Department of Electronic & Telecommunication Engineering,
University of Moratuwa,
Sri Lanka.

Project Report part-2
Comprehensive Design Documentation

Robot LUNA the Restaurant Robot

Group Members:

210174X	Gallella MMHHB
210218M	Herath HMSI
210205V	Haputhanthri HLNB
210069F	Basnayake BYN
210629A	Thamirawaran S

Submitted in partial fulfillment of the requirements for the module
EN2160 Electronic Design Realization

Date: 2023.04.02

Contents

Abstract
"Robot LUNA", a Restaurant waiter robot, uses a dual camera setup (wide-angle camera and a Kinect-2 depth camera) for navigation in the restaurant environment with 3D point cloud-based calculations. Its enhanced stability circuits ensure safe food delivery without spillage. Internally, it uses three Linux SBCs for parallel processing and an Atmega2560-based custom PCB to get sensor/encoder readings and to control motor drivers. It communicates with a restaurant's Computer server with Robot Operating System (ROS) via local wifi for path planning and precise navigation to order locations. This makes LUNA a reliable and efficient addition to the restaurant staff.

Figure 1: Project logo

1 Introduction to overall structure of the final design

The development of Robot LUNA, has been structured into several distinct subsystems to ensure a comprehensive and efficient approach. This division not only provided a systematic pathway for development but also fostered effective collaboration among team members by clearly dividing responsibilities and deadlines. The final design of Robot LUNA integrates multiple technologies and components, each playing a crucial role in its functionality and performance.

The 9 subsections of this document encompass various essential elements of LUNA's design and operation, here's a brief introduction to each subsection which we give more comprehensive details later.(Refer to chapters 2.1,2.2,....2.9 corresponding to Subsystems 1 to 9 in this design documentation)

1. **Central Restaurant Computer and Local WiFi Network:** The central restaurant computer serves as the main control hub, interfacing with Robot LUNA via a local WiFi network. This setup enables seamless communication for path planning and order management, ensuring that LUNA can navigate the restaurant environment effectively. For more details, refer to Chapter 2.1.

2. **Setting Up the Workspace:** Setting up the workspace involves configuring the environment to support LUNA's operations, including network configurations and initial setup procedures. This step is critical for ensuring that all components can communicate and function as intended. For more details, refer to Chapter 2.2.

3. **ROS 2 Control System Setup:** The Robot Operating System (ROS) 2 provides the framework for LUNA's control system, enabling advanced functionalities such as real-time sensor data processing and path planning. This subsystem ensures that LUNA can make intelligent decisions based on its environment. For more details, refer to Chapter 2.3.

4. **Serial Communication with PCB:** The custom PCB, based on the Atmega2560, communicates with LUNA's computer unit via serial communication. This subsystem is responsible for relaying sensor and encoder readings, which are essential for accurate motor control and navigation. For more details, refer to Chapter 2.4..

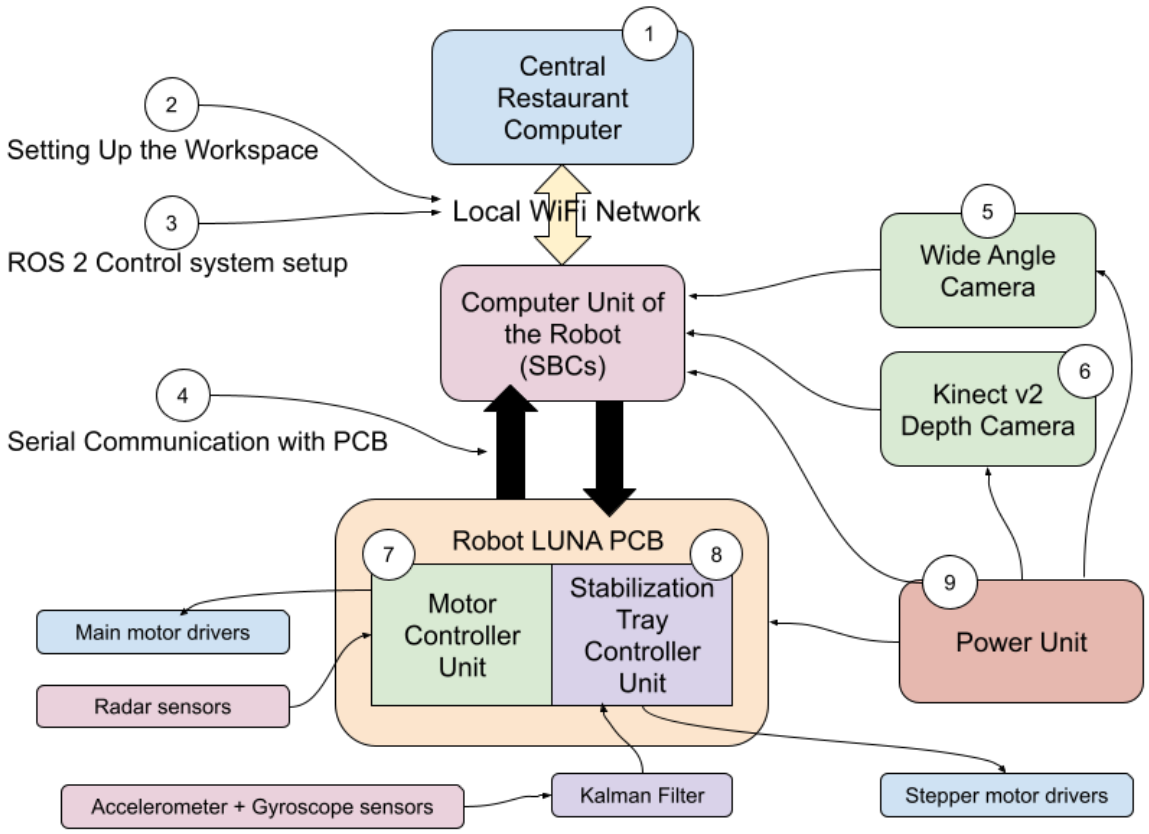


Figure 2: Sub Units Diagram

5. **Wide Angle Camera:** The wide-angle camera is a key component of LUNA's dual-camera setup, providing a broad field of view for navigation and obstacle avoidance. This camera works in conjunction with other sensors to create a comprehensive understanding of the environment. For more details, refer to Chapter 2.5.

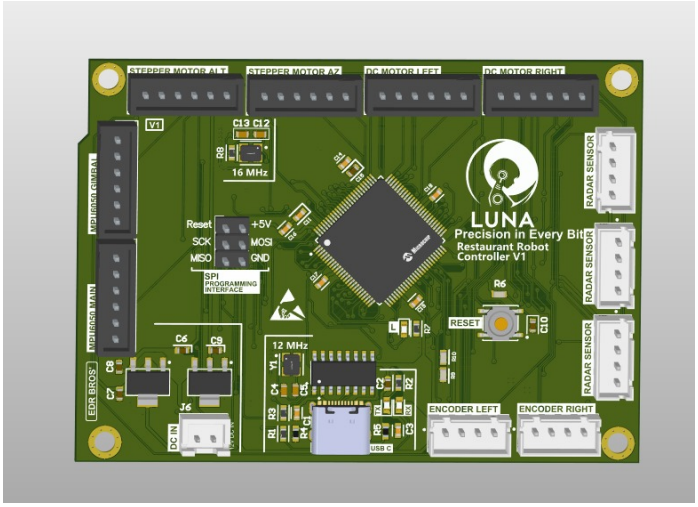
6. **Kinect v2 Depth Camera:** The Kinect v2 depth camera captures 3D point cloud data, which is crucial for precise navigation and obstacle detection. This data allows LUNA to understand the spatial relationships within its environment, enhancing its ability to deliver orders accurately. For more details, refer to Chapter 2.6.

7. **Motor Controller Unit:** The motor controller unit, integrated into the Robot LUNA PCB, manages the main motor drivers, enabling precise control over LUNA's movements. This unit ensures smooth and accurate navigation throughout the restaurant. For more details, refer to Chapter 2.7.

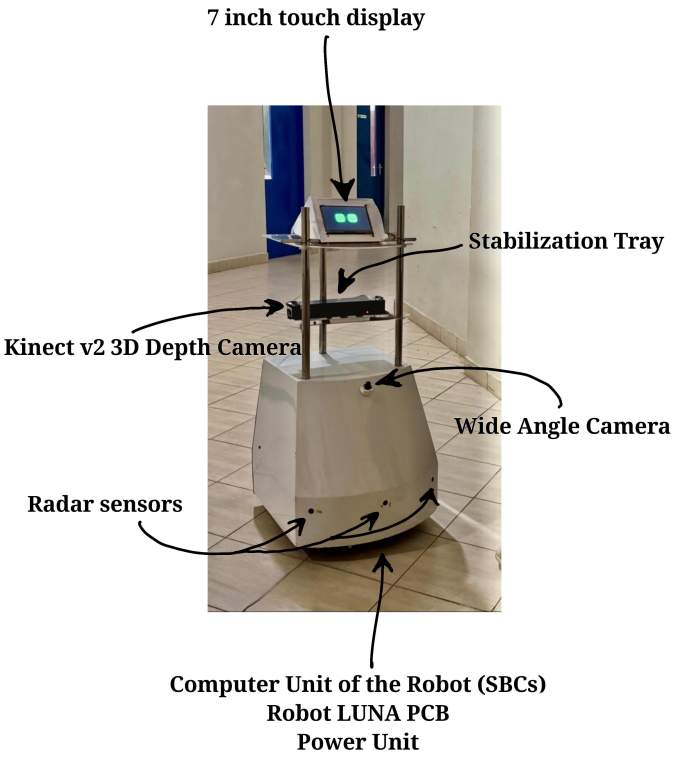
8. **Stabilization Tray Controller Unit:** This unit is responsible for maintaining the stability of the delivery tray, utilizing data from the accelerometer and gyroscope sensors. The Kalman filter processes this data to make real-time adjustments, preventing spillage during movement. For more details, refer to Chapter 2.8.

9. **Power Unit:** The power unit supplies the necessary power to all of LUNA's components, ensuring consistent and reliable operation. This subsystem is designed to provide stable power even under varying loads, supporting the robot's demanding functions. For more details, refer to Chapter 2.9.

Chapter 7 and 8 functionalities are controlled by a custom-designed PCB which includes two main units: the Motor Controller Unit and the Stabilization Tray Controller Unit. Below are the front view of the PCB:



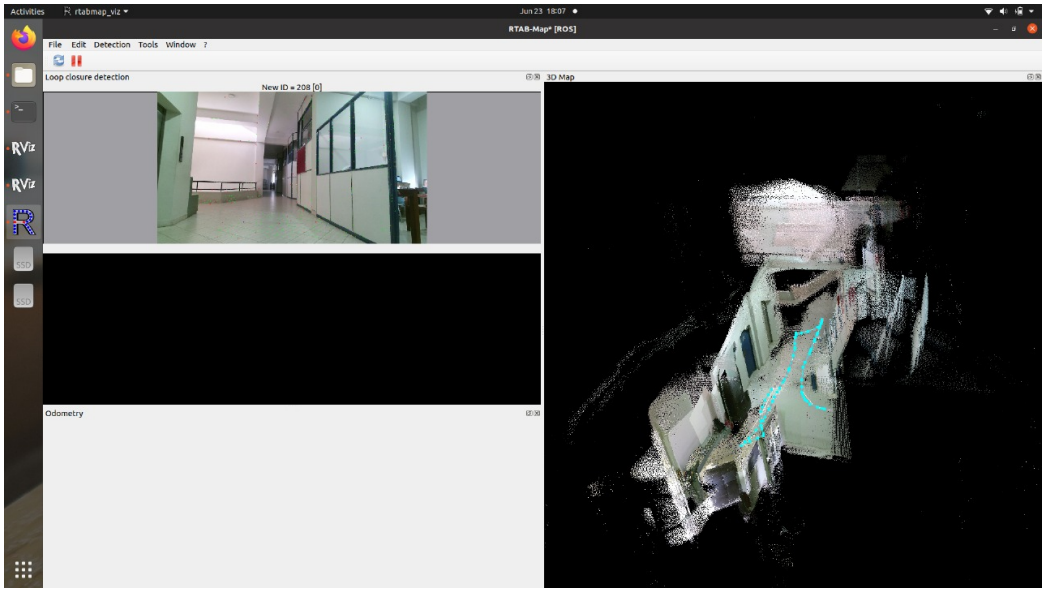
PCB front view



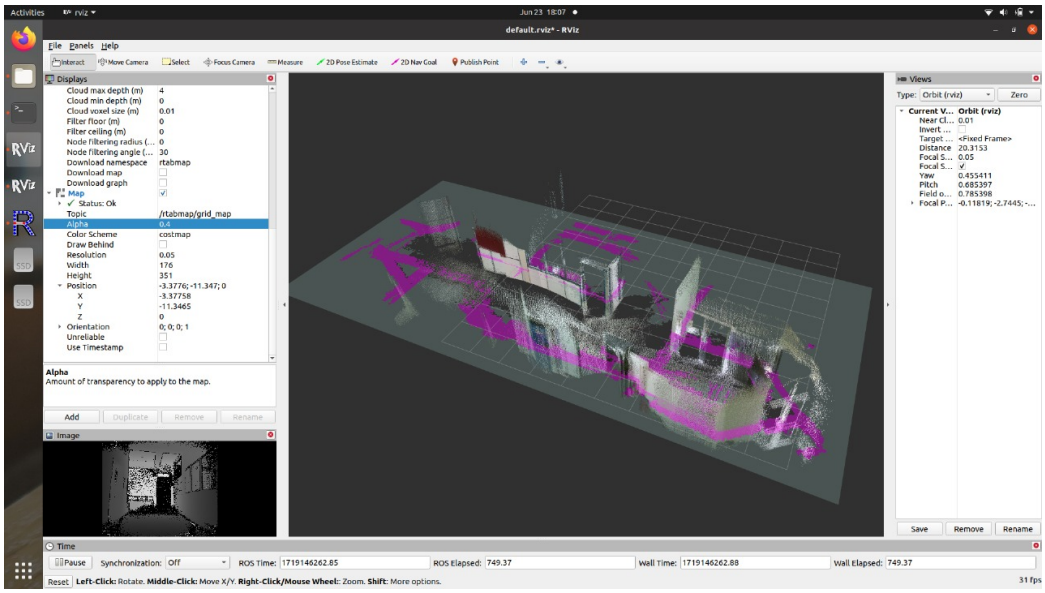
Robot LUNA Key Components

By addressing these key sections individually, the overall complexity of the project was effectively managed, allowing for a streamlined development process and ultimately leading to the successful realization of Robot LUNA.

Robot LUNA's Kinect V2 3D depth camera plays a crucial role in ROS which communicates with a restaurant's Computer server via local wifi for path planning and precise navigation to order locations.



Path planning in the ENTC department 2nd floor



ENTC department 2nd floor full 3D construction from Robot LUNA

Dividing the project into these 9 subsystems allowed for a focused and systematic approach to development. Each component received the attention it deserved, facilitating efficient collaboration among team members with clear responsibilities and deadlines. This structure ensured that every aspect of Robot LUNA's design was thoroughly developed and optimized, resulting in a reliable and efficient addition to the restaurant.

2 Comprehensive Design details

Refer this youtube channel for related videos.

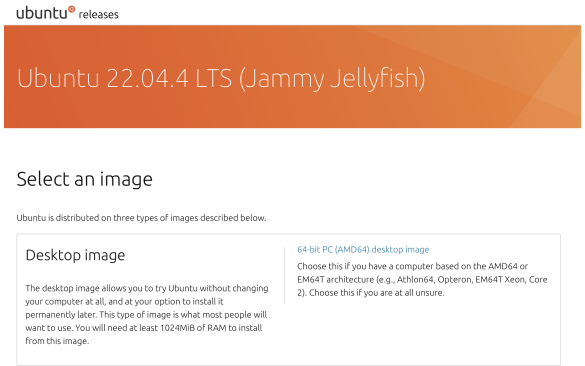
2.1 Setting Up Central Restaurant Computer

Installing Ubuntu

Ubuntu 22.04

As the recommended operating system for ROS 2 Iron is Ubuntu 22.04 Jammy Jellyfish, we have to set up Ubuntu 22.04 Jammy Jellyfish on both your AMD-based computer and Raspberry Pi/Jetson Nano (ARM version). Follow these steps:

1. **Download the ISO Image**: Download the .iso image files for Ubuntu 22.04 from the official site: https://releases.ubuntu.com/jammy/.



Ubuntu download page

2. **Create a Bootable USB Drive**:
   (a) Download Rufus, a tool for creating bootable USB drives, from https://rufus.ie/en/#google_vignette.

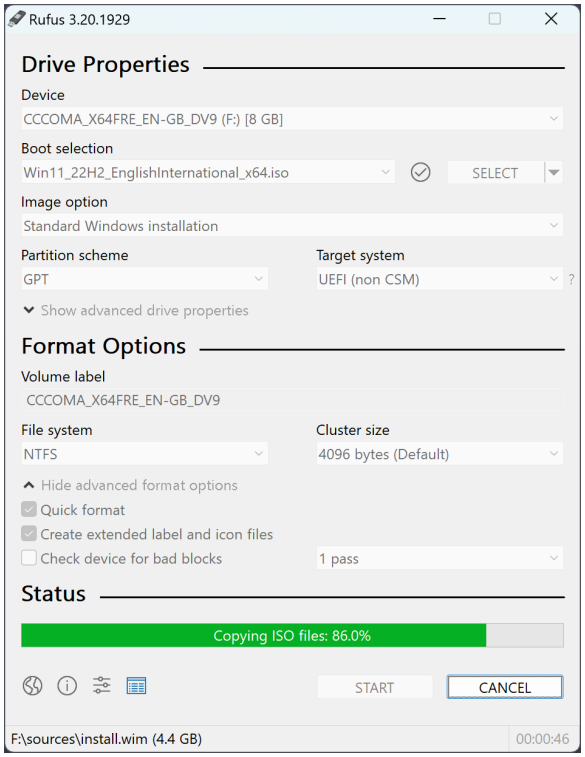
(b) Install Rufus on your computer.

(c) Insert the USB drive you want to make bootable.

(d) Open Rufus, select your USB drive from the Device dropdown menu.

(e) Under Boot selection, choose the .iso file you downloaded for Ubuntu 22.04.

(f) Follow the prompts to create the bootable USB drive.



Rufus software

3. **Booting the PC from the USB Drive**:

   (a) Insert the bootable USB drive into the computer where you want to install Ubuntu 22.04.

   (b) Restart the computer and enter the boot options menu. The key to press for entering the boot menu may vary depending on your computer's manufacturer (for some it is F8; you can find the specific key for your computer online).

   (c) Select the USB drive from the boot options menu.

   (d) Follow the on-screen instructions to install Ubuntu 22.04. If you are installing it on a specific partition of your hard disk, select that partition during the installation process.

4. **Follow the Installation Tutorial**: For detailed installation instructions, you can follow the official Ubuntu tutorial starting from step 6: https://ubuntu.com/tutorials/install-ubuntu-desktop#6-type-of-installation.

By following these steps, you will be able to install Ubuntu 22.04 Jammy Jellyfish on both your AMD-based PC and Raspberry Pi/Jetson Nano.

Installing ROS 2 on the Device

Ros2 Iron

System Setup

Set Locale Ensure you have a locale that supports UTF-8. In minimal environments (such as a Docker container), the locale may be set to something minimal like POSIX. The following settings are tested, but any UTF-8 supported locale should work.

```
locale  # check for UTF-8

sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

locale  # verify settings
```

Enable Required Repositories Add the ROS 2 apt repository to your system.
First, ensure that the Ubuntu Universe repository is enabled:

```
sudo apt install software-properties-common
sudo add-apt-repository universe
```

Add the ROS 2 GPG key with apt:

```
sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.
    key -o /usr/share/keyrings/ros-archive-keyring.gpg
```

Add the repository to your sources list:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
    ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os
    -release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.
    list.d/ros2.list > /dev/null
```

Install Development Tools (Optional) If you plan to build ROS packages or do development, install the development tools:

```
sudo apt update && sudo apt install ros-dev-tools
```

Install ROS 2 Update your apt repository caches:

```
sudo apt update
```

Ensure your system is up to date:

```
sudo apt upgrade
```

Warning: Due to early updates in Ubuntu 22.04, it is important that systemd and udev-related packages are updated before installing ROS 2. Installing ROS 2's dependencies on a freshly installed system without upgrading can trigger the removal of critical system packages. Please refer to https://github.com/ros2/ros2/issues/1272 and https://bugs.launchpad.net/ubuntu/+source/systemd/+bug/1974196 for more information.

```
sudo apt install ros-iron-desktop
```

ROS-Base Install (Bare Bones): Communication libraries, message packages, command line tools. No GUI tools

```
sudo apt install ros-iron-ros-base
```

Install Additional RMW Implementations (Optional) The default middleware that ROS 2 uses is Fast DDS. However, the middleware (RMW) can be replaced at runtime. See the guide on how to work with multiple RMWs for more information.

Setup Environment Set up your environment by sourcing the following file. Add this line to your `.bashrc` to make it permanent.

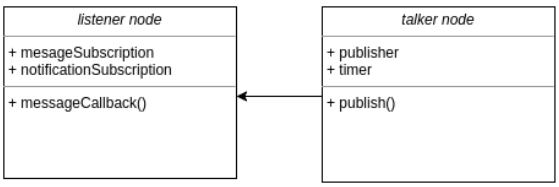
```
# Replace ".bash" with your shell if you're not using bash
# Possible values are: setup.bash, setup.sh, setup.zsh
echo "source /opt/ros/iron/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Try Some Examples If you installed `ros-iron-desktop`, you can try some examples.
In one terminal, source the setup file and then run a C++ talker:

```
source /opt/ros/iron/setup.bash
ros2 run demo_nodes_cpp talker
```

In another terminal, source the setup file and then run a Python listener:

```
source /opt/ros/iron/setup.bash
ros2 run demo_nodes_py listener
```

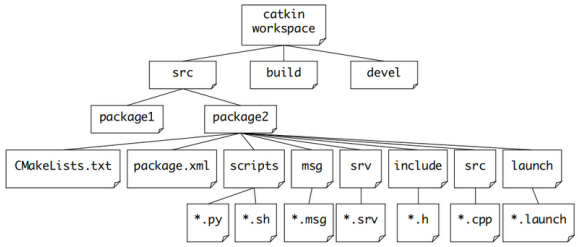


Ros 2 Talker and Listener inner view

You should see the talker publishing messages and the listener receiving those messages, verifying that both the C++ and Python APIs are working properly. Hooray!

2.2 Setting Up the Workspace

2.2.1 Setting Up the Catkin Workspace



Catkin workspace

After installing **ROS 2 Iron**, you need to set up the colcon workspace by following these commands:

```
cd ~
mkdir -p colcon_ws/src
cd colcon_ws
colcon build
```

You can install Visual Studio Code (VS Code) IDE for writing ROS scripts by using these commands and install the necessary extensions for Python and ROS (by Microsoft) to create and handle nodes:

```
sudo apt install snapd
sudo snap install --classic code
```

To set up the communication between your computer and a Single Board Computer (SBC), add these lines at the end of the `.bashrc` file. Replace `masterip.local` with the IP address of the computer that will run as the master:

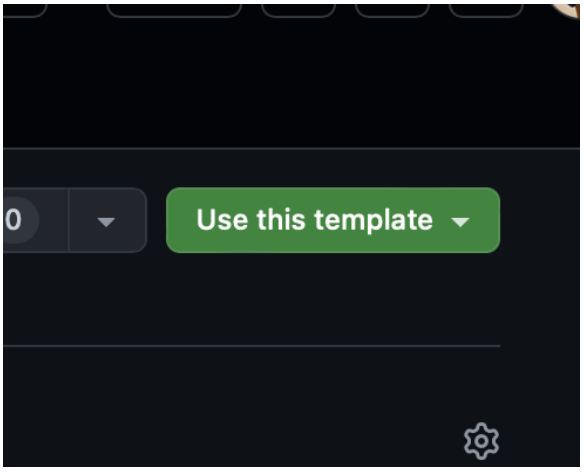
```
echo 'export ROS_DOMAIN_ID=0' >> ~/.bashrc
echo 'export ROS_MASTER_URI=http://masterip.local:11311' >> ~/.bashrc
source ~/.bashrc
```

These steps ensure your ROS 2 Iron environment is properly set up and configured for development and communication.

2.2.2 Setting up the Repository

GitHub Setup

1. Create a GitHub account and clone the project template.
   https://github.com/YasiruDEX/LUNA_workspace_template



Github template clone button

2. Inside launch/rsp.launch.py

```
pkg_path = os.path.join(get_package_share_directory(    my_bot    ))
Inside package.xml
<name>my_bot</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="my_email@email.com">MY NAME</maintainer>
```

Name the project by Changing the following inside the temaplate, customize, and commit changes. Changing "my_bot" to desired project/package name.

