushant/ros_ws/src/ros_tutorials/rospy_tutorials/005_add_two_ints/add_two_ints_client
#? 2
Requesting 48431+37837
48431 + 37837 = 86268
trushant@delta:~/ros_ws$

trushant@delta:~/ros_ws$ roslaunch rospy_tutorials add_two_ints_server
[roslaunch] You have chosen a non-unique executable, please pick one of the following:
1) /home/trushant/ros_ws/devel/share/rospy_tutorials/005_add_two_ints/add_two_ints_server
2) /home/trushant/ros_ws/src/ros_tutorials/rospy_tutorials/005_add_two_ints/add_two_ints_server
#? 2
Returning [48431 + 37837 = 86268]
Returning [48431 + 37837 = 86268]

```

Fig 6.8 ROS Service For Adding Two Integers

6.2.3 ROS Action Implementation

Starting roscore so that all the nodes can find each other as follow:

```

trushant@delta:~/test_ws$ roscore
... logging to /home/trushant/.ros/log/83a50c82-c96d-11ec-aa16-e3e81c414c9b/roslaunch-delta-11480.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://delta:34537/
ros_comm version 1.15.14

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [11488]
ROS_MASTER_URI=http://delta:11311/

setting /run_id to 83a50c82-c96d-11ec-aa16-e3e81c414c9b
process[rosout-1]: started with pid [11498]
started core service [/rosout]

```

Fig 6.9 ROS Core Execution

After the roscore is up and running, the action server will be executed. Here there is a simple action server which takes 20 as an input argument and generates fibonacci sequence till the first 20 digits.

```
trushant@delta:~/test_ws$ rosrn actionlib_tutorials fibonacci_server.py
[INFO] [1651423672.865679]: /fibonacci: Executing, creating fibonacci sequence of order 20 with speed 0, 1
[INFO] [1651423691.867252]: /fibonacci: Succeeded
█
```

Fig 6.10 ROS Fibonacci Action Server

In order to send this number 20 we will run an action client as follow:

```
trushant@delta:~/test_ws$ rosrn actionlib_tutorials fibonacci_client.py
Result: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765
trushant@delta:~/test_ws$ █
```

Fig 6.11 ROS Fibonacci Action Client

ROS actions provide feedback through which we can monitor progress of our actions. For that we will echo that particular action server's feedback topic.

