POLITECNICO DI TORINO

Master's Degree in Mechanical Engineering

Master's Degree Thesis

Development and implementation of an obstacle avoidance algorithm for an Autonomous Mobile Robot

Supervisors

Prof. Marina INDRI

Advisor at CIM 4.0

Ing. Orlando TOVAR ORDOÑEZ

Candidate

Shixian WANG

July 2022

Abstract

The ongoing Industrial Revolution necessitates a high level of automation in order to be flexible and adaptive enough to comply with the demands of the market. Autonomous and collaborative robots will play an ever-greater role in this context. Therefore, the FIXIT project is proposed by CIM4.0 with the goal of providing interactive support for the human operator within an industrial or logistic environment to meet the Industry 4.0 requirements.

The objective of this thesis is to develop and implement path planning algorithm with obstacle avoidance technology to be executed on the AMR of FIXIT. Firstly, the state of the art of path planning methods is introduced to find the most suitable one for our project. Secondly, the URDF model is built, based on which the autonomous navigation is simulated by Gazebo and the results are visualized by RViz. Lastly, a real robot with a three-layer mechanical system: chassis, control, and application layers; and a two-layer control system: a higher-level distributed computer system and a lower-level motion control system, is designed to perform the path planning algorithms.

To realize the ability of avoiding moving obstacles, D* Lite with high adaptability to dynamic environments, is chosen as the GPP algorithm, while a DAPF is proposed as the LPP algorithm, which takes into consideration the moving obstacle detected by OpenCV tools such as background subtraction and blob detection, and tracked by the Kalman filter estimating its position and velocity.

The feasibility of the algorithm is verified by MATLAB and ROS simulation. The experimental performances of different path planning methods on real robot in the laboratory environment are compared to stress the improvement introduced by the algorithm developed in this thesis.

Keywords: path planning; obstacle avoidance; ROS; AMR; APF; OpenCV; moving obstacles.

Acknowledgements

Throughout the process of writing this thesis, I have received a lot of help and support. I would first like to thank my supervisor, Prof. Marina Indri, whose expertise was crucial in developing the study questions and methodology. Your insightful feedback encouraged me to improve my thoughts and raised the caliber of my work. I would like to thank David and Fiorella for their kind support and helpful suggestions.

I would also like to thank all the CIMers who gave me the opportunity to do the thesis in CIM 4.0, especially my tutor, TOVAR ordoñez, for his invaluable guidance during my work. You provided me with the resources I required to choose the best direction and effectively finish my thesis.

I would particularly like to acknowledge my teammates, for their wonderful collaboration and patient support. We study together; we go to launch/lunch together; we fight together.

Additionally, I would like to thank my parents for their unconditional giving and everlasting support. You are always there for me.

Last but not least, I would like to thank my friends, who offered intriguing conversations and enjoyable diversions from my studies.

Grazie,
WANG SHIXIAN

Table of Contents

List of Tables

List of Figures

Acronyms

ACO
Ant Colony Optimization algorithm

AGV
Autonomous Guided Vehicles

AI
Artificial Intelligence

AIoT
Artificial intelligence of things

AMCL
Adaptive Monte Carlo Localization

AMR
Autonomous Mobile Robots

APF
Artificial Potential Field

BS
Background Subtraction

CAD
Computer Aided Design

DAPF
Dynamic Artificial Potential Field

DOF

Degree Of Freedom

DWA

Dynamic Window Approach

EB

Elastic Band

EKF

Extended Kalman filter

FBD

Free Body Diagram

GIS

Geographic Information System

GNRON

Goal Non-Reachable with Obstacles Nearby

GPP

Global Path Planning

GPS

Global Positioning System

IMU

Inertial Measurement Unit

IP

Internet Protocol

LPA*

Lifelong Planning A*

LPP

Local Path Planning

LQE

Linear Quadratic Estimation

MCL

Monte Carlo Localization

MLE

Maximum Likelihood Estimate

MP

Mobile Platform

ODE

Ordinary Differential Equation

OGRE

Object-Oriented Graphics Rendering Engine

OMR

Omnidirectional Mobile Robot

PID

Proportional Integral Derivative

PLC

Programmable Logic Controller

PRM

Probabilistic RoadMap

ROS

Robot Operating System

RRT

Rapidly-exploring Random Tree

SLAM

Simultaneous Localization and Mapping

TCP/IP

Transmission Control Protocol/Internet Protocol

TEB

Timed Elastic Band

TOF

Time Of Flying

TSP

Travelling Salesman Problem

UART

Universal Asynchronous Receiver/Transmitter

UAV

Unmanned Aerial Vehicle

UDP

User Datagram Protocol

URDF

Unified Robot Description Format

XML

Extensible Markup Language

Chapter 1

Introduction

Industry 4.0 represents a new paradigm for intelligent and autonomous manufacturing. It more profoundly integrates manufacturing operations systems with communication, information and intelligence technologies [1]. The ongoing Industrial Revolution necessitates a high level of automation in order to be flexible and adaptive enough to comply with the demands of market, faster and lower-cost. Autonomous and collaborative robots will play an ever-greater role in this context. Therefore, the FIXIT project is proposed by CIM4.0, with the goal of providing interactive support for the human operator within an industrial or logistic environment to meet the Industry 4.0 requirements.

Figure 1.1: FIXIT

The FIXIT team shown in Fig. 1.1 is composed of two parts: the Autonomous Mobile Robot (AMR) and the Unmanned Aerial Vehicle (UAV). The AMR is based on the Scout mini platform seen in Fig. 1.2, which is an Omnidirectional Mobile Robot (OMR), and equipped with various sensors to percept surroundings, for example, Lidar and camera. Besides, the chassis of UAV is produced by 3D print with optimal structures, installed with sensors and actuators to implement unmanned

flight. Above all, these two components will collaborate by communication, taking full advantages from each other and expanding the working area.

Figure 1.2: Scout mini

The objective of this thesis is to develop and implement an obstacle avoidance system to be executed on the AMR of FIXIT.

AMRs are a particular class of robot that can independently comprehend and navigate its surroundings. AMRs are distinct from their predecessors, Autonomous Guided Vehicles (AGVs), which depend on tracks or predetermined trajectories and frequently need operator supervision. Unrestrained by wired power, AMRs perceive and navigate through their environment using a complex set of sensors, artificial intelligence, machine learning, and computation for path planning. AMRs have sensors, so if they come into an unexpected obstacle while navigating their surroundings, such a fallen box or a crowd of people, they will employ a navigation strategy like collision avoidance to slow down, stop, or redirect their path around the object and then carry on with their work.

Path planning and obstacle avoidance technology is an important research field of AMR. So far, several algorithms are developed to implement autonomous navigation. However, most of them are focued on avoiding static obstacles. The moving obstacles such as human, vehicles and working machines, do exist at the working environment of an AMR. Therefore, the ability of avoiding moving obstacles during navigation is necessary for application of AMR in industry.

Considering the indeed demands, and exploiting the Robot Operating System (ROS), a navigation system with the path planning and obstacle avoidance technology is developed in this thesis, especially with the capability of avoiding moving obstacles.

1.1 Structure of thesis

Firstly, taking advantage of ROS, an AMR is designed and built with simplified processes. secondly, existing global path planning and local path planning algorithms

are improved. The Artificial Potential Field (APF) method is used as robot local path planning algorithm. A Dynamic Artificial Potential Field (DAPF) method with multi-target points is proposed, which enables the robot to walk along the global path, meanwhile, having a good obstacle avoidance ability against dynamic obstacles. Finally, path planning experiments in different situations were carried out with the AMR.

The main contents of this thesis are as follows:

Chapter 1 introduces the FIXIT project.

Chapter 2 introduces the state of art of path planning algorithms, including global path planning and local path planning algorithm.

Chapter 3 describes the architecture of a AMR based on ROS. Firstly, Robot Operating System (ROS) is introduced. Secondly, the kinematic model of OMR with Mecanum wheels is analysed. Thirdly, the URDF model of the robot is built and simulated in gazebo. Finally, the real robot is constructed. A two level control system including motion control system and distributed computer system is built, as well as the mechanical structure and perception system.

Chapter 4: D* Lite (GPP) is introduced as a Global Path Planning. A Dynamic Artificial Potential Field (DAPF) method is proposed to make up the drawbacks of traditional APF. More specifically, a strategy of following global path by setting local goal point in real time is proposed. Besides, the information of relative velocity between robot and dynamic obstacles is added to the potential field, resulting a better performance of collision avoidance when encountering with moving obstacles.

Chapter 5: the path planning algorithm is implemented in ROS simulation environment and on real robot. A strategy of estimating position and velocity of moving obstacle is applied. And the performance is compared to path planning algorithm provided by ROS.

Chapter 2

State of the art

2.1 Path Planning

Over the past decades, path planning techniques have had a wide range of applications in many fields, such as hierarchical routes for networks in wireless mobile communication [2], radar search avoidance of cruise missiles, Global Positioning System (GPS) navigation, road planning based on Geographic Information System (GIS), and resource management and configuration issues. Basically, the planning problem that can be topologically described as point-line networks can be solved by the method of path planning. Path planning is also a fundamental issue in mobile robotics, which aims to find an optimal or sub-optimal obstacle-free path from a starting location to a destination while optimizing some performance criteria [3], such as distance, time, or energy, with distance being the most commonly adopted criterion [4].

Generally, with respect to whether full knowledge of the environment is available or not [5], path planning can be categorized into Global Path Planning (GPP) and Local Path Planning (LPP). GPP, when used offline, chooses a path devoid of obstacles while being fully aware of its surroundings. LPP, also known as online path planning, creates an effective path utilizing little or no prestored environmental information [6]. Referring to the information from sensors, LPP considers the environment more complexly with dynamic changes compared to GPP provided with a static map. In terms of purpose, the goal of global path planning is to generate a path that meets certain optimization indicators, while local path planning tends to focus on the practicability and obstacle avoidance performance of the path. The differences between GPP and LPP are listed in table 2.1. Therefore, in practical applications, global and local path planning are often combined to achieve complementary advantages.

GPP	LPP
Map based	Sensor based
Deliberative system	Reactive system
Relative slower response	Fast response
Assume complete knowledge of the workspace area	Assume incomplete knowledge of the workspace area
Obtain a feasible path leading to goal	Follow path to while avoiding obstacles or objects while moving towards target

Table 2.1: Differences between GPP and LPP [7]

2.2 Global Path Planning

By combining accumulated sensor data and a priori knowledge, GPP is a method of enabling an AMR to choose the optimal path to a goal position [8]. It is a slow and deliberative process that finds the most efficient path to a long-term goal, concerning with long-range planning instead of vehicle stability or small-scale obstacles, which are left to the LPP. The planning method entails two basic steps: gathering the relevant data into an acceptable and effective configuration space; and then employing a search algorithm to choose the optimum path within that space based on the user's pre-defined criteria.

Traditional methods and intelligent methods are the two primary divisions of the path planning techniques now in use. Prominent traditional planning methods include the search-based algorithms and sampling-based algorithms. Search-based methods contain global visibility graph algorithm [9], for example, Dijkstra algorithm [10], A* algorithm and a series of variations, for instance, D* [11], D* lite [12] and Lifelong Planning A* (LPA*) [13]. Moreover, sampling-based algorithms include Probabilistic RoadMap (PRM) [14], Rapidly exploring Random Tree (RRT), RRT* and informed RRT*, etc. Besides, intelligent planning methods include fuzzy logic [15], neural networks [16] [17], Ant Colony Optimization algorithms (ACO), genetic algorithms [18] and particle swarm optimization algorithms [19]. Correspondingly, three representative algorithms are introduced in the following sections.

2.2.1 A* algorithm

A* is a best-first search algorithm that is informed, meaning it is programmed in terms of weighted graphs. It starts at a particular starting node in the graph and intends to find the shortest path to the specified goal node with the smallest cost

(least distance travelled, shortest time, etc.). It accomplishes this by keeping track of a tree of paths leading from the start node and extending each of those paths by one edge until its termination requirement is met.

A* decides which of its paths to extend for each iteration of its main loop, referring to the path's cost and an estimate of the cost needed to succeed the path all the way to the target. A* specifically chooses the path with the lowest:

$$f(n) = g(n) + h(n) \tag{2.1}$$

where n is the next node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic function [20] (Manhattan, Euclidean or Chebyshev distance) that estimates the cost of the cheapest path from n to the goal.

As an example to illustrate heuristic distance, since a straight line is physically the shortest distance that may exist between any two places, while looking for the shortest path on a map, $h(n)$ might stand in for that distance. Depending on the range of motions available (4-way or 8-way), employing the Manhattan distance or the octile distance becomes preferable for a grid map from a video game. This algorithm's advantage is that the distances used as a criterion can be changed, added, or substituted. This provides a large range of variations on this fundamental idea. For example, time, energy consumption, or safety can also be included in the function $f(n)$ [21]. The heuristic h is referred to as monotone or consistent if it meets the extra requirement $h(x) \leq d(x, y) + h(y)$ for every edge (x, y) of the graph, where d specifies that edge's length. With a consistent heuristic, A* is guaranteed to discover an optimal path without processing any node more than once. A* is comparable to performing Dijkstra's algorithm with the decreased cost $d'(x, y) = d(x, y) + h(y) - h(x)$.

Typical implementations of A* use a priority queue called "open set" or "fringe" to perform the repeated selection of minimum (estimated) cost nodes to expand. The node with the lowest $f(n)$ value is removed from the queue at each stage of the process, and its neighbors' f and g values are modified accordingly before they are added to the queue. The process keeps running until the removed node is the goal node, and as a result, the node with the lowest f value among all fringe nodes becomes the goal node. Since h at the target is zero in an admissible heuristic, the f value of that goal also equals the cost of the shortest path.

An illustration of A* search procedure on a rectangular grid with three obstacles is shown in Figure 2.1. The destination is labeled by 33 on the right and the start node is tagged with 0 on the left. White circles are generated nodes in OPEN, while black dots are expanded nodes in CLOSED (all of their successors have been generated before the target has been found). The Manhattan distance from the start node is used to calculate the g values of the nodes, which are represented by their labels. An ideal route from the start node to the goal would cost 33.

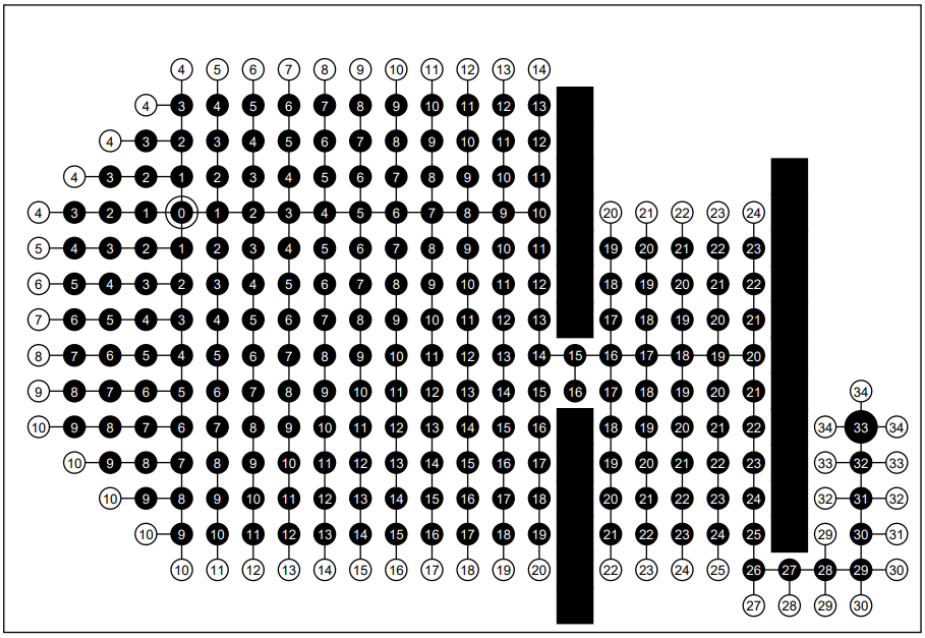


Figure 2.1: A 2D grid map with three obstacles: A* [22]

The nodes are connected via a traversal tree. The Manhattan distance to the goal, measured on a complete grid with no obstacles, serves as the heuristic h that directs the search process of A. Fig. 2.1 illustrates how the heuristic, as opposed to the breadth-first search approach, prunes the search: The majority of A* path candidates depart from the start node to the right. Path candidates in the breadth-first approach would radiate outward in all directions, resembling a "Manhattan ball" around the start node. Notably, nodes in Fig. 2.1 create substantial heaps in front of the black obstructions.

