SWROB2
Exam

Submitted by:

Study nr	Name	Study line
202005180	Nicolaj Meldgaard Pedersen	E
202105443	Johannes Baagøe	E
201270449	Anders Sandø Østergaard	EP
201905293	Daniel F. Borch Olsen	E

Århus Universitet

May 5, 2024

Abstract

This report presents the design, implementation, and evaluation of an advanced robotic system integrated with the Robot Operating System (ROS). The project leverages ROS's dynamic capabilities alongside sophisticated algorithms to enhance robotic functionalities in motion control, motion planning, perception using camera algorithms, localization, and mapping. Utilizing MATLAB and a specific hardware setup, the system demonstrates significant improvements in task efficiency and obstacle management within a controlled experimental setup. The findings highlight the system's robustness in real-time operations and its potential for adaptation in varied automation scenarios. This study not only showcases the successful application of ROS in complex robotic tasks but also sets a foundation for future advancements in robotic automation. The report concludes with an analysis of experimental results, discussing the system's performance against predefined objectives and suggesting areas for further research.

Contents

1 Introduction

In recent years, the integration of robotics into industrial and research applications has significantly advanced, driven by innovations in robotics operating systems and algorithm development. One of the most influential platforms in this evolution is the Robot Operating System (ROS), which provides tools and libraries designed to support the development of complex and robust robot behavior. This report focuses on a specific robotics system designed for the demonstration and testing of various robotics algorithms using ROS, providing insights into the practical application of motion control, planning, perception, localization, and mapping.

The primary objective of this project is to develop and evaluate a robot system that effectively demonstrates the capabilities of various algorithms in a controlled environment. By utilizing ROS and other software tools, such as Matlab, coupled with a meticulously selected hardware setup, the project aims to showcase efficient task handling, obstacle avoidance, and route optimization.

This report will cover several key aspects:

1. **Robot System Description:** An in-depth look at the overall setup, including detailed descriptions of the hardware and software components utilized in the project.
2. **Principles of ROS:** An explanation of how ROS operates, focusing on its architecture and the communication patterns that facilitate multi-component integration in robotics projects.
3. **Algorithms:** A discussion on the core algorithms developed and implemented, detailing their design, objectives, and impact on the system's performance.
4. **Experimental Results:** Presentation and analysis of the data collected from various tests, providing a quantitative basis for evaluating the effectiveness of the implemented algorithms.
5. **Discussion and Conclusion:** Insights derived from the project outcomes, including an assessment of the system's performance and recommendations for future enhancements.

The subsequent sections will delve into each of these areas, providing a comprehensive overview of the project from conception to execution, accompanied by a critical evaluation of the results and their implications for future robotics applications.

2 Robot System Description

2.1 Description of the Setup

This project utilizes the TurtleBot3, a popular and versatile platform widely used in robotics education and research. The TurtleBot3 is known for its modular design and compatibility with various ROS versions, making it an ideal candidate for demonstrating advanced robotics algorithms. The robot is tasked with performing a series of autonomous operations, including navigation, obstacle avoidance, and object recognition, within a controlled environment designed to simulate real-world conditions. The setup includes a predefined track with various obstacles and visual markers to challenge the robot's perception and motion planning capabilities.

2.2 Hardware/Software Used

The hardware foundation of the system is the TurtleBot3 robot, equipped with standard sensors such as Lidar for mapping and navigation, and a camera module for visual perception tasks. The primary software components include:

- **ROS1 Noetic:** The latest long-term support version of ROS1, serving as the middleware that facilitates communication between the robot's hardware components and high-level algorithms.

- **MATLAB:** Used for developing some of the robot's algorithms, especially in areas related to image processing and data analysis. MATLAB's integration with ROS allows for direct communication between the MATLAB scripts and ROS topics, enhancing the system's ability to perform complex calculations and algorithm testing efficiently.

This combination of cutting-edge software with robust hardware enables the TurtleBot3 to execute complex tasks required by the project while ensuring reliability and scalability.

