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

Buenaventura, J.G., Salcedo A.M., Leon J.A., Ramírez H.A. & Antunez F.

***Abstract*— This paper presents the development of an autonomous navigation and manipulation robot designed for operation in obstacle-rich environments. The robot is equipped with a differential drive system, a camera, and LiDAR sensors, enabling it to identify and localize target objects marked with ArUco codes. Utilizing advanced navigation algorithms and the Extended Kalman Filter (EKF) for sensor data fusion, the robot accurately calculates the position of the target and navigates towards it while dynamically avoiding obstacles. Upon reaching the target, the robot uses a gripper to securely grasp the object and transport it to a designated unloading zone. The system demonstrates significant potential for automating and optimizing tasks in complex environments, enhancing operational efficiency and safety in industrial applications. The integration of interdisciplinary technologies and iterative testing processes highlights the practical feasibility and effectiveness of the proposed autonomous robotic system.**

I. INTRODUCTION

Autonomous navigation and manipulation in obstacle-rich environments represent critical challenges in the field of robotics. The capability of a robot to navigate autonomously, identify and manipulate objects, and perform tasks without human intervention has vast implications for industrial applications, enhancing efficiency and safety. This paper presents the development and implementation of an autonomous mobile robot designed to operate in complex environments.

The robot is equipped with a differential drive system and integrates a suite of sensors, including a camera and LiDAR, to perceive its surroundings. The navigation system employs advanced algorithms such as the proportional controller and Bug0 for path planning and obstacle avoidance. To enhance the precision of the robot's movements and localization, the Extended Kalman Filter (EKF) is utilized to fuse sensor data and mitigate positional noise.

Central to the robot's functionality is the use of ArUco markers for object identification and localization. These markers provide a reliable means for the robot to detect and position target objects, facilitating accurate manipulation tasks. The robot's design includes a gripper mechanism capable of securely grasping and transporting objects, further extending its application potential in various industrial scenarios.

This research contributes to the advancement of autonomous robotics by demonstrating a comprehensive approach to integrating sensor fusion, navigation algorithms, and manipulation capabilities. The findings highlight the effectiveness of combining multiple technologies to achieve robust and reliable autonomous operation in dynamic and obstacle-rich environments.

II. PUZZLEBOT

A. 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).

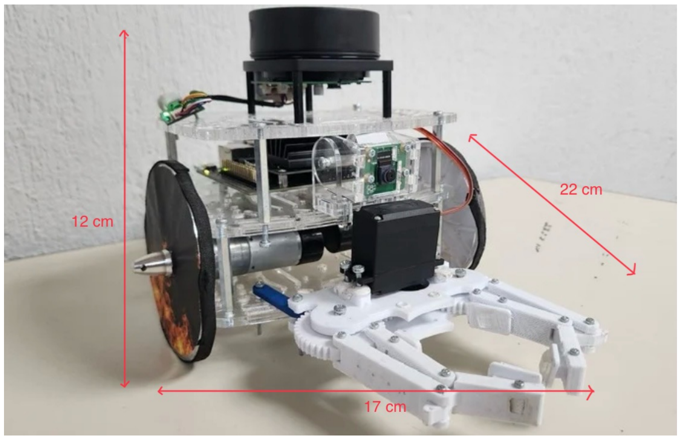


Fig. 1: Puzzlebot dimensions

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. This capability is essential for the robot to make informed decisions and navigate effectively.
- 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.

B. *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.

To better understand the internal workings of the Puzzlebot, a detailed illustration of the robot's circuit is provided below, showing how the various hardware components are interconnected and interact to enable the robot's functionality. (figure 2).

