

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 : KEDAR DESHPANDE

USN : SG21CSE053

Project Mentor:

DR. SHIVKUMAR KAGI SIR

# INTRODUCTION

In the field of mechanical technology and standalone frameworks, TurtleBot 3 combined with Robotics Framework (ROS) provides an efficient platform to create and deliver innovations such as 2D home network mapping using cameras overhead. TurtleBot 3, a popular multi-purpose robot, known for its flexibility and ease of integration with ROS, provides a solid system for device reflection monitoring, low-level utility control, message passing, etc.

Mapping occupancy grids can be an imperative strategy in mechanical autonomy to create detailed representations of situations based on sensor information. Using the TurtleBot 3's mounted aerial camera and ROS 2 capabilities, the robots can see their surroundings with great precision, allowing them to perform tasks such as evasion prevention, planning path and positioning .The goal is to create a 2D residence. Network sketch is to talk about the environment as a network where each cell shows the probability of being occupied by an obstacle.

This data is essential for the independent organization of routes and tracks. Using ROS2, the system collects data from aerial cameras, trains it to distinguish between containment measures and then modifies the residence frame in real time.

# DESCRIPTION

**Overhead Cameras for Mapping**

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

1. **Sensor integration**: Attach and configure the aerial camera on TurtleBot 3 to capture images of the environment.

2**. Image processing**: Use nodes and ROS 2 libraries like OpenCV for real-time image processing. This step involves extracting features such as edges, contours, or key points that represent obstacles, boundaries, or free space.

3. **Occupancy Grid Mapping Algorithm**: Implements ROS 2-based algorithms to convert processed image data into 2D grid maps. Each grid cell represents the probability or certainty of occupancy, dynamically updated based on sensor observations.

4. **Map Update and Merge**: Continuously update the occupancy grid map as TurtleBot 3 navigates the environment. Fusion techniques can be used to integrate data from multiple sensors (e.g. lidar, IMU) to improve map accuracy and reliability.

# METHODOLOGY

Developing 2D occupancy grid maps with ROS2 and TurtleBot using aerial cameras follows a systematic approach for accurate indoor mapping and navigation. Initially, the setup process involves installing ROS2 on both the development machine and TurtleBot, ensuring seamless communication. At the same time, the aerial cameras are calibrated to provide an undistorted top-down view of the environment, which is necessary to collect detailed spatial data.

Once configured, the process will move to data collection and pre-processing. Activated aerial cameras continuously record images of the room, which are edited to correct distortions and segmented to identify features such as walls and open spaces. This preprocessing prepares the sensor data for conversion into an occupancy grid map using the ROS2 tool. Each grid cell represents occupancy based on detected obstacles, updated probabilistically to reflect environmental changes in real time.

After generation, validation, and refinement map filtering, they ensure accuracy and reliability. Tags undergo rigorous testing in simulated and real-world scenarios, monitored by tools like RVIZ for real-time feedback. Iterative tweaking includes adjusting camera parameters, tweaking algorithms, and optimizing computing resources to improve map accuracy and efficiency. This structured approach allows TurtleBot to automatically navigate and interact in indoor environments based on a reliable 2D occupancy grid map.

