Robot Navigation in Presence of People

Konstantinos Christopoulos

Diploma Thesis

Supervisor: Konstantinos Vlachos

Ioannina, February, 2025

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Contents

Abstract

In this thesis, we present a novel multi-step pedestrian avoidance framework for mobile robots navigating dynamic environments. The proposed system integrates LiDAR-based pedestrian tracking, dynamic obstacle detection, and potential field navigation to ensure safe and efficient robot movement in human spaces. We implement on the Jackal robot, integrating advanced hardware components such as the Velodyne VLP-16 LiDAR to collect rich 3D point cloud data. The avoidance methodology employs multi-step filtering to distinguish static and dynamic obstacles, dynamic point clustering via DBSCAN, and Kalman filters for robust pedestrian tracking and prediction pedestrian trajectories. As far as the pedestrian movement, we use the Social Force Model (SFM), combining attractive and repulsive forces. A potential field approach, guides the robot toward a safe destination, while maintaining real-time adaptability to changing conditions. Experimental validation demonstrates the system's robustness and effectiveness in collision-free navigation. This work provides an essential contribution to the development of pedestrian-aware robotic systems, enhancing their applicability in real-world scenarios.

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Autonomous navigation in dynamic environments presents significant challenges for mobile robots, particularly when interacting with pedestrians. Human-robot interaction requires systems that can efficiently predict pedestrian movement and adapt robot trajectories to ensure safety in navigation. The presence of unpredictable human motion, varying obstacle densities, and real-time decision-making demands robust solutions that balance efficiency and reliability.

Consequently, there is a growing need for advanced navigation strategies that enable robots to anticipate and react to human behavior. This thesis aims to bridge these gaps by developing a robust pedestrian avoidance framework capable of real-time operation in dynamic environments.

1.2 Overview of Our Approach

To address these challenges, this thesis proposes a multi-step pedestrian avoidance system for mobile robots operating in dynamic environments. The approach integrates LiDAR-based tracking, pedestrian motion modeling, and potential field navigation into a cohesive framework.

The system begins by utilizing a Velodyne VLP-16 LiDAR sensor to capture detailed 3D point cloud data of the environment. Advanced filtering techniques, including map-based and motion-based filtering, isolate dynamic obstacles from static ones. Dynamic obstacle tracking is achieved through DBSCAN clustering and Kalman filters, enabling precise localization of pedestrians and prediction of their movement.

Building on these capabilities, the system employs a potential field-based navigation module to

move the robot to an optimal free position. By combining attractive forces toward goal points and repulsive forces from pedestrians and obstacles, the robot navigates safely and efficiently. The proposed method emphasizes real-time responsiveness and adaptability, ensuring seamless interaction with pedestrians and minimal disruption to their paths.

Through this integrated approach, the thesis demonstrates the feasibility and effectiveness of pedestrian-aware navigation for mobile robots, paving the way for more robust applications in dynamically crowded environments.

Chapter 2

Related Work

2.1 Pedestrian Avoidance in Robotics

Pedestrian avoidance is a critical topic in the field of autonomous robotics, particularly for robots operating in dynamic, crowded environments. In recent years, advancements in LiDAR technology have enabled robots to perceive their surroundings with higher accuracy and detail. LiDAR-based pedestrian detection methods, such as those proposed by Brščić et al. [1], demonstrate the ability to estimate human positions and movements effectively. These techniques, however, are often limited by their computational demands and susceptibility to noise in crowded environments.

2.2 Social Force Models for Human Motion Prediction

The Social Force Model (SFM), introduced by Helbing and Molnár [2], has become a cornerstone in pedestrian motion prediction. SFM models human behavior as a combination of attractive and repulsive forces, capturing interactions between pedestrians and their environment. This approach has been widely adopted in robotics for its ability to represent complex social dynamics. Moussaïd et al. [4] expanded upon this framework to include group behaviors and self-organization phenomena in crowds.

Despite its versatility, SFM faces limitations in real-time robotics applications due to its reliance on manually tuned parameters and its simplified assumptions about human behavior. Efforts to integrate SFM with machine learning techniques have shown promise in addressing these challenges, enabling more adaptive and context-aware motion prediction models.

2.3 Dynamic Obstacle Tracking

Accurate tracking of dynamic obstacles is essential for pedestrian avoidance. Techniques such as DBSCAN clustering and Kalman filtering have emerged as robust solutions for real-time multi-object tracking. DBSCAN, as a density-based clustering algorithm, is particularly effective for isolating pedestrians in noisy LiDAR data. Kalman filters complement this process by predicting the future states of tracked objects, accounting for measurement noise and uncertainties [6].

Multi-object tracking systems leveraging these methods have demonstrated significant improvements in pedestrian detection and tracking. However, challenges such as data association in crowded scenes and the computational cost of maintaining multiple trackers persist. Recent research has explored the use of deep learning-based object detection and tracking to further enhance robustness and scalability.

2.4 Navigation with Potential Fields

Potential field-based navigation is a well-established method for collision-free motion planning. Khatib [3] first proposed the use of artificial potential fields to guide robots, where attractive forces pull the robot toward the goal and repulsive forces push it away from obstacles. This approach has been widely adopted for its simplicity and effectiveness in real-time applications.

Recent advancements have extended potential field methods to dynamic environments, incorporating real-time updates for moving obstacles and pedestrians. Omar et al. [5] reviewed various enhancements to potential field methods, highlighting their adaptability to complex scenarios. However, potential field-based navigation can suffer from local minima, where the robot becomes trapped between conflicting forces. Hybrid approaches combining potential fields with global path planning algorithms have shown promise in mitigating this issue.

2.5 Contribution of This Work

This thesis builds upon the aforementioned advancements by integrating LiDAR-based pedestrian detection, Social Force Models, and potential field navigation into a unified system. By combining advanced filtering and tracking methods with a reactive navigation framework, the proposed approach aims to address the limitations of existing systems, such as their computational inefficiencies and difficulties in handling dynamic environments.

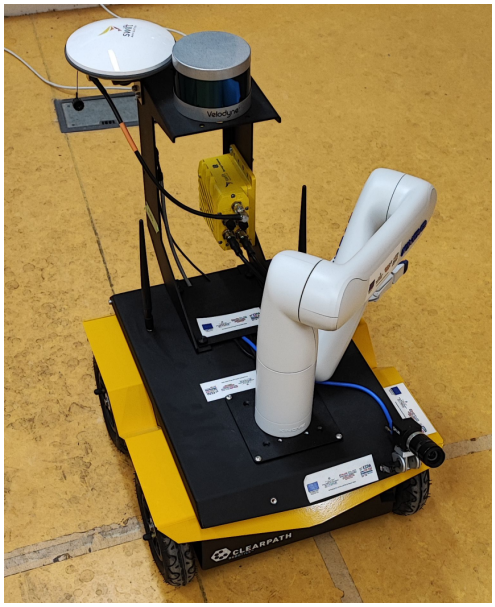
Chapter 3

System Architecture

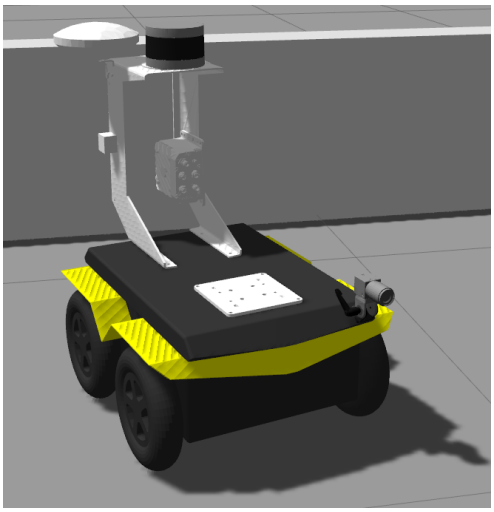
3.1 Hardware Setup

3.1.1 Jackal Robot

Our pedestrian avoidance and tracking system is deployed on the Clearpath Jackal—a four-wheeled rugged, indoor/outdoor mobile robotic platform (figure 3.1). The Jackal provides the necessary mobility and onboard computation for running the ROS (Robot Operating System) nodes associated with LiDAR scanning, navigation, and control.



(a) Jackal in the University lab.



(b) Jackal in Simulation (Gazebo). Note: The robotic arm is not included in the visualization because it is not used in the system.

Figure 3.1: Jackal Robot.

Key Features:

- **Mobility:** Differential drive system for precise maneuverability, with a maximum speed of 2 m/s.
- **Payload:** Supports a variety of sensors and payloads, including LiDAR, cameras, and additional computational units.

For this study, the Jackal serves as the mobile base for implementing the pedestrian avoidance system. It facilitates real-time LiDAR data acquisition, decision-making, and path planning. Additionally, its modular design allows seamless integration with the Velodyne VLP-16 LiDAR sensor and other hardware components.

3.1.2 Velodyne LiDAR

We use a Velodyne VLP-16 LiDAR sensor (figure 3.2) mounted on the robot to obtain rich 3D point cloud data of the environment. The VLP-16 offers a 360-degree horizontal field of view and a vertical coverage of ±15°, enabling the detection of pedestrians and static obstacles in typical indoor and outdoor spaces.

Figure 3.2: Velodyne VLP-16 LiDAR sensor.

Key Specifications:

- **Vertical Channels:** 16.
- **Horizontal Field of View:** 360°
- **Range:** Up to 100 m.
- **Rotational Rate:** Configurable (up to 20 Hz).

The LiDAR operates by emitting laser pulses that reflect off objects in the environment. These reflections are used to generate a 3D point cloud that accurately represents the surrounding space. This data is processed using filtering techniques to isolate dynamic points (e.g., pedestrians) from static obstacles.

Role in the System:

- **Obstacle Detection:** Captures detailed environmental data, enabling the identification of both static and dynamic obstacles.
- **Pedestrian Tracking:** Provides accurate positional data for clustering and tracking pedestrians using DBSCAN and Kalman filtering techniques.
- **Navigation:** Supplies real-time input for potential field-based navigation, allowing the robot to calculate collision-free paths.

The VLP-16 provides high-resolution point clouds, making it well-suited for detecting static and dynamic obstacles, such as pedestrians.

3.2 ROS Integration

We implement our system using ROS (Robot Operating System), leveraging a publisher–subscriber architecture to share data across multiple nodes. Below is a high-level overview of the main topics, frames, and nodes.

3.2.1 Topics

- **LiDAR Driver:**
  - **Topic:** */velodyne_points* or */velodyne_points_downsampled* (if using a downsampling pipeline).
  - **Frame:** */velodyne* - the sensor's local frame.
- **Localization:**
  - **Topic:** */amcl_pose* (PoseWithCovarianceStamped message type).
  - **Frame:** */map* - the global, navigation frame for localization.
- **Occupancy Grid:**
  - **Topic:** */map* (OccupancyGrid message type).

  - Provides static environment data for map-based filtering and navigation layers.

- **Pedestrian Tracking:**
  - **Topic:** */pedestrian_tracker* (Odometry message type).
  - Publishes the estimated pedestrians's pose and velocity.
- **Command Velocity:**
  - **Topic:** */cmd_vel* (Twist message type).
  - The robot's low-level velocity commands for linear and angular motion.

3.2.2 Node-level Overview

1. **Pedestrian Tracking Node (*pedestrian_tracking_node.py*)**
   - Subscribes to LiDAR data for filtering out static points.
   - Clusters the remaining data to track pedestrians using Kalman filters.
   - Publishes pedestrian odometry estimates on */pedestrian_tracker*.
2. **Pedestrian Detection Node (*pedestrian_detection_node.py*)**
   - Subscribes to the pedestrian odometry from the tracker.
   - Identifies if the pedestrian is approaching.
   - Finds an optimal avoidance point in the local environment and publishes that point.
3. **Static Obstacles Node (*static_obstacles_node.py*)**
   - Subscribes to */map* and LiDAR.
   - Performs static-obstacle filtering (opposite logic from dynamic filtering), publishing filtered point clouds on */static_points* topics.
4. **Robot Movement Node (*robot_movement_node.py*)**
   - Receives an "optimal point" from the detection node.
   - Computes potential field forces to guide the robot while avoiding both static and dynamic obstacles.
   - Publishes velocity commands on */cmd_vel*.

3.2.3 Overview of Data Flow

The Fig. 3.3 illustrates the typical data flow among our nodes:

1. **LiDAR Data Acquisition**
   - The Velodyne VLP-16 publishes raw (or downsampled) point clouds on */velodyne_points_downsampled* topic.
   - Each point cloud is time-stamped in the */velodyne* frame.
2. **Frame Transform and Filtering**
   - The pedestrian tracking node transforms LiDAR data to the */map* frame using TF.
   - We apply either map-based, motion-based, or combined filtering to remove static obstacles and isolate dynamic points.
3. **Clustering and Kalman Tracking**
   - Post-filtering, the node clusters the dynamic points, then applies a multi-object tracker (e.g., one Kalman filter per cluster) to estimate pedestrian states.
   - The resulting pedestrian state is published as an Odometry message on *pedestrian_tracker*.
4. **Pedestrian Detection and Avoidance**
   - A separate node (the "detection node") subscribes to *pedestrian_tracker*, identifies whether a pedestrian is approaching, and computes an "optimal avoidance point" if needed.
   - This "optimal point" is published in a suitable frame (e.g., *map*) for the navigation node.
5. **Potential Field Navigation**
   - The "robot movement node" or navigation node subscribes to the optimal point.
   - It also listens to *amcl_pose* for the robot's current pose and to */static_points* for additional obstacle data.
   - By combining attractive (toward the optimal point) and repulsive (away from obstacles and pedestrians) forces, it generates a velocity command on */cmd_vel* to drive the Jackal.