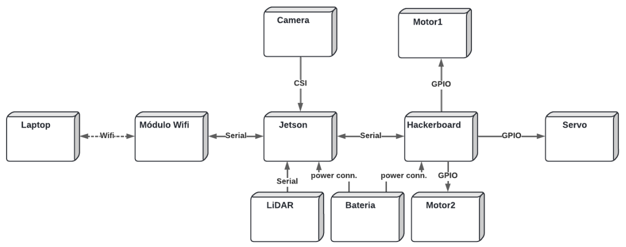


Fig. 2: Hardware robot connection

III. MATHEMATICAL MODELING

A. *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.

The transformation tree and the different frames can be seen in figures 3 and 4 respectively.

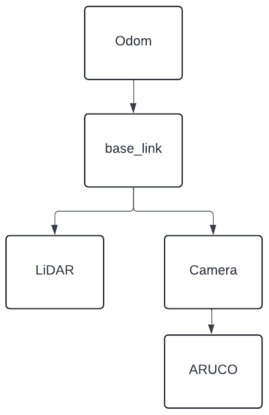


Fig. 3: Transformation Tree

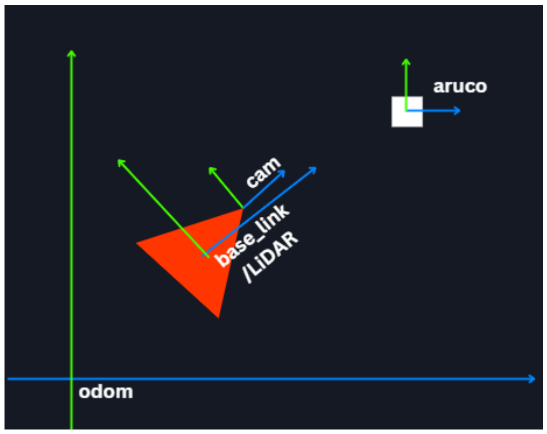


Fig. 4: diagram of frames

B. *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.

1) *Variables and parameters:*

- v: Linear velocity of the robot.
- w: Angular velocity of the robot.
- w_r: Angular velocity of the right wheel.
- w_l: Angular velocity of the left wheel.
- R: Radius of the wheels.
- L: Distance between the two wheels.

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

$$v = \frac{v_r + v_l}{2} = r\frac{w_r + w_l}{2} \quad (1)$$

$$w = \frac{v_r - v_l}{2} = r\frac{w_r - w_l}{l} \quad (2)$$

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.

3) Kinematic Equations: The Kinematic equations that describes the Puzzlebot motion are:

$$\dot{x} = vcos(\theta) \quad (3)$$

$$\dot{y} = vsin(\theta) \quad (4)$$

$$\dot{\theta} = w \quad (5)$$

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:

$$x_k = x_{k-1} + vcos(\theta) * dt \quad (6)$$

$$y_k = y_{k-1} + vsin(\theta) * dt \quad (7)$$

$$\theta_k = \theta_{k-1} + w * dt \quad (8)$$

IV. NAVIGATION

A. 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.

1) Proportional Controller: We utilized a distance wall 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.

2) Bug0: The Bug0 algorithm is activated when the Puzzlebot detects an obstacle, such as a wall using its LiDAR sensor at a minimum distance. Upon detection of an obstacle Bug0 algorithm takes contol.

Bug0 Algorithm is described with following states:

- Wall Detection: When a wall is detected, the robot activates Bug0.
- Initial Turn: The robot turns 90 degrees to the left.
- Wall Following: Using a P controller, the robot maintains a set distance from the wall and follows it.
- Realigment: Once the robot's angle aligns with the direction torwards the target point, it stops following the wall and proceed directly towards the point.
- Reactivation: If another obstacle is encountered, Bug0 is reactivated, and the process repeats until the robot reaches the desired point.

The detailed flowchart of the Bug0 algorithm can be seen in figure 5.

