
**A PROJECT REPORT ON**

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

**Submitted to:**
**INTEL Unnati**
**Industrial Training**
**Program 2024.**

## Submitted By:

**STUDENT NAME : SAMEER GHANEKAR**

**USN : SG21CSE120**

## Project Mentor:

Dr. SHIVKUMAR KAGI

# INTRODUCTION

In the realm of robotics and autonomous systems, TurtleBot 3 combined with Robot Operating System (ROS) offers a powerful platform for developing and deploying technologies like 2D Occupancy Grid Mapping using overhead cameras. TurtleBot 3, a popular mobile robot platform, is known for its versatility and ease of integration with ROS, which provides a robust framework for managing hardware abstraction, low-level device control, message passing, and more.

Occupancy Grid Mapping is a vital technique in robotics for creating detailed representations of environments based on sensor data. By utilizing overhead cameras mounted on TurtleBot 3 and the capabilities of ROS 2, robots can perceive their surroundings with high fidelity, enabling tasks such as obstacle avoidance, path planning, and localization.

The goal of creating a 2D Occupancy Grid Map is to represent the environment as a grid where each cell indicates the probability of being occupied by an obstacle. This information is crucial for autonomous navigation and path planning tasks. Using ROS2, the system captures data from overhead cameras, processes it to detect obstacles, and then updates the occupancy grid map in real-time.

# DESCRIPTION

## Overhead Cameras for Mapping

Overhead cameras provide a top-down view of the room, offering a comprehensive perspective that facilitates accurate mapping. In the context of ROS 2 and TurtleBot 3, the process typically involves:

1. **Sensor Integration**: Mounting and configuring overhead cameras on TurtleBot 3 to capture images of the environment.
2. **Image Processing**: Employing ROS 2 nodes and libraries like OpenCV for real-time image processing. This step involves extracting features such as edges, contours, or key points that signify obstacles, boundaries, or free space.
3. **Occupancy Grid Mapping Algorithm**: Implementing ROS 2-based algorithms to convert processed image data into a 2D grid map. Each grid cell represents the probability or certainty of occupancy, updated dynamically based on sensor observations.
4. **Mapping Update and Fusion**: Continuously updating the occupancy grid map as TurtleBot 3 navigates through the environment. Fusion techniques may be used to integrate data from multiple sensors (e.g., lidar, IMU) to enhance map accuracy and reliability.

# METHODOLOGY

Developing a 2D occupancy grid map with ROS2 and TurtleBot using overhead cameras follows a systematic methodology for accurate indoor mapping and navigation. Initially, setup involves installing ROS2 on both the development machine and TurtleBot, ensuring seamless communication. Concurrently, overhead cameras are calibrated to provide a distortion-free top-down view of the environment, essential for capturing detailed spatial data.

Once configured, the process shifts to data acquisition and pre-processing. Activated overhead cameras continuously capture room images, which undergo rectification to correct distortions and segmentation to identify features like walls and open spaces. This pre-processing prepares the sensor data for conversion into an occupancy grid map using ROS2 tools. Each grid cell denotes occupancy status based on detected obstacles, updated probabilistically to reflect real-time environmental changes.

Following map generation, validation and refinement ensure accuracy and reliability. The map undergoes rigorous testing in both simulated and real-world scenarios, monitored through tools like RVIZ for real-time feedback. Iterative refinement includes adjusting camera parameters, fine-tuning algorithms, and optimizing computational resources to enhance mapping precision and efficiency. This structured approach enables TurtleBot to autonomously navigate and interact within indoor environments based on a reliable 2D occupancy grid map.