2.3 Description of the Setup

Step 1: Prepare the Raspberry Pi 3

- **Install an appropriate OS:**
  - Install Ubuntu Mate 20.04 LTS for Raspberry Pi. Ubuntu Mate is recommended because it provides a stable environment for ROS and is compatible with the Raspberry Pi hardware.
  - Download the image from the Ubuntu Mate website and flash it onto an SD card using a tool like BalenaEtcher.
- **Set up the Raspberry Pi:**
  - Insert the SD card, connect your monitor, keyboard, and mouse, then power up the Raspberry Pi.
  - Go through the initial setup including configuring your locale, timezone, and WiFi.

Step 2: Install ROS Noetic on Raspberry Pi

- Install ROS Noetic:
  - Open a terminal and execute the following commands:

```
sudo apt update && sudo apt upgrade
sudo apt install curl
curl -sSL http://repos.ros.org/repos.key | sudo apt-key add -
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt
sudo apt update
sudo apt install ros-noetic-ros-base
```

- Initialize rosdep:

```
sudo rosdep init
rosdep update
```

- Setup your environment:

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

Step 3: Install TurtleBot3 Packages

- In the terminal, run:

```
sudo apt install ros-noetic-turtlebot3 ros-noetic-turtlebot3-msgs ros-noetic-turtlebot3-simulati
echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc  # change "burger" to your model: burger, waf
source ~/.bashrc
```

Step 4: Set Up Network Communication

- Configure network settings so the Raspberry Pi and Windows machine can communicate:
  - Determine the IP of the Raspberry Pi (ifconfig) and your Windows machine (using ipconfig in CMD).
  - Ensure both devices are on the same network.

Step 5: Install MATLAB on Windows and Configure for ROS

- Install MATLAB and include the Robotics System Toolbox.
- Setup the ROS Toolbox in MATLAB:
  - In MATLAB, configure the connection to the ROS master running on the Raspberry Pi:

```
rosinit('http://<Raspberry_Pi_IP>:11311')
```

Step 6: Test the System

- Test by controlling the TurtleBot3 from MATLAB:
  - You can publish velocity commands or subscribe to sensor topics from MATLAB to control the TurtleBot3 or process its sensor data.
  - Example MATLAB command to send a movement command:

```
pub = rospublisher('/cmd_vel', 'geometry_msgs/Twist');
msg = rosmessage(pub);
msg.Linear.X = 0.1;  % Move forward at 0.1 m/s
send(pub, msg);
```

2.4 Hardware/Software Used

The TurtleBot3 project utilizes a combination of specialized hardware and software to achieve its objectives. This setup is designed to support a variety of robotic applications, including navigation, obstacle avoidance, and data acquisition.

Hardware

- **TurtleBot3 Robot:** The core of the project, the TurtleBot3, is a compact and customizable mobile robot platform compatible with various ROS versions. It is specifically designed for education and research in robotics. The model used in this project is the TurtleBot3 Burger, which includes the following components:
  - **Raspberry Pi 3:** Serves as the main control unit.
  - **360-degree LiDAR sensor:** For environmental scanning and mapping.
  - **Camera:** Used for visual object and obstacle detection.
  - **Dynamixel motors:** Provide precise and powerful actuation for movement.
- **Additional Sensors:** Include ultrasonic sensors for obstacle detection and inertial measurement units (IMUs) for orientation and motion detection.

Software

- **ROS Noetic:** The Robot Operating System (ROS) serves as the middleware that offers a suite of tools and libraries designed to facilitate the development of robot applications. ROS Noetic, specifically tailored for long-term support versions of Ubuntu, such as 20.04 LTS, is used for its reliability and extensive community support.
- **MATLAB:** Utilized for advanced algorithm development, data analysis, and visualization, MATLAB integrates seamlessly with ROS, allowing for sophisticated control strategies and simulation capabilities to be developed. The Robotics System Toolbox in MATLAB is particularly used to prototype, test, and run these algorithms efficiently.

This hardware and software amalgamation not only enhances the TurtleBot3's functionalities but also simplifies the complexities involved in robotic research and education.