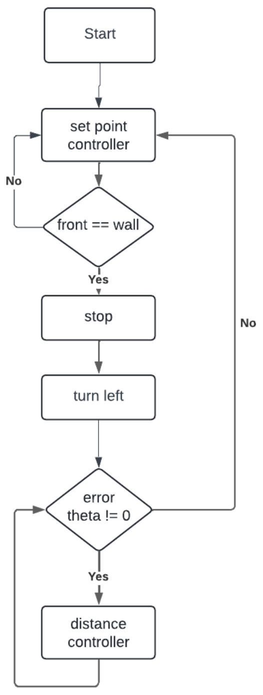


Fig. 5: Bug0 flowchart

B. State Machine

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

1) 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.

2) 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.

3) 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.

4) 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.

The state diagram illustrating these transitions and states can be seen in figure 6.

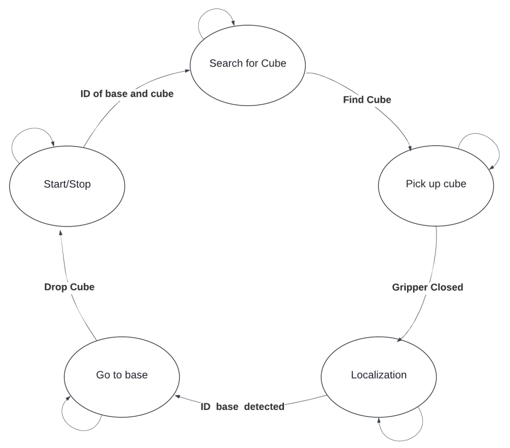


Fig. 6: State Machine Diagram

V. LOCALIZATION

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. Odometry is fundamental in robotics for providing real-time feedback about the robot's movement and for enabling autonomous navigation.

The basic principle of odometry relies on wheel encoders, which are sensors that measure the rotation of the wheels. By knowing the radius of the wheels and counting the number of rotations or partial rotations, the distance traveled by each wheel can be determined. Using this information, the robot's new position and orientation are calculated relative to its previous position. The process involves a series of mathematical computations, including trigonometric functions, to transform the wheel movements into changes in the robot's Cartesian coordinates and heading angle.

In our project, odometry plays a crucial role in the Puzzlebot's navigation system. By continuously monitoring the wheel encoder data, the robot can estimate its current position and orientation as it moves through the environment. This information is vital for tasks such as path planning, obstacle avoidance, and precise maneuvering. However, odometry is not without its challenges; it is prone to cumulative errors due to wheel slippage, uneven surfaces, and other factors that can introduce inaccuracies over time.

A. 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 messages 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.

B. 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. In our project, 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. This filter will be explain in the following section.

VI. EXTENDED KALMAN FILTER (EKF)

The Extended Kalman Filter (EKF) is an advanced mathematical algorithm used for estimating the state of a nonlinear dynamic system from a series of incomplete and noisy measurements. It extends the traditional Kalman Filter, which is applicable only to linear systems, by linearizing the system dynamics around the current estimate. This makes the EKF particularly useful in applications where the underlying system exhibits nonlinear behavior.

The EKF operates by iteratively predicting the system's future state and then updating this prediction based on new measurements. The process involves two main steps: the prediction step and the update step. During the prediction step, the EKF uses the system's model to estimate the next state and its associated uncertainty. In the update step, the filter incorporates the latest measurements to refine the state estimate, reducing the uncertainty. This iterative process allows the EKF to continuously correct the state estimate, making it highly effective for real-time applications.

In the context of robotics, the EKF is widely used for tasks such as localization, navigation, and sensor fusion. For instance, in our project, the EKF is employed to improve the accuracy of the Puzzlebot's position estimation by combining data from various sensors, such as the wheel encoders (odometry) and the LiDAR sensor. By fusing these sensor inputs, the EKF can provide a more reliable estimate of the robot's position, even in the presence of measurement noise and dynamic changes in the environment.

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.