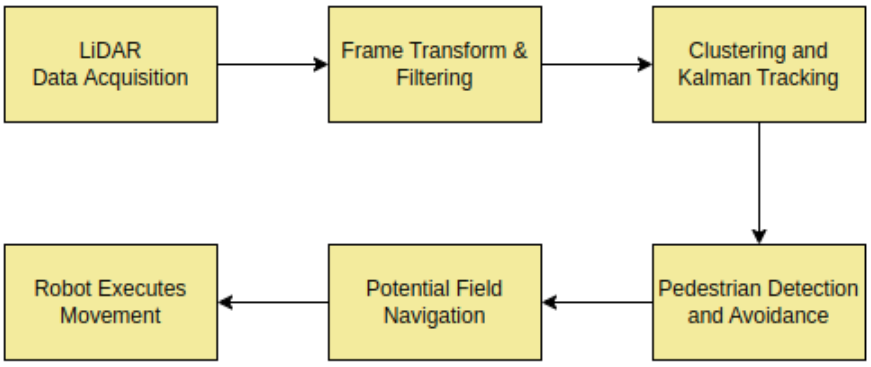


Figure 3.3: Overview of Data Flow

6. **Robot Executes Movement**

   - The Jackal's internal controller processes */cmd_vel* to adjust the wheels' velocities, physically moving the robot through the environment toward the goal.

Through these interconnected nodes, we maintain a robust perception–decision–control pipeline. The concurrency of ROS ensures that each node independently processes data at the required frequency. Ultimately, the Jackal can detect moving pedestrians, predict their motions, and adapt its path to avoid collisions in real time.

Chapter 4

Pedestrian Movement

4.1 Introduction

As far as pedestrian movement, we use the Social Force Model. The Social Force Model (SFM) has emerged as one of the most influential approaches in this domain, providing a mathematical framework that describes how individual agents interact with one another and their surroundings. Specifically, this model describes the dynamics of pedestrians as influenced by imaginary "social forces" (see Fig 4.1). These forces represent internal motivations rather than external physical forces and include elements such as the desire to maintain a certain speed to a destination and the need to maintain distance from other pedestrians and obstacles.

4.2 GitHub

In our work we use the GitHub repository *pedsim_ros* [2][4], that provides a ROS (Robot Operating System) package dedicated to simulating pedestrian behavior based on the Social Force Model (SFM). This model, originally introduced by Helbing and Molnár, seeks to capture the intricate interactions that arise when multiple pedestrians (or “agents”) navigate through shared spaces.The repository offers both a simulation environment and the core computational tools needed to study, visualize, and evaluate crowd dynamics in various scenarios.

Through a detailed examination of *pedsim_ros*, we manage to highlight the theoretical foundations of the Social Force Model, discuss its implementation and integration into ROS, and demonstrate its practical utility in simulating pedestrian movement.

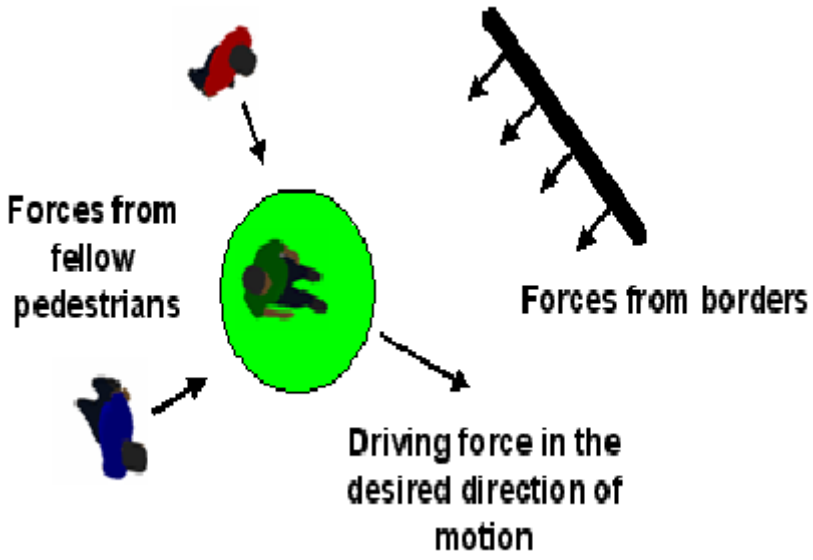


Figure 4.1: Social Force Model

4.3 Formulation of the Social Force Model

The model defines the forces that affect a pedestrian α and are as follows:

- The **Desired force** that represents a person's desire to move in a particular direction, independent of other people and obstacles.
- The **Obstacle force** which is a repulsive force to avoid conflicts with obstacles in the environment.
- The **Social force** caused by the interaction between individuals that causes them to avoid each other to avoid conflict.
- The **Robot force** which is a repulsive force to avoid conflicts with the robot in the environment.

4.3.1 Desired Force

This force pulls the pedestrian α towards the current destination. Each pedestrian aims to move towards a specific destination, usually the next target point r_a^k on his way to the final destination r_a^0. The desired direction (4.1) is calculated as the normalized vector pointing from the pedestrian's current position $r_a(t)$ to the next target r_a^k.

- Desired direction

$$e_a(t) := \frac{r_a^k - r_a(t)}{\|r_a^k - r_a(t)\|} \tag{4.1}$$

- Desired force

$$\overrightarrow{F_a^0}(\vec{u_a}, \vec{u_a}\vec{e_a}) := \frac{1}{\tau_a}(u_a^0\vec{e_a} - \vec{u_a}) \tag{4.2}$$

Where $\overrightarrow{F_a^0}$ represents the acceleration rate or the desired force that brings the actual speed of the pedestrian u_a closer to the desired speed. The factor $\frac{1}{\tau_a}$ introduces a relaxation time τ_a, which dictates how quickly the pedestrian adjusts his current speed to match his desired speed. A shorter τ_a indicates faster adaptation. This force (4.2) causes the pedestrian to match the desired direction and speed with the passage of time, reflecting his goal to reach his destination as comfortably and correctly as possible.

4.3.2 Obstacle Force

A pedestrian α also keeps a certain distance from borders of walls or obstacles. The pedestrian feels more uncomfortable the closer to a border he/she walks since he/she has to pay more attention to avoid the danger of getting hurt, e.g. by accidentally touching a wall. Therefore, a border B evokes a repulsive effect that can be described by:

$$F_{aB}(\vec{r}_{aB}) := -\nabla_{r_{aB}} U_{aB}(\|\vec{r_{aB}}\|), \tag{4.3}$$

where $\vec{r_{aB}} = r_a - r_B$ is the vector from the pedestrian α to the closest point to an obstacle B. We also use a repulsive and monotonic decreasing potential function $U_{aB}(||\vec{r_{aB}}||)$. This function (4.3) decreases monotonically with distance, reflecting the way in which the pedestrian feels more uncomfortable when approaching an obstacle. Specifically, we used an exponential function:

$$U_{aB}(\|\vec{r_{aB}}\|) := U_0 e^{-\frac{\|\vec{r_{aB}}\|}{B}}, \tag{4.4}$$

where U_0 is a constant that represents the power of repulsion (we assign it to $U_0 = 1$) and B is a parameter that controls the "width" of the repulsion. The larger the B, the more gradually the repulsive force decreases with distance. So, the direction of the repulsive force is:

$$F_{aB} = \frac{1}{B} e^{-\frac{\|\vec{r_{aB}}\|}{B}} \frac{\vec{r_{aB}}}{\|\vec{r_{aB}}\|} \tag{4.5}$$

4.3.3 Social Force

The motion of a pedestrian α is influenced by other pedestrians β. Especially, he/she keeps a certain distance from other pedestrians that depends on the pedestrian density. A pedestrian normally feels increasingly uncomfortable the closer he/she gets to a strange person. This results in repulsive effect of other pedestrians β that can be represented by vectorial quantities.

The pedestrian interactions are described between two components f_υ and f_θ representing the deceleration along the interaction direction $t_{\alpha\beta}$ and the directional changes along $n_{\alpha\beta}$ respectively.

- $\theta_{\alpha\beta}$ is the angle formed between the direction of interaction $t_{\alpha\beta}$ and the vector pointing from pedestrian α to β.
- The vector $n_{\alpha\beta}$ is reported to be the normalized vector of $t_{\alpha\beta}$ oriented to the left.
- We specify the interaction direction $t_{\alpha\beta}$ (4.7) as a composition of the direction of relative motion $(u_\alpha - u_\beta)$ and the direction:

$$e_{\alpha\beta} = \frac{x_\beta - x_\alpha}{\|x_\beta - x_\alpha\|} \tag{4.6}$$

in which pedestrian β is located, where x_α is the location of pedestrian α. This leads to:

$$t_{\alpha\beta} = \frac{D_{\alpha\beta}}{\|D_{\alpha\beta}\|} \tag{4.7}$$

with:

$$D_{\alpha\beta} = \lambda(u_\alpha - u_\beta) + e_{\alpha\beta}, \tag{4.8}$$

where the weight λ reflects the relative importance of the two directions.

The $D_{\alpha\beta}$ denotes the distance between two pedestrians α and β, and $\theta_{\alpha\beta}$ the angle between the interaction direction $t_{\alpha\beta}$ and the vector pointing from pedestrian α to β.

The resulting equations for the components are:

$$f_\upsilon(d, \theta) = -Ae^{-\frac{d}{B} - (n'B\theta)^2} \tag{4.9}$$

Here we describe the phenomenon of slowing down pedestrians. The force f_v decreases exponentially as the distance between 2 pedestrians increases ($d \uparrow$).

Furthermore, the term $(n'B\theta)^2$ ensures that the decrease of the force f_v is faster for larger angles θ. As a result, the force is stronger when the pedestrians are in front of each other and weaker to the sides.

$$f_\theta(d,\theta) = -AKe^{-\frac{d}{B}-(nB\theta)^2} \tag{4.10}$$

Here the function f_θ is similar to the f_v with the difference that now we

The force f_θ focus more on the directional changes. The parameter $n < n'$ indicates a wider range for angular interactions compared to deceleration.

Parameter B reflects that the strength of the interaction increases with higher relative speeds ($||D||$). This, in turn, represents the need for more significant adjustments in repulsive force to avoid "running" pedestrians.

The $K = \frac{\theta}{|\theta|}$ parameter represents the direction of angular movement (left or right). This reflects the binary decision of avoidance to the left or right, depending on the sign of the angle.

Finally we'll have the function of **Social force** (4.11):

$$F_{\alpha\beta}(d,\theta) = -Ae^{-\frac{d}{B}}[e^{-(n'B\theta)^2}t_{\alpha\beta} + e^{-(nB\theta)^2}n_{\alpha\beta}] \tag{4.11}$$

4.3.4 Robot Force

It behaves similarly to **Obstacle force** (4.3.2), with the only difference that now we take the distance between the human and the robot.

$$r_{robot,\alpha} = r_{robot} - r_\alpha \tag{4.12}$$

This is the vector (4.12) between the robot and the pedestrian α.

The **Robot force** will be:

$$F_{robot} = -e^{-\frac{\|r_{robot}-r_a\|}{\sigma_{robot}}}\frac{r_{robot}-r_a}{\|r_{robot}-r_a\|} \tag{4.13}$$

The parameter σ_{robot} indicates how fast the F_{robot} decreases with distance.

As far as the f_{robot} is concerned, the pedestrian is repelled by the robot in a way that decreases exponentially with distance, with stronger repulsion at closer distances.

4.3.5 Total Force

We can now set up the equation for a pedestrian's total motivation $\vec{F_{total}}$. Since all the previously mentioned effects influence a pedestrian's decision at the same moment, we will assume that their total effect is given by the sum of all effects, like this is the case for forces. This result in:

$$F_{total} = F_a^0(t) + F_{aB}(t) + F_{\alpha\beta}(t) + F_{robot} \tag{4.14}$$

4.3.6 Velocities & Positions

According to the 2nd law of the Newton we have:

$$\vec{F} = m\vec{a} \tag{4.15}$$

For reasons of simplification we consider that $m = 1$. The Social Force Model is now defined by:

$$\frac{\partial u_a}{\partial t} = \vec{a} = \vec{F_{total}} \tag{4.16}$$

In order to complete the model of pedestrian dynamics we should introduce a relation between the actual velocity $\vec{u_a}(t)$ and the prefered velocity $\vec{w_a}(t)$.
Since the actual speed is limited by a pedestrian's maximal acceptance speed u_a^{max}, we will assume that the realized motion is given by:

$$\frac{\partial r_a}{\partial t} = \vec{u_a}(t) = \vec{w_a}(t) g(\frac{u_a^{max}}{\|\vec{w_a}\|}) \tag{4.17}$$

with:

$$g(\frac{u_a^{max}}{||\vec{w_a}||}) := \begin{cases} 1 & \text{if } \|\mathbf{w}_\alpha\| \leq v_\alpha^{\max}, \\ \frac{v_\alpha^{\max}}{\|\mathbf{w}_\alpha\|} & \text{otherwise.} \end{cases} \tag{4.18}$$

After that we use the Euler method.We take small steps forward in time, and use the gradient at the current point to estimate the value at the next step. So, we calculate velocity and position

at each time step.

$$u_a(t+1) = u_a(t) + stepSize * \vec{a} \tag{4.19}$$

$$r_a(t+1) = r_a(t) + stepSize * u_a(t+1) \tag{4.20}$$

This process is repeated over time to simulate the movement of the pedestrian in its environment.

