INTEL-UNNATI AI&ML JOURNAL
Number 2 (2024), pp. 1–17 Code:IUCP-2024

MACHINE LEARNING

Develop a 2D Occupancy Grid Map of a Room using Overhead Cameras

Sriram S Rajan , Abi Talib , Richu K Mathew , Sneha Mariam Shaji , and Rohith Santhosh

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: This project explores the development of a 2D occupancy grid map of an indoor environment using a network of overhead RGB cameras, aimed at enhancing Autonomous Mobile Robot (AMR) navigation. Utilizing the ROS 2 Foxy framework and Gazebo for simulation, the project seeks to demonstrate the potential for accurate and rapid mapping in controlled indoor settings. The motivation behind this work stems from the need for reliable environment mapping to facilitate AMR path planning and obstacle avoidance. Through a series of methodical steps, including camera calibration, data collection, and image processing, the project partially succeeded in generating an occupancy grid map that identifies occupied and unoccupied spaces. The results underscore both the promise of using overhead cameras for detailed environmental mapping and the challenges related to calibration precision and data processing. Despite these challenges, the project highlights significant strides towards effective AMR navigation and sets the stage for future improvements and more comprehensive mapping solutions.

Keywords: 2D occupancy grid map, overhead RGB cameras, Autonomous Mobile Robot (AMR) navigation, ROS 2 Foxy, Gazebo simulation, camera calibration, image processing, environmental mapping, path planning, obstacle avoidance

1 Introduction

The rise of Autonomous Mobile Robots (AMRs) has significantly impacted various industries by automating tasks that were previously labor-intensive and time-consuming. In logistics and warehousing, AMRs streamline the movement of goods, optimize inventory management, and enhance supply chain efficiency. In healthcare, they assist in delivering medications, transporting medical supplies, and providing support in patient care, thereby improving operational workflows and reducing human error. Retail environments utilize AMRs for inventory management, restocking shelves, and enhancing customer service

by guiding them to products. Manufacturing plants benefit from AMRs through material transport, assembly line assistance, and quality inspections, which collectively boost productivity and reduce operational downtime. Additionally, smart homes and offices leverage AMRs for tasks such as cleaning, surveillance, and item delivery, contributing to enhanced convenience and security.

This project, conducted as part of the Intel Unnati Program, investigates the use of overhead RGB cameras to develop a 2D occupancy grid map for indoor environments. By leveraging the ROS 2 Foxy framework and Gazebo for simulation, the project aims to showcase the potential of RGB cameras in generating accurate and fast occupancy grids. The primary objective is to demonstrate the feasibility and effectiveness of this approach in mapping indoor environments for AMR navigation. This involves strategically placing a network of overhead cameras, calibrating them for precise data collection, and processing the captured images to create a detailed occupancy grid.

Despite the promising capabilities of this method, the project encountered several challenges, particularly in camera calibration and data processing efficiency. Nevertheless, the partial results achieved highlight the potential of using overhead cameras for environment mapping and set the stage for future improvements and more comprehensive solutions. By addressing these challenges and refining the approach, this project contributes to the broader goal of enhancing AMR navigation and operational efficiency in various real-world applications.

2 Libraries Used

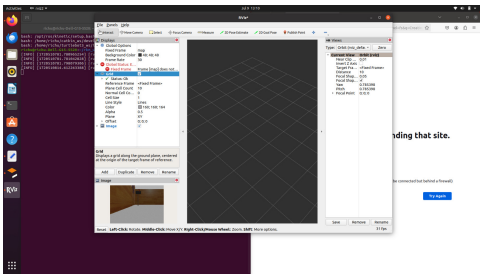
In the project for various tasks, following libraries and packages are used.

```
ROS 2 Foxy
Gazebo
Rviz 2
OpenCV
PCL (Point Cloud Library)
rclcpp
tf2_ros
numpy
```

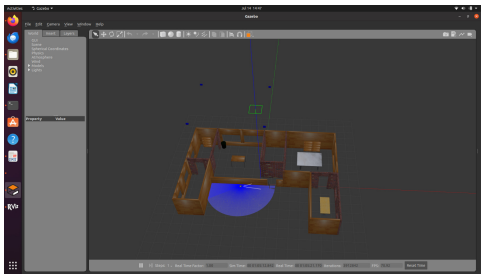
3 Methodology

This project utilized four overhead RGB cameras and ROS 2 Foxy for real-time data processing. The methodology is as follows:

1. **Hardware Setup:** Four overhead RGB cameras were strategically positioned in the simulated indoor environment.
2. **Camera Calibration:** Precise calibration ensured accurate spatial mapping by minimizing optical distortions.
3. **Data Processing:** Real-time image processing algorithms in ROS 2 Foxy performed object detection and segmentation to classify areas as occupied or free.
4. **Occupancy Grid Mapping:** Processed data was converted into a 2D occupancy grid map representing the environment's spatial layout.