A. 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.

1) 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 11) and another one for the linear and angular position (equation 12) making partial derivates of the system with respect to each of the system's states. In the equation 13 we can see the linearized model.

$$\dot{z} = \begin{cases} \dot{x} = vcos(\theta) \\ \dot{y} = vsin(\theta) \\ \theta = w \end{cases} \tag{9}$$

$$A = \begin{bmatrix} \frac{dVCos(\theta)}{dx} & \frac{dVCos(\theta)}{dy} & \frac{dVCos(\theta)}{d\theta} \\ \frac{dVSin(\theta)}{dx} & \frac{dVSin(\theta)}{dy} & \frac{dVSin(\theta)}{d\theta} \\ \frac{dw}{dx} & \frac{dw}{dy} & \frac{dw}{d\theta} \end{bmatrix} \tag{10}$$

$$A = \begin{bmatrix} 0 & 0 & -VSin(\theta) \\ 0 & 0 & VCos(\theta) \\ 0 & 0 & 0 \end{bmatrix} \tag{11}$$

$$B = \begin{bmatrix} \frac{dVCos(\theta)}{dv} & \frac{dVCos(\theta)}{dw} \\ \frac{dVSin(\theta)}{dv} & \frac{dVSin(\theta)}{dw} \\ \frac{dw}{dv} & \frac{dw}{dw} \end{bmatrix} = \begin{bmatrix} Cos(\theta) & 0 \\ Sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \tag{12}$$

$$\dot{z} = \begin{bmatrix} 0 & 0 & -VSin(\theta) \\ 0 & 0 & VCos(\theta) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} Cos(\theta) & 0 \\ Sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{13}$$

2) 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:

$$\Sigma_k = H_k \Sigma_{k-1} H_k^T + Q_k \tag{14}$$

Where:

- Σ_k is a 3x3 covariance matrix

$$\begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} \end{bmatrix} \tag{15}$$

- H_k is a 3x3 Linear model Jacobian of the robot

$$\begin{bmatrix} 1 & 0 & -\delta t \cdot v_k \cdot sin(\mu_{\theta,k-1} \\ 0 & 1 & \delta t \cdot v_k \cdot cos(\mu_{\theta,k-1} \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

- Q_k matrix is the nondeterministic error matrix, given by

$$Q_k = \nabla_{w_k} \cdot \Sigma_{\delta,k} \cdot \nabla_{w_k}^T \tag{17}$$

Where:

–
$$\Sigma_{\delta,k} = \begin{bmatrix} k_r|w_{r,k}| & 0 \\ 0 & k_l|w_{l,k}| \end{bmatrix} \tag{18}$$

–
$$\nabla_{w,k} = \frac{1}{2} r\delta t \begin{bmatrix} cos(s_{\theta,k-1} & cos(s_{\theta,k-1} \\ sin(s_{\theta,k-1} & sin(s_{\theta,k-1} \\ \frac{2}{l} & -\frac{2}{l} \end{bmatrix} \tag{19}$$

B. Kalman Filter Algorithm and Equations

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

1) 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 $\hat{\mu}_k$.

2) Linearized model and uncertainty: Calculate the linearized model (equation 16) to calculate the uncertainty propagation which is calculated by

$$\hat{\Sigma}_k = H_k \cdot \Sigma_{k-1} \cdot H_k^T + Q_k \tag{20}$$

Where Q_k is the motion model covariance matrix

3) 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.

$$\hat{z}_k = \begin{bmatrix} \hat{z}_{p,k} \\ \hat{z}_{\theta,k} \end{bmatrix} = \begin{bmatrix} \sqrt{\delta x^2 + \delta y^2} \\ atan2(\delta y, \delta x) - \hat{s}_{\theta,1} \end{bmatrix} \tag{21}$$

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

$$G_k = \begin{bmatrix} -\frac{\delta x}{\sqrt{p}} & -\frac{\delta y}{\sqrt{p}} & 0 \\ \frac{\delta y}{p} & -\frac{\delta x}{p} & -1 \end{bmatrix} \tag{22}$$

