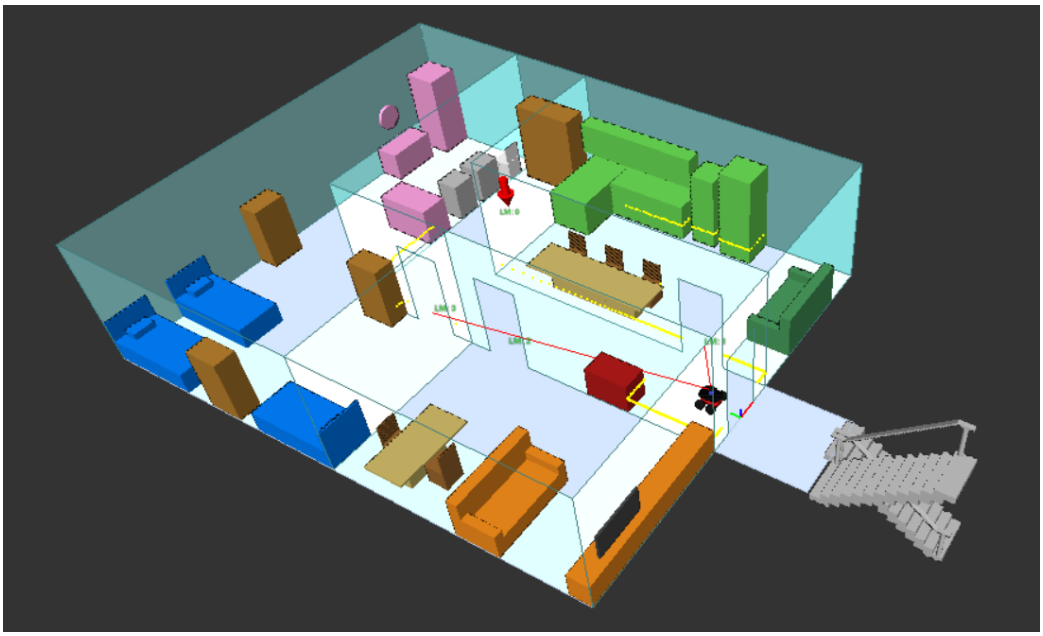
UNIVERSIDAD POLITÉCNICA DE MADRID

Patrol Robot for Senior Care Facilities

Guiado y Navegación - MUAR



Tom Lemmel - 24222
Edoardo Selvaggi - 24862

Máster en Robótica y Automática
Universidad Politécnica de Madrid

January 9, 2025

This document present our implementation of a patrol robot in a senior care facility using the Apolo simulator (see [2]) and Python

Contents

1 Introduction and Use Case

Elder care facilities play a vital role in society by providing essential support for aging populations. They help elderly individuals maintain a good quality of life by offering medical care, social interaction, and assistance with daily activities.

However, changes in the age pyramid (see Figure 1) are making it harder to meet the growing demand for elder care. With fewer young people and a rising number of elderly individuals, the balance is shifting. This puts increasing pressure on healthcare systems and long-term care facilities.

At the same time, there is a serious shortage of workers in elder care. Many countries struggle to find enough trained caregivers, nurses, and healthcare staff. The work is physically and emotionally demanding, wages are often low, and the number of retirees keeps increasing. As a result, elder care facilities face staffing shortages, which can affect the quality of care. That is why we chose this subject.

In our scenario, a elder care facility is designed across multiple levels. In this building the first floor is dedicated to the medical staff while the other floors different apartments in which three or four eldery people live together. In order to offer a good coverage of monitoring across the day and night, while having a restrict number of medical staff in every shift, each apartment is equipped with a patrol robot. The robot, called Marvin, periodically performs reconnaissance rounds in the apartment, moving through each room. If, during its round, it detects a patient on the floor, it goes to their aid. The robot is equipped with an emergency button and a microphone directly connected to a loudspeaker located on the first floor, where the medical staff is stationed. This allows the patient to call for help and enables the medical staff to provide immediate instructions to the victim by responding through a second microphone and a loudspeaker mounted directly on the robot. The detection of a person laying in the ground is, in this case, simulated by a landmark detectable by the robot by means of its range/bearing sensor. When the comunication between the patient and the medical staff has been completed the robot returns to the facility entrance, ready for a new patrol round.

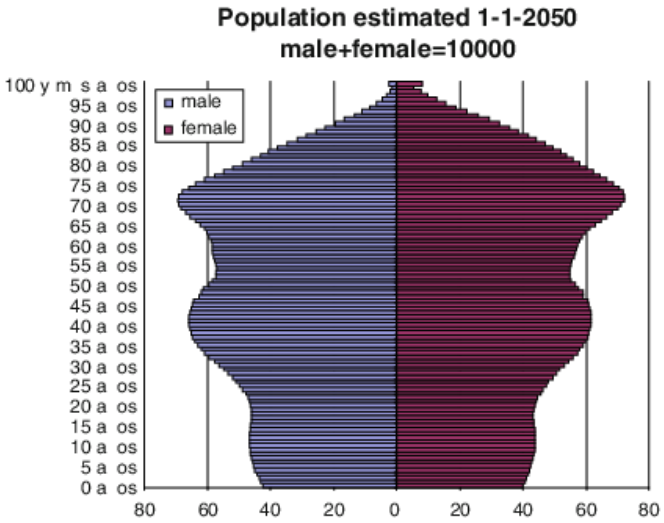


Figure 1: Spanish Age Pyramid Prediction for 2050 [3]

2 Robot “Marvin”

The robot employed in this work is a differential ”Pioneer3AT” wheeled robot available in the Apolo simulation platform, named “Marvin.” Designed as a service robot, Marvin is well-suited for the patrol tasks required in an elder care facility.

