Integración de Robótica y Sistemas Inteligentes Tecnológico de Monterrey

José Pablo Cedano Serna
A00832019

Rodrigo Escandón López Guerrero
A01704287

Luis Antonio Zermeño De Gorordo
A01781835

Luis Mario Lozoya Chairez
A00833364

***Abstract*—This paper presents the integration of computer vision, autonomous navigation, and AI methods with a voice-controlled forklift system and a web interface on a non-holonomic differential-drive mobile robot. The system is built on ROS 2 and comprises modular software nodes for perception, planning, control, and user interaction running on a Jetson Nano and an ESP32-based microcontroller. We detail the architecture of nodes, communication topics, and data flows between sensors, control modules, and the live web dashboard.**

***Index Terms*—Autonomous Navigation, ROS 2, SLAM, Voice Recognition**

NOMENCLATURE

ROS 2	Robot Operating System 2
SLAM	Simultaneous Localization and Mapping

I. THEORETICAL FRAMEWORK

A. ROS 2 as an Integration Platform

ROS 2 serves as the backbone for distributed real-time communication among all robot components. Each sensor, processor, and actuator runs as an independent *node*, publishing and subscribing to data streams called *topics*. Configurable Quality of Service (QoS) policies over DDS ensure reliable, low-latency exchanges.

B. Real-Time Mapping and Localization

A SLAM node executes on the Jetson Nano, fusing LiDAR scans with wheel-encoder odometry to generate and update a 2D occupancy map. Simultaneously, an Extended Kalman Filter (EKF) node subscribes to `/odom` and visual detections (ArUco markers) to publish corrected odometry, mitigating drift in pure dead-reckoning.

C. Path Planning and Obstacle Avoidance

A global planner node employs the A* algorithm to compute minimum-cost routes on the occupancy grid. When an unexpected obstacle appears (via `/scan`), a Bug-style avoidance node guides the robot along the obstacle boundary until the original path is recoverable—avoiding full replanning.

D. Voice Control

Voice commands (Start, Stop, Pause, Next) are recognized by a Hidden Markov Model (HMM) running off-board ROS 2. Interpreted commands publish to a `/voice_command` topic, where the main behavior node consumes them to control the forklift actuator.

E. Forklift Actuation

A dedicated FPGA module generates microsecond-precise PWM signals to drive the forklift motor.

F. gRPC Interface

A gRPC wrapper node exposes selected ROS 2 services and actions to external clients, permitting a web-based interface to display the robot camera stream and payload state.

II. ACTIVITY DEVELOPMENT

I. LOCALIZATION USING KALMAN FILTER WITH DEAD-RECKONING AND ARUCO MARKER DETECTION

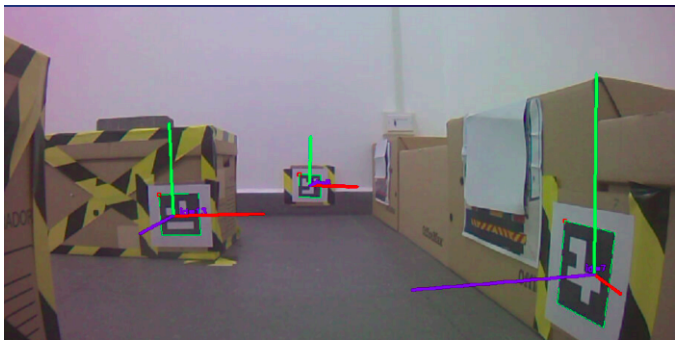

Figure 1. Aruco's detections

To correct the inherent drift of wheel odometry, which accumulates errors due to sensor noise and slippage , we fuse ArUco marker measurements with odometry using an Extended Kalman Filter (EKF). The system estimates the robot's pose (x, y, θ) in real-time.

The mathematical formulation of the implemented EKF is presented below.

Mathematical Model

1) State Vector: The robot's state $\mathbf{x}$ is defined as its pose in a global reference frame:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Where (x, y) is the position and θ is the orientation.

2) Process Model (Prediction): The differential drive motion model is used to predict the robot's next state from the wheel velocities:

$$\hat{\mathbf{x}}k = f(\mathbf{x}k-1, \mathbf{u}_k) = \begin{bmatrix} x_{k-1} + v\cos(\theta_{k-1})\Delta t \\ y_{k-1} + v\sin(\theta_{k-1})\Delta t \\ \theta_{k-1} + \omega\Delta t \end{bmatrix}$$

The state covariance is predicted using the Jacobian of the motion model, $\mathbf{F}$:

$$\hat{\mathbf{P}}k = \mathbf{F}_k\mathbf{P}k-1\mathbf{F}_k^T + \mathbf{Q}_k$$

$$\mathbf{F}_k = \begin{bmatrix} 1 & 0 & -v\sin(\theta_{k-1})\Delta t \\ 0 & 1 & v\cos(\theta_{k-1})\Delta t \\ 0 & 0 & 1 \end{bmatrix}$$