3 Principles of ROS

3.1 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source framework for writing robot software. It provides a structured communications layer above the host operating systems of a mixed compute cluster. ROS's utilities, libraries, drivers, conventions, and tools are designed to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS is not an operating system in the traditional sense of process management and scheduling; instead, it provides a structured framework that supports the development of distributed computing solutions.

3.2 Services

In ROS, services are a synchronous remote procedure call mechanism that allows nodes to send a request and receive a response. It is defined by a pair of messages: one for the request and one for the response. Services are useful for operations where you need to perform a specific function and wait for a result before proceeding, making them suitable for tasks that are not time-sensitive but require a guarantee of completion before moving on.

3.3 Topics

Topics are named buses over which nodes exchange messages. Communication on topics happens via a publish/subscribe mechanism where nodes send messages (publish) without knowing who, if anyone, will receive them, and receive messages (subscribe) without knowing who, if anyone, has sent them. This model is particularly well-suited for streaming data, such as sensor readings and control signals where it is not critical to handle every message that comes through.

3.4 Messages

Messages are simple data structures, comprising typed fields. They can include arrays and constants and are used to send data among nodes. Each topic and service uses a specific message type, which is a data structure predetermined by the sender and receiver in the communication process. ROS supports complex message structures, allowing a wide variety of information to be passed between nodes.

3.5 Publisher

A publisher is a node that creates and sends messages in a ROS network. Publishers declare the topic on which they will publish, the type of message they will send, and then publish messages to anyone who is subscribed. This setup allows for real-time data sharing and is crucial for tasks that involve broadcasting sensor data or status information.

3.6 Subscriber

A subscriber is a node that receives messages from a specific topic in a ROS network. Subscribers declare the topic they want to receive and the type of message it is. They are then responsible for handling these messages as they arrive. This mechanism is vital for nodes that need to perform actions based on data from other parts of the system, such as processing sensor inputs or executing commands based on received instructions.

4 Algorithms

4.1 Motion control

4.1.1 Pure Pursuit Algorithm

The Pure Pursuit algorithm is a geometric path tracking method commonly used in autonomous robotics and self-driving vehicles to follow a predefined path. The algorithm calculates the required steering angle to follow a path by continuously chasing a point ahead on the path, which is why it's called "pure pursuit."

Basic Concept

The fundamental idea behind the Pure Pursuit algorithm is relatively simple:

1. A look-ahead point is dynamically chosen on the path at a fixed distance ahead of the current position of the robot or vehicle.
2. The algorithm then calculates the curvature (steering angle) required for the vehicle to travel in a circular arc to reach this look-ahead point.
3. The curvature is recalculated at each update step as the vehicle progresses along the path, ensuring dynamic adjustment to the path's shape and the vehicle's current state.

Operational Details

To implement the Pure Pursuit algorithm, several steps must be followed:

- **Path Representation:** The path must be defined and stored in a format that allows for easy calculation of the look-ahead point. This is typically done using a series of waypoints that define the path.
- **Look-ahead Distance:** A key parameter in the Pure Pursuit algorithm is the look-ahead distance, which needs to be tuned based on the vehicle's speed, dynamics, and the path's curvature. Generally, faster speeds or sharper curves require longer look-ahead distances.
- **Calculation of the Steering Angle:** The steering angle is calculated based on the geometry formed by the vehicle's current position, its heading, and the look-ahead point. The goal is to minimize the heading error, which is the angle between the vehicle's current heading and the line segment connecting the vehicle's position to the look-ahead point.

Advantages and Limitations

Advantages:

- **Simplicity:** The Pure Pursuit algorithm is straightforward to implement and understand.
- **Effectiveness:** It is very effective for smooth paths and can be implemented in real-time applications due to its computational efficiency.

Limitations:

- **Performance in Sharp Turns:** The algorithm may struggle with very sharp turns or abrupt path changes because it only considers the current look-ahead point without anticipating future path changes.
- **Dependency on Look-ahead Distance:** The choice of look-ahead distance can greatly affect the performance of the algorithm, requiring careful tuning based on specific vehicle dynamics and path characteristics.