5) 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:

$$Z_k = G_k \cdot \hat{\Sigma}_k \cdot G_k^T + R_k \tag{23}$$

Where, R_k is the observation model covariance matrix

Kalman Gain:

$$K_k = \hat{\Sigma}_k \cdot G_k^T \cdot Z_k^- 1 \tag{24}$$

6) 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:

$$\mu_k = \hat{\mu}_k + K_k(z_k - \hat{z}_k \tag{25}$$

Covariance:

$$\Sigma_k = (I - K_k \cdot G_k) \cdot \hat{\Sigma}_k \tag{26}$$

In the figures 7 and 8 you can see the flow chart of our kalman filter algorithm and a diagram of the calculations with the puzzlebot.

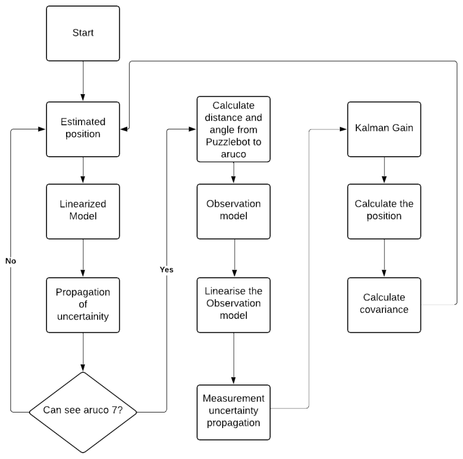

Start
Estimated position
Linearized Model
Propagation of uncertainity
Can see aruco 7?
No
Yes
Calculate distance and angle from Puzzlebot to aruco
Observation model
Linearise the Observation model
Measurement uncertainty propagation
Kalman Gain
Calculate the position
Calculate covariance

Fig. 7: Flow chart of Kalman Algorithm

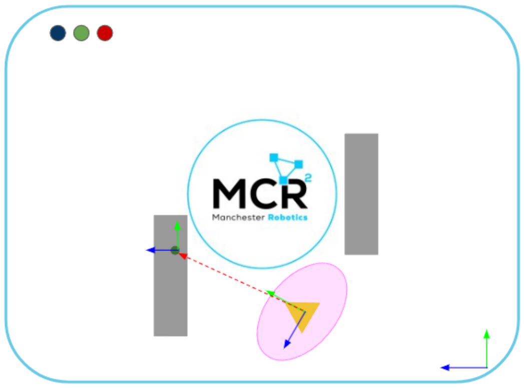


Fig. 8: Digram of kalman calculation

VII. VISION ALGORITHM

An ArUco marker is a type of fiducial marker used in computer vision applications for detecting and estimating the pose of objects. These markers are square patterns with a unique binary code that allows them to be easily identified and distinguished from one another. Each marker is composed of a black-and-white grid, with the binary code embedded within the grid. The high contrast between the black and white areas makes ArUco markers highly reliable for detection under various lighting conditions.

ArUco markers are extensively used in robotics, augmented reality, and camera calibration due to their simplicity and robustness. When placed within the field of view of a camera, these markers can be detected using computer vision algorithms, such as those provided by the OpenCV library. Once detected, the position and orientation of the marker relative to the camera can be estimated with high precision. This capability is particularly useful for tasks that require precise localization and navigation, such as guiding autonomous robots or tracking objects in augmented reality applications.

The use of ArUco markers in our project 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.

A. 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.

The algorithmic workflow for Aruco detection and integration can be observed in the following diagram (figure 12).

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.