3. Inside CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(my_bot)
```

Workspace Preparation

Set up a development machine with Ubuntu and ROS2 iron. Create a workspace, clone the repository, and build the project with the following command from the terminal.
To create a workspace, clone the repository, and build the project in ROS 2, you can follow these commands in the terminal:

1. Open a terminal* and navigate to your home directory (or wherever you prefer to create your workspace):

```
cd ~
mkdir -p dev_ws/src
cd dev_ws/src
```

2. Clone the repository

```
git clone YOUR_GITHUB_REPOSITORY_URL
cd ~/dev_ws
```

3. Source and build the workspace

```
source /opt/ros/iron/setup.bash
colcon build --symlink-install
source install/setup.bash
```

After running these commands, your ROS 2 workspace should be set up with the cloned repository, and the project should be built and ready to use.

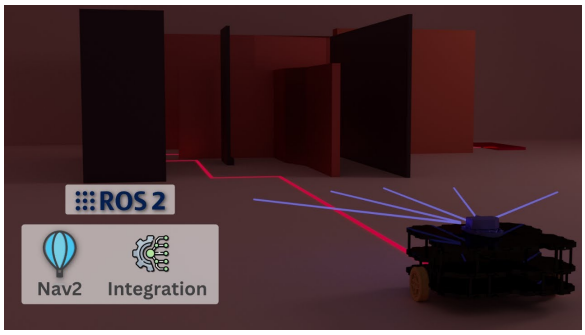
2.3 ROS 2 Control system setup

2.3.1 Project Goals

Teleoperation

Control the robot remotely using a phone with camera feedback.

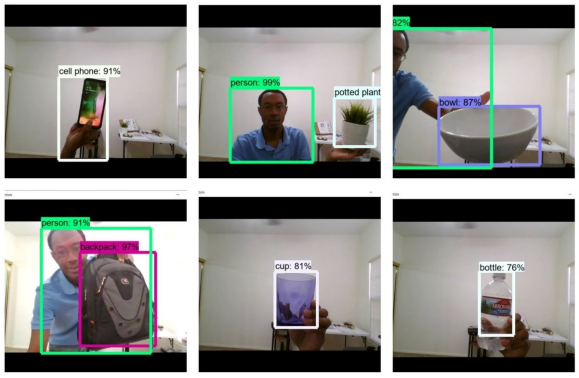
SLAM and Navigation



SLAM navigation

Use LiDAR with SLAM (Simultaneous Localization and Mapping) to create maps and navigate autonomously.

Object Detection



object detection

Use an onboard camera to detect and follow objects.

Optional Features

Potential additional features include speech and facial recognition.

2.3.2 Project Components Overview

Exterior

- **Wheels**: Differential drive with one large drive wheel on each side.
- **Sensors**: Camera and Kinect sensor.
- **Other**: Battery plug, switch, and screen for displaying information.

Interior

- **Motors and Motor Driver**: Control the wheels.
- **Single Board Computer**: The main control unit running ROS2.
- **ATmega2560 based controller**: Acts as an intermediary for motor control.

2.3.3 Creation of the URDF (Unified Robot Description Format) 3D Model

Add a new file named `robot.control.xacro` inside the `description` directory. Add the main closure tags to the XML file as given below:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
</robot>
```

Inside the `robot` tags, implement the URDF file for the robot.

Material Setup

```
<!-- Materials -->
<material name="white">
    <color rgba="1 1 1 1" />
</material>

<material name="blue">
    <color rgba="0.2 0.2 1 1" />
</material>

<material name="black">
    <color rgba="0 0 0 1" />
</material>

<material name="red">
    <color rgba="1 0 0 1" />
</material>

<material name="orange">
    <color rgba="1 0.3 0.1 1" />
</material>
```

Explanation: Define materials used throughout the XML file with specified color codes.

Macros for Inertials

```
<xacro:macro name="inertial_sphere" params="mass radius *origin">
    <!-- Inertial properties for a sphere -->
</xacro:macro>

<xacro:macro name="inertial_box" params="mass x y z *origin">
    <!-- Inertial properties for a box -->
</xacro:macro>

<xacro:macro name="inertial_cylinder" params="mass length radius *origin">
    <!-- Inertial properties for a cylinder -->
</xacro:macro>
```

Explanation: Macros to simplify defining inertial properties for different shapes.

Base Link and Footprint

```
<!-- Base Link -->
<link name="base_link"/>

<!-- Base Footprint Link -->
<joint name="base_footprint_joint" type="fixed">
    <!-- Joint connecting base_link to base_footprint -->
</joint>

<link name="base_footprint"/>
```

Explanation: Defines the base link and its footprint.

Chassis

```
<!-- Chassis Link -->
<joint name="chassis_joint" type="fixed">
<!-- Joint connecting base_link to chassis -->
</joint>

<link name="chassis">
<!-- Visual and collision elements defining the chassis -->
</link>

<gazebo reference="chassis">
<material>Gazebo/White</material>
</gazebo>
```

Explanation: Defines the chassis link and its properties.

Wheels

```
<!-- Left Wheel Link -->
<joint name="left_wheel_joint" type="continuous">
    <!-- Joint connecting base_link to left_wheel -->
</joint>

<link name="left_wheel">
    <!-- Visual and collision elements defining the left wheel -->
</link>

<gazebo reference="left_wheel">
    <material>Gazebo/Black</material>
</gazebo>

<!-- Right Wheel Link -->
<joint name="right_wheel_joint" type="continuous">
    <!-- Joint connecting base_link to right_wheel -->
</joint>

<link name="right_wheel">
    <!-- Visual and collision elements defining the right wheel -->
</link>

<gazebo reference="right_wheel">
    <material>Gazebo/Black</material>
</gazebo>
```

```
26
27 <!-- Caster Wheel Link -->
28 <joint name="caster_wheel_joint" type="fixed">
29     <!-- Joint connecting chassis to caster_wheel -->
30 </joint>
31
32 <link name="caster_wheel">
33     <!-- Visual and collision elements defining the caster wheel -->
34 </link>
35
36 <gazebo reference="caster_wheel">
37     <material>Gazebo/Black</material>
38     <mu1 value="0.001"/>
39     <mu2 value="0.001"/>
40 </gazebo>
```

Explanation: Defines the wheel links and their properties.

Importing Control Xacro File

To import the control XML file into the main URDF XML file, insert the following code:

```
1 <xacro:include filename="robot.control.xacro" />
```

2.3.4 Setting up Ros2 control for controlling the robot with teleop

For ROS2 control setup, create a new file named `robot.roscontrol.xml` in the `description` directory and add the following XML code:

```
1 <!-- Control Selection -->
2
3 <ros2_control name="GazeboSystem" type="system">
4     <!-- ROS2 control configuration -->
5 </ros2_control>
6
7 <gazebo>
8     <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
9         <parameters>$(find SLAM_ros2_bot)/config/my_parameters.yaml</
              parameters>
10     </plugin>
11 </gazebo>
```

Explanation: Configures ROS2 control for the robot and specifies Gazebo plugin settings.

Creating Parameters File

Create a new YAML file (e.g., `my_parameters.yaml`) inside the `config` directory with the following content:

```
1 controller_manager:
2   ros__parameters:
3     update_rate:  30
4     use_sim_time: true
5
6     diff_cont:
7       type: diff_drive_controller/DiffDriveController
8
9     joint_broad:
10      type: joint_state_broadcaster/JointStateBroadcaster
11
```

```
diff_cont:
  ros__parameters:
    publish_rate: 50.0
    base_frame_id: base_link
    left_wheel_names: ['left_wheel_joint']
    right_wheel_names: ['right_wheel_joint']
    wheel_separation: 0.35
    wheel_radius: 0.05
    use_stamped_vel: false
```

Explanation: Defines parameters for controller management and specific controller configurations.

Adding ROS Control Information to Launch File

Edit the `launch/rsp.launch.py` file to include ROS control information:

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.substitutions import LaunchConfiguration, Command
from launch.actions import DeclareLaunchArgument
from launch_ros.actions import Node
import xacro

def generate_launch_description():
    # Launch description generation
    return LaunchDescription([
        # Launch arguments
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use sim time if true'),
        DeclareLaunchArgument(
            'use_ros2_control',
            default_value='true',
            description='Use ros2_control if true'),

        # Robot state publisher node
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            output='screen',
            parameters=[{
                'robot_description': Command([
                    'xacro ',
                    os.path.join(get_package_share_directory('SLAM_ros2_bot
                        '),
                                 'description', 'robot.urdf.xacro'),
                    ' use_ros2_control:=',
                    LaunchConfiguration('use_ros2_control')])
            }]
        )
    ])
```

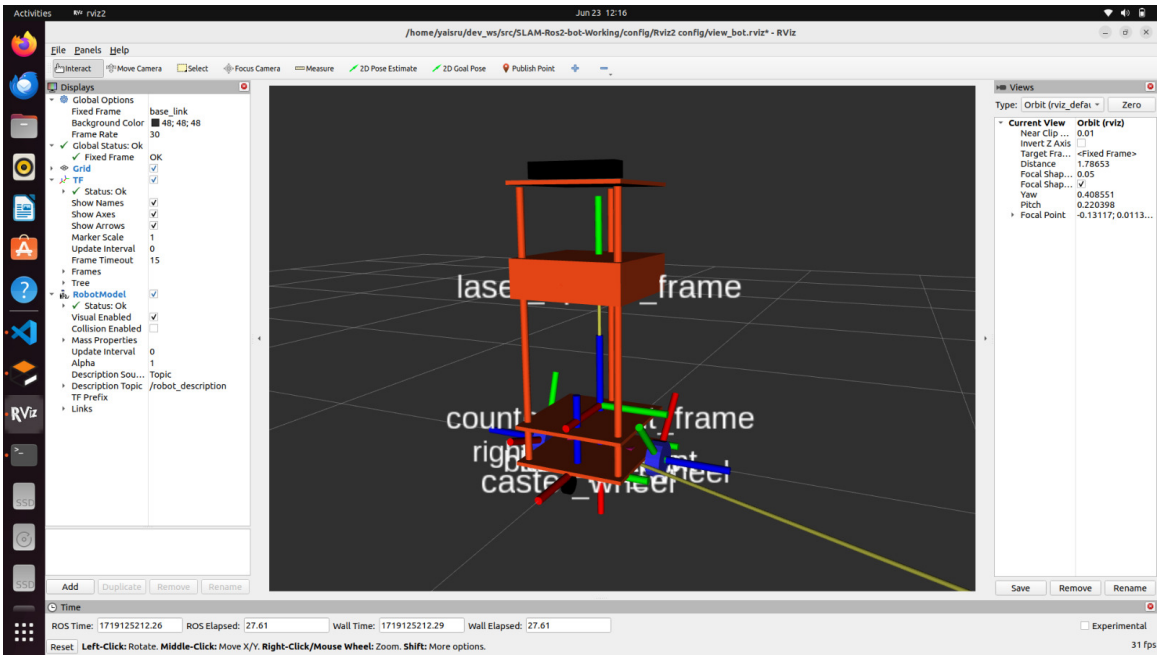
Explanation: Generates launch configuration for ROS nodes and includes robot state publisher.

Running the System

Follow these steps in a new terminal window:

```
cd dev_ws
source /opt/ros/iron/setup.bash
colcon build --symlink-install
source install/setup.bash
ros2 launch <Your Package Name> rsp.launch.py -use_sim_time:=true
```

Open RVIZ2 in another terminal window with the command `rviz2` to view the robot URDF model.



URDF model

Teleoperation Setup

For teleoperation setup, use a game controller connected via Bluetooth or wired connection. Run the following command in a new terminal:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Use the displayed keys to control the robot's wheels.

Setting up the game controller for teleoperation of the robot

1. Verify Gamepad Compatibility and Install Necessary Tools

- Ensure your gamepad is compatible with Linux and ROS.
- Install required tools:
  - `joystick`
  - `jstest-gtk`
  - `evtest`

2. Test Gamepad Connectivity and Configuration

- Connect your gamepad to the computer.
- Run `evtest` to check the event number assigned to your gamepad.
- Use `jstest-gtk` or `jstest` to verify axis and button functionality.

3. Configure ROS for Gamepad Input

- Set up the joystick configuration to publish to the `/joy` topic.
- Ensure the joystick configuration uses the newer Linux joystick drivers (`joy` package).

4. Create Launch and Parameter Files

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import DeclareLaunchArgument
import os
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():

    joy_params = os.path.join(get_package_share_directory('SLAM_ros2_bot'),'
        config','my_parameters.yaml')

    joy_node = Node(
            package='joy',
            executable='joy_node',
            parameters=[joy_params],
         )

    teleop_node = Node(
            package='teleop_twist_joy',
            executable='teleop_node',
            name='teleop_node',
            parameters=[joy_params],
            remappings=[('/cmd_vel','/diff_cont/cmd_vel_unstamped')]
         )

    return LaunchDescription([
        joy_node,
        teleop_node,
    ])
```

```
joy_node:
                ros__parameters:
                device_id: 0
                deadzone: 0.05
                autorepeat_rate: 20.0

        teleop_node:
  ros__parameters:
    axis_linear:
      x: 1
    scale_linear:
      x: 0.5
    scale_linear_turbo:
      x: 1.0

    axis_angular:
      yaw: 0
    scale_angular:
      yaw: 0.5
    scale_angular_turbo:
```

```
      yaw: 1.0


    enable_button: 4
    enable_turbo_button: 5

    require_enable_button: true
```

5. Test Joystick Input in ROS

- Launch ROS and verify joystick input:
  - Run `ros2 run joy joy_node` to publish joystick data to `/joy` topic.
  - Use `ros2 topic echo /joy` to monitor joystick input.

6. Implement Teleoperation with `teleop_twist_joy`

- Install `teleop_twist_joy` package if not already installed.

```
            apt install ros2-iron-teleop-twist-joy
```

- Create a launch file (`teleop.launch.py`) to launch `teleop_twist_joy` node.

7. Remap Twist Command Topic

- Remap the `/cmd_vel` topic in your launch file to `/diff_drive_controller/cmd_vel_unstamped`.

8. Test Teleoperation on Robot

- SSH into your robot.
- Launch controllers on the robot.
- Run the teleoperation launch file (`teleop.launch.py`).
- Test teleoperation using the gamepad:
  - Move the robot forward, backward, and turn to verify functionality.
  - Test regular and turbo mode speeds.

9. Set Up Feedback with RViz

- Launch RViz with the appropriate configuration file to visualize robot feedback.
- Configure RViz to display camera and LiDAR feeds from the robot.
- Ensure RViz provides real-time feedback during teleoperation.

10. Refinement and Additional Configurations

- Fine-tune joystick dead zones, scaling factors, and button mappings based on performance and user feedback.
- Implement any additional features such as safety controls or alternate control modes as needed.

By following these steps, you can effectively set up teleoperation for your robot using a gamepad, ensuring smooth control and reliable feedback during operation.

2.3.5 Setting up Gazebo for simulation and testing the system (optional step)

For the verification and tuning of the system, simulation using a tool like Gazebo integrated with ROS is beneficial. Here's how to set up Gazebo for your ROS 2 application:

Install Required Tools

Open a terminal and run the following commands to install necessary tools:

```
sudo apt install gazebo
sudo apt install ros2-iron-gazebo-ros2-control
```

These commands install Gazebo and ROS 2 control packages required for simulation.

Create Launch File

Create a new launch file named `launch/launch_sim.launch.py` and include the following code:

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
from launch.actions import TimerAction




def generate_launch_description():

    package_name='SLAM_ros2_bot'

    delayed_controller_manager_spawner = TimerAction(
        period=10.0,
        actions=[
            Node(
                package="controller_manager",
                executable="spawner",
                arguments=["diff_cont"],
            ),
            Node(
                package="controller_manager",
                executable="spawner",
                arguments=["joint_broad"],
            ),
        ],
    )

    rsp = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory(package_name),'launch','rsp.
                        launch.py'
                )]), launch_arguments={'use_sim_time': 'true', '
                    use_ros2_control': 'true'}.items()

    )

    gazebo_params_file = os.path.join(get_package_share_directory(
        package_name),'config','my_parameters.yaml')

```

```
    # Include the Gazebo launch file, provided by the gazebo_ros package
    gazebo = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory('gazebo_ros'), 'launch', '
                        gazebo.launch.py')]),
                    launch_arguments={'extra_gazebo_args': '--ros-args --
                        params-file ' + gazebo_params_file}.items()
             )

    # Run the spawner node from the gazebo_ros package. The entity name
        doesn't really matter if you only have a single robot.
    spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
                        arguments=['-topic', 'robot_description',
                                   '-entity', 'SLAM_ros2_bot'],
                        output='screen')

    # Launch them all!
    return LaunchDescription([
        rsp,
        gazebo,
        joystick,
        spawn_entity,
        delayed_controller_manager_spawner
    ])
```

Explanation

This Python script generates a launch description for a ROS 2 application involving a SLAM robot, ROS 2 control framework, and Gazebo for simulation. Let's break down the script:

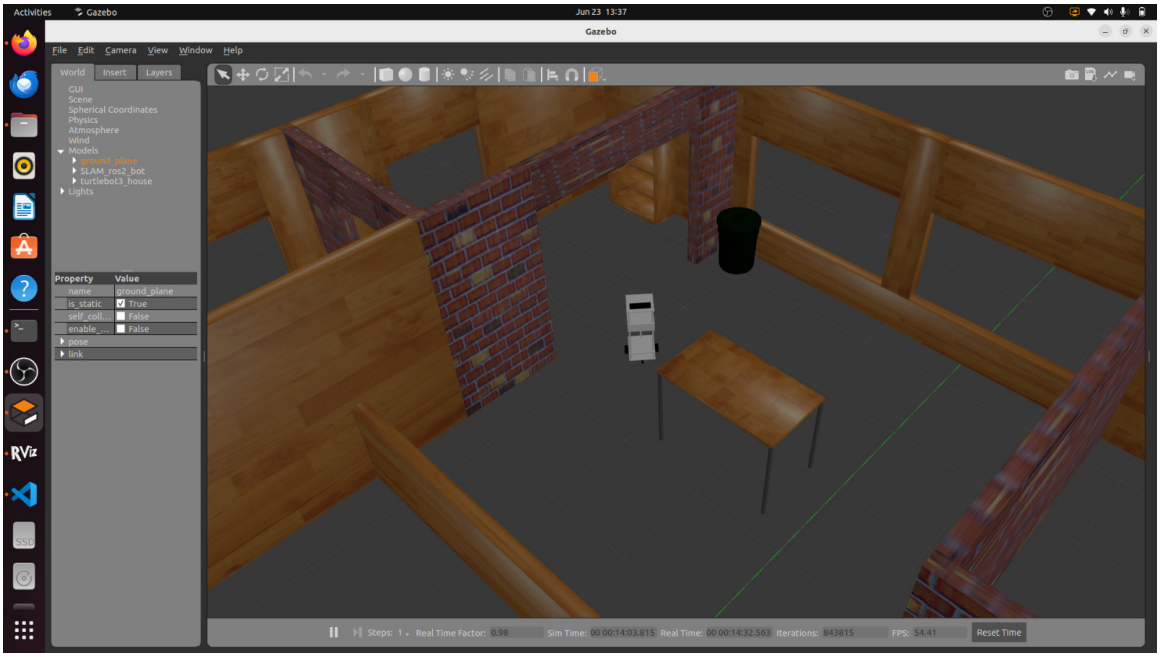
1. **Imports and Package Declaration**: Imports necessary modules and defines the package name.
2. **Delayed Controller Manager Spawner**: Uses a 'TimerAction' to delay controller spawner nodes from the 'controller_manager' package after a specified period.
3. **Robot State Publisher Inclusion**: Includes another launch file (`rsp.launch.py`) with specified arguments to enable simulation time and ROS 2 control.
4. **Gazebo Parameters File**: Specifies the path to the Gazebo parameters file (`my_parameters.yaml`).
5. **Gazebo Launch File Inclusion**: Includes the Gazebo launch file from the `gazebo_ros` package and passes additional arguments including the parameters file.
6. **Spawn Entity Node**: Uses `gazebo_ros` package to spawn the robot entity in Gazebo using `spawn_entity.py`.
7. **Launch Description Assembly**: Combines all actions into a `LaunchDescription` object.

Gazebo Parameters

Add the following parameters to your parameters file (`my_parameters.yaml`):

```
gazebo = IncludeLaunchDescription(
   PythonLaunchDescriptionSource([os.path.join(
       get_package_share_directory('gazebo_ros'), 'launch', 'gazebo.launch.
           py')]),
       launch_arguments={'extra_gazebo_args': '--ros-args --params-file ' +
           gazebo_params_file}.items()
)
```

```
ros2 launch <package_name> launch_sim.launch.py -use_sim_time:=true
    environment:=<location_to_world_file>
```



Gazebo simulation

This YAML file configures Gazebo parameters such as the publish rate.

Launch the Simulation

Launch the system using the following command in a new terminal:

```
        ros2 launch <package_name> launch_sim.launch.py -use_sim_time:=true
```

Replace `<package_name>` with your actual package name.

Load Custom Environment

To load a custom world environment file in Gazebo, use the following command:

```
ros2 launch <package_name> launch_sim.launch.py -use_sim_time:=true
    environment:=<location_to_world_file>
```

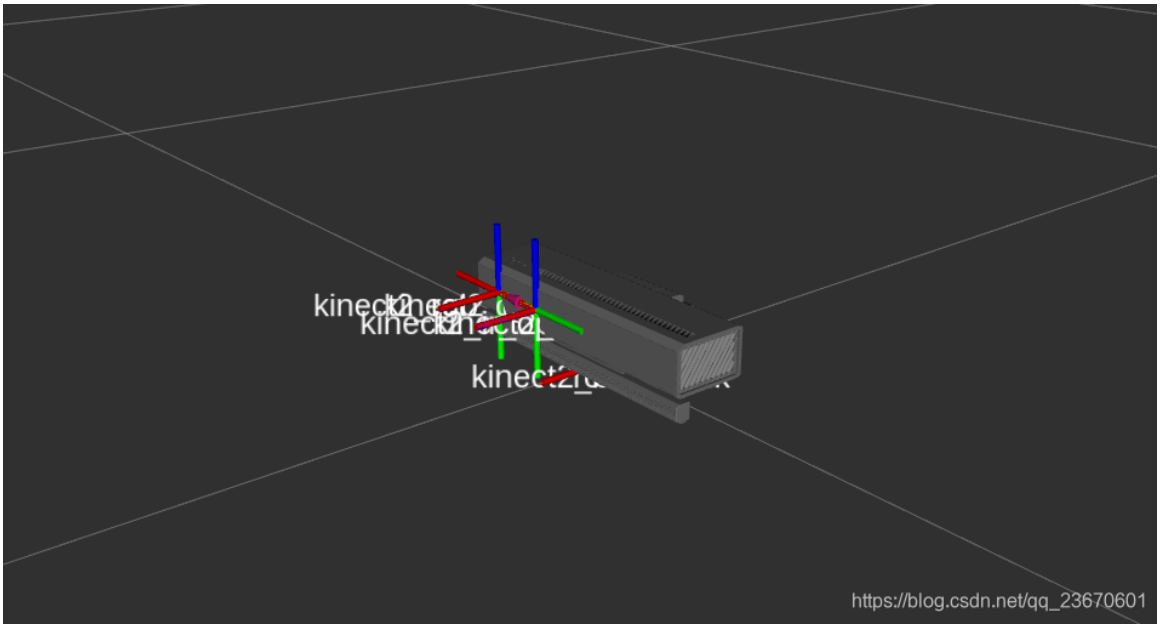
Replace `<location_to_world_file>` with the path to your world file.

If the simulation launches without errors and Gazebo displays your robot model in a 3D environment, you are ready to proceed with testing and tuning your system.

2.3.6 Setting up Kinect V2 sensor in the simulation envonment

To simulate the Kinect V2 sensor in Gazebo, follow these steps to create a URDF model and configure Gazebo plugins.

Create URDF Model



kinect URDF

Inside the `descriptions` directory, create a new file named `robot.kinect.xacro` and include the following XML code:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

<!-- //
    ////////////////////////////////////////////////////////////////////////////////////////////
     -->
    <!-- ////////////////////////Kinect V2
        ////////////////////////////////////////////////////// -->

    <!-- Kinect Link -->
    <link name="laser_frame">
        <visual>
        <geometry>
            <box size="0.05 0.23 0.04"/>
        </geometry>
        <origin xyz="0.13 0 0.26" rpy="0 0 0"/>
        <material name="black"/>
        </visual>
    </link>

    <!-- Kinect Joint -->
    <joint name="laser_joint" type="fixed">
        <parent link="base_link"/>
        <child link="laser_frame"/>
        <origin xyz="0 0 0.5" rpy="0 0 0"/>
    </joint>

    <!-- Kinect Gazebo Configuration -->
    <gazebo reference="laser_frame">
        <material>Gazebo/Black</material>
        <sensor type="depth" name="kinect_depth">
```

```
        <update_rate>30</update_rate>
        <pose>0 0 0 0 0 0</pose>
        <visualize>true</visualize>
        <depth>
            <clip>
            <near>0.1</near>
            <far>10.0</far>
            </clip>
            <horizontal_fov>1.047</horizontal_fov>
        </depth>
        <camera>
            <horizontal_fov>1.047</horizontal_fov>
            <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
            </image>
            <clip>
            <near>0.1</near>
            <far>10.0</far>
            </clip>
        </camera>
        <plugin name="kinect_camera_controller" filename="
            libgazebo_ros_camera.so">
            <alwaysOn>true</alwaysOn>
            <updateRate>0.0</updateRate>
            <cameraName>kinect_rgb</cameraName>
        </plugin>
        <plugin name="depth_camera_controller" filename="
            libgazebo_ros_depth_camera.so">
            <alwaysOn>true</alwaysOn>
            <updateRate>0.0</updateRate>
            <cameraName>kinect_depth</cameraName>
        </plugin>
        </sensor>
    </gazebo>

    <gazebo>
        <plugin name="gazebo_ros_openni_kinect" filename="
            libgazebo_ros_openni_kinect.so">
        <alwaysOn>true</alwaysOn>
        <updateRate>0.0</updateRate>
        <cameraName>kinect</cameraName>
        </plugin>
    </gazebo>

</robot>
```

Explanation

This XML snippet defines a URDF model for the Kinect V2 sensor in Gazebo with the following components:

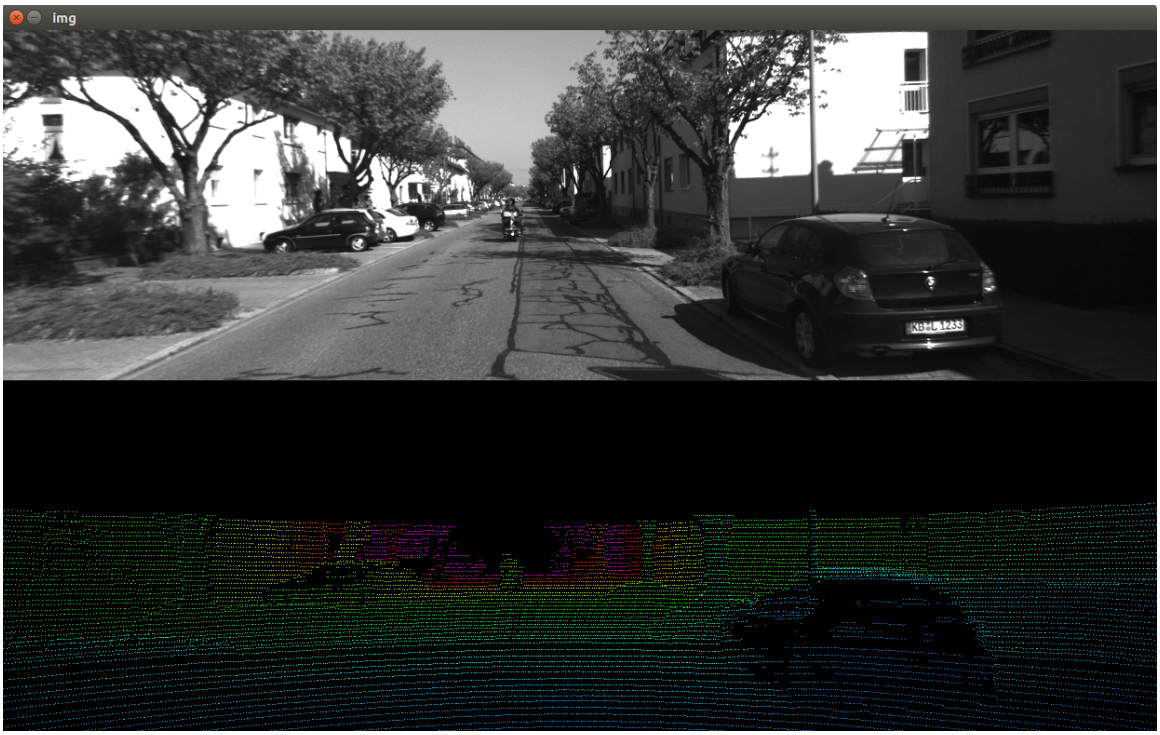
1. **Kinect Link**:
    - Defines a link named `laser_frame` for the Kinect sensor.
    - Specifies a visual representation as a box with dimensions and position.
    - Uses a black material.