3) Observation Model (Correction): When an ArUco marker with a known position (m_x, m_y) is detected, the observation model $h(\hat{\mathbf{x}}_k)$ predicts the expected measurement (range and bearing):

$$\mathbf{z}_{\text{expected}} = h(\hat{\mathbf{x}}_k) = \begin{bmatrix} \rho \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{(m_x - \hat{x}_k)^2 + (m_y - \hat{y}_k)^2} \\ \text{atan2}(m_y - \hat{y}_k, m_x - \hat{x}_k) - \hat{\theta}_k \end{bmatrix}$$

4) Correction Step: The difference between the actual measurement $\mathbf{z}_k$ and the expected one is called the innovation $\mathbf{y}_k$:

$$\mathbf{y}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_k)$$

The Kalman gain $\mathbf{K}_k$ is calculated to weigh the innovation, using the Jacobian of the observation model, $\mathbf{H}_k$:

$$\mathbf{K}_k = \hat{\mathbf{P}}_k\mathbf{H}_k^T(\mathbf{H}_k\hat{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

$$\mathbf{H}_k = \begin{bmatrix} -\frac{m_x - \hat{x}_k}{\rho} & -\frac{m_y - \hat{y}_k}{\rho} & 0 \\ \frac{m_y - \hat{y}_k}{\rho^2} & -\frac{m_x - \hat{x}_k}{\rho^2} & -1 \end{bmatrix}$$

Finally, the state and its covariance are updated to obtain a corrected estimate:

$$\mathbf{x}_k = \hat{\mathbf{x}}_k + \mathbf{K}_k\mathbf{y}_k$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\hat{\mathbf{P}}_k$$

This prediction and correction cycle repeats, continuously refining the robot's pose.

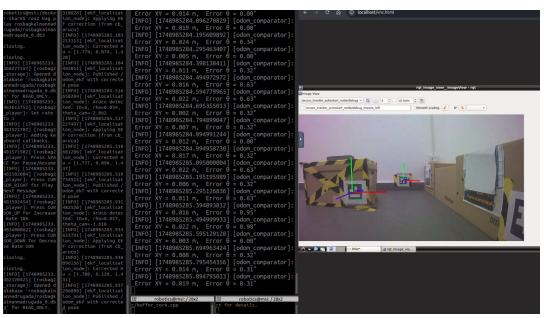

Figure 2. Pose correction cycle with EKF and ArUco detection.

Integration with ROS 2

In practice, this EKF is encapsulated within a dedicated ROS 2 node, acting as a central hub for localization. It seamlessly integrates with the broader robotic system by subscribing to raw sensor topics and publishing a filtered, more accurate pose estimate.

- **Prediction Input:** The prediction step is driven by the robot's internal odometry. The node subscribes to the `/VelocityEncR` and `/VelocityEncL` topics to receive wheel velocity data. This information constitutes the control input $\mathbf{u}_k$ and is used in a timer-based loop (at 20 Hz) to continuously predict the robot's forward motion.
- **Correction Input:** The correction step is event-driven. The node subscribes to the `/aruco_detections` topic. When a message containing the position of a known ArUco marker is received, it is used as the measurement vector $\mathbf{z}_k$ to correct the predicted state and reduce uncertainty.
- **Filtered Output:** The final output of the node is twofold. It publishes the fused pose (position and orientation) along with its covariance to the `/odom` topic as a `nav_msgs/msg/Odometry` message. Simultaneously, it broadcasts the transformation between the `odom` frame and the robot's `base_footprint` frame using the TF2 library. This transform is critical for the entire ROS 2 navigation stack, allowing other nodes to have a consistent and reliable understanding of the robot's position.

This prediction and correction cycle repeats continuously, refining the robot's pose in real-time.

II. Route Planning Using A*

In order to achieve the implementation of autonomous navigation in our puzzlebot, we first need to be able to determine the optimal path from our current position to our goal. As our puzzlebot was able to obtain a map of the environment through the implementation of SLAM, we are now able to implement path planning algorithms such as A*. This section presents the implementation of a global path planning system for mobile robots using the A (A-star) algorithm* in ROS2. Our system is divided in two main modules: the path planner logic and the ROS2 node integration. The chosen approach combines classic informed search algorithms with robust map handling and user-friendly ROS2 interfacing, enabling flexible and efficient navigation in grid-based environments.