VIII. MANIPULATOR

The development of the gripper for the robot involved a meticulous process of mechanical design and prototyping to achieve optimal functionality and performance. Initially, the team embarked on creating a 3D model of the gripper using CAD software such as Creo 5.0 and AutoCAD. The design approach revolved around the concept of a "rack and pinion" mechanism, with the intention of creating a versatile gripper

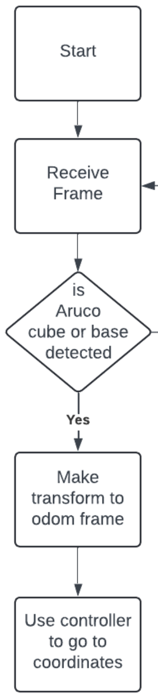
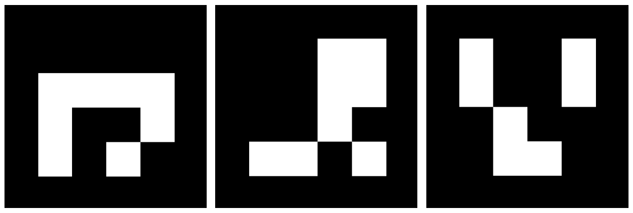

Start
Receive Frame
No
is Aruco cube or base detected
Yes
Make transform to odom frame
Use controller to go to coordinates

Fig. 9: Aruco Visual Servoing



(a) ArUco 1. (b) ArUco 2. (c) ArUco 3.

Fig. 10: ArUcos to identify Stations.

capable of securely grasping and releasing ArUco markers within the track and positioning them as desired.

The iterative design process spanned over multiple iterations, with the team experimenting and refining various aspects of the gripper design. Parameters such as the length of the rack, the overall size, and the length of the gripper (the area where the ArUco markers would be grasped) underwent careful adjustment and modification in pursuit of an efficient and effective gripper design.

To gain a deeper insight into the design process, we will now showcase an image of the team's original 3D model, providing a visual representation of the proposed gripper design (figure 13).

Despite initial efforts, the first iterations of the gripper did not meet the desired performance standards. Upon 3D printing the gripper prototypes using the Ender3 printer, it became evident that the rack and pinion design did not provide sufficient gripping force to securely transport the

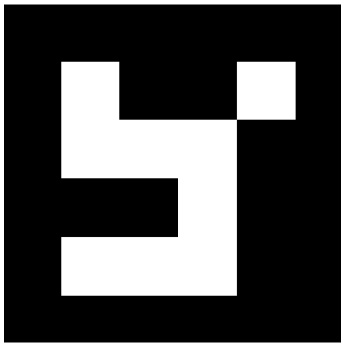


Fig. 11: ArUco to identify cube.

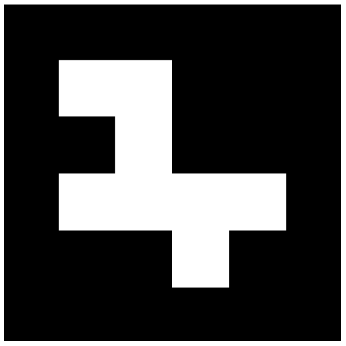


Fig. 12: ArUco to apply kalman filter.

robot without dislodging it. This realization prompted the team to explore alternative gripper designs and solutions.

Drawing inspiration from various online resources and videos, the team pivoted towards a gripper design based on a system of gears and linkages, resembling the functionality of a human hand closing in a circular motion (in the shape of a "C"). This design shift aimed to enhance gripping strength and stability while maintaining adaptability to different object shapes and sizes.

To expedite the prototyping process and leverage existing designs, the team discovered 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 the team modified and customized to align with the specific requirements and constraints of their project. Adjustments were made to accommodate the dimensions and shapes of the ArUco markers, ensuring a snug and secure grip during transportation and positioning tasks.

Below, an image of the proposed gripper, fully disassembled and printed in 3D, is displayed. This visual aid provides insight into the individual components and their arrangement, aiding in comprehension of the gripper's assembly process (figure 14).