```
feedback:
  sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584]
---
header:
  seq: 18
  stamp:
    secs: 1651423689
    nsecs: 867263793
  frame_id: ''
status:
  goal_id:
    stamp:
      secs: 1651423672
      nsecs: 863937616
    id: "/fibonacci_client_py-1-1651423672.864"
  status: 1
  text: "This goal has been accepted by the simple action server"
feedback:
  sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
---
header:
  seq: 19
  stamp:
    secs: 1651423690
    nsecs: 867191553
  frame_id: ''
status:
  goal_id:
    stamp:
      secs: 1651423672
      nsecs: 863937616
    id: "/fibonacci_client_py-1-1651423672.864"
  status: 1
  text: "This goal has been accepted by the simple action server"
feedback:
  sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765]
---
█
```

Fig 6.12 ROS Fibonacci Action Server Feedback

6.2.4 ROS TurtleSim Implementation

In order to learn ROS there is a tool called TurtleSim which is basically a 2D simulator with which we can control a turtle in 2D plane. Following is its execution of roscore in TurtleSim Window.

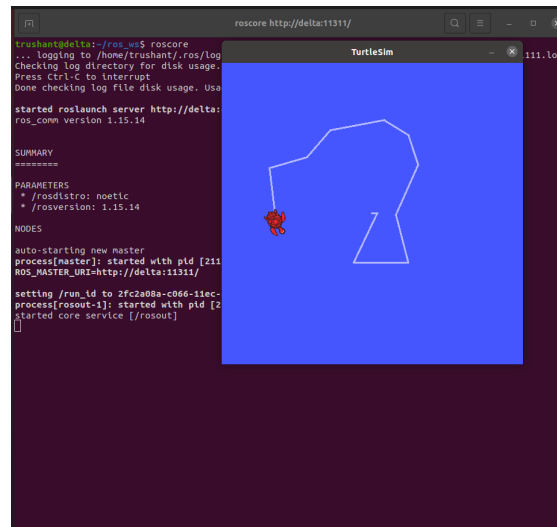


Fig 6.13 ROS TurtleSim And ROS Core

After this we can launch a teleop node as shown in the bottom terminal of the following figure. With the help of a teleop node one can use a keyboard to move turtles in the 2D plane.

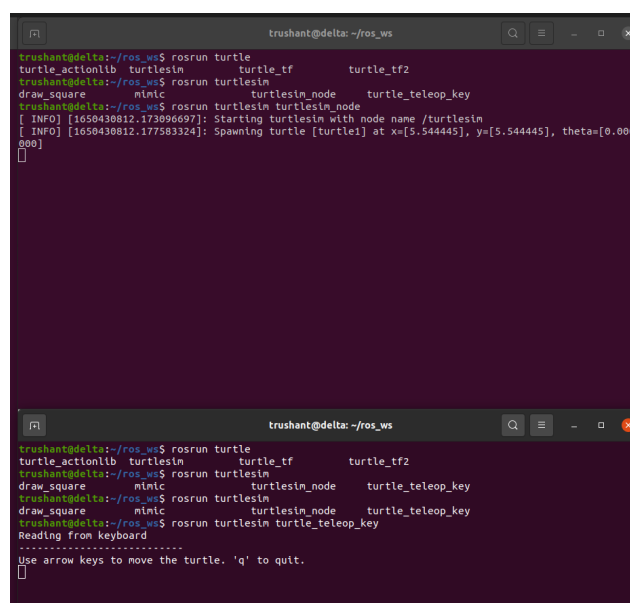


Fig 6.14 ROS Teleop

6.2.5 ROS RVIZ AMR Implementation

In the initial days of the internship I would give 5-10min a day to learn something new and different in ROS. In that process I designed a 2 wheel differential drive robot in Fusion360 and converted it into URDF. This built a package called `micro_bot_description` using which I was able to visualize robot models in a ROS tool called RVIZ. RVIZ is used to visualize sensor data of a robot.

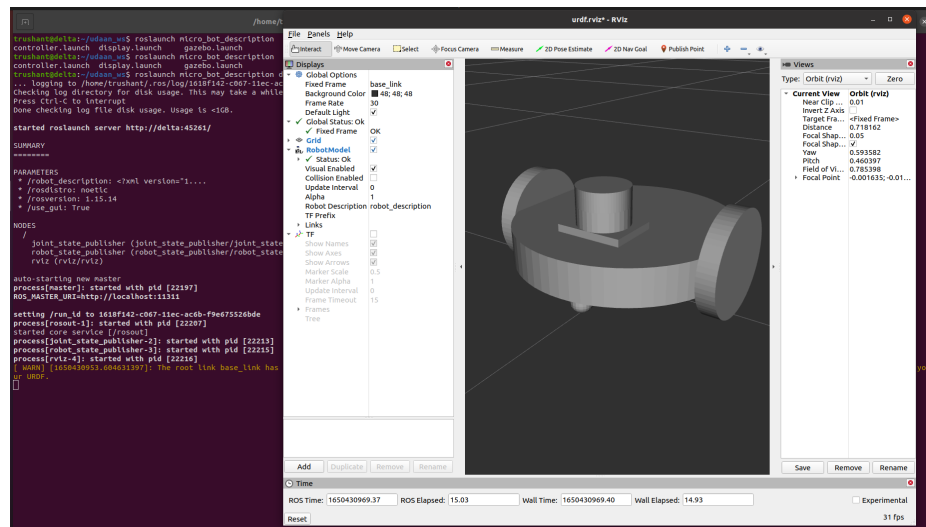


Fig 6.14 Differential Drive Robot (RVIZ)

6.2.6 ROS Gazebo AMR Implementation

Gazebo is a 3D simulator which can be used with ROS. After visualizing a robot model in RVIZ I thought of importing that model in Gazebo, attaching a LiDAR plugin with it and running the robot in the Gazebo world with a keyboard. Following is the implementation of that.

All of the above illustrated tasks were performed during the internship to brush up ROS skills and evaluate pros and cons of different features that ROS provides.

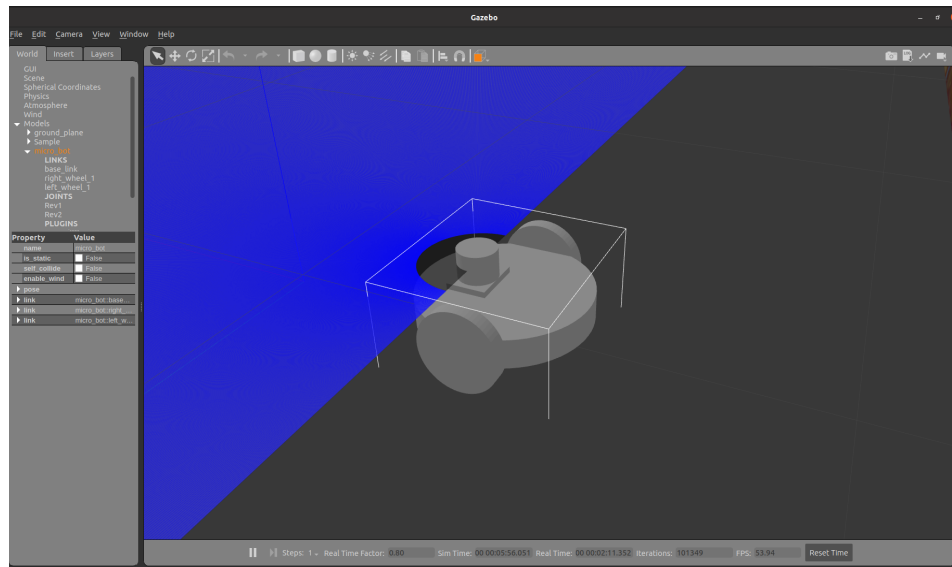


Fig 6.16 Differential Drive Robot (Gazebo Environment)

6.3 Updating apt-source list

Upfront, the `/etc/apt/source.list` is a configuration file for Linux's Advanced Packaging Tool, that holds URLs and other information for remote repositories from where software packages and applications are installed.

When considering the case scenario of SBC like Jetson it is essential to update this apt-source list because most of the time we would be using older versions of dependencies.

If apt-source list is not modified then it would not be possible to update the system and that will restrict in performing any modification in the system. It is also essential if one wants to update an existing installation, like if the OpenCV version needs to be upgraded from 3.4.6 to 3.4.8.

