ISSN 1870-4069

Development of an Autonomous Navigation and Manipulation Robot for Obstacle-Rich Environments

Andrea M. Salcedo-Vázquez, José A. León-Navarro,
Hortencia A. Ramírez-Vázquez, José G. Buenaventura-Carreón,
Fernando Antunez-Arnold

Tecnólogico de Monterrey,
Escuela de Ingeniería y Ciencias,
Monterrey, Mexico

{asalcedo2702,jantonioleon.irs}@gmail.com,
ha.ramirez@outlook.com,
{jgusbc,fer.antunez02}@hotmail.com

Abstract. This paper presents the development of an advanced autonomous navigation and manipulation robot specifically designed to operate efficiently in environments with numerous obstacles. The robot features a differential drive system, complemented by a camera and LiDAR sensors, which enable it to identify and locate target objects marked with ArUco codes. Leveraging advanced navigation algorithms and the Extended Kalman Filter (EKF) for sensor data fusion, the system achieves high-precision localization and calculates the target's position with notable accuracy. During operation, the robot dynamically plans its path to navigate toward the target while skillfully avoiding obstacles. Upon reaching the identified object, the robot uses a gripper to securely grasp the target and transport it to a designated unloading area. The integration of these functionalities underscores the robot's capability to perform complex tasks autonomously, significantly enhancing efficiency and safety in industrial environments. This research highlights the synergy of interdisciplinary technologies, combining robotics, sensor fusion, and control systems. Through rigorous iterative testing, the reliability and practicality of the system in real-world scenarios were validated. The proposed robotic solution offers a promising framework for automating tasks in complex environments, showcasing potential applications in various industries, including logistics and manufacturing, where precision and adaptability are critical.

Keywords: Differential mobile robot, navigation, manipulation, extended Kalman filter, optimization, autonomous.

1 Introduction

1.1 State of the Art in Mobile Robots

Mobile robots have significantly advanced, integrating sophisticated hardware and software to achieve high levels of autonomy and functionality.

Some of the most notable examples include Boston Dynamics Atlas, which utilizes high-performance hardware such as advanced actuators and sensors to perform complex tasks such as dynamic navigation and manipulation ([6]). Similarly, the TurtleBot series, commonly used in research and education, leverages ROS (Robot Operating System) and features hardware like the Intel RealSense camera and powerful processors to perform tasks such as mapping and human interaction ([10]).

Industrial autonomous mobile robots (AMRs) like KUKA's KMR iisy and KMP series incorporate AI and advanced path planning algorithms. These robots are equipped with high-end sensors, powerful on-board processors, and robust mechanical systems, enabling them to operate efficiently in dynamic and cluttered environments ([7]). For example, the KMR iisy includes an industrial-grade LiDAR, multiple cameras, and high-capacity batteries to support long operational hours.

In contrast, the Puzzlebot used in our research, developed by Manchester Robotics, operates with more constrained hardware resources. This robot is equipped with a Nvidia Jetson Nano featuring 2 GB of RAM, a Hackerboard for motor control, a Raspberry Pi Camera for vision, and a LiDAR sensor for obstacle detection. Despite these limitations, the Puzzlebot demonstrates the potential for effective autonomous navigation and manipulation through optimized algorithm implementation.

1.2 Contribution of This Paper

The advancements in mobile robotics are remarkable, but there is a critical challenge in adapting these sophisticated technologies to operate on less powerful hardware. This paper addresses this gap by optimizing various algorithms to work effectively on constrained hardware resources. Specifically, we adapted advanced navigation and manipulation algorithms to run on a Jetson Nano, a compact and less powerful computing module, as opposed to the high-performance GPUs typically used in state-of-the-art systems.

The primary contribution of this research is the demonstration of robust autonomous navigation and manipulation capabilities on a hardware platform with limited computational power. By fine-tuning the algorithms for real-time performance and efficient resource utilization, we showcase a cost-effective approach that broadens the accessibility and applicability of autonomous mobile robots in various fields, particularly where high-end hardware is not feasible.