The collaborative effort and iterative design approach culminated in the development of a robust and versatile gripper capable of reliably grasping and releasing ArUco markers within the track environment. The final gripper design incorporated elements of adaptability, strength, and precision, empowering the robot to execute its navigation

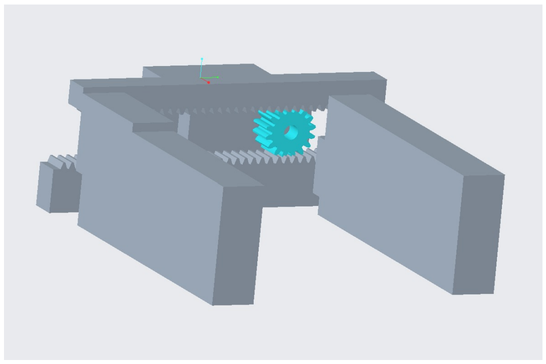


Fig. 13: Design proposal 1

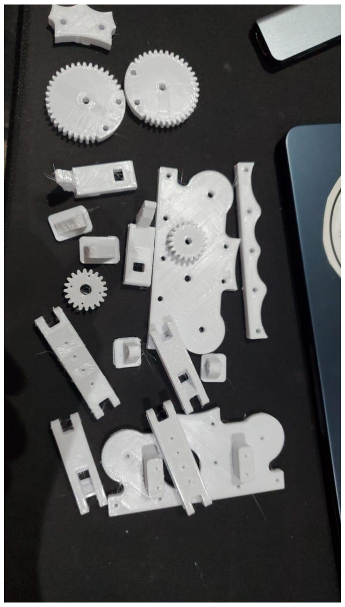


Fig. 14: Disassembled and printed gripper

and manipulation tasks with efficiency and accuracy.

To better grasp the intricacies of the newly proposed gripper, an illustrative image showcasing its design based on FRC robot grippers is presented below. This visualization elucidates how the components integrate and function together, elucidating the gripper's operational mechanism. (figure 15).

IX. RESULTS

The challenge undertaken involved the development and implementation of an autonomous navigation and manipulation system for the Puzzlebot robot. The primary objective was to enable the robot to autonomously navigate through an environment with obstacles, detect and approach target objects marked with ArUco codes, and manipulate them using a gripper. This required the integration of various technologies and algorithms, including sensor data processing, path planning, obstacle avoidance, and object manipulation.

The development of the autonomous navigation and manipulation system relied on a combination of programming tools and libraries, each serving specific functions in the project:

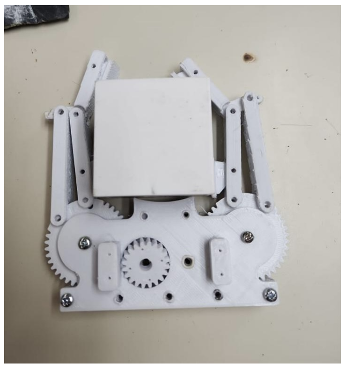


Fig. 15: Final design for the gripper

- **ROS 2 (Robot Operating System 2):** ROS 2 served as the backbone of the software architecture, providing a robust framework for building modular, distributed systems for robotics applications. It facilitated seamless communication between various components of the system, enabling nodes to exchange messages, share sensor data, and coordinate actions in real-time. The flexibility and scalability of ROS 2 allowed for the integration of diverse software modules, simplifying development and enhancing system interoperability.
- **OpenCV (Open Source Computer Vision Library):** OpenCV emerged as a cornerstone in the project's image processing pipeline, empowering the robot with advanced computer vision capabilities. Leveraging a vast array of functions and algorithms, OpenCV enabled tasks such as object detection, feature extraction, image segmentation, and pattern recognition. From identifying ArUco markers to analyzing camera images for obstacle detection, OpenCV played a pivotal role in enhancing the robot's perception and decision-making capabilities in dynamic environments.
- **Python:** Python emerged as the primary programming language for implementing the high-level logic and control algorithms of the autonomous system. Renowned for its simplicity, readability, and extensive library support, Python facilitated rapid prototyping and development of complex software modules. Its rich ecosystem of libraries, including NumPy, SciPy, and Pandas, provided powerful tools for mathematical computation, data manipulation, and algorithm development, enabling seamless integration with ROS 2 and OpenCV.
- **Aruco Detections Library:** The Aruco Detections library emerged as a specialized tool for detecting and decoding ArUco markers, essential for target identification and localization in the robot's environment. Leveraging computer vision techniques, the library offered robust algorithms for detecting marker patterns, estimating their pose relative to the camera, and extracting relevant information for navigation and manipulation tasks. Its efficient implementation and ease of use facilitated seamless integration into the project's software stack,