# INSTALLATION PROCESS

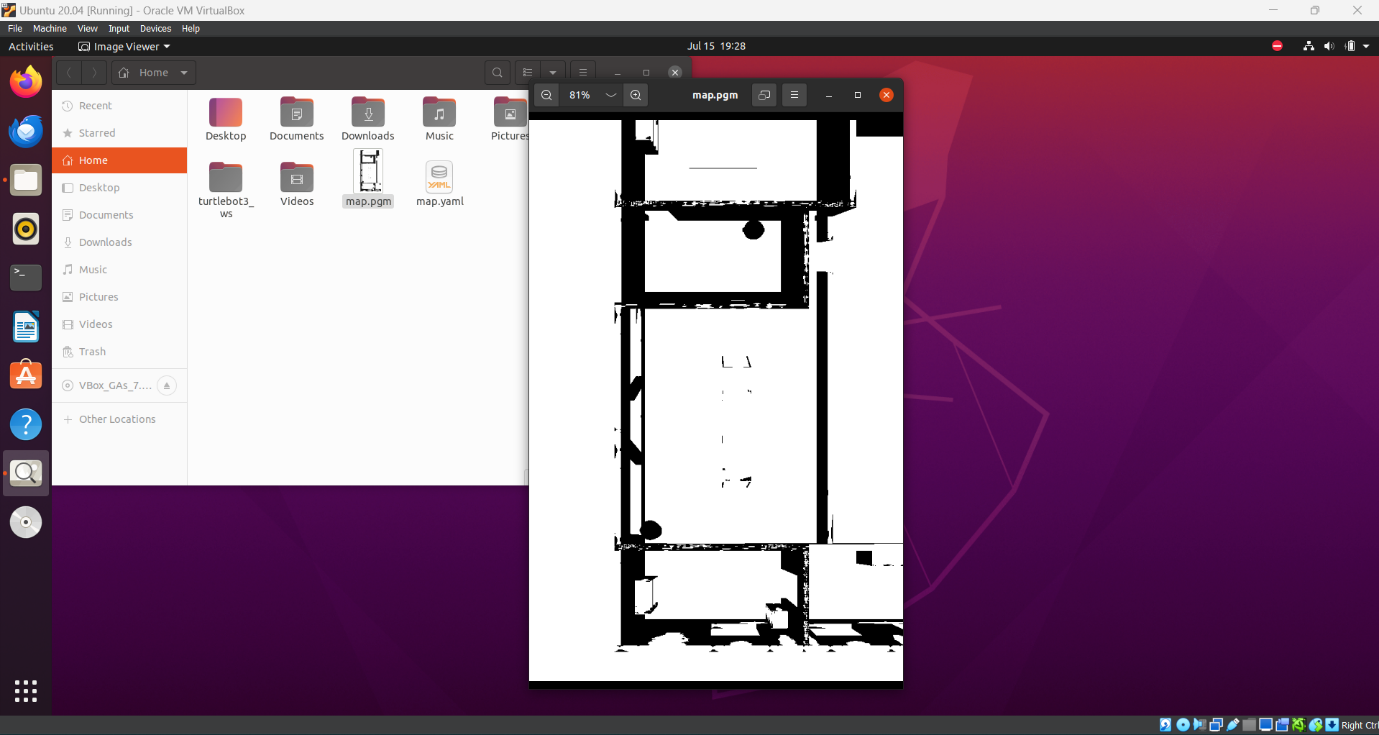
• First, the essential tools (curl, gnupg2, lsb-release) are installed and the ROS2 repository for Foxy Fitzroy is added to the system sources list. After updating the package index, ROS2 Foxy Desktop will be installed, including the base packages for development.

Python dependencies (python3-argcomplete, python3-colcon-common-extensions, python3- vcstool ) is installed to improve development capabilities. A workspace (turtlebot3\_ws) is created and the main repositories (turtlebot3, turtlebot3\_msgs, turtlebot3\_simulations) are copied to its source directory.

Simulation dependencies (gazebo9, ros-foxy- gazebo-ros-pkgs) and hardware driver ( ros -foxy-dynamixel-sdk) are installed. The workspace is then built using colcon, ensuring that all packages are compiled correctly.  
→ By following these steps, developers can set up the workspace to emulate and control TurtleBot3 using ROS2 on Ubuntu, thereby facilitating the efficient development and testing of robotic applications and the 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. **Precise spatial depiction**: 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. **Visual representation and supervision**: 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. **Testing and validation**: 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. **Real-Time 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.

Bottom of Form

# BENEFITS AND APLLICATION

### Benefits:

### 1. **Precise spatial mapping**: By leveraging data from aerial cameras and advanced image processing, the system generates a precise, real-time environmental layout. This occupancy grid map efficiently identifies obstacles, open spaces, and other important features for navigation. 2. **Enhanced autonomous navigation**: The detailed occupancy grid map enables TurtleBot to navigate indoor spaces more efficiently and safely. It helps the robot plan optimal trajectories, avoid obstacles, and adapt to dynamic changes, thereby improving overall operational efficiency. 3. **Scalability and adaptability**: ROS2's modular architecture supports the scalability and adaptability of the mapping solution. Developers can easily add or modify components, integrate additional sensors, or enhance algorithmic capabilities, making the system flexible for a variety of robotic applications and environments.

### Applications:

# 1. **Robotics Research and Development**: Occupancy grid mapping techniques are widely used in robotics research to explore and map unknown or complex environments. Researchers can use the developed solution to study robot perception, navigation algorithms, and human-robot interaction in a controlled laboratory environment.