A* comes to an end if there are no paths that can be extended or if the path it chooses to extend leads from start to goal. A* is guaranteed to deliver a least-cost path from start to goal if the heuristic function is admissible, which means that it never overestimates the real cost of getting there.

Since all nodes in closed set remember their predecessors, the path made up by nodes with correct sequence can be searched reversely starting from the goal point, and keep tracking the predecessor of the node until the start node is some node's predecessor.

The following presents the pseudocode for A* [23]:

Algorithm 1 ALGORITHM A*

(1) Put the start node s into OPEN.

(2) IF OPEN is empty THEN exit with failure.

(3) Remove from OPEN and place in CLOSED a node n for which f is minimum. (Resolve ties for minimal f value, but always in favor of any goal node.)

(4) IF n is a goal mode THEN exist successfully with the solution obtained by tracing back the pointers back to n.

(5) ELSE expand n, generating all its successors, and attach to them pointers back to n. FOR every successor n' of n DO

(5.1) IF n' is not already in OPEN or CLOSED THEN estimate $h(n')$, and calculate $f(n') = g(n') + h(n')$, where $g(n') = g(n) + c(n, n')$ and $g(s) = 0$, and put n' into OPEN.

(5.2) IF n' is already in OPEN or CLOSED THEN direct its pointer along the path yielding the lowest $g(n')$.

(5.3) IF n' required pointer adjustment and was in CLOSED THEN reopen it. END FOR .

(6) GO TO step 2.

2.2.2 Rapidly-exploring Random Tree algorithm

The motion planning issue is PSPACE-hard [24] from the perspective of computational complexity, indicating that any complete algorithm, i.e., one that yields a solution if one exists and returns failure otherwise, is destined to be computationally intractable [25].

Practical motion planning approaches typically loosen the completion constraints to gain computational efficiency. A recent branch of this study is sampling-based methods, which includes algorithms like the PRM and the RRT. The majority of sampling-based algorithms are probabilistically complete, which means that when the number of samples approaches infinity, the chance that the algorithm finds a solution, if one exists, converges to one.

The benefit of sampling-based algorithms is that, even in high-dimensional state spaces, they can find a feasible motion plan very rapidly (if such a plan

exists). Additionally, systems with differential constraints can be handled well by the RRT in particular. Due to these features, the RRT is a useful method for motion planning on cutting-edge robotic platforms [26].

An RRT grows a tree rooted at the starting configuration by using random samples from the search space [27]. A connection is made between each sample and the closest state in the tree as it is being drawn. The new state is added to the tree if the link is viable (passes totally through free space and adheres to any constraints). Once the new state is the goal point or within the tolerance of reaching the goal, the path is found, as figure 2.2 shows. The likelihood of expanding an existing state is proportional to the size of its Voronoi area when the search space is sampled uniformly. The tree expands preferentially toward vast unexplored areas because the greatest Voronoi regions are found in the states that are at the frontier of the search.

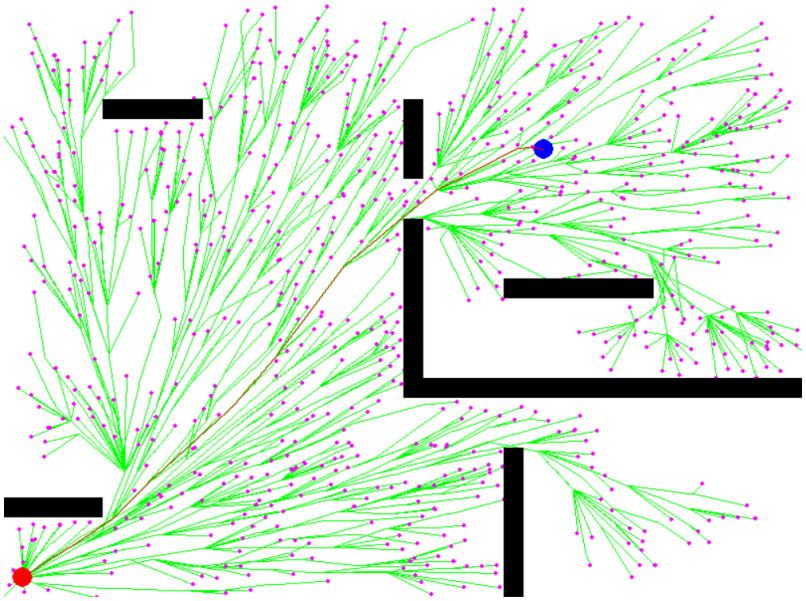


Figure 2.2: Rapidly-exploring Random Tree algorithm [28]

A growth factor frequently sets a restriction on the length of the connection between the tree and a new state. A new state with a maximum distance along the line from the tree to the random sample is employed in place of the random sample itself if the random sample is farther from its nearest state in the tree than this limit permits. The growth factor then controls the rate of the tree's growth, while the random samples govern its direction. This limits the size of the incremental expansion while maintaining the RRT's space-filling bias.

By raising the probability of sampling states from a particular region, RRT

growth can be biased. This is used by the majority of practical RRTs implementations to direct the search toward the objectives of the planning problem. To achieve this, the state sampling process is modified to include a small possibility of sampling the target. The tree expands more greedily toward the goal with a higher this probability.

However, RRTs are not asymptotically optimal, and the algorithm only sets the connections between nodes once rather than performing rewiring. The development of RRT* [29] altered this since it enables rewiring of the tree connections, shortening the distance from the root to the leaf. RRT* is asymptotically optimal, but in large environments in particular, its convergence is slow. Informed RRT* [30] introduced a focused sampling method that samples new nodes inside an ellipsoid. The starting and goal points serve as the ellipsoid's focal points. Particularly in big environments, this strategy accelerated RRT*'s convergence rate. In another way, the Bi-RRT algorithm was proposed by Kuffner and LaValle [31] in which two trees were grown from the initial state and target state, respectively, thus improving the algorithm's exploring and convergence speed. RRT-connect was then put forth as a way to increase node extension efficiency [32].

2.2.3 Ant Colony Optimization algorithm

When it comes to intelligent algorithms, ACO is probably best known as a bionic optimization. It is a global optimization algorithm. The key challenge of this approach is how to increase the algorithm's capacity for global searches and rate of convergence. To find the global optimal path quickly, the ant colony algorithm must make the search space as large as possible and take advantage of prior knowledge. The main goal is to resolve discrepancies between the algorithm's randomization and the intensity of the pheromone update. As a result of this issue, numerous researchers have conducted in-depth research on this subject from two angles: search strategy and pheromone update strategy, including bidirectional different search strategies and taboo table optimization strategy [33], turn-back search strategy [34], and Max-Min ant system [35]. In addition, numerous hybrid intelligent optimization algorithms have been put up in recent years as answers to the path-planning problem. A fusion algorithm of ant colony and particle swarm [36] was introduced to navigate a mobile robot in an environment filled with obstacles. And [37] proposed a combination of an ant colony algorithm with an immune algorithm to solve the Travelling Salesman Problem (TSP).

ACO is inspired by the foraging behavior of ants. At the core of this behavior is the indirect communication between the ants with the help of chemical pheromone trails [38], which enables them to find short paths between their nest and food sources, as shown in Fig. 2.3. Blum [39] used ACO algorithms to solve global optimization issues by taking advantage of this trait of actual ant colonies. Dorigo

[40] developed the first ACO algorithm and since then numerous improvements of the ant system have been proposed. The ACO algorithm has good scattered calculative mechanisms and strong robustness. ACO is simple to combine with other approaches and performs well in solving challenging optimization issues. ACO uses basic mathematical formulas to move these ants in the search space in accordance with the transition probability and total amount of pheromone in the area to optimize a problem.

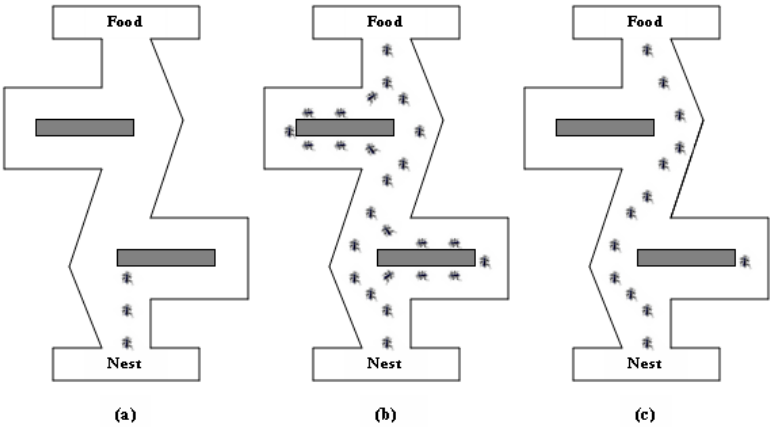


Figure 2.3: Ant colony optimization [41]

Each ant advances from state x to state y, which corresponds to a more complete intermediate solution, at each step of the process. Thus, in each iteration, each ant k computes a set $A_k(x)$ of viable expansions to its current state, and in probability, it travels to one of them. For an ant k, the likelihood p_{xy}^k that it will move from state x to state y depends on the interaction of two values: the attractiveness of the move (as determined by some heuristic), which indicates its a priori desirability, and the trail level of the move (a measure of how effective it has been in the past). The trail level serves as an a posteriori assessment of the move's desirability.

In general, the kth ant moves from state x to state y with probability

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in \text{allowed}_x}(\tau_{xz}^\alpha)(\eta_{xz}^\beta)} \tag{2.2}$$

where τ_{xy} is the quantity of residual pheromone for transition from state x to $y, \alpha \geq 0$ is a factor to control the effect of τ_{xy}, η_{xy} is the desirability of state transition xy (a priori knowledge, typically $1/d_{xy}$, where d is the distance) and $\beta \geq 1$ is a parameter to regular the impact of η_{xy}. τ_{xz} and η_{xz} represent the trail level and attraction for the other possible state transitions.

When all ants have finished their solution, the trails are typically updated, with the level of the trails increasing or decreasing to reflect moves that were a part

of "good" or "bad" solutions, respectively. A typical rule to update the global pheromone is:

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k}^{m} \Delta\tau_{xy}^{k} \tag{2.3}$$

where τ_{xy} is the is the quantity of residual pheromone for a state transition xy, ρ is the pheromone evaporation coefficient, m is the number of ants and $\Delta\tau_{xy}^{k}$ is the amount of pheromone deposited by kth ant, typically given for a TSP (with moves corresponding to arcs of the graph) by

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where L_k is the cost of the kth ant's travel (typically distance) and Q is a constant.

2.3 Local Path Planning

When the robot moves along the global path but in a dynamic or unknown environment, the information of the real environment refreshes over time. For example, some obstacles may newly appear or move. To react to the obstacles and changes in the environment according to the information updated by the perception system in real time, Local Path Planning (LPP) is mandatory.

In LPP, normally, a robot is guided by a global path generated by GPP from a starting point to the target point, which is the shortest path, and the robot follows the path till it senses obstacles. Then the robot performs an obstacle avoidance algorithm by deviating from the path and, at the same time, updates some important information such as the new distance from the current position to the target point, obstacle leaving point, etc. To precisely reach the objective in this sort of path planning, the robot must constantly be aware of the distance between the target point and its current location. The local planner converts the global path into local one piece by piece while taking into account the dynamic obstacles and the kinematic or dynamic constraints of the robot. The map is therefore condensed to the area around the robot and updated as it moves in order to recalculate the path at a certain rate. The global map cannot be directly used for LPP because the sensors cannot reach and update it in every region, and using a high number of cells would increase the processing cost.

Varying structures and characteristics of robots results in various kinematic and dynamic constraints; and the application environments are also different, for example, AMR working in factory, autonomous vehicle moving on the street. From the specified requirements came diverse LPP methods.

2.3.1 Dynamic Window Approach

The Dynamic Window Approach (DWA) [42] is a velocity-based local planner that calculates the optimal collision-free ('admissible') velocity for a robot required to reach its goal. A Cartesian target (x, y) is converted into a velocity command (v, w) for a moving robot. An advantage of this method is that it taking into account the kinematic and dynamic constraints of a mobile robot during planning (many of the vector field and vector field histogram approaches do not).

The determination of a valid velocity search space and the choice of the optimal velocity are the two key objectives of DWA. The set of velocities that create a safe trajectory, or allow the robot to halt before colliding, given the set of velocities the robot can achieve in the following time slice considering its dynamics (the "dynamic window"), constitute the search space. To acquire the heading that is closest to the goal, maximize the robot's clearance and velocity, the optimal velocity is chosen. In [42] and [43], several recommendations are made for the utility function, including the norm of the resulting velocity vector (a large norm is preferred, allowing robots to move at the highest efficiency under the limit of velocity), minimum clearances to obstacles, the possibility of stopping at the target point, and components of velocity alignment with the preferred direction.

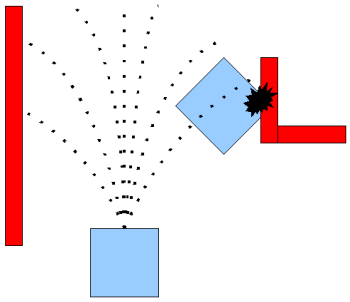


Figure 2.4: Dynamic Window Approach [44]

The basic processes of DWA are as follows:

1. Discretely sample in the control space of the robot $(dx, dy, d\theta)$.

2. Perform a forward simulation from the robot's current state for each sampled velocity to predict the future state in a short time slot, which will generates trajectories with same amount of velocity samples.

3. Use a scoring system that takes into account factors like speed, distance to obstacles, goal, and the global path to evaluate each trajectory that emerged from the forward simulation. Discard infeasible trajectories (those that collide with obstacles).

4. Select the trajectory with the best score, and send the corresponding velocity to the mobile base.

5.Rinse and repeat.

Furthermore, the Global Dynamic Window Approach [43] uses an NF1 navigation function to guide the DWA. The NF1 function is a lookup table that has already been constructed that contains the shortest path lengths between each cell in an occupancy grid and the target. The NF1 database is too expensive to update frequently and only captures obstacles as a static snapshot. The motion planning algorithms must ultimately modify their behavior in response to changing environmental conditions. Implementing an intelligence that modifies these parameters from motion planning algorithms is therefore crucial. Depending on the local conditions, several investigations adjust DWA constants using novel solutions. In the past, Hong et al. [45] adapted DWA using a fuzzy logic controller, achieving new suitable weights. The desired position and the position of the obstacles are taken into account while changing these weights. Abubakr et al. [46] adopted the same controller; however, the weights were adjusted with obstacle distribution information.

2.3.2 Time Elastic Band

In 1993, Sean Quinlan et al [47] proposed the Elastic Band (EB) algorithm for obstacle avoidance. The algorithm regards the path as a rubber band consisting of waypoints connected by springs that generate internal tension. One end of the elastic band is fixed to the vehicle, as depicted in Fig. 2.5, and obstacles are modeled as external points with repulsive forces, causing the deformation of the rubber band, in other words, the path.

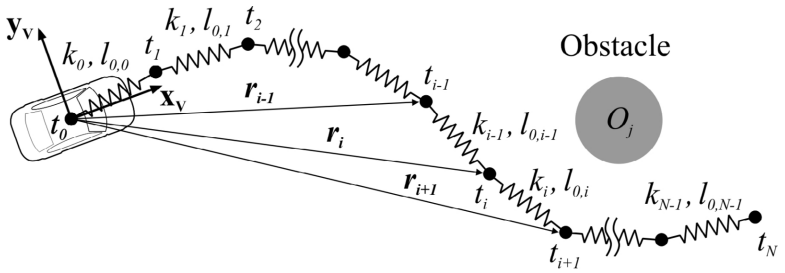


Figure 2.5: Single elastic band described in a vehicle fixed reference frame (x_V, y_V) and an obstacle O_j [48]

Since EB does not consider constraints like the minimum turning radius, maximum speed and acceleration of the moving robot, it has not received much attention and got general application. With the continuous development of science and technology, Christoph Rosmann et al [49]. proposed a Time Elastic Band (TEB) path planning algorithm based on the EB algorithm in 2012. The TEB transforms

an initial path made up of a series of waypoints into a trajectory with a clear dependence on time, allowing for real-time control of the robot. The approach can be easily expanded to include additional objectives and constraints due to its modular formulation. Incorporating the idea of graph optimisation and taking into account time interval information, the algorithm has velocity constraints, acceleration constraints, kinematic constraints, and incomplete constraints on the robot, as well as constraints on obstacle distances. Thus, the LPP problem is transformed into a multi-constraint objective optimisation problem depending on a few consecutive configurations, which leads to the sparse structure. Fig. 2.6 shows the general framework of graph optimization, g2o, which could be used to quickly and effectively solve the sparse structure like TEB based on the hyper-graph, improving the calculation speed, where vertex s stands for pose, vertex dt stands for differential time, edge h represents kinematic constraint, edge ρ represents obstacle constraint, edge v is velocity constraint, and a is velocity constraint. Last but not least, edge t denotes optimal time constraint. By solving this graph optimisation problem, the optimal controlled variables such as velocity and acceleration can be derived, as can the path.