```
jetson-sources.list
1 # See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
2 # newer versions of the distribution.
3
4 deb http://ports.ubuntu.com/ubuntu-ports/ trusty main restricted
5 deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty main restricted
6
7 ## Major bug fix updates produced after the final release of the
8 ## distribution.
9 deb http://ports.ubuntu.com/ubuntu-ports/ trusty-updates main restricted
10 deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-updates main restricted
11
12 ## Uncomment the following two lines to add software from the 'universe'
13 ## repository.
14 ## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
15 ## team. Also, please note that software in universe WILL NOT receive any
16 ## review or updates from the Ubuntu security team.
17 # deb http://ports.ubuntu.com/ubuntu-ports/ trusty universe
18 # deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty universe
19 # deb http://ports.ubuntu.com/ubuntu-ports/ trusty-updates universe
20 # deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-updates universe
21
22 ## N.B. software from this repository may not have been tested as
23 ## extensively as that contained in the main release, although it includes
24 ## newer versions of some applications which may provide useful features.
25 ## Also, please note that software in backports WILL NOT receive any review
26 ## or updates from the Ubuntu security team.
27 # deb http://ports.ubuntu.com/ubuntu-ports/ trusty-backports main restricted
28 # deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-backports main restricted
29
30 deb http://ports.ubuntu.com/ubuntu-ports/ trusty-security main restricted
31 deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-security main restricted
32 # deb http://ports.ubuntu.com/ubuntu-ports/ trusty-security universe
```

Fig 6.17 Apt Source List for Jetson

6.4 ROS Workspace and Package Configuration

As everything in ROS is in the form of packages, it is essential to manage the Catkin workspace. There are multiple options available from `catkin_make`, `catkin build` to `catkin_make_isolated`. Generally, I used `catkin_make` to build and `catkin build` to configure my ROS workspace.

Dependency management was complex with `CMakeLists.txt` as it includes various macros ranging from adding packages, configuring them which might be needed either at build time or run time. For this purpose 3 workspace methods were used by me. One was for testing, another for development and last one to review and finalize one.

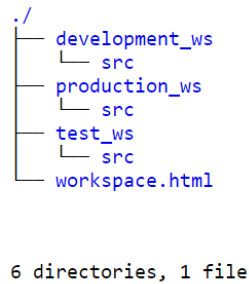


Fig 6.18 Workspace Structure For ROS

A package in ROS should have the following structure and one important consideration for Python packages is where `__init__.py` file will go. One of the solutions is to create an abstract folder with the same name as the package name and place `__init__.py` inside it. Now you can have a folder called `scripts` where you can place all of your Python executables and 3rd party modules are kept inside `src` again with abstraction of package name.

At last there is one more file required called `setup.py` in the root directory of the package which will tell the catkin system how to initialize this package.

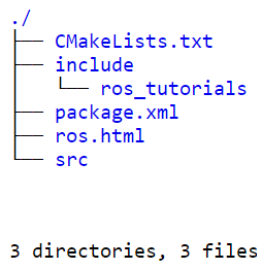


Fig 6.19 Package Structure For ROS

I found 2 ways in which a Python executable can be called. One was to add `scripts` directory path in the `setup.py` file itself and while building workspace it will store all the python scripts in a single `bin` directory. This is not a wise choice from a security point of view as well as coding point of view.

Another way is to uncomment `catkin_python_install()` from `CMakeLists.txt` of that specific package. This way catkin will know the location of the Python script and while executing will read from the `src` directory.

```
from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

d = generate_distutils_setup(
    packages=['mypkg'],
    scripts=['bin/myscript'],
    package_dir={'': 'src'})

setup(**d)
```

Fig 6.20 Setup.py File

6.5 TM4C1294 Integration with ROS Serial

6.5.1 TivaWare SDK and Cross Compiler Setup

In order to build firmware around Tiva boards, it is essential to have TivaWare SDK which comes with board specific details, header files and some examples to test. As Tiva boards are arm based, I had to configure a cross-compiler which supports arm boards. One such cross-compiler is arm-none-eabi (GNU Toolchain). After Ubuntu 14 one cannot apt-get it with PPA and hence one has to manually download it. Also, after installation symbolic links are to be configured.

```
trushant@delta:~$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1-20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

trushant@delta:~$ arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10.3-2021.10) 10.3.1 20210824 (release)
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

trushant@delta:~$ g++ --version
g++ (Ubuntu 9.4.0-1ubuntu1-20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

trushant@delta:~$ arm-none-eabi-g++ --version
arm-none-eabi-g++ (GNU Arm Embedded Toolchain 10.3-2021.10) 10.3.1 20210824 (release)
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

trushant@delta:~$
```

Fig 6.21 Check Installation Of GCC And Cross Compiler

6.5.2 Automation Script for Tiva Setup

Manual setup takes a lot of time and it is tedious. Hence, I developed a bash script in which one just needs to download the TivaWare SDK and place it in the tar_file directory. After that bash script will install and configure everything on its own.