Based on a review of the literature [4, 5], Marvin’s sensing system was designed to meet the needs of a patrol robot in an elder care facility. The LMS100 LIDAR sensor [1] was chosen as the main device because it provides precise distance measurements and a clear view of the surroundings. This is especially important for safely navigating and avoiding obstacles in the complex and dynamic environment of elder care spaces.

In real-life applications, facilities often don’t have pre-made maps available, meaning the environment needs to be mapped before deploying Marvin. While this work doesn’t focus on mapping, we chose to use the LIDAR instead of simpler ultrasonic sensors to ensure Marvin is ready for real-world scenarios as LIDAR also supports navigation and could handle more advanced tasks like mapping if needed later.

3 Simulation Environement

To properly simulate the robot action in the elder care facility one entire apartment has been reconstracted in the software Apolo. The flat measures in total 120 squared meter (10 m by 12 m) and features in total three main rooms linked together by a corridor (created using a set of faces). The first room, entering to the right, is the kitchen. Continuing to the left, there is the entrance to the second room, the living room. Finally, a little further ahead are the bedroom and the bathroom. The corridor continues with a right turn, forming an L-shape. All the main furniture and pieces typically found in an apartment have been recreated (with simplified but to-scale shapes) and appropriately placed within the rooms. Some of them were designed using SolidWorks, a 3D modeling software, and then exported in STL format. Others were created directly by using simple objects available in Apollo, such as prisms. For proper interaction between the robot and these objects using the selected sensors, all STL objects imported into Apollo were "filled" with appropriately sized prisms matching the exact dimensions of the original STL objects.

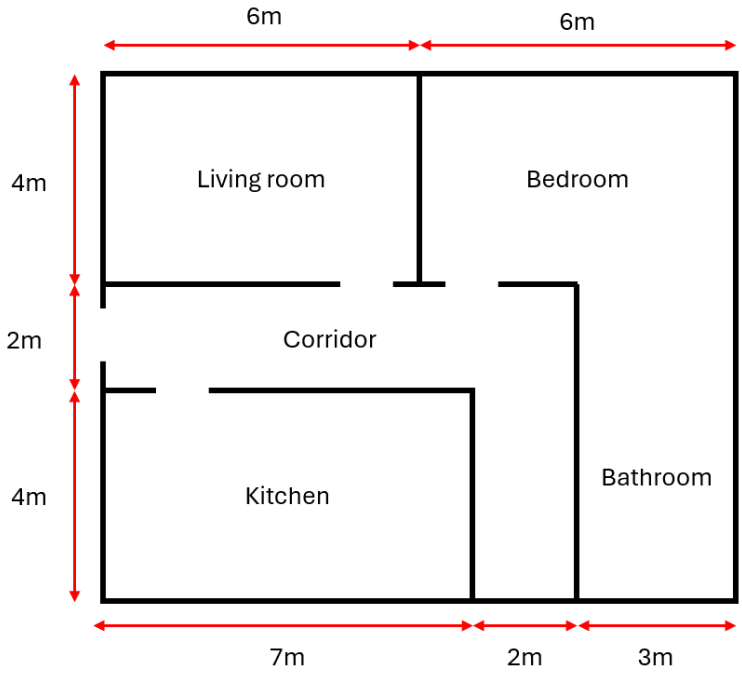


Figure 2: Map of the apartment with main dimensions.

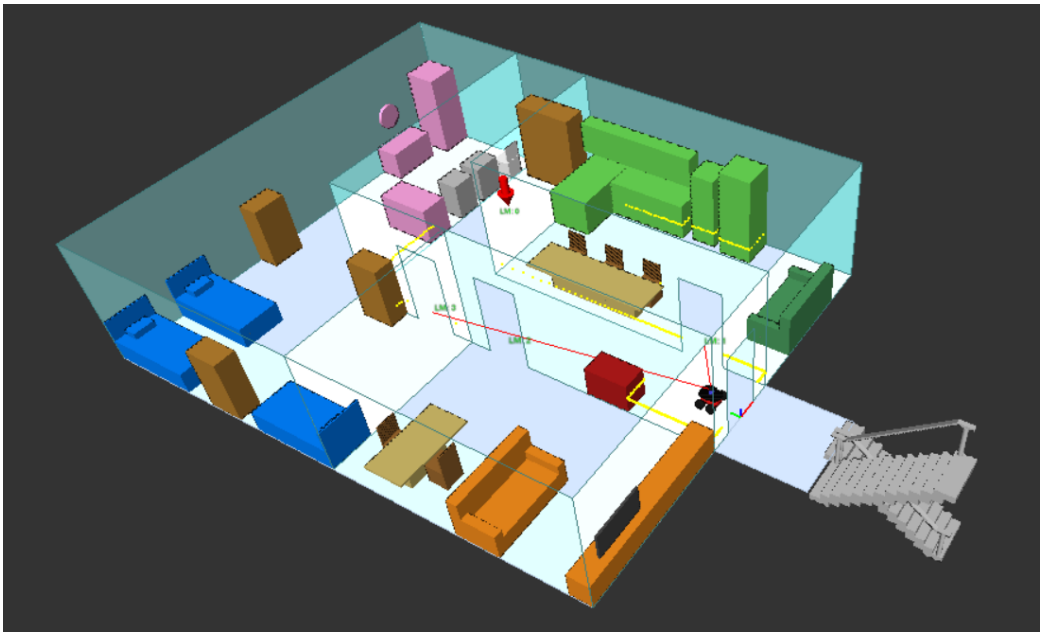


Figure 3: Perspective view of the apartment.

4 Controller Architecture and Main Functions

The controller is designed to follow a predefined route inside the apartment while constantly searching for a victim that may require rescue. If a victim is detected, the robot immediately interrupts its current route and proceeds directly to the victim's position.

