TE3003B Challenge: Integration of Robotics and Intelligent Systems Team 2

Cano F. Frida (A01752953), Cervantes H. Mario J. (A00711813), Cruz C. Ricardo G. (A01284147), Fernández C. Abiel (A01197654), Ramirez R. Miguel O. (A00828311)

***Abstract*—This project presents the development of a two-wheeled differential drive robot capable of autonomous navigation and task execution using a gripper mechanism. Through the integration of odometry, sensor fusion, and navigation algorithms like Bug 0 and Extended Kalman Filter, the system achieves precise movement and obstacle avoidance. This comprehensive learning experience highlights the potential of autonomous systems in real-world applications, strengthening skills in robotics and preparing for future endeavors. Additionally, the implementation of vital functions such as mapping, localization, and collision avoidance underscores the project's significance in mobile robotics..**

***Index Terms*—Mobile robotics, Differential robot, Odometry, Robotic navigation, Sensor fusion, Kalman Filter, Localization, Gripper, ROS, Python, Puzzlebot.**

I. Introduction

A. Mobile robotics

Mobile robotics refers to the field of technology and engineering that involves the design, manufacturing, and application of robots that can move and operate the robot in simple and complex environments. These robots are capable of navigate autonomously, take decision-making, which allows the robot to interact with their environments with no human intervention. Commonly mobile robots are equipped with different type of sensors, control systems, algorithms that enable the robot performs tasks, such as: *SLAM*, path planning, obstacle avoidance, localization and execute specific tasks.

B. Differential Robot

A differential robot is a two-wheeled robot with each wheel driven separately. This mechanical configuration, with independent control for each wheel, allows the robot to maneuver by varying the speed and direction of each wheel.

C. Applications of differential robots

We will indicate 5 applications of differential robots specifically in load-carrying capabilities: *1.Warehouse Automation*: d.robots can navigate through aisles for picking, delivery and dropping tasks to specific locations, enhancing efficiency in the inventory management and order fulfilment. *2.Materials Handling*: differential robots are used in manufacturing processes for moving raw material, working process items and finished products over the shop-floor. *3.Agricultural Load Transport:* Differential robots are very useful in load-carrying harvested crops, fertilizers. *4.Construction Site Logistics:* on construction sites differential robots can transport construction materials, tools. And reduce manual work for workers. *5.Hospital Supply Delivery:* in the healthcare industry, these robots can transport medical supplies, medications. Robots can navigate into elevators and hallways.

D. Definition and Application of Odometry

Odometry is a technique used in robotics and navigation to estimate the position and orientation [pose] of a robot. These data is obtained from sensors, wheel encoders, acelerometers, gyroscopes, IMUs. Some applications of the odometry are: 1. Autonomous Navigation: as explained before, odometry provides the postion and orientation of a robot as it moves. Also, it is crucial for the path planning, obstacle avoidance and to reach the target goal. 2. Mapping and localization: Odometry is crucial for the Simultaneous Localization and Mapping (SLAM) algorithms. 3. Aerial and Underwater drones: odometry is crucial to keep stable flight in the drones and for UUV due to there is no GPS, odometry provide total data for odometry to explore underwater.

E. Definition and applications of robotic navigation

This is the methodology and technology that integrates many concepts explained before. Robotic navigation the integration of different sensors, algorithms and control systems that enable the robot to read its environment, make decisions, and execute movements with a specific tasks and targets while avoiding obstacles and adapting dynamics conditions. Some applications are the following: *1. Autonomous Vehicles*: it could be cars, trucks, trains, UAVs, UUV use advanced navigation systems to move from one location to another without human intervention. *2.Warehouse Automation*: AGVs and AMRS (Automated Guide Vehicles) and (Autonomous Mobile Robots) *3.Other applications are Service Robots, Exploration and Search and Rescue, Healthcare Robotics.*

F. Sensor Fusion

Sensor fusion is a process that combine the data from several sensors to obtained more accurate, reliable and understanding of an environment/system robot. The goal is to leverage the strengths of each sensor, enhancing the robots perception and making decision capabilities. The process for sensor fusion is as follows: *1. Data collection:* Multiple sensors data is collected; cameras, LIDAR, IMUS. For instance they provide visual information, distances to objects and accelerations/orientation. *2. Reprocessing:* initially sensors provide raw data, then data needs to reprocessing to filter out noise, correct errors and align the data in a consistent and usable way. *3. Data Integration:* Sensor fusion algorithms are used to integrate all data from sensors. One of the most common filtering algorithms is Kalman Filter, Extended Kalman Filter [EKF], particle filter, deep learning methods.These algorithms combine data to produce more accurate and robust estimation of the robot states. *4. State Estimation:* all this includes determining the robot's precise location, mapping its environment, detecting obstacles, and understanding its orientation in real time. *5. Decision making:* Once the data is better obtained, it allows the robot to take informed decisions.