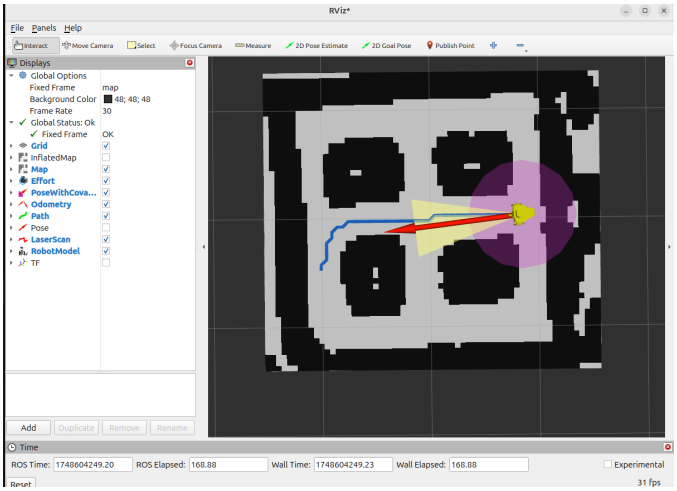

Figure 3. A star simulation

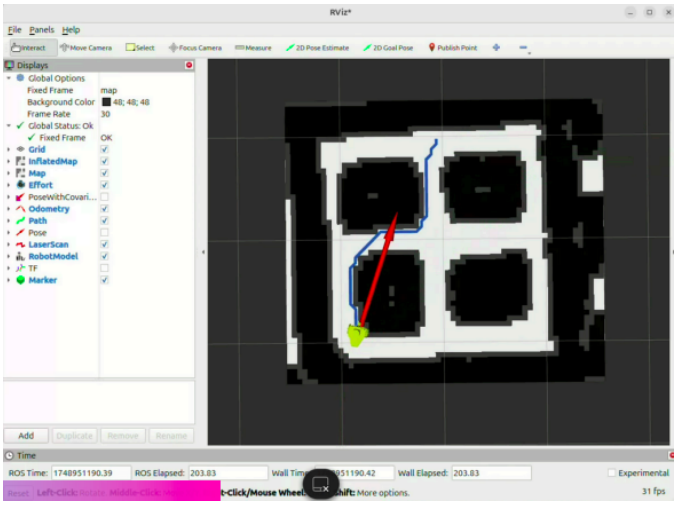

Figure 4. A star simulation

The A*algorithm module defines the core of the theoretical and logical calculations made for the pathfinding. The map obtained from the environment through SLAM is represented as an occupancy grid, where each cell/node can be free, occupied or punished (inflated obstacles are also considered). The planner models each cell as a node with associated costs: g (cost from the start), h (heuristic to the goal), and f (total cost). It explores possible moves in 8 directions (cardinal and diagonal), always expanding the node with the lowest total cost.

The heuristic is computed using the Euclidean distance, ensuring the planner is both optimal and efficient. The algorithm employs a priority queue for fast node selection, and reconstructs the path once the goal is reached. Safety is enhanced by checking that both start and goal are within map bounds and not inside obstacles. The ROS2 node serves as the interface between the planner and the robot's operating environment. It subscribes to map, odometry, initial pose, and goal pose topics, converting map data to a 2D NumPy array for fluid processing. Before any path planning goes on, the node inflates obstacles using a BFS-based approach to take into consideration the PuzzleBots physical dimensions, ensuring paths are actually viable and safe for the robot to navigate through.

Upon receiving both the start and goal pose, the node calls for the path planner, transforms the resulting path into ROS2 path messages, and publishes them for visualization and as a stream for navigation. We decided for the planner to wait upon not only the final goal but also the initial pose for a more interactive system. The functionality of the system offers real time interactive path planning with robust map inflation, ideal for odometry based starting positions, and allowing a visualization of the results. The division on the nodes modular design allows for the programmers to encounter errors, correct them or maintain the system with ease.