2. **Kinect Joint**:
    - Creates a fixed joint named `laser_joint` to connect the Kinect frame to the robot's `base_link`.
    - Positions the joint at an offset.
3. **Kinect Gazebo Configuration**:
    - Configures Gazebo for the Kinect frame:
        - Sets visual material and adds a depth sensor plugin.
        - Defines depth sensor parameters such as update rate, clipping ranges, and field of view.
        - Configures an RGB camera with specific resolution and format.
    - Includes plugins to control the camera and depth camera in Gazebo.
4. **Additional Gazebo Plugin**:
    - Adds another Gazebo plugin (`gazebo_ros_openni_kinect`) to simulate the Kinect sensor.
    - Configures the plugin with options like update rate and camera name.

Usage

After creating the URDF model and configuring Gazebo, launch the simulation environment. Use RVIZ to subscribe to topics like `kinect/depth_image` to view data received from the Kinect sensor.

2.3.7 Conversion of the Kinect sensor depth data into fake-lidar data



depth image to lidar conversion

To convert Kinect depth image data into laser scan data compatible with SLAM toolbox and AMCL for mapping and localization in ROS 2, follow these steps:

Modify Launch File for Compatibility

Navigate to `/opt/ros/iron/share/depthimagetolaser/depthimage_to_laserscan-launch.py`, copy the file to your workspace's launch directory, and edit it as shown below:

```
1
2  # Copyright 2019 Open Source Robotics Foundation, Inc.
3  #
4  # Licensed under the Apache License, Version 2.0 (the "License");
5  # you may not use this file except in compliance with the License.
6  # You may obtain a copy of the License at
7  #
8  #     http://www.apache.org/licenses/LICENSE-2.0
9  #
10 # Unless required by applicable law or agreed to in writing, software
11 # distributed under the License is distributed on an "AS IS" BASIS,
12 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.
15
16 # /* Author: Gary Liu */
17
18 import os
19
20 from ament_index_python.packages import get_package_share_directory
21 from launch import LaunchDescription
22 from launch_ros.actions import Node
23
24
25 def generate_launch_description():
26     param_config = os.path.join(
27         get_package_share_directory('SLAM_ros2_bot'), 'cfg', 'param.yaml')
28     return LaunchDescription([
29         Node(
30             package='depthimage_to_laserscan',
31             executable='depthimage_to_laserscan_node',
32             name='depthimage_to_laserscan',
33             remappings=[('depth', '/kinect_depth/depth/image_raw'),
34                         ('depth_camera_info', '/kinect_depth/camera_info')
                            ],
35             parameters=[{'output_frame': 'laser_frame'}])
36     ])
```

Explanation

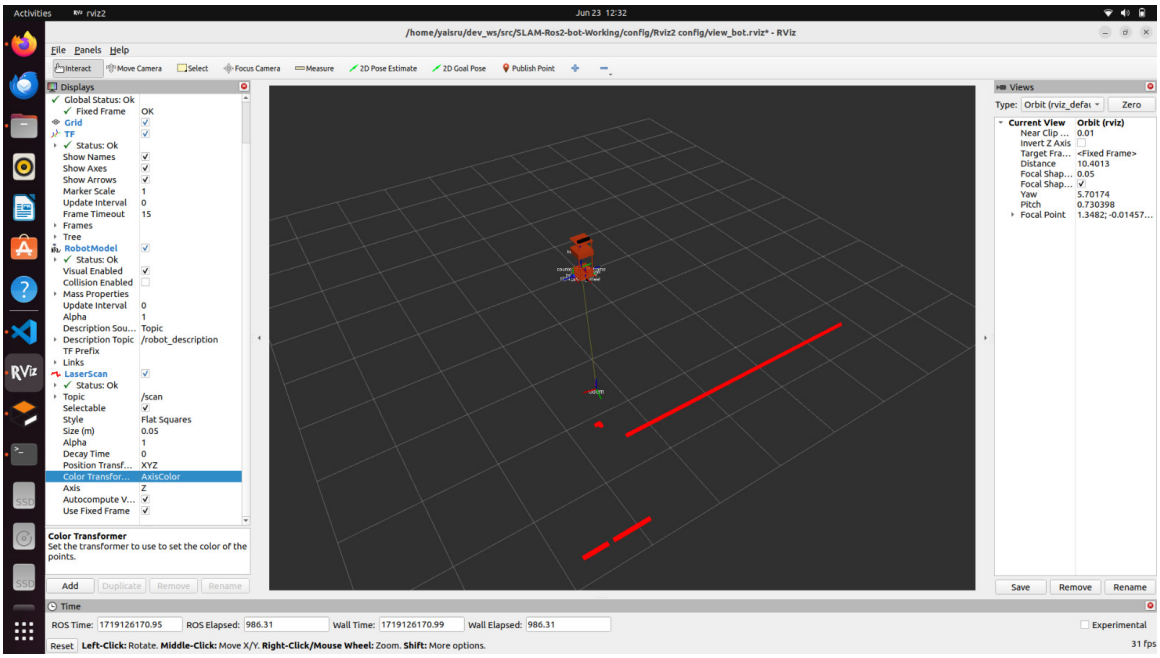
This modified launch file configures the depth image to laser scan conversion package (`depthimage_to_laserscan`) to publish laser scan data from Kinect depth images:

- **Parameters**:
  - Sets the output frame to `laser_frame`, which is compatible with SLAM toolbox and AMCL.
- **Node Configuration**:
  - Launches `depthimage_to_laserscan_node` from the package.
  - Remaps input topics `depth` and `depth_camera_info` to Kinect depth image topics.
  - Sets the output frame to `laser_frame`.

Launching the Conversion Package

Open a new terminal while the main launch file is running in another window and execute the following command:

```
ros2 launch <package_name> depthimage_to_laserscan-launch.py
```



Laserscan Rviz

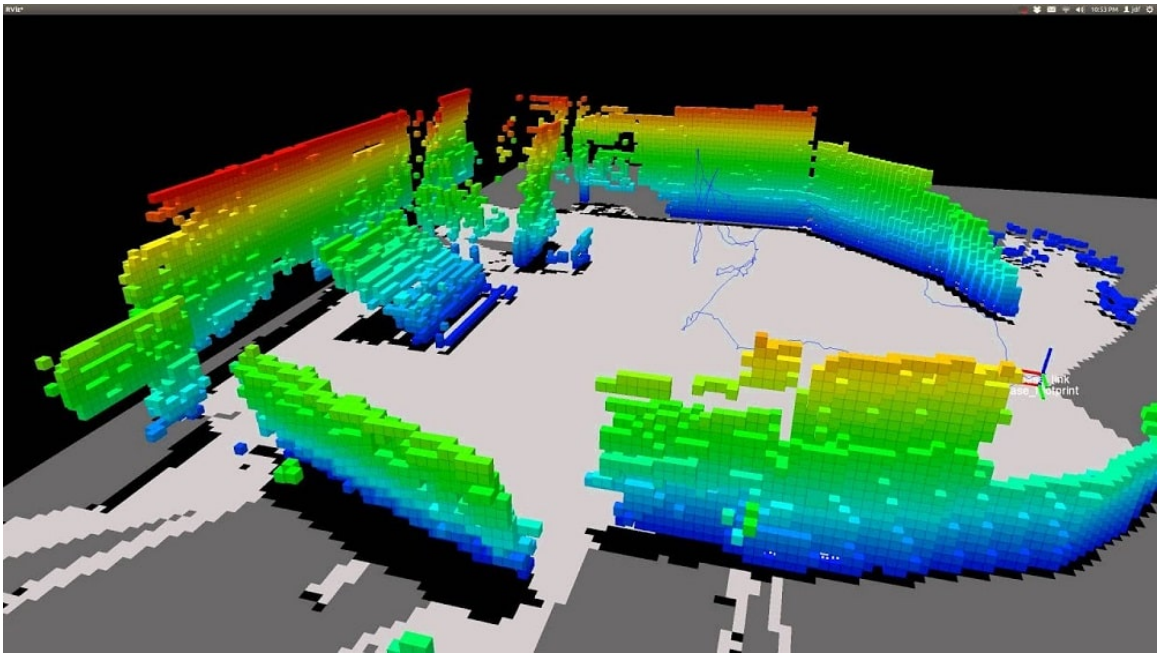
Visualization in RVIZ

Use RVIZ to visualize the laser scan data conversion:

1. Open RVIZ and add a LaserScan display.
2. Select the topic `/scan` to visualize the converted laser data.

By following these steps, you can effectively convert Kinect depth image data into laser scan data and visualize it using RVIZ, ensuring compatibility with SLAM and AMCL tools for mapping and localization in ROS 2.

2.3.8 Using SLAM tool box for mapping the envionment



SLAM mapping

Prerequisites

Ensure you have a pre-made environment in Gazebo or a real-world robot setup with Kinect sensor data publishing to ROS. Additionally, you need teleoperation support.

Installation

First, install the SLAM Toolbox package using the following command:

```
sudo apt install ros-iron-slam-toolbox
```

Setting up SLAM Toolbox

Copy the original launch file to your workspace and modify it as follows:

```

import os

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory


def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time')
    slam_params_file = LaunchConfiguration('slam_params_file')

    declare_use_sim_time_argument = DeclareLaunchArgument(
        'use_sim_time',
        default_value='true',
        description='Use simulation/Gazebo clock')
```

```
    declare_slam_params_file_cmd = DeclareLaunchArgument(
        'slam_params_file',
        default_value=os.path.join(get_package_share_directory("
            SLAM_ros2_bot"),
                                   'config', 'my_parameters.yaml'),
        description='Full path to the ROS2 parameters file to use for the
            slam_toolbox node')

    start_async_slam_toolbox_node = Node(
        parameters=[
          slam_params_file,
          {'use_sim_time': use_sim_time}
        ],
        package='slam_toolbox',
        executable='async_slam_toolbox_node',
        name='slam_toolbox',
        output='screen')

    ld = LaunchDescription()

    ld.add_action(declare_use_sim_time_argument)
    ld.add_action(declare_slam_params_file_cmd)
    ld.add_action(start_async_slam_toolbox_node)

    return ld
```

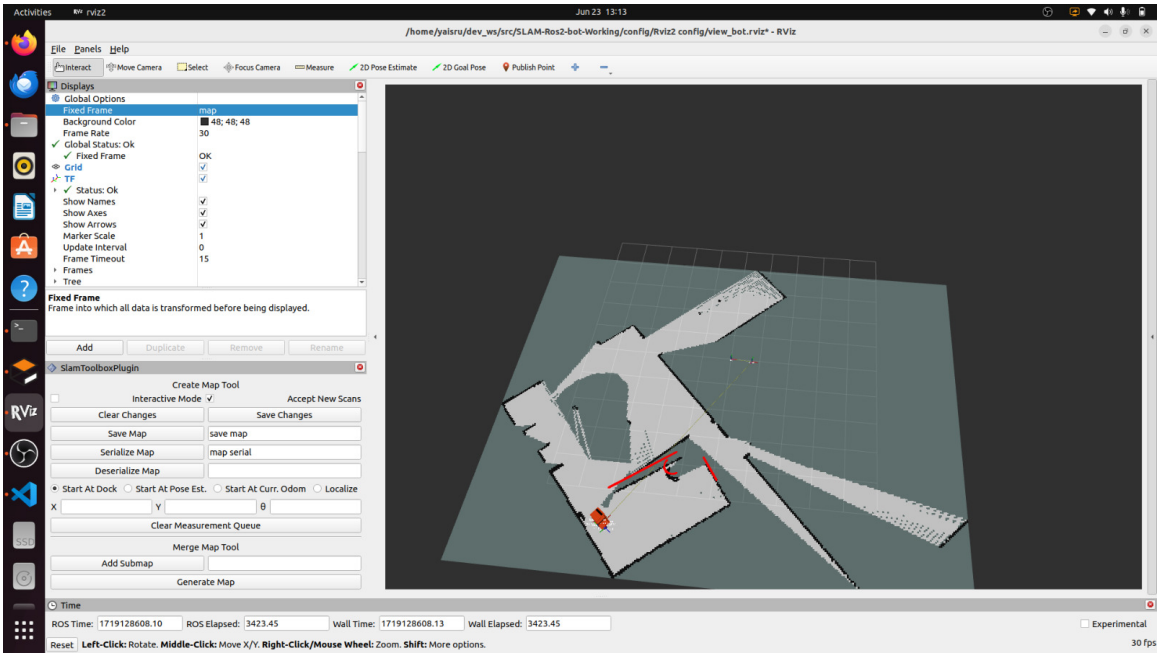
Parameters File for SLAM Toolbox

Copy the parameters provided into your parameters file for SLAM Toolbox (`my_parameters.yaml`).

Launching SLAM Toolbox

Launch SLAM Toolbox with Gazebo simulation running in the background:

```
ros2 launch <package_name> online_async_launch.py use_sim_time:=true
```



SLAM toolbox mapping

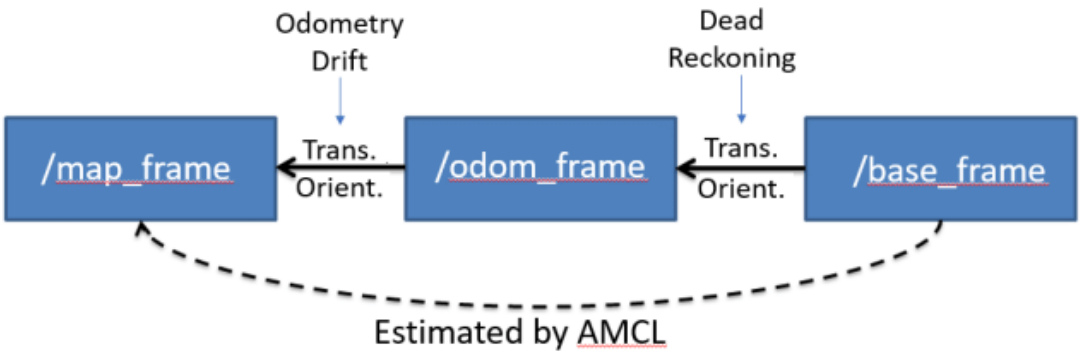
Visualizing in RViz

Open RViz and subscribe to the `/map` topic using the map tool. Set the base topic to `/map` to view the robot and map in the Gazebo window.

Mapping with Teleoperation

Use teleoperation commands (e.g., twist_keyboard or gamepad) to navigate the robot slowly within the environment. The map will generate as you move. Save the map using the SLAM Toolbox panel in RViz.

Setup for Automatic Localization



AMCL localization

Install Required Packages

Install the necessary packages for AMCL:

```
sudo apt install ros-iron-navigation2
sudo apt install ros-iron-bringup
sudo apt install ros-iron-turtlebot3
```

Host the Saved Map

Host the saved map as a topic using `map_server`. Run the following command:

```
        ros2 run nav2_map_server map_server --ros-args -p yaml_filename:=
            my_map_save.yaml -p use_sim_time:=true
```

Ensure to replace `my_map_save.yaml` with your actual map file path.

Run AMCL

Start the AMCL node to localize the robot using the hosted map:

```
ros2 run nav2_amcl
```

Visualize in RViz

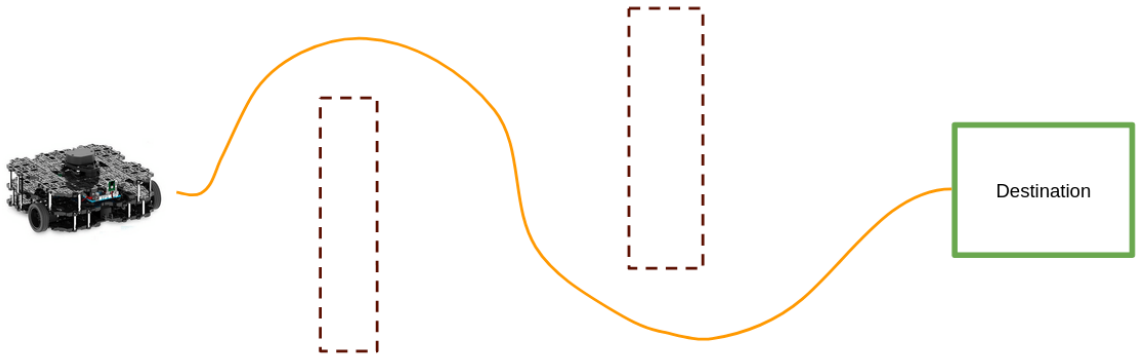
- Open RViz:

```
rviz2
```

- Load the map:
  - Set the ‘Fixed Frame‘ to `/map`.
  - Add a ‘Map‘ display and set the topic to `/map`.
  - Set the ‘Map Topic‘ type to ‘Transient Local‘ in the display settings.
- Localize the Robot:
  - While Gazebo is running, use the ‘2D Pose Estimate‘ button in RViz.
  - Click on the map in RViz to set the approximate location and heading of the robot.

  AMCL will localize the robot based on the map and your input.

2.3.9 Using nav2 for navigation



nav2 navigation

1. Install Required Packages

Ensure you have installed the necessary packages for Nav2:

```
    sudo apt install ros-iron-navigation2
    sudo apt install ros-iron-bringup
    sudo apt install ros-iron-turtlebot3
```

2. Launch the Nav2 Stack

Create a launch file named `nav2_bringup_launch.py` in your package's launch directory:

```
python
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node

def generate_launch_description():
   use_sim_time = LaunchConfiguration('use_sim_time', default='true')
   map_yaml_file = LaunchConfiguration('map', default=os.path.join(
       get_package_share_directory('my_package'), 'maps', 'my_map.yaml'))
   params_file = LaunchConfiguration('params', default=os.path.join(
       get_package_share_directory('my_package'), 'params', 'nav2_params.
       yaml'))

```

```
    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='true',
            description='Use simulation (Gazebo) clock if true'),

        DeclareLaunchArgument(
            'map',
            default_value=map_yaml_file,
            description='Full path to map file to load'),

        DeclareLaunchArgument(
            'params',
            default_value=params_file,
            description='Full path to the ROS2 parameters file to use'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([os.path.join(
                get_package_share_directory('nav2_bringup'),
                'launch',
                'bringup_launch.py')]),
            launch_arguments={
                'map': map_yaml_file,
                'use_sim_time': use_sim_time,
                'params_file': params_file
            }.items(),
        )
    ])
```

```
python
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node

def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time', default='true')
    map_yaml_file = LaunchConfiguration('map', default=os.path.join(
        get_package_share_directory('my_package'), 'maps', 'my_map.yaml'))
    params_file = LaunchConfiguration('params', default=os.path.join(
        get_package_share_directory('my_package'), 'params', 'nav2_params.
        yaml'))

    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='true',
            description='Use simulation (Gazebo) clock if true'),

        DeclareLaunchArgument(
            'map',
            default_value=map_yaml_file,
            description='Full path to map file to load'),

        DeclareLaunchArgument(
            'params',
            default_value=params_file,
```

```
            description='Full path to the ROS2 parameters file to use'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([os.path.join(
                get_package_share_directory('nav2_bringup'),
                'launch',
                'bringup_launch.py')]),
            launch_arguments={
                'map': map_yaml_file,
                'use_sim_time': use_sim_time,
                'params_file': params_file
            }.items(),
        )
    ])
```

3. Create Nav2 Parameters File

Create a `nav2_params.yaml` file in your package's `params` directory with appropriate parameters for each node:

```
amcl:
  ros__parameters:
    use_sim_time: true
    ...
lifecycle_manager:
  ros__parameters:
    use_sim_time: true
    autostart: true
    node_names:
      - "map_server"
      - "amcl"
      - "planner_server"
      - "controller_server"
      - "bt_navigator"
      - "waypoint_follower"
map_server:
  ros__parameters:
    use_sim_time: true
    yaml_filename: "path/to/your/map.yaml"
planner_server:
  ros__parameters:
    use_sim_time: true
    ...
controller_server:
  ros__parameters:
    use_sim_time: true
    ...
bt_navigator:
  ros__parameters:
    use_sim_time: true
    ...
```

4. Run the Navigation Launch File

Launch the Nav2 stack using the following command:

```
ros2 launch my_package nav2_bringup_launch.py
```

5. Setting Up RViz

Open RViz to visualize the robot and navigation:

```
rviz2
```

Configure RViz as follows: - Add a ‘Map‘ display, set the topic to `/map`. - Add a ‘Pose‘ display, set the topic to `/amcl_pose`. - Add a ‘Path‘ display, set the topic to `/plan`. - Add a ‘RobotModel‘ display to visualize the robot.

6. Sending Navigation Goals

You can send navigation goals programmatically. Here's an example Python script to send a goal:

```
#python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import PoseStamped
from action_msgs.msg import GoalStatus

class Nav2GoalSender(Node):
    def __init__(self):
        super().__init__('nav2_goal_sender')
        self.goal_pub = self.create_publisher(PoseStamped, 'goal_pose', 10)
        self.send_goal()

    def send_goal(self):
        goal_msg = PoseStamped()
        goal_msg.header.frame_id = 'map'
        goal_msg.header.stamp = self.get_clock().now().to_msg()
        goal_msg.pose.position.x = 2.0
        goal_msg.pose.position.y = 2.0
        goal_msg.pose.orientation.w = 1.0
        self.goal_pub.publish(goal_msg)
        self.get_logger().info('Goal sent!')

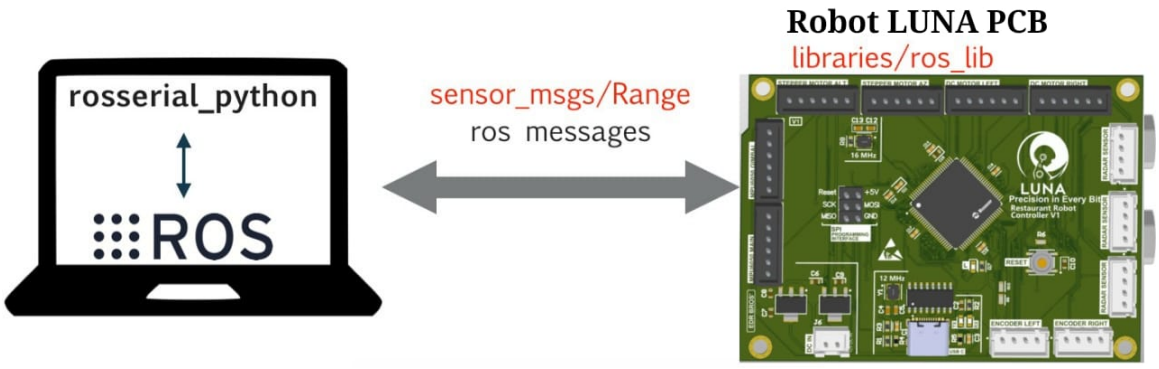
def main(args=None):
    rclpy.init(args=args)
    node = Nav2GoalSender()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Save this script and run it to send a goal to the robot.

By following these steps, you will have set up the Navigation2 stack for your robot, enabling it to navigate autonomously in the environment using the previously saved map and AMCL for localization.

2.4 Serial Communication with PCB



Ros serial

As we are using microcontroller ATmega2560 microcontroller for the motor controlling and encoder reading collection for our robot we have to have an connection pathway from that microcontroller to our SBC. So we used UART interface to directly publish and subscribe ROS topics from the microcontroller itself through the protocol. For adding that functionality we have to install ros serial library. It can be installed from the following command.

```
sudo apt-get install ros-${ROS_DISTRO}-rosserial-arduino
sudo apt-get install ros-${ROS_DISTRO}-rosserial
```

Navigate to the folder where arduino libraries are installed. If it is not installed you can follow this tutorial to install arduino IDE on your linux machine https://forum.arduino.cc/t/arduino-appimage-from-application-on-ubuntu-linux-and-add-to-favorite-menu/1146433.

After navigating in to libraries folder(may depend on arduino libarary installed directory. According to the tutorial it is in /.local/share/applications/) execute the following command.