G. Description of the project

This document describes the complete functionality and development of a two-wheeled robot that navigates autonomously. We present the mathematical definition of odometry, the navigation algorithm, the vision algorithm, and the Extended Kalman Filter. We also explain the robot's capabilities in navigating and executing a specific task using a gripper mechanism to hold a cube. The main goal of the robot is to search for a cube, pick it up, and drop it in a specific BASE as indicated by a letter [A,B,C]

II. PUZZLEBOT

A. Puzzlebot description

Puzzlebot is a universal tool for robotics, to help others learn, create and innovate their own robotics projects. There are four versions:

1) The Hacker Edition
2) The Laser Edition
3) NVIDIA Jetson Edition
4) NVIDIA Jetson/LiDAR Edition

For this project, we used the Jetson/Lidar Edition.
This version is an extension of the Puzzlebot Hacker Edition, therefore, we contemplated these components; hacker edition contains all the essential components needed to access meaningful robotics capabilities quickly providing a user-friendly platform for incorporating a wide range of advanced add-on feature sets. Powered by the Hacker Board for algorithms which require real-time processing capabilities. The Puzzlebot NVIDIA Jetson/LiDAR encompasses an NVIDIA Jetson CPU, a Raspberry Pi Camera and a LiDAR, this allows users to implement research-level, real-time algorithms such as AI & Computer Vision, obstacle avoidance and SLAM and autonomous driving algorithms using ROS.
The construction of the robot consists of 2 floors where the motors, wheels, hackerboard, jetson and camera are mounted.1

B. Hardware and software description

As previously mentioned, the main hardware components that make up the Puzzlebot are:

- Hackerboard
- Jetson
- LiDAR
- Camera

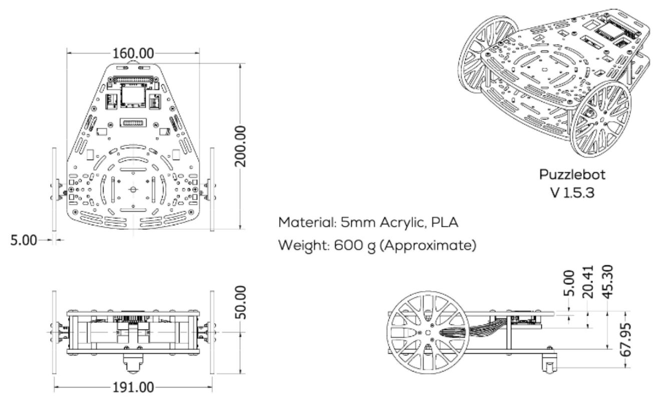

Fig. 1. Puzzlebot general dimensions (mm)

- DC Motors
- Power supply

Additionally we used an adapted power supply to feed our Puzzlebot through cables and modulators to ensure that it does not exceed the current required for the system and there are no malfunctions, this implementation made it easier for us to use the robot and perform the necessary tests at all times. As for the software running on the NVIDIA Jetson Nano:

- Jetpack 4.6
- ROS Humble
- OpenCV for ArUco detection

III. MATHEMATICAL MODEL

A. Contextual Definition

When we refer to a differential drive robot, in our case, it means it has two independently driven wheels located on either side of the robot body. The kinematic model of such a robot describes the relationship between the control inputs which are given by robot motion (linear and angular velocities). It does not include any external force, just velocities. The following is an explanation of the kinematic model:

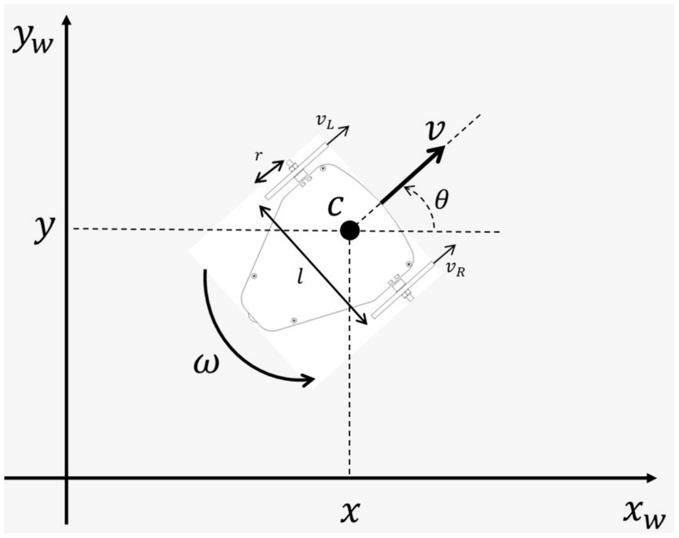

Fig. 2. Differential drive schematic

B. Kinematic Model

C. Notation and Parameters

v_R : Linear velocity of the right wheel
v_L : Linear velocity of the left wheel
r : Radius of the wheels
L : Distance between the two wheels (wheelbase)
v : Linear velocity of the robot
(center point between the wheels)
ω : Angular velocity of the robot
(about its center point)
θ : Orientation of the robot
with respect to a fixed reference frame
x, y : Position coordinates of the robot
in the fixed reference frame
Δt : Delta time