```
#!/bin/bash
VER=10.3-2021.10
echo "Extracting..."
tar -xf tar_file/gcc-arm-none-eabi.tar.bz2
echo "Generating gcc-arm-none-eabi folder..."
mkdir gcc-arm-none-eabi
mv gcc-arm-none-eabi-*/ gcc-arm-none-eabi/
echo "moving arm-toolchain to /usr folder"
sudo mv gcc-arm-none-eabi /usr/share/
echo "Removing temporary files..."
rm -r gcc-arm-none-eabi-*
echo "Creating symbolic link of arm-none-eabi-* in /usr/bin"
sudo ln -s /usr/share/gcc-arm-none-eabi/bin/* /usr/bin/
echo "Done."
```

Fig 6.22 Bash Script For Tiva Setup

6.5.3 TM4C1294 Setup and Test

In order to develop firmware for TivaC, a new ROS workspace called emb_ws was created. Next task was to set up roserial and roserial_tivac. Roserial is the main ROS package which generates roslib which will be compiled with code for TM4C1294.

Roserial_tivac is a client library which provides us utilities built on top of catkin_make for generating axf file, flashing it and monitoring the program's memory footprint.

When using roserial_tivac it is important to build roslib first so that it can be compiled when an axf file is generated. If this is not done then custom ROS messages won't build and header not found error will emerge.

One another issue is that the first roserial_tivac package needs to be built with catkin_make and after that install it with catkin_make install. Only

after this custom rosserial based code could be written and executed in another package.

After the entire setup was complete I was able to flash a simple blink program from the TivaWare SDK and flash it using lm4flash. To test integration with ROS was successful or not I programmed a Tiva based publisher as follow:

```
GNU nano 4.8
/*
 * rosserial Publisher Example
 * Prints "hello world!"
 */
#include <ros.h>
#include <std_msgs/String.h>

ros::NodeHandle nh;
std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char hello[13] = "hello world!";

void setup()
{
  nh.initNode();
  nh.advertise(chatter);
}

void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(1000);
}
```

Fig 6.23 TivaC ROS Publisher

This is written in the scripting language used by Energia and I was able to receive output as expected.

6.6 Conveyor Velocity Estimation with ArUco Marker

6.6.1 Camera Calibration

When considering pinhole cameras, they have a major issue of distortion in the images. There can be radial or tangential distortion in the images. Hence, to rectify these images we need a distortion coefficient and camera

matrix. To get intrinsic and extrinsic properties camera calibration is essential.

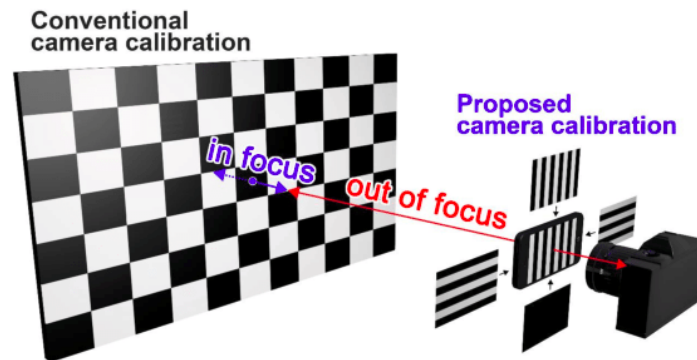


Fig 6.24 Camera Calibration Procedure
“Courtesy of (Hyowon et al. 2015)”

In order to perform camera calibration I found a package in ROS called camera_calibration which provided me with output in the form of a YAML file.

6.6.2 ArUco Marker Generation and Detection

After the camera calibration, I wrote a python script which will generate an ArUco maker on the basis of input dictionary type and class. Here, I am using the ArUco module by OpenCV.

```
trushant@delta:~/aruco_maker/aruco_marker_generation/tags$ ls
DICT_4X4_50_id1.png DICT_4X4_50_id2.png DICT_4X4_50_id3.png DICT_4X4_50_id4.png
trushant@delta:~/aruco_maker/aruco_marker_generation/tags$ feh DICT_4X4_50_id1.png
```

The screenshot shows a terminal window with the following commands and output:

```
trushant@delta:~/aruco_maker/aruco_marker_generation/tags$ ls
DICT_4X4_50_id1.png DICT_4X4_50_id2.png DICT_4X4_50_id3.png DICT_4X4_50_id4.png
trushant@delta:~/aruco_maker/aruco_marker_generation/tags$ feh DICT_4X4_50_id1.png
```

Below the terminal output, a window titled 'feh [1 of 1] - DICT_4X4_50...' displays a white ArUco marker on a black background.

Fig 6.25 ArUco Marker Generation Script

With the help of a camera calibration file I was able to create another script for ArUco Marker pose estimation which gave coordinates of markers with reference to the camera in mm.