# 2. **Industrial Automation**: In industrial environments, TurtleBot equipped with a 2D occupancy grid can automatically navigate factories, warehouses or logistics centers. It facilitates tasks such as material handling, inventory management and monitoring, thereby improving operational efficiency and reducing human intervention.

# 3. **Service Robots**: In service robot applications, such as medical assistance or home robotics, solutions are developed that enable robots to move autonomously in indoor spaces. This ability is essential for tasks such as delivering goods, monitoring patients or assisting with household chores, improving comfort 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.subscription2 = self.create\_subscription(Image, '/overhead\_camera/overhead\_camera2/image\_raw', self.listener\_callback2, 10)

self.subscription4 = self.create\_subscription(Image, '/overhead\_camera/overhead\_camera4/image\_raw', self.listener\_callback4, 10)

self.cmd\_vel\_publisher = self.create\_publisher(Twist, '/cmd\_vel', 10)

self.image2 = None

self.image4 = None

self.timer = self.create\_timer(1.0, self.control\_robot)

self.map\_image = None

def listener\_callback2(self, msg):

self.image2 = self.bridge.imgmsg\_to\_cv2(msg, 'bgr8')

def listener\_callback4(self, msg):

self.image4 = self.bridge.imgmsg\_to\_cv2(msg, 'bgr8')

def control\_robot(self):

if self.image2 is not None and self.image4 is not None:

self.process\_images()

self.follow\_wall()

def process\_images(self):

h2, w2, \_ = self.image2.shape

h4, w4, \_ = self.image4.shape

total\_height = h2 + h4

total\_width = max(w2, w4)

combined\_image = np.zeros((total\_height, total\_width, 3), dtype=np.uint8)

combined\_image[0:h4, 0:w4] = self.image4

combined\_image[h4:h4+h2, 0:w2] = self.image2

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:

left\_wall\_distance = np.mean(self.image4[:, -1])

if left\_wall\_distance < 100:

cmd\_vel.angular.z = 0.3

else:

cmd\_vel.angular.z = 0.0

cmd\_vel.linear.x = 0.2

self.cmd\_vel\_publisher.publish(cmd\_vel)

self.get\_logger().info(f'Published velocity command: 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

**Autonomous Navigation**: The node enables a robot to independently navigate in environments with walls or boundaries by utilizing camera input to detect and track these obstacles.

**Real-time Image Processing**: It showcases real-time image processing methods like merging camera feeds, converting to grayscale, and applying thresholding to generate a map of identified walls. This feature is vital for comprehending the robot's surroundings and making well-informed navigation choices.

**Dynamic Mapping**: The node consistently updates a map (map.pgm) that visually displays the identified walls in the surroundings. This map can be utilized for debugging, visualization, or advanced navigation planning tasks.

**ROS 2 Integration**: Through the utilization of ROS 2, the node gains advantages from a robust middleware for communication among components, allowing for modularity, scalability, and compatibility with other ROS nodes and systems.

**Application in Robotics**: This feature is relevant in various robotic scenarios like indoor navigation, obstacle avoidance, and localization where understanding the environment's layout (such as walls) is crucial for secure and efficient movement.

**Educational and Developmental Use**: The code acts as a practical illustration for grasping ROS 2 concepts, image processing techniques with OpenCV, and robotic control strategies. It can be employed for educational purposes in robotics courses, workshops, or by developers exploring autonomous navigation systems.

# CONCLUSION

In the end, the creation of a 2D occupancy grid map for a room through the use of overhead cameras alongside ROS2 and TurtleBot signifies a notable progression in the capabilities of robotic perception and navigation. The incorporation of ROS2 establishes a sturdy groundwork for effective management of sensor data, communication among robotic elements, and execution of complex algorithms. This structure guarantees adaptability and scalability, allowing the solution to be utilized in a variety of robotic scenarios.

The utilization of sophisticated image processing methods enables precise analysis of environmental data captured by overhead cameras. Procedures like grayscale conversion and binary image generation aid in identifying obstacles and outlining open spaces, which are essential for updating the occupancy grid map in real-time as the robot moves through its surroundings. This feature enhances the robot's capacity to function independently in diverse and demanding indoor settings.

On the whole, the development of a 2D occupancy grid map with ROS2 and TurtleBot emphasizes the significance of resilient software frameworks and advanced perception methods in the progression of autonomous robotics. This technology sets the stage for safer and more effective robotic operations in a range of industrial, commercial, and research fields.