III. Obstacle Avoidance Using Reactive Algorithms such as Bug-Type Methods

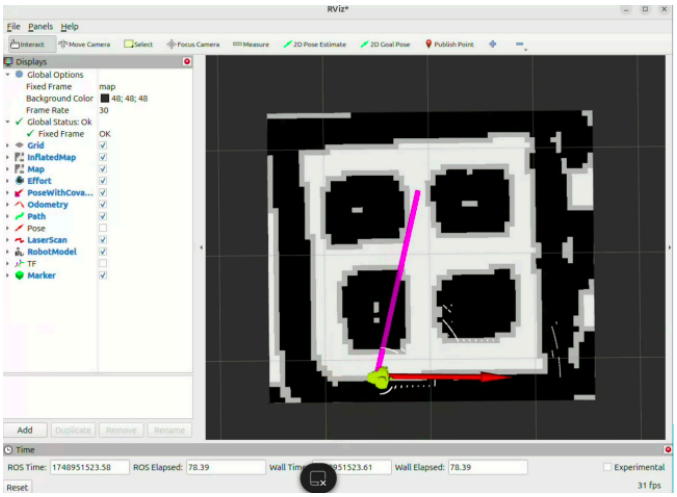

Figure 5. Bug Algorithm in simulation

The project aimed for the Puzzlebot to rely on the principles of an autonomous navigation system implemented within the ROS2 environment. The base of the approach blends reactive navigation based on real time localization carried through local perception with minimal global planning. In order to allow the robot to reach multiple sequential waypoints in an environment with unknown obstacles, we reviewed reactive algorithms type "BUG". The main method implemented in our project is a hybrid navigation strategy. Our bug system consists of two main states: switching between moving directly towards the goal ("go to goal") and the state following obstacle boundaries ("follow obstacle"), this state is based on the laser scans sensor input processed in real time generated by the Lidar. After developing full understatement on the different types of "BUG" algorithms ("bug0, bug1, bug2") we came to realize that an adapted version of BUG2 and BUG Tangent would

be the best option for our implementation. The Tangent Bug algorithm is an advanced variant of classical Bug algorithms with the addition of taking into account data readings of the sensor "Lidar". Addressing the implementation of Bug Tangent algorithms, in our code, the robot continuously evaluates the straight-line distance to its goal ("m-line"), using lidar data to detect obstacles. If the path is clear, it drives directly to the goal. If an obstacle is detected in the direct path, the robot analyzes gaps (free spaces) and applies distance-based heuristics to select the best tangential direction, prioritizing wide openings or circumnavigating the obstacle until it finds a clear sector that brings it closer to the goal than where it started following the boundary. Thoroughly analyzing the implementation of the algorithm, our code's logic implements a state machine with two main stages ("go to goal and follow obstacle") the interaction and switch dynamic being dictated by the sensor readings (Lidar). To ensure a correct obstacle following implementation, we developed a proportional derivative (PD) control, used to maintain a consistent distance from the walls/obstacles. The velocity commands ("linear and angular") are smoothed over time passing using a sliding window, with the objective to ensure stable and smooth motion.

The functionality of the code developed as follows: the code manages a sequence of waypoints (navigation goals) declared inside a "tupla" and uses ROS2 callbacks to process odometry and laser scan data. Different visual aids are displayed in RViz such as the "m-line" providing insights into the puzzle-bots navigation behavior helping with the debugging process. The system has a robust approach with many safety features developed after encountering many errors and a thorough debugging process, being able to handle invalid sensor readings, can detect and escape certain corner traps and securely stopping the robot after all goals are achieved.

IV. Map Creation Using SLAM Toolbox

Our 2D map is built using `slam_toolbox` in online mode. It subscribes to the following topics:

- `/odom` – Odometry information
- `/scan` – LIDAR scan data

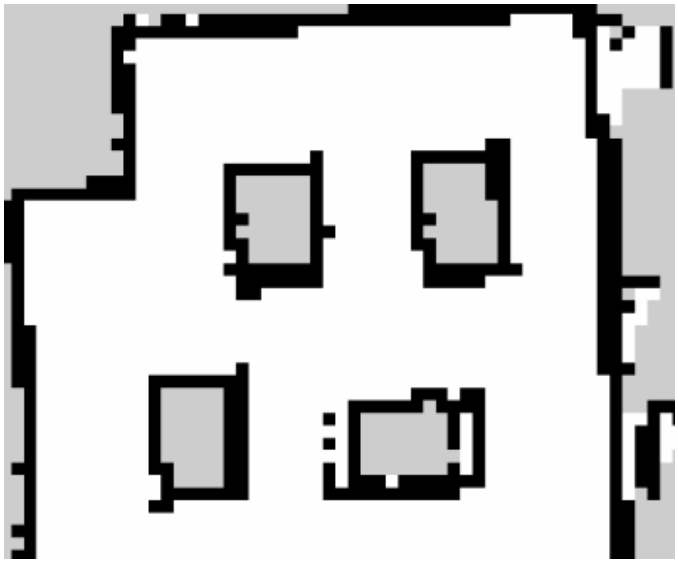

Figure 6. Map generated using SLAM Toolbox during online mapping session.