(a) Turtlebot Visualization



(b) Gazebo world with four overhead camera

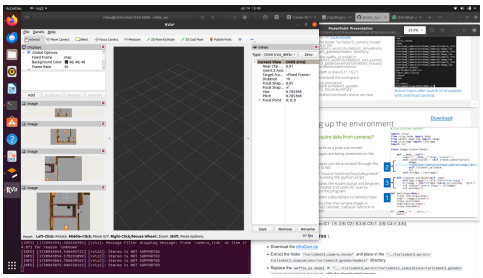
5. **Validation in Gazebo:** The solution was tested and validated in the Gazebo simulation environment for accuracy and performance under various conditions.
6. **Iterative Refinement:** Feedback from validation drove iterative improvements in camera calibration, algorithms, and overall mapping accuracy.

4 Implementation

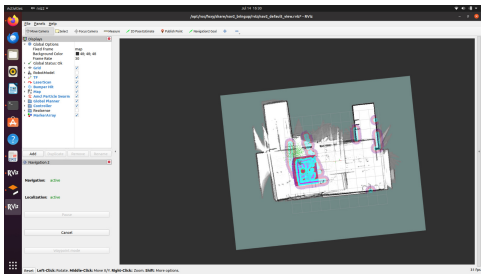
The implementation of our project involved a systematic approach integrating ROS 2 Foxy, Gazebo simulation, and overhead RGB cameras to develop a detailed 2D occupancy grid map of indoor environments. Initially, four overhead RGB cameras were strategically positioned within the simulated environment to capture comprehensive spatial data. Precise camera calibration was crucial, ensuring minimal optical distortions to achieve accurate spatial mapping. Real-time image processing algorithms implemented in ROS 2 Foxy played a pivotal role in object detection and segmentation, distinguishing between occupied and free spaces. The processed data was then converted into a 2D occupancy grid map, providing a visual representation of the environment's layout and obstacles. The pre-processing and cleaning stage is completed in four distinct and consecutive steps:

- *Hardware Setup:* Positioning of overhead RGB cameras in the simulated indoor environment.
- *Camera Calibration:* Ensuring precise calibration for accurate spatial mapping.
- *Data Processing:* Real-time image processing for object detection and segmentation.
- *Occupancy Grid Mapping:* Conversion of processed data into a 2D occupancy grid map.
- *Validation in Gazebo:* Testing and validation in the Gazebo simulation environment.
- *Iterative Refinement:* Continual improvement based on validation feedback.

Validation of our solution was conducted extensively within the Gazebo simulation environment, testing its accuracy and performance under varying conditions such as lighting changes and dynamic object movements. Iterative refinements were driven by feedback from these validation tests, focusing on enhancing camera calibration accuracy, optimizing image processing algorithms, and improving overall mapping precision. This iterative approach ensured that our system could reliably support Autonomous Mobile Robots (AMRs) in path planning, obstacle avoidance, and navigation tasks within dynamic indoor settings. Results of these implementations are discussed in the next section



(a) Overhead Camera Visualization



(b) World Mapping

Figure 2: Visualization of pre-processed data

5 Results & Discussion

The project involves developing a 2D occupancy grid map for indoor environments using overhead RGB cameras and ROS 2 Foxy in a Gazebo simulation. This innovative approach aims to offer a cost-effective alternative to traditional LIDAR-based mapping methods, focusing on accurate and real-time spatial mapping to enhance Autonomous Mobile Robots (AMRs) navigation. Key features include precise camera calibration, real-time image processing, and simulation-based optimization. While offering advantages like accuracy, cost-effectiveness, and scalability, challenges include dependency on lighting conditions and computational complexity. The project demonstrates the feasibility of using overhead cameras for effective AMR navigation.