This paper presents the development and implementation of an autonomous mobile robot equipped with a differential drive system, a camera, and LiDAR sensors. Using a suite of algorithms, including the Extended Kalman Filter (EKF) for sensor data fusion, the Bug0 algorithm for obstacle avoidance, and a proportional controller for path planning. The system's ability to navigate complex environments, identify and manipulate objects marked with ArUco codes, and perform tasks autonomously underscores its potential for industrial applications, particularly in settings requiring cost-effective and efficient robotic solutions.

By focusing on optimizing software to compensate for hardware limitations, this research contributes to the advancement of autonomous robotics, enabling the deployment of capable mobile robots in a wider range of scenarios, including educational, research, and industrial applications.

2 Puzzlebot

2.1 Hardware

The Puzzlebot is an educational mobile robot designed for learning and experimentation in robotics. It features a compact and robust design with dimensions typically measuring around 22 cm in length, 17 cm in width, and 12 cm in height. This size makes it suitable for navigating and performing tasks in various environments, from classroom settings to more complex obstacle courses. (figure 1). The hardware of the Puzzlebot includes several key components.

– Jetson Nano, a powerful and versatile computing module from NVIDIA. The Jetson Nano provides the processing power necessary for running complex algorithms and machine learning tasks, making it possible for the Puzzlebot to perform real-time image processing and sensor data fusion.

– LiDAR sensor, the Puzzlebot can measure distances to surrounding objects with high precision. LiDAR works by emitting laser pulses and measuring the time it takes for the pulses to reflect back from objects, creating a detailed map of the environment. This sensor is crucial for obstacle detection and avoidance, enabling the robot to navigate safely through its environment.

– Hackerboard, a versatile microcontroller board that handles lower-level control tasks. The Hackerboard interfaces with various sensors and actuators, including the motors that drive the robot's wheels. It ensures smooth and precise control of the robot's movements, executing commands from the main processing unit (Jetson Nano) and managing the real-time operation of the robot's hardware components.

2.2 Software

On the software side, the Puzzlebot runs on an environment composed by different platforms:

– Linux-based operating system, leveraging the powerful capabilities of the Jetson Nano.

– ROS (Robot Operating System), which provides a flexible framework for writing robot software. ROS offers tools and libraries for handling sensor input, motion control, and inter-component communication, streamlining the development process.

– For Simulation, we implemented different softwares as Gazebo and RViz:
  – Gazebo, a robust robotics simulator that allows us to create realistic 3D environments where the Puzzlebot can be tested. Gazebo helps in testing and refining algorithms before deploying them on the physical robot, reducing development time and ensuring safer initial trials.
  – RViz, a visualization tool that works with ROS to visualize sensor data and the robot's state in real-time. RViz enables us to monitor the robot's perception of its environment and its planned path, providing valuable insights into its behavior and performance.

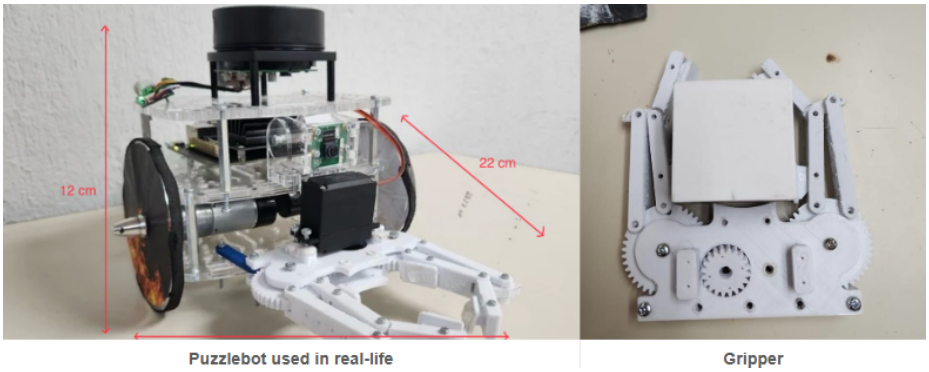

Fig. 1. Puzzlebot an gripper.