V. Voice Command Control with HMMs

We implemented a voice command module using Hidden Markov Models. After preprocessing and extracting MFCC features, we decode commands using the Viterbi algorithm. Below is the pseudocode for Viterbi decoding:

Algorithm 1 Viterbi Decoding for HMM–based Commands

Input: observation sequence $O = o_1, \ldots, o_T$
Initialize $\delta[1][i] = \pi_i \, b_i(o_1), \quad \psi[1][i] = 0$ for each state i
for $t = 2$ **to** T **do**
  for each state j **do**
    $\delta[t][j] = \max_i \left[\delta[t-1][i] \cdot a_{ij}\right] \cdot b_j(o_t)$
    $\psi[t][j] = \arg\max_i \left[\delta[t-1][i] \; a_{ij}\right]$
  end for
end for
$P^* = \max_i \delta[T][i], \quad q_T^* = \arg\max_i \delta[T][i]$
for $t = T-1$ **to** 1 **step** -1 **do**
  $q_t^* = \psi[t+1][\, q_{t+1}^* \,]$
end for
Output: best state sequence $q_1^*, \ldots, q_T^*$ and its log–probability P^*

VI. Forklift Motor Control Using an FPGA

A finite state machine on an FPGA Tang Nano 20K controls an MG90S servo. States include Idle, Initialize, Drive Up, Hold, Drive Down, and Error. Transitions are triggered by sensor inputs and command signals.

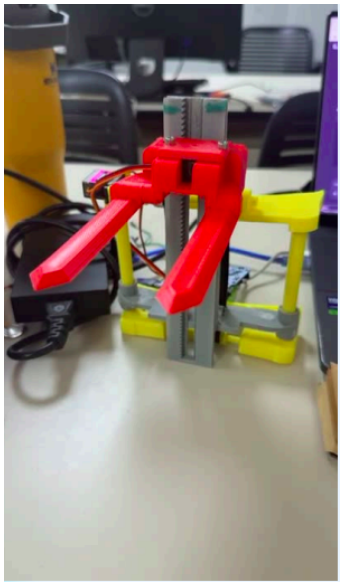

Figure 7. Forklift printed in 3d

VII. Web Visualization Interface

The interface implements a multi-language approach as seen in class with Ceron. The interface serves as the HMI for the puzzlebot by seamlessly integrating ROS2, GO, Python, and a HTML web interface. The methodology works around a service oriented architecture using gRPC (Google Remote Procedure Calls) as the bridge between the backend services, which were written in Go, ROS2 nodes in python, and the user web interface using FLASK and written in HTML.

The backend component is developed in Go (go-gateway.go), where it acts as a gateway that exposes robot functionalities as gRPC services. The Go backend manages protocol definitions (with .proto files) and uses generated code for both Go and Python to enforce strict message schemas and ensure reliable cross-language RPC calls. This layer handles incoming gRPC requests, processes them, and relays commands to the appropriate system component, ensuring efficient handling of client requests and message passing.

On the Puzzle Bot dynamic the Python script (ros2-grpc-wrapper.py) operates as a ROS2 node and a gRPC server/client. It connects the Go backend with the robot's ROS2 ecosystem, translating gRPC calls into ROS2 service or topic interactions (such as controlling motors, reading sensors, or getting robot status). The auto-generated Python gRPC classes (generated_proto_python/) provide type-safe communication based on the shared protocol definition. This ensures that commands sent from the web or Go backend are correctly executed on the robot and that results or feedback can be sent back to the client.

The web interface is built using Flask (web/app.py) and standard HTML templates (web/templates/interfaz.html). This frontend provides a user-friendly means for users to send commands, visualize robot data. The web application communicates with the Go backend via gRPC, abstracting technical

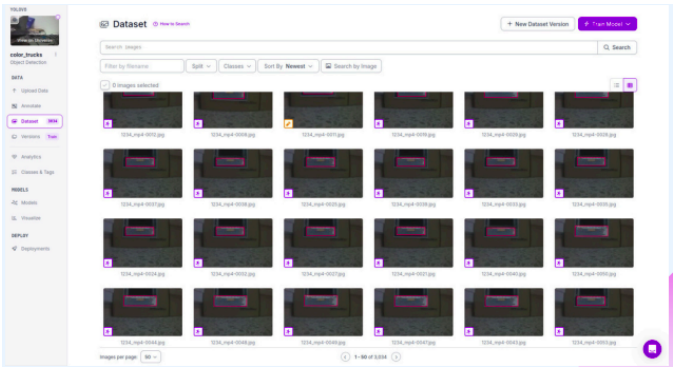

Figure 8. AI Training

complexities and offering an intuitive control panel for end-users.

VIII. QR Code Processing and Integration

The QR code module detects and decodes codes with OpenCV's `QRCodeDetector`. Unique codes trigger route planning.

Algorithm 2 QR Code Detection & Navigation Integration

```
processed_set ← ∅
while camera is active do
    frame ← capture frame
    codes ← detect QR codes in frame
    for each code in codes do
        if code ∉ processed_set then
            add code to processed_set
            if code is URL then
                open browser with code
            else
                parse logistics info from code
                delivery_route ← plan route based on parsed data
                send delivery_route to navigation module
            end if
        end if
    end for
    overlay visual feedback on frame
end while
```

IX. Delivery Trailer Detection Using AI

As part of the integrated system of the Puzzlebot we needed to develop an approach to carry out recognition and distinguish the targeted trucks, based on specific characteristics given to us: 1) if the truck was rusty, 2) if the truck had circular lights and last but not least 3) if the rear of the truck had a diagonal structure.