empowering the robot with precise localization capabilities.

- **LiDAR library:** The LiDAR library played a crucial role in processing data from the LiDAR sensor, enabling the robot to generate accurate maps of its surroundings and detect obstacles in real-time. Leveraging point cloud processing techniques, the library facilitated tasks such as point cloud segmentation, surface reconstruction, and obstacle clustering, providing valuable insights into the robot's environment and enhancing its navigation capabilities. Its integration with ROS 2 streamlined data acquisition and processing, enabling efficient utilization of LiDAR data for path planning and obstacle avoidance.
- **ROS 1 (for simulating Bug 0 and Bug 2):** ROS 1 was employed for simulating classic path planning algorithms, Bug 0 and Bug 2, to evaluate the performance of the autonomous navigation system in simulated environments. By leveraging ROS 1's simulation capabilities, the project team could validate the effectiveness of different navigation strategies, refine algorithm parameters, and optimize system performance before deployment on the physical robot. This simulation-based approach facilitated iterative development and testing, accelerating the refinement of the autonomous navigation system and enhancing its robustness in real-world scenarios.

Despite encountering challenges during the development process, including real-time ArUco detection, gripper construction issues, and occasional difficulties in base detection for placing ArUco markers, the project ultimately achieved its goal of creating a fully functional autonomous navigation and manipulation system.

Through iterative testing and refinement, the system demonstrated robust performance in various environments, successfully navigating around obstacles, accurately detecting and approaching target objects, and manipulating them with the gripper. The integration of advanced algorithms for path planning, obstacle avoidance, and object manipulation allowed the robot to adapt to dynamic environments and execute complex tasks autonomously.

By overcoming technical hurdles and leveraging the capabilities of ROS 2, OpenCV, Python, and other programming tools, the project showcased the feasibility and effectiveness of autonomous robotics systems in real-world applications. The successful completion of the project underscores the importance of interdisciplinary collaboration, innovation, and perseverance in addressing challenges in robotics research and development.

X. Resources

- *Demonstrative video: Puzzlebot final challenge.*
- GitHub Repository: Puzzlebot algorithms

XI. CONCLUSION

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.

ACKNOWLEDGMENT

This work was developed thanks to the support of Manchester Robotics, Tecnológico de Monterrey, and the guidence of our professors Luis Alberto Muñoz Ubando, Arturo Eduardo Cerón Lopez, Jesús Armando Miranda Moya and Alfonso Ávila Ortega.

References

[1] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to Autonomous Mobile Robots (2nd ed.). MIT Press.

[2] Dudek, G., & Jenkin, M. (2010). Computational Principles of Mobile Robotics (2nd ed.). Cambridge University Press.

[3] Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic Robotics. MIT Press.

[4] Borenstein, J., Everett, H. R., Feng, L., & Wehe, D. (1997). Mobile Robot Positioning: Sensors and Techniques. Journal of Robotic Systems, 14(4), 231-249.

[5] Siciliano, B., & Khatib, O. (Eds.). (2016). Springer Handbook of Robotics. (2nd ed.). Springer.

[6] Luo, R. C., & Kay, M. G. (1989). Multisensor Integration and Fusion in Intelligent Systems. IEEE Transactions on Systems, Man, and Cybernetics, 19(5), 901-931.