```
trushant@delta:~/aruco_maker$ ls
aruco_marker_generation  aruco_tracker.py  detect_aruco_video.py  video_testing.py
trushant@delta:~/aruco_maker$ tree
.
├── aruco_marker_generation
│   ├── opencv_generate_aruco.py
│   ├── README.md
│   └── tags
│       ├── DICT_4X4_50_id1.png
│       ├── DICT_4X4_50_id2.png
│       ├── DICT_4X4_50_id3.png
│       └── DICT_4X4_50_id4.png
├── aruco_tracker.py
├── detect_aruco_video.py
└── video_testing.py

2 directories, 9 files
trushant@delta:~/aruco_maker$
```

Fig 6.26 ArUco Marker Detection Script Directory

Now, the above test was done with Python, and now I had to integrate it with ROS. To integrate the USB camera with ROS I used the `usb_cam` package in ROS and passed the `camera_configuration` file to `camera_info_manager`.

Atlast I programmed a launch file to execute `usb_cam`, `aruco_detection` and `img_view` node which gave following output:

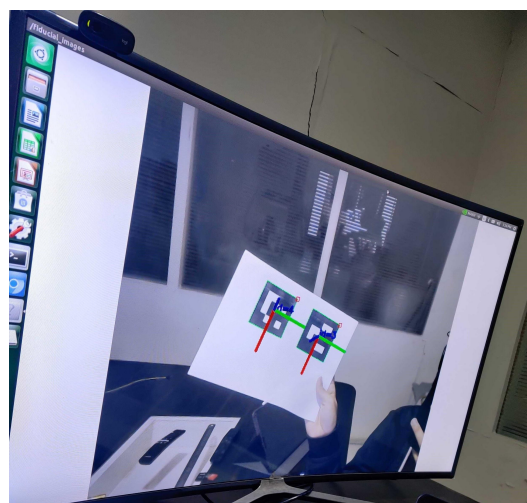


Fig 6.27 ArUco Marker Pose Estimation

6.6.3 Conveyor Velocity Estimation

After receiving pose estimation of ArUco Marker with USB camera, I integrated it with Pylon GiGe camera and performed camera calibration. One issue I faced with the Pylon camera was that detection of markers was not steady and to solve this problem I integrated image_proc package which rectified output images from camera. Even after this another issue of large image overhead was a problem. For this instead of using raw transport I started working with compressed transport. This reduced bandwidth of the system significantly.

Finally I was able to test the velocity of the conveyor belt with the help of ArUco marker. The velocity that I received was in about 2% error range when validated with external optical encoder.

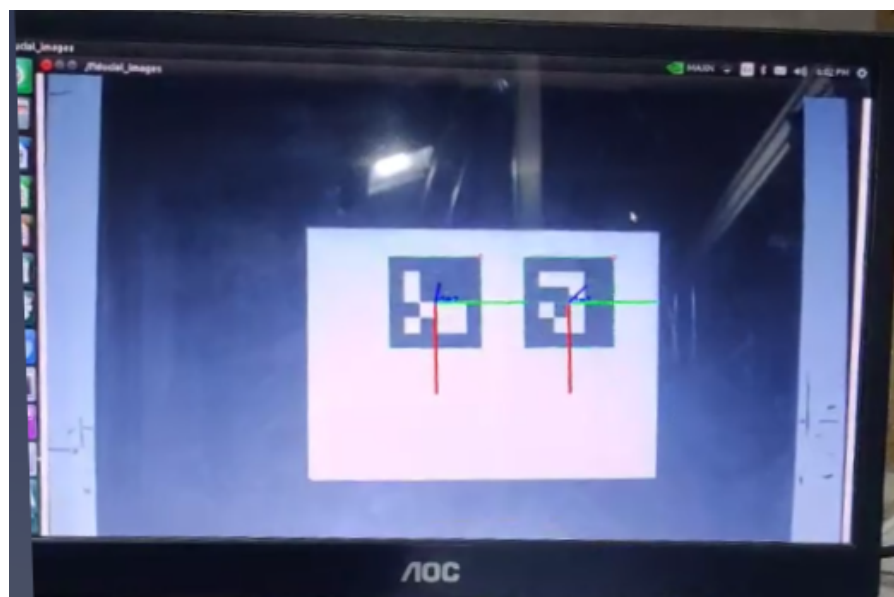


Fig 6.28 Conveyor Velocity Estimation With ArUco Marker

6.7 Operator's UI Integration with ROS

As the operator will need to interact with the robot, I had to integrate Web based UI with rosbridge. For testing purposes I developed a sample HTML document

and connected a web socket with it through which I was able to communicate with the ROS system. After that I was able to integrate ROS messages and visualize them on the Web UI.

In order to update system information on the operator's UI. I developed a diagnostic node which will subscribe messages from the ROS system and publish them to the UI. With this I was able to map those data to UI tags and visualize them.

String data transmission to rosbridge was simple but I had some issue with image stream visualization on the UI. I was trying to send raw_image to rosbridge which was working but rosbridge won't be able to store it in placeholder and display it on UI. This is because message serialization protocol for images only considers compressed images.

For launching Web UI a server was required which will run locally on SBC. For this I used the http.server library in Python. There were two options available for the routing server. One was using localhost with which no one will be able to access it even in the same gateway and another option was to use IP given to the device by router through DNS. But the issue with this approach was that anyone in the same gateway with the IP of the device can access the operator's UI. Considering the security I fixated on the 1st approach.

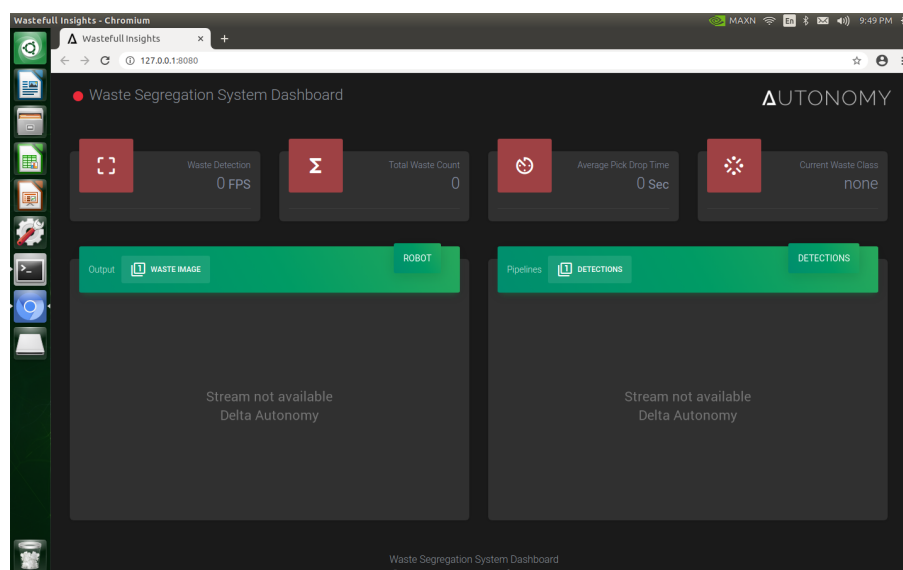


Fig 6.29 Operator's UI

6.8 Modified ROS Software Architecture

Following were some of the major modifications done in the Software Architecture of the robot:

- Integration of Scheduler to optimize pick and place
- UUID Assignment of each detected waste
- Single ROS message and srv generation package
- Faster computation with numpy
- Customized messages and services
- Dynamic time and velocity parameters instead of static
- Structured workspace and packages
- Customized Kalman Tracker
- Operator's UI for robot and detection control with security

Chapter 7

Testing

7.1 Testing Plan

I used a unit testing approach throughout the internship. Every module that I wrote was tested in an iterative manner. All the tests were done on a video feed and once the entire system gave satisfactory results, real-time testing took place.