3 Mathematical Modeling

3.1 Reference Frames Definition

To have all the elements of the system in the same coordinate frame, we define our fixed frame, which in this case we call ”odom”. We perform a transform from the robot’s base to ”odom” to obtain the coordinates of the Puzzlebot with respect to the ”odom” reference frame. Similarly, we need to perform a transform from the LiDAR to the robot’s base. This way, the point cloud shown by the LiDAR moves as the robot moves, allowing us to map our environment and determine the distances to obstacles. Finally, we perform a transform from Aruco to the center of the camera and from the camera to the base. This allows us to know the position of the Aruco with respect to the robot and its location within the ”odom” frame.

3.2 Kinematic Model of a Differential Robot

A differential drive robot has two independently driven wheels mounted on either side of the robot, which allows it to move and steer by varying the relative speeds of the wheels. The kinematic model describes the relationship between the wheel speeds and the robot’s overall movement.

Equations. The linear and angular velocities of the robot are related to the wheel velocities as follows:

$$v = \frac{v_r + v_l}{2} = r\frac{w_r + w_l}{2}, \tag{1}$$

$$w = \frac{v_r - v_l}{2} = r\frac{w_r - w_l}{l}. \tag{2}$$

The robot’s pose (position and orientation can be represented by (x, y, θ), where:

– (x, y) is the position of the robot in the plane odom.

– θ is the orientation of the robot with respect to the x-axis of odom.

Kinematic Equations. The Kinematic equations that describes the Puzzlebot motion are:

$$\dot{x} = v\cos(\theta), \tag{3}$$

$$\dot{y} = v\sin(\theta), \tag{4}$$

$$\dot{\theta} = w. \tag{5}$$

These equations can be integrated over time to simulate the robot's trajectory given the angular wheel velocities w_r and w_l using the robot encoders. Getting as result:

$$x_k = x_{k-1} + v\cos(\theta) \times dt, \tag{6}$$

$$y_k = y_{k-1} + v\sin(\theta) \times dt, \tag{7}$$

$$\theta_k = \theta_{k-1} + w \times dt. \tag{8}$$

4 Navigation

4.1 Robot Navigation Algorithms

For the robot's navigation, we used two main algorithms: proportional controller to reach a set point, and bug 0 to avoid obstacles [4].

– **Proportional Controller.** We utilized a proportional controller to facilitate movement from point A to point B. This controller regulates the Puzzlebot's linear and angular velocities by calculating the distance an angle error between the starting an destination points. The P controller ensures that the robot can efficiently reach its target by adjusting it speeds based on the proximity and orientation to the goal.

– **Bug0.** The Bug0 algorithm is activated when the Puzzlebot detects an obstacle, such as a wall, using its LiDAR sensor at a minimum distance:

Algorithm 1 Bug 0 algorithm.

```
if Wall then
    Turn 90 degrees
    while not wall do Follow Wall
        if angle aligns with the target point then
            Go to point
            break
        end if
    end while
end if
```

4.2 State Machine

A state machine was designed to control the Puzzlebot actions to guide its behavior through different tasks:

- **Start/Stop:** The robot starts in the initial state, waiting for the ID of the cube and the base where it should be dropped off.
- **Search and pick up the cube:** Upon receiving the cube ID and base location, the robot transitions to searching for the cube. Once the cube is found, the robot approaches and grabs it, switching to the localization state.
- **Localization and Navigation:** In the localization state, the robot activates the P controller to navigate towards the target point. If a wall is encountered, the Bug0 algorithm is triggered, guiding the robot around obstacles until it is aligned to move directly towards the point.
- **Go to base:** Upon reaching the desired point, the robot changes to the state of searching for the base. Once the Aruco marker identifying the base is detected, the robot approaches it, places the cube, and returns to the initial state.

5 Localization

5.1 Odometry

Odometry is a method used to estimate the position and orientation of a mobile robot by tracking the motion of its wheels or other actuators. This technique involves measuring the distance traveled by each wheel and using these measurements to calculate the robot's change in position over time. By continuously monitoring the wheel encoder data, the robot can estimate its current position and orientation as it moves through the environment. However, odometry has its challenges; it is prone to cumulative errors due to wheel slippage, uneven surfaces, and other factors that can introduce inaccuracies over time.