The logic of the controller is structured into two phases:

In the first phase, when the robot has not yet detected the victim, it proceeds through the waypoints in sequence. To reach each waypoint, it first aligns itself toward the target using a PI controller for rotation, then moves forward while constantly checking for potential collisions. The movement forward also follows a PI control law. If a collision is predicted, the robot computes an intermediate point to avoid the obstacle before continuing to the current goal. This behavior is depicted in **Behavior Tree 1** (see Figure 4).

In the second phase, once the victim is detected, the robot treats the victim's location as the new goal and moves directly toward it. Again, both the rotation and forward movement are controlled using PI laws. Upon reaching the victim, the robot returns home to complete its task. This behavior is represented in **Behavior Tree 2** (see Figure 5).

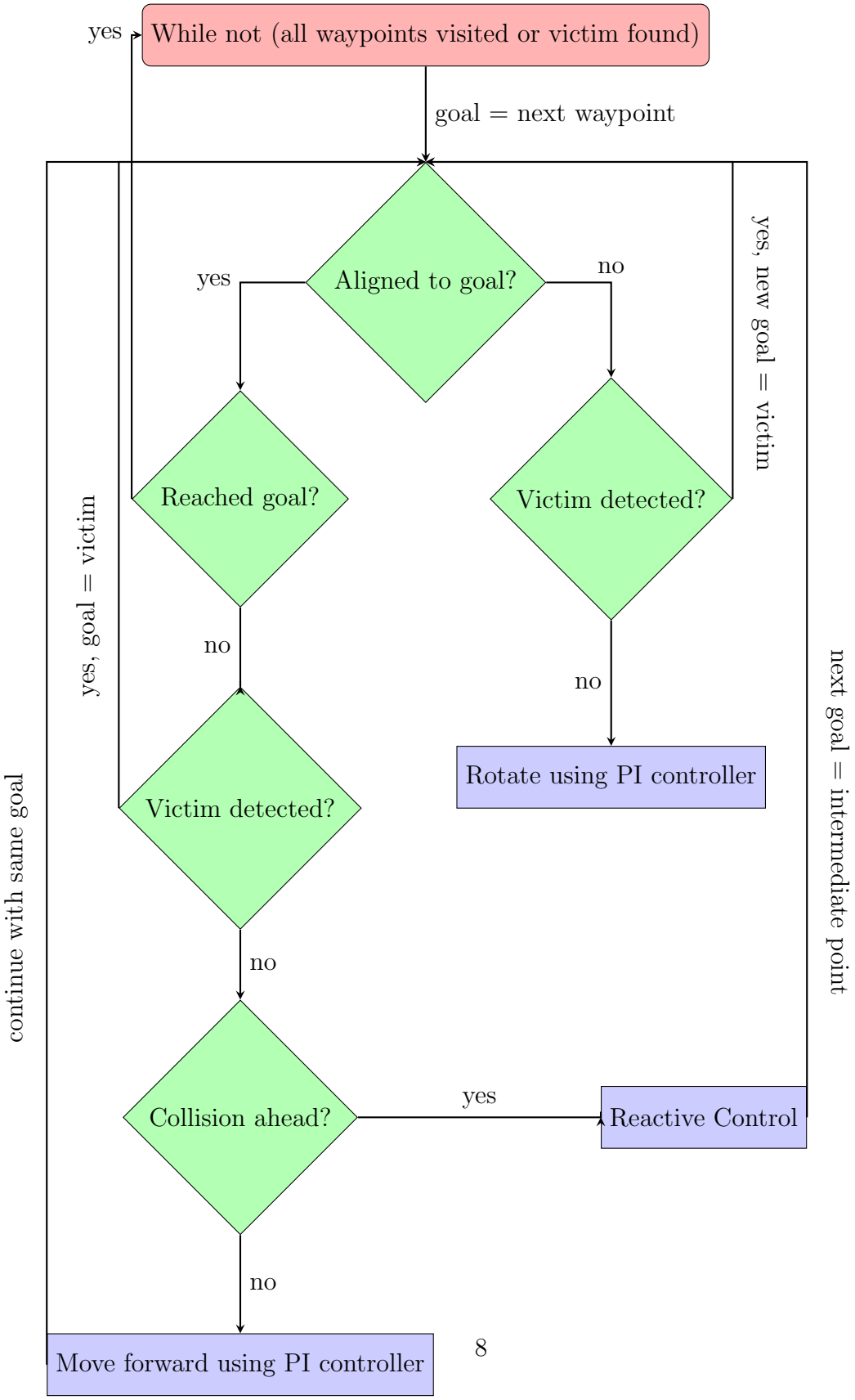


Figure 4: Behavior Tree 1: Navigation while looking for the victim

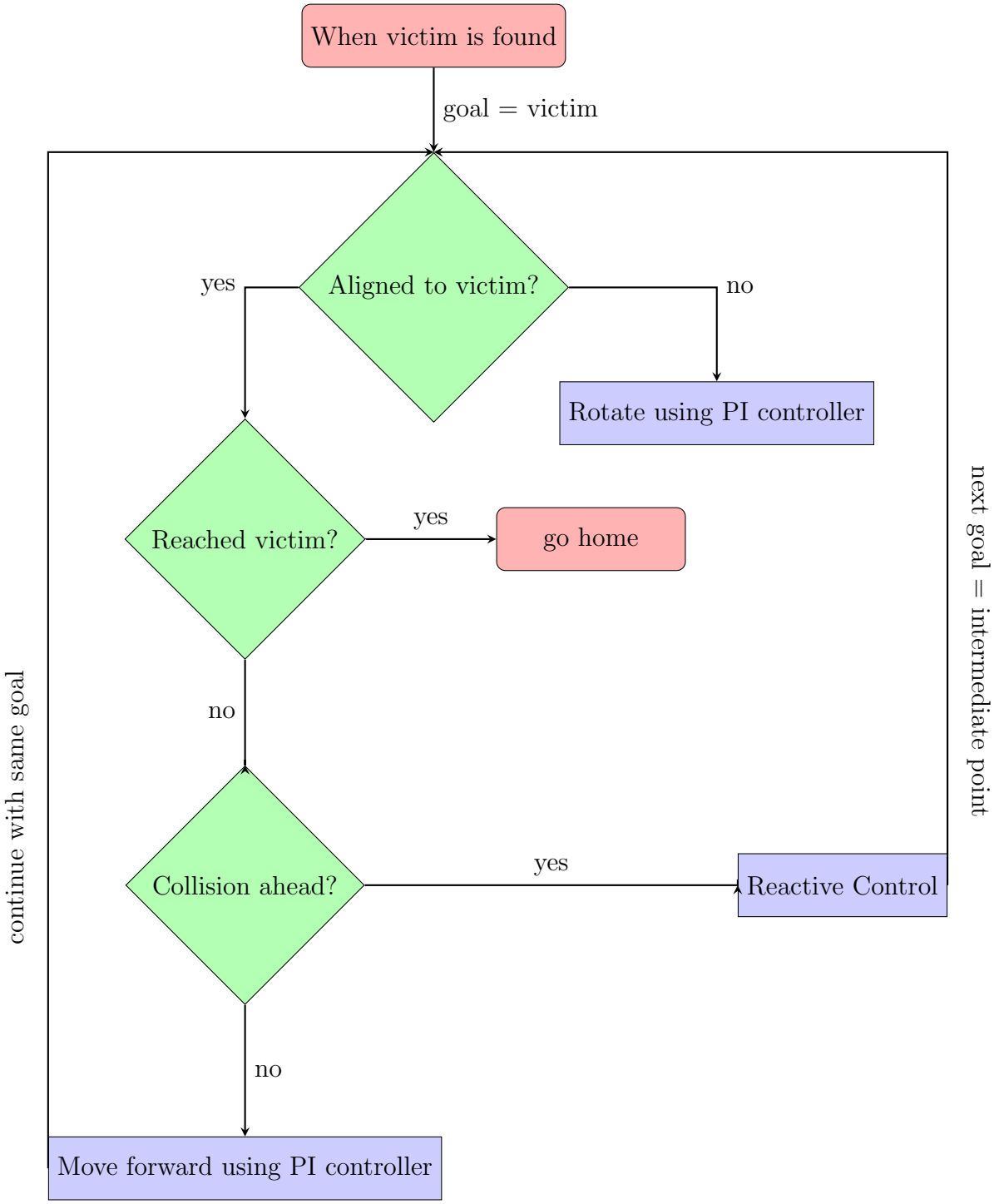


Figure 5: Behavior Tree 2: Rally victim position

4.1 Reactive control

The reactive control system is designed to anticipate and avoid potential collisions as the robot moves toward its designated goal, which could be a victim, a waypoint, or an intermediate navigation point.

4.1.1 Collision Prediction

To predict collisions, the system generates a *prediction polygon* extending from the robot's current position toward the goal, capped at 3 meters. The width of this polygon is equal to the robot's size plus a safety margin. The robot then analyzes LiDAR sensor data to check whether any detected obstacles fall within this polygon.

If an obstacle is detected inside the polygon, it indicates a potential collision if the trajectory remains unchanged. In response, the robot must adjust its path.

4.1.2 Path Adjustment