This recognition process is fundamental, allowing the puzzlebot to correctly execute the package delivery. Before the robot can approach, align, and deliver the package into the correct trailer, it must first identify and locate the trailer in

the robot's camera field of view. For this purpose, a robust object detection module based on the YOLOv8 (You Only Look Once) model was developed and seamlessly integrated with ROS2 and OpenCV.

The core of this module is a ROS2 node that loads a custom-trained YOLOv8 object detection model. This model was trained and evaluated on a large dataset of truck images, as evidenced by the Roboflow dashboard (with metrics: 96.5, Precision of 95.9, and Recall of 91.6), ensuring high reliability for real-world robot perception. The code initializes a video stream from the robot's onboard camera, processes each frame in real time, and applies the YOLOv8 model to detect target objects—specifically, the colored trucks used in the project scenario.

For every camera frame, the module runs inference through the YOLOv8 model, generating bounding boxes and class predictions. The results are visualized live by drawing the detections on the image, allowing both the system and human operators to verify detections instantly. The detection results can be further integrated into downstream processes, such as navigation to the detected trailer or triggering alignment routines.

A crucial aspect of this solution is real-time performance and high detection accuracy, which ensures that the Puzzlebot responds quickly to environmental changes and can reliably detect its target trailer even in challenging or cluttered scenes. The use of a state-of-the-art deep learning model allows the system to generalize well across different lighting, backgrounds, and truck appearances.

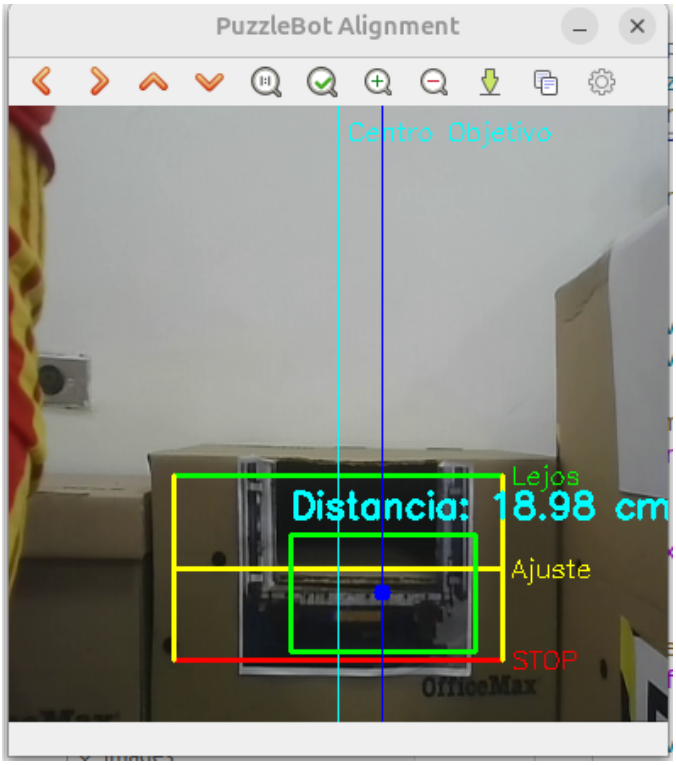

Figure 9. Trailer aligment

X. Trailer Alignment via Computer Vision

Once the Puzzlebot successfully navigated to the goal position carrying the package (cube), the next challenge was to accurately drop the package inside the trailer (truck). While applying AI for recognizing the trailers was an essential first step, precise alignment was necessary to ensure the package was delivered correctly. For this purpose, we developed a dedicated computer vision based trailer alignment system. This system addresses the need for both robust trailer detection and fine-grained alignment, bridging the gap between global navigation and the final delivery action.

This module of the project provides an automatic vision based alignment system for the puzzlebot to accurately carry out aligning and approach maneuvers using real time camera inputs and computer vision techniques. The code leverages ROS2 for the practical and smooth robot communication and OpenCV for image processing, implementing both feature-matching (seen with Ceron on the Fotogrametria project) and visual tracking in order to accurately detect and follow the trailer based on certain characteristics.

The alignment system is dynamic, it supports static images, webcam streams or images from a ROS camera topic. This on the Camera/Image analisis versatility allowed us to carry out a test phase without experiencing complex difficulties upon having only one puzzlebot but many other modules to test out. The system first extracts ORB (Oriented Fast and Rotated BRIEF) features so that we can robustly describe key points. In the meantime camera frames are undistorted using precalibrated camera parameters and the systema attempts trailer detection via template matching and homography estimation. Once detected it employs tracking for real time frame to frame tracking.