5.2 Equations

To calculate the odometry pose we use the equations 6, 7 and 8. These equations allow us to update the robot's position and orientation over time. Additionally, we utilized ROS2 odom message to publish the odometry data to other nodes. This enables real-time visualization in RViz, where the angle is sent in a quaternion format to ensure accurate orientation representation.

5.3 Map Based Localization

For localization, we relied on map-based techniques to determine the robot's position and orientation using odom frame as out map. We match the sensor data as the LiDAR distances or the ARUCO position to accurately localize itself and navigate effectively in its environment.

To mitigate these errors, odometry is often combined with other sensors and algorithms, such as the Extended Kalman Filter (EKF) and LiDAR, to improve the overall accuracy of the robot's position estimate. This sensor fusion approach leverages the strengths of each sensor, providing a more robust and reliable navigation solution. The integration of odometry with the EKF and other sensory inputs allows the Puzzlebot to navigate autonomously with high precision, even in complex and dynamic environments.

6 Extended Kalman Filter (EKF)

The implementation of the EKF in our system enhances the robot's ability to navigate autonomously and avoid obstacles with high precision. By continuously refining the state estimate, the EKF ensures that the robot's control algorithms receive accurate and up-to-date information, enabling smooth and efficient navigation. This makes the EKF an essential component in achieving robust performance in complex and dynamic environments [3].

6.1 System Linearization and Covariance Matrix

To enhance the accuracy of our odometry, we performed a linearization of the system. This involves approximating the nonlinear system around a specific operating point to simplify the analysis and control design.

Linearization Process. The kalman filter is a very powerful tool optimal for linear systems with gaussian noise. Hoewever in real world there are many non-linear systems, such as the differential mobile robots, which is whu we must linearize the system to apply the kalman filter. The linearization of our system was made by calculating the Jacobian matrices of our systems. We calculate two matrices, one for the position of the robot in the axis x, y and z (equation 10) and another one for the linear and angular position (equation 11) making partial derivates of the system with respect to each of the system's states. In the equation 12 we can see the linearized model:

$$\dot{z} = \begin{cases} \dot{x} = v\cos(\theta), \\ \dot{y} = v\sin(\theta), \\ \theta = w, \end{cases} \tag{9}$$

$$A = \begin{bmatrix} \frac{dV\,\mathrm{Cos}(\theta)}{dx} & \frac{dV\,\mathrm{Cos}(\theta)}{dy} & \frac{dV\,\mathrm{Cos}(\theta)}{d\theta} \\ \frac{dV\,\mathrm{Sin}(\theta)}{dx} & \frac{dV\,\mathrm{Sin}(\theta)}{dy} & \frac{dV\,\mathrm{Sin}(\theta)}{d\theta} \\ \frac{dw}{dx} & \frac{dw}{dy} & \frac{dw}{d\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -V\,\mathrm{Sin}(\theta) \\ 0 & 0 & V\,\mathrm{Cos}(\theta) \\ 0 & 0 & 0 \end{bmatrix}, \tag{10}$$

$$B = \begin{bmatrix} \frac{dV\,\mathrm{Cos}(\theta)}{dv} & \frac{dV\,\mathrm{Cos}(\theta)}{dw} \\ \frac{dV\,\mathrm{Sin}(\theta)}{dv} & \frac{dV\,\mathrm{Sin}(\theta)}{dw} \\ \frac{dw}{dv} & \frac{dw}{dw} \end{bmatrix} = \begin{bmatrix} \mathrm{Cos}(\theta) & 0 \\ \mathrm{Sin}(\theta) & 0 \\ 0 & 1 \end{bmatrix}, \tag{11}$$

$$\dot{z} = \begin{bmatrix} 0 & 0 & -V\,\mathrm{Sin}(\theta) \\ 0 & 0 & V\,\mathrm{Cos}(\theta) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \mathrm{Cos}(\theta) & 0 \\ \mathrm{Sin}(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}. \tag{12}$$

Covariance Matrix Calculation. The covariance matrix Σ_k is crucial for estimating the uncertainty in the robot's position and orientation. It evolves over time based on the system dynamics and measurement updates. It calculation is defined by:

$$\Sigma_k = H_k \Sigma_{k-1} H_k^T + Q_k, \tag{13}$$

where:

– Σ_k is a 3x3 covariance matrix:

$$\begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} \end{bmatrix}. \tag{14}$$

– H_k is a 3x3 Linear model Jacobian of the robot:

$$\begin{bmatrix} 1 & 0 & -\delta t \cdot v_k \cdot \sin(\mu_{\theta,k-1}) \\ 0 & 1 & \delta t \cdot v_k \cdot \cos(\mu_{\theta,k-1}) \\ 0 & 0 & 1 \end{bmatrix}. \tag{15}$$

– Q_k matrix is the nondeterministic error matrix, given by:

$$Q_k = \nabla_{w_k} \cdot \Sigma_{\delta,k} \cdot \nabla_{w_k}^T, \tag{16}$$

where:

$$\Sigma_{\delta,k} = \begin{bmatrix} k_r|w_{r,k}| & 0 \\ 0 & k_l|w_{l,k}| \end{bmatrix}, \tag{17}$$

$$\nabla_{w,k} = \frac{1}{2} r \delta t \begin{bmatrix} \cos(s_{\theta,k-1}) & \cos(s_{\theta,k-1}) \\ \sin(s_{\theta,k-1}) & \sin(s_{\theta,k-1}) \\ \frac{2}{l} & -\frac{2}{l} \end{bmatrix}. \tag{18}$$

6.2 Kalman Filter Algorithm and Equations

For the implementation of the kalman filter, we followed these steps:

– **Position Estimation:** Initially, we estimated the position using odometry calculated from the encoder data. This provided us with an initial estimate of the robot's location and we called it $\hat{\mu}_k$.

– **Linearized model and uncertainty:** Calculate the linearized model (equation 15) to calculate the uncertainty propagation which is calculated by:

$$\hat{\Sigma}_k = H_k \cdot \Sigma_{k-1} \cdot H_k^T + Q_k, \tag{19}$$

where Q_k is the motion model covariance matrix

– **Observation model:** If the Puzzlebot could detect the Aruco marker with the camera, we calculated the distance and angle from the Puzzlebot to the Aruco marker using the camera data which will be our assumed measurements $\hat{z}$. And then we construct the observation model by calculating the distance and angle from the puzzlebot to the aruco marker using the aruco and robot positions:

$$\hat{z}_k = \begin{bmatrix} \hat{z}_{p,k} \\ \hat{z}_{\theta,k} \end{bmatrix} = \begin{bmatrix} \sqrt{\delta x^2 + \delta y^2} \\ \operatorname{atan} 2(\delta y, \delta x) - \hat{s}_{\theta,1} \end{bmatrix}. \tag{20}$$

– **Observation Model Linearization:** The observation model was linearized to facilitate the estimation process. This step involved computing the Jacobian matrix of the observation model:

$$G_k = \begin{bmatrix} -\frac{\delta x}{\sqrt{p}} & -\frac{\delta y}{\sqrt{p}} & 0 \\ \frac{\delta y}{p} & -\frac{\delta x}{p} & -1 \end{bmatrix}. \tag{21}$$

– **Uncertainty Propagation and Kalman Gain Calculation:** We measured the propagation of uncertainty and calculated the Kalman gain. The Kalman gain determines how much weight is given to the prediction and observation updates in the estimation process.

– **Measurement Uncertainty:**

$$Z_k = G_k \cdot \hat{\Sigma}_k \cdot G_k^T + R_k, \tag{22}$$

where, R_k is the observation model covariance matrix.

– **Kalman Gain:**

$$K_k = \hat{\Sigma}_k \cdot G_k^T \cdot Z_k^- 1. \tag{23}$$

Position and Covariance Estimation Finally, we used the Kalman filter to estimate the position and covariance of the Puzzlebot. This iterative process helped us reduce uncertainty and improve the accuracy of the estimated position.

– **Robot position:**

$$\mu_k = \hat{\mu}_k + K_k(z_k - \hat{z}_k). \tag{24}$$

– **Covariance:**

$$\Sigma_k = (I - K_k \cdot G_k) \cdot \hat{\Sigma}_k. \tag{25}$$

7 Vision Algorithm

The use of ArUco markers involves attaching them to target objects that the Puzzlebot needs to identify, approach, and manipulate. By leveraging the ArUco detection library, the robot can accurately determine the position of these markers in real-time, enabling it to perform complex tasks such as picking up and placing objects at designated locations. This technology enhances the robot's ability to interact with its environment in a controlled and predictable manner, ensuring successful task execution.