Applications

The Pure Pursuit algorithm is extensively used in scenarios where path tracking is crucial, such as in autonomous vehicles, unmanned ground vehicles in agriculture, or robotic systems in industrial settings.

In conclusion, while the Pure Pursuit algorithm has some limitations, its simplicity and effectiveness make it a popular choice for real-world applications requiring robust path following capabilities under varying operational conditions.

4.2 Perception – Camera Algorithms

Camera-based perception algorithms are pivotal for enabling autonomous robots to navigate and understand their surroundings effectively. These algorithms allow robots to process visual information, enabling them to detect and interact with objects, recognize patterns, and navigate through their environments.

4.2.1 Overview of Camera-Based Perception

In robotic systems, cameras serve as the eyes, providing high-dimensional data that can be processed to extract meaningful information. The processing of this data is achieved through various algorithms designed to interpret complex visual cues from the environment:

- **Image Processing:** Basic manipulation of images to enhance the visibility of features, such as edge detection, filtering, and color segmentation.
- **Feature Detection and Matching:** Algorithms like Scale-Invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF) identify and match features from different images to help in object recognition or localization.
- **Object Detection:** Techniques that recognize objects within the field of view of the camera using trained machine learning models, such as Convolutional Neural Networks (CNNs).
- **Depth Perception:** Using stereo vision or structured light to gauge the distance to objects, which is crucial for navigation and manipulation tasks.

4.2.2 Key Algorithms in Camera-Based Perception

1. **Convolutional Neural Networks (CNNs):** These are particularly effective for image classification and object recognition tasks. CNNs automatically learn and optimize their filters to capture essential features for analysis.
2. **Optical Flow:** This algorithm estimates the motion of objects between consecutive frames captured by the camera based on the visual inputs. It is crucial for tasks requiring movement detection and tracking.
3. **Real-time Object Detection Frameworks:** Frameworks like You Only Look Once (YOLO) and Single Shot MultiBox Detector (SSD) enable the detection of objects in real-time with high accuracy and speed.

4.2.3 Challenges in Camera-Based Perception

Despite their capabilities, camera-based perception algorithms face several challenges:

- **Variable Lighting Conditions:** Changes in environmental lighting can significantly affect the performance of vision algorithms, making robustness a critical concern.
- **Occlusions and Overlaps:** Objects may be partially or fully occluded by other objects, complicating detection and recognition tasks.
- **High Computational Demand:** Processing high-resolution video in real-time requires substantial computational resources, which can strain the processing capabilities of autonomous robots.

4.2.4 Applications

Camera-based perception is extensively used in:

- **Autonomous Vehicles:** For navigation and to detect and respond to other vehicles, pedestrians, and road signs.
- **Industrial Automation:** In manufacturing environments, where precision and reliability in identifying and manipulating objects are required.
- **Service Robotics:** In environments such as hospitals and homes to assist individuals based on visual cues.

In conclusion, camera algorithms play an indispensable role in the perception systems of modern robots, allowing them to perform complex tasks autonomously. Continued advancements in computational power and algorithm efficiency are critical to overcoming current limitations and expanding the capabilities of robotic systems.

4.3 Localization

Localization is a fundamental aspect of autonomous robotics, enabling a robot to determine its position within an environment. In the context of the final project, the TurtleBot must accurately identify its location within the Shannon building to navigate between specified points and perform tasks effectively.

4.3.1 Project Requirements and Localization Strategy

The project requires the TurtleBot to start at a fixed location (Location A) and autonomously travel to Locations B and C to perform specific tasks. The sequence of tasks includes object recognition and navigation around obstacles, demanding precise and reliable localization capabilities. The key steps involving localization are:

1. **Initial Positioning:** The robot starts from Location A. Initial localization is straightforward, as the starting point is predefined and known.
2. **Navigation to Location B:** The robot must autonomously navigate from Location A to B, avoiding any obstacles. This requires dynamic localization to continually update the robot's current position and adjust its path.
3. **Object Recognition and Position Adjustment at Location B:** Upon arriving at B, the robot must locate a red circle on the wall, approach it to a distance of 50 cm, and take a picture. This task requires the robot to fine-tune its localization to position itself precisely relative to the target object.
4. **Path Planning to Location C:** After completing the task at Location B, the robot must find the optimal route to Location C. This involves complex path planning algorithms that depend on accurate localization to navigate around static and dynamic obstacles.
5. **Object Recognition and Position Adjustment at Location C:** Similar to Location B, the robot must locate a purple circle, approach, and photograph it. This again requires precise localization to successfully complete the task.