Chapter 5

Robot Avoidance

5.1 Introduction

We implement a multi-step pedestrian avoidance system for robotic navigation, integrating map-based and motion-based filtering for static obstacle removal, multi-object tracking with Kalman filters for pedestrian localization, and a local avoidance module that identifies safe destinations to move the robot. First, we leverage LiDAR data and occupancy grid information to isolate static and dynamic obstacles. Next, we cluster and track the dynamic points over time to estimate velocities. Finally, a potential field approach ensures collision-free navigation by combining attractive and repulsive forces to guide the robot to an optimal, safe goal location.

5.2 Obtaining People Point Clouds from LiDAR Data

In many robotics applications, especially those involving human-robot interaction, it is crucial to distinguish static obstacles (e.g., walls, furniture) from dynamic entities (e.g., moving pedestrians). We use LiDAR data filtered both by a static occupancy grid map and by comparing an "initial" cloud snapshot with the latest data. This ensures that only truly dynamic points (likely pedestrians or other moving objects) remain for further tracking and navigation. [1]

5.2.1 Map-Based Filtering with KD-Tree

In map-based filtering, we remove points that coincide with static obstacles known from an occupancy grid map. Here, we transform the LiDAR point cloud from the sensor frame (*/velodyne*) to the global frame (*/map*), and then query a k-dimensional tree (KD-tree) built from the occupied cells of the map.

- First we take the occupnacy grid map from the topic */map*, which represented free and occupied spaces on the map and we convert it to 2D coordinates. The occupancy grid map is a 1D array where free spaces are indicated by "0" (indicating that the area is free). Obstacles are declared with "100".

  When converting the 1D array (*/map* topic) to 2D map coordinates, we start the conversion from the 1st element of the grid, which corresponds to the "bottom left corner" of the grid. The conversion continued along the columns/horizontal for each row, and once the end of a row was reached we moved on to the next one from the top. The grid runs from left to right for each row and then moves upwards from the bottom left corner.

  Each occupied cell (i, j) in the occupancy grid corresponds to a real-world coordinate (x, y), computed from:

$$\begin{aligned} x_{grid} &= i\%width \\ y_{grid} &= i//width \end{aligned} \tag{5.1}$$

  where $width$ is the number of cells along the length of the x-axis and i is the pointer 1D to the array.

$$\begin{aligned} x_{map} &= x_{grid} resolution + origin.x \\ y_{map} &= y_{grid} resolution + origin.y \end{aligned} \tag{5.2}$$

  where $resolution$ is the size of each cell in meters (e.g. for 0.1m means that each cell represents an area of $0.1x0.1$ meters). The $origin$ is the 2D coordinates of the bottom left corner of the map in the world frame.

- Once we collect all occupied points into a numpy array, we build a cKDTree. The cKDTRee partitions the map points, enabling fast searches (queries) to find the nearest neighbours of a data point. Our goal is to quickly locate and filter point cloud points that are close to known static map points, thus isolating dynamic elements in the environment.

- For each point $p = (x_p, y_p)$ in the transformed LiDAR cloud, we compute the distance d to the nearest occupied cell using:

$$d = \min_{\mathbf{m} \in M} \|\mathbf{p} - \mathbf{m}\|, \tag{5.3}$$

where $\mathbf{M}$ is the set of all occupied points (indexed in the KD-tree).

- If $d < \delta$ (a chosen threshold), we label that LiDAR point as "static" (i.e., part of the map) and remove it from the dynamic set.

5.2.2 Motion-Based Filtering (Comparisons of Initial vs. Current Clouds)

Whereas map-based filtering removes points that match the known static environment, motion-based filtering works by comparing the current LiDAR scan to an earlier "initial" snapshot. Points that have appeared or shifted significantly are treated as dynamic.

- Once again, we build a KD-Tree from the initial point cloud (C_{init}).
- For each point in C_{curr}, we compute:

$$d(\mathbf{p}_{curr}) = \min_{\mathbf{p}_{init} \in C_{init}} \|\mathbf{p}_{curr} - \mathbf{p}_{init}\| \tag{5.4}$$

- If $d(\mathbf{p}_{curr}) > \delta_{motion}$, we consider $\mathbf{p}_{curr}$ to be new or "moved", hence dynamic point.

It is also important to mention that we update the "initial" snapshot at regular intervals to reduce the likelihood of the robot identifying points corresponding to static obstacles as dynamic. This can happen after the robot has moved to a new position and the lidar "hit" to an obstacle that it didn't see before when we set the "initial" snapshot, and it detects it as a dynamic while it's static.

5.2.3 Motion Based Filtering for Static Points

In the previous subsection (5.2.2), we mentioned the detection of dynamic points in the environment using the "Motion-Based Method". So now we will use the same method again with the difference that we will locate static points.

The methodology is the same as the subsection (5.2.2), with the difference that now we want the distance of the points (5.4) to be less than the threshold (5.5).

$$d(\mathbf{p}_{curr}) \leq \delta_{motion} \tag{5.5}$$

where δ_{motion} is the threshold.

5.3 Dynamic Obstacle Detection and Tracking

Once static obstacles are removed from the LiDAR point cloud (via the filtering approaches described previously), the remaining points typically correspond to moving or previously unmapped objects. These dynamic points are crucial for pedestrian detection and avoidance. In our system, we first apply DBSCAN clustering to group nearby dynamic points together, then track each cluster over time with a multi-object tracker powered by per-cluster Kalman filters. Proper data association is paramount to ensure that measurements (i.e., cluster centroids) are assigned to the correct existing tracks.

5.3.1 DBSCAN Clustering of Dynamic Point Clouds

We use DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to cluster the post-filtered "dynamic" points. DBSCAN can automatically identify how many clusters (pedestrians or other moving objects) exist in a given scan based on density parameters.

The DBSCAN, clusters closely related data, allowing the identification of clusters of arbitrary shapes and sizes.

- **Core Points:** for each data point, the DBSCAN measures how many adjacent points fall within a specified radius (radius ϵ). If this measurement meets or exceeds a predetermined threshold (min_points), the point is designated as a core point.
- **Directly Reachable Points:** The points within this radius ϵ of a core point are considered to be directly accessible from that core point.
- **Cluster Formation:** The DBSCAN start with an unvisited point and check if there is a core point. If it is, it starts a new cluster and all points directly accessible from this core point are added to the cluster. The process continues recursively for each core point cluster.
- **Noise Identification:** Points that are neither centroids nor directly accessible from any core point are classified as noise.

The reasons we use DBSCAN are as follows:

- **Identifies clusters of arbitrary size:** The pedestrians may appear in different positions and orientations.
- **It handles noise reliably:** LiDAR data often contains noise from environmental factors so the algorithm distinguishes noise points and ensures that only significant groups are

detected.

- Unlike some algorithms, DBSCAN does not require a priori determination of components, which is advantageous when the number of pedestrians is unknown.

After clustering, each cluster is summarized by computing its centroid:

$$\mathbf{c} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{p}_i, \tag{5.6}$$

where p_i are the points in the cluster.

This centroid $c = (x_c, y_c)$ becomes the measurement for subsequent tracking.

5.3.2 Kalman Filter

The Kalman Filter offers a principled Bayesian framework for fusing noisy sensor data and modeling the evolution of a system's state over time [6]. By assuming (near) constant velocity or other linear motion constraints, it provides a computationally efficient way to predict an object's position and update estimates when new measurements arrive. This approach is especially useful in robotics, where sensors such as LiDAR frequently produce uncertain or partial observations of moving targets. It is widely used in tracking applications due to its simplicity and efficiency.

The Kalman filter operates in a prediction-update cycle to obtain estimates of unknown variables that are more accurate than those based on a single measurement. In our implementation, we use pedestrian tracking in a 2D plane using a constant velocity model (**Standard Kalman Filter**).

Mathematical Formulation:

- **State Model:** We represent each pedestrian's state $\vec{X}$ as a 4D vector,

$$\vec{X} = \begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^T, \tag{5.7}$$

  where (x, y) is position, and (v_x, v_y) is velocity. This concise model allows us to capture both location and motion.

- **Process Model:** It describes the way in which the situation evolves over time, usually using a linear system model.

- **Predict Step:** We use the process model to predict the next situation and uncertainty. It takes into account the dynamics of the system and the noise of the process.

  1. Predict the next stage:

$$\vec{X}_{pred} = \mathbf{F}\vec{X}_{prev} \tag{5.8}$$

  2. Predict the error covariance

$$\vec{P}_{pred} = \mathbf{F}\vec{P}_{prev}\mathbf{F}^T + \mathbf{Q} \tag{5.9}$$

  where:

  – **F** is a constant-velocity transition matrix. This matrix models the constant velocity assumption, where position updates depend on velocity. It represents the time interval between measurements δt.

  – **P** is the covariance matrix, which represents the uncertainty about the state variables. A higher value indicates more uncertainty.

  – **Q** is process noise covariance. Represents the uncertainty in the process model, for example unexpected accelerations.

- **Update Step:** We incorporate the new measurement to improve the assessment of the situation. The '**Kalman Gain**' determines how much the predictions are corrected based on the measurement. It balances the uncertainty between the prediction and the measurement.

  1. Compute Kalman Gain:

$$\mathbf{K} = \mathbf{P}_{\text{pred}}\mathbf{H}^T \left(\mathbf{H}\mathbf{P}_{\text{pred}}\mathbf{H}^T + \mathbf{R}\right)^{-1}, \tag{5.10}$$

  2. Update the estimate with measurement $\mathbf{z} = (\mathbf{z_x}, \mathbf{z_y})$:

$$\mathbf{y} = \mathbf{z} - \mathbf{H}\mathbf{x}_{\text{pred}} \tag{5.11}$$

$$\mathbf{x} = \mathbf{x}_{\text{pred}} + \mathbf{K}\mathbf{y} \tag{5.12}$$

3. Update the error covariance:

$$\mathbf{P} = \mathbf{P}_{\text{pred}} - \mathbf{KHP}_{\text{pred}}. \tag{5.13}$$

where:

- **H** is a measurement matrix. Since we only measure position (x, y), the measurement matrix extracts these components from the state vector.
- **R** is the measurement noise covariance. This represents the measurement noise, indicating our confidence in the sensor readings.

- **Recursive Operation:** We repeat the prediction and update cycle for each new measurement. The filter continuously improves its estimates over time.

Using Kalman filters enables robust real-time tracking of pedestrians by smoothing out measurement noise and predicting locations even when observations momentarily vanish (e.g., occlusions). The balance between prediction and correction ensures that our system remains stable in the presence of sensor inaccuracies, ultimately improving the reliability of pedestrian detection and avoidance. Also, it requires only the previous estimation and current measurement, making it suitable for real-time applications.

5.3.3 Multi-Object Tracking Architecture

Robots operating in dynamic environments often encounter multiple people moving simultaneously. A multi-object tracker allows the system to instantiate separate filters for each tracked entity and maintain their states over time. This architecture ensures we reliably handle multiple pedestrians interacting with the robot's path.

How We Use It

We maintain a dictionary "tracks" that maps each track_id to a dedicated Kalman filter. Each tracked pedestrian has its own filter maintaining a unique state vector.

Lifecycle Management:

- **Initialization:** When a new pedestrian is detected (via clustering a new group of LiDAR points), we spawn a new Kalman filter initialized with position but zero velocity.
- **Maintenance:** For each update, the system predicts every track's state, then attempts to associate newly computed cluster centroids to those existing tracks (see 5.3.4).

- **Aging and Confirmation:** Tracks that fail to get updated for several frames ("max_age") are deleted. Tracks that have been consistently observed ("min_hits") are labeled "confirmed" and used in downstream modules (like avoidance).

5.3.4 Data Association and Track Management

Data association aligns new measurements (cluster centroids) with existing tracks. In our code, we use a simple nearest-neighbor approach. This is critical in multi-target tracking to correctly match measurements to tracks.

We compute the Euclidean distance d_{ij} (5.14) between each track's predicted position (from Kalman Filter) and each new measurement (pointcloud message from the LiDAR). After that, we assign measurements to orbits based on the smallest distance, using a threshold to avoid false correlations.

$$d_{ij} = \|\mathbf{x}_{track_i} - \mathbf{z}_j\| \tag{5.14}$$

where $\mathbf{x}_{track_i}$ is the predicted track position (x, y) and $\mathbf{z}_j$ is the j-th centroid (measurement).

Accurate data association ensures that each pedestrian's observed motion is correctly fed into the same tracker instance, preserving track continuity over time. This is particularly critical if multiple pedestrians are located close together: robust association avoids mislabeling one pedestrian's movement as belonging to another. By properly managing and pruning tracks, we avoid clutter and keep computational overhead manageable.

5.4 Pedestrian Detection

In highly dynamic environments, robots must not only track pedestrians but also proactively avoid collisions. Our avoidance method detects pedestrians moving toward the robot and chooses a safe direction to move, computing and transforming an optimal way-point (the "best free point") to ensure the robot can navigate away from potential collisions.

5.4.1 Identify Aprroaching Pedestrian

A key first step in avoidance is recognizing when a pedestrian truly poses a threat—namely, when the pedestrian's movement vector indicates a direct approach. We employ geometric checks on the relative angles and distances between the pedestrian and the robot.

How We Use It

1. **Pedestrian Velocity:** We extract the pedestrian's velocity v_{ped} from the Kalman filter tracking.