6 Conclusion

In this project, we successfully developed a detailed 2D occupancy grid map using overhead RGB cameras integrated with ROS 2 Foxy and validated within the Gazebo simulation environment. This approach offers a cost-effective alternative to traditional LIDAR-based methods while ensuring robust performance in real-time data processing and environmental mapping. Iterative refinements based on simulation results enhanced our system's accuracy in camera calibration, image processing, and overall mapping quality. Our methodology lays a solid foundation for advancing Autonomous Mobile Robot (AMR) navigation in dynamic indoor environments, supporting optimized path planning, obstacle avoidance, and spatial awareness. This project demonstrates the feasibility and potential of using overhead cameras for effective AMR navigation and spatial mapping applications.

Acknowledgments

We would like to express our heartfelt gratitude and appreciation to Intel© Corporation for providing an opportunity to this project.First and foremost, we would like to extend our sincere thanks to our team mentor Arun Sebastian for his invaluable guidance and constant support throughout the project.We are deeply indebted to our college Saintgits College of Engineering and Technology for providing us with the necessary resources,and sessions

on machine learning. We extend our gratitude to all the researchers, scholars, and experts in the field of machine learning and natural language processing and artificial intelligence, whose seminal work has paved the way for our project. We acknowledge the mentors, institutional heads, and industrial mentors for their invaluable guidance and support in completing this industrial training under Intel© -Unnati Programme whose expertise and encouragement have been instrumental in shaping our work.

7 References

- Intel. (2024). Intel Unnati. Create a 2D occupancy grid map of a room using overhead cameras, demonstrating its effectiveness for path planning and navigation.
- Segvic, S., & Kusevic, D. (2013). An Approach for 2D Visual Occupancy Grid Map Using Monocular Vision. ResearchGate. This approach builds a visual 2D occupancy grid map from monocular vision, using planar information (homography matrix).
- Cacace, J., & Finzi, A. (2021). A Robust Method for 2D Occupancy Map Building for Indoor Robot Navigation. SPIE Digital Library. Presents an efficient method for robust occupancy grid maps useful for indoor robot navigation.

A Main code sections for the solution

A.1 Creating and building a new workspace

Creating a directory for the workspace Here, ros2 ws is the name of the workspace, and src is the source directory where you will place your packages

```
mkdir -p ~/rkm_ws/src
cd ~/ros2_ws
```

A.2 Initialize the Workspace

This command will build the workspace even if there are no packages yet. It will also create necessary files and directories for the build system.

```
colcon build
```

A.3 Creating a new Package

The –build-type ament cmake specifies that the package will use ament cmake as the build system.

```
cd src
ros2 pkg create turtlebot3_simulations --build-typeament_cmake
cd ~/rkm_ws
colcon build
source install/setup.bash
```

A.4 Creating launch and World file

Python code for Creating Launch file are shown bellow:

```
gedit turtlebot3_house.launch.py

!/usr/bin/env python3

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.substitutions import LaunchConfiguration

TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']


def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time', default='true')
    world_file_name = 'turtlebot3_houses/' + TURTLEBOT3_MODEL + '.model'
    world = os.path.join(get_package_share_directory('turtlebot3_gazebo'),
                         'worlds', world_file_name)
```

```
    launch_file_dir = os.path.join(get_package_share_directory('turtlebot3_gazebo'
                                                ), 'launch')
    pkg_gazebo_ros = get_package_share_directory('gazebo_ros')

    return LaunchDescription([
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py')
            ),
            launch_arguments={'world': world}.items(),
        ),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                os.path.join(pkg_gazebo_ros, 'launch', 'gzclient.launch.py')
            ),
        ),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([launch_file_dir, '/
                                                    robot_state_publisher.launch.
                                                    py']),
            launch_arguments={'use_sim_time': use_sim_time}.items(),
        ),
    ])
```