1. Linear and angular velocities equations

$$v = \frac{V_R + V_L}{2} \tag{1}$$
$$\omega = \frac{V_R + V_L}{L} \tag{2}$$
$$v = \frac{(\omega_R + \omega L)r}{2} \tag{3}$$
$$\omega = \frac{r(\omega_R - \omega L)}{l} \tag{4}$$
$$\tag{5}$$

2.- Wheel Velocities:

$$V_R = v + \frac{L}{2} \tag{6}$$
$$V_L = v - \frac{L}{2}\omega \tag{7}$$

3.- Pose of the Robot:

$$\dot{x} = v\cos\theta \tag{8}$$
$$\dot{y} = v\sin\theta \tag{9}$$

4.- Odometry:

$$x(t+\Delta t) = x(t) + v\cos(\theta)\Delta t \tag{10}$$
$$y(t+\Delta t) = y(t) + v\sin(\theta)\Delta t \tag{11}$$
$$\theta(t+\Delta t) = \theta(t) + \omega\Delta t \tag{12}$$

D. Explanation of equations

Linear Velocity v: This is the average of the velocities of the two wheels. When both wheels move at the same speed, the robot moves straight.
Angular Velocity ω: This is the difference in velocities of the two wheels divided by the distance between them. When the wheels move at different speeds, the robot rotates.
Pose Update: The pose update equations use the current velocities to compute the new position and orientation of the robot over a small time step Δt.

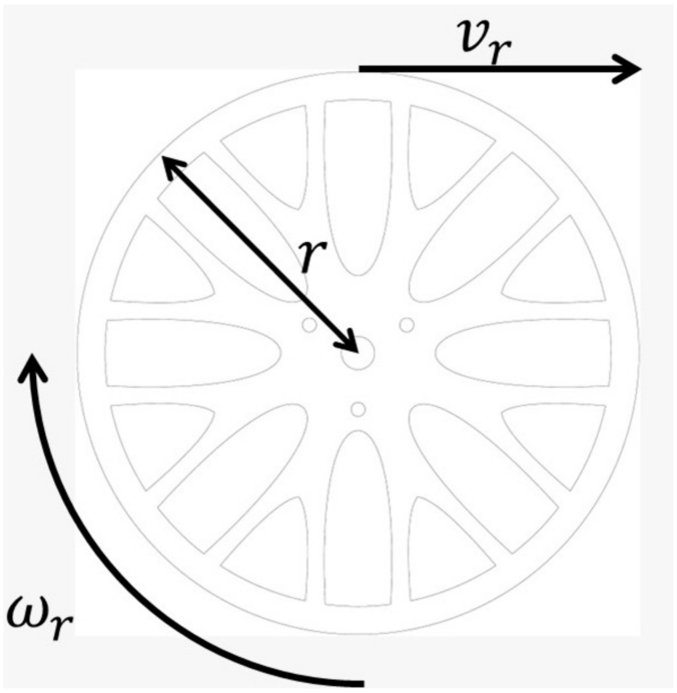

Fig. 3. Wheel nomenclature

E. Linearization Matrix to obtaine robot pose

$$\mathbf{v}, \omega = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

These matrix is equal to the following expression

$$\mathbf{v}, \omega = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}$$

Finally to obtain the robot **ODOMETRY (x,y,θ)** we integrate the matrix

$$\int \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} dt$$

IV. Behaviour Planning

To implement the mapping, searching, gripping and delivery behaviour, we created a node that can keep track of all possible states and change between them accordingly. The main states: MAPPING, MOVE_TO_POINT, GRIP, RETURN_TO_HOME. At the MAPPING state, the robot moves around the map while avoiding forward-facing obstacles. Once a cube is seen, the GRIP state is used and the algorithm VII-B.