4.3.2 Techniques and Tools for Effective Localization

To achieve effective localization, several techniques and tools are employed:

- **Odometry:** Utilized to estimate the robot's change in position over time. This is achieved by interpreting data from wheel encoders.
- **Simultaneous Localization and Mapping (SLAM):** Allows the robot to build a map of its environment while simultaneously keeping track of its location within it. For this project, SLAM can be particularly useful during the initial mapping phase and subsequent navigation tasks.
- **Sensors:** The TurtleBot uses a combination of sensors, including LiDAR for obstacle detection and a camera for visual identification of the target markers (red and purple circles).

4.3.3 Challenges and Considerations

- **Dynamic Obstacles:** The ability to avoid dynamic obstacles requires the robot's localization system to be highly responsive and accurate in real-time.
- **Environmental Factors:** Changes in the environment, such as moved furniture or altered lighting conditions, can pose challenges to localization accuracy.

4.4 Mapping

Mapping is a crucial capability in robotics, particularly for autonomous navigation tasks. It involves creating a model of the robot's environment that can be used to plan and execute movements accurately. In the context of this project, the TurtleBot is required to map the Shannon building in order to navigate effectively and autonomously perform tasks in Locations B and C.

4.4.1 Role of Mapping in the Project

The mapping process for the TurtleBot involves several key functions:

1. **Environmental Model Creation:** Before the TurtleBot can begin its tasks, it needs a detailed map of the Shannon building. This map serves as the fundamental guide for all navigation and task execution.
2. **Integration with Localization:** Mapping is closely integrated with localization; as the TurtleBot moves and captures data about its environment, it simultaneously updates its position within the map, ensuring accurate navigation.
3. **Obstacle Identification:** The map must include information about static obstacles (like walls and furniture) and allow for the recognition and avoidance of dynamic obstacles.

4.4.2 Mapping Techniques and Technologies

To effectively create and utilize maps, the TurtleBot employs several advanced technologies and techniques:

- **Simultaneous Localization and Mapping (SLAM):** SLAM techniques are used to construct the map and localize the robot within that map in real time. This is critical for the TurtleBot as it allows for dynamic interaction with the environment.
- **LiDAR and Sensors:** LiDAR sensors provide detailed and accurate distance measurements that feed into the SLAM process, helping to create a precise map of the environment. Additional sensors, such as cameras, are used for detecting specific features and markers like the red and purple circles.
- **Path Planning Algorithms:** Once a map is created, path planning algorithms calculate the most efficient route from one point to another, considering both the mapped environment and current localization data.

4.4.3 Challenges in Effective Mapping

Mapping in a dynamic environment like the Shannon building poses several challenges:

- **Dynamic Changes:** Changes within the building, such as moved furniture or temporary obstructions, require the TurtleBot to update its map regularly or recalibrate its path planning.
- **Computational Requirements:** High-resolution mapping requires significant computational resources, which can challenge the processing capabilities of onboard systems like those in the TurtleBot.
- **Accuracy and Resolution:** The accuracy of the map directly affects the robot's ability to perform tasks precisely, such as stopping at a specific location to take a picture.

4.4.4 Applications of Mapping

- **Task Execution:** Accurate maps are essential for precisely executing tasks, such as locating markers and navigating to specific points within the building.
- **Safety and Navigation:** Effective mapping ensures that the TurtleBot avoids collisions and navigates safely around people and objects.

In summary, mapping is a foundational aspect of this project, enabling the TurtleBot to perform its tasks with high efficiency and precision. The ability to dynamically interact with the environment through continuous mapping and localization adjustments is key to the success of the TurtleBot's operations in the Shannon building.

5 Results

6 Discussion

7 Conclusion

8 Appendices

9 References