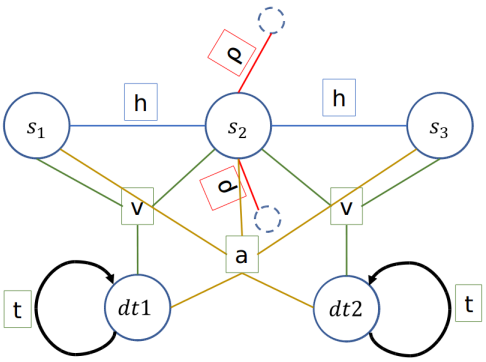


Figure 2.6: g2o frame of Timed Elastic Band algorithm [50]

Compared with the traditional EB algorithm, the TEB algorithm considers more comprehensive and practical constraints, and has strong adaptability to dynamic obstacles, so it is widely used in autonomous navigation of robots.

2.3.3 Artificial Potential Field method

The Artificial Potential Field (APF) method originates from the idea of field theory in physics and was first proposed by Khabit and Krogh [51]. The foundation of APF approach is the gradient descent search method, which aims to minimize

the potential function. Goal point is surrounded by an attractive potential field, whereas obstacles that must be avoided are surrounded by repulsive potential fields. If the surroundings is unimpeded, the attractive potential is typically a bowl-shaped energy well that pulls an object toward its center. However, in an environment with obstructions, repulsive potential energy hills are added to an attractive potential field at the sites of the obstructions in order to repel the objects, as depicted in Fig.2.7. A force equivalent to the potential's negative gradient is applied to the item. This force pushes the object downward until it reaches the location where it expends the least amount of energy. The technique is frequently used in the field of real-time obstacle avoidance and path planning due to its benefits of straightforward mathematical analysis, cheap computer complexity, and a smooth path [52].

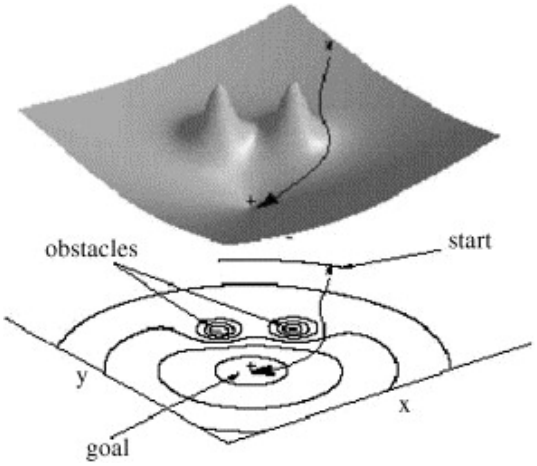


Figure 2.7: Artificial potential field method [53]

However, this approach has certain drawbacks: 1) There is no way to go past obstacles that are closely placed; 2) In a narrow passage, the robot overshoots its equilibrium position and either oscillates or runs in a closed loop; and 3) Goal Non-Reachable with Obstacles Nearby (GNRON) problems. The robot might be caught in a local minimum before completing its task. Therefore, the APF has been actively researching the avoidance of local minima. The use of effective search algorithms that can escape from a local minimum and the redefinition of potential functions with no or few local minima have both helped to solve this issue. The former class of strategies include: repulsive potential functions with circular thresholds [54] or Gaussian shapes [55], the navigation function [56], and the superquadratic potential function [57].

Chapter 3

Architecture of Autonomous Mobile Robots based on ROS

An AMR system can be divided into two major subsystems: software system and hardware system. A mobile robot's hardware consists of mechanical and electronic components that must work in a synergistic way. Consequently, the design phases must take into account both mechanical and electronic processes and procedures [58]. The software system is composed of the upper computer system and the bottom drive system. Generally speaking, the hardware structure of most mobile robots is similar, mainly including chassis, power module, actuators, controllers and sensors, and the physical connection of these components are easy to build. But writing software for robots is challenging, especially as robotics' scale and scope continue to expand. The hardware of various robots can differ greatly, making code reuse complicated. Additionally, the quantity of the necessary code can be overwhelming because it must include a deep stack that extends from driver-level software up through perception, abstract thinking, and beyond. Robotics software designs must also facilitate extensive software integration efforts, as the required breadth of knowledge is far beyond the scope of any individual researcher [59]. To solve this problem, the increasingly popular Robot Operating System (ROS) is used as the framework of the robotics software which greatly simplifies the process of building the software system.

3.1 Robot Operating System

ROS is an open-source, meta-operating Robot Operating System. ROS offers a structured communications layer on top of the host operating systems of a heterogeneous compute cluster rather than functioning as an operating system in the classic sense of process management and scheduling, more specifically, providing services, such as hardware abstraction, low-level device control, common functionality implementation, message-passing between processes, and package management. Additionally, it offers resources and libraries for accessing, creating, writing , and executing code across numerous machines.

3.1.1 Design goals

It is possible for executables to be separately programmed and loosely connected at runtime because to the distributed infrastructure of processes known as ROS Nodes. These processes can be organized into Stacks and Packages that are simple to share and distribute. In order to promote distributed cooperation, ROS also provides a federated system of code repositories. This design allows for individual decisions to be made about development and implementation at every level, from the filesystem to the community, but it can all be gathered via ROS infrastructure tools.

In support of sharing and collaboration, the philosophical goals of ROS can be summarized as:

1. Thinness: ROS is made to be as thin as possible. It is quite simple to adhere to this "thin" philosophy because the ROS build system uses CMake and conducts modular builds inside the source code tree. Since almost all complexity is contained in libraries, and only tiny executables that expose library functionality to ROS are produced, code can be extracted and reused more easily than it was intended to. There is no need to wrap *main*() function, so that code written for ROS can be used with other robot software frameworks. This has the logical consequence that ROS is simple to connect with various robot software frameworks: OpenRAVE, Orocos, and Player have already been integrated with ROS.

2. ROS-agnostic libraries: The desired development approach is to create libraries that are independent of ROS and have clear, useful interfaces.

3. Language independence: The ROS framework is simple to implement in any current programming language, demonstrating its language independence. It is already implemented in Python, C++, Octave, and LISP, besides, and it has experimental libraries in Java and Lua.

4. Simple testing: The unit/integration test framework integrated into ROS, rostest, makes it simple to set up and take down test setups.

5. Scaling: Both big runtime systems and extensive development processes are suitable for ROS.

6. Peer-to-Peer: A system created using ROS is made up of several processes that may be running on various hosts and are linked in real time by a peer-to-peer topology. Peer-to-Peer connectivity, combined with buffering or "fanout" software modules where necessary, avoids the issue of unnecessary traffic flowing across the slow wireless link.

7. Tools-based: Rather than creating a monolithic development and runtime environment, ROS is designed with a microkernel, where a large number of small tools are utilized to generate and run the many ROS components.

3.1.2 Computation graph level

The peer-to-peer network of ROS processes that are working together to process data is known as the Computation Graph. Nodes, Masters, Parameter Servers, Messages, Services, Topics, and Bags are the fundamental concepts of ROS' Computation Graph, all of which contribute data to the Graph in various ways.

Nodes are processes that perform computation. It is practical to represent peer-to-peer communications as a graph when multiple nodes are active, with processes acting as graph nodes and peer-to-peer links as arcs. There are three main ways of communication between nodes: publishing and subscribing topics, requesting and responding to services, and setting parameter servers. The topic is the carrier of information (message) in ROS. A node communicates with another node by publishing information on a topic or subscribing to information on a topic. This method does not require nodes to know each other's existence, similar to UDP. Correspondingly, the service is similar to the TCP/IP, a node transmits information by requesting a service from another node; and the parameter server can pass parameters before or when the node runs. These three communication methods enable nodes to be loosely coupled together, and the collapse or change of some nodes will not cause the overall collapse. The basic structure of ROS node communication is shown in Figure 3.1.

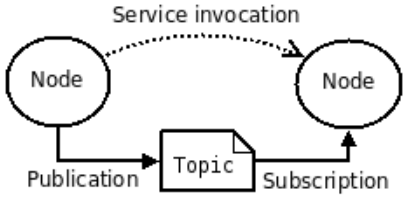


Figure 3.1: ROS node communication [60]

As mentioned above, ROS has built a complete robot software framework for

developers, and there are many ready-made algorithms available that do not require developers to rewrite the relevant code. For example, when studying the problem of path planning, robot localization, mapping, and other tasks must be completed. Fortunately, ROS provides developers with a complete navigation framework. The goal of this thesis is path planning. Therefore, the algorithm verification function can be realized just by replacing the path planning algorithm in the navigation system framework (as it will be described in more detail in Chapter 4), which allows developers to focus on the implementation of the algorithm itself, saves most of the time, and provides the possibility for rapid algorithm verification [59].

3.2 Kinematic model of OMR with Mecanum wheels

Wheeled mobile robots, tracked mobile robots, and legged mobile robots are examples of typical mobile robots. There are constraints at the point where the wheels make contact with the ground for the wheeled mobile robot. Wheeled mobile robots are typically divided into three categories: car-like robots, differential robots, and omnidirectional robots, depending on the extent of the constraints. Similar to an automobile, the car-like mobile robot cannot move sideways or rotate in place due to the limits of Ackermann steering (as shown in Fig. 3.2), making it impossible to use it in confined spaces. In-situ rotational movement is possible for differential robots with two independent driving wheels and one or more truckles, but sideways movement is not possible. The omnidirectional robots have extremely powerful mobility because they can move not only in-situ but also sideways. These mobile robots are ideal for use in automated factories, hospitals, homes, and other narrow spaces and have a wide range of potential uses [61]. The benefits and drawbacks of these three kinds of wheeled models are listed in table 3.1. As a result, OMR is preferred due to its high flexibility, which is required for collision avoidance in a dynamic environment.

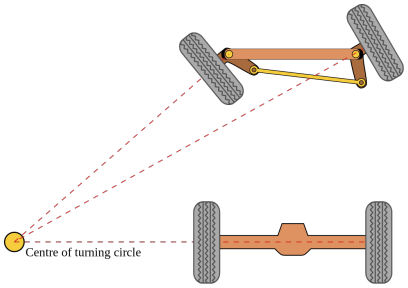


Figure 3.2: Ackermann geometry [62]

Kinematic model	Differential	Car-like	Omnidirectional
Pros	Good movement performance, easy to control	Similar to real cars	Good mobility track path precisely
Cons	Steering with slip, high wear	Limited steering radius	High requirement of road condition, complex in structure, high wear

Table 3.1: Pros and cons of three kinds of wheeled models

3.2.1 Omnidirectional Mobile Robot

Mobile robots should be able to move and maneuver well in confined spaces and around obstacles. The design of the wheels largely determines these capacities. To increase the capacity of mobile robotic systems to navigate autonomously, research is ongoing in this area [63].

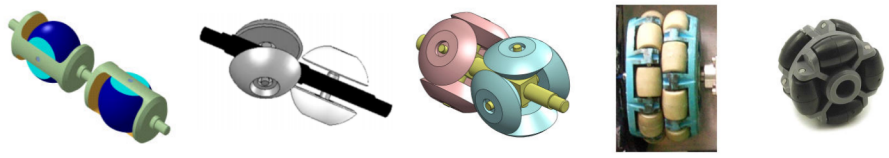


Figure 3.3: Omnidirectional wheel mechanisms : (a) Longitudinal Orthogonal-wheel. (b)MY wheel. (c) MY wheel-II. (d) Swedish wheel. (e) Omni-wheel. [64]

Various omnidirectional wheel mechanisms were proposed in the past few decades. These mechanisms can be classified into two categories: special wheels and conventional wheels. Many special wheel structures are designed based on the concept that traction is achieved with constraints in one direction and passive motion is allowed with the constraints removed in another direction. A traditional special wheel design involves mounting a number of passive rollers on a regular wheel's outer rim so that the driven wheels' rotating spindles are perpendicular to or at an angle to the normal wheel, such as the Mecanum wheel. Other special wheel mechanisms include orthogonal wheel and ball wheel mechanism. These OMRs with special wheels have good mobility, stability, and the capacity for precisely tracking the path, but they typically have a complex structural design. In addition, these

robots typically have drawbacks like a limited capacity for carrying loads and car body vibration that is easily caused by driven rollers, among other things. Because small particles can readily fit into the rotating spindle of the powered roller and prevent the robot from moving, these wheel structures are not suitable for floors covered in dirt and debris. The simple mechanism of the orthogonal wheel cannot support enough weight. The structure of the flexibly driven ball wheel system is very complicated. The OMRs based on conventional wheel mechanism mainly adopt the casters individually driven in the rotation direction and revolving as their driving wheels, which has poor stability and agility although it has strong load capacity. A short comparison between Mecanum drive, holonomic drive and swerve drive is presented in Fig. 3.4.

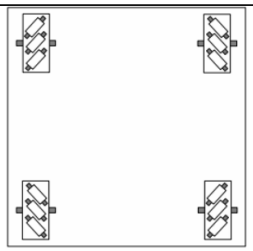
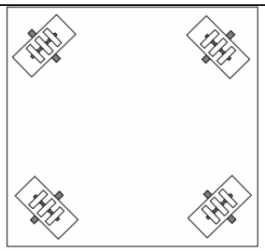
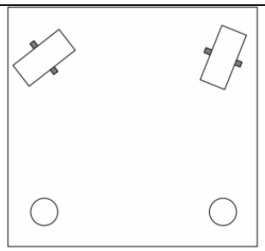
	Mecanum drive	**Holonomic drive**	**Swerve drive**
Description	 Wheels with angled rollers	 Wheels with "straight" rollers (omniwheels)	 Independently steered drive modules
Advantages	- compact design - high load capacity - simple to control - less speed and pushing force when moving diagonally	- low weight - compact design - simple to control - less speed and pushing force when moving diagonally	- simple conceptually - simple wheels - continuous wheel contact - high load capacity - robust to floor conditions
Disadvantages	- very complex conceptually - discontinuous wheel contact - high sensitivity to floor irregularities - complex wheel design	- more complex conceptually - discontinuous wheel contact or variable drive-radius - sensitive to floor irregularities - lower traction	- complex mechanical design - heavy and massive design - complex to program and control - high friction and scrubbing while steering

Figure 3.4: Comparison between different types of drives [65]

Invented in 1973 by a Swedish engineer (Mecanum Company), named Ilon (Ilon, 1975), The mecanum wheel is an omnidirectional wheel that enables any direction movement for a land-based vehicle. The Mecanum wheel is designed after a tireless wheel and has a number of external rubberized rollers that are affixed obliquely to the entire circle of its rim. Each of these rollers normally has a rotational axis that is 45 degrees from both the wheel plane and the axle line, as depicted in Fig. 3.5. A pushing force perpendicular to the roller axle is produced as each Mecanum wheel

spins. This force can be divided into a longitudinal component and a transverse component in reference to the vehicle. Each Mecanum wheel is an independent non-steering drive wheel with its own motor.

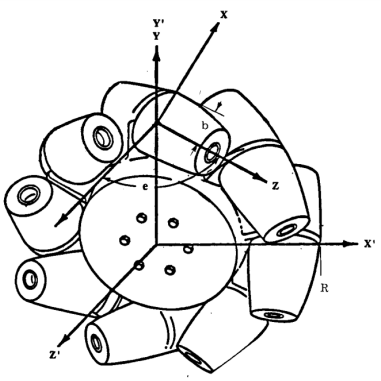


Figure 3.5: A Mecanum wheel with coordinate system [66]

The typical Mecanum design is the four-wheel configuration as demonstrated by one of the Scout mini (Fig. 1.2). In this approach, the thrust produced by each wheel will be about parallel to the appropriate frame diagonal. The vehicle may move around with little requirement for space by adjusting the rotational speed and direction of each wheel. The resultant force of the force vectors exerted on each wheel generates both linear motions and/or rotations of the AMR. For example:

1. A forward/backward movement results from driving all four wheels in the same direction at the same speed because the longitudinal force vectors add up while the transverse force vectors cancel out;

2. Because the transverse vectors cancel out but the longitudinal vectors couple to produce a torque around the vehicle's central vertical axis, running both wheels, all at the same speed, on one side in one direction while the other side in the opposite direction results in a stationary rotation of the vehicle;

3. The sideways movement is caused by the transverse vectors adding up but the longitudinal vectors canceling out when the diagonal wheels are moving in one direction while the other diagonal is moving in the other direction, all at the same speed.