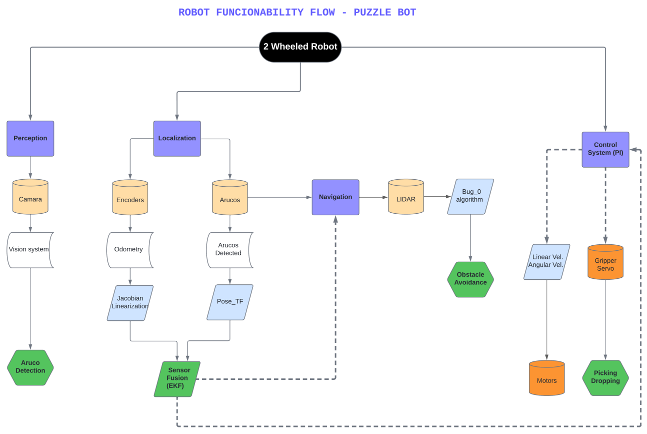

Fig. 4. Overview functionality

V. Navigation

A. Description of navigation algorithm

For the navigation of Puzzle bot we implement an algorithm to avoid obstacles. The one selected was the bug 0 algorithm which is a simple yet effective approach for robot navigation in environments with obstacles. This algorithm relies on two primary behaviors: moving toward the target and wall-following. The robot starts by moving directly towards the target, and if it encounters an obstacle, it switches to a wall-following behavior sliding along the obstacle boundary in left or right direction (just one) until it can resume its path towards the target. For better performance we determined 3 sections in LiDAR sensor to decide which are the best movements for the robot in case it detects an obstacle from its front, left or right side.

B. Robot Behaviour

The robot maintains a state variable to switch between different behaviors (moving towards the target, wall-following and stopping).
The logic used for algorithm consist in 4 stages:

- Move towards the target
  Initially, the robot moves directly towards the goal using a simple proportional controller that adjusts its linear and angular velocities based on the distance and angle to the target, and continuously calculates the error in the x and y coordinate relative to the target position and adjusts its movement to minimize the error.
- Obstacle detection
  As the robot moves towards the target, it uses a laser scanner (LiDAR) to detect obstacles in its path. If the robot detects an obstacle within a specified distance (0.2 m), it switches from the target-following behavior to wall-following behavior.
- Wall-following behavior
  In wall-following mode, the robot navigates around the obstacle by maintaining a certain distance from the wall while continuously monitoring the distances to the obstacles on its left, front, and right sides. Based on these distances, the robot decides whether to turn left, turn right, or move forward to navigate around the obstacle. Once the robot has navigated around the obstacle and the path to the target is clear, it switches back to moving directly towards the target.
- Stop
  If the robot reaches the target position within a specified tolerance, it stops.

5

C. Kalman Filter

Kalman Filters (KF) can be used for filtering and predicting a linear system's predicted mean μ_t and covariance Σ_t. This makes it such that a system's state can be predicted given it's control inputs and sensor measurements to perform corrections. The "uncertainty" in the predicted state can be determined from the covariance matrix Σ_t, where a larger Σ_t represents a higher uncertainty in the current predicted state. Given a control signal u_t and measurement z_t, belief at t is estimated by 13 and 14 by incorporating the control signal u_t. Measurement z_t is incorporated in equations 15, 16 and 17. Here, a Kalman gain K_t in 15 is computed, where the degree to which the measurement will be incorporated into the new estimated state. In 16 the mean μ_t is modified proportional to the Kalman gain K_t and the difference between predicted state and measured state. The new covariance of the belief is calculated in equation 17 [1].

$$\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t \tag{13}$$
$$\bar{\Sigma} = A_t\Sigma_{t-1}A_t^T + R_t \tag{14}$$
$$K_t = \bar{\Sigma}_t C_t^T (C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1} \tag{15}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t) \tag{16}$$
$$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t \tag{17}$$

D. Extended Kalman Filter

The Extended Kalman Filter (EKF) is a modification of the Kalman Filter such that it can handle non-linear systems using

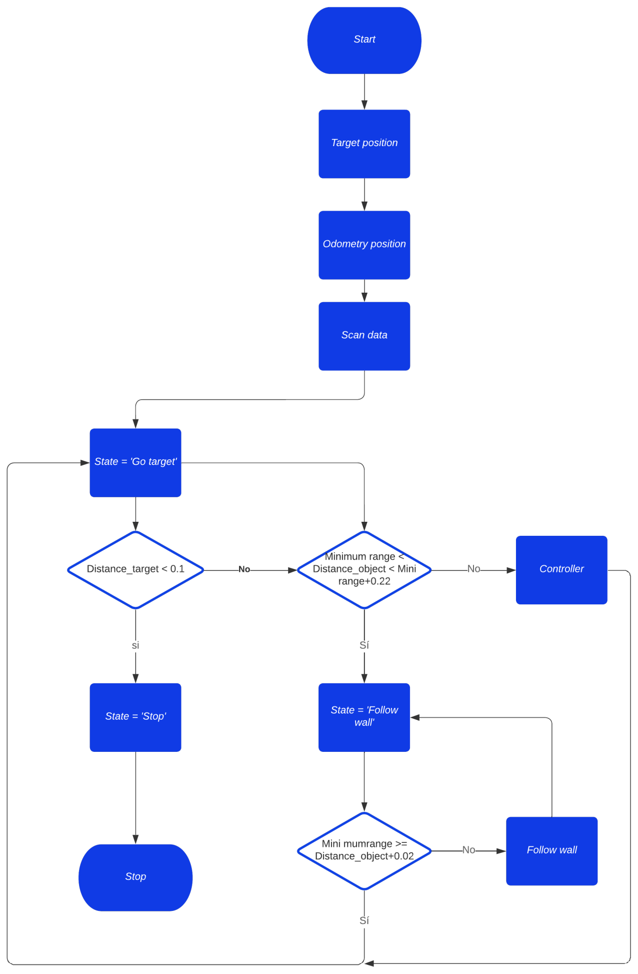

Fig. 5. Flow chart of navigation algorithm

a linearization around the estimated state and covariance. State prediction 18 is done with a non-linear model g, as opposed to the linear prediction done in Kalman Filters 13. Jacobians G_t and H_t are used in place of the linear system matrices used in Kalman filters [1].