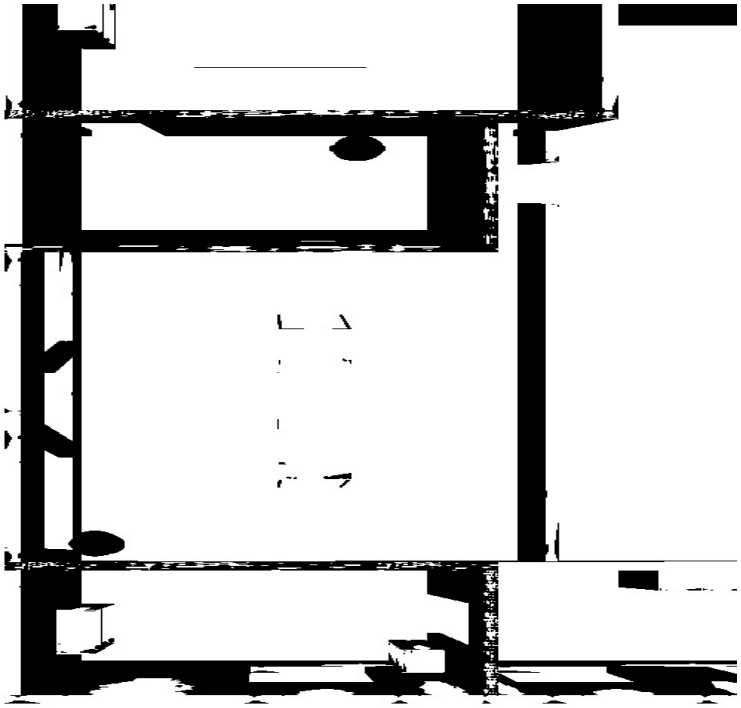
# INSTALLATION PROCESS

- First, essential tools (curl, gnupg2, lsb-release) are installed and the ROS2 repository for Foxy Fitzroy is added to the system's sources list. After updating the package index, ROS2 Foxy Desktop is installed, including core packages for development.
- Python dependencies (python3-argcomplete, python3-colcon-common-extensions, python3-vcstool) are installed to enhance development capabilities. A workspace (turtlebot3_ws) is created, and key repositories (turtlebot3, turtlebot3_msgs, turtlebot3_simulations) are cloned into its source directory.
- Dependencies for simulation (gazebo9, ros-foxy-gazebo-ros-pkgs) and hardware control (ros-foxy-dynamixel-sdk) are installed. The workspace is then built using colcon, ensuring all packages are compiled correctly.
- By following these steps, developers can establish a functional environment to simulate and control TurtleBot3 using ROS2 on Ubuntu, facilitating the development and testing of robotic applications efficiently and commands are given below.

- sudo apt update && sudo apt install curl gnupg2 lsb-release
- curl-s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add-
- sudo sh-c 'echo "deb [arch=$(dpkg--print-architecture)] http://packages.ros.org/ros2/ubuntu $(lsb_release main" >/etc/apt/sources.list.d/ros2-latest.list'
- sudo apt update
- sudo apt install ros-foxy-desktop
- source /opt/ros/foxy/setup.bash
- sudo apt install python3-argcomplete python3-colcon-common-extensions python3-vcstool
- mkdir-p ~/turtlebot3_ws/src
- cd ~/turtlebot3_ws

- source /opt/ros/foxy/setup.bash
- colcon build
- cd ~/turtlebot3_ws/src
- sudo apt install git
- git clone-b foxy-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
- git clone-b foxy-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
- git clone-b foxy-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
- cd ~/turtlebot3_ws
- sudo apt update
- sudo apt install gazebo9
- gazebo –version
- sudo apt install libgazebo9-dev
- sudo apt installros-foxy-gazebo-ros-pkgs
- sudo apt install ros-foxy-dynamixel-sdk
- colcon build –symlink-install
- cd ..
- source /opt/ros/foxy/setup.bash
- source ~/turtlebot3_ws/install/setup.bash
- export TURTLEBOT3_MODEL=waffle_pi
- ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py

# RESULTS AND DISCUSSION



The result of developing a 2D occupancy grid map of a room using overhead cameras with ROS2 and TurtleBot is a structured representation of the environment's layout in a grid format. This map divides the space into individual cells, each characterized by its occupancy status: occupied (by obstacles), free (open space), or unknown. The development process typically involves several key outcomes:

1. **Accurate Spatial Representation**: The map accurately reflects the spatial layout of the room based on the images captured by overhead cameras. High-resolution images and precise camera calibration contribute to detailed mapping, distinguishing between different types of obstacles and free spaces.
2. **Integration with ROS2 Tools**: ROS2 provides a comprehensive framework for processing sensor data and converting it into the occupancy grid map format. This integration leverages ROS2 libraries for image rectification, feature extraction (e.g., identifying walls and

furniture), and probabilistic algorithms to update grid cell values based on sensor inputs.