Instead of following a single straight line from its current position to the goal, the robot introduces an *intermediate waypoint*, creating a new trajectory composed of two consecutive straight segments:

1. From the robot's current position to the intermediate waypoint.
2. From the intermediate waypoint to the goal.

Choosing the Direction (Right or Left)

The first step in determining the intermediate waypoint is deciding whether to navigate around the obstacle to the right or left. To do this:

- The robot calculates the *centroid* of all obstacle points detected within the prediction polygon.
- It determines whether this centroid is to the right or left of the original trajectory (the straight line connecting the robot to the goal).
- The robot then chooses to move in the opposite direction to avoid the obstacle.

Placing the Intermediate Waypoint

Once the direction is chosen, the robot calculates the intermediate waypoint as follows:

1. It identifies the *farthest point* of the obstacle on the chosen side (right or left).
2. From this point, it determines the waypoint by moving a set offset distance away from the obstacle.
3. The offset direction follows the perpendicular to the original trajectory.

For example, if the robot needs to move to the right:

- It finds the farthest obstacle point on the right.
- It determines the intermediate waypoint by shifting rightward, away from the obstacle, by the predefined offset distance.
- The same logic applies if the robot must move left.

This approach allows the robot to dynamically adjust its trajectory, ensuring smooth and efficient navigation while avoiding collisions.

To achieve the final version of the reactive control, multiple tests were conducted. In particular, the control was tested with obstacles of varying sizes and with different relative positions of the obstacles in relation to the robot and the original trajectory. The values for the safety margin (0.3 m), used to create the prediction polygon, and the offset (0.5 m), used to calculate the intermediate point, are the result of a tuning process following the tests conducted.