$$\hat{\mu}_t = g(u_t, \mu_{t-1}) \tag{18}$$

$$\hat{\Sigma}_t = G_t \Sigma t - 1 G_t^T + R_t \tag{19}$$

$$K_t = \hat{\Sigma}_t H_t^T (H_t \hat{\Sigma}_t H_t^T + Q_t)^{-1} \tag{20}$$

$$\mu_t = \hat{\mu}_t + K_t (z_t - h(\hat{mu}_t)) \tag{21}$$

$$\Sigma_t = (I - K_t H_t) \hat{\Sigma}_t \tag{22}$$

1) EKF Simultaneous Localization and Mapping with known correspondences: We implemented the algorithm described in [1] for simultaneous localization and mapping (SLAM). For our features, we can use the ArUco markers, as we can identify each tag by it's unique ID. This can be done by having a state estimation of robot pose x_t with a state estimation of all tags encountered along the way. This makes it such that our state x contains robot state x, y, θ and position state for each tag x_t, y_t. The implementation of the EKF is as described in V-D, where encoder measurements are used to create a new mean prediction 18, and the sensor measurement's range and bearing are integrated into state depending on which tag is being seen.

This makes it so that our Puzzlebot is able to map where each ArUco tag is, and localize itself in the environment.

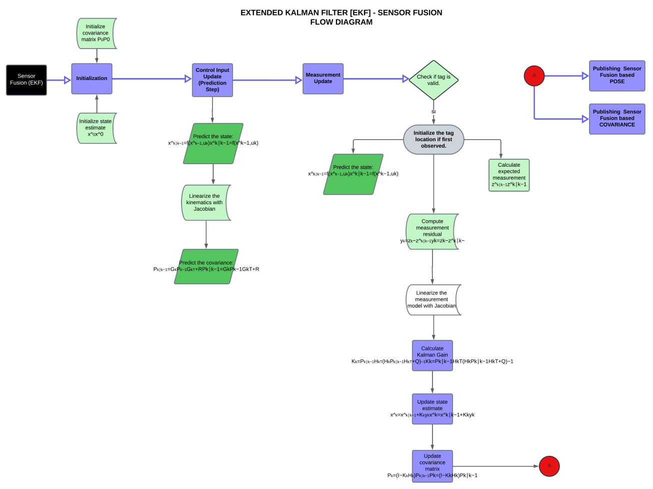
EXTENDED KALMAN FILTER [EKF] - SENSOR FUSION FLOW DIAGRAM
Initialize covariance matrix P₀P0
Sensor Fusion (EKF)
Initialization
Initialize state estimate x^₀x^0
Control Input Update (Prediction Step)
Predict the state: x^ₖ|ₖ₋₁=f(x^ₖ₋₁,uₖ)x^k|k−1=f(x^k−1,uk)
Linearize the kinematics with Jacobian
Predict the covariance: Pₖ|ₖ₋₁=GₖPₖ₋₁Gₖᵀ+RPk|k−1=GkPk−1GkT+R
Measurement Update
Check if tag is valid.
Si
Initialize the tag location if first observed.
Predict the state: x^ₖ|ₖ₋₁=f(x^ₖ₋₁,uₖ)x^k|k−1=f(x^k−1,uk)
Calculate expected measurement z^ₖ|ₖ₋₁z^k|k−1
Compute measurement residual yₖ=zₖ−z^ₖ|ₖ₋₁yk=zk−z^k|k−
Linearize the measurement model with Jacobian
Calculate Kalman Gain Kₖ=Pₖ|ₖ₋₁Hₖᵀ(HₖPₖ|ₖ₋₁Hₖᵀ+Q)₋₁Kk=Pk|k−1HkT(HkPk|k−1HkT+Q)−1
Update state estimate x^ₖ=x^ₖ|ₖ₋₁+Kₖyₖx^k=x^k|k−1+Kkyk
Update covariance matrix Pₖ=(I−KₖHₖ)Pₖ|ₖ₋₁Pk=(I−KkHk)Pk|k−1
A
A
Publishing Sensor Fusion based POSE
Publishing Sensor Fusion based COVARIANCE