ROS provides two testing platforms. Gtest for testing C++ based code and nosetest for Python. I used rosunit and rostest in some cases to validate actual and expected output in the system.

ROS also has two different types of testing constructs, rosunit is for testing functionalities inside a single node and rostest in between multiple nodes.

7.2 Test Result and Analysis

System was able to pass almost all the test cases except in some cases OpenCV resize function would fail. To overcome this a check for a non null frame was kept in place. Another important parameter was the transmission delay in the entire system with frame delay. As it is dynamic in nature a separate node which acts as an evaluator was designed to take care of it (“Quality/Tutorials/UnitTesting” 2019).

7.2.1 Test Cases

While performing unit tests, some tests were very crucial and they got bundled into test suites. Following are some of the important test cases:

- Frame captured by camera is not none
Every frame of a camera once resized needs to pass conversion test or else the AI model won't be able to detect waste optimially.

- CvBridge is successfully able to convert camera images to ROS message type with specific encoding
- Time taken by detection model is variable in nature, but there is an upper bound to it which should not be crossed
- UUID generated by type uuid4 are unique in nature
Unique IDs generated by uuid modules depend on system configuration and if they get duplicated considering the time parameter of the system or device's network details get exposed then it will be a security concern.
- No message is being dropped while transmission
Rate of publishing data and that of subscribing needs to be in coherence with each other so that there is no loss of data.
- Message buffer is considered accordingly in each node
- Kalman Tracker is effectively able to track waste without being affected by increased detections
- On selecting any particular sub-sequence, it should have the least loss count. Loss count is generally given when selecting the nth element in the sequence won't let us select (n+1)th element because it will be out of the robot's ROI.
- Average velocity of the conveyor is within 2% of the value derived by external optical encoder
- Scheduler is able to prioritize sequences with least effort and avoids local maxima

Chapter 8

Conclusion and Discussion

8.1 Result and Analysis of Project Viabilities

Right now there are two architectures on which the system can run. Previous architecture relied on sequential flow of execution due to which it will take a greedy approach of picking up waste. Basically, there was no block which would validate and compare different picks in the system. Consider a case scenario where an object sequence of ID (1,2,3) is detected in a frame and when the robot picks an object with ID 1, meanwhile an object with ID 2 will be out of the robot's region of interest. So, it will miss objects with ID 2. This is the issue with greedy approaches.

To overcome the local maxima problem, a new architecture is designed considering that it will perform comparison between permutations of sequences for a given number of objects in the frame. Consider the previous example, but in this architecture permutations of ID (1,2,3), (1,3,2) and so on will be evaluated. And the sequence in which we are able to pick the maximum number of objects is finalized. This will overcome the issue of local maxima.

8.2 Workflow Problems Encountered and Possible Solutions

One of the major issues that I faced during the internship was in regards to balancing time between research and implementation. I had to optimize the system and I found multiple research papers, some of which were efficient and others might not even fit our case scenario. This was a major hindrance to me in the first month of internship but later on I was able to mitigate this issue.

Not using git extensively leads me to the situations where I have to perform the same task again and again on multiple systems. For example I modified rospkg paths in my local system then I have to do the same on another Jetson Nano and last in Jetson NX. This issue got later resolved at the end of internship and the