2. **Pedestrian-to-Robot Vector:** We Compute the vector from the pedestrian to the robot

$$r = \mathbf{p}_{robot} - \mathbf{p}_{ped} \tag{5.15}$$

   where $\mathbf{p}_{robot}$ is the (x, y) position of the robot in the map and correspondingly $\mathbf{p}_{ped}$ is the position of the pedestrian.

3. **Angle Calculation:** Then we calculate the angle between the velocity vector and the pedestrian-to-robot vector.

$$\theta = \cos^{-1}\left(\frac{\mathbf{v}_{\text{ped}} \cdot \mathbf{r}}{\|\mathbf{v}_{\text{ped}}\|\|\mathbf{r}\|}\right). \tag{5.16}$$

   If θ is below a threshold (e.g, 30°), we assume that the pedestrian is heading towards the robot.

4. **Distance:** If distance from the pedestrian to the robot ($\|r\|$) is also below a certain limit (e.g., $2.5m$) and the pedestrian is heading towards the robot, we then calculate the best direction in which the robot will move (left or right) based on the available free space (see 5.4.2).

By focusing only on pedestrians whose paths are convergent with the robot, we minimize unnecessary evasion maneuvers. This reduces computation (since we only avoid imminent collisions) and fosters more human-like, efficient navigation.

5.4.2 Analyzing Robot's Surroundings

When we detect an approaching pedestrian, the next decision is how to shift the robot's path. We analyze the robot's LiDAR data in its local coordinate frame to see which side—left or right—offers a clearer path.

How We Use It

- **Forward Vector:** We take the pedestrian-to-robot direction $\mathbf{r}$ (5.15) as a reference "forward" vector in the robot's local frame.

- **Left/Right Vectors:** Rotate $\mathbf{r}$ by $\pm 90°$ to define the left and right directions.
- **Obstacle Counts:** For each incoming LiDAR point $\mathbf{p}$ (in $\pm 45°$ cones around the left or right vector), count how many points lie within a certain distance ($\leq 2.5m$)
- **Decision:** We choose the side with fewer LiDAR data returns, implying more free space.

This local side-check approach lets the robot quickly decide which lateral direction to shift toward, mimicking human "step aside" logic. Minimizing obstacle density reduces the chance of collision and simplifies later steps for finding precise free-space coordinates.

5.4.3 Computing the Best Free Point

Once we find the direction we want the robot to move (left-right), we locate a specific "best free point, which is the point where we will then send the robot towards it. In particular, we select the optimal point in the robot's local surroundings that maximizes distance from obstacles, given a preferred direction in which to move (e.g., left or right side). By systematically sampling possible positions around the robot, evaluating their viability, and selecting the safest option, this step is a critical component for collision-free maneuvering in cluttered or dynamic environments.

This point must be:

- This point must be within a specified distance from the robot.
- It must be within the selected directional sector.
- As far away as possible from obstacles.
- Not be adjacent to obstacles detected by the LIDAR sensor.

How We Use It

1. **Determines Movement Direction:** Based on the preferred direction specifying whether the robot should move to the left or right of an oncoming pedestrian, we orients ourselves relative to the pedestrian-robot vector $\mathbf{r}$ (5.15). This orientation ensures only candidate locations in the chosen lateral sector are considered. Specifically, we take the vector $(movement_direction)$that is perpendicular to the vector of the pedestrian-to-robot by adding or subtracting 90° ($\frac{\pi}{2} radians$).
    - If the preferred direction is left:

$$movement\_direction = \mathbf{r} + \frac{\pi}{2} \tag{5.17}$$

- If the preferred direction is right:

$$movement\_direction = \mathbf{r} - \frac{\pi}{2} \tag{5.18}$$

2. **Samples a Candidate Grid:** We define a rectangular area in robot's base frame, like this:

   - $\mathbf{x}_{min} = \mathbf{y}_{min} = -\mathbf{max}_{distance}$
   - $\mathbf{x}_{max} = \mathbf{y}_{max} = \mathbf{max}_{distance}$

   After that, we generate a grid of potential (x, y) positions around the robot, constrained by minimum and maximum distances. This step focuses on areas that are neither too close nor too far for safe and efficient avoidance.

3. **Filters Candidates by Angles:** For each point in the grid, we calculate its angular alignment relative to the intended *movement_direction* (5.17 or 5.18). Only those within a narrow angular corridor (e.g., $\pm 30°$) remain, ensuring the final chosen point supports a clear, direct maneuver.

4. **Checks Clearance Using Obstacle Data:** A proximity check with a nearest-neighbor search, assesses how close each candidate point is to known obstacles detected by the robot's LiDAR. Points that fail to meet a safe clearance threshold are removed.

5. **Selects the Safest Location:** Among the remaining points, we identify the position with the largest minimum distance to obstacles. This "best free point" helps ensure robust collision avoidance and provides a suitable waypoint to steer toward.

By focusing on a specific quadrant (left/right), we offer an immediate, lateral escape strategy when a pedestrian approaches. Also, its clearance-based selection maximizes the robot's separation from obstacles, enhancing reliability even when multiple obstacles or clutter are present.

5.5 Potential Field Navigation

Once we find the best point so that the robot can move towards it, we use Artficial Potential Fields (APF) to move the robot. In this approach, the robot is modeled as a point under the influence of virtual forces: an attractive force pulling it toward the goal and repulsive forces pushing it away from obstacles. This method is widely used to overcome unknown dynamic scenario, taking into account the realities of the current environment of the robot motion [5].

Consider that the Cartesian coordinates of the robot is $q = (x, y)^T$. So we can represent the APF function as:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{5.19}$$

where:

- $U(q)$: artificial potential field
- $U_{att}(q)$: attractive field
- $U_{rep}(q)$: repulsive field

The attractive force is the negative gradient of attractive field and the repulsive force is the negative gradient of the repulsive field. Thus, the artificial force of the robot as shown in .....

$$\begin{aligned} F(q) &= -\nabla U(q) \\ &= -\nabla U_{att}(q) - \nabla U_{rep}(q) \\ &= F_{att}(q) + F_{rep}(q) \end{aligned} \tag{5.20}$$

where:

- $F(q)$: artificial potential force
- $F_{att}(q)$: attractive force
- $F_{rep}(q)$: repulsive force

5.5.1 Attractive Force

The attractive force aims to pull the robot steadily toward a specified target location (the "goal") in the environment. By treating the goal as a potential well, the robot experiences a force that directs it to the final destination. In our case, we used for the attractive potential a parabolic quadratic function as shown at (5.21). With this selection, we manage to have higher values when the robot is further from the goal.

$$\begin{aligned} U_{att}(q) &= \frac{1}{2} k_{att}(q - q_d)^2 \\ &= \frac{1}{2} k_{att} \rho^2(q) \end{aligned} \tag{5.21}$$

where:

- k_{att} = positive constant controlling the magnitude of the pull.
- q = current position vector of the robot.
- q_d = current position vector of the target.
- $\rho_{goal}(q) = \|q - q_d\|$ is the Euclidean distance from the robot's position to the goal position.

The Attractive Force on robot is calculated as the negative gradient of attractive potential field as shown at (5.22):

$$\begin{aligned} F_{att}(q) &= -\nabla \\ &= -\frac{1}{2} k_{att} \rho^2(q) \\ &= -k_{att}(q - q_d) \end{aligned} \tag{5.22}$$

$F_a(q)$ is a direct vector toward q_d with magnitude linearly related to the distance from q to q_d.

5.5.2 Repulsive Force

To remain collision-free, the robot must be repelled from both static obstacles (e.g., walls) and dynamic entities (e.g., pedestrians) that come too close. On the contrary, if the robot is away from obstacles, it's motion must be taken into account as not affected by obstacles.

$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{d(q)} - \frac{1}{d_0} \right)^2, & d(q) \leq d_0 \\ 0, & d(q) \geq d_0 \end{cases} \tag{5.23}$$

where:

- k_{rep} = repulsive gain.
- $d(q)$ = distance from the robot to the obstacle.
- d_0 = distance of the obstacle repulsive force field.

When the robot is close to the goal, this function gives the smaller value. When the robot has reached the goal, the function will be 0.

Assume $d = \|q - q_{obs}\|$ as the distance between the robot and obstacles (static or dynamic), meanwhile d_0 is the largest impact distance of single obstacle. The repulsion field equation will be:

$$F_{\text{rep}}(q) = \begin{cases} k_{rep}\left(\frac{1}{d(q)} - \frac{1}{d_0}\right)\left(\frac{1}{d^2(q)}\right)\frac{\partial d(q)}{\partial x}, & d(q) \leq d_0 \\ 0, & d(q) \geq d_0 \end{cases} \tag{5.24}$$

$$F_{\text{rep}}(q) = \begin{cases} k_{rep}\left(\frac{1}{d(q)} - \frac{1}{d_0}\right)\left(\frac{1}{d^2(q)}\right)\frac{q - q_{obs}}{\|q - q_{obs}\|}, & d(q) \leq d_0 \\ 0, & d(q) \geq d_0 \end{cases} \tag{5.25}$$

There is no impact for the robot when the distance between the robot and obstacles is greater than d_0. We take into account both static and dynamic obstacles and

Specifically, each time we take the 5 (at most) closest static obstacles that we have calculated from (5.2.3). Then, from these, for each of these 5 obstacles we calculate the repulsive force (5.25). Furthermore, we take into account the repulsive force of the pedestrian and calculate it (5.25). Finally, we add all the repulsive forces of both the static obstacles and the pedestrian, and obtain the total repulsive force (5.26).

$$F_{rep}^{total} = [\sum_{i=1}^{5} F_{\text{rep}}^{i}(q)] + F_{\text{rep}}^{ped}(q) \tag{5.26}$$

5.5.3 Total Force

The Total Force is the sum of the attraction and repulsion of the robot (5.27).

$$F_{\text{total}} = F_{\text{att}} + F_{rep}^{total} \tag{5.27}$$

This force determines the robot's movement direction and speed as explained in the subsection (5.5.4).

5.5.4 Velocities Update

Once we have calculated the total force, we continue by updating the linear and angular velocity to move the robot to the target position [3].

First, we compute the desired velocity (5.28) and the speed (5.29):

$$V_{desired} = F_{total} \tag{5.28}$$

$$u = \|V_{desired}\| \tag{5.29}$$

After that, we compute the desired heading angle based on desired velocity and the heading error:

$$\theta_{desired} = \arctan2(V_{desired_y}, V_{desired_x}) \tag{5.30}$$

$$\theta_{error} = \theta_{desired} - \theta_{robot} \tag{5.31}$$

where θ_{robot} is the orientation of the robot.

Now, we compute the velocities and we limit them to a maximum value (linear and angular speed):

- **Linear Velocity:** It is determined by the magnitude of the total force u, with a heading-based factor adjusted by $\cos(\theta_{error})$ to reduce the forward speed when the directional error θ_{error} is large (the robot is not facing the direction of the goal). Also, we limit forward speed, if the heading error is large. Thus when:
  - $\theta_{error} = 0$: The robot has the desired direction to the goal position, $\cos(0) = 1 => u_{linear} = u$
  - $\theta_{error} = \pm\frac{\pi}{2}$: The robot is vertical to the desired direction of the goal position, $\cos(\pm\frac{\pi}{2}) = 0 => u_{linear} = 0$.

$$u_{linear} = \begin{cases} u\cos(\theta_{error}), & u_{linear} \leq u_{linear}^{max} \\ u_{linear}^{max}, & u_{linear} \geq u_{linear}^{max} \end{cases} \tag{5.32}$$

where u_{linear}^{max} is the maximum linear velocity.

- **Angular Velocity:** It is proportional to the difference between the robot's orientation and the desired orientation of the total force vector θ_{error}, causing the robot to rotate towards the desired direction.. The k_{att} acts as a gain to control the responsiveness of the robot's rotation. Also here, we limit the rotational speed with a maximum value.

$$u_{angular} = \begin{cases} k_{att}\theta_{error}, & u_{angular} \leq u_{angular}^{max} \\ u_{angular}^{max}, & u_{angular} \geq u_{angular}^{max} \end{cases} \tag{5.33}$$

where $u_{angular}^{max}$ is the maximum angular velocity.

Near obstacles, $\|F_{total}\|$ might be smaller (due to strong repulsion from obstacles), so the robot automatically slows down. In open space with no obstacles, the total force is mostly the attractive force, letting the robot speed up to u_{linear}^{max}.By combining these, the robot moves smoothly towards the goal while avoiding obstacles.