Fig. 6. Flow chart EKF

VI. Localization

A. Odometry definition

The general definition of odometry is the use of data from motion sensors, such as wheel encoders, to estimate change in robot's position over time. Odometry is used by the Puzzlebot to estimate its position and orientation relative to its starting location given in terms of an x and y position around the z (upward) axis as the Puzzlebot moves.[2]

B. Equations applied for odometry

For the calculation of the odometry, we used the velocity data from the Puzzlebot's wheels. By obtaining the current wheel velocities and applying mathematical formulas, we calculated the robot's linear and angular velocity. Once we had these values, we determined the robot's current pose by multiplying the velocity by a time differential and the sine or cosine of the angle, depending on the axis in question.

$$v = \frac{r * (w_r + w_l}{2}) \tag{23}$$

$$\omega = \frac{r * (w_r - w_l)}{L} \tag{24}$$

$$x_t = x_{t-1} + cos(\theta) * v * \Delta t \tag{25}$$

$$y_t = y_{t-1} + sin(\theta) * v * \Delta t \tag{26}$$

$$\theta_t = \theta_{t-1} + \omega * \Delta t \tag{27}$$

C. *Map based localization*

Initially, the robot's odometry starts at zero, taking the starting position as the origin because it is the first time the program runs. The localization is based on the detection of markers around the track. With respect to the determined positions of these markers, the robot approximates its position on the map. Once the track mapping is completed, the robot is ready to navigate from point A to the desired point B.

VII. Vision algorithm

A. *ArUco definition*

An ArUco marker is a square marker developed by the University of Cordoba for computer vision purposes. This square is composed of a binary black and white matrix with a black border. Each square of the matrix represents a bit, meaning that a 5x5 matrix is composed of 25 bits. Each unique array of bits belongs to a specific ID and multiple ArUco dictionaries exist. For ArUco detection, the OpenCV ArUco library is utilized. The cv2.aruco.detectMarkers() function is utilized for marker detection. This function operates in the following manner:

1) Marker candidates are found by analyzing the image and finding square shapes. This is done by segmentation and contour extraction.
2) Then, its inner codification is considered to ensure that it is a marker. Their bits are extracted by separating the black and white cells.

The black borders and binary codification of the markers facilitate their detection. Since the contours are extracted from the markers, their corresponding corners are known allowing for a Perspective-n-Point (PnP) pose computation. A PnP pose computation is a technique utilized for determining the pose of an object relative to the camera based on a set of 3D points and their 2D projections in the camera's image. This calculation results in the translation and rotation vectors of the camera in relation to the object. This method is applied with the OpenCV function cv2.solvePnP() where the corners, the marker size and the camera distortion coefficients are needed.[3]

B. *Detection and approximation algorithm*

The behaviour for the detection, alignment and gripping node is described in figure 7. Here, the robot aligns with the target's aruco tag. Once aligned in the image, the puzzlebot approaches the cube and sends the servo close command.

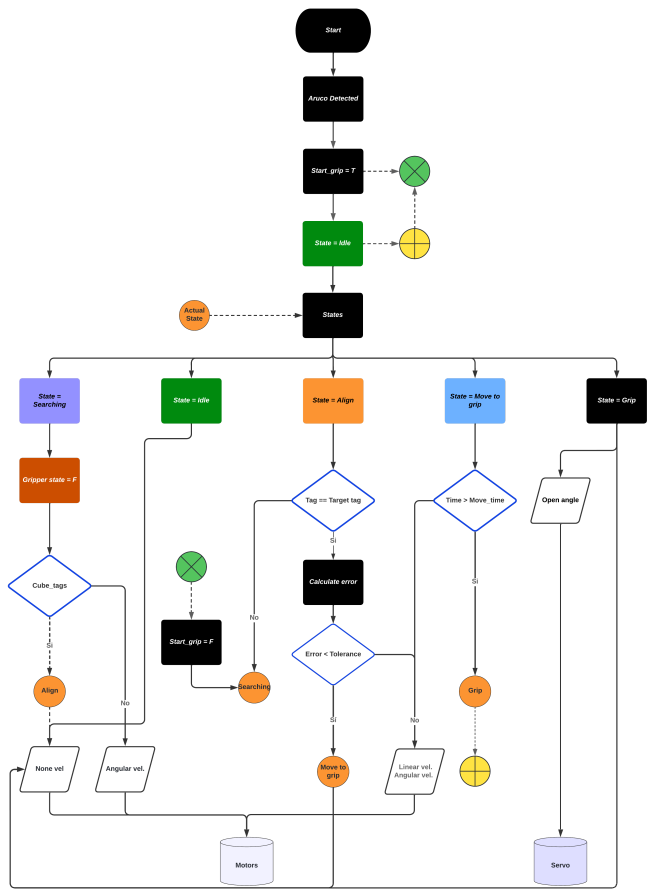

Fig. 7. Flow chart strategy vision

VIII. Hardware Integration - Gripper

The robot must securely pick up the cube and transport it to the goal point without human intervention. To ensure that and according to the hardware limitations we decided to design a gripper that utilize one servo-motor connected to the Hackerboard and manipulated through a ROS2 topic (*/ServoAngle*). For that we decided to design a gripper, here's an overview of the design process and the challenges we faced:

A. *Mechanical design description*

The gripper was designed on OnShape [4] web 3D design platform and 3D printed with PLA (Polylactic Acid) due to its strength, durability, and ease of printing. The assembly is conformed by four important parts: the base, the links, the grippers and the center circle that links the servo with the gripper 8.