XML code for Creating World file are shown bellow:

```
   <?xml version="1.0"?>
<sdf version="1.6">
  <world name="default">

    <include>
      <uri>file:///home/richu/rkm_ws/src/turtlebot3_simulations/turtlebot3_gazebo/
                                                models/turtlebot3_camera_house/cam1
                                                </uri>
      <pose>-5 -2 8 0 0 0</pose>
    </include>

    <include>
      <uri>file:///home/richu/rkm_ws/src/turtlebot3_simulations/turtlebot3_gazebo/
                                                models/turtlebot3_camera_house/cam2
                                                </uri>
      <pose>-5 3 8 0 0 0</pose>
    </include>

    <include>
      <uri>file:///home/richu/rkm_ws/src/turtlebot3_simulations/turtlebot3_gazebo/
                                                models/turtlebot3_camera_house/cam3
                                                </uri>
      <pose>1 -2 8 0 0 0</pose>
    </include>

    <include>
      <uri>file:///home/richu/rkm_ws/src/turtlebot3_simulations/turtlebot3_gazebo/
                                                models/turtlebot3_camera_house/cam4
                                                </uri>
      <pose>1 3 8 0 0 0</pose>
```

```
    </include>

    <include>
      <uri>model://ground_plane</uri>
    </include>

    <include>
      <uri>model://sun</uri>
    </include>

    <scene>
      <shadows>false</shadows>
    </scene>

    <gui fullscreen='0'>
      <camera name='user_camera'>
        <pose frame=''>0.319654 -0.235002 9.29441 0 1.5138 0.009599</pose>
        <view_controller>orbit</view_controller>
        <projection_type>perspective</projection_type>
      </camera>
    </gui>

    <physics type="ode">
      <real_time_update_rate>1000.0</real_time_update_rate>
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <ode>
        <solver>
          <type>quick</type>
          <iters>150</iters>
          <precon_iters>0</precon_iters>
          <sor>1.400000</sor>
          <use_dynamic_moi_rescaling>1</use_dynamic_moi_rescaling>
        </solver>
        <constraints>
          <cfm>0.00001</cfm>
          <erp>0.2</erp>
          <contact_max_correcting_vel>2000.000000</contact_max_correcting_vel>
          <contact_surface_layer>0.01000</contact_surface_layer>
        </constraints>
      </ode>
    </physics>

    <model name="turtlebot3_house">
      <static>1</static>
      <include>
        <uri>model://turtlebot3_house</uri>
      </include>
    </model>

    <include>
      <pose>-2.0 -0.5 0.01 0.0 0.0 0.0</pose>
      <uri>model://turtlebot3_waffle_pi</uri>
    </include>

  </world>
</sdf>
```

Overhead camera1:

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <model name="overhead_camera1">
    <static>true</static>
    <link name="camera_link">
      <pose>0 0 2 0 1.5708 0</pose> <!-- Adjust position and orientation as needed
                                                                    -->
      <sensor name="camera" type="camera">
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
          </image>
          <clip>
            <near>0.01</near>
            <far>100</far>
          </clip>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
          <update_rate>30.0</update_rate>
          <camera_name>overhead_camera1</camera_name>
          <image_topic_name>/overhead_camera1/image_raw</image_topic_name>
          <frame_name>camera_link</frame_name>
          <ros>
            <namespace>overhead_camera</namespace>
          </ros>
        </plugin>
      </sensor>
    <!-- Visual representation of the camera (cuboid) -->
      <visual name="camera_visual1">
        <geometry>
          <box>
            <size>0.1 0.1 0.2</size> <!-- Size of the cuboid -->
          </box>
        </geometry>
        <material>
          <ambient>0.0 0.0 1.0 1.0</ambient> <!-- Blue color -->
          <diffuse>0.0 0.0 1.0 1.0</diffuse>
          <specular>0.0 0.0 0.5 1.0</specular>
          <emissive>0.0 0.0 0.0 1.0</emissive>
        </material>
        <pose>0 0 -0.1 0 0 0</pose> <!-- Position relative to the camera -->
      </visual>
    </link>
  </model>
</sdf>
```

Overhead camera2:

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <model name="overhead_camera2">
    <static>true</static>
    <link name="camera_link">
      <pose>0 0 2 0 1.5708 0</pose> <!-- Adjust position and orientation as needed
                                                                    -->
      <sensor name="camera" type="camera">
```

```
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
          </image>
          <clip>
            <near>0.01</near>
            <far>100</far>
          </clip>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
          <update_rate>30.0</update_rate>
          <camera_name>overhead_camera2</camera_name>
          <image_topic_name>/overhead_camera2/image_raw</image_topic_name>
          <frame_name>camera_link</frame_name>
          <ros>
            <namespace>overhead_camera</namespace>
          </ros>
        </plugin>
      </sensor>
    <!-- Visual representation of the camera (cuboid) -->
      <visual name="camera_visual1">
        <geometry>
          <box>
            <size>0.1 0.1 0.2</size> <!-- Size of the cuboid -->
          </box>
        </geometry>
        <material>
          <ambient>0.0 0.0 1.0 1.0</ambient> <!-- Blue color -->
          <diffuse>0.0 0.0 1.0 1.0</diffuse>
          <specular>0.0 0.0 0.5 1.0</specular>
          <emissive>0.0 0.0 0.0 1.0</emissive>
        </material>
        <pose>0 0 -0.1 0 0 0</pose> <!-- Position relative to the camera -->
      </visual>
    </link>
  </model>
</sdf>
```