Chapter 6

Experimental Results

6.1 Simulation Results

This section presents the experimental results from the simulation tests conducted with the Jackal robot. The results are divided into two cases:

1. The robot navigating without any nearby obstacles.
2. The robot navigating while encountering an obstacle.

6.1.1 Case A: No Obstacles Near the Robot

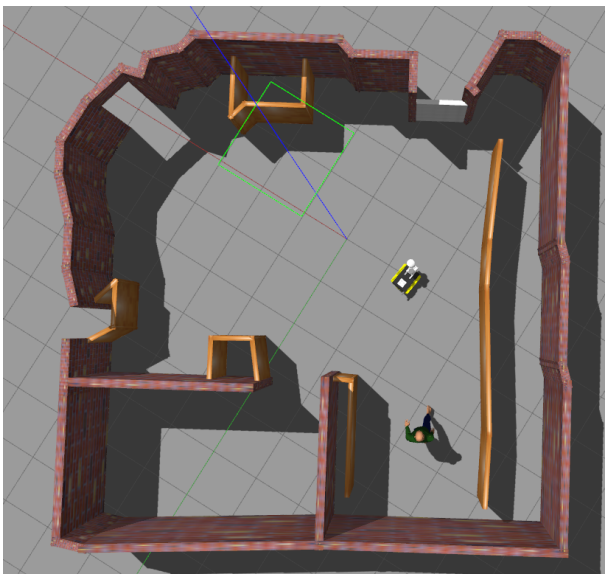


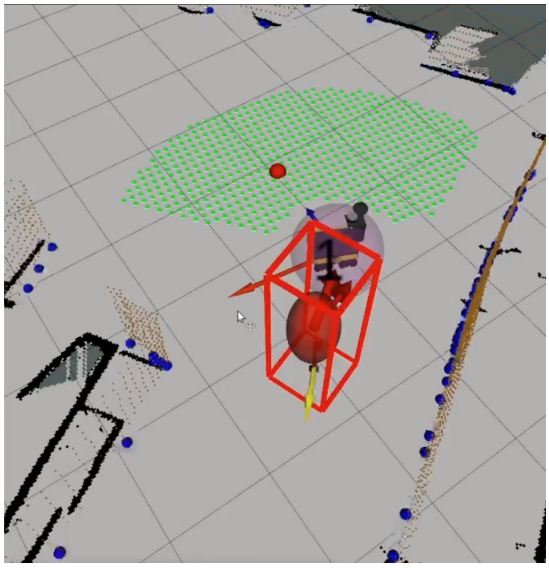
Figure 6.1: Robot position and orientation in the Simulation.

In Case A, the pedestrian is walking directly toward the robot, with the robot itself slightly angled relative to the pedestrian's path (see Fig. 6.1). Because there are no significant static

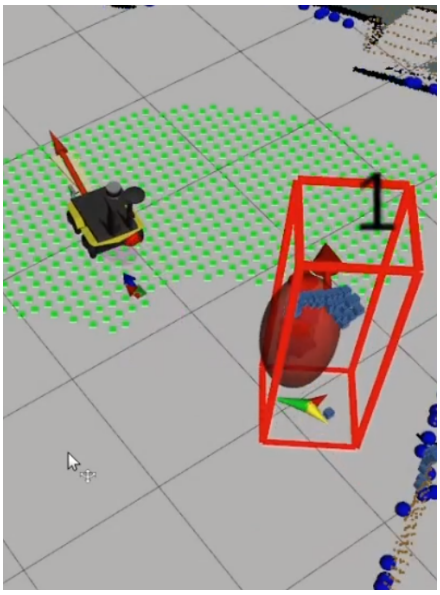
obstacles in the surrounding environment (apart from the lab walls), the robot can concentrate on avoiding the approaching pedestrian without additional constraints. This situation illustrates how the robot balances the pull of its goal location against the repulsive influence of the pedestrian, achieving a safe and efficient route in the absence of external interference.

(a) Snapshot (Gazebo) of the pedestrian heading towards the robot, before the robot detect him and start moving to a "safer" position

(b) Snapshot (Gazebo) seconds after the robot has reached the "safer" position and has completed avoid collision.



(c) Snapshot (RViz) of the pedestrian heading towards the robot, before the robot detect him and start moving to a "safer" position



(d) Snapshot (RViz) seconds after the robot has reached the "safer" position and has completed avoid collision.

Figure 6.2: Screenshots from the Gazebo and RViz for Case A, visualizing the starting and the final position of the robot, while also showing the pedestrian position.

The Fig. 6.2 displays the robot behavior in case of the pedestrian collision in Case A. Specifically,

- In the Fig. 6.2c the robot appears to be in an initial position, facing somewhat off to

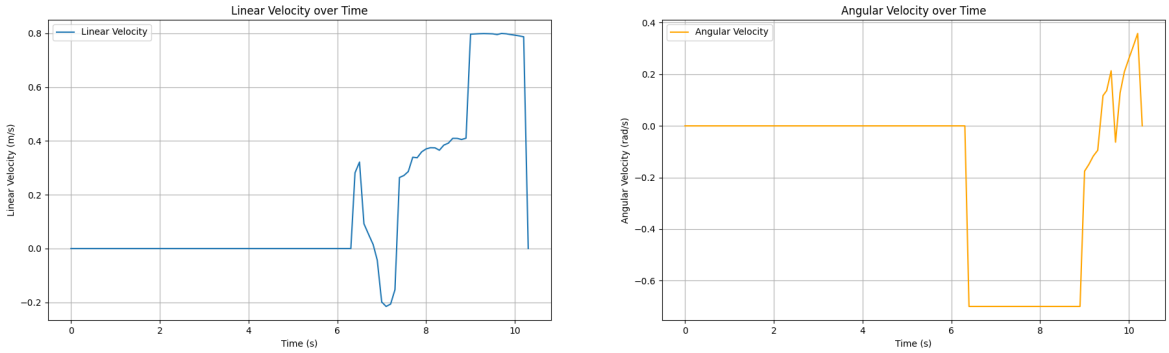
the side as the pedestrian approaches. We observe no other objects or obstructions close by—indicating that the robot's primary focus is the oncoming pedestrian.

- By contrast, in the Fig. 6.2d, the robot has already executed its avoidance maneuver and repositioned itself out of the pedestrian's direct path, confirming that it successfully moved laterally to avoid a collision.

The robot's decisions align closely with the steps in the side-selection process outlined in Section 5.4: it registers the oncoming pedestrian, assesses both lateral directions for open space, and calculates a specific "best free point." Notably, because there are no nearby obstacles beside the robot in this scenario, the algorithm defaults to moving the robot to the right.

The final position, depicted in Fig. 6.2d, confirms that the robot indeed follows the intended avoidance strategy. In essence, it "steps aside" in the chosen lateral direction and settles at the safest candidate location, following the principles of the potential-field and side-selection methods introduced in Sections 5.4 and 5.5.

Linear and Angular Velocity Over Time



(a) Linear velocity over time.

(b) Angular velocity over time.

Figure 6.3: Linear and angular velocity of the robot over time when no obstacle is near.

As shown in Fig. 6.3, the linear and angular velocities of the robot evolve dynamically as the potential field forces are resolved into velocity commands. Specifically:

- In the beginning, the robot remains stationary because the pedestrian is still far enough away that its influence on the robot is minimal. Around the 7-second mark, as the pedestrian gets closer, the robot's linear velocity begins to rise. This increase reflects the growing impact of the net force vector, which combines attractive and repulsive components.

  As explained in Section 5.5.4, we know that the linear velocity is then modulated by the factor $cos(\theta_{error})$ so that if the robot's heading is largely misaligned with $\theta_{desired}$, the

robot will slow down until it reorients. Thus, it is observed that at the beginning of the movement, the robot moves very slowly until it is oriented towards the target. Also, when the linear velocity is negative, means that the robot moves backwards. This behavior occurs when it's desired heading lies behind its current orientation. In mathematical terms the θ_{error} exceeds the $\pm\frac{\pi}{2}$ (i.e., the desired direction is more than 90 degrees away from the robot's current orientation), then the $cos(\theta_{error})$ becomes negative. Multiplying a positive force magnitude by a negative $cos(\theta_{error})$ yields a negative linear velocity, effectively commanding the robot to move "backwards".

Once the robot has adequately turned toward the goal, its linear velocity escalates, eventually reaching a peak of around $0.8m/s$ (see Fig. 6.3a). In the end, the velocity decreases and settles at zero, indicating that the robot has arrived at its final destination.

- The angular velocity Fig. 6.3b, illustrates how the robot adjusts its heading as it manages angle errors and navigational requirements. Initially, the angular velocity remains near zero, implying that that the robot is simply waiting in place. As soon as a significant heading correction is required due to pedestrian's distance, the angular velocity deviates sharply from zero. Negative values (around $-0.6m/s$) indicate the robot turning clockwise, while positive peaks indicate a counterclockwise turn.

  Because the robot intends to shift to the right, it first rotates in that direction, effectively "looking" along its desired heading. Once it has turned enough, the angular velocity diminishes, showing that the robot has aligned itself and can now proceed smoothly toward its target.

This dynamic adjustment of both linear and angular velocities illustrates the robot's adherence to the potential field-based navigation framework and its ability to adapt to the pedestrian's movement in real time.

Forces Over Time

The Fig. 6.4, demonstrates the dynamic interplay between the attractive, repulsive, and total forces acting on the robot.

Initially, all the forces have zero values due to the pedestrian's distance. However, as the pedestrian approaches the robot (around the 7-second mark), the repulsive and attractive forces increase significantly. Specifically:

- The attractive force (green color), the attractive component (green) starts modestly and

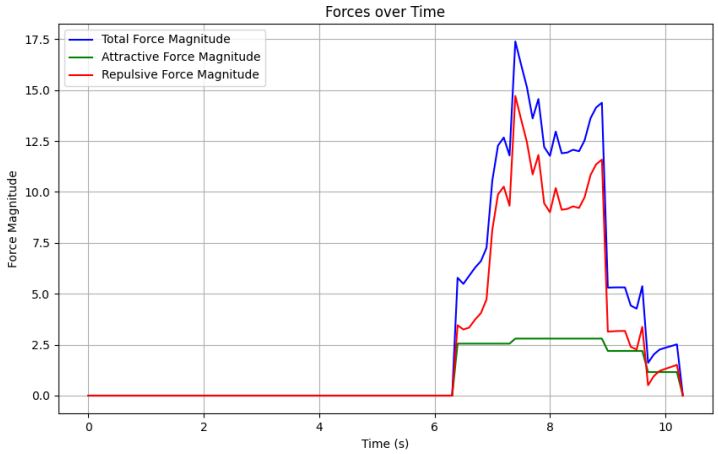


Figure 6.4: Total force, attractive force, and repulsive force over time (no obstacle).

remains relatively constant, reflecting the robot's ongoing pull toward the goal. Subsequently, as the robot progresses toward its target, the attractive force gradually diminishes, ultimately reaching zero when the robot arrives at the target location. This behavior is consistent with the theoretical framework described in Section 5.5.1, where the magnitude of the attractive force is directly proportional to the Euclidean distance between the robot and the target. This decrease in force reflects the natural reduction in distance as the robot approaches its destination.

- Similarly, the repulsive force (red color), reaches its peak magnitude at approximately 7 seconds, coinciding with the moment the pedestrian is detected. As the robot moves away from the pedestrian, the repulsive force gradually decreases. This behavior aligns with the theoretical principles outlined in Section 5.5.2, which state that the repulsive force increases significantly when the pedestrian is in close proximity to the robot and diminishes as the distance between them grows.
- The F_{total} force is plotted in blue color. It rises sharply when the robot must urgently counter the pedestrian's approach. As the robot successfully navigates away, repulsion starts to drop, and with fewer immediate collision threats, the net force also subsides.

Forces in the Map

The Fig. 6.5, provides a spatial snapshot of:

- The robot's path in blue.
- The pedestrian's path in red.

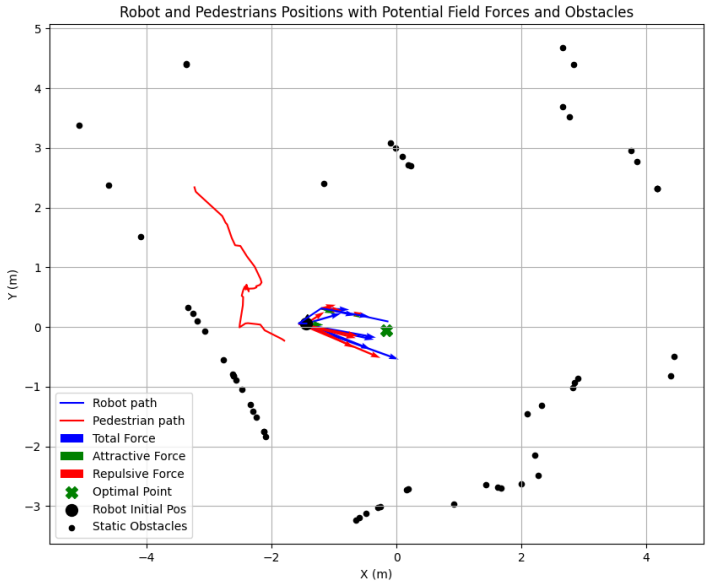


Figure 6.5: Visualization of forces in the map (x, y axes) when no obstacle is near.

- Static obstacles in black.
- The goal location marked with a green cross.
- The arrows over the blue trajectory illustrate the resultant vectors of attraction (green) and repulsion (red), with the larger blue arrow representing the net (total) force.

Based on these observations, we manage to see the interaction of attractive and repulsive force as described in Sections 5.5.1 and 5.5.2. The attractive force vectors consistently point toward the goal, reflecting the goal-seeking behavior of the robot. Conversely, the repulsive force vectors, originating from the pedestrian, extend outward and push the robot away. The resultant force is shown as a combination of these vectors. This adaptive behavior validates the effectiveness of the potential field algorithm in real-time environments.