Vehicle mobility in practically any direction and with any rotation is possible with a combination of differential wheel motions, as shown in Fig. 3.6.

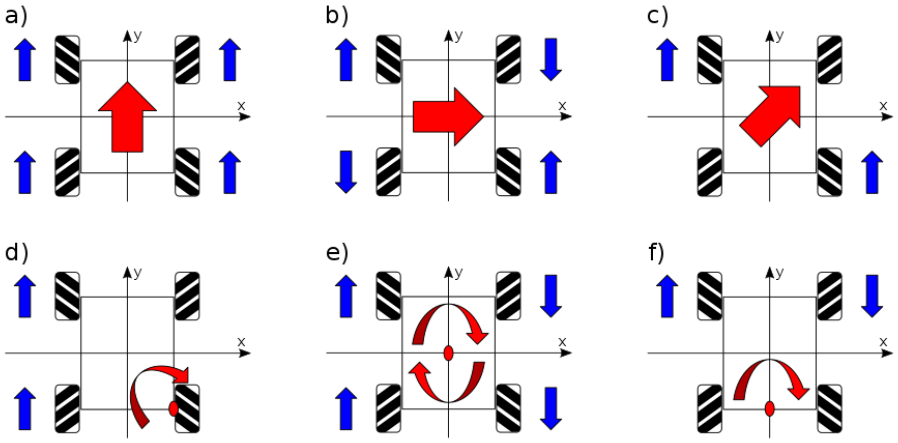


Figure 3.6: Movements to any directions: blue: wheel drive direction; red: vehicle moving direction a) Moving straight ahead, b) Moving sideways, c) Moving diagonally, d) Moving around a bend, e) Rotation, f) Rotation around the central point of one axle [67]

3.2.2 Kinematic model

A mathematical model of the robot should be extracted in order to create a control system and collision avoidance plan. As a result, the robot's kinematic and dynamic modeling is carried out in this part, and consequently the differential equations for the robot's motion are developed [68].

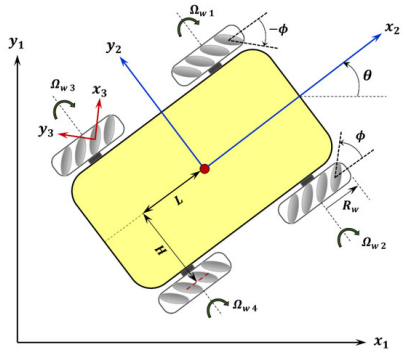


Figure 3.7: The schematic model of OMR and the attached frames [68]

Figure 3.7 shows the schematic representation of the OMR with four Mecanum wheels. The Mecanum wheels give the robot an additional degree of freedom to move in a lateral direction in comparison to conventional nonholonomic wheeled mobile robots. Three coordinate frames are constructed as indicated in Figure

3.7 in order to calculate the robot's position, speed, and acceleration. Coordinate $frame$ 1 is the inertial reference frame. The body coordinate, $frame$ 2, is aligned to the robot's center of mass and spins with it around the z axis. When coordinate $frame$ 3_i; $i = 1...4$ is mounted to the center of rollers that are in contact with the ground, its z axis is parallel to $frame$ 2's z axis and its x axis coincides with the rollers' axis of rotation. The corresponding wheel rotates with this coordinate frame, so their angular velocities are equal. Therefore, the roller rotates around its x axis at a relative angular speed Ωri. Three generalized coordinates could be used to describe the position and orientation of the robot in $frame$ 1, $q = (x, y, \theta)$. As shown in Figure 3.7, the angles of rollers with respect to x axis of $frame$ 2 are $\Gamma = [\frac{\pi}{2} + \phi \ \phi \ \phi \ \frac{\pi}{2} + \phi]$ respectively. The coordinate transformation matrices between $frame$ 2 *and* 1 and $frame$ 3 *and* 2 are as follows.

$$ {}^1_2R = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}; \ {}^2_1R = {}^1_2 R^T \tag{3.1} $$

$$ {}^2_3R_i = \begin{bmatrix} cos\Gamma_i & -sin\Gamma_i & 0 \\ sin\Gamma_i & cos\Gamma_i & 0 \\ 0 & 0 & 1 \end{bmatrix}; \ i = 1...4 \tag{3.2} $$

The movable platform (MP), the wheels, and the rollers that are in contact with the ground are each given a number from 1 to 9, respectively, to simplify the notation. As a result, the linear and angular velocities of the MP are as follows using the time derivative of generalized coordinates:

$$ \begin{aligned} {}^1V_1 &= \begin{bmatrix} \dot{x}, \dot{y}, 0 \end{bmatrix}^T \\ {}^1\omega_1 &= {}^2 \omega_1 = \begin{bmatrix} 0, 0, \dot{\theta} \end{bmatrix}^T \end{aligned} \tag{3.3} $$

$$ {}^2V_1 = {}^2_1 R {}^1V_1 \tag{3.4} $$

where the vector's frame number is indicated by the left superscript. The linear velocity of the center of each wheel in $frame$ 2 is calculated in terms of its position with respect to the robot's center of gravity and the MP velocity.

$$ {}^2V_i = {}^2 V_1 + {}^2 \omega_1 \times {}^2 r_{i/G}; \ i = 2...5 \tag{3.5} $$

where ${}^2r_{i/G}$ stands for the position vector of each wheel in $frame$ 2. It is presumptive that each wheel can independently rotate with angular velocity $\Omega_{wi}; i = 1...4$ relative to MP. Thus, the angular velocity of wheels is determined as follows.

$$
\begin{aligned}
&{}^2\omega_i = [0\ \Omega_{wj}\ \dot{\theta}]^T;\ i = 2...5,\ j = 1...4 \\
&{}^1\omega_i =_1^2 R^2\omega_i
\end{aligned}
\tag{3.6}
$$

The rollers of the Mecanum wheels can also rotate freely with respect to the wheel's body by $\Omega_{ri}; i = 1...4$ and their angular velocity vectors could be written in the following form.

$$
\begin{aligned}
&{}^3\omega_i = [\Omega_{rj}\ 0\ \dot{\theta}]^T;\ i = 6...9,\ j = 1...4 \\
&{}^2\omega_i =_3^2 R^3\omega_i
\end{aligned}
\tag{3.7}
$$

Using (3.6), the linear velocity of the center of each roller is calculated as follows:

$$
{}^2V_j =^2 V_i - [R_w\Omega_{wj}\ 0\ 0]^T;\ i = 2...5,\ j = 6...9 \tag{3.8}
$$

where R_w is the radius of wheels. It is expected that the Mecanum wheels' rollers are moving without slipping on the surface. As a result, the velocity at each roller's point of contact with the ground is zero. (3.9) can also be used to calculate the linear velocity in the center of the rollers.

$$
{}^2V_j =^2 \omega_i \times [0\ 0\ R_r]^T;\ i = 6...9,\ j = 6...9 \tag{3.9}
$$

where R_r is the radius of rollers. (3.8) and (3.9) can be combined to form a system of eight algebraic equations that can be solved to determine the following values for the angular speeds of the four wheels with respect to the MP (Ω_{wi}) and the four rollers with respect to the body of the wheels (Ω_{ri}).

$$
\Omega_w =^1 J_w\dot{q} =^1 J_w = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix};\ \Omega_r =^1 J_r\dot{q} =^1 J_r = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}; \tag{3.10}
$$

where Ω_w *and* Ω_r (4×1) are vectors of angular speeds of the wheels and rollers respectively, and J_w is the Jacobian matrix between wheels angular speed and the MP velocity in $frame$ 1 in the following form.

$$
J_w = \frac{1}{R_w}\begin{bmatrix} sin(\theta+\phi)/sin\phi & -cos(\theta+\phi)/sin\phi & -(Lcos\phi + Hsin\phi)/sin\phi \\ cos(\theta+\phi)/sin\phi & sin(\theta-\phi)/sin\phi & (Hcos\phi + Lsin\phi)/cos\phi \\ cos(\theta+\phi)/sin\phi & sin(\theta-\phi)/sin\phi & -(Hcos\phi + Lsin\phi)/cos\phi \\ sin(\theta+\phi)/sin\phi & -cos(\theta+\phi)/sin\phi & (Lcos\phi + Hsin\phi)/sin\phi \end{bmatrix}; \tag{3.11}
$$

The Jacobian matrix J_r with a similar form can be transformed into $frame$ 2 using coordinate transformation matrix from (3.1).

The following relationship between the angular velocities of the wheels is clear from (3.10) and (3.17), and it imposes a constraint resulting from the MP's 3 DOF planar motion.

$$\Omega_{w1} + \Omega_{w2} - \Omega_{w3} - \Omega_{w4} = 0 \tag{3.12}$$

Using (3.17) and the pseudo invers of the Jacobian matrix ${}^1J_w$, it was able to calculate the robot's linear and angular velocities in terms of the angular speeds of its wheels.

$$\begin{aligned} {}^1J^* &= ({}^1J^{T1}J)^{-11}J^T; \\ \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} &= {}^1J^*\Omega_w; \end{aligned} \tag{3.13}$$

(3.13) can be used to determine the MP's velocity in terms of the angular speed of the wheels and is in fact the kinematic model of the OMR.

The linear and angular accelerations of each robot component should be described in order to obtain the robot's dynamic model. Following are the calculations for the MP's linear and angular acceleration:

$$\begin{aligned} {}^1a_1 &= [\ddot{x}, \ddot{y}, 0]^T \\ {}^1\alpha_1 &=^2 \alpha_1 = [0,0, \ddot{\theta}]^T \\ {}^2a_1 &=^2_1 R^1a_1 \end{aligned} \tag{3.14}$$

The linear and angular velocities of the rollers and the angular velocities of the wheels are calculated in terms of q *and* $\dot{q}$ by substituting Ω_{wi} and Ω_{ri} from (3.10) in (3.6) and (3.8). Therefore, the linear and angular acceleration of these portions are determined by using the time derivative of (3.5) and (3.8) in $frame$ 1 as follows

$${}^2\alpha_i = \sum_{j=1}^{3} \frac{\partial({}^2w_i)}{\partial q_j}\dot{q}_j + \sum_{j=1}^{3} \frac{\partial({}^2w_i)}{\partial q_j}\ddot{q}_j +^2 w_1 +^2 w_i;\ i = 2...9 \tag{3.15}$$

$${}^2a_i = \sum_{j=1}^{3} \frac{\partial({}^2V_i)}{\partial q_j}\dot{q}_j + \sum_{j=1}^{3} \frac{\partial({}^2V_i)}{\partial q_j}\ddot{q}_j +^2 w_1 +^2 V_i;\ i = 2...9 \tag{3.16}$$

3.3 Simulation

3.3.1 The URDF model

Unified Robotics Description Format (URDF), is an XML format used in academia and industry to model multibody systems including animatronic robots for theme parks and robotic manipulator arms for manufacturing assembly lines. Users of ROS, which provides built-in support for URDF models, are particularly fond of URDF. Parallel robots are currently not a possibility because only tree structures can be represented at this time. Furthermore, flexible components are not supported; the specification implies that the robot is made of stiff links connected by joints.The specification covers:

- Kinematic and dynamic description of the robot
- Visual representation of the robot
- Collision model of the robot

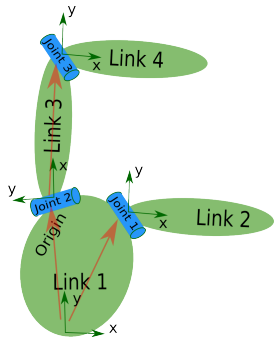


Figure 3.8: Sketch of a general URDF model [69]

The description of a robot consists of a set of link elements, and a set of joint elements connecting the links together, as shown in Fig. 3.8. So a typical robot description looks something like this:

```
<robot name="test_robot">
    <link name="link1">
    <visual/>
    <collision/>
    <inertial/>
    </link1>

    <link name="link2">
    <visual/>
    <collision/>
    <inertial/>
```

```
12      </link2>
13
14      <link name="link3">
15      <visual/>
16      <collision/>
17      <inertial/>
18      </link3>
19
20      <joint name="joint1" type="continuous">
21          <parent link="link1"/>
22          <child link="link2"/>
23          <origin xyz="0 0 0" rpy="0 0 0"/>
24      </joint>
25
26      <joint name="joint2" type="continuous">
27          <parent link="link1"/>
28          <child link="link3"/>
29          <origin xyz="0 0 0" rpy="0 0 0"/>
30      </joint>
31  </robot>
```

In the above file, the link element describes a rigid body with inertia, collision and visual features, as depicted in Fig. 3.10 (a). ROS provides transformation tool for users to input models from popular Computer Aided Design (CAD) tools such as Solidworks, Pro-engineer, Blender, etc. Although the sub-element, visual, defines robot appearance, in order to get collision detection and simulate the robot in Gazebo, the collision element is needed as well. In most of cases of practice, to get quicker processing during collision detection, simpler geometries in the collision element is preferred comparing to complex meshes that usually applied for visual elements. In addition, it is required to define several physical properties of the robot, i.e. the inertia and contact coefficients like friction, stiffness and damping coefficients. More specifically, a 3×3 rotational matrix is specified with the inertia element. This can only be expressed by 6 elements because it is symmetrical.

$$\begin{bmatrix} i_{xx} & i_{xy} & i_{xz} \\ i_{xy} & i_{yy} & i_{yz} \\ i_{xz} & i_{yz} & i_{zz} \end{bmatrix} \tag{3.17}$$

Modeling applications like MeshLab can supply this data. The moment of inertia tensors listed on Wikipedia can be used to calculate the inertia of geometric primitives like cylinders, boxes, and spheres.

The joint element sets the joint's safety limits in addition to describing the kinematics and dynamics of joints (Fig. 3.10 (b)). There are quite a few types of joints, but the most common are:

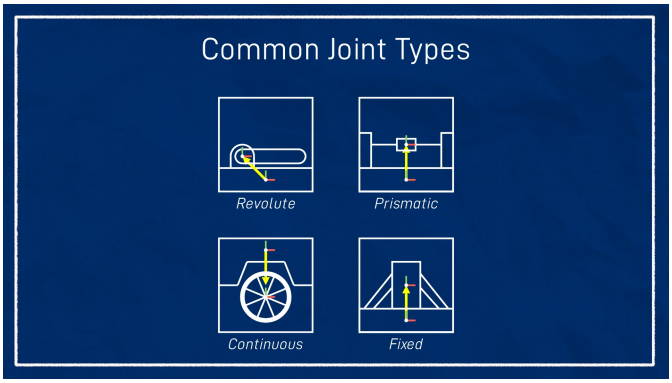


Figure 3.9: Common URDF joint types [70]

• revolute - a hinge joint that rotates along the axis within a limited range specified by the upper and lower limits.

• continuous - a continuous hinge joint that rotates around the axis without upper and lower limits.

• prismatic - a sliding joint that slides along the axis within a limited range defined by the upper and lower limits.

• fixed - This joint cannot move, hence it is not actually a joint. Every DOF is locked. There is no need for an axis, calibration, dynamics, limit, or safety controller with this kind of joint.

• floating - This joint allows motion for all 6 DOF.

• planar - This joint allows motion in a plane perpendicular to the axis.