```
rosrun rosserial_arduino make_libraries.py
```

After that check whether you can import ros.h library. You can use your arduino ide to program the atmega microcontrollers to subscribe and publish to ROS topics and to create ROS nodes.

2.5 Wide angle Camera data transferring

The wide-angle camera plays a pivotal role in enhancing the functionality and safety of Robot LUNA. While the Kinect v2 3D depth camera is the primary sensor for navigation and obstacle detection, the Night Version 150 Degree Wide Angle 5M Pixel 1080P camera provides additional security confirmation and captures detailed customer interactions. This chapter delves into the integration, setup, and utilization of the wide-angle camera within the ROS (Robot Operating System) environment, highlighting the code and commands necessary for efficient data transfer and processing.

2.5.1 Integration of the Wide Angle Camera

Integrating the wide-angle camera into Robot LUNA's system involves connecting it to the computer unit and configuring the necessary software to ensure seamless data transfer. The camera interacts with users and helps in identifying their commands, thereby enhancing the overall user experience. The camera's wide field of view is particularly useful for capturing a comprehensive image of the restaurant environment, which aids in security and obstacle detection.

2.5.2 Setting Up the Camera in ROS 1

To set up the wide-angle camera in ROS 1, follow these steps:

1. **Connect the Camera:** First, connect the Night Version 150 Degree Wide Angle 5M Pixel 1080P camera to one of the available USB ports on the SBC (Single Board Computer).

2. **Install Necessary Packages:** Ensure that the necessary ROS packages for USB cameras are installed. You can install the 'usb$_cam$'$package, which supports a wide range of USB cameras.$

```
sudo apt-get update
sudo apt-get install ros-noetic-usb-cam
```

3. **Launch File Configuration:** Create a ROS launch file to configure and start the camera node. This launch file will specify parameters such as the camera device path, frame rate, and resolution.

Create a new launch file named wide$_angle_camera.launch$:

```
```xml
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen">
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="mjpeg" />
    <param name="io_method" value="mmap" />
    <param name="frame_rate" value="30" />
  </node>
</launch>
```
```

4. **Launching the Camera Node:** Use the following terminal command to launch the camera node and start capturing data:

```
```bash
roslaunch your_package_name wide_angle_camera.launch
```
```

Replace " your$_package_name$" $with the actual name of your ROS package.$

2.5.3 Data Transfer and Processing

The wide-angle camera's data needs to be transferred efficiently within the ROS framework for real-time processing. Below is an example of a ROS node written in Python to subscribe to the camera's image topic and process the data.

Python Node for Image Processing:

Create a Python script named 'image$_processor.py$' :

```
```python
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import cv2

class ImageProcessor:
    def __init__(self):
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/usb_cam/image_raw", Image,
            self.callback)

    def callback(self, data):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            print(e)
```
```

```
19
20          # Process the image (e.g., display it)
21          cv2.imshow("Wide Angle Camera", cv_image)
22          cv2.waitKey(3)
23
24  def main():
25      rospy.init_node('image_processor', anonymous=True)
26      ic = ImageProcessor()
27      try:
28          rospy.spin()
29      except KeyboardInterrupt:
30          print("Shutting down")
31      cv2.destroyAllWindows()
32
33  if __name__ == '__main__':
34      main()
```

Running the Image Processor:
Use the following terminal command to run the image processor node:

```
1  ```bash
2  rosrun your_package_name image_processor.py
```

2.5.4 Utilization of Wide Angle Camera Data

The data captured by the wide-angle camera is utilized in several ways. Firstly, it enhances the security of Robot LUNA by providing an additional layer of obstacle detection, ensuring that the robot can confirm the presence of obstacles detected by the Kinect v2 depth camera. Secondly, it captures detailed images of customers, which can be used to identify commands and interactions more accurately. This dual-camera setup ensures that Robot LUNA operates safely and efficiently in a dynamic restaurant environment.

By leveraging the wide-angle camera alongside the Kinect v2 depth camera, Robot LUNA can navigate complex environments with higher accuracy and reliability. The integration of these advanced sensors makes LUNA a robust and efficient addition to any restaurant, capable of delivering exceptional service while maintaining high safety standards. For more details on the primary Kinect v2 3D depth camera, refer to Chapter 2.6.

2.6 Setting up Kinect 2 depth camera for 3D point clouds

2.6.1 Installing Drivers for Kinectv2

Kinect is a line of motion sensing input devices produced by Microsoft. The devices generally contain RGB cameras, and infrared projectors and detectors that map depth through either structured light or time of flight calculations, which can in turn be used to perform real-time gesture recognition and body skeletal detection,
We used ROS specially for communicate the data captured through the device and to process them. We have to installed the drivers and take readings, We have used the following commands.

Before using the kinect2 on our machine we need to setup the drivers to use that device on our computer. For that there are two distributions named openNi and libfreenect. Sadly openni was taken by apple inc. so the only available option is setting up libfreenect2 drivers. Open the terminal and run the following.

```
1 sudo apt-get install build-essential cmake pkg-config  -y  # Build tools
2 sudo apt-get install libusb-1.0-0-dev          # libusb
3 sudo apt-get install libturbojpeg0-dev -y      # TurboJPEG
4 sudo apt-get install libglfw3-dev -y           # OpenGL
5 sudo apt-get install libva-dev libjpeg-dev -y  # VAAPI for Intel only
```

Figure 3: Xbox Kinect 2

```
sudo apt-get install libopenni2-dev -y          # OpenNI2

git clone https://github.com/OpenKinect/libfreenect2.git
cd libfreenect2

mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
make
make install

sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/
```

We can run the test program to check whether the drivers are correctly installed by running

```
./bin/Protonect
```

Output of the Protonect

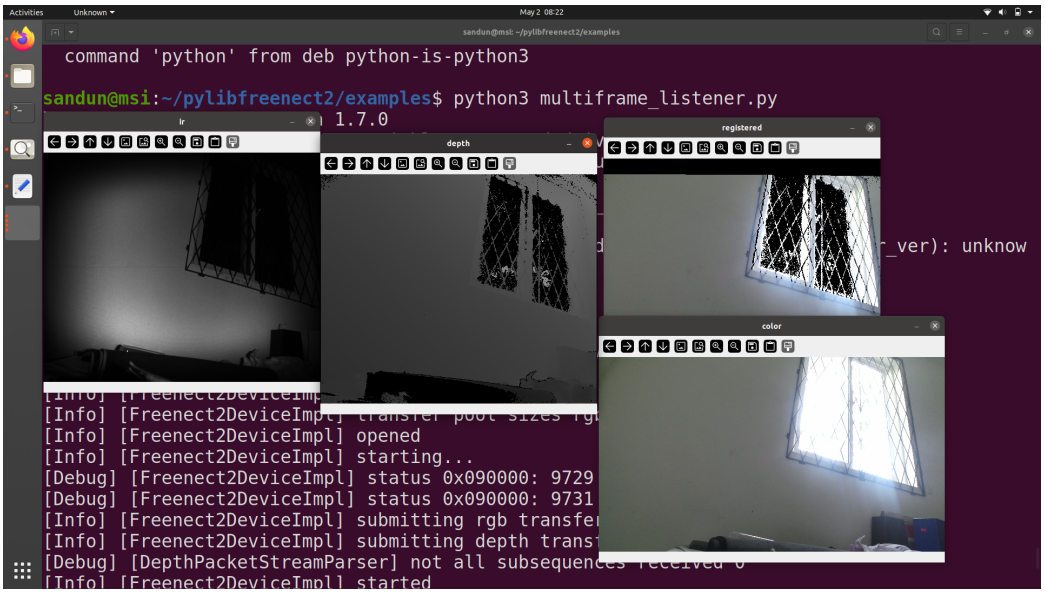
2.6.2 Installing Pylibfreenect2

Although it is optional to install pylibfreenect2 on our computer we installed it mainly for our testing purposes. It is a python library which helps to deal with kinect2 depth data through.First we have to install few dependancies as,

```
pip install wheel
pip install numpy
pip install cython
pip install opencv-python
pip install pylibfreencect2
```

but pip install pylibfreenect2 gives us an error.So we cloned the github repo https://github.com/r9y9/pylibfreenect2.git https://github.com/r9y9/pylibfreenect2.git and setup path of libfreenect2 and installed. Another popup error is solved by following the steps mentioned in Appendix 4.8.

```
cd ~
git clone https://github.com/r9y9/pylibfreenect2.git
cd pylibfreenect2
python3 setup.py
For verify installation run
cd ~/pylibfreenect2/examples
python3 multiframe_listner.py
```



Form pylibfreenect

2.6.3 Receiving Depth Images from Kinect for further processing

As we setup pylibfreenect2 for easy interaction with kinectv2 data we got captured depth data from the camera for further processing by running the following python script which is added in the Appendix 4.11 after plugging in kinect.

2.6.4 Mapping the environment with Kinect

2.6.5 Using Kinect's Fake Lidar Scanner

2.7 Motor controller Unit

We have written the following C++ code to control the motors of the robot. The microcontroller takes linear velocity and rotational velocity as inputs from ROS and computes the speed for each motor. We then use a PID controller to control the motor speeds. Additionally, we have integrated 3 radar sensors to detect dynamic obstacles and prevent emergency accidents. The actual motor speeds are calculated using magnetic encoders, and the speeds along with the time are sent to ROS. Here is the C++ code for motor control:

Libraries and Initial Setup

The following libraries are included:

```
#include <Adafruit_MotorShield.h>
#include <Wire.h>
#include <PID_v1.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <geometry_msgs/Twist.h>
#include <ros/time.h>
```

Constants and Variables

```
// Define loop time in milliseconds
#define LOOPTIME 100

// Maximum loops without communication
const byte noCommLoopMax = 10;
unsigned int noCommLoops = 0;

// Left motor speed command
double speed_cmd_left2 = 0;

// Pin numbers for motor encoders
const int PIN_ENCOD_A_MOTOR_LEFT = 2, PIN_ENCOD_B_MOTOR_LEFT = 4,
          PIN_ENCOD_A_MOTOR_RIGHT = 3, PIN_ENCOD_B_MOTOR_RIGHT = 5;

// Pin number for side light LED
const int PIN_SIDE_LIGHT_LED = 46;

// Last millisecond timestamp
unsigned long lastMilli = 0;

// Physical parameters of the robot
const double radius = 0.04, wheelbase = 0.187, encoder_cpr = 990,
             speed_to_pwm_ratio = 0.00235, min_speed_cmd = 0.0882;

// Speed and angular speed requirements
double speed_req = 0, angular_speed_req = 0;

// Left motor speed requirements, actual speed, and command
double speed_req_left = 0, speed_act_left = 0, speed_cmd_left = 0;

// Right motor speed requirements, actual speed, and command
double speed_req_right = 0, speed_act_right = 0, speed_cmd_right = 0;

// Maximum speed of the robot
const double max_speed = 0.4;

// PID controller parameters for left and right motors
const double PID_left_param[] = { 0, 0, 0.1 }, PID_right_param[] = { 0, 0, 0.1 };

// PID controllers for left and right motors
volatile float pos_left = 0, pos_right = 0;
PID PID_leftMotor(&speed_act_left, &speed_cmd_left, &speed_req_left,
                  PID_left_param[0], PID_left_param[1], PID_left_param[2], DIRECT);
PID PID_rightMotor(&speed_act_right, &speed_cmd_right, &speed_req_right,
                   PID_right_param[0], PID_right_param[1], PID_right_param[2], DIRECT)
                       ;

// Adafruit Motor Shield and motors
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *leftMotor = AFMS.getMotor(1), *rightMotor = AFMS.getMotor(2);

// ROS node handle
```

```
ros::NodeHandle nh;
```

ROS Callback Function

The function `handle_cmd` handles incoming velocity commands from ROS.

```
// Handle incoming velocity commands
void handle_cmd(const geometry_msgs::Twist& cmd_vel) {
    // Reset communication loop count
    noCommLoops = 0;

    // Extract linear and angular speed commands
    speed_req = cmd_vel.linear.x; // Linear speed
    angular_speed_req = cmd_vel.angular.z; // Angular speed

    // Calculate individual speed commands for left and right motors
    speed_req_left = speed_req - angular_speed_req * (wheelbase / 2); // Adjusted
        left speed
    speed_req_right = speed_req + angular_speed_req * (wheelbase / 2); // Adjusted
        right speed
}
```

ROS Subscriber and Publisher

The following lines set up the ROS subscriber and publisher.

```
// Subscribe to "cmd_vel" topic, handle incoming Twist messages with "handle_cmd"
ros::Subscriber<geometry_msgs::Twist> cmd_vel("cmd_vel", handle_cmd);

// Create a Vector3Stamped message for publishing speed information
geometry_msgs::Vector3Stamped speed_msg;

// Publisher for "speed" topic, publishing speed information
ros::Publisher speed_pub("speed", &speed_msg);
```

Setup Function

The `setup` function initializes the motor shield, ROS node, and encoders.

```
// Setup function to initialize hardware and ROS communication
void setup() {
    // Set side light LED pin as output and turn it on
    pinMode(PIN_SIDE_LIGHT_LED, OUTPUT);
    analogWrite(PIN_SIDE_LIGHT_LED, 255);

    // Initialize ROS node
    nh.initNode();
    nh.getHardware()->setBaud(57600);
    nh.subscribe(cmd_vel); // Subscribe to "cmd_vel" topic
    nh.advertise(speed_pub); // Advertise "speed" topic

    // Initialize Adafruit Motor Shield
    AFMS.begin();

    // Stop and brake both motors
    leftMotor->setSpeed(0);
    leftMotor->run(BRAKE);
    rightMotor->setSpeed(0);
    rightMotor->run(BRAKE);

    // Set PID controller sample time and output limits
    PID_leftMotor.SetSampleTime(95);
    PID_rightMotor.SetSampleTime(95);
    PID_leftMotor.SetOutputLimits(-max_speed, max_speed);
    PID_rightMotor.SetOutputLimits(-max_speed, max_speed);
    PID_leftMotor.SetMode(AUTOMATIC);
    PID_rightMotor.SetMode(AUTOMATIC);

    // Setup left motor encoder pins and attach interrupt
    pinMode(PIN_ENCOD_A_MOTOR_LEFT, INPUT);
    pinMode(PIN_ENCOD_B_MOTOR_LEFT, INPUT);
    digitalWrite(PIN_ENCOD_A_MOTOR_LEFT, HIGH);
    digitalWrite(PIN_ENCOD_B_MOTOR_LEFT, HIGH);
    attachInterrupt(0, encoderLeftMotor, RISING);

    // Setup right motor encoder pins and attach interrupt
    pinMode(PIN_ENCOD_A_MOTOR_RIGHT, INPUT);
    pinMode(PIN_ENCOD_B_MOTOR_RIGHT, INPUT);
    digitalWrite(PIN_ENCOD_A_MOTOR_RIGHT, HIGH);
    digitalWrite(PIN_ENCOD_B_MOTOR_RIGHT, HIGH);
    attachInterrupt(1, encoderRightMotor, RISING);
}
```

Main Loop

The `loop` function handles the main control loop.

```
// Main loop function
void loop() {
```

```
    // Handle ROS communication
    nh.spinOnce();

    // Check if it's time to run the control loop
    if ((millis() - lastMilli) >= LOOPTIME) {
        // Measure distances from sensors
        long distance1 = measureDistance(TRIGGER_PIN_1, ECHO_PIN_1);
        long distance2 = measureDistance(TRIGGER_PIN_2, ECHO_PIN_2);
        long distance3 = measureDistance(TRIGGER_PIN_3, ECHO_PIN_3);

        // Find minimum distance among sensors
        long minDistance = min(distance1, min(distance2, distance3));

        // Check if any sensor detects an object within threshold distance
        if (minDistance < 30) {
            // Reset last millisecond count
            lastMilli = millis();

            // Adjust side light LED intensity based on ROS connection status
            if (!nh.connected()) {
                analogWrite(PIN_SIDE_LIGHT_LED, lightValueNoComm[lightInc]);
                lightInc = (lightInc + 1) % 25;
            } else {
                analogWrite(PIN_SIDE_LIGHT_LED, lightValue[lightInc]);
                lightT = 3000 - ((2625 / max_speed) * ((abs(speed_req_left) + abs(
                    speed_req_right)) / 2));
                lightInc = (lightInc + (30 / (lightT / LOOPTIME))) % lightIncNumber;
            }

            // Calculate actual speeds from encoder counts
            speed_act_left = ((pos_left / encoder_cpr) * 2 * PI) * (1000 / LOOPTIME)
                * radius;
            speed_act_right = ((pos_right / encoder_cpr) * 2 * PI) * (1000 / LOOPTIME
                ) * radius;
            pos_left = 0; // Reset left encoder count
            pos_right = 0; // Reset right encoder count

            // Constrain and compute left motor speed control
            speed_cmd_left = constrain(speed_cmd_left, -max_speed, max_speed);
            PID_leftMotor.Compute();
            PWM_leftMotor = constrain(((speed_req_left + sgn(speed_req_left) *
                min_speed_cmd) / speed_to_pwm_ratio) + (speed_cmd_left /
                speed_to_pwm_ratio), -255, 255);

            // Set left motor speed and direction based on control output
            if (noCommLoops >= noCommLoopMax || speed_req_left == 0) {
                leftMotor->setSpeed(0);
                leftMotor->run(BRAKE);
            } else if (PWM_leftMotor > 0) {
                leftMotor->setSpeed(abs(PWM_leftMotor));
                leftMotor->run(BACKWARD);
            } else {
                leftMotor->setSpeed(abs(PWM_leftMotor));
                leftMotor->run(FORWARD);
            }

            // Constrain and compute right motor speed control
            speed_cmd_right = constrain(speed_cmd_right, -max_speed, max_speed);
            PID_rightMotor.Compute();
            PWM_rightMotor = constrain(((speed_req_right + sgn(speed_req_right) *
                min_speed_cmd) / speed_to_pwm_ratio) + (speed_cmd_right /
                speed_to_pwm_ratio), -255, 255);

            // Set right motor speed and direction based on control output
            if (noCommLoops >= noCommLoopMax || speed_req_right == 0) {
                rightMotor->setSpeed(0);
                rightMotor->run(BRAKE);
            } else if (PWM_rightMotor > 0) {
```

```
                rightMotor->setSpeed(abs(PWM_rightMotor));
                rightMotor->run(FORWARD);
            } else {
                rightMotor->setSpeed(abs(PWM_rightMotor));
                rightMotor->run(BACKWARD);
            }

            // Check for timeout
            if ((millis() - lastMilli) >= LOOPTIME) {
                Serial.println(" TOO LONG ");
            }

            // Increment communication loop count
            noCommLoops++;
            if (noCommLoops == 65535) {
                noCommLoops = noCommLoopMax;
            }

            // Publish speed information
            publishSpeed(LOOPTIME);
        }
    }
}
```

Obstacle Distance measuring Function

This function finds the distances to obstacles.

```
// Function to measure distance using an ultrasonic sensor
long measureDistance(int triggerPin, int echoPin) {
    // Send a short pulse to trigger the sensor
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);

    // Measure the duration of the pulse from the echo pin
    long duration = pulseIn(echoPin, HIGH);

    // Calculate the distance based on the duration of the pulse
    long distance = duration * 0.034 / 2;

    return distance; // Return the measured distance
}
```

Publish Speed Function

This function publishes the odometry information to the ROS topic.

```
// Function to publish speed information
void publishSpeed(double time) {
    // Update timestamp in the message header
    speed_msg.header.stamp = nh.now();

    // Set left and right motor speeds in the message
    speed_msg.vector.x = speed_act_left;
    speed_msg.vector.y = speed_act_right;

    // Set the time duration in the message
    speed_msg.vector.z = time / 1000;

    // Publish the speed message
    speed_pub.publish(&speed_msg);

    // Handle ROS communication
```

```
    nh.spinOnce();

    // Log information about publishing odometry
    nh.loginfo("Publishing odometry");
}
```

Encoder Interrupt Service Routines

These functions handle the encoder interrupts for the left and right motors.

```
// Update encoder count for the left motor
void encoderLeftMotor() {
    // Read the encoder pins
    int MSB1 = digitalRead(encoder1APin); // Most significant bit
    int LSB1 = digitalRead(encoder1BPin); // Least significant bit

    // Combine the two pin values into a single number
    int encoded1 = (MSB1 << 1) | LSB1;

    // Combine the current and previous readings for processing
    int sum1 = (lastEncoded1 << 2) | encoded1;

    // Increment or decrement position based on the sum value
    if (sum1 == 0b1101 || sum1 == 0b0100 || sum1 == 0b0010 || sum1 == 0b1011)
        pos_left++;
    if (sum1 == 0b1110 || sum1 == 0b0111 || sum1 == 0b0001 || sum1 == 0b1000)
        pos_left--;

    // Store the current reading for the next iteration
    lastEncoded1 = encoded1;
}

// Update encoder count for the right motor
void encoderRightMotor() {
    // Read the encoder pins
    int MSB2 = digitalRead(encoder2APin); // Most significant bit
    int LSB2 = digitalRead(encoder2BPin); // Least significant bit

    // Combine the two pin values into a single number
    int encoded2 = (MSB2 << 1) | LSB2;

    // Combine the current and previous readings for processing
    int sum2 = (lastEncoded2 << 2) | encoded2;

    // Increment or decrement position based on the sum value
    if (sum2 == 0b1101 || sum2 == 0b0100 || sum2 == 0b0010 || sum2 == 0b1011)
        pos_right++;
    if (sum2 == 0b1110 || sum2 == 0b0111 || sum2 == 0b0001 || sum2 == 0b1000)
        pos_right--;

    // Store the current reading for the next iteration
    lastEncoded2 = encoded2;
}
```

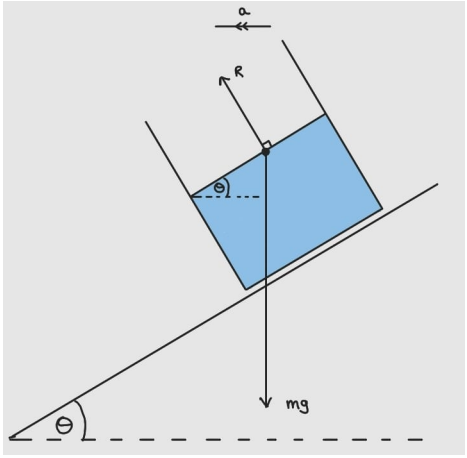
Sign Function Template

This template function returns the sign of a value.

```
    // Function to return the sign of a value
    template <typename T>
    int sgn(T val) {
        // Return 1 if positive, -1 if negative, 0 if zero
        return (T(0) < val) - (val < T(0));
    }
```

2.8 Stabilization Tray Controler Unit

We have developed a Self-Stabilizing Tray to keep the food and beverages no spilling. Let's consider the situation in which the water is not moving concerning the glass. Here is the Physics behind this: Let's Take a situation in which the water of a cup of water does not move with respect to the cup.



By Using the Newton's Second Law:

$$\overline{F} = m \cdot a$$

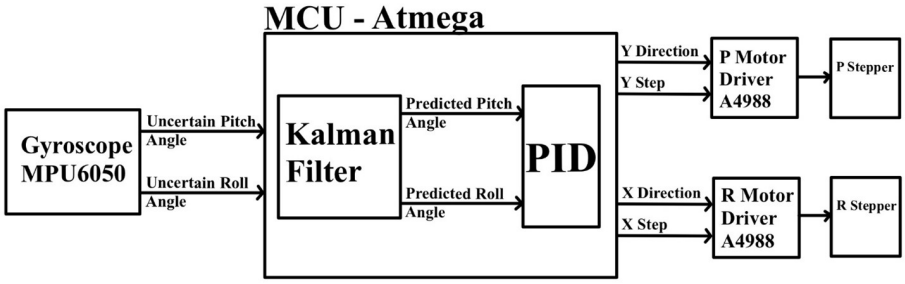
By substituting F from the Taking the projections:

$$mg \tan\theta = m \cdot a$$

Find the value of θ:

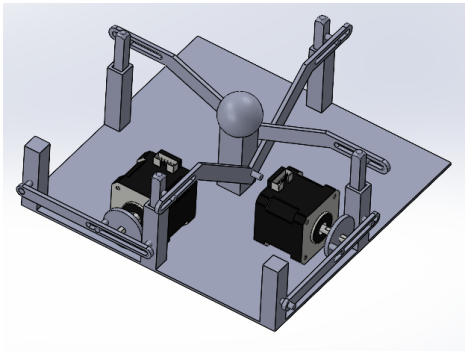
$$\theta = \arctan\left(\frac{a}{g}\right)$$

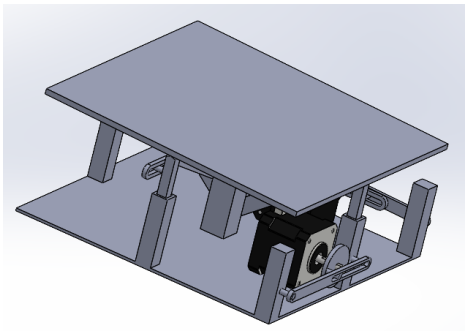
Uncertain roll and pitch angles are got from gyroscope as inputs to the microcontroller. But it is not the actual value. Because there will be some noise and uncertainty in the reading. There fore we are using Kalman's filter to predict the pitch and roll angles with less error. Here, we are using 2-D Kalman's filter. This filter gets roll and pitch angles, and accelerations on X, Y, Z directions, then it calculates predicted roll and pitch angles.
In order to stabilize the array, two stepper motors are used to adjust the roll and pitch angles of the tray. A4998 motor driver is used to drive the steppers. Motor driver needs inputs such as direction to which rotate, steps that needs to be rotated. Both are calculated using PID controller for each angles. There will be some time consuming trigonometric calculations should be done by the microcontroller.



Tray Stabilization Block Diagram

Here is the working machanism of our Stabilizing Tray:





Here is the C++ code for the Stabilization Tray Controler Unit: The following libraries are included:

```
#include <Wire.h>
```

Constants and Variables

```
// Define stepper motor pins for motor 1
#define STEP_PIN1 2
#define DIR_PIN1 3

// Define stepper motor pins for motor 2
#define STEP_PIN2 2
#define DIR_PIN2 3

// Delay between steps in milliseconds
#define STEP_DELAY 100

// Initialize MPU6050 sensor
MPU6050 mpu2;

// Variables for sensor data and calibration
float RateRoll, RatePitch, RateYaw;
float RateCalibrationRoll, RateCalibrationPitch, RateCalibrationYaw;
int RateCalibrationNumber;

// Variables for accelerometer data
float AccX1, AccY1, AccZ1, AccX2, AccY2, AccZ2;
double g = 9.8;

// Variables for angle calculation
float AngleRoll, AnglePitch;

// Timer variable
uint32_t LoopTimer;

// Kalman filter variables for roll and pitch angles
float KalmanAngleRoll = 0, KalmanUncertaintyAngleRoll = 2 * 2;
float KalmanAnglePitch = 0, KalmanUncertaintyAnglePitch = 2 * 2;
float Kalman1DOutput[] = { 0, 0 };

// PID controller parameters and variables for motor 1
double kp1 = 0.00001;
double ki1 = 0.03;
double kd1 = 10;
double Output1 = 0;
double Lasterror1 = 0;
double Integral1 = 0;
double error1 = 0;
```

```

// PID controller parameters and variables for motor 2
double kp2 = 0.00001;
double ki2 = 0.03;
double kd2 = 10;
double Output2 = 0;
double Lasterror2 = 0;
double Integral2 = 0;
double error2 = 0;

// Variables for storing calculated angles
double theta1 = 0;
double theta2 = 0;

// Counter variable
int k = 0;

// Time variable
int nowTime = 0;
```

The Kalman Filter

We use this Kalman Filter for get the angle of the tray very accurately and stable.