solution was to disintegrate all the packages in the form of git repos. Every time a change is made locally it is committed on the git and with simple commands like git pull all the system will have uptodate code.

Initially I had a tendency to study the entire sub-system or functionality in depth to such an extent that I will spend countless hours reading its implementation. There was a bright side to this as I was able to think of various approaches to optimize the system but this took my time from the current task which I was performing. To mitigate this issue I will consider sub-blocks of the system as black box and consider only input output parameters.

8.3 Summary / Conclusion of Internship

I can honestly say that my time spent interning with Wastefull Insights resulted in one of the best times of my life. Not only did I gain practical skills but also had the opportunity to meet many fantastic people. The atmosphere at the office was always welcoming which made me feel right at home or at yet another home. Additionally, I felt like I was able to contribute to the company by assisting and working on optimization of their waste segregation robot. For example, I restructured underlying ROS architecture to make it more flexible, modular and robust. Also, developed an Operator's UI with which one can control both robot and camera detection. In addition to these tasks, I developed velocity estimation algorithm and optimization algorithm to increase overall throughput of the robot.

While I was able to learn a lot from normal office life, my two most memorable days were the day I joined and the day I tested my architecture on the robot. From not knowing anything about the system to restructuring the entire architecture with optimization gave me a huge sense of accomplishment and confidence.

Overall, my internship at Wastefull Insights has been a success. I was able to gain practical skills, work in a fantastic environment, and make connections that will last a lifetime. I could not be more thankful to Ms. Manali Agarwal and Mr. Rishabh Shah for providing me this opportunity and pushing me to break my self

perceived limitations. I also received a full time offer to work with them, looking forward to contributing to this case of “Zero Waste World”.

8.4 Limitation and Future Enhancement

The newly implemented architecture is a progression on the older one but there are some limitations to it. As we are performing permutations for a given set of objects in a frame, it might happen that there are say 10 detected objects. For instance, if we select a sequence of 10 objects then total permutations will be 3628800 sequences. It is not possible to evaluate all these sequences in a period of 3 seconds which is time for objects in the camera's region of interest.

Inorder to solve this problem there needs to be a reduces which will evaluate previous drop position of robot and store evaluation time of the previous element in a specific sequence. With this we wouldn't have to calculate all the sequences. Also, we can add a filter for minimum acceptable loss count.

References

Andrew, West, Arvin Farshad, Martin Horatio, Watson Simon, and Lennox Barry. 2018.

ROS Integration for Miniature Mobile Robots.

Bowman, James, and Patrick Mihelich. 2010. "cv_bridge." ROS Wiki.

http://wiki.ros.org/cv_bridge

Central Pollution Control Board. 2021. "Annual Report 2019-20 on Implementation of Plastic Waste Management Rules, 2016." Central Pollution Control Board.

https://cpcb.nic.in/uploads/plasticwaste/Annual_Report_2019-20_PWM.pdf.

"C++ Introduction." n.d. W3Schools. Accessed April, 2022.

https://www.w3schools.com/cpp/cpp_intro.asp.

"Cross Compilation Toolchain for ARM - Example with Raspberry Pi." n.d.

Microcontrollers Lab. Accessed April, 2022.

<https://microcontrollerslab.com/cross-compilation-toolchain-for-arm-example-with-raspberry-pi/>.

"Cross compiler." n.d. Wikipedia. Accessed April, 2022.

https://en.wikipedia.org/wiki/Cross_compiler.

Debapriya, Chatterjee, Deorio Andrew, and Bertacco Valeria. 2009. *Event-driven*

gate-level simulation with GP-GPUS. 10.1145/1629911.1630056.

"EK-TM4C1294XL Evaluation board | TI.com." 2014. Texas Instruments.

<https://www.ti.com/tool/EK-TM4C1294XL>.

"EK-TM4C1294XL Texas Instruments." n.d. Mouser. Accessed April, 2022.