Taking floating for example, the joint can be written as:

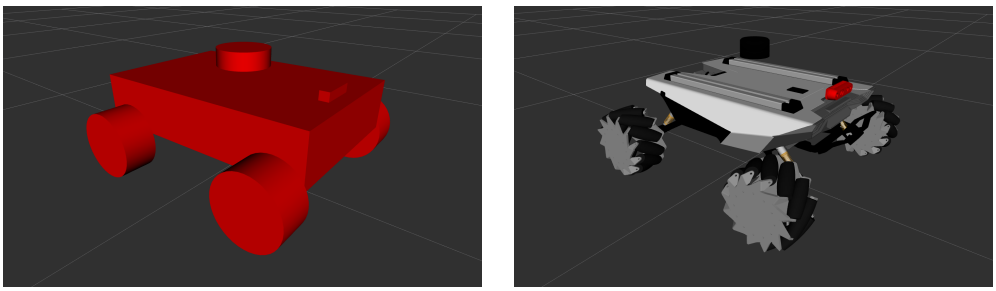
```
<joint name="my_joint" type="floating">
   <origin xyz="0 0 1" rpy="0 0 3.1416"/>
   <parent link="link1"/>
   <child link="link2"/>

   <calibration rising="0.0"/>
   <dynamics damping="0.0" friction="0.0"/>
   <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
   <safety_controller k_velocity="10" k_position="15"
 soft_lower_limit="-2.0" soft_upper_limit="0.5" />
</joint>
```

(a) link (b) joint

Figure 3.10: Sketches of link and joint [71]

To verify the path planning and obstacle avoidance algorithm, an URDF model is built according to the basic AMR, which consist of base, wheels, Lidar and camera as links, whose connections is defined by joints. Components like the chassis and camera are simply defined as cubic links, while Lidar and wheels are cylindrical links. Joints between wheels and chassis are defined as continuous, and joints between Lidar and chassis, camera and chassis are fixed. Then the visual elements, the meshes of each component, are attached to the corresponding link. In addition, the inertia matrices of the components are set. Fig. 3.11 shows the URDF model visualized by RViz, a 3D visualization tool for ROS, where (a) is the visual element while (b) collision element. This basic structure will be extended after verifying the feasibility of the path planning algorithms.



(a) collision element (b) visual element

Figure 3.11: URDF model of Scout mini with sensors

3.3.2 tf: the transform library

A robotic system typically has motors, sensors, controllers and communication. Each component defines its own coordinate. As a result, there are many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. When a robot implements tasks, it is necessary to acquire

the precise relative position between the parts of robot, as well as the objects in the working environment. The collision avoidance of an AMR can be taken as an example to demonstrate this. The task is simply moving the robot away from the detected obstacles. To do this task the relative location between the obstacles and the AMR must be obtained. For sure, the sensors which the AMR is equipped with, such as Lidar and camera, can detect the target, but the detected position is depends on the coordinate of the sensors. Firstly, the transformation between Lidar and camera should be obtained for sensor fusing, then the fused data should be transferred to the same coordinate frame with the base and wheels, usually, the world frame. Another example is a robotic arm that recognizes and grasps objects. The transformations between the base of the arm, the camera, the gripper, and the objects should be required. In a word, for any different types of robots execute diverse tasks, the transforms between all the coordinate frames within the system must be known. As the complexity of the robotic system growing quickly and application of distributed system expanding, keeping track of coordinate frames becomes increasingly difficult, which is a common pain point of developers.

To solve this problem, tf library is available [72], aiming to keep track of multiple coordinate frames over time by maintaining the relationship between coordinate frames in a tree structure buffered in time, which allows the individual component user transform points, vectors, etc between any two coordinate frames at any desired point in time without requiring knowledge of all the coordinate frames in the system.

The transform information is processed by the publish and subscribe mechanism mentioned in Section 3.1.2. Firstly, the message of transform information contain two pieces of data: the represented coordinate frame and the valid time, which referred as a Stamp. Secondly, the publisher broadcasts the data at certain frequency set by users. Lastly, the subscriber receives the values from publishers in a time slot as shown in Fig. 3.12, then put them into a sorted list where the interpolation can be done between two adjacent values. Thanks to the publish and subscribe mechanism, the tf library can accept inputs asynchronously, while the interpolation enhances robustness of the tf library. Besides, users can acquire desired transformation between any coordinate frames by function "lookupTransform".

Transformations and coordinate frames can be represented as a graph, with edges representing transforms and nodes representing coordinate frames. To reach quick look ups the graph is limited as tree shape, each node can only have one parent node but several child nodes, avoiding ambiguous searching result of transform. As an example, the tf tree of the model built in Section 3.3.1 is reported in Fig. 3.13:

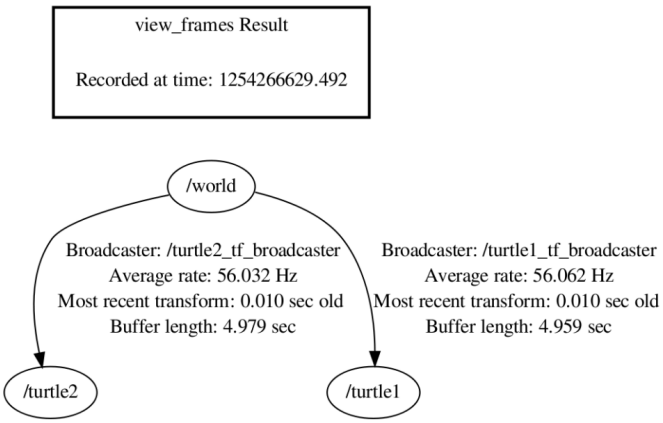


Figure 3.12: A simple tf tree from two turtles (i.e., two simple virtual robots) in one of the ROS tutorials, with debugging information. [72]

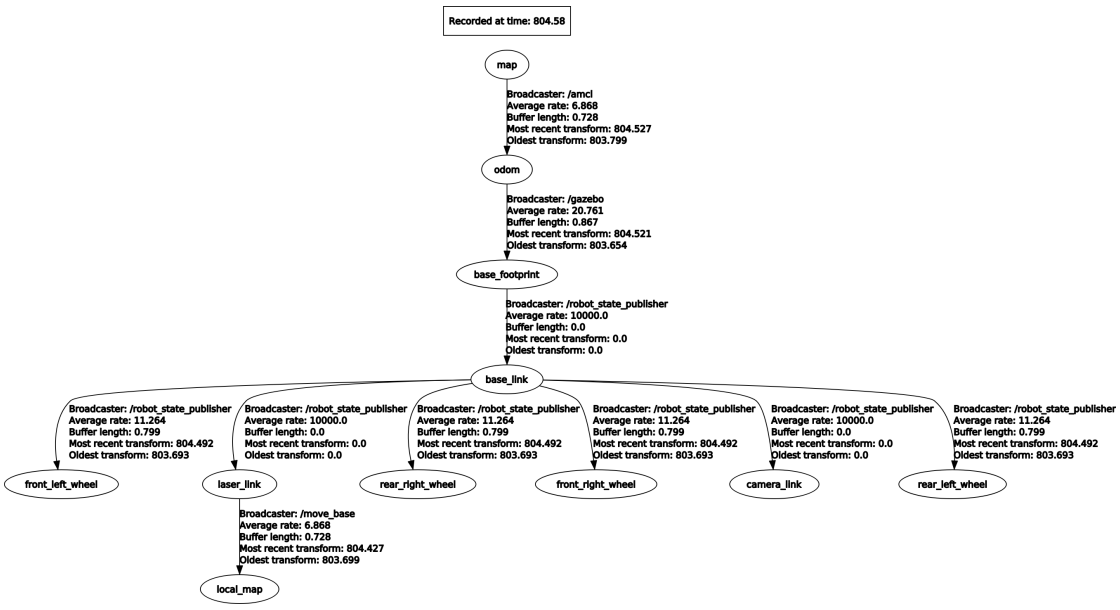


Figure 3.13: tf tree of the AMR model

Furthermore, the tf tree allowing dynamic changes according to the dynamic robot structure by modifying the nodes and edges. There are two ways most commonly used to publish tf tree when building the AMR model. One is publishing static tf transform directly by ROS package "static_transform_publisher", since although different components are installed on the base, their position are fixed with respect to base. For example:

```
<launch>
<node pkg="tf" type="static_transform_publisher" name="
 link1_broadcaster" args="1 0 0 0 0 0 1 link1_parent link1 100" />
</launch>
```

where, the arguments are "x y z yaw pitch roll frame_id child_frame_id period_in_ms".

The other way is by URDF model combined with " robot_state_publisher" and "joint_state_publisher", packages provided by ROS. In this way, the transforms are generated according to the robot URDF model, and after that, the corresponding tf tree is published, proving a straightforward presentation of transforms among frames, which is adopted by this thesis.

In the RViz rendering tool, the tf tree has been integrated into the Object-Oriented Graphics Rendering Engine OGRE scene graph. Fig. 3.14 shows the tf tree both rendered as elements in the OGRE scene graph, as well as being used to tell the OGRE scene graph of the positions of the meshes of the AMR model.

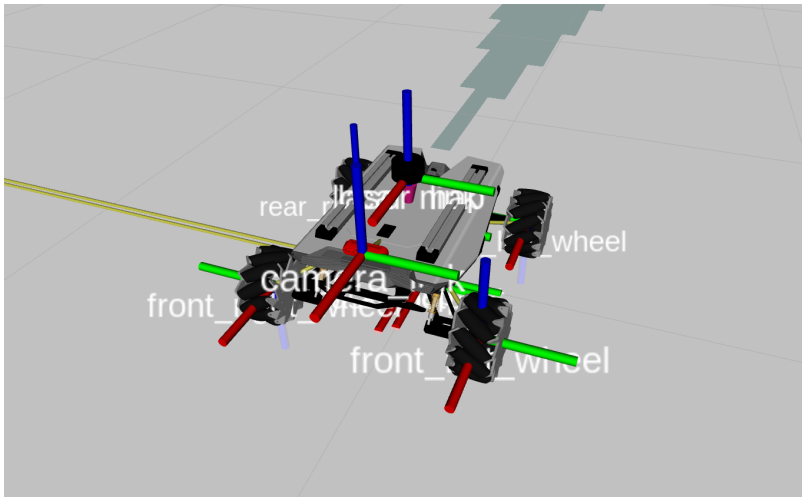


Figure 3.14: AMR model in RViz

3.3.3 Gazebo

Simulators have played an important role in robotics research, allowing developers to test novel concepts, methods, and algorithms quickly and efficiently. A robotics simulator is used to develop embedded robot applications without having to use the actual equipment, which saves money and time. In some cases, these applications can be directly translated to the real robot [73].

Gazebo is an open-source 3D robotics simulator including libraries for physics simulation, rendering, user interface, communication, and sensor generation. Gazebo can use multiple high-performance physics engines, such as ODE, Bullet, etc (the default is ODE) to generate physical interactions between objects. It also provides realistic rendering of environments such as high-quality lighting, shadows, and textures. We can use Gazebo to build a virtual "environment" and insert simulations of our robots into it. Simulated sensors can detect the environment, and publish the data to the same ROS topics that real sensors would, allowing easy testing of algorithms. After that, forces may be applied to the robot's simulated actuators while accounting for factors like friction. In a world, Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments, making it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios.

Utilizing the Gazebo simulator, a rough world model is built according to the CIM4.0 laboratory, as depicted in Fig. 3.15, which is the testing environment of FIXIT.

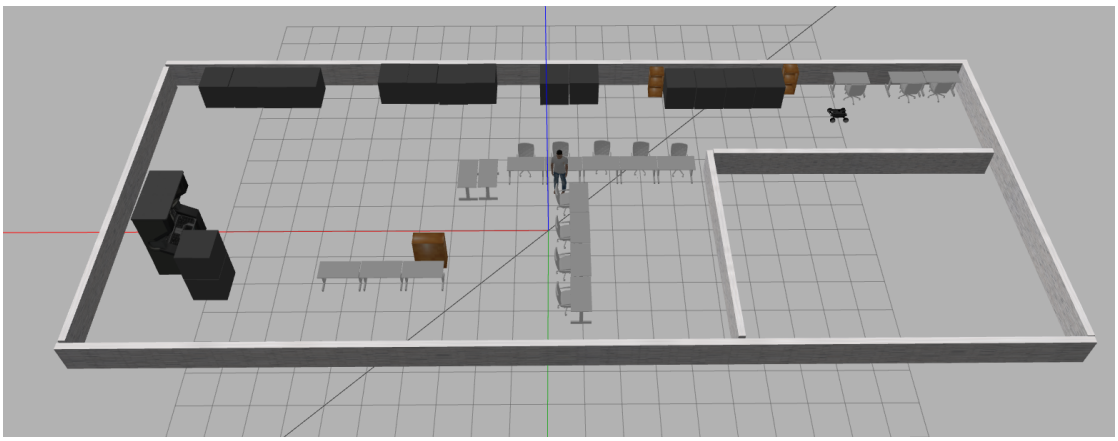


Figure 3.15: World model of CIM4.0 laboratory in Gazebo

Gazebo plugins give URDF models greater functionality, attaching the simulated sensors and actuators to the corresponding links, and can tie in ROS messages and service calls for these sensor output and motor input. It is Gazebo plugin that connects Gazebo and ROS, as shown in Fig. 3.16.

The plugins used for the basic AMR are Laser Plugin and Planar Move Plugin. The Laser Plugin simulates laser range sensor by broadcasting LaserScan message as described in sensor_msgs. Planar Move Plugin written below is a model plugin that allows arbitrary objects (for instance cubes, spheres and cylinders) to be moved along a horizontal plane using a geometry_msgs/Twist message, and publish simulated odometry in the meanwhile. Every cycle, the plugin gives the

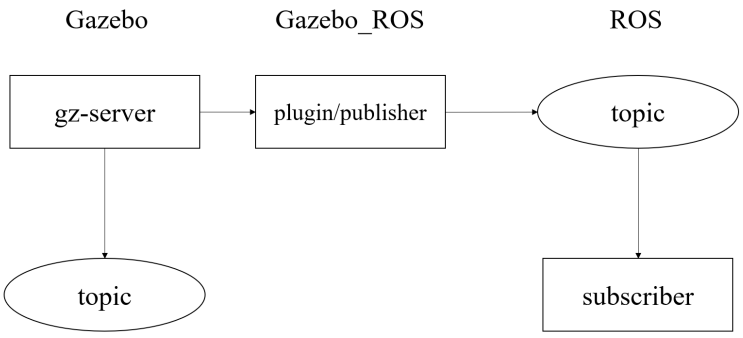
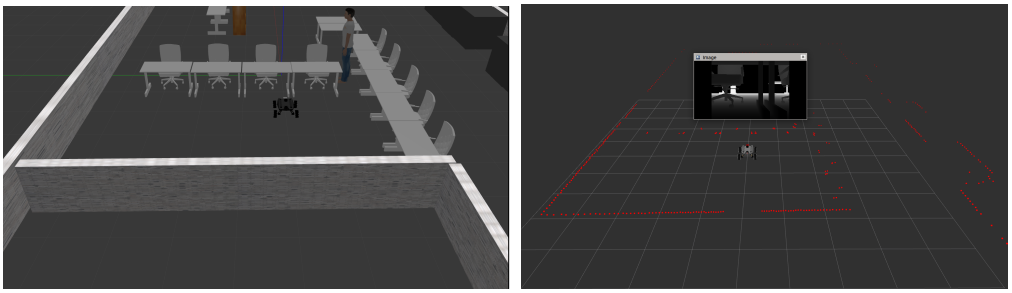


Figure 3.16: Gazebo plugins

object an angular velocity (Z) and a linear motion (XY). Given the fact that the omnidirectional robot we used, Planar Move Plugin is suitable to simulate the movement of our AMR.

```
<gazebo>
  <plugin name="object_controller" filename="
  libgazebo_ros_planar_move.so">
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <odometryRate>20.0</odometryRate>
    <robotBaseFrame>base_footprint</robotBaseFrame>
  </plugin>
</gazebo>
```

The data generated by Gazebo simulated sensors can be visualized by RViz. Fig. 3.17 (a) shows the Gazebo simulation of the AMR in the CIM4.0 laboratory, while (b) presents the visualized information in RViz including the Lidar data (red point clouds) and the depth image.



(a) Simulation in Gazebo (b) Visualized by RViz

Figure 3.17: Visualize Gazebo simulated information by RViz

3.3.4 Navigation stack

When it comes to the AMR based on ROS, the Navigation Stack is one of the most extensively used and mature packages, as shown in Fig 3.18, which includes "map_server", providing the 2D occupancy or cost map generated from the accumulated sensors data; "amcl", Adaptive Monte Carlo Localization (AMCL), localizing the robot in real time through the given map; "move_base", performing a cordinated two-layer planning based on the estimated position, as mentioned in Section 2.1, GPP for finding a path to a chosen goal, while LPP for following the global path and avoiding obstacles simultaneously. After computation, the velocity command is sent to the mobile robot by LPP. Moreover, when the move_base component becomes stuck, it can perform automatic recovery actions including clearing the current cost map and rotating in situ to refresh the map.