```
// Function to perform Kalman filtering for 1D systems
void kalman_1d(float KalmanState, float KalmanUncertainty, float KalmanInput, float
    KalmanMeasurement) {
    // Prediction step: Update Kalman state and uncertainty based on input
    KalmanState = KalmanState + 0.004 * KalmanInput;
    KalmanUncertainty = KalmanUncertainty + 0.004 * 0.004 * 4 * 4;

    // Update step: Calculate Kalman gain and update state and uncertainty based on
        measurement
    float KalmanGain = KalmanUncertainty * 1 / (1 * KalmanUncertainty + 3 * 3);
    KalmanState = KalmanState + KalmanGain * (KalmanMeasurement - KalmanState);
    KalmanUncertainty = (1 - KalmanGain) * KalmanUncertainty;

    // Store Kalman state and uncertainty in output array
    Kalman1DOutput[0] = KalmanState;
    Kalman1DOutput[1] = KalmanUncertainty;
}
```

The MPU6050 sensor 1

This sensor is located in the tray. This sensor gets the angles and the acceleration of the tray. It feeds the accelerometer and the gyroscope values to the Kalman filter.

```
// Function to read data from the gyro sensor and calculate roll and pitch angles
void gyro_signals(void) {
    // Configure gyro sensor registers for accelerometer and gyroscope sensitivity
        settings
    Wire.beginTransmission(0x68);
    Wire.write(0x1A);
    Wire.write(0x05);
    Wire.endTransmission();

    Wire.beginTransmission(0x68);
    Wire.write(0x1C);
    Wire.write(0x10);
    Wire.endTransmission();

    // Read accelerometer data
    Wire.beginTransmission(0x68);
    Wire.write(0x3B);
    Wire.endTransmission();
    Wire.requestFrom(0x68, 6);
```

```
    int16_t AccXLSB = Wire.read() << 8 | Wire.read();
    int16_t AccYLSB = Wire.read() << 8 | Wire.read();
    int16_t AccZLSB = Wire.read() << 8 | Wire.read();

    // Configure gyro sensor register for gyroscope sensitivity setting
    Wire.beginTransmission(0x68);
    Wire.write(0x1B);
    Wire.write(0x8);
    Wire.endTransmission();

    // Read gyroscope data
    Wire.beginTransmission(0x68);
    Wire.write(0x43);
    Wire.endTransmission();
    Wire.requestFrom(0x68, 6);
    int16_t GyroX = Wire.read() << 8 | Wire.read();
    int16_t GyroY = Wire.read() << 8 | Wire.read();
    int16_t GyroZ = Wire.read() << 8 | Wire.read();

    // Calculate roll and pitch rates from gyroscope data
    RateRoll = (float)GyroX / 65.5;
    RatePitch = (float)GyroY / 65.5;
    RateYaw = (float)GyroZ / 65.5;

    // Calculate accelerometer angles
    AccX1 = (float)AccXLSB / 4096 - 0.05;
    AccY1 = (float)AccYLSB / 4096 + 0.01 + 0.01;
    AccZ1 = (float)AccZLSB / 4096 - 0.11 - 0.11;

    // Calculate roll and pitch angles from accelerometer data
    AngleRoll = atan(AccY1 / sqrt(AccX1 * AccX1 + AccZ1 * AccZ1)) * 1 / (3.142 / 180)
        ;
    AnglePitch = -atan(AccX1 / sqrt(AccY1 * AccY1 + AccZ1 * AccZ1)) * 1 / (3.142 /
        180);
}
```

The MPU6050 sensor 2

This sensor is located in the Robot which allows us to know the robot's acceleration.

```
// Function to read accelerometer data from a second sensor
void acc_signals(void) {
    // Begin transmission to the second accelerometer sensor
    Wire.beginTransmission(0x69);

    // Select the register containing accelerometer data
    Wire.write(0x3B);

    // End transmission to the sensor
    Wire.endTransmission();

    // Request accelerometer data from the sensor
    Wire.requestFrom(0x69, 6);

    // Read accelerometer data for each axis
    int16_t AccXLSB = Wire.read() << 8 | Wire.read();
    int16_t AccYLSB = Wire.read() << 8 | Wire.read();
    int16_t AccZLSB = Wire.read() << 8 | Wire.read();

    // Convert raw accelerometer data to g-force and adjust for calibration offsets
    AccX2 = (float)AccXLSB / 4096 - 0.05;
    AccY2 = (float)AccYLSB / 4096 + 0.01 + 0.01;
    AccZ2 = (float)AccZLSB / 4096 - 0.11 - 0.11;
}
```

Stepper Motor control

This code allows to control of the stepper motors with the PID controller.

```
// Function to control motor 1
void motor1() {
    // Check if the absolute error is greater than 0.5
    if (absV(error1) > 0.5) {
        // Set motor direction based on output
        if (Output1 < 0) {
            digitalWrite(DIR_PIN1, HIGH);  // Set direction
        } else {
            digitalWrite(DIR_PIN1, LOW);  // Set direction
        }

        // Calculate step delay based on output and move motor
        delayMicroseconds(absV(1 / Output1));
        digitalWrite(STEP_PIN1, HIGH);
        delayMicroseconds(STEP_DELAY);
        digitalWrite(STEP_PIN1, LOW);
        delayMicroseconds(STEP_DELAY);
    }
}

// Function to control motor 2
void motor2() {
    // Check if the absolute error is greater than 0.5
    if (absV(error2) > 0.5) {
        // Set motor direction based on output
        if (Output2 < 0) {
            digitalWrite(DIR_PIN2, HIGH);  // Set direction
        } else {
            digitalWrite(DIR_PIN2, LOW);  // Set direction
        }

        // Calculate step delay based on output and move motor
        delayMicroseconds(absV(1 / Output2));
        digitalWrite(STEP_PIN2, HIGH);
        delayMicroseconds(STEP_DELAY);
        digitalWrite(STEP_PIN2, LOW);
        delayMicroseconds(STEP_DELAY);
    }
}
```

Absolute value fucntion

```
// Function to return the absolute value of a double
double absV(double value) {
    // Check if the value is negative
    if (value < 0) {
        return -value; // Return the negation of the value
    } else {
        return value; // Return the value as is
    }
}
```

PID controller function

This PID controller Functions helps to rotate the stepper motors precisely.

```
// Function to calculate PID control for motor 1
void PID1(int Setpoint) {
    // Calculate error between setpoint and current angle
    error1 = Setpoint - KalmanAngleRoll;

    // Accumulate error for integral term
```

```
    Integral1 += error1;

    // Calculate change in error for derivative term
    double dError = error1 - Lasterror1;

    // Calculate PID output
    Output1 = kp1 * error1 + ki1 * Integral1 + kd1 * dError;

    // Update last error for next iteration
    Lasterror1 = error1;
}

// Function to calculate PID control for motor 2
void PID2(int Setpoint) {
    // Calculate error between setpoint and current angle
    error2 = Setpoint - KalmanAngleRoll;

    // Accumulate error for integral term
    Integral2 += error2;

    // Calculate change in error for derivative term
    double dError = error2 - Lasterror2;

    // Calculate PID output
    Output2 = kp2 * error2 + ki2 * Integral2 + kd2 * dError;

    // Update last error for next iteration
    Lasterror2 = error2;
}
```

The Setup Function

```
void setup() {
    // Set pin modes for motor control pins
    pinMode(STEP_PIN1, OUTPUT);
    pinMode(DIR_PIN1, OUTPUT);
    pinMode(STEP_PIN2, OUTPUT);
    pinMode(DIR_PIN2, OUTPUT);

    // Initialize serial communication
    Serial.begin(57600);

    // Set built-in LED pin as output and turn it on
    pinMode(13, OUTPUT);
    digitalWrite(13, HIGH);

    // Configure I2C clock speed and begin communication
    Wire.setClock(400000);
    Wire.begin();
    delay(250);

    // Initialize MPU6050 sensor
    Wire.beginTransmission(0x68);
    Wire.write(0x6B);
    Wire.write(0x00);
    Wire.endTransmission();

    // Calibrate gyro sensor
    for (RateCalibrationNumber = 0; RateCalibrationNumber < 2000;
        RateCalibrationNumber++) {
        gyro_signals();
        RateCalibrationRoll += RateRoll;
        RateCalibrationPitch += RatePitch;
        RateCalibrationYaw += RateYaw;
        delay(1);
    }
```

```
    RateCalibrationRoll /= 2000;
    RateCalibrationPitch /= 2000;
    RateCalibrationYaw /= 2000;
    LoopTimer = micros();

    // Begin communication with MPU6050 2
    Wire.begin();

    // Initialize MPU6050 2
    mpu2.initialize();
    Serial.println("Initializing MPU6050 2...");

    // Calibrate gyro and accelerometer for MPU6050 2
    mpu2.calibrateGyro();
    mpu2.calibrateAccel();
}
```

The loop Function

```
  // Read gyro and accelerometer signals
gyro_signals();
acc_signals();

// Subtract calibration values from gyro rates
RateRoll -= RateCalibrationRoll;
RatePitch -= RateCalibrationPitch;
RateYaw -= RateCalibrationYaw;

// Perform Kalman filtering for roll angle
kalman_1d(KalmanAngleRoll, KalmanUncertaintyAngleRoll, RateRoll, AngleRoll);
KalmanAngleRoll = Kalman1DOutput[0];
KalmanUncertaintyAngleRoll = Kalman1DOutput[1];

// Perform Kalman filtering for pitch angle
kalman_1d(KalmanAnglePitch, KalmanUncertaintyAnglePitch, RatePitch, AnglePitch);
KalmanAnglePitch = Kalman1DOutput[0];
KalmanUncertaintyAnglePitch = Kalman1DOutput[1];

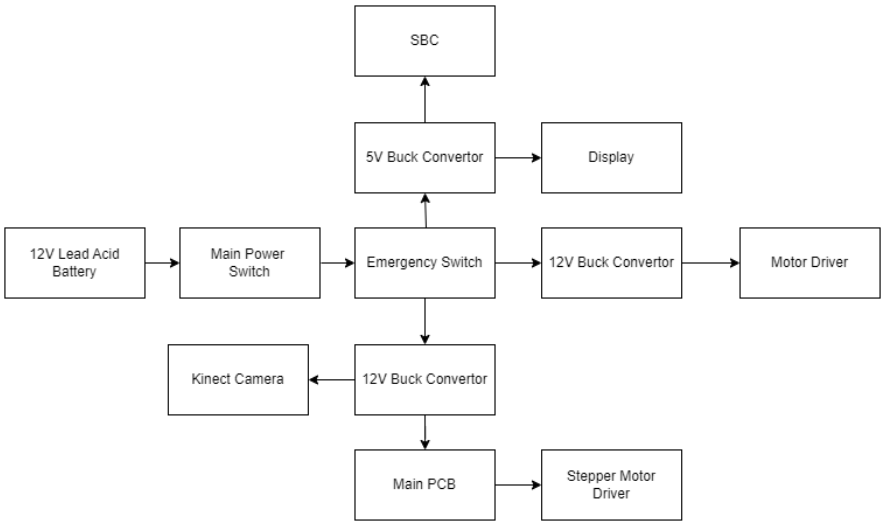
// Calculate pitch and roll angles from accelerometer data
theta1 = atan(AccX2 / g);
theta2 = atan(AccY2 / g);

// Perform PID control for pitch and roll
PID1(theta1);
PID2(theta2);

// Control motors based on PID outputs
motor1();
motor2();

// Wait until the loop time is reached
while (micros() - LoopTimer < 4000)
    ;
LoopTimer = micros(); // Update loop timer
```

2.9 Power Unit



- **12V Lead Acid Battery:** The primary power source for the entire system.
- **Main Power Switch:** Controls the connection between the battery and the rest of the system. It allows the user to turn the entire system on or off.
- **Emergency Switch:** Provides an additional layer of safety, allowing for a quick shutdown of the system in case of an emergency.
- **Buck Converters:**
  - **5V Buck Converter:** Converts the 12V from the battery down to 5V, which is needed to power the SBC (Single Board Computer) and the display.
  - **12V Buck Converters:** There are multiple 12V buck converters in the system: One converter directly powers the motor driver. Another converter powers the main PCB and the Kinect camera.
- **SBC (Single Board Computer):** A central processing unit that receives 5V power from the buck converter and likely controls or coordinates various functions of the system.
- **Display:** Powered by the 5V buck converter, it provides visual output for the user.
- **Motor Driver:** Receives power from one of the 12V buck converters and controls the motors based on signals from the main PCB or SBC.
- **Main PCB:** The main printed circuit board that interfaces with various components, receiving power from the 12V buck converter.
- **Kinect Camera:** A sensor device that requires 12V power, also supplied by the 12V buck converter connected to the main PCB.
- **Stepper Motor Driver:** Controlled by the main PCB, it receives signals to drive stepper motors, which are likely part of the mechanical system being controlled.

3 Conclusion

In conclusion, the development of Robot LUNA represents a significant milestone in the realm of restaurant automation. Through careful planning and execution, the project has demonstrated the feasibility and effectiveness of employing robotics in food service environments.

The division of the project into six distinct subsections proved instrumental in navigating the complexities inherent in such a multifaceted undertaking. Each subsection played a crucial role in shaping LUNA's design and functionality, from its sensory perception capabilities to its motor control systems and beyond.

Moving forward, Robot LUNA stands as a testament to the potential of robotics to enhance efficiency and customer experience in the restaurant industry. With its reliable performance and innovative features, LUNA promises to revolutionize the way restaurants operate, ushering in a new era of automation and convenience for both staff and patrons alike.

References

[1] "Robot Operating System (ROS)," ROS Wiki. [Online]. Available: https://wiki.ros.org/.[Accessed: Feb. 20, 2024]

[2] University of Houston testing new robot waiter https://www.youtube.com/watch?v=nyw65$_Y$yQuM

[3] *Local restaurant using robot waiter(2023).*. YouTube. Retrieved from https://www.youtube.com/watch?v=tJaj3_54S_E&t=21s.

[4] Run Kinect 2 on Ubuntu 20.04 LTS . https://www.notaboutmy.life/posts/run-kinect-2-on-ubuntu-20-lts/.

[5] *PUDU Robotics website.* . https://www.emerald.com/insight/content/doi/10.1108/JTF-04-2020-0070/full/html.

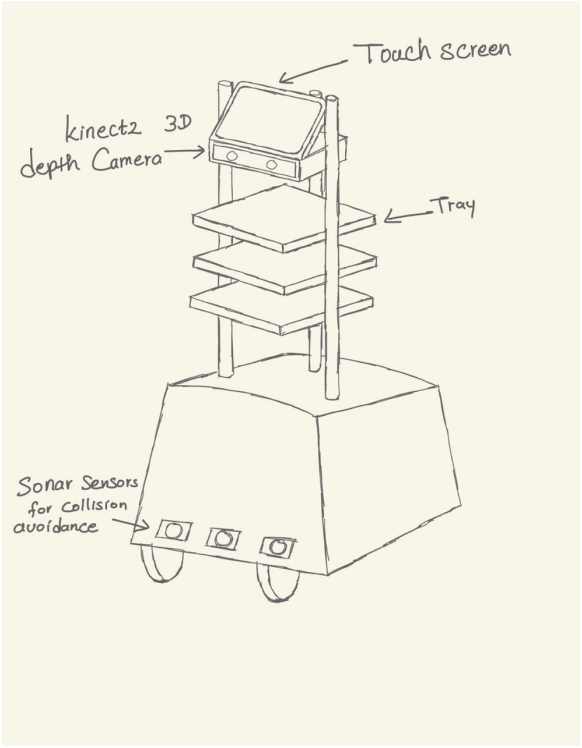
4 Appendix: Daily Log Entries- Hardware

4.1 Finding the Chasis for the Robot(16 -23 February 2024)

We initially planned to fabricate the robot chassis from scratch. Our design included two 45W motors to actuate the main components, namely the left and right wheels of the robot. To support the chassis, we included two caster wheels with a diameter of 5cm, attached at both the front and back ends. The chassis was designed to have a rounded appearance to maximize access to the food compartments from various angles.

The preliminary sketch featured two compartments. One of these compartments included a mechanical design with two stepper motors to stabilize beverages against the forces exerted by the robot's acceleration. We had to develop a control system to ensure that the liquids within this compartment remained perpendicular to the resultant force acting on them.

The top portion of the chassis was designated for the user interface (UI) design, which included an animated face of the robot created by another team. Below the UI section, another compartment was allocated for the Kinect v2 sensor, utilized for capturing the environment as a 3D map. The top tray was designed for food delivery, following a standard tray design. The robot's "brain" was intended to be housed inside the base, accommodating all motor drivers, single-board computer (SBC), and the controller printed circuit board (PCB) along with their respective connections.



Sketch of the robot design

However, during our exploration, we came across a robot structure within the department that closely resembled the one we were aiming to build. After careful consideration, we decided to proceed with this existing robot structure, as it aligned well with our project requirements.

Image of the found robot structure

4.2 Overview of Existing Solutions (23 - 26 February 2024)

While robot waiters are a relatively new technology, there are a growing number of companies developing solutions in this scope. Existing restaurant waiter robots tend to focus on similar functionalities as LUNA, including navigation, food and beverage delivery, and interaction with customers. Here, we will explore some of the common design elements found in existing restaurant waiter robots.

- **Navigation** :Many existing robots use a combination of cameras, LiDAR sensors, and other technologies to navigate around restaurants. This allows them to avoid obstacles and safely deliver food to customers.
- **Delivery Mechanism** :Existing solutions utilize different mechanisms for delivering food and beverages. Some robots use trays, while others employ shelves or grippers. The choice of mechanism depends on factors such as the type of food being served and the overall design of the robot.
- **Interaction with humans** :The level of interaction between robots and customers varies depending on the specific solution. Some robots may simply deliver food and leave, while others may be equipped with touchscreens or voice recognition capabilities to allow for more interaction.

We researched the strengths and weaknesses of existing solutions when planning about the restaurant robot project because, by understanding what is already available on the market, we can identify areas for improvement and develop a robot that is well-suited to the needs of the restaurant industry.

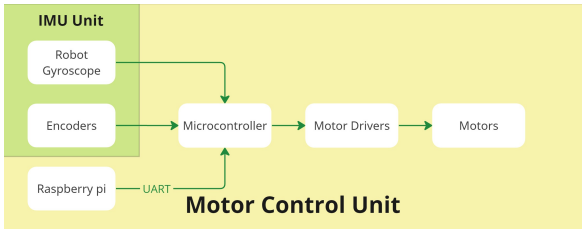
4.3 Navigation Planning

During the first week, our focus was on planning the navigation system for the LUNA robot. We considered:

- Set objectives for the navigation system, including smooth, efficient movement through the restaurant.
- Identified the necessary sensors (sonar sensor) and hardware required for navigation.
- Developed a preliminary roadmap that included potential challenges and proposed solutions.

Required hardware:

- Atmega2560
- Gyrosensor
- Motor driver x 2
- Motor x 2



Motor Control Unit Block Diagram

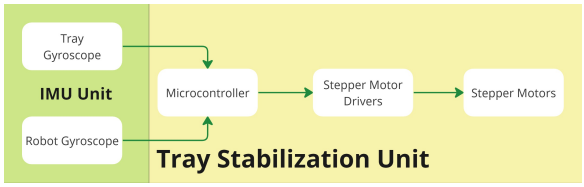
4.4 Tray Stabilization Planning

In the second week, we focused on how to stabilize the tray and what are the hardware resources needed.

- Defined the goals for tray stabilization, such as minimizing spills and maintaining balance while the robot is in motion.
- Conducted research on existing stabilization technologies and selecting the most appropriate approach (mechanical vs. electronic stabilization).
- Created an initial plan for designing, prototyping, and testing the stabilization mechanism.

Required hardware:

- Atmega2560
- Gyrosensor
- Stepper motor driver x 2
- Stepper motor x 2

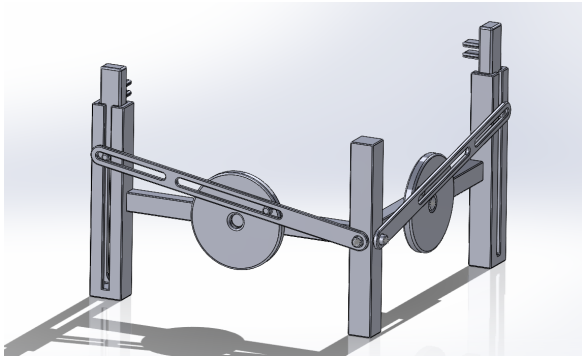


Tray Stabilization Block Diagram

4.5 Tray Stabilization Mechanism Design

In the third week, we have focused to develop a better tray stabilization mechanism.

- Developed initial concepts and sketches for the stabilization mechanism.
- Simulated the mechanism's performance using Solidworks software to ensure feasibility.



Initial Mechanism

4.6 Stabilization Mechanism Modification

During the fourth week, the initial stabilization mechanism underwent modifications based on initial tests.

- Evaluated the performance of the initial prototypes and identifying areas for improvement.
- Made iterative adjustments to the design to enhance stability and robustness.
- Conducted further tests to validate the modifications and ensure the mechanism meets stability requirements.

4.7 PID Implementation for Navigation

In the fifth week, we focused on implementing the PID control algorithm for navigation.

- Wrote and implementing the PID control algorithms to manage the robot's speed and direction.
- Conducted initial tuning of the PID parameters to achieve stable and responsive navigation.

4.8 Navigation Code Modification

In the sixth week, modifications were made to the navigation code to enhance integration with the stabilization system.

- Navigation algorithms updated in order to incorporate feedback from stabilization tests.
- Enhancing the code to improve pathfinding accuracy and obstacle avoidance.
- Testing the modified code in various scenarios to ensure reliability and efficiency.