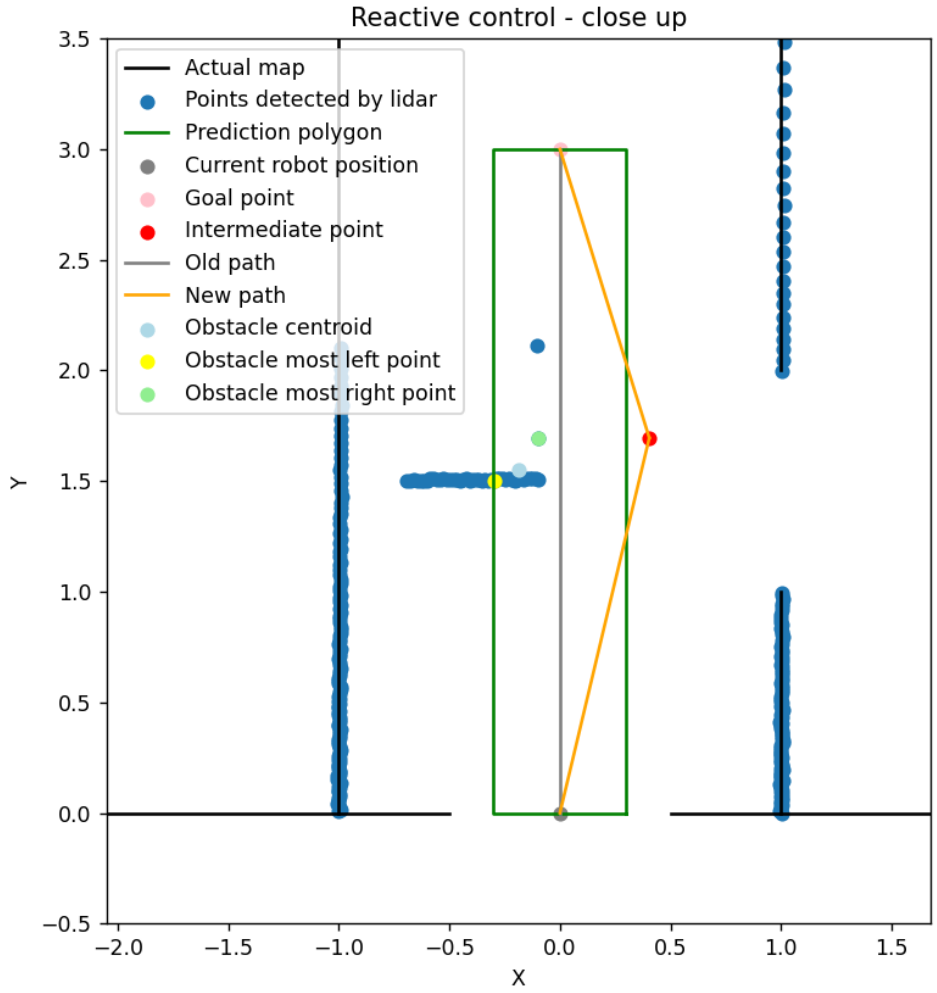


Figure 6: Reactive control visualization example.

4.1.3 Recursivity

Once the intermediate point is calculated, the robot starts moving toward it, effectively making it the new goal point. During this phase, reactive control remains active, and if a new obstacle is encountered while moving toward the new goal point, the function is called again. The system can thus reach a situation where reactive control is invoked recursively multiple times, and several intermediate points are calculated in a cascading sequence.
If the system were left free to evolve under these conditions, the robot would attempt to reach the last calculated intermediate point, then the second to last, then the third to last, and so on. However, this is problematic because:

- The robot's actual task is to reach the initial goal point, the one defined before reactive control was first triggered, rather than a series of intermediate points.
- The risk of collision with an obstacle increases drastically since there is nothing preventing an intermediate point from falling directly inside an obstacle.

Thus, to resolve this situation, using two boolean variables, the reactive control ensures that once any intermediate point is reached, the robot will once again aim for the original goal point.

4.2 Reverse the path using A*

In order to go back to the entry of the facility and be ready for new patrols. Marvin does not simply backtrack it's entire path, that would be very inefficient as he does not need to explore again all the rooms. In order to find the shortest path we first defined a graph of the facility using the navigation waypoints. (see Figure 7)

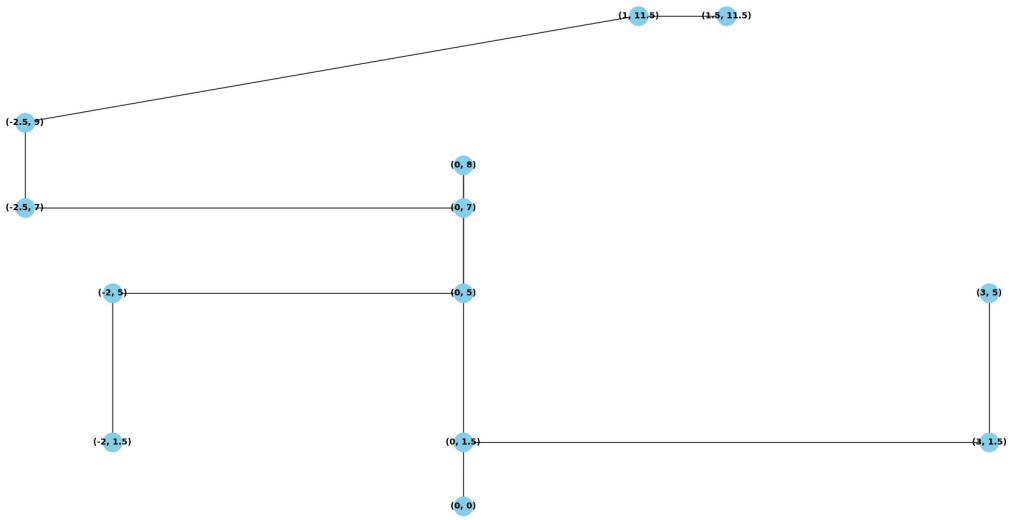


Figure 7: Graph of the facility

Once Marvin has reached the victim position an A* algorithm using the euclidian distance for heuristic is used to compute to shortest path home. Beforehand the current position is added to the graph and an edge is made with the point Marvin was heading to before detecting the balisa. (see Figure

8, here the robot found the victim in the bathroom at (2,12) and first saw it when going to (1.5,11.5))