3. **Visualization and Monitoring**: Tools like RViz allow for real-time visualization of the occupancy grid map. This visualization capability helps developers and operators monitor the map's accuracy and effectiveness as TurtleBot navigates the environment. It facilitates debugging and fine-tuning of mapping algorithms to ensure the map remains reliable and responsive to changes in the environment.
4. **Validation and Testing**: The developed occupancy grid map undergoes rigorous validation and testing to ensure its accuracy and suitability for autonomous navigation tasks. Testing scenarios include simulated environments and real-world deployments to verify obstacle detection, map updating mechanisms, and overall system performance.
5. **Practical Applications**: The resulting occupancy grid map serves as a foundational element for autonomous navigation systems. It enables TurtleBot or similar robots to localize themselves within the environment, plan optimal paths to navigate around obstacles, and execute tasks safely and efficiently.

Overall, the result of developing a 2D occupancy grid map with ROS2 and TurtleBot enhances the robot's ability to operate autonomously in indoor environments. It provides a detailed spatial understanding crucial for safe navigation and interaction with the surroundings, demonstrating the effectiveness of ROS2's capabilities in robotics applications.

# BENEFITS AND APLLICATION

## Benefits:

1. **Accurate Spatial Mapping**: The solution provides a precise representation of the environment's layout in real-time. By integrating data from overhead cameras and processing it using advanced image processing techniques, the occupancy grid map accurately identifies obstacles, open spaces, and other features critical for navigation.
2. **Enhanced Autonomous Navigation**: With a detailed occupancy grid map, TurtleBot can autonomously navigate indoor environments with greater efficiency and safety. The map allows the robot to plan optimal paths, avoid obstacles, and adapt to dynamic changes in its surroundings, thereby improving operational effectiveness.
3. **Scalability and Adaptability**: ROS2 integration facilitates scalability and adaptability of the mapping solution. Developers can leverage ROS2's modular architecture to add or modify components, integrate additional sensors, or enhance algorithmic capabilities, making the system suitable for various robotic applications and environments.

## Applications:

1. **Robotics Research and Development**: The occupancy grid mapping technique is widely used in robotics research for exploring and mapping unknown or complex environments. Researchers can utilize the developed solution to study robot perception, navigation algorithms, and human-robot interaction in controlled laboratory settings.
2. **Industrial Automation**: In industrial settings, TurtleBot equipped with a 2D occupancy grid map can autonomously navigate factory floors, warehouses, or logistics centers. It facilitates tasks such as material handling, inventory management, and surveillance, improving operational efficiency and reducing human intervention.
3. **Service Robotics**: In service robotics applications, such as healthcare assistance or household robotics, the developed solution enables robots to navigate indoor spaces autonomously. This capability is crucial for tasks like delivery of goods, monitoring of patients, or assisting with household chores, enhancing convenience and safety.

# SOURCE CODE

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

class WallFollowingRobot(Node):

    def __init__(self):
        super().__init__('wall_following_robot')

        self.bridge = CvBridge()

        self.camera1_sub = self.create_subscription(
            Image,
            '/overhead_camera/overhead_camera1/image_raw',
            self.camera1_callback,
            10)

        self.camera2_sub = self.create_subscription(
            Image,
            '/overhead_camera/overhead_camera2/image_raw',
            self.camera2_callback,
            10)

        self.camera3_sub = self.create_subscription(
            Image,
            '/overhead_camera/overhead_camera3/image_raw',
            self.camera3_callback,
            10)

        self.camera4_sub = self.create_subscription(
            Image,
            '/overhead_camera/overhead_camera4/image_raw',
            self.camera4_callback,
            10)

        self.cmd_vel_pub = self.create_publisher(Twist, '/cmd_vel', 10)
```

```
        self.image1 = None
        self.image2 = None
        self.image3 = None
        self.image4 = None

        self.timer = self.create_timer(1.0, self.control_robot)

        self.map_image = None

    def camera1_callback(self, msg):
        self.image1 = self.bridge.imgmsg_to_cv2(msg, 'bgr8')

    def camera2_callback(self, msg):
        self.image2 = self.bridge.imgmsg_to_cv2(msg, 'bgr8')

    def camera3_callback(self, msg):
        self.image3 = self.bridge.imgmsg_to_cv2(msg, 'bgr8')