7.1 Algorithm

For Aruco detection, we utilized a ROS2 library called ros_aruco_opencv. This library provides us with the Aruco ID and the position (x, y, z, theta) relative to the camera frame. To integrate this information into our navigation system, we initially perform a coordinate transformation from the camera frame to the base_link and then to the odometry frame. This allows us to determine the Aruco's position in the map coordinate system. Subsequently, we activate the proportional controller to guide the Puzzlebot towards the Aruco marker, positioning it for cube manipulation tasks.

We used different ArUco markers for different purposes, such as identify different objects and places or to apply kalman filter so the robot's position is more precise. The ArUcos we used are generated in a 4x4 dictionary and a size of 45 mm. We used ArUco ids 1, 2 and 3 to identify the stations where the object must be dropped, the id 6 to identify the cube the robot must grab and the id 7 to apply kalman filter.

8 Manipulator

Drawing inspiration from various online resources and videos, we a suitable gripper model on Thingiverse designed for robots participating in the FIRST Tech Challenge competition. The chosen model served as a foundational template, which we adapted to align with the specific requirements and constraints of the tests. Adjustments were made to accommodate the dimensions and shapes of the ArUco markers, ensuring a snug and secure grip during transportation and positioning tasks.

9 Results

This section presents the results of our autonomous navigation and manipulation robot's performance, comparing its behavior in simulated and real-world environments. Our primary focus is on the robot's stopping distance and the implications of the differences observed due to varying processing capabilities.

9.1 Simulation Performance

In the simulated environment, we employed a high-performance Nvidia RTX 4060 Ti GPU with 4352 CUDA cores and 16 GB of RAM. This powerful setup allowed for rapid and precise computations, enabling the robot to stop accurately at a distance of 30 centimeters from detected obstacles. The simulation environment, managed using ROS 2 and Gazebo, provided a controlled setting where sensor data processing and algorithm execution could occur without significant latency.

9.2 Real-world Performance

Conversely, the real-world tests were conducted using a Jetson Nano with 2 GB of RAM and a GPU featuring only 128 cores. This considerable disparity in processing power impacted the robot's performance, most notably in its stopping distance. The real-world robot consistently crash into the wall and we need to adjust the values so it stopped at 30 centimeters from obstacles, highlighting a discrepancy from the simulation results.

9.3 Analysis of Discrepancies

The observed difference in stopping distances—30 centimeters in simulation versus 30 centimeters in the real world—can be attributed to several factors:

– **Processing Capability:** The Jetson Nano's limited processing power affected real-time sensor data processing and decision-making speed. The RTX 4060 Ti's superior computational capacity allowed for faster and more precise calculations, resulting in a more accurate stopping response.
– **Sensor Data Throughput:** In the simulation, data throughput and processing are optimized by the high-performance GPU, reducing latency in sensor fusion and decision-making processes. In contrast, the Jetson Nano's limited throughput capacity led to slower data processing, contributing to the reduced stopping distance.

Fig. 2. Result of the tests.

– **Algorithm Efficiency:** The real-world implementation faced practical challenges such as sensor noise and physical limitations, which were less pronounced in the controlled simulation environment. These real-world factors necessitated adjustments to the robot's algorithms, further affecting its stopping distance.
– **Environmental Variability:** The real-world environment introduces variability in sensor readings due to factors such as lighting conditions, surface textures, and physical obstacles, which are typically idealized in simulations.

9.4 Performance Implications

The discrepancy between the simulation and real-world performance underscores the importance of considering hardware limitations and environmental factors when deploying robotic systems. While simulations provide valuable insights and a controlled platform for initial algorithm testing, real-world trials are crucial for fine-tuning and validating the robot's performance. Once we identified the discrepancies between the performance in simulation and in the real-world scenarios, we decided to optimize the sensor usage on the robot. Instead of activating all sensors simultaneously, we configured the robot to utilize only the necessary sensors for each specific state of its operation. This approach ensured that the robot could efficiently perform its tasks without overwhelming its limited processing capacity.