```
#include <Arduino.h>
#include <Wire.h>


// Motor A pins
const int RPWM1 = 6;
const int LPWM1 = 7;
const int R_EN1 = 24;  //PWM pin for Speed control
```

```
const int L_EN1 = 25;  //PWM pin for Speed control

// Motor B pins
const int RPWM2 = 8;
const int LPWM2 = 9;
const int R_EN2 = 26;  //PWM pin for Speed control
const int L_EN2 = 27;  //PWM pin for Speed control

// Define Speeed values
const int MAX_SPEED = 200;
const int MIN_SPEED = 100;
const int BASE_SPEED = 150;

//Encoder values
volatile int lastEncoded1 = 0;
volatile long encoderValue1 = 0;  //This is the current encoder value
int lastMSB1 = 0;
int lastLSB1 = 0;

volatile int lastEncoded2 = 0;
volatile long encoderValue2 = 0;
int lastMSB2 = 0;
int lastLSB2 = 0;

bool encoderFlag = true;
bool TimeFlag = true;
bool distanceflag = true;

// Define encoder pins
const int encoder1APin = 3;
const int encoder1BPin = 4;
const int encoder2APin = 5;
const int encoder2BPin = 6;

//Encoder Details
int encoderResolution = 3000;  //Not sure

//Radius values
const double R = 50;
const double r = 10;

// Define PID parameters
double Kp = 1.0;  // Proportional gain
double Ki = 0.0;  // Integral gain
double Kd = 0.0;  // Derivative gain

// Define variables
//double setpoint1;  // Desired speed for both motors
//double setpoint2 ;  // Desired speed for both motors
double actualMotorSpeed1 = 0.0;  // Current speed of motor 1
double actualMotorSpeed2 = 0.0;  // Current speed of motor 2

double errorSum = 0.0;   // Integral of error
double lastError = 0.0;  // Previous error

//Angle errors
double errorSum1;
double errorSum2;

double lastError1;
double lastError2;

// Variables for tracking counts and time
volatile long prevCount1 = 0;
volatile long prevCount2 = 0;
unsigned long prevTime = 0;
unsigned long currentTime;
unsigned long nowTime;
```

```
77
78 //Motor speed control pins
79 //const int leftMotorChannel = 0;
80 //const int rightMotorChannel = 1;
81
82 // Ultrasonic minimum distance
83 int distance_to_Obstacle = 30;
84
85 // Define UART values
86 const int STOP = 0;
87 const int SLOW_FORWARD = 1;
88 const int MEDIUM_FORWARD = 2;
89 const int FAST_FORWARD = 3;
90 const int SLOW_RIGHT = 4;
91 const int SLOW_LEFT = 5;
92 const int RIGHT_90 = 6;
93 const int LEFT_90 = 7;
94
95 int receivedInt = 0;
96
97 int expectedRotationValue;
98
99 int startEncoderValue1;
100 int startEncoderValue2;
101 int starttime;
102
103 int newLocation1;
104 int newLocation2;
105
106
107 // Function to read distance from a single ultrasonic sensor
108 int readDistance(int address) {
109   Wire.beginTransmission(address);
110   Wire.write(byte(0x00));
111   Wire.write(byte(0x51));
112   Wire.endTransmission();
113
114   delay(70);
115
116   Wire.beginTransmission(address);
117   Wire.write(byte(0x02));
118   Wire.endTransmission();
119
120   Wire.requestFrom(address, 2);
121
122   if (2 <= Wire.available()) {
123     int reading = Wire.read() << 8;
124     reading |= Wire.read();
125     return reading;
126   }
127
128   return -1;  // Return -1 if reading was unsuccessful
129 }
130
131 //Find the absolute value
132 double absvalue(double value) {
133   if (value < 0) {
134     return -value;
135   } else {
136     return value;
137   }
138 }
139
140 // PID control function
141 void pidControl_speed(double setpoint1, double setpoint2) {
142   // Measure actual speeds of motors (encoder feedback or other methods)
143   double actualSpeed1 = absvalue(getActualSpeed1());
144   double actualSpeed2 = absvalue(getActualSpeed2());
```

```
145
146   // Calculate error
147   double error1 = setpoint1 - actualSpeed1;
148   double error2 = setpoint2 - actualSpeed2;
149
150   // Integral term
151   errorSum += error1 + error2;
152
153   // Derivative term
154   double dError = (error1 + error2) - lastError;
155
156   // Calculate PID output
157   double output1 = Kp * error1 + Ki * errorSum + Kd * dError;
158   double output2 = Kp * error2 + Ki * errorSum + Kd * dError;
159
160   // Apply PID output to adjust motor speeds
161   adjustMotorSpeed(output1, output2, BASE_SPEED);
162
163   // Update last error
164   lastError = error1 + error2;
165 }
166
167 void pidControl_distance(double setpoint1, double setpoint2) {
168   if (distanceflag) {
169     newLocation1 = encoderValue1;
170     newLocation2 = encoderValue2;
171     distanceflag = false;
172   }
173   // Calculate error
174   double error1 = setpoint1 - (encoderValue1 - newLocation1);
175   double error2 = setpoint2 - (encoderValue2 - newLocation1);
176
177   // Integral term
178   errorSum += error1 + error2;
179
180   // Derivative term
181   double dError = (error1 + error2) - lastError;
182
183   // Calculate PID output
184   double output1 = Kp * error1 + Ki * errorSum + Kd * dError;
185   double output2 = Kp * error2 + Ki * errorSum + Kd * dError;
186
187   // Apply PID output to adjust motor speeds
188   adjustMotorSpeed(output1, output2, 0);
189
190   // Update last error
191   lastError = error1 + error2;
192 }
193
194 // PID control function for angle
195 void pidControl_angleR(int startEncoderValue1, int startEncoderValue2) {
196
197   // Calculate error
198   double error1 = (encoderValue1 - startEncoderValue1) - expectedRotationValue;
199   double error2 = (encoderValue2 - startEncoderValue2) + expectedRotationValue;
200
201   // Integral term
202   errorSum1 += error1;
203   errorSum2 += error2;
204
205   // Derivative term
206   double dError1 = error1 - lastError1;
207   double dError2 = error2 - lastError2;
208
209   // Calculate PID output
210   double output1 = Kp * error1 + Ki * errorSum1 + Kd * dError1;
211   double output2 = Kp * error2 + Ki * errorSum2 + Kd * dError2;
212
```

```
  adjustMotorSpeed(output1, output2, 0);

  // Update last error
  lastError1 = error1;
  lastError2 = error2;
}

// PID control function for angle
void pidControl_angleL(int startEncoderValue1, int startEncoderValue2) {

  // Calculate error
  double error1 = (encoderValue1 - startEncoderValue1) + expectedRotationValue;
  double error2 = (encoderValue2 - startEncoderValue2) - expectedRotationValue;

  // Integral term
  errorSum1 += error1;
  errorSum2 += error2;

  // Derivative term
  double dError1 = error1 - lastError1;
  double dError2 = error2 - lastError2;

  // Calculate PID output
  double output1 = Kp * error1 + Ki * errorSum1 + Kd * dError1;
  double output2 = Kp * error2 + Ki * errorSum2 + Kd * dError2;

  // Apply PID output to adjust motor speeds
  adjustMotorSpeed(output1, output2, 0);

  // Update last error
  lastError1 = error1;
  lastError2 = error2;
}


//Set the motor speed
/*void speed(int leftSpeed, int rightSpeed) {
  // Set speed for left motor
  analogWrite(leftMotorChannel, leftSpeed);

  // Set speed for right motor
  analogWrite(rightMotorChannel, rightSpeed);
}*/


// Function to adjust motor speeds based on PID output
void adjustMotorSpeed(double output1, double output2, double SPEED) {
  // Adjust motor speeds using PID output

  int offset1 = 0;
  int offset2 = 0;
  actualMotorSpeed1 = SPEED + output1 + offset1;
  actualMotorSpeed2 = SPEED + output2 + offset2;
}

// Function to get actual speed of motor 1 (replace with your own implementation)
double getActualSpeed1() {

  // Calculate time interval
  unsigned long timeInterval = currentTime - prevTime;

  // Read encoder counts
  long currentCount1 = encoderValue1;

  // Calculate speed for motor 1
  double speed1 = calculateSpeed(prevCount1, currentCount1, timeInterval);

  // Update previous count and time
```

```
281   prevCount1 = currentCount1;
282   prevTime = currentTime;
283   // Measure actual speed of motor 1 using encoders or other feedback mechanisms
284   return speed1;
285 }
286
287 double getActualSpeed2() {
288
289   // Calculate time interval
290   unsigned long timeInterval = currentTime - prevTime;
291
292   // Read encoder counts
293   long currentCount2 = encoderValue2;
294
295   // Calculate speed for motor 1
296   double speed2 = calculateSpeed(prevCount2, currentCount2, timeInterval);
297
298   // Update previous count and time
299   prevCount2 = currentCount2;
300   prevTime = currentTime;
301   // Measure actual speed of motor 1 using encoders or other feedback mechanisms
302   return speed2;
303 }
304
305
306 // Function to calculate actual speed
307 double calculateSpeed(long prevCount, long currentCount, unsigned long timeInterval)
      {
308   long deltaCount = currentCount - prevCount;
309   double revolutions = deltaCount / encoderResolution;  // Convert counts to
        revolutions
310   double timeSeconds = timeInterval / 1000.0;           // Convert milliseconds to
        seconds
311   double speed = (revolutions / timeSeconds) * 60.0;    // Convert to RPM
312   return speed;
313 }
314
315 void updateEncoder() {
316
317   //Encoder Update for Encoder 1
318   int MSB1 = digitalRead(encoder1APin);  //MSB = most significant bit
319   int LSB1 = digitalRead(encoder1BPin);  //LSB = least significant bit
320
321   int encoded1 = (MSB1 << 1) | LSB1;          //converting the 2 pin value to single
        number
322   int sum1 = (lastEncoded1 << 2) | encoded1;  //adding it to the previous encoded
        value
323
324   if (sum1 == 0b1101 || sum1 == 0b0100 || sum1 == 0b0010 || sum1 == 0b1011)
        encoderValue1++;
325   if (sum1 == 0b1110 || sum1 == 0b0111 || sum1 == 0b0001 || sum1 == 0b1000)
        encoderValue1--;
326
327   lastEncoded1 = encoded1;  //store this value for next time
328
329   //Encoder Update for Encoder 2
330   int MSB2 = digitalRead(encoder2APin);  //MSB = most significant bit
331   int LSB2 = digitalRead(encoder2BPin);  //LSB = least significant bit
332
333   int encoded2 = (MSB2 << 1) | LSB2;          //converting the 2 pin value to single
        number
334   int sum2 = (lastEncoded2 << 2) | encoded2;  //adding it to the previous encoded
        value
335
336   if (sum2 == 0b1101 || sum2 == 0b0100 || sum2 == 0b0010 || sum2 == 0b1011)
        encoderValue2++;
337   if (sum2 == 0b1110 || sum2 == 0b0111 || sum2 == 0b0001 || sum2 == 0b1000)
        encoderValue2--;
```

```
338
339   lastEncoded2 = encoded2;  //store this value for next time
340 }
341
342
343
344
345 /*void controlMotors(int pinA1, int pinA2, int pinB1, int pinB2, double A_SPEED =
      150, double B_SPEED = 150) {
346   //Motor 1 Control
347   digitalWrite( motorA1, pinA1);
348   digitalWrite( motorA2, pinA2);
349
350   //Motor 2 Control
351   digitalWrite( motorB1, pinB1);
352   digitalWrite( motorB2, pinB2);
353
354   //Control the speeds of the both motors
355   pidControl_speed(A_SPEED, B_SPEED);
356 }*/
357 void distance(int set_point) {
358   pidControl_distance(set_point, set_point);
359   pidControl_speed(actualMotorSpeed1, actualMotorSpeed2);
360   int error = set_point - (encoderValue1 - newLocation1);
361   if (error >= 10) {
362     Forward(actualMotorSpeed1, actualMotorSpeed2);
363   } else {
364     if (TimeFlag) {
365       starttime = currentTime;
366       TimeFlag = false;
367     }
368     if (currentTime - starttime < 2000) {
369       Forward(actualMotorSpeed1, actualMotorSpeed2);
370     } else {
371       TimeFlag = true;
372     }
373   }
374 }
375 // Function to stop the motors
376 void stopMotors() {
377   //controlMotors(0, 0, 0, 0);
378   analogWrite(LPWM1, 0);
379   analogWrite(RPWM1, 0);
380
381
382   analogWrite(LPWM2, 0);
383   analogWrite(RPWM2, 0);
384 }
385
386 void Forward(int A_Speed, int B_Speed) {
387   //controlMotors(1, 0, 1, 0, A_Speed, B_Speed);
388   pidControl_speed(A_Speed, B_Speed);
389   analogWrite(LPWM1, actualMotorSpeed1);
390   digitalWrite(RPWM1, HIGH);
391
392
393   analogWrite(LPWM2, actualMotorSpeed2);
394   digitalWrite(RPWM2, HIGH);
395 }
396
397 void Motor1L(int Speed) {
398   analogWrite(LPWM1, Speed);
399   digitalWrite(RPWM1, HIGH);
400 }
401
402 void Motor1R(int Speed) {
403   digitalWrite(LPWM1, HIGH);
404   analogWrite(RPWM1, Speed);
```

```
}

void Motor2R(int Speed) {
  digitalWrite(LPWM2, HIGH);
  analogWrite(RPWM2, Speed);
}

void Motor2L(int Speed) {
  analogWrite(LPWM2, Speed);
  digitalWrite(RPWM2, HIGH);
}

/*void Rotate_Slow(int direction){
  controlMotors(1, 0, 0, 1, BASE_SPEED, BASE_SPEED);
}*/

void Rotate_90(int direction) {

  if (encoderFlag) {
    startEncoderValue1 = encoderValue1;
    startEncoderValue2 = encoderValue2;
    encoderFlag = false;
  }


  if (direction == 0) {  // 0 for right 90 rotation
    pidControl_angleR(startEncoderValue1, startEncoderValue2);
    if (lastError1>= 10 && lastError2 >= 10) {
        Motor1L(actualMotorSpeed1);
        Motor2R(actualMotorSpeed2);
      }

    else {
      if (TimeFlag) {
        starttime = currentTime;
        TimeFlag = false;
      }
      if (currentTime - starttime < 2000) {
        if (lastError1>= 0 && lastError2 >= 0) {
            Motor1L(actualMotorSpeed1);
            Motor2R(actualMotorSpeed2);
          }
        else if (lastError1 < 0 && lastError2 < 0) {
            Motor1R(actualMotorSpeed1);
            Motor2L(actualMotorSpeed2);
          }
        else if (lastError1 < 0){
            Motor1R(actualMotorSpeed1);
          }

        else if (lastError2 < 0) {
            Motor2L(actualMotorSpeed2);
          }
      } else {
        TimeFlag = true;
      }
    }
  }

  if (direction == 1) {  // 0 for left 90 rotation
    pidControl_angleL(startEncoderValue1, startEncoderValue2);
    if (lastError1>= 10 && lastError2 >= 10) {
        Motor1R(actualMotorSpeed1);
        Motor2L(actualMotorSpeed2);
      }
  } else {
    if (TimeFlag) {
      starttime = currentTime;
```

```
473       TimeFlag = false;
474     }
475     if (currentTime - starttime < 2000) {
476       if (lastError1>= 0 && lastError2 >= 0) {
477           Motor1R(actualMotorSpeed1);
478           Motor2L(actualMotorSpeed2);
479         }
480       else if (lastError1 < 0 && lastError2 < 0) {
481           Motor1L(actualMotorSpeed1);
482           Motor2R(actualMotorSpeed2);
483         }
484       else if (lastError1 < 0) {
485           Motor1L(actualMotorSpeed1);
486         }
487
488       else if (lastError2 < 0) {
489           Motor2R(actualMotorSpeed2);
490         }
491     } else {
492       TimeFlag = true;
493     }
494   }
495
496   /*
497         if ((absvalue(encoderValue1 - startEncoderValue1) <= expectedRotationValue)
                && (absvalue(encoderValue2 - startEncoderValue2) <= expectedRotationValue
                )){
498             pidControl_speed(A_SPEED, B_SPEED);
499             analogWrite(LPWM1, actualMotorSpeed1);
500             analogWrite(RPWM1, 0);
501
502
503             analogWrite(LPWM2, 0);
504             analogWrite(RPWM2, actualMotorSpeed2);
505         }
506         else if (absvalue(encoderValue1 - startEncoderValue1) < expectedRotationValue
                ){
507           controlMotors(1, 0, 0, 0, MAX_SPEED, MAX_SPEED);
508         }
509         else if (absvalue(encoderValue2 - startEncoderValue2) < expectedRotationValue
                ){
510           controlMotors(0, 0, 0, 1, MAX_SPEED, MAX_SPEED);
511
512         else{
513           encoderFlag = true;
514           receivedInt = 0;
515         }
516     }
517     else if (direction == 1){ // 1 for left 90 rotation
518
519         if (((absvalue(encoderValue1 - startEncoderValue1) <= expectedRotationValue))
                && (absvalue(encoderValue2 - startEncoderValue2) <=
                expectedRotationValue)){
520             pidControl_speed(A_SPEED, B_SPEED);
521             analogWrite(LPWM1, 0);
522             analogWrite(RPWM1, actualMotorSpeed1);
523
524
525             analogWrite(LPWM2, actualMotorSpeed2);
526             analogWrite(RPWM2, 0);
527         }
528         /*else if (absvalue(encoderValue1 - startEncoderValue1) <
                expectedRotationValue){
529           controlMotors(1, 0, 0, 0, MAX_SPEED, MAX_SPEED);
530         }
531         else if (absvalue(encoderValue2 - startEncoderValue2) < expectedRotationValue
                ){
532           controlMotors(0, 0, 0, 1, MAX_SPEED, MAX_SPEED);
```

```
        }
        else{
          encoderFlag = true;
          receivedInt = 0;
        }
    }*/
}

void setup() {
  currentTime = millis();
  //Encoder Setup
  Serial.begin(9600);

  //Motor1 pin define
  pinMode(RPWM1, OUTPUT);
  pinMode(LPWM1, OUTPUT);
  pinMode(R_EN1, OUTPUT);
  pinMode(L_EN1, OUTPUT);

  ////Motor1 pin define
  pinMode(RPWM2, OUTPUT);
  pinMode(LPWM2, OUTPUT);
  pinMode(R_EN2, OUTPUT);
  pinMode(L_EN2, OUTPUT);

  //Pull up the EN pins of the motor driver
  digitalWrite(R_EN1, HIGH);
  digitalWrite(L_EN1, HIGH);

  digitalWrite(R_EN2, HIGH);
  digitalWrite(L_EN2, HIGH);

  pinMode(encoder1APin, INPUT);
  pinMode(encoder1BPin, INPUT);

  digitalWrite(encoder1APin, HIGH);  //turn pullup resistor on
  digitalWrite(encoder1BPin, HIGH);  //turn pullup resistor on

  //call updateEncoder() when any high/low changed seen
  //on interrupt 0 (pin 2), or interrupt 1 (pin 3)
  attachInterrupt(0, updateEncoder, CHANGE);
  attachInterrupt(1, updateEncoder, CHANGE);

  expectedRotationValue = (R / (4 * r)) * encoderResolution;
}

void loop() {
  nowTime = micros();

  if (Serial.available() > 0) {
    // Read the incoming byte
    receivedInt = Serial.parseInt();
  }

  // Check the received command and control motors accordingly
  switch (receivedInt) {
    case STOP:
      stopMotors();
      break;
    case SLOW_FORWARD:
      Forward(MIN_SPEED, MIN_SPEED);
      break;
    case MEDIUM_FORWARD:
      Forward(BASE_SPEED, BASE_SPEED);
      break;
    case FAST_FORWARD:
      Forward(MAX_SPEED, MAX_SPEED);
      break;
```

```
/*     case SLOW_RIGHT:
       Rotate_Slow(0);
       break;
     case SLOW_LEFT:
       Rotate_Slow(1);
       break;*/
     case RIGHT_90:
       Rotate_90(0);
     case LEFT_90:
       Rotate_90(1);
       break;
     default:
       // Stop the motors if invalid command received
       stopMotors();
       break;
   }
   while (true) {
     if (micros() - nowTime > 2000) {
       break;
     }
   }
}
```

4.9 Kalman Filter Implementation

In the seventh week, we have implemented the Kalman filter to improve stabilization.

- Wrote and integrated the Kalman filter into the stabilization code to process sensor data more accurately
- Conducted tests to validate the performance of the Kalman filter in different operating conditions.

```
#include <Wire.h>
float RateRoll, RatePitch, RateYaw;
float AngleRoll, AnglePitch;
float AccX, AccY, AccZ;
float AccZInertial;
float LoopTimer;
uint16_t dig_T1, dig_P1;
int16_t dig_T2, dig_T3, dig_P2, dig_P3, dig_P4, dig_P5;
int16_t dig_P6, dig_P7, dig_P8, dig_P9;
float AltitudeBarometer, AltitudeBarometerStartUp;
int RateCalibrationNumber;
#include <BasicLinearAlgebra.h>
using namespace BLA;
float AltitudeKalman, VelocityVerticalKalman;
BLA::Matrix<2,2> F; BLA::Matrix<2,1> G;
BLA::Matrix<2,2> P; BLA::Matrix<2,2> Q;
BLA::Matrix<2,1> S; BLA::Matrix<1,2> H;
BLA::Matrix<2,2> I; BLA::Matrix<1,1> Acc;
BLA::Matrix<2,1> K; BLA::Matrix<1,1> R;
BLA::Matrix<1,1> L; BLA::Matrix<1,1> M;
BLA::Matrix<1,1> LI;
BLA::Matrix<2,2> GI={100,0,0,100};
void kalman_2d(void) {
  Acc = { AccZInertial };
  S = F * S + G * Acc;
  P = F * P * ~F + Q;
  L = H * P * ~H + R;
  LI=L;
  Invert(LI);
  K = P * ~H * LI;
  M = { AltitudeBarometer };
  S = S + K * (M - H * S);
  AltitudeKalman = S(0, 0);
```

```
  VelocityVerticalKalman = S(1, 0);
  P = (I - K * H) * P;
}
void barometer_signals(void) {
  Wire.beginTransmission(0x76);
  Wire.write(0xF7);
  Wire.endTransmission();
  Wire.requestFrom(0x76, 6);
  uint32_t press_msb = Wire.read();
  uint32_t press_lsb = Wire.read();
  uint32_t press_xlsb = Wire.read();
  uint32_t temp_msb = Wire.read();
  uint32_t temp_lsb = Wire.read();
  uint32_t temp_xlsb = Wire.read();
  unsigned long int adc_P = (press_msb << 12) | (press_lsb << 4) | (press_xlsb >> 4);
  unsigned long int adc_T = (temp_msb << 12) | (temp_lsb << 4) | (temp_xlsb >> 4);
  signed long int var1, var2;
  var1 = ((((adc_T >> 3) - ((signed long int)dig_T1 << 1))) * ((signed long int)
      dig_T2)) >> 11;
  var2 = (((((adc_T >> 4) - ((signed long int)dig_T1)) * ((adc_T >> 4) - ((signed
      long int)dig_T1)))>> 12)* ((signed long int)dig_T3))>> 14;
  signed long int t_fine = var1 + var2;
  unsigned long int p;
  var1 = (((signed long int)t_fine) >> 1) - (signed long int)64000;
  var2 = (((var1 >> 2) * (var1 >> 2)) >> 11) * ((signed long int)dig_P6);
  var2 = var2 + ((var1 * ((signed long int)dig_P5)) << 1);
  var2 = (var2 >> 2) + (((signed long int)dig_P4) << 16);
  var1 = (((dig_P3 * (((var1 >> 2) * (var1 >> 2)) >> 13)) >> 3) + ((((signed long int
      )dig_P2) * var1) >> 1)) >> 18;
  var1 = ((((32768 + var1)) * ((signed long int)dig_P1))>> 15);
  if (var1 == 0) { p = 0; }
  p = (((unsigned long int)(((signed long int)1048576) - adc_P) - (var2 >> 12))) *
      3125;
  if (p < 0x80000000) {p = (p << 1) / ((unsigned long int)var1);
  } else {p = (p / (unsigned long int)var1) * 2;}
  var1 = (((signed long int)dig_P9) * ((signed long int)(((p >> 3) * (p >> 3)) >> 13)
      )) >> 12;
  var2 = (((signed long int)(p >> 2)) * ((signed long int)dig_P8)) >> 13;
  p = (unsigned long int)((signed long int)p + ((var1 + var2 + dig_P7) >> 4));
  double pressure = (double)p / 100;
  AltitudeBarometer = 44330 * (1 - pow(pressure / 1013.25, 1 / 5.255)) * 100;
}
void gyro_signals(void) {
  Wire.beginTransmission(0x68);
  Wire.write(0x1A);
  Wire.write(0x05);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x1C);
  Wire.write(0x10);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  Wire.endTransmission();
  Wire.requestFrom(0x68, 6);
  int16_t AccXLSB = Wire.read() << 8 | Wire.read();
  int16_t AccYLSB = Wire.read() << 8 | Wire.read();
  int16_t AccZLSB = Wire.read() << 8 | Wire.read();
  Wire.beginTransmission(0x68);
  Wire.write(0x1B);
  Wire.write(0x8);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x43);
  Wire.endTransmission();
  Wire.requestFrom(0x68, 6);
  int16_t GyroX = Wire.read() << 8 | Wire.read();
  int16_t GyroY = Wire.read() << 8 | Wire.read();
```

```
  int16_t GyroZ = Wire.read() << 8 | Wire.read();
  RateRoll = (float)GyroX / 65.5;
  RatePitch = (float)GyroY / 65.5;
  RateYaw = (float)GyroZ / 65.5;
  AccX = (float)AccXLSB / 4096 - 0.05;
  AccY = (float)AccYLSB / 4096 + 0.01 + 0.01;
  AccZ = (float)AccZLSB / 4096 - 0.11 - 0.11;
  AngleRoll = atan(AccY / sqrt(AccX * AccX + AccZ * AccZ)) * 1 / (3.142 / 180);
  AnglePitch = -atan(AccX / sqrt(AccY * AccY + AccZ * AccZ)) * 1 / (3.142 / 180);
}
void setup() {
  Serial.begin(57600);
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
  Wire.setClock(400000);
  Wire.begin();
  delay(250);
  Wire.beginTransmission(0x68);
  Wire.write(0x6B);
  Wire.write(0x00);
  Wire.endTransmission();
  Wire.beginTransmission(0x76);
  Wire.write(0xF4);
  Wire.write(0x57);
  Wire.endTransmission();
  Wire.beginTransmission(0x76);
  Wire.write(0xF5);
  Wire.write(0x14);
  Wire.endTransmission();
  uint8_t data[24], i = 0;
  Wire.beginTransmission(0x76);
  Wire.write(0x88);
  Wire.endTransmission();
  Wire.requestFrom(0x76, 24);
  while (Wire.available()) {
    data[i] = Wire.read();
    i++;
  }
  dig_T1 = (data[1] << 8) | data[0];
  dig_T2 = (data[3] << 8) | data[2];
  dig_T3 = (data[5] << 8) | data[4];
  dig_P1 = (data[7] << 8) | data[6];
  dig_P2 = (data[9] << 8) | data[8];
  dig_P3 = (data[11] << 8) | data[10];
  dig_P4 = (data[13] << 8) | data[12];
  dig_P5 = (data[15] << 8) | data[14];
  dig_P6 = (data[17] << 8) | data[16];
  dig_P7 = (data[19] << 8) | data[18];
  dig_P8 = (data[21] << 8) | data[20];
  dig_P9 = (data[23] << 8) | data[22];delay(250);
  for (RateCalibrationNumber = 0;RateCalibrationNumber < 2000;RateCalibrationNumber
      ++) {
    barometer_signals();
    AltitudeBarometerStartUp +=
      AltitudeBarometer;delay(1);
  }
  AltitudeBarometerStartUp /= 2000;
  F = { 1, 0.004,
        0, 1 };
  G = { 0.5 * 0.004 * 0.004,
        0.004 };
  H = { 1, 0 };
  I = { 1, 0,
        0, 1 };
  Q = G * ~G *GI   ;
  R = { 30 * 30 };
  P = { 0, 0,
        0, 0 };
```