    def camera4_callback(self, msg):
        self.image4 = self.bridge.imgmsg_to_cv2(msg, 'bgr8')

    def control_robot(self):
        if self.image1 is not None and self.image2 is not None and self.image3
is not None and self.image4 is not None:
            self.process_images()
            self.follow_wall()

    def process_images(self):
        height1, width1, _ = self.image1.shape
        height2, width2, _ = self.image2.shape
        height3, width3, _ = self.image3.shape
        height4, width4, _ = self.image4.shape

        total_height = height1 + height2 + height3 + height4
        total_width = max(width1, width2, width3, width4)

        combined_image = np.zeros((total_height, total_width, 3),
dtype=np.uint8)

        combined_image[0:height1, 0:width1] = self.image1
        combined_image[height1:height1+height2, 0:width2] = self.image2
        combined_image[height1+height2:height1+height2+height3, 0:width3] =
self.image3
```

```
        combined_image[height1+height2+height3:height1+height2+height3+height4
, 0:width4] = self.image4

        grayscale_image = cv2.cvtColor(combined_image, cv2.COLOR_BGR2GRAY)

        _, binary_image = cv2.threshold(grayscale_image, 128, 255,
cv2.THRESH_BINARY)

        if self.map_image is None or self.map_image.shape !=
grayscale_image.shape:
            self.map_image = np.ones_like(grayscale_image) * 255

        self.map_image[binary_image == 0] = 0

        if int(time.time()) % 10 == 0:
            cv2.imwrite('map.pgm', self.map_image)
            self.get_logger().info('Map saved as map.pgm')

    def follow_wall(self):
        cmd_vel = Twist()

        if self.image4 is not None:
            avg_intensity = np.mean(self.image4[:, -1])
            if avg_intensity < 100:
                cmd_vel.angular.z = 0.3
            else:
                cmd_vel.angular.z = 0.0

        cmd_vel.linear.x = 0.2
        self.cmd_vel_pub.publish(cmd_vel)
        self.get_logger().info(f'Published velocity: Linear x:
{cmd_vel.linear.x}, Angular z: {cmd_vel.angular.z}')

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

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

# SOLUTION FEATURES

Developing a 2D occupancy grid map of a room using overhead cameras with ROS2 and TurtleBot integrates essential features to optimize robotic perception and navigation. ROS2 integration provides a robust middleware framework for efficient sensor data management, node communication, and algorithm implementation. This integration enhances scalability and modularity, crucial for adapting the mapping solution to diverse robotic applications.

Incorporating advanced image processing capabilities like grayscale conversion, thresholding, and binary image creation enables accurate interpretation of overhead camera data. These techniques extract crucial environmental features such as obstacles and free spaces, facilitating real-time map updates as the robot moves through its surroundings. This capability ensures reliable performance across varied lighting conditions and complex environments.

Supporting multiple overhead cameras enhances spatial perception by integrating diverse viewpoints into a unified occupancy grid map. This multi-camera approach improves coverage, accuracy in obstacle detection, and overall situational awareness. By leveraging multiple perspectives, the mapping solution enables more effective navigation in intricate indoor environments, where comprehensive spatial understanding is essential for autonomous robotic operations.

# CONCLUSION

In conclusion, developing a 2D occupancy grid map of a room using overhead cameras with ROS2 and TurtleBot represents a significant advancement in robotic perception and navigation capabilities. The integration of ROS2 provides a robust foundation for efficient sensor data management, communication between robotic components, and implementation of sophisticated algorithms. This framework ensures scalability and modularity, making the solution adaptable to various robotic applications.

The incorporation of advanced image processing techniques enables the accurate interpretation of environmental data captured by overhead cameras. Techniques such as grayscale conversion and binary image creation facilitate the detection of obstacles and delineation of free spaces, crucial for updating the occupancy grid map in real-time as the robot navigates its environment. This capability enhances the robot's ability to operate autonomously in diverse and challenging indoor environments.

Overall, the development of a 2D occupancy grid map with ROS2 and TurtleBot underscores the importance of robust software frameworks and advanced perception techniques in advancing autonomous robotics. This technology paves the way for safer, more efficient robotic operations across various industrial, commercial, and research domains.