IX. Prototyping and iterations: Gripper

A. *Size and Fit*

Our initial designs encountered a significant issue: the gripper was not large enough to wrap around the cube when it was placed diagonally. This resulted in the gripper failing to securely grasp the cube, leading to instability during transportation 9.

B. *Iterative Improvements*

To address this issue, we revisited our design and increased the dimensions of the gripper. This adjustment ensured that the gripper had a better tolerance and wider aperture, allowing it to wrap around the cube more effectively, even when the cube was oriented diagonally 10.

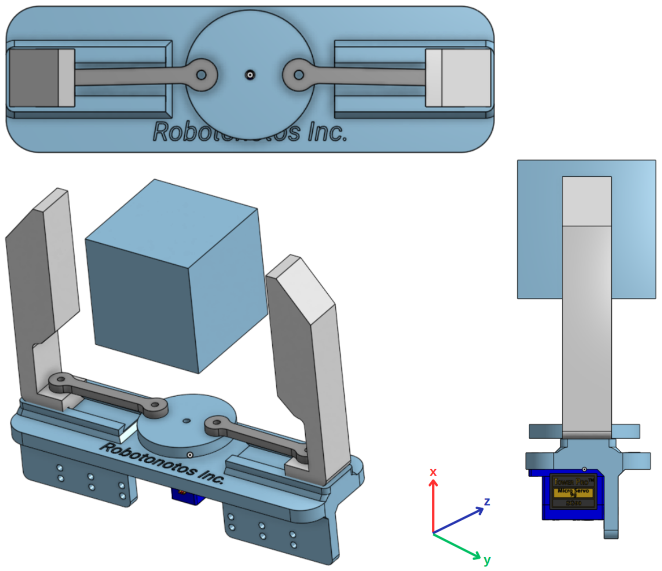

Fig. 8. Gripper Assembly

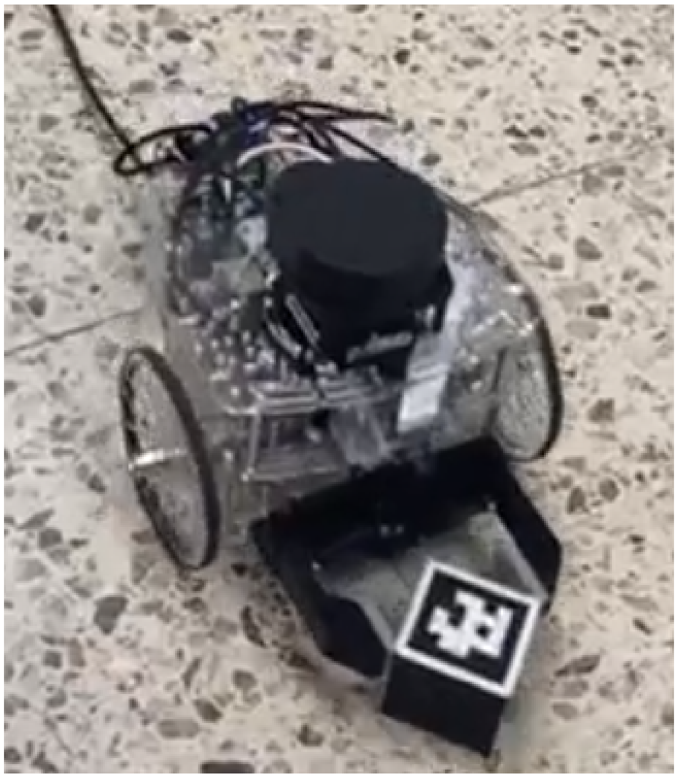

Fig. 9. Gripper with size issue

X. Results

A. Experiment Performed

For this experiment, we started the Puzzlebot in a random position within our testing environment. During the first 40 seconds, we do not place the cube down, such that the robot has time to map where the AruCo tags are within it's environment. Afterwards, we placed the cube in a random location, giving the Puzzlebot the opportunity to search and grip it. We had defined a predetermined a delivery location and home location.