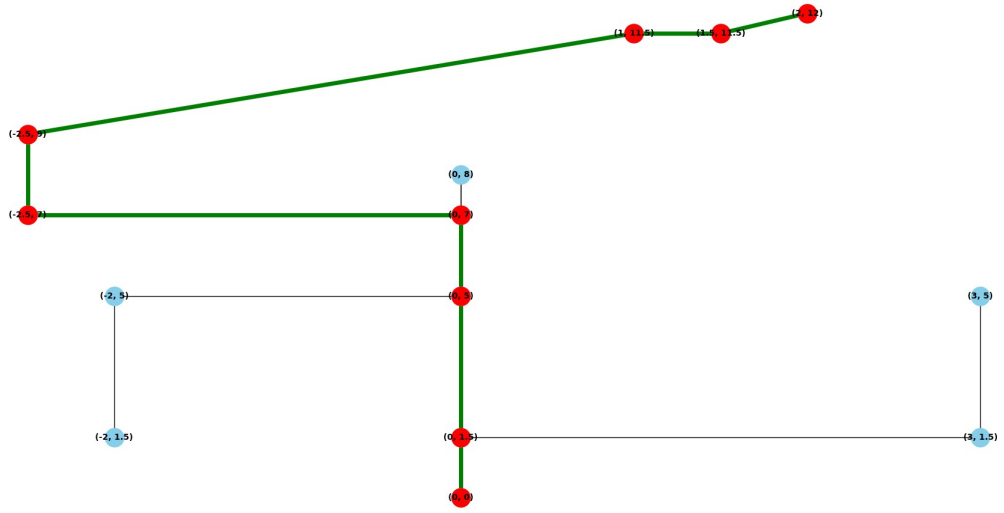


Figure 8: A* path used to backtrack from [2,12]

5 Localization - Extended Kalman Filter

The localization process employs an Extended Kalman Filter (EKF) for pose estimation. This algorithm is first initialized to set up key matrices: the **R matrix**, representing the measurement variances and covariances; the **P matrix**, representing the variances and covariances of the pose estimation; and the **X matrix**, which stores the pose estimation itself. [6]

5.1 Initialization of Matrices

- **R Matrix**: The measurement variance and covariance matrix is initialized using experimentally determined variances (see 5.4). These values are critical to modeling the uncertainties in the sensor measurements.
- **P Matrix**: The pose estimation variance and covariance matrix is also initialized based on experimental findings (see 5.5). This matrix captures the uncertainty in the robot's pose.
- **X Matrix**: This matrix is initialized to the robot's initial pose and is updated iteratively during the localization process.

5.2 Inputs to the Algorithm

At each step, the EKF algorithm takes the following inputs:

- **Control Command**: The command sent to the robot since the previous iteration and its duration.
- **Odometry Measurements**: The increments in x, y, and yaw (θ) provided by the simulator.
- **Landmark Positions**: The positions of the landmarks used for observations (see 5.7)

5.3 Algorithm Workflow

At every step, the algorithm proceeds as follows:

1. **Pose Prediction**: Using a linear approximation of the robot's kinematic model, the previous pose estimation, and the odometry increments, the algorithm predicts the next pose of the robot.

2. **Observations**: Observations are made using landmarks (see 5.7). These landmarks are strategically placed in the middle of doorways. Passing through narrow doors represents scenarios where precise state estimation is crucial, so corrections are designed to occur in these regions.

3. **Correction**: The observed landmark positions are compared to the predicted positions using the EKF correction step. The algorithm gives greater weight to the data source with lower uncertainty (either odometry or sensor observations), leading to an improved pose estimate.

Through experimental analysis, we found that reducing the weight of the observations (Kalman Gain) was necessary to stabilize the system. This adjustment might be related to an underlying calculation error that we have not yet been able to identify

The result of this process is a refined estimation of the robot's position. This estimated pose is subsequently provided to the robot's controller, which determines the actions the robot should take.

5.4 Sensor Calibration

The sensor used in the Kalman filter for observation is the **LMS100** laser scanner (see 2), operating in range/bearing mode. Since the bearing is used in the observation model, it is crucial to determine the variances and covariances associated with this measurement.

To calibrate the sensor, we first conducted an experiment to determine whether the variance of the bearing measurement depends on the distance to the target. This was done by taking measurements at various distances from a fixed landmark. However, after analyzing the data, we found no correlation between the distance and the variance of the bearing.

Next, we conducted another experiment where we fixed the distance to the landmark and varied the angle to the landmark. Again, we found no correlation between the angle and the variance of the bearing.

Based on the results from both experiments, the model used for the noise in the observation process assumes a **fixed variance with no covariance**, meaning that the variance does not depend on the distance or angle of the measurement. The fixed variance value for the bearing measurement is $\mathbf{Variance} = 0.00045 rads$**.**

Each of these experiments consisted of a series of measurements taken over a long period of time (hundreds of seoonds). The long duration ensures that the results are statistically robust and provide an accurate representation of the sensor's behavior under typical operating conditions.

5.5 Odometric System Calibration

A similar set of tests was conducted to determine the variances and covariances of the odometry measurements for the x, y, and yaw noise components. During the calibration process, we found no correlation (no covariances) between the noises in the x, y, and yaw directions.

The variances were also found to be independent from the angular and linear speeds of motions.

- **Yaw Variance:** The yaw variance is fixed at a value of $0.00036 rads$.
- **X Variance:** The variance in the x-direction was determined to be $0.335m$.
- **Y Variance:** The variance in the y-direction was determined to be $0.335m$.

5.6 Cinematic Model and Control Model

The Jacobians of the motion and control models are essential components of the Kalman filter prediction step. These matrices linearize the motion and control models around the current state and control inputs, enabling the propagation of uncertainty.

Jacobian of the Motion Model A_k: This matrix represents the relationship between the robot's current state and the predicted state after applying the control commands.

Motion Model

In a differential drive robot, the motion model describes how the robot's state evolves over time based on its control inputs, specifically the linear velocity v and angular velocity w. The robot's state is represented by x_k, y_k, and θ_k, which denote the robot's position and orientation at time k.