6.1.2 Case B: Obstacle Near the Robot

In Case B, the pedestrian is heading directly toward the robot, and the robot is actively rotating, as shown in the Fig. 6.6, to evaluate its surroundings and adjust its trajectory. Unlike the previous scenario, there are static obstacles nearby, requiring the robot to consider both the repulsive force exerted by the approaching pedestrian and the obstacles in its vicinity. The robot must dynamically balance these forces using the potential field navigation algorithm to decide the optimal path, ensuring safe navigation while avoiding both the pedestrian and the surrounding obstacles. This scenario demonstrates the robot's ability to handle more complex

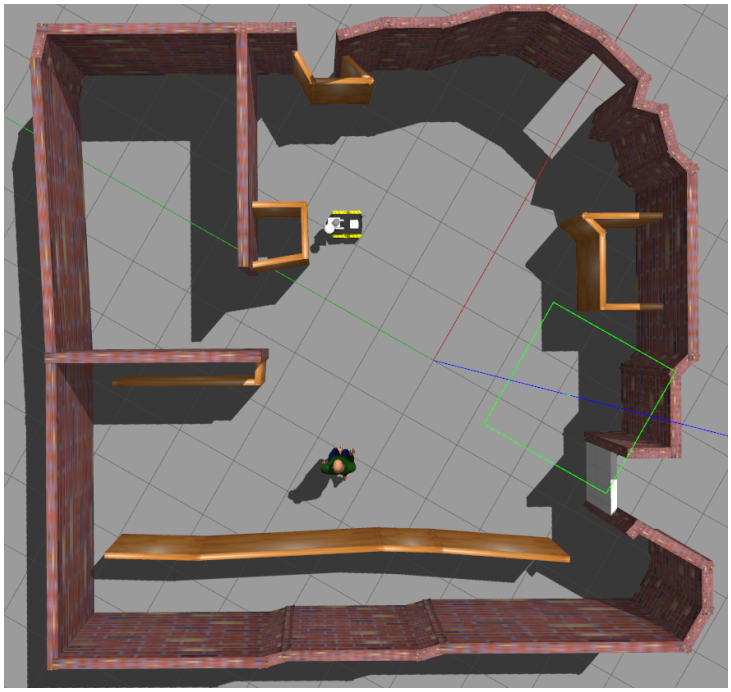


Figure 6.6: Robot position and orientation in the lab.

environments with multiple dynamic and static constraints.

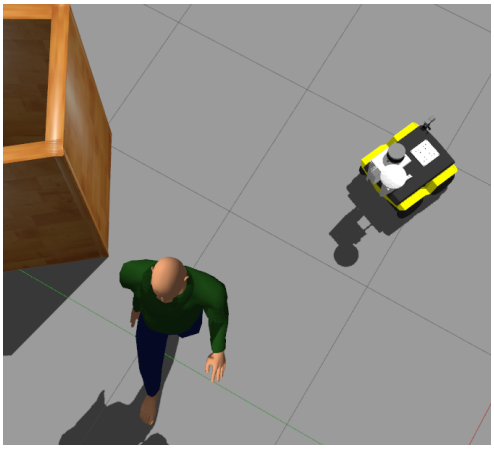
The Fig. 6.7 displays the robot behavior in case of the pedestrian collision in case B. Specifically:

- In the Fig. 6.7c the robot has just recognized an oncoming pedestrian (the red sphere at the bottom) and has populated the space in front of it with a grid of green candidate points. A single red sphere identifies the "best free point" chosen among these candidates. We also notice the cluster of LiDAR data (orange dots behind the robot), indicating the static obstacle that the robot should consider.
- In the Fig. 6.7d, the robot has moved to occupy the chosen red sphere's location, indicating that it has fully executed its sidestep maneuver to steer clear of both the pedestrian and the static obstacle.

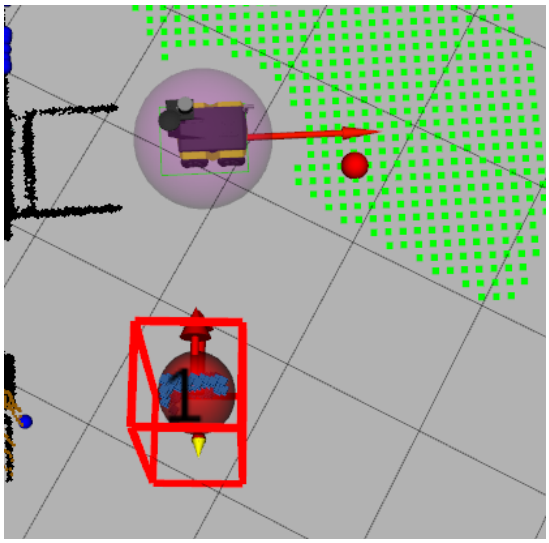
According to the algorithm, the "forward" reference vector $\mathbf{r}$ (5.15) is defined from the pedestrian to the robot. Because in this scenario the robot is partly rotated, it effectively sees the pedestrian's approach from an off-center orientation. This "forward" vector in the robot's local frame is used to define the pedestrian's incoming direction.

In our case, the algorithm has to choose between the side behind the robot which indicates the left side of the "forward" vector and the in front of the robot side which indicates the right side of the "forward". Here, it appears the chosen side is in front of (right side), because there is a static obstacle behind the robot (left side). This is because the LiDAR data has a higher density

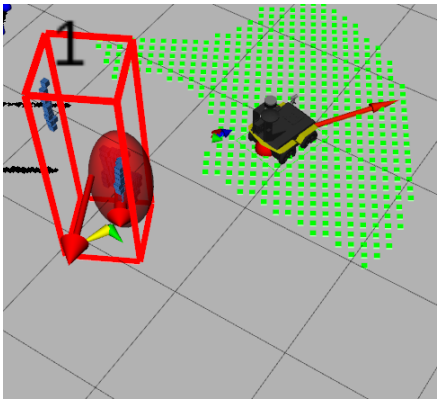
(a) Snapshot (Gazebo) of the pedestrian heading towards the robot, before the robot detect him and start moving to a "safer" position



(b) Snapshot (Gazebo) seconds after the robot has reached the "safer" position and has completed avoid collision.



(c) Snapshot (RViz) of the pedestrian heading towards the robot, before the robot detect him and start moving to a "safer" position



(d) Snapshot (RViz) seconds after the robot has reached the "safer" position and has completed avoid collision.

Figure 6.7: Screenshots from the Gazebo and RViz for Case A, visualizing the starting and the final position of the robot, while also showing the pedestrian position.

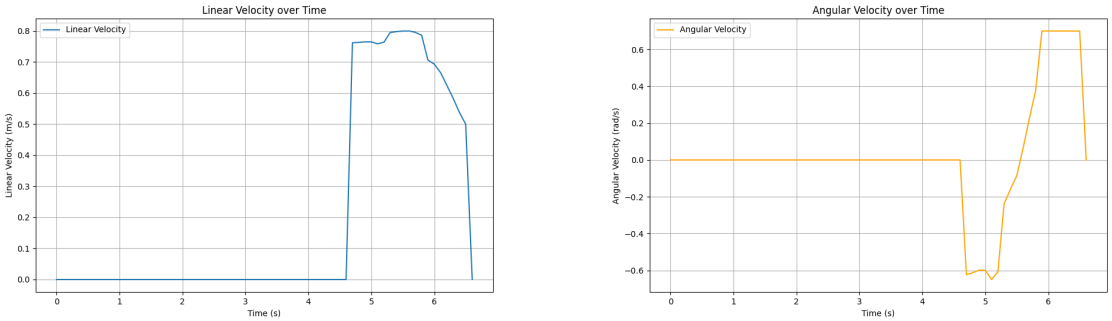
on the left side, so the algorithm chooses the right side to move the robot.

After that, we confirm that the real-world movement aligns with the theoretical framework that we describe in Section 5.5, as the robot has reached the optimal point.

Linear and Angular Velocity Over Time

As shown in Fig. 6.8, all the forces have zero values due to the pedestrian's distance. However, as the pedestrian approaches the robot (around the 7-second mark), the linear and angular velocities increase significantly. Specifically:

- The linear velocity reaches its highest value at approximately 7 seconds, aligning with the moment the robot first detects the pedestrian. In this scenario, the robot orientation and the desired heading to the goal has small difference ($\theta_{error} \approx 0$, so the $cos(\theta_{error}) \approx 1$), resulting in a very high linear velocity (about $0.8 m/s$ in the Fig. 6.8a) from the beginning of the robot's motion until the robot reaches the target.
- In a similar manner, the angular velocity also peaks at approximately 7 seconds, corresponding to the moment when the pedestrian comes within the robot's range of influence. In this case, the angular velocity consists of small variations in its values. This behavior is based on the fact that the robot does not need to rotate enough until it reaches it's desired heading to the goal (($\theta_{error} \approx 0$).



(a) Linear velocity over time (with obstacle). (b) Angular velocity over time (with obstacle).

Figure 6.8: Linear and angular velocity of the robot over time when an obstacle is nearby.

Forces Over Time

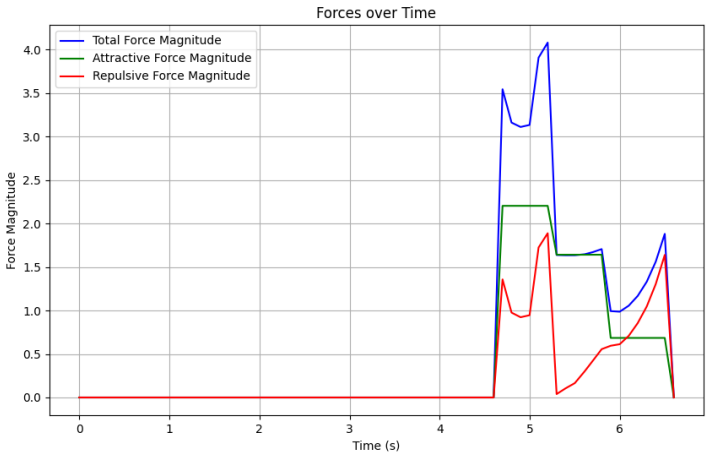


Figure 6.9: Total force, attractive force, and repulsive force over time (with obstacle).

The Fig. 6.9, demonstrates the dynamic interplay between the attractive, repulsive, and total forces acting on the robot.

Initially, all the forces have zero values due to the pedestrian's distance. However, as the pedestrian approaches the robot (around the 7-second mark), the repulsive and attractive forces increase significantly. Specifically:

- The attractive force (green arrow), reaches its maximum magnitude at approximately 7 seconds, coinciding with the moment the robot detects the pedestrian and is still positioned far from its target point. Subsequently, as the robot progresses toward its target, the attractive force gradually diminishes, ultimately reaching zero when the robot arrives at the target location. This behavior is consistent with the theoretical framework described in Section 5.5.1, where the magnitude of the attractive force is directly proportional to the Euclidean distance between the robot and the target. This decrease in force reflects the natural reduction in distance as the robot approaches its destination.
- Similarly, the repulsive force (red arrow), reaches its peak magnitude at approximately 7 seconds, coinciding with the moment the pedestrian is detected. As the robot moves away from the pedestrian, the repulsive force gradually decreases. This behavior aligns with the theoretical principles outlined in Section 5.5.2, which state that the repulsive force increases significantly when the pedestrian is in close proximity to the robot and diminishes as the distance between them grows.

Forces in the Map

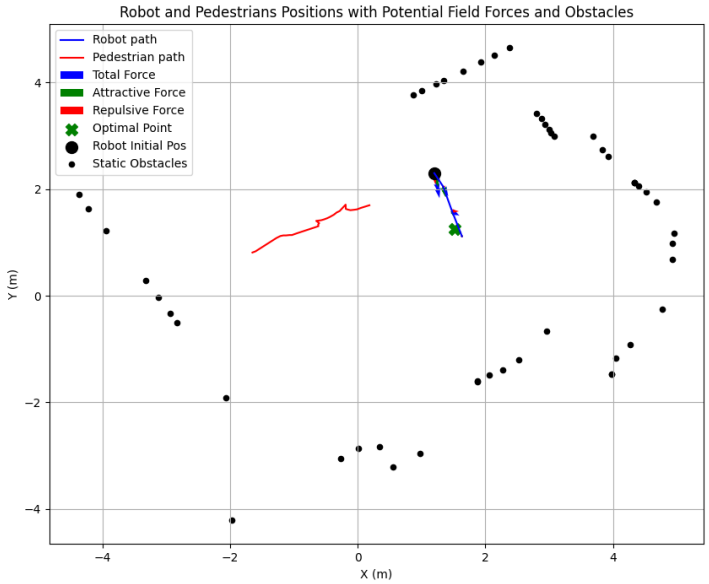


Figure 6.10: Visualization of forces in the map (x, y axes) when an obstacle is nearby.

The Fig. 6.10, shows the robot's path (blue line) alongside the pedestrian's path (red line), the static obstacles (black dots), and the goal position (green cross). It also visualizes the spatial distribution of the attractive (green arrow) and repulsive (red arrow) forces acting on the robot, showcasing their interaction as described in Sections 5.5.1 and 5.5.2. The attractive force vectors consistently point toward the goal, reflecting the goal-seeking behavior of the robot. Conversely, the repulsive force vectors, originating from the pedestrian, extend outward and push the robot away. The resultant force is shown as a combination of these vectors. This adaptive behavior validates the effectiveness of the potential field algorithm in real-time environments.

6.2 Lab Results

This section presents the experimental results from the lab tests conducted with the Jackal robot. The results are divided into two cases:

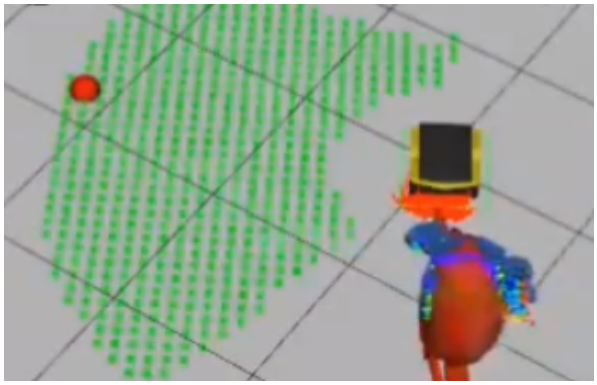
1. The robot navigating without any nearby obstacles.
2. The robot navigating while encountering an obstacle.