When the trailer's alignment gap (the "hole") is located using homography and bounding box extraction, the code estimates the distance to the trailer with a pinhole camera model, leveraging the known real-world width of the gap. The system then computes lateral and longitudinal errors relative to the ideal drop-off point, using proportional control to generate velocity commands. These commands are sent as ROS2 Twist messages, allowing the robot to continuously adjust its position and heading for perfect alignment. Visual guides are drawn on the live video feed, offering clear feedback to the operator.

A key feature of this module is its real-time feedback loop: the robot automatically adjusts its approach, correcting both its distance and angular alignment with respect to the trailer. The system stops the robot once the gap is both centered in the image and at the correct distance, maximizing drop-off accuracy. If tracking fails, the system automatically re-detects the trailer, enhancing robustness. Detection results are validated through geometric constraints to ensure only valid alignment opportunities trigger the drop-off sequence.

Individual Reflections — José Pablo Cedano Serna

This was definitely the most complex problem situation we've faced throughout the entire degree. We had highs, lows, and—above all—a lot of learning. I learned the value of planning and executing projects using the **SCRUM methodology**, which allows us to iterate on tasks, track progress, and manage team dynamics. As is often the case, the hardest part of any project is working effectively as a team. That challenge was very present for us, and overcoming it was a valuable part of the learning process.

Beyond the soft skills, I was truly fascinated by how, the deeper you dive into **ROS 2**, the more you understand why its relevance keeps growing. The fact that it's *open source* allows us to build upon existing foundations, improve them, and adapt them to our needs. I would love to explore this more deeply and work with ROS 2 professionally after I graduate.

SLAM is an incredibly useful tool in the context of mobile robotics. In our project, we implemented it on a scaled-down robot, but in real-world applications, it powers autonomous cars—which is amazing to think about. It's exciting to know that I now understand the foundations behind such widely talked-about technology. It goes hand-in-hand with route planning and dynamic obstacle avoidance—being able to compute alternate paths in real-time when faced with changes in the environment.

But none of these algorithms would matter if we couldn't make them connect intuitively with the user. That's where **Human-Machine Interfaces (HMIs)** come in. Because what's the point of advanced technology if it isn't user-friendly?

Robotics is a fascinating field of engineering. It's where algorithms meet the physical world through actuators. In the future, robotics will become more and more integrated into everyday life. That's why, personally, I wish it had greater relevance in **Mexico**, because it's not very prominent yet. But that opens up a huge opportunity—to be among the first to bring it here and help build that market from the ground up.

My Contributions to the Project

If I had to highlight my contributions to this project, I would say I spent a significant amount of time serving as the **Scrum Master**, in addition to working on the technical side.

- I developed the **base simulation for the Puzzlebot** used in all the mini-challenges, including the transform tree, odometry, and motion controller—everything related to the simulation was built from scratch.
- I implemented **ArUco marker detection** and helped with **camera calibration**.
- I laid the foundation for the **Extended Kalman Filter**.
- I made the **initial implementation of SLAM**. While it still needed debugging, it was a functional starting point.
- I conducted extensive research to make our **Jetson Nano** run as efficiently as possible.
- I configured a **modem as a Wi-Fi hotspot** for the Puzzlebot, which significantly improved connectivity.

Individual Reflections — Luis Antonio Zermeño De Gorordo

This project significantly challenged my abilities and knowledge, demanding a comprehensive understanding of nearly all semester coursework, alongside new concepts introduced just weeks before the final prototype submission. Additionally, the interpersonal dynamics, or teamwork aspect, presented another crucial challenge. Therefore, the primary challenges were academic and related to teamwork, and this reflection will detail my key learnings in both areas and highlight my individual contributions within the team.

Throughout the semester, various assignments served as precursors to the final project tasks, which our team divided accordingly. A pivotal learning experience was gaining a deeper understanding of the robot's odometry and kinematic model through debugging algorithms such as SLAM and Kalman. Although the kinematics concepts were introduced in previous semesters, their practical implementation significantly reinforced the mathematical principles. Furthermore, the classes that we had about algorithms utilizing codebooks, features, and triangulation patterns were the key for comprehending both previous Monte Carlo approaches and the current functioning of SLAM.

Unexpectedly, I revisited network concepts due to persistent SSH connection issues between the puzzlebot and the computer, necessitating the configuration of multiple network setups to establish a reliable connection. The Kalman filter also presented significant challenges; the complexities of its mathematics and transformations were compounded by our decision to divide the code. Low-latency dependencies were managed on the Jetson, while less data-intensive components ran on the computer. Consequently, elements like the camera with ArUco markers, Kalman filter, odometry, and LiDAR were deployed on the puzzlebot, with all other elements residing on the computer, requiring two separate launch files.