The motion model is typically described as:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + v \cdot \Delta t \cdot \cos(\theta_k + \frac{w \cdot \Delta t}{2}) \\ y_k + v \cdot \Delta t \cdot \sin(\theta_k + \frac{w \cdot \Delta t}{2}) \\ \theta_k + w \cdot \Delta t \end{bmatrix}$$

The above model describes how the robot moves under the influence of its control inputs v and w, over a time duration of Δt.

Jacobian of the Motion Model A_k

The Jacobian A_k linearizes the motion model. It provides information on how small changes in the robot's state (x_k, y_k, θ_k) affect the predicted state after applying the control commands.

The Jacobian A_k is:

$$A_k = \begin{bmatrix} 1 & 0 & -v \cdot \Delta t \cdot \sin\left(X_k[2] + \frac{w \cdot \Delta t}{2}\right) \\ 0 & 1 & v \cdot \Delta t \cdot \cos\left(X_k[2] + \frac{w \cdot \Delta t}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

Control Model

The control model captures how the robot's control inputs, specifically the linear velocity v and angular velocity w, influence its motion over a time step Δt. The change in the robot's state due to these inputs is given by the Jacobian B_k.

Jacobian of the Control Model B_k

The Jacobian B_k reflects the dependence of the robot's movement on both control inputs v and w, considering the duration Δt.

The Jacobian B_k is:

$$B_k = \begin{bmatrix} \Delta t \cdot \cos\left(X_k[2] + \frac{w \cdot \Delta t}{2}\right) & -0.5 \cdot v \cdot \Delta t^2 \cdot \sin\left(X_k[2] + \frac{w \cdot \Delta t}{2}\right) \\ \Delta t \cdot \sin\left(X_k[2] + \frac{w \cdot \Delta t}{2}\right) & 0.5 \cdot v \cdot \Delta t^2 \cdot \cos\left(X_k[2] + \frac{w \cdot \Delta t}{2}\right) \\ 0 & \Delta t \end{bmatrix}$$

Both matrices are essential for the prediction step in the Kalman filter, allowing the uncertainty in the state estimate to be propagated correctly.

5.7 Observation Model

The sensor used in the Kalman filter for observation is the **LMS100** laser scanner (see 2), operating in range/bearing mode toward landmarks placed at door entries.

The observation provided by the sensor is the **bearing angle** measured between the robot and the landmark. Specifically, the bearing angle ϕ is calculated as the angle relative to the robot's orientation θ. The observation model is expressed as follows:

$$\mathbf{z} = \begin{bmatrix} \phi \end{bmatrix} = \text{atan2}(y_l - y, x_l - x) - \theta$$

Where:

- (x_l, y_l) are the coordinates of the landmark.
- (x, y) are the robot's coordinates.
- θ is the robot's orientation (yaw).
- $\text{atan2}(y_l - y, x_l - x)$ gives the angle of the landmark relative to the robot's position in the global frame.

The Jacobian matrix H of the observation model with respect to the robot's state $\mathbf{x} = [x, y, \theta]^T$ is given by:

$$H = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \theta} \end{bmatrix}$$

Where the partial derivatives are:

$$\frac{\partial \phi}{\partial x} = \frac{y_l - y}{(x_l - x)^2 + (y_l - y)^2}, \quad \frac{\partial \phi}{\partial y} = \frac{x - x_l}{(x_l - x)^2 + (y_l - y)^2}, \quad \frac{\partial \phi}{\partial \theta} = -1$$

Thus, the Jacobian matrix is:

$$H = \begin{bmatrix} \frac{y_l - y}{(x_l - x)^2 + (y_l - y)^2} & \frac{x - x_l}{(x_l - x)^2 + (y_l - y)^2} & -1 \end{bmatrix}$$

5.8 Results

Influence of various parameters

The performance of the EKF localization is heavily influenced by several key parameters. In this section, we analyze the most critical factors. As a baseline for comparison, we use the trajectory shown in Figure 9.

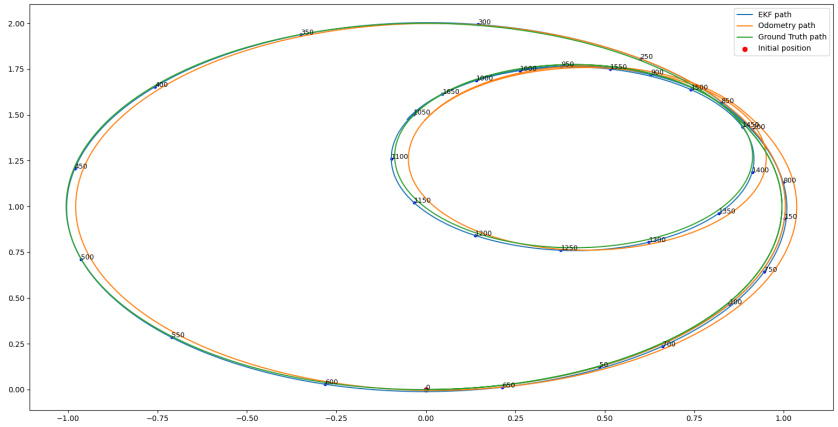


Figure 9: EKF performance on a circular, converging trajectory. - Adequate Tuning

5.8.1 Overestimating R

When the measurement noise covariance R is overestimated, the Kalman gain W_k for corrections is reduced. Consequently, the filter places significantly more trust in the odometry than in the observations. This behavior, which also occurs when the process noise Q_k is underestimated, results in the trajectory deviating closer to the odometry path, as illustrated in Figure 10.