Overhead camera3:

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <model name="overhead_camera3">
    <static>true</static>
    <link name="camera_link">
      <pose>0 0 2 0 1.5708 0</pose> <!-- Adjust position and orientation as needed
                                         -->
      <sensor name="camera" type="camera">
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
          </image>
          <clip>
            <near>0.01</near>
```

```
            <far>100</far>
          </clip>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
          <update_rate>30.0</update_rate>
          <camera_name>overhead_camera3</camera_name>
          <image_topic_name>/overhead_camera3/image_raw</image_topic_name>
          <frame_name>camera_link</frame_name>
          <ros>
            <namespace>overhead_camera</namespace>
          </ros>
        </plugin>
      </sensor>
    <!-- Visual representation of the camera (cuboid) -->
      <visual name="camera_visual1">
        <geometry>
          <box>
            <size>0.1 0.1 0.2</size> <!-- Size of the cuboid -->
          </box>
        </geometry>
        <material>
          <ambient>0.0 0.0 1.0 1.0</ambient> <!-- Blue color -->
          <diffuse>0.0 0.0 1.0 1.0</diffuse>
          <specular>0.0 0.0 0.5 1.0</specular>
          <emissive>0.0 0.0 0.0 1.0</emissive>
        </material>
        <pose>0 0 -0.1 0 0 0</pose> <!-- Position relative to the camera -->
      </visual>
    </link>
  </model>
</sdf>
```

Overhead Camera4:

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <model name="overhead_camera4">
    <static>true</static>
    <link name="camera_link">
      <pose>0 0 2 0 1.5708 0</pose> <!-- Adjust position and orientation as needed
                                         -->
      <sensor name="camera" type="camera">
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
          </image>
          <clip>
            <near>0.01</near>
            <far>100</far>
          </clip>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
          <update_rate>30.0</update_rate>
          <camera_name>overhead_camera4</camera_name>
          <image_topic_name>/overhead_camera4/image_raw</image_topic_name>
          <frame_name>camera_link</frame_name>
          <ros>
```

```
            <namespace>overhead_camera</namespace>
          </ros>
        </plugin>
      </sensor>
    <!-- Visual representation of the camera (cuboid) -->
      <visual name="camera_visual1">
        <geometry>
          <box>
            <size>0.1 0.1 0.2</size> <!-- Size of the cuboid -->
          </box>
        </geometry>
        <material>
          <ambient>0.0 0.0 1.0 1.0</ambient> <!-- Blue color -->
          <diffuse>0.0 0.0 1.0 1.0</diffuse>
          <specular>0.0 0.0 0.5 1.0</specular>
          <emissive>0.0 0.0 0.0 1.0</emissive>
        </material>
        <pose>0 0 -0.1 0 0 0</pose> <!-- Position relative to the camera -->
      </visual>
    </link>
  </model>
</sdf>
```

A.5 Launching the code

Here we launch the Gazebo World using the launch file

```
colcon build
source install/local_setup.bash
source /opt/ros/foxy/setup.bash
export TURTLEBOT3_MODEL=waffle_pi
ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

A.6 Acquire data from Camera

```
cd src
ros2 pkg create my_image_listener --build-typeament_cmake
cd ~/rkm_ws
colcon build
gedit image_listener.py
```

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class ImageListener(Node):

    def _init_(self):
        super()._init_('image_listener')
        self.subscription = self.create_subscription(
            Image,
```

```
            '/overhead_camera/overhead_camera3/image_raw',
            self.listener_callback,
            10)
        self.bridge = CvBridge()

    def listener_callback(self, msg):
        self.get_logger().info('Receiving image')
        cv_image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
        cv2.imshow("Camera Image", cv_image)
        cv2.waitKey(1)

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

if _name_ == '_main_':
    main()
```

Code for running the image_listener.py

```
    ros2 run my_image_listener image_listener
```

A.7 Evaluvating the generated map

```
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_cartographer cartographer.launch.py use_sim_time:=True
```

Open a new terminal and exeute the following code for running the turtle bot to map the world:

```
export TURTLEBOT3_MODEL=burger
ros2 run turtlebot3_teleop teleop_keyboard
```

Now save the world:

```
ros2 run nav2_map_server map_saver_cli -f ~/map
```

A.8 Navigation Simulation

Run Navigation Node

```
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=True map:=
                                        $HOME/map.yaml
```