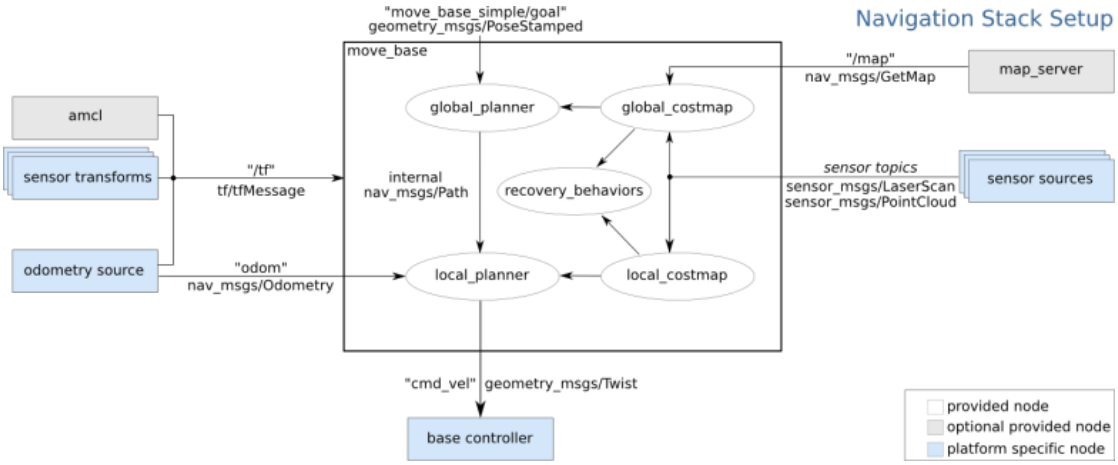


Figure 3.18: Navigation Stack [74]

costmap_2d

The *costmap_2d* package offers a configurable structure that maintains an occupancy grid with information about the accessible path of the robot. Through the *costmap_2d* :: *Costmap2DROS* object, the costmap leverages sensor data and information from the static map to store and update information about obstacles in the world. The *costmap_2d* :: *Costmap2DROS* object provides a purely two dimensional interface to its users, meaning that queries about obstacles can only be made in columns. For instance, the equivalent cell in the *costmap_2d* :: *Costmap2DROS* object's costmap would have the same cost value if a table and a shoe were placed in the same location in the XY plane but had different Z positions. This is intended to assist in planar space planning.

Figure 3.19: Costmap [75]

The red polygon represents the robot's footprint, the blue cells represent obstacles inflated by the robot's inscribed radius, and the obstacles in the costmap are represented by the red cells in the image above. In order for the robot to avoid colliding with other objects, neither its center point nor its footprint should ever cross a blue cell.

The main interface of *costmap_2d* is *costmap_2d :: Costmap2DROS* which maintains much of the ROS related functionality. To keep track of all the layers, it includes a *costmap_2d :: LayeredCostmap*. Using pluginlib, each layer is created and added to the *LayeredCostmap* in the *Costmap2DROS*. The layers themselves may be compiled individually, allowing arbitrary changes to the costmap to be made through the C++ interface. The fundamental data structure for storing and accessing the two dimensional costmap is implemented by the *costmap_2d :: Costmap2D* class.

Odometry

The ability of a mobile robot to locate itself is a prerequisite for navigation and path planning. Wheel odometry is the most widely used and cost-effective localization approach for mobile robots. It's used by almost all wheeled robots. Odometry is the use of motion sensors to calculate the change in the robot's location in relation to a predetermined point. For instance, if a robot is moving straight forward and knows the diameter of its wheels, it may calculate the distance traveled by counting the number of wheel spins, as depicted in Fig. 3.20. Shaft encoders, which generate a fixed number of pulses per revolution, are frequently mounted on the driving wheels of robots. The processor can calculate the distance traveled by counting these pulses. However, due to factors such as a finite wheel encoder resolution or wheel slippage relative to the ground, the accuracy of the resulting position estimation is limited. With increasing path length, wheel odometry becomes more unreliable due to the accumulation of errors.

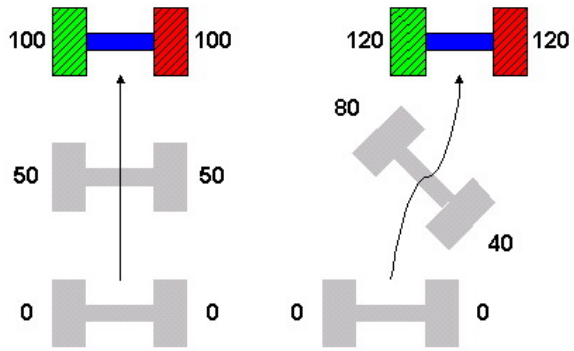


Figure 3.20: Odometry of two differential wheels [76]

Robots may periodically need to use other sensors to precisely determine the robot's position to prevent excessive error buildup. One way providing correction is the AMCL localization.

Adaptive Monte Carlo Localization (AMCL)

The Monte Carlo localization algorithm, which uses the particle filter algorithm, is a probabilistic localization algorithm applied to a two-dimensional occupancy grid map. In known maps, particle swarm is utilized to describe and track the present probable pose of mobile robots. With a modest amount of processing and a small memory footprint, it could estimate global posture.

As illustrated in Fig. 3.21 [77], the Monte Carlo localization procedure is separated into the following four parts.

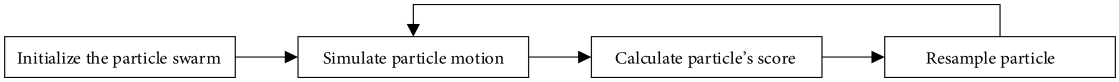


Figure 3.21: Monte Carlo localization procedure [77]

When a laser sensor is used to locate a robot on a 2D grid map, it is fairly simple to calculate the agreement between the laser beams and the occupied grid if the robot pose is known. As a result, the MCL algorithm, which describes the robot's pose with multiple particles, can be applied, as illustrated in Fig. 3.22 [77]. Calculate the particle's weight based on the map's agreement, then find the estimated pose and locate the robot. The MCL algorithm, on the other hand, has some drawbacks: it can't handle the robot kidnapping problem. The localisation will fail if the pose changes are not continuous. Many particles must be added to improve localization accuracy, which results in a slow rate of localization convergence.

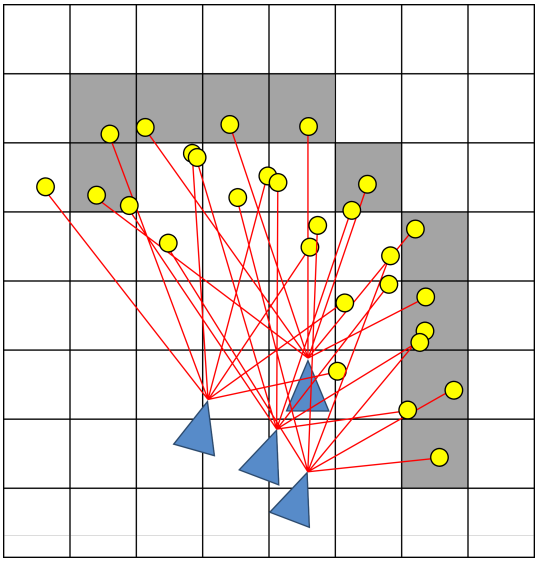


Figure 3.22: Matching particle swarms to grid maps [77]

To tackle the difficulties mentioned, the AMCL algorithm is derived from the MCL algorithm. During resampling, the AMCL method inserts free particles at random. The key idea is to bound the error introduced by the sample-based representation of the particle filter. The real posterior is thought to be represented by a discrete, piecewise constant distribution, such as a discrete density tree or a multidimensional histogram, in order to calculate this bound. For such a representation we can determine the number of samples so that the distance between the Maximum Likelihood Estimate (MLE) based on the samples and the true posterior does not exceed a pre-specified threshold. The number of particles required is proportional to the inverse of this threshold, as is ultimately determined.

nav_core

The package, nav_core, provides common interfaces for navigation specific robot actions.To design actions that can quickly switch their planner, local controller, or recovery behavior for fresh versions adhering to the same interface, this package currently offers the BaseGlobalPlanner, BaseLocalPlanner, and RecoveryBehavior interfaces. All planners and recovery behaviors that expected to be used as plugins in the move_base node must adhere to these interfaces, as depicted in Fig. 3.23.

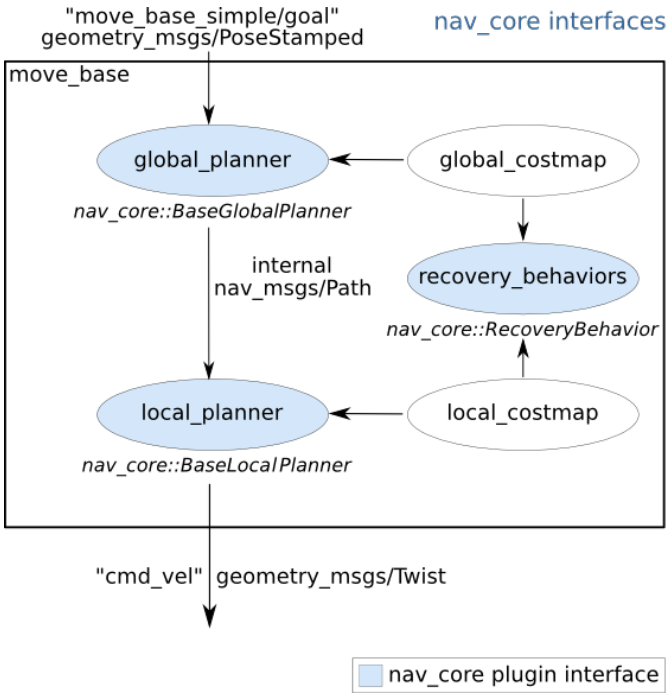


Figure 3.23: nav_core interfaces [78]

For global planners used in navigation, the nav_core::BaseLocalPlanner provides

an interface. This interface must be followed by any global planners created as move base node plugins. The following global planners currently make use of the nav_ core::BaseLocalPlanner interface:

• navfn: a grid-based global planner that calculates a robot's path using a navigation function. (pluginlib name: "navfn/NavfnROS")

• global_planner: a quick, interpolated global planner created to replace navfn with something more adaptable. (pluginlib name: "global_planner/GlobalPlanner")

• carrot_planner: a straightforward global planner that makes an effort to bring the robot as near as possible to a user-specified objective point, even if that point is inside an obstruction. (pluginlib name: "carrot_planner/CarrotPlanner")

The nav_core::BaseLocalPlanner provides an interface for local planners used in navigation. All local planners written as plugins for the move_base node must adhere to this interface which contains four main virtual functions that we need to realise when rewriting the LPP algorithm.

```
1
2  Public Member Functions
3  virtual bool     computeVelocityCommands (geometry_msgs::Twist &
       cmd_vel)=0
4   /* Given the current position, orientation, and velocity of the
       robot, compute velocity commands to send to the base.*/
5  virtual void     initialize (std::string name, tf::TransformListener *
       tf, costmap_2d::Costmap2DROS *costmap_ros)=0
6   /* Constructs the local planner.*/
7  virtual bool     isGoalReached ()=0
8   /* Check if the goal pose has been achieved by the local planner.*/
9  virtual bool     setPlan (const std::vector< geometry_msgs::
       PoseStamped > &plan)=0
10  /*Set the plan that the local planner is following.*/
11 virtual      ~BaseLocalPlanner ()
12  /* Virtual destructor for the interface.*/
13 Protected Member Functions
14     BaseLocalPlanner ()
```

Current local planners using the nav_core::BaseLocalPlanner interface are:

• base_local_planner: outlines how the Dynamic Window Approach (DWA) and Trajectory Rollout techniques to local control are implemented.

• dwa_local_planner: More flexible y axis variables for holonomic robots are available in the modular DWA implementation than in base local planner's DWA.

• eband_local_planner: Implements the Elastic Band method on the SE2 manifold

• teb_local_planner: Implements the Timed-Elastic-Band method for online trajectory optimization

• mpc_local_planner: Provides several model predictive control approaches embedded in the SE2 manifold

An interface for recovery behaviors used in navigation is provided by nav core::RecoveryBehavior. This interface must be followed by all recovery behaviors created as Move base node plugins. Utilizing the nav core::RecoveryBehavior interface, current recovery behaviors include:

• clear_costmap_recovery: Outside of a user-specified range, a recovery behavior that reverts the costmaps used by move base to the static map

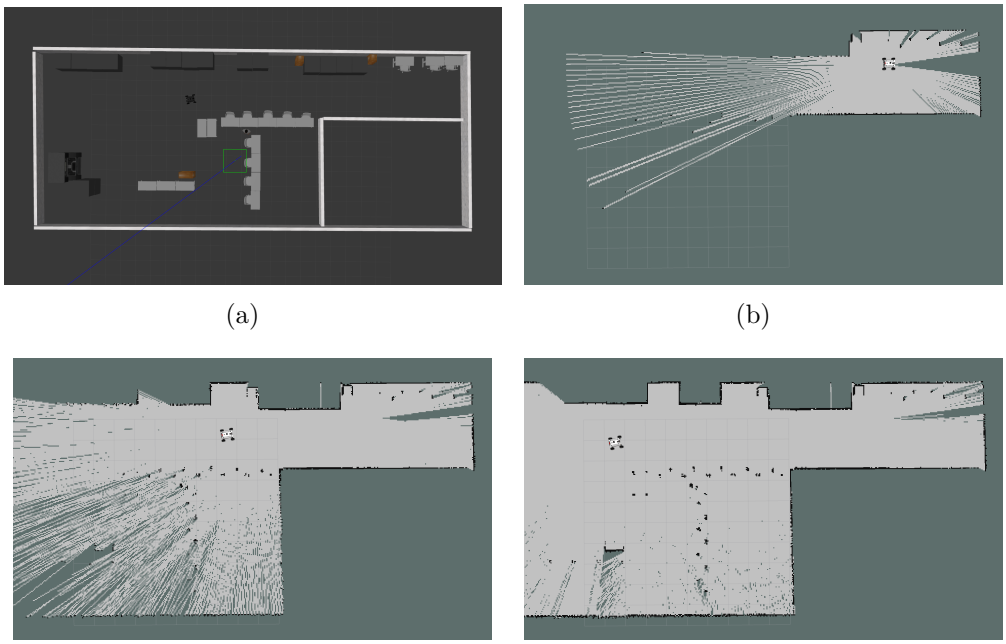
• rotate_recovery: a recovery behavior that rotates the robot 360 degrees in an effort to clear some space.

3.3.5 Simultaneous Localization and Mapping (SLAM)

The Navigation Stack gets costmap from *map_sever*, and the costmap stored in *map_server* is generated by *gmapping*, a ROS package that provides laser-based SLAM.

Many activities envisioned for mobile robots, such as transportation, search and rescue, or automated vacuum cleaning robots, require a map of the environment to be solved efficiently. The availability of a precise map enables the construction of devices that can operate in complex situations only using on-board sensors rather than relying on external reference systems like GPS. The SLAM problem is a term used to describe the problem of building maps while posing uncertainty. A wide range of solutions to this problem can be found in the literature. These methods can be categorized as either filtering or smoothing. The problem is modeled as an on-line state estimation in filtering techniques, with the state of the system consisting of the current robot location and the map. By integrating fresh measurements as they become available, the estimate is augmented and adjusted. Kalman and information filters, particle filters, and information filters are examples of popular approaches. The filtering procedures are sometimes referred as as on-line SLAM methods to emphasize their incremental nature. Smoothing approaches, on the other hand, use all of the measurements to estimate the robot's whole trajectory. These strategies typically rely on least-square error minimization techniques to address the so-called full SLAM problem. To build a 2D grid map, *gmapping* used the Rao-Blackwellized Particle Filter (RBPF) using input from both a laser sensor and a robot posture [79].

The process of *gmapping* is reported in Fig. 3.24, where (a) shows the Gazebo simulation; (b), (c), (d) present that the robot accumulates the Lidar information to build the map gradually utilizing the pose provided by the simultaneous localization.