<https://www.mouser.in/ProductDetail/Texas-Instruments/EK-TM4C1294XL?qs=YaA%252B3EhhSt21BnIfZSMnRQ%3D%3D>.

“Embedded Systems Developer Kits & Modules from NVIDIA Jetson.” n.d. Nvidia.

Accessed April, 2022.

<https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/>.

“Energia IDE.” n.d. Energia IDE. Accessed April, 2022. <https://energia.nu/>.

Foote, Tully. 2018. “ROS Melodic Morenia Logo and Tshirt Campaign.” ROS.

<https://www.ros.org/news/2018/04/ros-melodic-morenia-logo-and-tshirt-campaign.html>.

Git. n.d. Git SCM. Accessed April, 2022. <https://git-scm.com/>.

Han, Shuai D., Si W. Feng, and Jingjin Yu. 2019. “Toward Fast and Optimal Robotic Pick-and-Place on a Moving Conveyor.” *CoRR* abs/1912.08009.

<http://arxiv.org/abs/1912.08009>.

Hyowon, Ha, Bok Yunsu, Joo Kyungdon, Jung Jiyoung, and Kweon Inso. 2015. *Accurate Camera Calibration Robust to Defocus Using a Smartphone*.

10.1109/ICCV.2015.101.

“JavaScript | MDN.” 2022. MDN.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

“Jetson Xavier NX for Embedded & Edge Systems | NVIDIA.” n.d. Nvidia. Accessed April, 2022.

<https://www.nvidia.com/en-gb/autonomous-machines/embedded-systems/jetson-xavier-nx/>.

“Kalman filter.” n.d. Wikipedia. Accessed April, 2022.

https://en.wikipedia.org/wiki/Kalman_filter.

“List of build automation software.” n.d. Wikipedia. Accessed April, 2022.

https://en.wikipedia.org/wiki/List_of_build_automation_software.

Mace, Jonathan. 2017. "rosbridge_suite." ROS Wiki. http://wiki.ros.org/rosbridge_suite.

"Machine learning." n.d. Wikipedia. Accessed April, 2022.

https://en.wikipedia.org/wiki/Machine_learning.

Marcin, Szlenk, Zielinski Cezary, Figat Maksym, and Kornuta Tomasz. 2015.

"Reconfigurable Agent Architecture for Robots Utilising Cloud Computing."

Advances in Intelligent Systems and Computing 351 (1).

10.1007/978-3-319-15847-1_25.

"NumPy Library." n.d. NumPy.org. Accessed April, 2022. <https://numpy.org/>.

Nvidia. n.d. "CUDA Zone." NVIDIA Developer. Accessed April, 2022.

<https://developer.nvidia.com/cuda-zone>.

"NVIDIA Jetson Nano Developer Kit." n.d. NVIDIA Developer. Accessed April, 2022.

<https://developer.nvidia.com/embedded/jetson-nano-developer-kit>.

"OpenCV: Detection of ArUco Markers." n.d. OpenCV documentation. Accessed April,

2022. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

"OpenCV Library." n.d. OpenCV. Accessed April, 2022. <https://opencv.org/about/>.

Open Robotic. n.d. "Robot Operating System." ROS: Home. Accessed April, 2022.

<https://www.ros.org/>.

Programiz. n.d. "Learn C Programming." Accessed April, 2022.

<https://www.programiz.com/c-programming>.

"Quality/Tutorials/UnitTesting." 2019. ROS Wiki.

<http://wiki.ros.org/Quality/Tutorials/UnitTesting>.

"ROS Actions Overview - MATLAB & Simulink." 2022. MathWorks.

<https://www.mathworks.com/help/ros/ug/ros-actions.html>.

“ROS/Concepts - ROS.” 2014. ROS Wiki. <http://wiki.ros.org/ROS/Concepts>.

“roslibjs.” 2019. ROS Wiki. <http://wiki.ros.org/roslibjs>.

“rosterial.” 2018. ROS Wiki. <http://wiki.ros.org/rosterial>.

Sears, Addison. 2020. “How to Create a Publisher Node in ROS Noetic – Automatic Addison.” Automatic Addison.

<https://automaticaddison.com/how-to-create-a-publisher-node-in-ros-noetic/>.

Sears, Addison. 2020. “How to Create a Service in ROS Noetic – Automatic Addison.” Automatic Addison.

<https://automaticaddison.com/how-to-create-a-service-in-ros-noetic/>.

“uuid — UUID objects according to RFC 4122 — Python 3.10.4 documentation.” n.d. Python Docs. Accessed April, 2022. <https://docs.python.org/3/library/uuid.html>.

Vankeirsbilck, Jens. 2020. “Introduction of Robot Operating Systems 2: ROS2 – SAFER AUTONOMOUS SYSTEMS.” SAFER AUTONOMOUS SYSTEMS.

<https://etn-sas.eu/2020/03/23/introduction-of-robot-operating-systems-2-ros2/>.

van Rossum, Guido. n.d. “Python (programming language).” Wikipedia. Accessed April, 2022. [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)).

Wastefull Insights. 2019. “Wastefull Insights | LinkedIn.” LinkedIn India.

<https://in.linkedin.com/company/wasteful-insights>.

“What is Build Tool? - Definition from Techopedia.” 2011. Techopedia.

<https://www.techopedia.com/definition/16359/build-tool>.

Y. Maruyama, S. Kato, and T. Azumi. 2016. *Exploring the Performance of ROS2*. N.p.: In Proc. of ACM EMSOFT.