6.2.1 Case A: No Obstacles Near the Robot

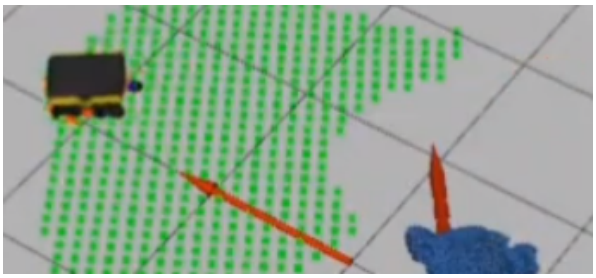
Figure 6.11: Robot position and orientation in the lab.

In case A, the pedestrian is heading directly towards the robot and the robot's orientation is aligned with the pedestrian's approach (as shown in Fig. 6.11). There are no static obstacles in the robot's vicinity (beyond the walls of the laboratory), allowing the robot to focus solely on avoiding the pedestrian. This scenario highlights the robot's ability to balance the attractive force driving it towards its target with the repulsive force generated by the pedestrian, ensuring safe and efficient navigation without external interference.

The Fig. 6.12 displays the robot behavior in case of the pedestrian collision in Case A. Specifi-



(a) Snapshot seconds before the robot starts moving to the goal position (red circle), after it detects the pedestrian, who is heading towards it.



(b) Snapshot seconds after the robot has reached the goal position.

Figure 6.12: Screenshots from the RViz for Case A, visualizing the starting and the final position of the robot.

cally,

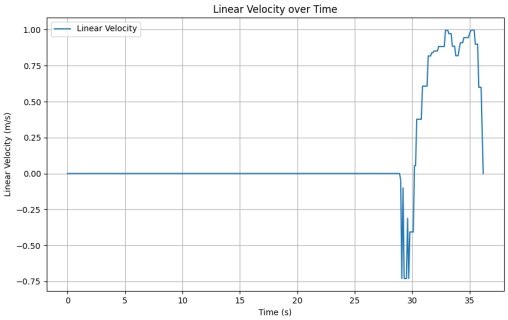
- In the Fig. 6.12a the robot has just detected the pedestrian (on the right in the image, the big red sphere) and has generated a dense array of green points around itself—these are candidate positions for the "best free point". The single red sphere (toward the upper-left) indicates the optimal point that the algorithm has chosen.
- By contrast, in the Fig. 6.12b, the robot has moved so that it is now very close to the red sphere (i.e., the "best free point"), indicating that the robot executed the avoidance maneuver and re-positioned itself away from the approaching pedestrian.

The robot's actions align well with the algorithmic steps described in the side-selection logic in Section 5.4: it detects the oncoming pedestrian, determines which side (left or right) is less cluttered, and computes a specific "best free point". It is also important to note that in this case, because we have no static obstacles next to the robot, the algorithm chooses by default the right side to move the robot.

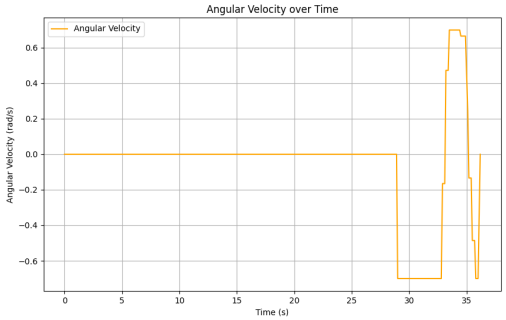
The final position of the robot in the Fig. 6.12b confirms that it follows the predicted avoidance maneuver. In other words, it is indeed "stepping aside" in the chosen lateral direction, reaching the safest candidate point consistent with the potential-field and side-selection approach outlined in Sections 5.4 and 5.5.

Linear and Angular Velocity Over Time

As shown in Fig. 6.13, the linear and angular velocities of the robot evolve dynamically as the potential field forces are resolved into velocity commands. Specifically:



(a) Linear velocity over time.



(b) Angular velocity over time.

Figure 6.13: Linear and angular velocity of the robot over time when no obstacle is near.

- Initially, the robot remains stationary, as the pedestrian has minimal influence due to the distance that has to robot. Around the 28-second mark, as the pedestrian approaches, the robot's linear velocity begins to increase steadily. This behavior is attributed to the magnitude of the resultant force vector, which combines the attractive and repulsive forces.

  According to Section 5.5.4, we know that the linear velocity is then modulated by the factor $cos(\theta_{error})$ so that if the robot's heading is largely misaligned with $\theta_{desired}$, the robot will slow down until it reorients. Thus, it is observed that at the beginning of the movement, the robot moves very slowly until it is oriented towards the target. Also, the linear velocity is negative, meaning that the robot moves backwards. This behavior occurs when it's desired heading lies behind its current orientation. In mathematical terms the θ_{error} exceeds the $\pm\frac{\pi}{2}$(i.e., the desired direction is more than 90 degrees away from the robot's current orientation), then the $cos(\theta_{error})$ becomes negative. Multiplying a positive force magnitude by a negative $cos(\theta_{error})$ yields a negative linear velocity, effectively commanding the robot to move "backwards".

  Once the robot turns sufficiently to the target position, the linear velocity increases, eventually approaching a peak (about $1.0m/s$ in the Fig. 6.13a). Finally, the velocity tapers off again and becomes zero, meaning that the robot has reached it's destination.

- The angular velocity Fig. 6.13b, shows how the robot reorients itself in response to heading errors and obstacle avoidance requirements. Initially, the angular velocity remains near zero, implying that that the robot is simply waiting in place. As soon as a significant heading correction is required due to pedestrian's distance, the angular velocity deviates sharply from zero. Negative values indicate the robot turning clockwise, whereas positive peaks denote turning in the opposite direction.

So, since the robot wants to move to the right, it is observed that at first it turns to the right, in order to "look along" the desired direction. After that, Once the robot has turned sufficiently he angular velocity settles back to lower levels, aligning it's direction so that it moves smoothly in the direction of the target.

This dynamic adjustment of both linear and angular velocities illustrates the robot's adherence to the potential field-based navigation framework and its ability to adapt to the pedestrian's movement in real time.

Forces Over Time

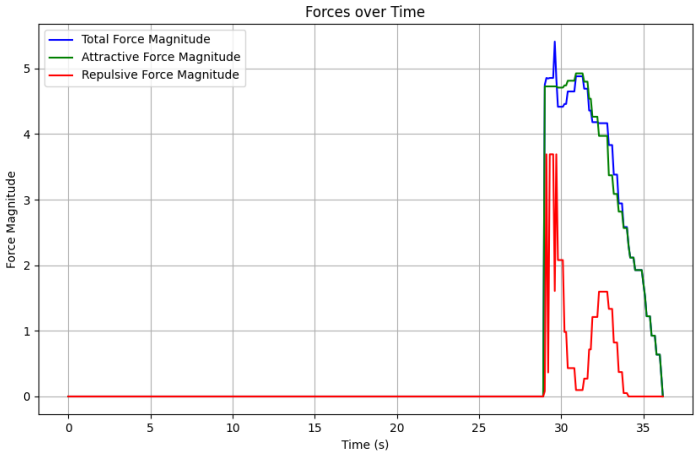


Figure 6.14: Total force, attractive force, and repulsive force over time (no obstacle).

The Fig. 6.14, demonstrates the dynamic interplay between the attractive, repulsive, and total forces acting on the robot.

Initially, all the forces have zero values due to the pedestrian's distance. However, as the pedestrian approaches the robot (around the 28-second mark), the repulsive and attractive forces increase significantly. Specifically:

- The attractive force (green color), reaches its maximum magnitude at approximately 28 seconds, coinciding with the moment the robot detects the pedestrian and is still positioned far from its target point. Subsequently, as the robot progresses toward its target, the attractive force gradually diminishes, ultimately reaching zero when the robot arrives at the target location. This behavior is consistent with the theoretical framework described in Section 5.5.1, where the magnitude of the attractive force is directly proportional to the Euclidean distance between the robot and the target. This decrease in force reflects the natural reduction in distance as the robot approaches its destination.

- Similarly, the repulsive force (red color), reaches its peak magnitude at approximately 28 seconds, coinciding with the moment the pedestrian is detected. As the robot moves away from the pedestrian, the repulsive force gradually decreases. This behavior aligns with the theoretical principles outlined in Section 5.5.2, which state that the repulsive force increases significantly when the pedestrian is in close proximity to the robot and diminishes as the distance between them grows.

- The F_{total} force is plotted in blue color. We see how it rises above both individual components around $28 - 32sec$, peaking when repulsion and attraction jointly influence the robot's motion. Afterward, as the robot maneuvers away from obstacles and gets closer to the goal, both forces diminish. Also, the repulsive term eventually returns to zero if the robot is no longer within an pedestrian's range.

Forces in the Map

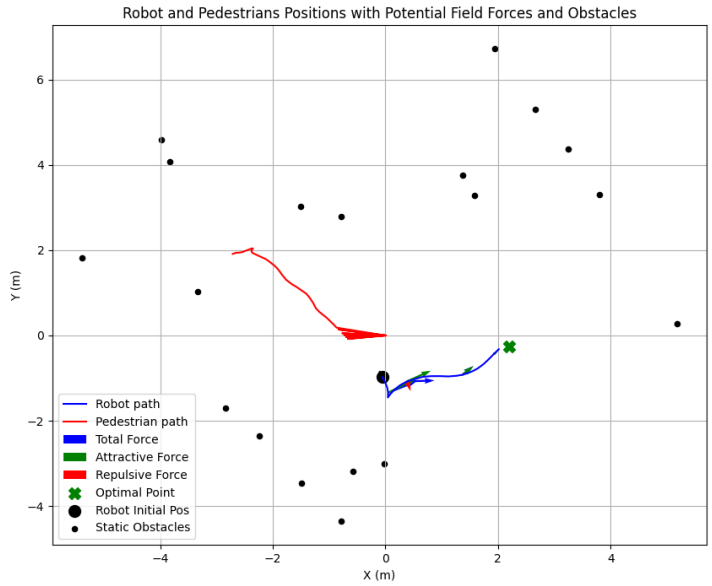


Figure 6.15: Visualization of forces in the map (x, y axes) when no obstacle is near.

The Fig. 6.15, provides a spatial snapshot of the robot's path (blue line) and the pedestrian's path (red line), along with the locations of static obstacles (black dots), the robot's start position (black circle), and the goal (green cross). The arrows over the blue trajectory illustrate the resultant vectors of attraction (green) and repulsion (red), with the larger blue arrow representing the net (total) force. Based with this Fig. 6.15, we manage to observe the interaction of attractive and repulsive force as described in Sections 5.5.1 and 5.5.2. The attractive force vectors consistently point toward the goal, reflecting the goal-seeking behavior of the robot. Conversely,

the repulsive force vectors, originating from the pedestrian, extend outward and push the robot away. The resultant force is shown as a combination of these vectors. This adaptive behavior validates the effectiveness of the potential field algorithm in real-time environments.

6.2.2 Case B: Obstacle Near the Robot

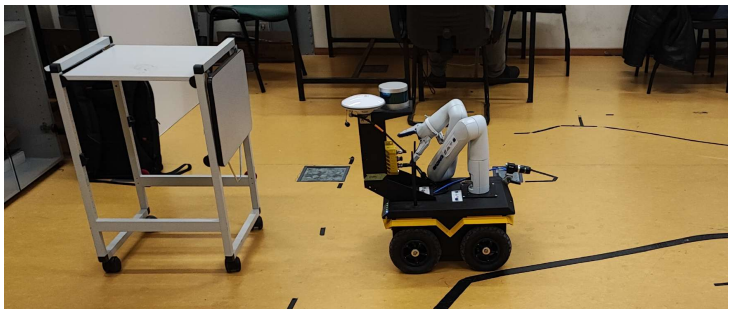
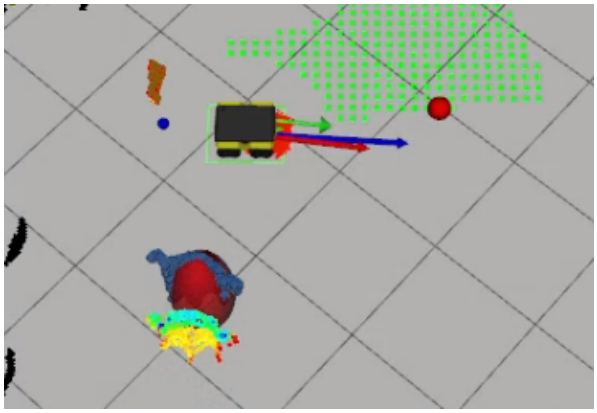


Figure 6.16: Robot position and orientation in the lab.