(a) (b)

(c) (d)

Figure 3.24: Process of gmapping

The final map built by gmapping according to the CIM4.0 laboratory model is:

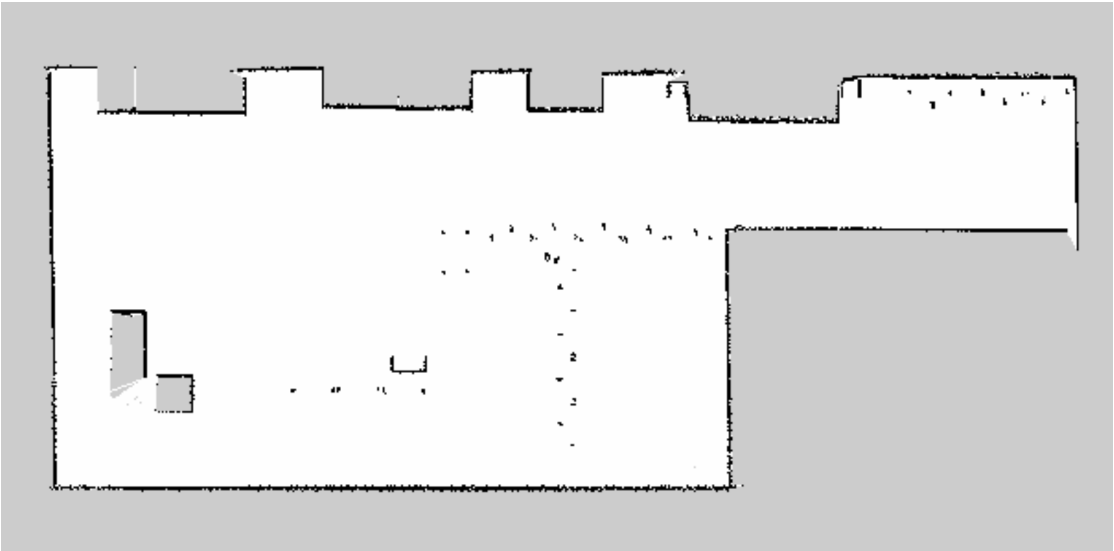


Figure 3.25: Costmap built by gmapping

3.4 Real robot

3.4.1 Mechanical system

The mechanical architecture of the OMR, Scout mini, is mainly composed of three layers: the chassis layer, the control layer and the application layer.

• The Chassis Layer

The movement mechanism and the power supply are both located on this stratum. Along the rotational axis of the Mecanum wheel, a DC brushless servomotor and a harmonic reduction are connected with the wheel. The output power of each DC brushless servomotor is 150 W, and the harmonic reducer has a 50:1 ratio. Four wheels are positioned symmetrically along the longitudinal center axis of the chassis. In the meanwhile, the mobile platform is powered by a 24 V 15 Ah lithium battery.

• The Control Layer

The control system, some on-board sensors, and power converters are all part of this layer. The control system is made up with controller and motor drives. Each brushless servomotor is moved by a DC drive with a built-in PID controller. The PID parameters can be manually adjusted to achieve appropriate dynamic responses referring to the drive's expected input speed and feedback from the encoder installed at the end of servomotor. A FS RC transmitter is also provided for manually controlling the mobile platform with joysticks.

• The Application Layer

This layer allows a variety of extensions to satisfy diverse requirements. Slide rails are reserved for quick building top load, e.g., a light-weight robot manipulator or a fork arm.

3.4.2 Control system

The control system is split into two levels: higher and lower level, in order to perform motion control and autonomous navigation. A Programmable Logic Controller (PLC) and an distributed computer system are used in the lower-level and higher-level controllers, respectively. These two levels communicate through Ethernet. Since if directly connect the drivers and sensors of Scout mini to the higher-level controller, the higher-level controller not only needs to execute the path planning algorithm, but also perform motion control and tedious data processing tasks. Such a structure will inevitably affect the real-time performance and reliability of the system.

Lower-level controller

Therefore, it is better to set a lower-level controller to complete the control of the driver and information collection, which relieves the heavy calculation of the host computer, so that the upper computer only focuses on the execution of the path planning algorithm. The hardware block diagram is shown in Figure 3.26. A unified communication protocol is used to communicate between the two levels to ensure the reliability of data transmission.

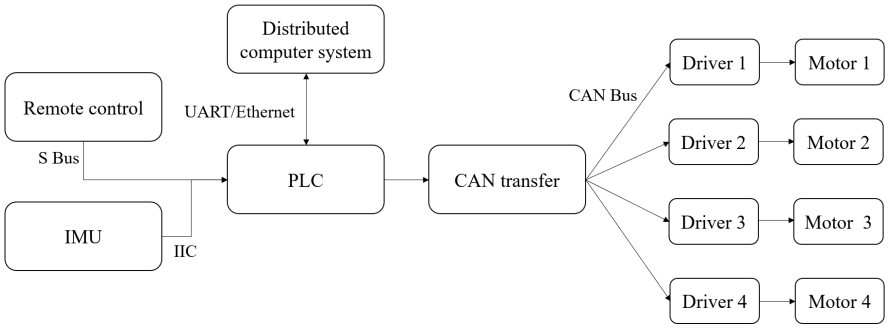


Figure 3.26: Lower-level controller

The main tasks of the lower-level controller, or the motion controller, are to receive instructions (x, y, θ), the velocity along x and y coordinates and the rotational velocity around z coordinate, from the higher-level controller by ROS topic "cml" or remote control. Then, the rotational speed of each wheel is calculated according to kinematic model of OMR, and sent to the drives with PID control. At the same time, the lower-level controller should upload the information of sensors such as IMU data, odometry and temperature of motor. To control the wheel, a voltage signal is provided to each servomotor drive. The mobile robot may move in a variety of motion modes, including longitudinal, lateral, diagonal, and rotational movements, by adjusting the ratios of 4-channel voltage signals. Each encoder mounted at the back of the servomotor sends a signal to both the relevant drive and the PLC at the same time. As a result, the PLC can use odometry to calculate the movement of the mobile chassis, As showing in the block diagram in Fig. 3.27.

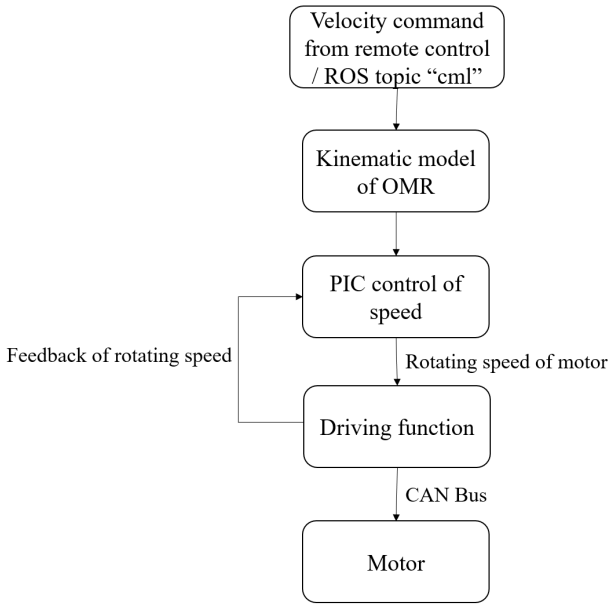


Figure 3.27: Motion control

Higher-level controller

Since two cameras and two Lidars are introduced in the perception system, requiring processing huge amount of data, a distributed architecture is developed to achieve the targets, as shown in Figure 3.28. These computers communicate by ROS messages published on corresponding ROS topics. The autonomous navigation function is based on ROS Navigation Stack.

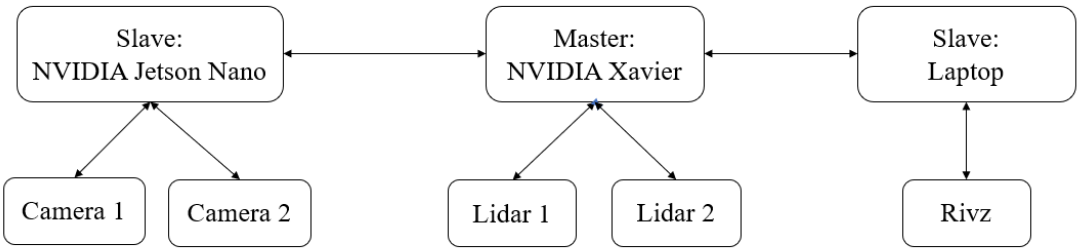


Figure 3.28: Distributed architecture

3.4.3 Microcomputers and Perception sensors

Microcomputers

High-performance AI embedded systems like Jetson Xavier NX (Fig. 3.29) are made for commercial robotics, medical devices, smart cameras, high-resolution sensors, automated optical inspection, smart factories, and other AIoT embedded systems.

Figure 3.29: JETSON XAVIER NX [80]

The technical specifications of the JETSON XAVIER NX are as Table 3.2:

A compact, potent computer called the NVIDIA Jetson Nano (Fig. 3.30) enables programmers to run several neural networks concurrently for tasks like speech processing, object detection, segmentation, and picture classification. All of this is contained in a simple platform that uses as little as 5 watts of power. It's the ideal method for rapidly prototyping AI-based products and delivering them to market.

Figure 3.30: JETSON XAVIER NANO [81]

The technical specifications of the JETSON XAVIER NANO are as Table 3.3:

	Jetson Xavier NX 16GB
AI Performance	21 TOPS
GPU	384-core NVIDIA Volta™ GPU with 48 Tensor Cores
CPU	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6MB L2 + 4MB L3
Memory	16 GB 128-bit LPDDR4x59.7GB/s
Storage	16 GB eMMC 5.1
Power	10 W | 15 W | 20 W
PCIe	1 x1 (PCIe Gen3) + 1 x4 (PCIe Gen4), total 144 GT/s*
CSI Camera	Up to 6 cameras (24 via virtual channels) 14 lanes (3x4 or 6x2) MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps)
Video Encode	2x 4K60 | 4x 4K30 | 10x 1080p60 | 22x 1080p30 (H.265) 2x 4K60 | 4x 4K30 | 10x 1080p60 | 20x 1080p30 (H.264)
Video Decode	2x 8K30 | 6x 4K60 | 12x 4K30 | 22x 1080p60 | 44x 1080p30 (H.265) 2x 4K60 | 6x 4K30 | 10x 1080p60 | 22x 1080p30 (H.264)
Display	2 multi-mode DP 1.4/eDP 1.4/HDMI 2.0
DL Accelerator	2x NVDLA Engines
Vision Accelerator	7-Way VLIW Vision Processor
Networking	10/100/1000 BASE-T Ethernet
Mechanical	69.6 mm x 45 mm 260-pin SO-DIMM connector

Table 3.2: Technical specifications of the JETSON XAVIER NX [80]

GPU	NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores 0.5 TFLOPs (FP16)
CPU	Quad-core ARM® Cortex®-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4 1600MHz - 25.6 GB/s
Storage	16 GB eMMC 5.1 Flash
Video Encode	250 MP/sec 1x 4K @ 30 (HEVC) 2x 1080p @ 60 (HEVC) 4x 1080p @ 30 (HEVC)
Video Decode	500 MP/sec 1x 4K @ 60 (HEVC) 2x 4K @ 30 (HEVC) 4x 1080p @ 60 (HEVC) 8x 1080p @ 30 (HEVC)
Camera	12 lanes (3x4 or 4x2) MIPI CSI-2 DPHY 1.1 (18 Gbps)
Connectivity	Wi-Fi requires external chip 10/100/1000 BASE-T Ethernet
Display	HDMI 2.0 or DP1.2 | eDP 1.4 | DSI (1 x2) 2 simultaneous
UPHY	1 x1/2/4 PCIE, 1x USB 3.0, 3x USB 2.0
I/O	1x SDIO / 2x SPI / 4x I2C / 2x I2S / GPIOs -> I2C, I2S
Size	69.6 mm x 45 mm
Mechanical	260-pin edge connector

Table 3.3: Technical specifications of the JETSON XAVIER NX [81]

Perception sensors

Whether they are localizing or building a map, robots rely on sensors to obtain environmental information. Compared with image sensors, ranging sensors are mostly used for mobile robot navigation because they are not affected by light, have simple processing methods and high data accuracy. Ranging sensors mainly include ultrasonic, infrared and laser sensors. The ultrasonic sensor determines the distance of the object by the time difference of the reflected ultrasonic signal on the surface of the object. It is widely used in robots due to the simple, fast and low price of information processing. However, due to the large measurement blind area and poor directionality, it is often used as auxiliary sensor in practical applications. Infrared sensor is an effective proximity sensor, similar to sonar, it works in the state of transmitting/receiving. It is not affected by visible light and electromagnetic waves, but the color and direction of the object can cause some measurement error, and the distance measurement range is relatively small, generally within 30cm. Lidar is one of the most used sensors in mobile robots. It is a high-precision, high-resolution external sensor based on the Time of Flight (TOF) principle, with a very short sampling period and low measurement Error, the measurement distance is long, so it has become the main distance measurement device for mobile robots.

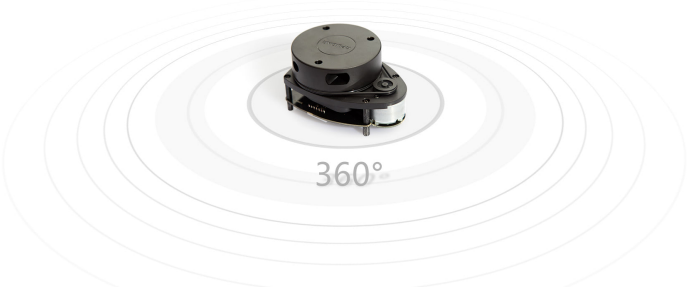


Figure 3.31: RPLIDAR A1 [82]

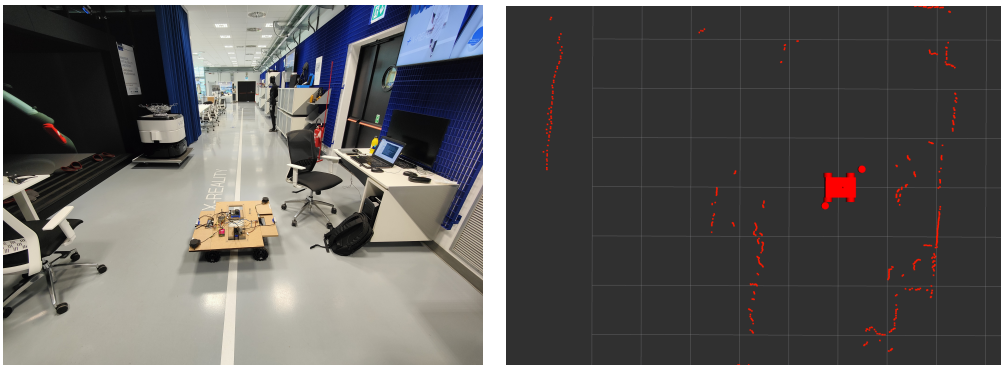
The RPLIDAR A1 (Fig. 3.31) is adopted, the key parameters are attached in table 3.4.

The data generated by RPLIDAR A1 can be visualized by RViz in an analogous way to the simulation, as shown in Fig. 3.32. Since the navigation stack is run by the onboard microcomputer JETSON XAVIER NX without installing Gazebo, the meshes, or the visual elements, can not be rendered, thus only collision elements are presented. Besides, in order to show the basic structure of the AMR the supporting component of sensors, the wooden part shown in Fig. 3.32 (a), is not added to the

Recommended applications	Vacuum robot,Home robot
Measuring Range	0.15m - 12m
Sampling Frequency	8K
Rotational Speed	5.5Hz
Angular Resolution	$\leq 1°$
Dimensions	96.8 x 70.3 x 55mm
System Voltage	5V
System Current	100mA
Power Consumption	0.5W
Output	UART Serial (3.3 voltage level)
Temperature Range	0℃-40℃
Angular Range	360°
Range Resolution	$\leq$ 1% of the range ($\leq$12m) $\leq$ 2% of the range (12m-16m)
Accuracy	1% of the range ($\leq$3m) 2% of the range (3-5m) 2.5% of the range (5-25m)

Table 3.4: Key parameters of RPLIDAR A1 [83]

model, which has no effect on the functionality of the model at all.



(a) Real robot in CIM4.0 laboratory (b) Lidar data visualized by RViz

Figure 3.32: Visualize RPLIDAR A1 information by RViz

But considering RPLIDAR A1 is a 2D Lidar, which returns only the point cloud of surroundings in certain height, it is not enough to detected completed information of environment. Thus, two depth cameras: Intel RealSense Depth Camera D435i (Fig. 3.33), are added to detect objects in 3D view. The datasheet is attached as Table 3.5.