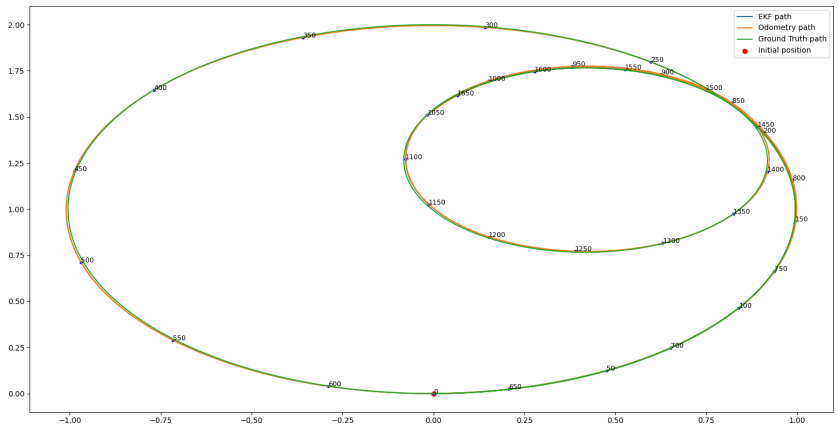


Figure 10: Trajectories with overestimated measurement noise.

5.8.2 Overestimating Q_k or $P(0)$

On the other hand, when the process noise Q_k or the initial covariance $P(0)$ is overestimated, the system relies excessively on noisy observations. This over-reliance can disrupt stability, as shown in Figure 11. In such cases, the trajectory may oscillate or diverge from the expected path.

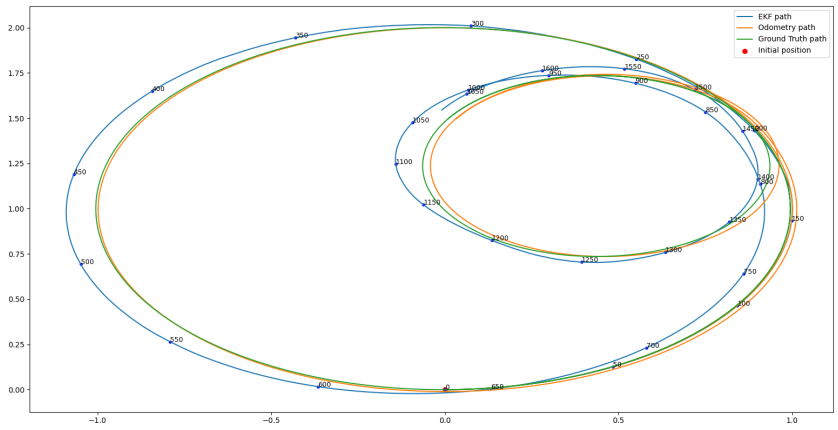


Figure 11: Trajectories with overestimated process noise or initial covariance matrix $P(0)$.

Final Results

The following figure shows the recorded trajectories during a patrol mission where Marvin successfully located a victim.

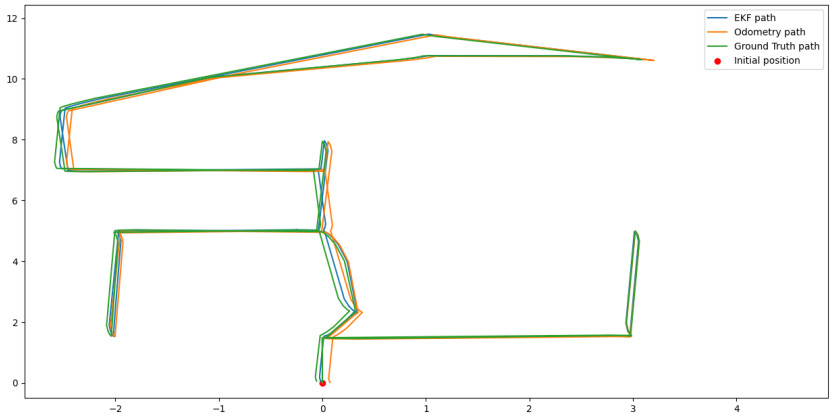


Figure 12: Robot trajectories during a patrol mission.

6 Conclusion

In this work, we developed and implemented a patrol robot in senior care facilities. Using the Apolo simulator and a differential drive "Pioneer3AT" robot model.

Key strategies such as reactive control for collision avoidance, path planning with A* search algorithms, and an Extended Kalman Filter (EKF) for accurate localization were implemented. The combination of these modules allowed the robot "Marvin" to effectively perform its patrol duties, detecting emergency situations.

7 How to Test

To test the code, it is accessible from this GitHub repository: here.

To run the simulation in Apolo, open **map.xml**. Then, launch **main.py** to start execution.

During the simulation:

- The patient (landmark 0) is randomly placed in accessible areas of the apartment.
- Paths and covariances will be plotted at the end.

- Intermediate points will be displayed when reactive control is triggered.
- The A* path will be plotted when used.

These pop-up windows must be closed to continue the simulation.

8 Work Distribution

Responsibility	Assigned To
Common	1. Selection of application and environment. 2. Calibration and variances estimation. 3. System architecture design. 4. Reactive control design. 5. System testing and tuning. 6. Report writing.
Tom	1. Waypoint navigation. 2. Behavior trees implementation (see 4, 5). 3. A* and "Get Home" behavior. 4. EKF development.
Edoardo	1. Environment creation and mapping. 2. 3D object modeling for the inside of the environment. 3. Robustness improvements and edge cases of reactive control.

Table 1: Work Distribution Among Team Members

9 Bibliografía

References

[1] SICK AG. Sick lms100 lidar sensor, n.d. Accessed: 2025-01-09.

[2] Manuel Hernando. https://github.com/mhernando/apolo, 2025. Accessed: 2025-01-07.

[3] Spanish National Statistics Institute. Population pyramid 2050 spain, 2023. Accessed: 2025-01-07.

[4] Mdpi. Indoor monitoring system for patients with alzheimer's disease using lidar technology. *MDPI Sensors*, 22(20):7934, 2022. Accessed: 2025-01-09.

[5] Lidar News. Monitoring the elderly in homes using lidar, 2021. Accessed: 2025-01-09.

[6] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical Report TR 95-041, University of North Carolina at Chapel Hill, February 2001.