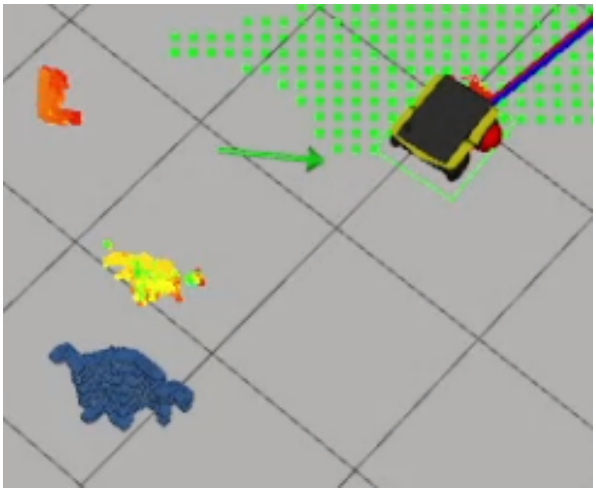
In Case B, the pedestrian is heading directly toward the robot, and the robot is actively rotating, as shown in the Fig. 6.16, to evaluate its surroundings and adjust its trajectory. Unlike the previous scenario, there are static obstacles nearby, requiring the robot to consider both the repulsive force exerted by the approaching pedestrian and the obstacles in its vicinity. The robot must dynamically balance these forces using the potential field navigation algorithm to decide the optimal path, ensuring safe navigation while avoiding both the pedestrian and the surrounding obstacles. This scenario demonstrates the robot's ability to handle more complex environments with multiple dynamic and static constraints.

The Fig. 6.17 displays the robot behavior in case of the pedestrian collision in case B. Specifically:

- In the Fig. 6.17a the robot has just recognized an oncoming pedestrian (the red sphere at the bottom) and has populated the space in front of it with a grid of green candidate points. A single red sphere identifies the "best free point" chosen among these candidates. We also notice the cluster of LiDAR data (orange dots behind the robot), indicating the static obstacle that the robot should consider.
- In the Fig. 6.17b, the robot has moved to occupy the chosen red sphere's location, indicating that it has fully executed its sidestep maneuver to steer clear of both the pedestrian and the static obstacle.



(a) Snapshot seconds before the robot starts moving to the goal position (red circle), after it detects the pedestrian, who is heading towards it.



(b) Snapshot seconds after the robot has reached the goal position.

Figure 6.17: Screenshots from the RViz for Case B, visualizing the starting and the final position of the robot.

According to the algorithm, the "forward" reference vector **r** (5.15) is defined from the pedestrian to the robot. Because in this scenario the robot is partly rotated, it effectively sees the pedestrian's approach from an off-center orientation. This "forward" vector in the robot's local frame is used to define the pedestrian's incoming direction.

In our case, the algorithm has to choose between the side behind the robot which indicates the left side of the "forward" vector and the in front of the robot side which indicates the right side of the "forward". Here, it appears the chosen side is in front of (right side), because there is a static obstacle behind the robot (left side). This is because the LiDAR data has a higher density on the left side, so the algorithm chooses the right side to move the robot.

After that, we confirm that the real-world movement aligns with the theoretical framework that we describe in Section 5.5, as the robot has reached the optimal point.

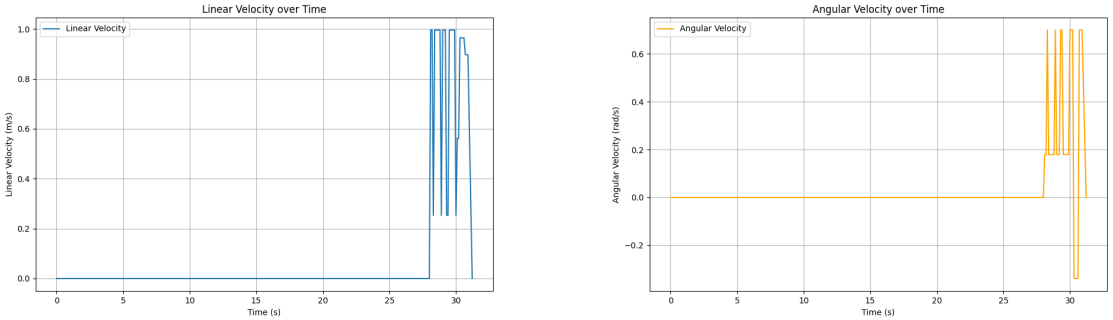
Linear and Angular Velocity Over Time

As shown in Fig. 6.18, all the forces have zero values due to the pedestrian's distance. However, as the pedestrian approaches the robot (around the 28-second mark), the linear and angular velocities increase significantly. Specifically:

- The linear velocity reaches its highest value at approximately 28 seconds, aligning with the moment the robot first detects the pedestrian. In this scenario, the robot orientation and the desired heading to the goal has small difference ($\theta_{error} \approx 0$, so the $cos(\theta_{error}) \approx 1$), resulting in a very high linear velocity (about $1.0 m/s$ in the Fig. 6.18a) from the beginning

of the robot's motion until the robot reaches the target.

- In a similar manner, the angular velocity also peaks at approximately 28 seconds, corresponding to the moment when the pedestrian comes within the robot's range of influence. In this case, the angular velocity consists of small variations in its values. This behavior is based on the fact that the robot does not need to rotate enough until it reaches it's desired heading to the goal ($(\theta_{error} \approx 0)$.



(a) Linear velocity over time (with obstacle). (b) Angular velocity over time (with obstacle).

Figure 6.18: Linear and angular velocity of the robot over time when an obstacle is nearby.

Forces Over Time

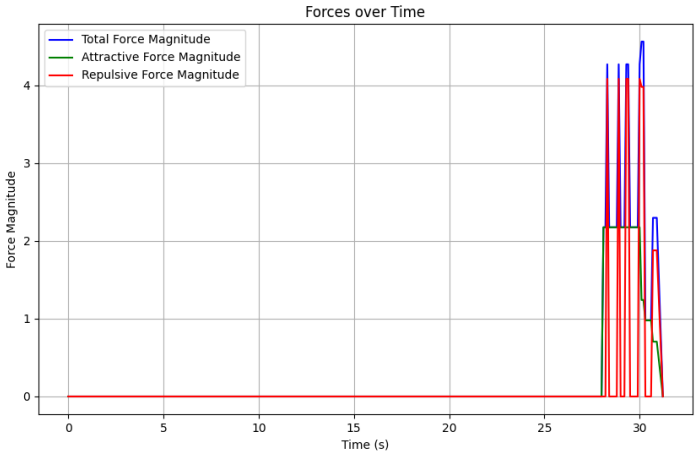


Figure 6.19: Total force, attractive force, and repulsive force over time (with obstacle).

The Fig. 6.19, demonstrates the dynamic interplay between the attractive, repulsive, and total forces acting on the robot.

Initially, all the forces have zero values due to the pedestrian's distance. However, as the pedestrian approaches the robot (around the 28-second mark), the repulsive and attractive forces increase significantly. Specifically:

- The attractive force (green arrow), reaches its maximum magnitude at approximately 28 seconds, coinciding with the moment the robot detects the pedestrian and is still positioned far from its target point. Subsequently, as the robot progresses toward its target, the attractive force gradually diminishes, ultimately reaching zero when the robot arrives at the target location. This behavior is consistent with the theoretical framework described in Section 5.5.1, where the magnitude of the attractive force is directly proportional to the Euclidean distance between the robot and the target. This decrease in force reflects the natural reduction in distance as the robot approaches its destination.
- Similarly, the repulsive force (red arrow), reaches its peak magnitude at approximately 28 seconds, coinciding with the moment the pedestrian is detected. As the robot moves away from the pedestrian, the repulsive force gradually decreases. This behavior aligns with the theoretical principles outlined in Section 5.5.2, which state that the repulsive force increases significantly when the pedestrian is in close proximity to the robot and diminishes as the distance between them grows.

Forces in the Map

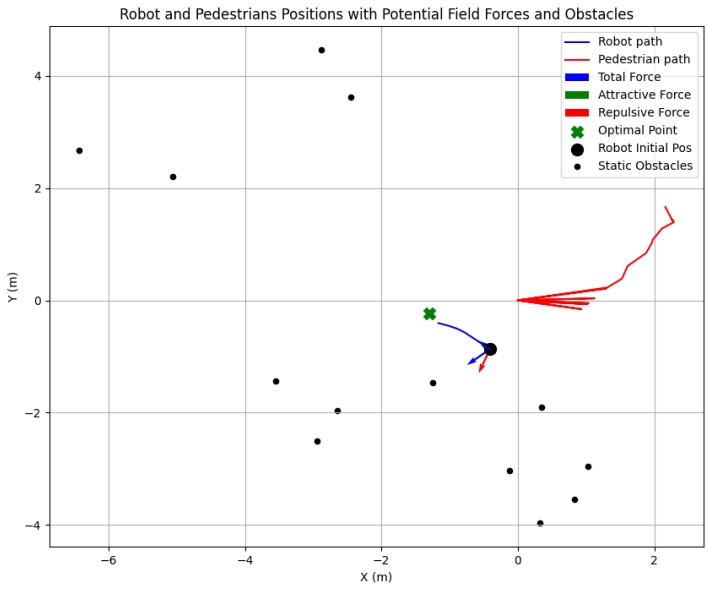


Figure 6.20: Visualization of forces in the map (x, y axes) when an obstacle is nearby.

The Fig. 6.20, it shows the robot's path (blue line) alongside the pedestrian's path (red line), the static obstacles (black dots), and the goal position (green cross). It also visualizes, the spatial distribution of the attractive (green arrow) and repulsive (red arrow) forces acting on the robot, showcasing their interaction as described in Sections 5.5.1 and 5.5.2. The attractive force vectors

consistently point toward the goal, reflecting the goal-seeking behavior of the robot. Conversely, the repulsive force vectors, originating from the pedestrian, extend outward and push the robot away. The resultant force is shown as a combination of these vectors. This adaptive behavior validates the effectiveness of the potential field algorithm in real-time environments.

Chapter 7

Discussion

7.1 Analysis of Strengths and Limitations

Overall, this thesis presents a comprehensive pedestrian avoidance framework that integrates LiDAR-based perception, pedestrian tracking, and potential field-based navigation.

Strengths: One of the primary advantages of the proposed approach is its **real-time responsiveness**, allowing the robot to dynamically adjust its trajectory based on pedestrian movements. The integration of multi-step filtering and tracking ensures continuous and adaptive navigation. Additionally, the system benefits from the **combination of multiple techniques**, including LiDAR-based perception, DBSCAN clustering, Kalman filtering, and potential field navigation, which collectively enhance pedestrian tracking and safe path computation.

Another significant strength is the system's ability to **handle dynamic environments** effectively. By distinguishing between static and dynamic obstacles, the robot can react appropriately to moving pedestrians while maintaining a well-defined navigation objective. Furthermore, the system's **scalability** within the ROS ecosystem allows for easy adaptation to different robotic platforms and further enhancements. Lastly, **experimental validation** in both simulated and real-world laboratory environments confirms the robustness of the approach in achieving pedestrian avoidance while maintaining smooth navigation.

Limitations: Despite these strengths, the system also exhibits certain limitations that affect its applicability in more complex environments. One major limitation is its **susceptibility to sensor noise**, as the accuracy of pedestrian tracking is dependent on LiDAR measurements, which can be impacted by sensor noise or occlusions. Errors in pedestrian localization may

subsequently affect the effectiveness of avoidance maneuvers.

Another limitation is the system's performance in **multi-pedestrian scenarios**. While the method performs well in single-pedestrian cases, its behavior in environments with multiple pedestrians moving unpredictably can be less stable. The potential field navigation approach also introduces the risk of **local minima**, where the robot may become trapped in a situation where opposing repulsive forces prevent further movement.

Additionally, **computational complexity** remains a concern for real-world deployment in crowded areas, particularly when processing high-density LiDAR data in real time.

7.2 Potential Improvements

To further enhance the performance and applicability of the proposed system, several improvements can be considered.

Enhanced Sensor Fusion: Integrating additional sensing modalities, such as cameras, could complement the LiDAR data and improve pedestrian tracking accuracy. This would mitigate issues related to occlusions and sensor noise.

Machine Learning-Based Prediction Models: Instead of relying solely on the Social Force Model, a data-driven approach using deep learning for pedestrian trajectory prediction could enhance the system's ability to anticipate future movements, leading to more proactive navigation decisions.

Multi-Pedestrian Interaction Handling: Extending the system to effectively manage multiple pedestrians simultaneously would improve its applicability in real-world scenarios. This could be achieved by incorporating reinforcement learning techniques or multi-agent modeling.

7.3 Conclusion

This work contributes a robust pedestrian-aware navigation framework for mobile robots, addressing key challenges in dynamic environments. While the proposed system has demonstrated its effectiveness in both simulation and laboratory settings, further advancements in sensing, learning-based prediction, and computational efficiency can significantly improve its real-world applicability. Future research directions should focus on refining multi-agent interactions and enhancing robustness in diverse environments.

References

[1] Dražen Brščić et al. “Using a rotating 3d lidar on a mobile robot for estimation of person’s body angle and gender”. In: *Sensors* 20.14 (2020), p. 3964.

[2] Dirk Helbing and Peter Molnar. “Self-organization phenomena in pedestrian crowds”. In: *arXiv preprint cond-mat/9806152* (1998).

[3] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[4] Mehdi Moussaıd et al. “Experimental study of the behavioural mechanisms underlying self-organization in human crowds”. In: *Proceedings of the Royal Society B: Biological Sciences* 276.1668 (2009), pp. 2755–2762.

[5] Rosli Omar, EN Sabudin, Che Ku Melor CK, et al. “Potential field methods and their inherent approaches for path planning”. In: *ARPN Journal of Engineering and Applied Sciences* 11.18 (2016), pp. 10801–10805.

[6] G Welch. “An Introduction to the Kalman Filter”. In: (1995).