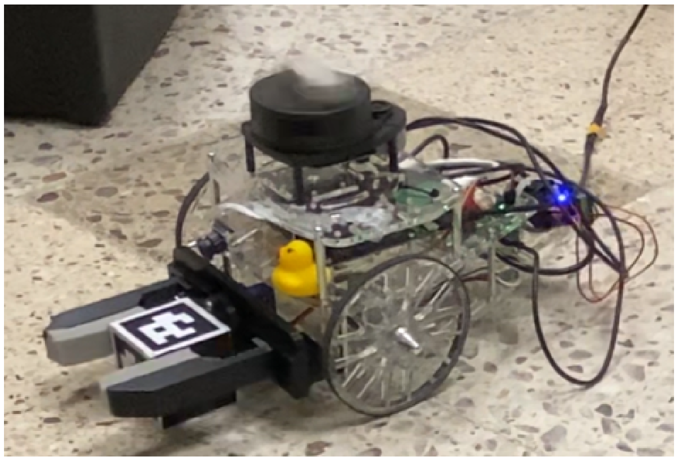

Fig. 10. Final iteration

B. Software Used

For this project, we worked with a Jetson Nano running Ubuntu 20.04. We used ROS2 Humble, and developed our code in Python. We used the NumPy library for matrix maths and OpenCV for image processing and AruCo detections.

C. Results Obtained

With our implementation, we were able to successfully complete the challenge. With our behavior node, the Puzzlebot is able to autonomously change its state depending on the current status of the environment. The Puzzlebot was able to map its environment using EKF SLAM V-D1, manipulate the cube with our gripper VIII, and bring it to the desired location. A demo can be seen in: https://youtu.be/_EVcV9XmOeo. The repository for the source code is: https://github.com/romi2002/TE3003B/tree/puzzlebot_final_challenge.

XI. Personal conclusions

- Miguel O. Ramirez R.
  In this project, we successfully designed, developed, and tested a two-wheeled differential drive robot capable of autonomous navigation and performing tasks using a gripper mechanism. By integrating odometry, sensor fusion, and robotic navigation algorithms, we achieved a robust system for precise movement and task execution in dynamic environments. We implemented the Bug 0 navigation algorithm for obstacle avoidance, allowing the robot to maneuver around obstacles while maintaining a path towards the target. The Extended Kalman Filter (EKF) enhanced the robot's accuracy in positioning and orientation.
- Frida Cano F.
  In conclusion, the development of Puzzlebot was a comprehensive learning experience that allowed us to delve into various aspects of robotics, including sensor integration, localization, and autonomous navigation. Our project successfully demonstrated the ability to autonomously detect, localize, and transport an object to a specified goal, showcasing the potential of autonomous systems in real-world applications. The challenges we faced and overcame have strengthened our understanding and skills in robotics, preparing us for future endeavors in the field.
- Ricardo G. Cruz C.
  For this project we developed some of the core functionalities of mobile robotics, a system that can perform mapping and localization. A variety of methods were implemented, sensor fusion, localization with PnP and ArUcos, obstacle mapping with a LiDAR, Extended Kalman Filter, etc. All these methods converged for the achievement of one of the main goals, collision avoidance. Also, the implementation of gripper for delivering a payload was a nice showcase of an application of an autonomous robot in real-world scenarios. Other than broadening our technical expertise, this experience provided tremendous insight into the complexities of system integration, multidisciplinary collaboration and problem-solving.
- Abiel Fernandez C.
  With this project, I was able to apply what I had learned in class, by developing a complete solution for a two-wheeled differential drive robot, giving it autonomous capabilities. I found it interesting to learn how to implement an EKF algorithm for localization, as I had always heard of it but I had never gotten the opportunity to use it. Another aspect of the project that I enjoyed, was designing the gripper for the puzzlebot, as I was able to strengthen my CAD skills by iterating over a design until we found one that worked well.
- Mario J. Cervantes Hernandez
  Working in an interdisciplinary team with different competencies represents the most valuable aspect of this project to me; each of us has special hard and soft skills which are invaluable in making things happen in this challenge. Secondly, I want to thank all my team members because they truly helped me to make this project possible and were constantly teaching me throughout every mini-challenge we encountered during the project. I am proud of my team. Finally, this project gave me the chance to gain a better understanding about differential robots and also clarified how robot architecture is deployed and developed step by step, solving both hardware and software algorithms. It was fascinating for me to see how the Kalman Filter process significantly improves sensor fusion techniques, enhancing a robot's functionality and performance. Additionally, I realized that when we are under pressure, things can be solved in just a few hours, and I believe this is the speed at which we should work for future projects. Thank you!.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2006.

[2] C. Fairchild and D. T. L. Harman, *ROS Robotics By Example - Second Edition: Learning How to Build and Program Your Own Robots with the Most Popular Open Source Robotics Programming Framework*, 2nd ed. Packt Publishing, 2017.

[3] G. Bradski, “The opencv library,” *Dr. Dobb's Journal of Software Tools*, 2000.

[4] P. T. Corporation, “Onshape,” https://www.onshape.com/, 2012.