Additionally, we reduced the data processing load by minimizing the number of messages being sent and received. By streamlining the communication and focusing on essential data, we achieved a more efficient and responsive system, enhancing the robot's overall performance in real-world applications. Furthermore, the implementation of the Kalman filter and ArUco marker detection significantly improved the precision of the real robot. This enhancement was evident when comparing the simulation results with the real-world outcomes, both before and after optimization 2.

10 Resources

– Demonstrative video (Puzzlebot final challenge): youtu.be/YZyWaMJPywo?si=ayveFs6nnmhTK7tX

– GitHub Repository (Puzzlebot algorithms): github.com/soyhorteconh/Puzzlebot_Lidar_ROS1_ROS2
– Flow Diagrams Folder: drive.google.com/drive/folders/1k8Eu3JkrMkmCKipy-2vJNTDl004k5s8H?usp=sharing

11 Conclusions

The research presented has successfully developed an autonomous navigation and manipulation robot tailored for obstacle-rich environments. By integrating advanced sensors, such as LiDAR and cameras, with robust navigation algorithms, the robot demonstrated efficient and precise operations in complex scenarios.

The use of the Extended Kalman Filter significantly improved the accuracy of the robot's localization, enhancing its ability to navigate and manipulate objects. The implementation of ArUco markers for object identification and target localization proved to be effective, allowing the robot to perform tasks with a high degree of accuracy. Additionally, the differential drive system, combined with the Bug0 and proportional control algorithms, enabled reliable obstacle avoidance and target acquisition. This study underscores the potential of autonomous robots in industrial applications, where they can enhance operational efficiency and safety.

The integration of interdisciplinary technologies and the iterative refinement of both hardware and software components were crucial to the project's success. The findings from this research contribute to the broader field of robotics, offering insights into the development of more advanced and capable autonomous systems for future applications.

References

1. Borenstein, J., Everett, H. R., Feng, L., Wehe, D.: Mobile robot positioning: Sensors and techniques. Journal of Robotic Systems, vol. 14, no. 4, pp. 231–249 (1997)
2. Dudek, G., Jenkin, M.: Computational principles of mobile robotics. Cambridge University Press (2010)
3. Li, Q., Li, R., Ji, K., Dai, W.: Kalman filter and its application. In: Proceedings of the 8th International Conference on Intelligent Networks and Intelligent Systems, pp. 74–77 (2015) doi: 10.1109/icinis.2015.35
4. Lumelsky, V. J., Stepanov, A. A.: Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, vol. 2, no. 1–4, pp. 403–430 (1987) doi: 10.1007/bf01840369
5. Luo, R., Kay, M.: Multisensor integration and fusion in intelligent systems. IEEE Transactions on Systems, Man, and Cybernetics, vol. 19, no. 5, pp. 901–931 (1989) doi: 10.1109/21.44007
6. Parker, L. E.: Current state of the art in distributed autonomous mobile robotics. Distributed Autonomous Robotic Systems, pp. 3–12 (2000) doi: 10.1007/978-4-431-67919-6_1
7. Sharma, N., Pandey, J. K., Mondal, S.: A review of mobile robots: Applications and future prospect. International Journal of Precision Engineering and Manufacturing, vol. 24, no. 9, pp. 1695–1706 (2023) doi: 10.1007/s12541-023-00876-7
8. Siciliano, B., Khatib, O.: Springer handbook of robotics. Springer, 2nd Ed. (2016)

9. Siegwart, R., Nourbakhsh, I. R., Scaramuzza, D.: Introduction to autonomous mobile robots. MIT Press (2011)
10. Tagliavini, L., Colucci, G., Botta, A., Cavallone, P., Baglieri, L., Quaglia, G.: Wheeled mobile robots: State of the art overview and kinematic comparison among three omnidirectional locomotion strategies. Journal of Intelligent and Robotic Systems, vol. 106, no. 3 (2022) doi: 10.1007/s10846-022-01745-7
11. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT Press (2005)