This division created issues with coordinate transformations. Beyond specific problem-solving, I also refined existing knowledge of ROS2 and acquired new insights that would have significantly streamlined my research, particularly regarding URDF understanding, transformations, path-planning and Gazebo simulations. I learned to calibrate cameras using various algorithms, expanding my practical skillset. Finally, I experienced the difficulty of integrating all the codes into a cohesive system. Merging the nodes proved to be a frustrating challenge. This process demanded a deep understanding of each new node's core functionality and required significant tweaking, as individual components, while perfect in isolation, often needed restructuring to meet the standards of the overall project. I had anticipated this would be the easy part, but it turned out to be one of the most difficult aspects of the project, mixed with multiple Jetson related issues. However, certain challenges emerged that, in retrospect, could have been more effectively addressed through more resolute leadership and by promptly soliciting expert assistance from faculty members when hardware systems experienced critical malfunctions.

INDIVIDUAL REFLECTIONS — RODRIGO ESCANDON LOPEZ GUERRERO

Working on this project was highly challenging. The Puzzlebot system was an intense yet highly educational experience. Each component, from global path planning to vision-based alignment and cross-language communication, presented unique challenges that pushed me to deepen my technical understanding and adapt my problem-solving strategies.

During this project we also tried to work and developer our project management level abilities, applying the Scrum methodology was a valuable exercise in modular development and task assignment. While the approach clarified individual responsibilities and facilitated parallel work, it also exposed limitations—particularly when some team members did not match the overall group's level of effort or struggled with regular attendance at meetings. These coordination issues sometimes led to uneven progress and inconsistent deliverable quality, reminding me that technical excellence must be paired with effective teamwork and communication.

The development of the project not only allowed me to grow and develop skills that I have been fascinated by during my years in college but also allowed me to work on those that I often find myself struggling with. I not only got the chance to work on one of the fields that I often most enjoy "Computer Vision" but I also discovered a certain passion for Path Finding and search algorithms. We encountered many problems during the project but we always found a way to evolve and find solutions to them.

During the implementation of the A path planner*, one of the main hurdles was the correct inflation of the occupancy grid. I realized that inflating obstacles not only accounted for the robot's dimensions but also inadvertently amplified errors introduced by the lidar, making the navigation environment more restrictive than intended. Fine-tuning the padding to match the Puzzlebot's actual physical characteristics required extensive testing and led to several iterations to minimize false positives for obstacles while still ensuring safety. Additionally, the tuning of the PD controller was crucial, as early versions led to disproportionate angular versus linear velocity, sometimes causing the robot to get stuck in oscillatory behaviors. Ultimately, introducing a penalization mechanism for nodes near obstacles created a "grey zone" that improved path safety while preserving necessary flexibility for the planner.

The Bug (Tangent Bug) algorithm presented a different set of issues, mainly related to real-world navigation dynamics. I observed that when the Puzzlebot encountered an obstacle directly in front and failed to react in time, it could end up with the object caught between its wheels. This often led to indecisive behavior, such as repeatedly switching between turning right and left without making progress. Certain map corners could also cause the robot to get stuck, requiring the implementation of specific escape behaviors and smarter state transitions.

Integrating gRPC communication between ROS2 and the web interface revealed the hidden complexity of real-world networking. This was one of the more challenging tasks of the project as the debugging took a long time in which we finally realized that the problem wasn't necessarily something in our code but on the capcities of our components. Despite correctly handling image compression and message serialization, the system was sometimes unable to transmit images reliably, which we ultimately traced to limited WiFi bandwidth and instability. This highlighted the importance of robust network infrastructure and error handling in distributed robotic systems.

The vision-based alignment system using computer vision was both fascinating and challenging, its one of my favorite fileds within the aspects of the project. The greatest difficulty came from the need to account for environmental variability—changes in lighting, reflections, color shifts, and even geometric distortions all impacted the system's robustness. Building a solution that could adapt to these variations required careful parameter tuning and iterative design, underscoring the inherent unpredictability of real-world deployment versus controlled laboratory conditions.

REFERENCES

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *IEEE Proceedings*, vol. 77, no. 2, pp. 257–286, 1989.

[2] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260–267, 1967.

[3] D. Yu and L. Deng, "Deep learning and its applications to signal and information processing," *IEEE Signal Process. Mag.*, vol. 28, no. 1, pp. 145–154, 2011.

[4] S. Young et al., "The HTK book (for HTK version 3.4)," Cambridge University Engineering Department, 2006.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] J. Durbin and S. J. Koopman, *Time Series Analysis by State Space Methods*, 2nd ed. Oxford University Press, 2012.

[7] L. R. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[8] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., Pearson, 2023.