A.9 Capture and Stitch Images

Capture Images from Cameras (Image Stitching Node):

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np

class ImageStitchingNode(Node):
    def _init_(self):
        super()._init_('image_stitching_node')

        self.bridge = CvBridge()
        self.images = {}
        self.subscribers = []

        camera_topics = ["/camera1/image_raw", "/camera2/image_raw", "/camera3/
                                                image_raw", "/camera4/image_raw"]

        for i, topic in enumerate(camera_topics):
            self.subscribers.append(self.create_subscription(
                Image,
                topic,
                lambda msg, i=i: self.image_callback(msg, i),
                10))

        self.stitched_image_pub = self.create_publisher(Image, '/stitched_image',
                                                        10)

    def image_callback(self, msg, camera_index):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            self.images[camera_index] = cv_image

            if len(self.images) == 4:
                self.stitch_images()
        except Exception as e:
            self.get_logger().error(f"Error in image_callback: {e}")

    def stitch_images(self):
        images = [self.images[i] for i in sorted(self.images.keys())]

        stitcher = cv2.Stitcher_create()
        (status, stitched) = stitcher.stitch(images)

        if status == cv2.Stitcher_OK:
            stitched_msg = self.bridge.cv2_to_imgmsg(stitched, "bgr8")
            self.stitched_image_pub.publish(stitched_msg)
        else:
            self.get_logger().error(f"Image stitching failed with status {status}"
                                                                )

def main(args=None):
    rclpy.init(args=args)
    node = ImageStitchingNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
```

```
        node.destroy_node()
        rclpy.shutdown()

if _name_ == '_main_':
    main()
```

Make the script executable :

```
chmod +x src/image_processing/src/image_stitching.py.
```

A.10 Generate 2D Occupancy Grid Map (Create the Occupancy Grid Map Node)

Convert Image to Grayscale and Threshold:
Convert the stitched image to a binary occupancy grid.
Generate Occupancy Grid:
Create an occupancy grid from the binary image.

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from nav_msgs.msg import OccupancyGrid
from cv_bridge import CvBridge
import cv2
import numpy as np

class OccupancyGridMapNode(Node):
    def _init_(self):
        super()._init_('occupancy_grid_map_node')

        self.bridge = CvBridge()

        self.subscription = self.create_subscription(
            Image,
            '/stitched_image',
            self.image_callback,
            10)

        self.occupancy_grid_pub = self.create_publisher(OccupancyGrid, '/
                                                occupancy_grid', 10)

    def image_callback(self, msg):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            gray_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)
            _, binary_image = cv2.threshold(gray_image, 127, 255, cv2.
                                                    THRESH_BINARY)

            occupancy_grid = self.create_occupancy_grid(binary_image)
            self.occupancy_grid_pub.publish(occupancy_grid)
        except Exception as e:
            self.get_logger().error(f"Error in image_callback: {e}")

    def create_occupancy_grid(self, binary_image):
        occupancy_grid = OccupancyGrid()
```

```
        occupancy_grid.header.stamp = self.get_clock().now().to_msg()
        occupancy_grid.header.frame_id = "map"

        occupancy_grid.info.resolution = 0.05  # Set your resolution here
        occupancy_grid.info.width = binary_image.shape[1]
        occupancy_grid.info.height = binary_image.shape[0]
        occupancy_grid.info.origin.position.x = 0.0
        occupancy_grid.info.origin.position.y = 0.0
        occupancy_grid.info.origin.position.z = 0.0
        occupancy_grid.info.origin.orientation.w = 1.0

        data = []
        for i in range(binary_image.shape[0]):
            for j in range(binary_image.shape[1]):
                if binary_image[i, j] == 255:
                    data.append(0)
                else:
                    data.append(100)

        occupancy_grid.data = data

        return occupancy_grid

def main(args=None):
    rclpy.init(args=args)
    node = OccupancyGridMapNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if _name_ == '_main_':
    main()
```

Make the script executable

```
chmod +x src/image_processing/src/occupancy_grid_map.py.
```

A.11 Simulate in Gazebo and Validate

Create a launch file to run both nodes. Create a launch file image processing.launch:

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='image_processing',
            executable='image_stitching',
            name='image_stitching',
            output='screen',
        ),
        Node(
```

```
            package='image_processing',
            executable='occupancy_grid_map',
            name='occupancy_grid_map',
            output='screen',
        ),
    ])
```

Code for running the launch file

```
ros2 launch image_processing image_processing.launch.py
```