```
  S = { 0,
        0 };
  LoopTimer = micros();
}
void loop() {
  gyro_signals();
  AccZInertial = -sin(AnglePitch * (3.142 / 180)) * AccX+ cos(AnglePitch * (3.142 /
      180)) * sin(AngleRoll * (3.142 / 180)) * AccY + cos(AnglePitch * (3.142 / 180))
       * cos(AngleRoll * (3.142 / 180)) * AccZ;
  AccZInertial = (AccZInertial - 1) * 9.81 * 100;
  barometer_signals();
  AltitudeBarometer -= AltitudeBarometerStartUp;
  kalman_2d();
  Serial.print("Altitude [cm]: ");
  Serial.print(AltitudeKalman);
  Serial.print(" Vertical velocity [cm/s]: ");
  Serial.println(VelocityVerticalKalman);
  while (micros() - LoopTimer < 4000);
  LoopTimer = micros();
}
```

4.10 Stabilization Code Implementation

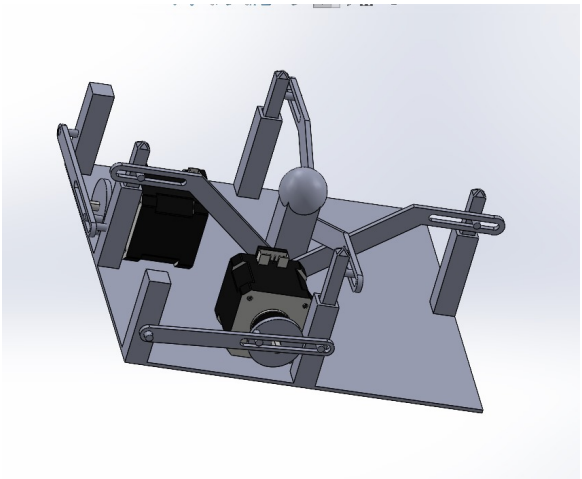
In the eighth week, the final stabilization code was implemented and tested.

- Completing the development of stabilization algorithms based on the Kalman filter outputs.

4.11 3D Design of Stabilization Mechanism

The ninth week, we have created detailed 3D design of the stabilization mechanism.

- Solidworks software was used to create precise 3D models of the final stabilization mechanism.
- Mechanism was simulated to verify the mechanical integrity and functionality of the design.



Modified Mechanism

4.12 Testing and Optimization

The tenth week was dedicated to testing and optimizing the integrated system.

- Analyzed test results to identify any issues or areas for improvement.

4.13 Code Modification

During the eleventh week, we focused on refining and optimizing the code for both tray stabilization and navigation.

- Fine-tuned stabilization and navigation algorithms were modified based on extended testing results.
- Enhanced the efficiency and accuracy of the code to ensure smooth operation under various conditions.

5 Daily Log Entries- Software

This section describes about our daily progress in developing the restaurant robot with the relavant timeframe. While the initial part describes mainly about setting up the hardware and ROS1 later part describe about setting up our robot for ROS2 with the reasons for transition.

5.1 Planning about the Navigation Method for our robot (15 - 19 February 2024)

Our robot navigation system plays an important role in its ability to function effectively within the restaurant environment. To achieve this, we plan to combine the power of a Kinect v2 depth camera, point cloud processing, and the Robot Operating System (ROS). Here's a breakdown of our planned navigation approach:

1. **Utilizing the Kinect v2:**
   The Kinect v2 provides depth data, allowing us to construct real-time 3D point clouds of the surroundings. This point cloud data offers a rich representation of the environment, including walls, furniture, and potential obstacles. This point cloud data can be used to localize our robot in the restaurant environment.
2. **Point Cloud Processing and Laser Scan Conversion:**
   We get to know about ROS packages like depthimage_to_laserscan to convert the obtained 3D point cloud information into a 2D laser scan format. Laser scans are a standard format used in robot navigation due to their efficiency and ease of integration with ROS navigation tools. So we can use this device rather than a LiDar to achieve the same functionality(Not same but managable and its somewhat hard to set the things up as it seems)
3. **ROS Navigation Stack for Path Planning and Obstacle Avoidance:**
   As ROS offers a robust navigation stack, including navigation packages like move_base and cartographer. These tools can utilize the converted laser scan data to build a map of the restaurant layout and plan safe trajectories for LUNA to navigate. The navigation stack will also factor in real-time sensor data to identify and avoid obstacles dynamically.

We finalized this plan for navigation of the robot due to these benifits of the approach,

- **Rich Environmental Understanding:** The Kinect v2 provides a detailed point cloud, allowing for a more comprehensive understanding of the environment compared to traditional laser scanners alone.
- **Flexibility and Customization:** ROS offers a wide range of navigation tools and libraries, enabling customization of the navigation behavior to suit the specific needs of the restaurant environment.

5.2 Selecting Robot Operating System version (19 - 21 February 2024)

Developing software for robots can be a complex task. This is where Robot Operating Systems (ROS) come in. ROS 1 and ROS 2 are two popular open-source frameworks that provide a collection of tools and libraries to simplify robot software development. While ROS 1 has been the established standard for many years, ROS 2 is the next generation offering advancements in performance, scalability, and architecture. Understanding the strengths and considerations of each version is crucial for choosing the right fit for the project. However, with the development of ROS2, significant improvements have been made to address some of the limitations of the original ROS. So we have to consider about both the ROS versions when deciding a suitable framework for our Restaurant Robot Project.

ROS1 (Robot Operating System 1) has been the go-to framework for robot software development for many years. It provides a collection of tools and libraries that simplify common tasks in robot programming, like sending messages between different parts of a robot's software, managing hardware devices, and packaging reusable robot code.

One of the key strengths of ROS1 is its **large and active community**. This means a wealth of existing libraries and tutorials are available, making it easier to find solutions and get started with robot development. Additionally, ROS 1 has been battle-tested in numerous real-world robotics applications, demonstrating its reliability and robustness.Here's a closer look at some of the core functionalities of ROS1:

- Message-Passing System: At its heart, ROS 1 utilizes a custom message-passing system that allows different software components on a robot to communicate with each other. This system is facilitated by a central server called ROS Master, which keeps track of all the communicating entities and routes messages accordingly.
- Separate APIs: ROS 1 offers separate APIs for C++ (roscpp) and Python (rospy). This can be advantageous for seperate language proficiencies

Overall, ROS 1 is a mature and well-established framework with a rich ecosystem of tools and resources.**So we decided to move forward with ROS1.**

5.3 ROS Serial Communication with the PCB (21 - 23 February 2024)

For serial communication with the PCB we have decided to use the UART protocol. for enabling serial communication with the SBC we have to use few libraries namely rosserial and setup micro-controller programming IDE to program the micro-controller with use of ros.h library we have used the following commands.

```
sudo apt-get install ros-${ROS_DISTRO}-rosserial-arduino
sudo apt-get install ros-${ROS_DISTRO}-rosserial
```

For enabling serial communication with UART protocol in the SBC we have to enable pins of the SBC to communicate with micro-controller.

```
ln -s /boot/firmware/cmdline.txt /boot/cmdline.txt
echo "#enable_uart=1" >> example.txt
```

Always we have to enable the serial connection with ros We enable the serial connection with the selected port. Here we have selected the port ttyS0 and activate the nodes to be communicated. We use ros.h library when programming the microcontroller to get node handling functionalities.

```
%add chmod command
rosrun rosserial_python serial_node.py /dev/ttyS0
rosrun resrob_serial publisher.py  #resrob_serial is the rospackage we store
    related scripts
```

5.4 Setting up the Device and Install Drivers (23 - 28 March 2024)

When searching for drivers for kinect v2 we found drivers from two authors namely OpenNi and Freenect. But we were unable to make OpenNi drivers although it is used widely as that project was terminated as it was bought by Apple Inc. So we decided to move on with libfreenect2 Drivers. As in the directed sub section. When following the above tutorial we have faced few issues as mentioned below

```
{[Error] [VaapiFrame] vaGetImage(display, surface, 0, 0, image.width, image.
    height, image.image_id): unknown libva error Segmentation fault (core
    dumped)}
```

After following lots of solutions this terminal command which specifies the libva driver names worked for us.

```
export LIBVA_DRIVER_NAME=i965
```

We got this kind of a image as a output consisting IR, Depth and grayscaled versions.

Output of the Protonect

5.5 Image Transferring between SBC and computer through ROS (1 - 4 March 2024)

As earlier we planned on to use a main server computer for data processing we tried transferring image stream between computer and SBC(mainly for study purposes and to check the feasibility by identifying the speed of the transfer). For that we created a ros package by,

```
    cd ~/catkin_ws/src/
    catkin_create_pkg resrob_controller rospy cv_bridge sensor_msgs
    cd resrob_controller
    mkdir scripts
    cd scripts
    cd ~/catkin_ws
    catkin_make
```

We used the following python script for image transfer between SBC and computer. As we are running SBC as publisher we create a node name ImagePublisher and as computer as subscriber we

create a node named ImageSubscriber. We used cv_bridge library to convert the image data to cv2 image format. Create a python script named image_publisher.py in the scripts folder of the package and add the following code.

```
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import CompressedImage
from cv_bridge import CvBridge
import cv2

nodeName = 'cameraSensorPublisher'
topicName = 'video_topic'

rospy.init_node(nodeName, anonymous=True)
publisher = rospy.Publisher(topicName, CompressedImage, queue_size=60)
rate = rospy.Rate(10)
video_capture = cv2.VideoCapture(0)
bridge = CvBridge()

while not rospy.is_shutdown():
    return_value, image = video_capture.read()
    if return_value:
        rospy.loginfo("Publishing image")
        compressed_image_message = bridge.cv2_to_compressed_imgmsg(image)
        publisher.publish(compressed_image_message)
    rate.sleep()
```

Follow the earlier steps on computer to create a package and create a python script named image_subscriber.py in the scripts folder of the package and add the following code.

```
import rospy
from sensor_msgs.msg import CompressedImage
from cv_bridge import CvBridge
import cv2

subscriber = "cameraSubscriber"
topic = "video_topic"

def callback(data):
    bridge = CvBridge()
    rospy.loginfo("Receiving compressed image")
    cv_image = bridge.compressed_imgmsg_to_cv2(data)
    cv2.imshow("camera", cv_image)
    cv2.waitKey(1)

rospy.init_node(subscriber, anonymous=True)
rospy.Subscriber(topic, CompressedImage, callback)
rospy.spin()
cv2.destroyAllWindows()
```

Then you can check the image transfer by running the following commands. After running **roscore** in the both devices.
On SBC

```
roscore
rosrun resrob_controller image_publisher.py
```

On Computer

```
roscore
rosrun resrob_controller image_subscriber.py
```

5.6 Working on with Taking User Gestures for Communicating with the Customers(4 - 8 March 2024)

As we haven't observed an considerable delay with transmission we planned to implement the gesture detecting model on the subscriber node which is the restaurant computer according to us. That model was implemented by us specially identify signals which are given by users to interact with them much smoothly with the help of user interface of the restaurant robot.
For implement the node on server computer we have to install few dependancies by

```
pip install tensorflow
pip install mediapipe
```

and we have to create the node for subscribing from 'video_topic' and to publish for the 'usr_command' topic which is subscribed by the user interface part. The script for the node as follows,

```
#not finished
#!/home/sandun/miniconda3/envs/gale/bin python

import rospy
from sensor_msgs.msg import CompressedImage
from cv_bridge import CvBridge
import cv2
import numpy as np
import mediapipe as mp
import tensorflow as tf
from tensorflow.keras.models import load_model

subscriber = "cameraSubscriber"
topic = "video_topic"


# initialize mediapipe
mpHands = mp.solutions.hands
hands = mpHands.Hands(max_num_hands=1, min_detection_confidence=0.7)
mpDraw = mp.solutions.drawing_utils

# Load the gesture recognizer model
model = load_model('mp_hand_gesture')

# Load class names
f = open('gesture.names', 'r')
classNames = f.read().split('\n')
f.close()

def callback(data):
    bridge = CvBridge()
    frame = bridge.compressed_imgmsg_to_cv2(data)


    x, y, c = frame.shape

    # Flip the frame vertically
    frame = cv2.flip(frame, 1)
    framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Get hand landmark prediction
    result = hands.process(framergb)

    # print(result)

    className = ''

    # post process the result
    if result.multi_hand_landmarks:
        landmarks = []
        for handslms in result.multi_hand_landmarks:
```

```
            for lm in handslms.landmark:
                # print(id, lm)
                lmx = int(lm.x * x)
                lmy = int(lm.y * y)

                landmarks.append([lmx, lmy])

            # Drawing landmarks on frames
            mpDraw.draw_landmarks(frame, handslms, mpHands.HAND_CONNECTIONS)

            # Predict gesture
            prediction = model.predict([landmarks])
            # print(prediction)
            classID = np.argmax(prediction)
            className = classNames[classID]


    # show the prediction on the frame
    cv2.putText(frame, className, (10, 50), cv2.FONT_HERSHEY_SIMPLEX,
                   1, (0,0,255), 2, cv2.LINE_AA)

    cv2.imshow("Output", frame)
    cv2.waitKey(1)

rospy.init_node(subscriber, anonymous=True)
rospy.Subscriber(topic, CompressedImage, callback)
rospy.spin()
cv2.destroyAllWindows()
```

5.7 Saving data for testing purposes(8 - 9 March 2024)

We used the following python script for achieving depth and RGB data for our testing purposes.

```
# coding: utf-8

import numpy as np
import cv2
import sys
from pylibfreenect2 import Freenect2, SyncMultiFrameListener
from pylibfreenect2 import FrameType, Registration, Frame
from pylibfreenect2 import createConsoleLogger, setGlobalLogger
from pylibfreenect2 import LoggerLevel

try:
    from pylibfreenect2 import OpenGLPacketPipeline
    pipeline = OpenGLPacketPipeline()
except:
    try:
        from pylibfreenect2 import OpenCLPacketPipeline
        pipeline = OpenCLPacketPipeline()
    except:
        from pylibfreenect2 import CpuPacketPipeline
        pipeline = CpuPacketPipeline()
print("Packet pipeline:", type(pipeline).__name__)

# Create and set logger
logger = createConsoleLogger(LoggerLevel.Debug)
setGlobalLogger(logger)

fn = Freenect2()
num_devices = fn.enumerateDevices()
if num_devices == 0:
    print("No device connected!")
    sys.exit(1)

serial = fn.getDeviceSerialNumber(0)
device = fn.openDevice(serial, pipeline=pipeline)
```

```
listener = SyncMultiFrameListener(
    FrameType.Color | FrameType.Ir | FrameType.Depth)

# Register listeners
device.setColorFrameListener(listener)
device.setIrAndDepthFrameListener(listener)

device.start()

# NOTE: must be called after device.start()
registration = Registration(device.getIrCameraParams(),
                           device.getColorCameraParams())

undistorted = Frame(512, 424, 4)
registered = Frame(512, 424, 4)

# Optinal parameters for registration
# set True if you need
need_bigdepth = False
need_color_depth_map = False

bigdepth = Frame(1920, 1082, 4) if need_bigdepth else None
color_depth_map = np.zeros((424, 512),  np.int32).ravel() \
    if need_color_depth_map else None

while True:
    frames = listener.waitForNewFrame()

    color = frames["color"]
    ir = frames["ir"]
    depth = frames["depth"]

    registration.apply(color, depth, undistorted, registered,
                       bigdepth=bigdepth,
                       color_depth_map=color_depth_map)

    # NOTE for visualization:
    # cv2.imshow without OpenGL backend seems to be quite slow to draw all
    # things below. Try commenting out some imshow if you don't have a fast
    # visualization backend.
    cv2.imshow("ir", ir.asarray() / 65535.)
    cv2.imshow("depth", depth.asarray() / 4500.)

    #print(cv2.resize(cv2.resize(color.asarray(),(int(1920 / 3), int(1080 / 3)))[:,:,
                                                0:1],color.asarray(),(int(1920 / 3),
                                                int(1080 / 3)))[:,:,1:2],cv2.resize(
                                                color.asarray(),(int(1920 / 3), int(
                                                1080 / 3)))[:,:,2:3])
    print(depth.asarray() / 4500.)


    listener.release(frames)

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

device.stop()
device.close()

sys.exit(0)
```

We converted achived data in to tensors consisting RGB-Depth data using anothe python script.

5.8 Ubuntu Installation (9 - 11 March 2024)

5.8.1 Ubuntu Installation

We installed Ubuntu on the Single Board Computer (SBC) following the instructions provided in the previous section. The installation process was straightforward and completed without any issues.

5.8.2 ROS Installation

During the installation of ROS, we encountered several errors:

```
W: GPG error: http://packages.ros.org/ros/ubuntu focal InRelease: The following
    signatures couldn't be verified because the public key is not available: NO\
    _PUBKEY F42ED6FBAB17C654}
E: The repository 'http://packages.ros.org/ros/ubuntu focal InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled
    by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

Cause of the Errors

The cause of these errors was identified as the old GPG key having been revoked due to a recent security incident with `build.ros.org`.

Solution

To resolve these errors, we performed the following steps:

1. Deleted the old GPG key using the command:

```
sudo apt-key del 421C365BD9FF1F717815A3895523BAEEB01FA116
```

2. Added the new GPG key with the command:

```
sudo -E apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-
    key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Outcome

After executing the above commands, the GPG errors were resolved, and the ROS installation process continued without further issues. This solution addressed the security concerns by ensuring that the repository updates are verified with the correct public key.

5.9 Installing Drivers for Kinect v2 (11 - 19 March 2024)

When searching for drivers for Kinect v2, we found options from two authors: OpenNi and Freenect. However, we were unable to use OpenNi drivers despite their wide usage because the project was terminated after being acquired by Apple Inc. Consequently, we decided to use `libfreenect2` drivers as directed in the relevant subsection.

Driver Installation

Following the tutorial for `libfreenect2`, we encountered several issues. The primary problem we faced was:

```
[Error] [VaapiFrame] vaGetImage(display, surface, 0, 0, image.width, image.height,
    image.image_id): unknown libva error Segmentation fault (core dumped)
```

Cause and Solution

After exploring numerous solutions, we found that specifying the `libva` driver names via a terminal command resolved the issue:

1. Set the `LIBVA_DRIVER_NAME` environment variable:

```
export LIBVA_DRIVER_NAME=i965
```

Outcome

After executing the above command, the Kinect v2 successfully produced an output consisting of IR, Depth, and grayscale images.

Errors Encountered and Solutions

During the driver setup, the key issue was a segmentation fault caused by an unknown `libva` error. The detailed steps and their solutions are as follows:
Error: Unknown `libva` error causing segmentation fault

```
[Error] [VaapiFrame] vaGetImage(display, surface, 0, 0, image.width, image.height,
    image.image_id): unknown libva error Segmentation fault (core dumped)
```

Solution: Setting the `LIBVA_DRIVER_NAME` environment variable to `i965` resolved the issue:

```
export LIBVA_DRIVER_NAME=i965
```

Results

With the successful setup of the `libfreenect2` drivers, we obtained output images from the Kinect v2 device, including IR, Depth, and grayscale versions.

5.10 Installing Pylibfreenect (19 - 23 March 2024)

Overview

`Pylibfreenect` is a Python plugin for Kinect that allows interaction using the Python programming language. Since many of our group members are familiar with Python, we decided to install this package for better handling and examination of data.

Installation Process

We followed several tutorials for installation, but none worked, resulting in errors about invalid paths. We examined the `setup.py` file of the library, tried different commands, and identified possible path mismatches.

Solution

By modifying the `setup.py` file, we managed to resolve the path issues. Specifically, we changed line 17 of `setup.py` to:

```
libfreenect2_install_prefix = os.environ.get("LIBFREENECT2_INSTALL_PREFIX",
    "/home/sandun/freenect2")
```

Commands Executed

After modifying the `setup.py` file, we ran the following commands to install `pylibfreenect`:

1. Navigate to the `pylibfreenect` directory:

```
cd ~/pylibfreenect
```

2. Run the setup script:

```
python3 setup.py install
```

Errors Encountered and Solutions

During the installation, we faced the following error:

```
Error: Invalid path specified in setup.py
```

The error was resolved by modifying the path in the `setup.py` file as described above.

5.11 Setting up gesture detection model (23 - 27 March 2024)

Overview

We observed minimal delay in transmission, which allowed us to implement the gesture detection model on the subscriber node (the restaurant computer). This model identifies user signals to interact with them smoothly via the user interface of the restaurant robot.

Installation of Dependencies

To implement the gesture detection node on the server computer, we needed to install a few dependencies:

```
pip install tensorflow
pip install mediapipe
```

Creating the Gesture Detection Node

We created a node that subscribes to the `video_topic` and publishes to the `usr_command` topic, which is subscribed to by the user interface part.

Script Overview

The script initializes MediaPipe for hand detection, loads the gesture recognizer model, and processes the video stream to detect gestures. The detected gestures are then published for the user interface to act upon.

Errors Encountered and Solutions

During the implementation, we faced several issues:

1. **Error: Permission denied for accessing the video device**

   Solution: Ensure the script is run with appropriate permissions or use `sudo`.

2. **Error: TensorFlow version compatibility**

   Solution: Ensure that the TensorFlow version installed is compatible with the installed version of MediaPipe.

3. **Error: Model file not found**

   Solution: Verify the path to the model file in the script and ensure it is correctly specified.

Saving Data for Testing Purposes

We used a Python script to capture depth and RGB data for testing purposes. This data was later converted into tensors for further analysis.

Implementation Details

The script uses `pylibfreenect2` to capture frames from the Kinect sensor. The frames are processed and displayed for verification.

Errors Encountered and Solutions

During the data capturing process, the following issues were encountered:

1. **Error: No device connected**

   Solution: Ensure that the Kinect device is properly connected and recognized by the system.

2. **Error: Packet pipeline initialization failure**

   Solution: Try initializing different pipelines (OpenGL, OpenCL, or CPU) based on the system's capabilities.

3. **Error: Slow visualization with cv2.imshow**

   Solution: Use fewer `imshow` calls or opt for a more efficient visualization method.

4. **Error: Incorrect frame registration**

   Solution: Verify the parameters used for frame registration and ensure they match the device specifications.

Data Conversion

We converted the captured RGB-Depth data into tensors using another Python script. This allowed us to perform further analysis and testing on the data.

5.12 Transition to ROS 2 (28 March 2024)

We determined that integrating ROS 1 into our system was not feasible due to the lack of necessary tools. Consequently, we decided to switch to ROS 2. Given that ROS 2 Iron supports Ubuntu 22.04, we needed to install this operating system.

5.12.1 Setting Up Central Restaurant Computer

5.12.2 Installing Ubuntu

As the recommended operating system for ROS 2 Iron is Ubuntu 22.04 Jammy Jellyfish, we proceeded to set up Ubuntu 22.04 on both our AMD-based computer and Raspberry Pi/Jetson Nano (ARM version). The steps followed are outlined below:

1. **Download the ISO Image:**

   - Download the .iso image files for Ubuntu 22.04 from the official site: https://releases.ubuntu.com/jammy/.

2. **Create a Bootable USB Drive:**

   (a) Download Rufus, a tool for creating bootable USB drives, from https://rufus.ie/en/.

   (b) Install Rufus on your computer.

(c) Insert the USB drive you want to make bootable.

(d) Open Rufus, select your USB drive from the Device dropdown menu.

(e) Under Boot selection, choose the .iso file you downloaded for Ubuntu 22.04.

(f) Follow the prompts to create the bootable USB drive.

3. **Booting the PC from the USB Drive:**

   (a) Insert the bootable USB drive into the computer where you want to install Ubuntu 22.04.

   (b) Restart the computer and enter the boot options menu. The key to press for entering the boot menu may vary depending on your computer's manufacturer (for some it is F8; you can find the specific key for your computer online).

   (c) Select the USB drive from the boot options menu.

   (d) Follow the on-screen instructions to install Ubuntu 22.04. If you are installing it on a specific partition of your hard disk, select that partition during the installation process.

4. **Follow the Installation Tutorial:**

   - For detailed installation instructions, follow the official Ubuntu tutorial starting from step 6: https://ubuntu.com/tutorials/install-ubuntu-desktop#6-type-of-installation.

By following these steps, we successfully installed Ubuntu 22.04 Jammy Jellyfish on both the AMD-based PC and Raspberry Pi/Jetson Nano.

Issues Faced and Solutions

Issue: Bootable USB Drive Not Recognized

Solution: Ensure that the USB drive is correctly formatted and the bootable image is properly written. Double-check the BIOS/UEFI settings to make sure USB booting is enabled.

Issue: System Freezes During Installation

Solution: Try using a different USB port or another USB drive. Additionally, verify the integrity of the downloaded .iso file using checksums provided on the Ubuntu download page.

Issue: No Internet Connection After Installation

Solution: Install the necessary drivers manually. For example, for some Wi-Fi cards, you may need to install drivers from the Ubuntu repository or from the manufacturer's website.

Issue: Incorrect Display Resolution

Solution: Update the graphics drivers and configure the display settings using the terminal or the GUI provided in Ubuntu.

Next Steps

With Ubuntu 22.04 installed and configured, the next steps involve installing ROS 2 Iron and setting up the necessary development environment for our restaurant robot project. We will also focus on integrating the various components of the system and ensuring seamless communication between the devices.

5.13 Installing ROS 2 on the Device (28 March - 1 April 2024)

System Setup

Set Locale Ensure you have a locale that supports UTF-8. In minimal environments (such as a Docker container), the locale may be set to something minimal like POSIX. The following settings are tested, but any UTF-8 supported locale should work.

1. Check current locale settings:

```
locale
```

2. Install necessary locales:

```
sudo apt update \&\& sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
```

3. Verify settings:

```
locale
```

Enable Required Repositories Add the ROS 2 apt repository to your system.

1. Ensure that the Ubuntu Universe repository is enabled:

```
sudo apt install software-properties-common
sudo add-apt-repository universe}
```

2. Add the ROS 2 GPG key with apt:

```
sudo apt update \&\& sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/
    ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
```

3. Add the repository to your sources list:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
    keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/
    ubuntu $(. /etc/os-release \&\& echo \$UBUNTU_CODENAME) main" | sudo
     tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Install Development Tools (Optional) If you plan to build ROS packages or do development, install the development tools:

1. Update your apt repository caches:
   `sudo apt update`
2. Install development tools:
   `sudo apt install ros-dev-tools`

Install ROS 2 Iron

```
sudo apt install ros-iron-desktop
sudo apt install ros-iron-ros-base
```

Install Additional RMW Implementations (Optional) The default middleware that ROS 2 uses is Fast DDS. However, you can install additional RMW implementations if needed.

Setup Environment Set up your environment by sourcing the ROS 2 setup file. Add this line to your .bashrc to make it permanent:

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

Try Some Examples If you installed `ros-iron-desktop`, you can try running some examples to verify installation:

1. In one terminal, source the setup file and run a C++ talker:

```
source /opt/ros/iron/setup.bash
ros2 run demo_nodes_cpp talker
```

2. In another terminal, source the setup file and run a Python listener:

```
source /opt/ros/iron/setup.bash
ros2 run demo_nodes_py listener
```

You should see the talker publishing messages and the listener receiving those messages, confirming that both the C++ and Python APIs are functioning correctly.

Issues Faced and Solutions

Issue: Dependency Conflict during ROS 2 Installation

Solution: Ensure that systemd and udev-related packages are updated before installing ROS 2 to prevent conflicts and critical package removals. Refer to the ROS 2 issue tracker and Ubuntu bug reports for detailed information and updates.

Issue: ROS 2 Packages Not Found in Repository

Solution: Double-check the ROS 2 repository configuration and make sure the correct ROS 2 distribution (e.g., Iron) and Ubuntu version (e.g., 22.04) are specified. Update the apt cache and repository lists before attempting installation.

Issue: Locale Settings Causing Package Installation Errors

Solution: Verify that the locale settings are correctly configured to support UTF-8. Use the `locale` command to check current settings and update them as necessary before proceeding with ROS 2 installation.

Next Steps

With ROS 2 successfully installed and configured on our devices, the next steps involve integrating ROS 2 with our existing components and testing its functionality within our restaurant robot project. We will continue developing and refining the system to ensure seamless operation and effective communication with customers.

5.14 Setting up the Development Environment (1 - 5 May 2024)

5.14.1 Setting Up the Catkin Workspace

After installing ROS 2 Iron, setting up the colcon workspace is essential for organizing and building your ROS packages.

1. Navigate to home directory and create the colcon workspace:

```
cd ~
mkdir -p colcon_ws/src
cd colcon_ws
```

2. Build the workspace using colcon:

```
colcon build
```

Installing Visual Studio Code (Optional)

Visual Studio Code provides a convenient IDE for writing ROS scripts. Follow these steps to install and set up VS Code:

1. Install snapd (if not already installed):

```
sudo apt install snapd
```

2. Install Visual Studio Code:

```
sudo snap install --classic code
```

3. Install necessary extensions for Python and ROS by Microsoft within VS Code.

5.14.2 Setting Up Communication Between Computers

To establish communication between your computer and a Single Board Computer (SBC), update your .bashrc file with the appropriate ROS environment variables:

1. Add the following lines at the end of /.bashrc, replacing ‘masterip.local‘ with the actual IP address of the ROS master:

```
echo 'export ROS_DOMAIN_ID=0' >> ~/.bashrc
echo 'export ROS_MASTER_URI=http://masterip.local:11311' >> ~/.bashrc
source ~/.bashrc
```

5.14.3 Setting up the GitHub Repository

GitHub Setup

Setting up the repository involves creating a GitHub account, cloning the project template, and configuring project-specific details.

1. Create a GitHub account and clone the project template:

```
git clone https://github.com/YasiruDEX/LUNA_workspace_template
```

2. Customize project details:
   - Inside `launch/rsp.launch.py`, set the package path.
   - Inside `package.xml`, update package name, version, description, and maintainer details.
   - Inside `CMakeLists.txt`, define project details.

5.14.4 Issues Faced and Solutions

Issue: Dependency Conflict during Colcon Build

Solution: Ensure all ROS 2 dependencies and system packages are updated and compatible with ROS 2 Iron. Resolve any conflicts by updating or reinstalling conflicting packages.

Issue: VS Code Extensions Not Working Properly

Solution: Reinstall the Python and ROS extensions within Visual Studio Code. Ensure they are compatible with the ROS 2 Iron version installed.

Issue: Incorrect ROS Master URI Configuration

Solution: Double-check the ROS Master URI configuration in `/.bashrc` to ensure it points correctly to the ROS master's IP address. Source `/.bashrc` after making changes.

Issue: Git Clone Authentication Errors

Solution: Ensure correct GitHub credentials are used for cloning. Verify SSH keys or use HTTPS with username and password for authentication.

5.14.5 Next Steps

With the Catkin workspace and GitHub repository set up, the next steps involve developing and integrating specific ROS nodes and functionalities for the restaurant robot project. Testing and iterating on these components will ensure robust performance and functionality.

5.15 Creation of the URDF 3D Model (5 - 9 May 2024)

During the creation of the URDF 3D model for the robot, several key components were implemented, including materials setup, macros for inertials, base link and footprint, chassis, wheels, and importing the control Xacro file.

Issues Faced

1. **Understanding URDF Structure:** Initially, grasping the hierarchical structure and syntax of URDF files posed a challenge, especially in defining links, joints, and their interconnections.
2. **Material Definitions:** Configuring and applying materials correctly using URDF specifications required referring to ROS and URDF documentation extensively.
3. **Integration of Xacro Macros:** Incorporating and utilizing Xacro macros for inertial properties and other repetitive elements necessitated careful attention to parameter passing and syntax.
4. **Joint and Link Definitions:** Defining various joints (fixed, continuous) and ensuring correct link definitions (visual, collision) for each component like wheels and chassis posed specific challenges.
5. **Xacro File Inclusion:** Ensuring proper inclusion and referencing of the control Xacro file within the main URDF XML file required understanding file paths and dependencies.

Solutions Implemented

1. **Consulting Documentation:** Referencing official ROS and URDF documentation helped clarify URDF structure and syntax nuances, ensuring accurate file construction.

2. **Validation and Testing:** Validating URDF files using tools like ‘urdf_tutorial‘ and ‘rviz‘ aided in identifying and rectifying syntax errors and inconsistencies.
3. **Material Definition Review:** Reviewing and adjusting material definitions based on ROS and Gazebo standards ensured consistent visual representation across simulation platforms.
4. **Parameterized Macros:** Parameterizing Xacro macros effectively streamlined inertial property definitions for various geometric shapes, enhancing code reusability.
5. **File Management Practices:** Adopting structured file management practices and using relative paths facilitated seamless integration of the control Xacro file, minimizing path-related errors.

Next Steps

With the URDF 3D model framework established, the next steps involve integrating additional components such as sensors, actuators, and refining joint configurations. Testing and simulation validation will be crucial to ensure accurate representation and functionality in the robotic platform.

5.16 Setting up ROS 2 Control for Robot Teleoperation (9 - 13 May 2024)

Setting up ROS 2 control for teleoperating the robot involved creating configuration files, integrating control parameters, and launching the system.

Issues Faced

1. **Understanding ROS 2 Control Configuration:** Initially, comprehending the structure and configuration requirements of ROS 2 control (including system setup and Gazebo integration) posed challenges due to complex plugin dependencies.
2. **Parameter File Configuration:** Creating and configuring the YAML parameter file for the ROS 2 controller manager and specific controllers (like ‘DiffDriveController‘ and ‘JointStateBroadcaster‘) required precise syntax and understanding of ROS 2 control parameters.
3. **Integration with Launch Files:** Integrating ROS 2 control information into the launch file (‘rsp.launch.py‘) while ensuring compatibility with other ROS nodes and correct parameter passing was non-trivial.
4. **Build and Installation Issues:** Ensuring proper build and installation (‘colcon build‘) of the workspace after configuring ROS 2 control plugins and launching nodes sometimes led to dependency conflicts or missing packages.

Solutions Implemented

1. **Documentation and Tutorials:** Referencing ROS 2 control documentation and tutorials provided clarity on setting up system configurations and understanding plugin dependencies.
2. **Validation and Testing:** Validating parameter files (‘my_parameters.yaml‘) using ROS tools like ‘ros2 launch‘ and ‘rqt‘ helped in detecting and rectifying syntax errors and ensuring correct parameter passing.
3. **Incremental Development:** Adopting an incremental development approach for integrating ROS 2 control information into the launch file (‘rsp.launch.py‘) allowed for step-by-step debugging and resolution of compatibility issues.
4. **Dependency Management:** Implementing robust dependency management practices and using virtual environments ensured smooth ‘colcon build‘ processes and resolved missing package issues.

Next Steps

Moving forward, the focus will be on testing the teleoperation functionality using RVIZ2 and verifying the integration of ROS 2 control with the robot's simulation environment in Gazebo. Further enhancements may include sensor integration and refining controller behaviors for improved robotic performance.

Setting Up Teleoperation with Game Controller (13 - 16 May 2024)

Setting up teleoperation with a game controller involved several steps to ensure compatibility, configuration, and testing with the robot.

Issues Faced

1. **Gamepad Compatibility:** Verifying if the gamepad was compatible with Linux and ROS proved challenging. Different drivers and configurations were needed based on the type of gamepad.
2. **Joystick Package Installation:** Installing the necessary ROS packages (‘joystick‘, ‘teleop_twist_joy‘) and ensuring they were correctly integrated with ROS 2 Iron posed initial dependency and compatibility issues.
3. **Parameter Tuning:** Configuring parameters (‘my_parameters.yaml‘) for the gamepad, such as deadzone, scaling factors for linear and angular motion, and button mappings, required iterative adjustments to achieve smooth teleoperation.
4. **Remapping Twist Commands:** Correctly remapping the ‘cmd_vel‘ topic to ‘diff_drive_controller cmd_vel_unstamped‘ in the launch file (‘teleop.launch.py‘) to ensure compatibility with the robot's control system.
5. **Testing and Validation:** Ensuring proper functionality and real-time feedback in RViz during teleoperation sessions involved troubleshooting initial setup issues and ensuring all components were properly initialized.

Solutions Implemented

1. **Driver Compatibility Checks:** Researching and verifying Linux kernel support for the specific gamepad model and ensuring the necessary drivers (‘joy‘ package) were installed and configured correctly resolved compatibility issues.
2. **Package Installation via ROS 2 Tools:** Using ‘apt‘ to install ROS 2 packages (‘ros2-iron-teleop-twist-joy‘) and checking compatibility with the current ROS 2 Iron release version resolved initial dependency conflicts.
3. **Parameter Fine-Tuning:** Iteratively adjusting parameters in ‘my_parameters.yaml‘ using feedback from test runs (‘ros2 run joy joy_node‘ and ‘ros2 topic echo /joy‘) ensured optimal responsiveness and control precision.
4. **Launch File Configuration:** Correcting the remapping of twist commands in ‘teleop.launch.py‘ ensured that teleoperation commands were correctly directed to the robot's differential drive controller, facilitating smooth control.
5. **Integration Testing with RViz:** Launching RViz with a configured visualization file (‘rviz2‘) and validating real-time feedback from the robot's sensors (camera, LiDAR) ensured accurate teleoperation performance and visual verification.

Next Steps

Moving forward, the focus will be on refining joystick configurations based on user feedback, implementing safety controls, and exploring additional control modes or features to enhance the teleoperation experience.

5.17 Setting up Gazebo for Simulation (16 - 23 May 2024)

Setting up Gazebo for simulation involved several steps to integrate it with ROS 2 and verify the system's functionality in a virtual environment.

Issues Faced

1. **Package Installation:** Installing Gazebo and ROS 2 control packages (‘ros2-iron-gazebo-ros2-control‘) initially led to dependency conflicts due to version compatibility issues between ROS 2 Iron and Gazebo.
2. **Launch File Configuration:** Creating the launch file (‘launch_sim.launch.py‘) to properly include and configure Gazebo alongside ROS 2 components required understanding the correct syntax and parameter passing for Gazebo integration.
3. **Gazebo Parameter Setup:** Configuring parameters in ‘my_parameters.yaml‘ specific to Gazebo, such as simulation time settings and environment variables, required detailed understanding of Gazebo's interaction with ROS 2.
4. **Environment Loading:** Loading a custom world file (‘¡location_to_world_file¿‘) in Gazebo initially failed due to incorrect path specification and missing world model definitions.
5. **Simulation Validation:** Ensuring that the robot model appeared correctly in the Gazebo environment and interacted with simulated sensors like LiDAR and camera feeds was crucial but initially showed discrepancies in model spawning and sensor data integration.

Solutions Implemented

1. **Dependency Resolution:** Resolving ROS 2 and Gazebo version conflicts by updating ROS 2 packages (‘ros2-iron-gazebo-ros2-control‘) and ensuring compatibility with the installed Gazebo version (‘sudo apt install gazebo‘) resolved initial installation issues.
2. **Launch File Debugging:** Debugging the ‘launch_sim.launch.py‘ file by referencing ROS 2 launch file examples and adjusting parameters such as ‘extra_gazebo_args‘ correctly passed to Gazebo ensured proper integration and functionality.
3. **Parameter Fine-Tuning:** Iteratively adjusting parameters in ‘my_parameters.yaml‘, such as ‘publish_rate‘ and ‘use_sim_time‘, based on Gazebo's requirements and ROS 2 control settings, achieved synchronized simulation timing and accurate sensor feedback.
4. **Environment Path Correction:** Correcting the path to the custom world file ‘¡location_to_world_file¿‘ and ensuring all necessary models and world definitions were correctly referenced in Gazebo resolved issues with environment loading and robot model placement.
5. **Validation and Testing:** Validating the simulation setup by launching ‘ros2 launch package_name launch_sim.launch.py -use_sim_time:=true environment:=location_to_world_file¿‘ successfully displayed the robot model in Gazebo with accurate sensor feedback, confirming proper integration and functionality.

Next Steps

Moving forward, the focus will be on refining simulation parameters, conducting comprehensive testing scenarios in Gazebo to validate robot behavior, and integrating more complex environments and scenarios for advanced testing and tuning.

5.18 Setting up Kinect V2 Sensor in Gazebo (23 - 28 May 2024)

To simulate the Kinect V2 sensor in Gazebo, I followed these steps to create a URDF model and configure Gazebo plugins.

Issues Faced

1. **URDF Model Definition:** Creating the URDF model (‘robot.kinect.xacro‘) for the Kinect V2 sensor involved understanding the correct XML syntax for defining links, joints, and gazebo plugins. Initially, incorrect placement of tags and parameters led to parsing errors in the URDF model.
2. **Gazebo Plugin Configuration:** Configuring Gazebo plugins (‘libgazebo_ros_camera.so‘ and ‘libgazebo_ros_depth_camera.so‘) to simulate the Kinect’s RGB camera and depth sensor required detailed specification of parameters such as update rates, field of view, and camera formats. Incorrect plugin configurations resulted in the plugins not loading or producing incorrect sensor data.
3. **Integration with Robot Model:** Integrating the Kinect URDF model with the existing robot model (‘base_link‘) in Gazebo was challenging due to discrepancies in joint positions and orientations. This led to misalignment of the Kinect sensor relative to the robot’s frame of reference.
4. **Visualization in RVIZ:** Visualizing the Kinect sensor data, especially depth images and RGB camera feeds, in RVIZ for real-time feedback initially failed due to incorrect topic subscriptions and improper configuration of RVIZ display settings.

Solutions Implemented

1. **URDF Model Debugging:** Debugging the ‘robot.kinect.xacro‘ file involved thorough validation of XML syntax, ensuring correct nesting of tags (link, joint, gazebo), and validating dimensions and positions of visual elements (visual).
2. **Plugin Parameter Adjustment:** Adjusting parameters in the Gazebo plugin configuration for both the RGB camera and depth sensor plugins, including ‘update_rate‘, ‘fov‘, and ‘clip‘ parameters, based on Kinect V2 specifications and Gazebo requirements resolved issues with plugin initialization and data accuracy.
3. **Joint Alignment Correction:** Correcting the position and orientation of the fixed joint (‘laser_joint‘) between ‘base_link‘ and ‘laser_frame‘ in the URDF model ensured proper alignment of the Kinect sensor within the robot’s coordinate system, resolving misalignment issues.
4. **RVIZ Configuration:** Configuring RVIZ to correctly display Kinect sensor data involved setting up subscriptions to topics such as ‘kinect/depth/image_raw‘ and ‘kinect/rgb/image_raw‘ and adjusting display settings for image formats (‘R8G8B8‘), image resolution (640x480), and camera frame (‘kinect_frame‘).

Next Steps

Moving forward, We plan to conduct extensive testing of the simulated Kinect sensor in Gazebo, validate sensor data accuracy under different environmental conditions, and integrate real-time processing algorithms to leverage the sensor data for robot perception tasks.

5.19 Conversion of Kinect Sensor Depth Data to Fake-Lidar Data (28 May - 2 June 2024)

To convert Kinect depth image data into laser scan data compatible with SLAM toolbox and AMCL for mapping and localization in ROS 2, I followed these steps:

Issues Faced

1. **Configuration of depthimage_to_laserscan:** Modifying the launch file (‘depthimage_to_laserscan-launch.py‘) to correctly configure the depthimage_to_laserscan package was challenging initially due to incorrect remapping of input topics (‘/kinect_depth/depth/image_raw‘ and ‘/kinect_depth/camera_info‘) and parameter settings (‘output_frame‘).

2. **Integration with SLAM Toolbox and AMCL:** Ensuring compatibility of the converted laser scan data with SLAM Toolbox and AMCL required precise parameter settings (‘output_frame‘) and remapping of topics to align with the robot's frame of reference (‘laser_frame‘). Incorrect configurations led to mismatched sensor data in the mapping and localization processes.

3. **Visualization in RVIZ:** Visualizing the converted laser scan data in RVIZ initially failed due to improper setup of the LaserScan display and incorrect topic subscriptions (‘/scan‘). This hindered the ability to validate the accuracy and consistency of the converted data visually.

Solutions Implemented

1. **Launch File Debugging:** Debugging the ‘depthimage_to_laserscan-launch.py‘ involved verifying the correct specification of parameters such as ‘output_frame‘ set to ‘laser_frame‘ and ensuring proper remapping of input topics (‘/kinect_depth/depth/image_raw‘ and ‘/kinect_depth/camera_info‘) to match the Kinect sensor data.

2. **Parameter Adjustment:** Adjusting parameters within the launch file to specify the output frame (‘laser_frame‘) correctly aligned the converted laser scan data with the robot's coordinate system, resolving discrepancies in sensor data integration with SLAM and AMCL.

3. **RVIZ Configuration:** Configuring RVIZ to correctly visualize the converted laser scan data involved setting up the LaserScan display with the correct topic (‘/scan‘) and adjusting display settings for optimal visualization of laser scan readings from the Kinect sensor.

Next Steps

Moving forward, We plan to conduct extensive testing and validation of the Kinect sensor simulation with fake-lidar data conversion in Gazebo. This includes testing under various environmental conditions to ensure robustness and accuracy in mapping and localization tasks using SLAM Toolbox and AMCL in ROS 2.

5.20 Using SLAM Toolbox for Mapping the Environment (2 - 9 June 2024)

To map the environment using SLAM Toolbox in ROS 2, I followed these steps:

Issues Faced

1. **SLAM Toolbox Installation:** Initially, installing the SLAM Toolbox package (‘ros-iron-slam-toolbox‘) using ‘sudo apt install‘ command faced issues due to missing dependencies and conflicting versions, which required resolving through package management and updating repositories.

2. **Launch File Configuration:** Modifying the original launch file (‘online_async_launch.py‘) to correctly specify parameters such as ‘use_sim_time‘ and ‘slam_params_file‘ involved understanding the correct directory paths and ensuring proper YAML configuration for SLAM Toolbox.

3. **Integration with Gazebo Simulation:** Ensuring seamless integration of SLAM Toolbox with Gazebo simulation was challenging initially due to discrepancies in time synchronization

(‘use_sim_time‘) and misalignment of parameters between the simulation environment and SLAM Toolbox configuration.

4. **Visualization in RViz:** Visualizing the mapped environment in RViz posed challenges in setting up the correct topic subscriptions (‘/map‘) and configuring RViz tools for optimal display of the generated map, requiring adjustments in topic settings and base frame alignment.

Solutions Implemented

1. **Dependency Management:** Resolving package dependencies and updating ROS repositories enabled successful installation of SLAM Toolbox using ‘sudo apt install ros-iron-slam-toolbox‘, ensuring compatibility and version consistency across packages.

2. **Launch File Debugging:** Debugging the ‘online_async_launch.py‘ involved validating directory paths using ‘os.path.join‘ for ‘slam_params_file‘, correcting YAML parameter configurations, and ensuring ‘use_sim_time‘ was set to ‘true‘ for proper time synchronization in Gazebo.

3. **Environment Integration:** Aligning SLAM Toolbox parameters with the Gazebo simulation environment required adjusting parameter settings (‘slam_params_file‘) and verifying ROS 2 launch configurations to establish reliable communication and data exchange between simulation components.

4. **RViz Configuration:** Configuring RViz to visualize the mapped environment involved setting up the Map display tool with ‘/map‘ topic subscription and aligning the base frame to ‘map‘, enabling accurate visualization of the robot’s position and environment mapping in real-time.

Next Steps

Moving forward, We plan to conduct extensive testing and validation of the mapped environment using SLAM Toolbox in ROS 2. This includes refining parameter settings, optimizing RViz visualization for enhanced mapping accuracy, and integrating teleoperation support for comprehensive environment exploration and mapping.

5.21 Setting up Automatic Localization using AMCL in ROS 2 (9 - 14 June 2024)

To enable automatic localization using Adaptive Monte Carlo Localization (AMCL) in ROS 2, I followed these steps:

Issues Faced

1. **Package Installation:** Installing necessary packages (‘ros-iron-navigation2‘, ‘ros-iron-bringup‘, ‘ros-iron-turtlebot3‘) initially faced dependency issues and required resolving through package management and updating repositories.

2. **Map Hosting with Map Server:** Hosting the saved map (‘my_map_save.yaml‘) using the map server (‘ros2 run nav2 map_server map_server‘) required configuring the correct YAML file path and ensuring ‘use_sim_time‘ was set to ‘true‘ for time synchronization.

3. **Starting AMCL Node:** Initiating the AMCL node (‘ros2 run nav2_amcl‘) for robot localization involved configuring node parameters and validating ROS 2 launch configurations to establish communication with the hosted map and sensor data.

4. **RViz Visualization:** Configuring RViz (‘rviz2‘) to visualize the map and localize the robot posed challenges in setting the correct fixed frame (‘/map‘), configuring the map display topic (‘/map‘), and utilizing tools like ‘2D Pose Estimate‘ for accurate robot localization.

Solutions Implemented

1. **Dependency Management:** Resolving package dependencies and updating ROS repositories enabled successful installation of navigation and robot control packages using ‘sudo apt install‘.

2. **Map Server Setup:** Configuring the map server to host the saved map (‘my_map_save.yaml‘) involved specifying the correct YAML file path and setting ‘use_sim_time‘ to ‘true‘ for synchronization with simulation time.

3. **AMCL Node Configuration:** Configuring the AMCL node (‘nav2_amcl‘) included adjusting parameters for map resolution, sensor data input, and particle filter settings to optimize robot localization accuracy.

4. **RViz Configuration:** Setting up RViz for visualization of the map and robot localization required configuring the ‘Fixed Frame‘ to ‘/map‘, adding a ‘Map‘ display with the topic set to ‘/map‘, and utilizing the ‘2D Pose Estimate‘ tool to localize the robot accurately within the map.

5.22 Setting up Navigation2 (Nav2) for Autonomous Navigation in ROS 2 (14 - 19 June 2024)

To enable autonomous navigation using Navigation2 (Nav2) in ROS 2, I followed these steps:

Issues Faced

1. **Package Installation:** Installing required packages (‘ros-iron-navigation2‘, ‘ros-iron-bringup‘, ‘ros-iron-turtlebot3‘) initially encountered dependency issues, which were resolved by updating repositories and managing package dependencies.

2. **Launch File Creation:** Creating the Nav2 bringup launch file (‘nav2_bringup_launch.py‘) required configuring launch arguments (‘use_sim_time‘, ‘map‘, ‘params‘) and including launch descriptions for map loading and parameter setup.

3. **Parameters Configuration:** Setting up the ‘nav2_params.yaml‘ file involved defining parameters for AMCL, lifecycle management, map server, planner server, controller server, behavior tree navigator, and waypoint follower nodes to ensure proper navigation stack initialization.

4. **RViz Configuration:** Configuring RViz (‘rviz2‘) for visualization of map, robot pose, planned path, and robot model required setting appropriate topics (‘/map‘, ‘/amcl_pose‘, ‘/plan‘) and adding displays accordingly.

5. **Sending Navigation Goals:** Implementing a Python script to programmatically send navigation goals encountered minor issues in topic setup and message publishing, which were resolved through ROS message debugging and node lifecycle management.

Solutions Implemented

1. **Dependency Management:** Resolving package dependencies and updating ROS repositories enabled successful installation of Navigation2 and related packages using ‘sudo apt install‘.

2. **Launch File Setup:** Configuring launch arguments and including necessary launch descriptions facilitated the creation of the Nav2 bringup launch file (‘nav2_bringup_launch.py‘) to initialize the navigation stack.

3. **Parameters Definition:** Defining parameters in the ‘nav2_params.yaml‘ file for AMCL, lifecycle management, and navigation nodes ensured proper configuration and initialization of the navigation stack.

4. **RViz Setup:** Configuring RViz displays for map visualization, robot pose estimation, path planning visualization, and robot model display provided a comprehensive visualization tool for monitoring navigation behavior.

5. **Goal Sending Script:** Developing a Python script to send navigation goals programmatically enabled testing of navigation capabilities and goal reaching behavior of the robot within the mapped environment.