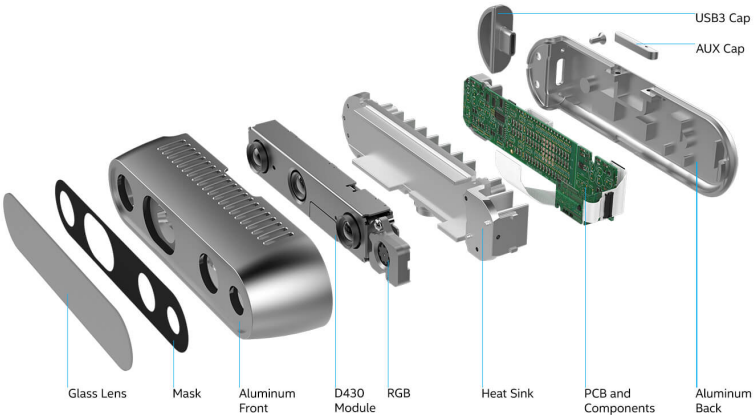
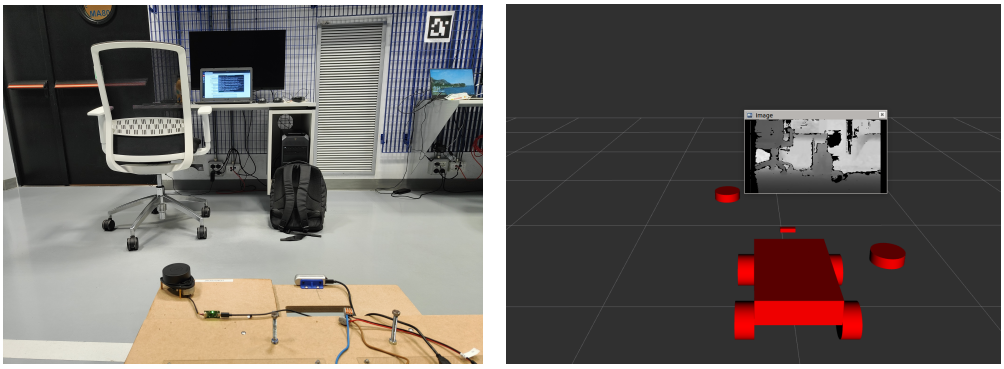


Figure 3.33: Intel RealSense Depth Camera D435i [84]

Features	Use environment: Indoor/Outdoor Image sensor technology: Global Shutter	Ideal range: 0.3 m to 3 m
Depth	Depth technology: Stereoscopic Minimum depth distance (Min-Z) at max resolution: 28 cm Depth Accuracy: $\leq 2\% at 2m$[1]	Depth Field of View (FOV): 87° × 58° Depth output resolution: Up to 1280 × 720 Depth frame rate: Up to 90 fps
RGB	RGB frame resolution: 1920 × 1080 RGB frame rate: 30 fps RGB sensor technology: Rolling Shutter	RGB sensor FOV (H × V): 69° × 42° RGB sensor resolution: 2 MP
Major Components	Camera module: Intel RealSense Module D430 + RGB Camera	Vision processor board: Intel RealSense Vision Processor D4
Physical	Form factor: Camera Peripheral Length × Depth × Height: 90 mm × 25 mm × 25 mm	Connectors: USB-C* 3.1 Gen 1* Mounting mechanism: – One 1/4-20 UNC thread mounting point. – Two M3 thread mounting points.

Table 3.5: Datasheet of Intel RealSense Depth Camera D435i [84]

The depth image taken by Intel RealSense Depth Camera D435i can be also visualized by RViz, as shown in Fig. 3.34.



(a) Real robot in CIM4.0 laboratory (b) Lidar data visualized by RViz

Figure 3.34: Visualize depth image by RViz

Chapter 4

Development of path planning

4.1 GPP using D* Lite

Generally, GPP is implemented providing that full knowledge of the environment is available, i.e., the environment is static. However, to improve the performance of the AMR working in dynamic environment, some algorithms with the ability of re-planning when discovering changes from environment, are proposed to allow the AMR having faster reaction in dynamic environment, combined with the LPP. Considering the APF method chosen for LPP, there is another reason for applying the kind of GPP except for the high dynamic performance. A static global path may result in an unreachable local target point for APF (the local target point will be introduced in Section 4.2.3) when obstacles newly appear on the global path in a dynamic environment.

Incremental search methods like DynamicSWSF-FP [85] are beneficial to handle dynamic path planning, which reuse the knowledge from previous searches considering the changes of environment instead of researching from scratch, saving computation time for re-planning. On the other hand, heuristic search methods, for instance A*, exploit heuristic function that estimates the cost of the cheapest path leading to the goal to guide the search, completing searching with higher efficiency compared with greedy search methods. By taking advantages from both of these two kinds of method, as a combination, incremental heuristic search methods are created. Lifelong Planning A* (LPA*) [86] is one of them, which generalizes both DynamicSWSF-FP and A* and so employs two different strategies to shorten its planning time. When re-planning its paths after encountering previously undetected obstacles, the robot might employ traditional graph-search techniques. However, given the huge terrains that are frequently used, the resulting planning

times might be on the range of minutes, which results in significant idle times. Comparing to this, D* [87] is much faster, a sophisticated algorithm modifying previous search result partially to speedup the process of repeating A*. Although the D* has advantage in planning time, the drawback is complexity of the algorithm. Therefore, the D* Lite is proposed [12], basing on LPA*, and using similar strategy as D*. D* Lite is significantly shorter than D*, compares priorities using only one tie-breaking criterion, making it easier to maintain the priorities, and does not require nested if-statements with intricate conditions that can take up to three lines apiece, making it easier to analyze the program flow. These characteristics also make it simple to extend, such as by using unacceptable heuristics and other tie-breaking criteria to increase efficiency.

4.1.1 D* Lite compared with A* and LPA*

The D*lite method can handle the scenario of a dynamic or an unknown environment far better than other incremental heuristic search algorithms. When an unanticipated new obstacle enters the environment, the robot can instantly update the information of the nodes surrounding it, relist any nodes that have been discontinuous as a result of the emergence of new obstacles, and then swiftly re-plan.

Table 4.1 shows that in a static environment, the A* method has a high path search efficiency, while in a dynamic context, the LPA* approach has a substantially higher path search efficiency. The priority of the extended node is controlled by the size of the f value, and the cost function of the A* algorithm is $f = g+h$ (definitions of these variables are introduced in Section 2.2.1). An incremental variation of the A* algorithm is the LPA* algorithm.The LPA* method maintains two estimates of the initial distance of each node s, $g^*(s)$ and $rhs(s)$, and determines the node status by comparing the values of $g^*(s)$ and $rhs(s)$. The order of growing nodes in the LPA* algorithm is dependant on the size of the two-dimensional key value.

Algorithm	Application of Environment	The Method of Search	Valuation function
A* Algorithm	High search efficiency in static environment, and not suitable for dynamic environment	Heuristic search	$f(s) = g(s) + h(s)$
LPA* algorithm	Both static and dynamic environment are applicable	Incremental search for rapid re-planning in a dynamic environment	Introduce the $rhs(s)$ variable as the minimum cost estimate

Table 4.1: Comparison of A* and LPA* Algorithms [88]

On the principle of the LPA* algorithm, the D* Lite algorithm is deepened. The beginning point is fixed in the version of the LPA* algorithm that searches from one node to the next. The D* Lite algorithm's search direction, however, is the exact opposite of the LPA* algorithm's. The starting node can be altered, and the search proceeds from the target node to the initial node. Furthermore, the definitions of variables like h, g, and rhs are completely the reverse of those of LPA*. D* Lite algorithm is superior to LPA* algorithm, as seen in Table 4.2.

Algorithm	Application of Environment	The Method of Search	Re-planning operations
LPA* algorithm	Both static and dynamic environment are applicable	Forward search from the starting node to the target node	Update the path information of the node from starting point
D* Lite algorithm	Applicable environment is more flexible, and the starting point can change with time	Reverse search from target node to starting node	Update the path information of the current node

Table 4.2: Comparison of LPA* and D* Lite Algorithms [88]

4.1.2 Principle of D* Lite

Both based on the A* algorithm, D* Lite is an extension of the LPA* algorithm. The foundation of the D* Lite algorithm is the incremental search path based on the unknown area as a free space. D* Lite algorithm and LPA* algorithm add rhs variable on the basis of A* algorithm. Every time, the D* Lite method chooses the expansion path with the lowest rhs value as the best option, then iteratively explores and computes the cost estimates of the eight adjacent lattices until it locates the target point. D* Lite maintains an estimate $g(s)$ of the start distance $g^*(s)$ of each vertex s. These values directly correspond to the g values of an A* search. D* also maintains a second kind of estimate of the start distances. The rhs values are one-step lookahead values based on the g values and thus potentially better informed than the g values. The $g(s)$ value will be recalculated after expanding 8 adjacent grids surrounding the grid since the D*lite algorithm's search direction is the reverse of that of the A* algorithm and the LPA* method. Choose the g(s) with the smallest generation value after updating the value of g(s). The formula for calculating rhs(s), which is derived from the g value of its forward point, is as follows:

$$\mathrm{rh}s(s) = \begin{cases} 0, & s = s_{goal} \\ \min\limits_{s' \in \mathrm{Pred}(s)} (g(s') + c(s', s)), & others \end{cases} \tag{4.1}$$

where $c(s', s)$ represents the edge cost from node s' to node s, which is typically reported as 1. The set of predecessors of node $s \in S$ are indicated by$S' \in pred(s)$. Nodes are consistent when$g(s) = rhs(s)$; otherwise, it is non-uniform. $g(s) > rhs(s)$, one of them, is known as local over-consistency, and $g(s) < rhs(s)$, local under-conformity. D* Lite additionally includes $key(s)$ value for comparison while assessing the estimated value of grid points, where $key(s)$ has two values $key(s) = [key(s1); key(s2)]$ and satisfies the following formulas:

$$\begin{aligned} \text{key1(s)} &= \min(\text{g(s)}, \text{rhs(s)}) + \text{h(s)} \\ \text{key2(s)} &= \min(\text{g(s)}, \text{rhs(s)}) \end{aligned} \tag{4.2}$$

The parameters for the priority queue configuration are $k1$ and $k2$. The size of the $key(s)$ value is utilized as the priority for growth, which establishes the order in which the queue's nodes expand. Compare the size of the $K1$ value first, choose the grid with the smallest $k1$ value as the next grid, and then take the size of the $K2$ value into account if the $k1$ values are equal. The object will always travel in the direction of the target point thanks to the $h(s)$ heuristic function, which is the same as the heuristic function in the LPA* algorithm and reflects the estimated value between the current node and the starting point. The speed and precision of the algorithm search are greatly influenced by the value of $h(s)$. The cost $c(s, s')$ from node s to node s' and the value of the heuristic function from node s' to the target point s_{goal} are both included in the search heuristic function of node s. The $h(s)$ calculation formula is:

$$h(s, s_{start}) = \begin{cases} 0, & s = s_{goal} \\ c(s', s) + h(s', s_{goal}), & others \end{cases} \tag{4.3}$$

Fig. 4.1 provides a description of the D* Lite path planning algorithm.

4.2 LPP using DAPF method

Compared with GPP that needs a complete map, LPP just requires the local information around the robot to mainly avoid the dynamic obstacles. Due to the uncertainty of the state of the robot and the environment during the moving process, path planning in a dynamic environment becomes a highly complex problem. Besides, LPP has a higher requirement on the real-time, which determines it should more rely on the information from the real-time sensor instead of the built-in map.

The APF method is often used to solve the LPP problem of moving robots. It usually only needs the position and distance of obstacles without precise environmental models. And it is convenient for lower-level control. Therefore, the APF has good real-time performance, which is suitable for LPP. But the APF

procedure CalculateKey(s)
{01'} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;

procedure Initialize()
{02'} $U = \emptyset$;
{03'} $k_m = 0$;
{04'} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{05'} $rhs(s_{goal}) = 0$;
{06'} U.Insert(s_{goal}, CalculateKey(s_{goal}));

procedure UpdateVertex(u)
{07'} if $(u \neq s_{goal})$ $rhs(u) = \min_{s' \in Succ(u)}(c(u, s') + g(s'))$;
{08'} if $(u \in U)$ U.Remove(u);
{09'} if $(g(u) \neq rhs(u))$ U.Insert$(u$, CalculateKey$(u))$;

procedure ComputeShortestPath()
{10'} while (U.TopKey()$\dot{<}$CalculateKey(s_{start}) OR $rhs(s_{start}) \neq g(s_{start})$)
{11'} $k_{old} =$ U.TopKey();
{12'} $u =$ U.Pop();
{13'} if ($k_{old} \dot{<}$CalculateKey(u))
{14'} U.Insert$(u$, CalculateKey$(u))$;
{15'} else if $(g(u) > rhs(u))$
{16'} $g(u) = rhs(u)$;
{17'} for all $s \in Pred(u)$ UpdateVertex(s);
{18'} else
{19'} $g(u) = \infty$;
{20'} for all $s \in Pred(u) \cup \{u\}$ UpdateVertex(s);

procedure Main()
{21'} $s_{last} = s_{start}$;
{22'} Initialize();
{23'} ComputeShortestPath();
{24'} while $(s_{start} \neq s_{goal})$
{25'} /* if $(g(s_{start}) = \infty)$ then there is no known path */
{26'} $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
{27'} Move to s_{start};
{28'} Scan graph for changed edge costs;
{29'} if any edge costs changed
{30'} $k_m = k_m + h(s_{last}, s_{start})$;
{31'} $s_{last} = s_{start}$;
{32'} for all directed edges (u, v) with changed edge costs
{33'} Update the edge cost $c(u, v)$;
{34'} UpdateVertex(u);
{35'} ComputeShortestPath();

Figure 4.1: D* Lite path planning algorithm [12]

has the problem of local minimum and unreachable targets. What is more, the APF does not effectively use the obstacle information to generate an optimal path. How to solve the shortcomings of APF has become a hot spot in the research field. In response to the problems above, this thesis proposes an improved Dynamic Artificial Potential Field (DAPF) method, which uses points on the global path as additional local targets and introduces the potential field function generated from

the relative velocities. This method not only plans an optimal path by taking the global path into consideration, but can also avoid moving obstacles.

4.2.1 Classical Artificial Potential Field method

The APF method is derived from the idea of field theory in physics and is first applied to the path planning of robotic arms. The core idea is to assume that there is a virtual attractive field around the target point and a repulsive field around obstacles, as depicted in Fig. 4.2.

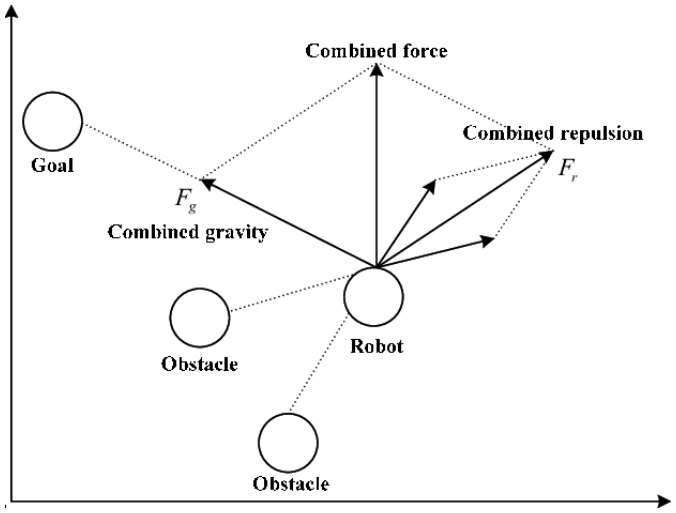


Figure 4.2: FBD of robot in artificial potential field [89]

Under the driving force of the resultant field, the robot will finally reach the goal. The robot is simplified as a point moving in 2D space. The moving direction of the robot at any position $X(X = [xy]^T)$ in the motion space is decided by the resultant field of the attractive field of target and repulsive field of obstacles.

the gravitational potential function is:

$$U_{att}(X) = \frac{1}{2}k\rho^2(X_R, X_G) \tag{4.4}$$

where, k is the attractive gain factor, $\rho(X_R, X_G) = ||X_G - X_R||$, is the distance between robot and target. correspondingly, the attractive function is the negative gradient of the attractive field:

$$F_{att}(X) = -\nabla U_{att}(X) = k(X_G - X_R) \tag{4.5}$$

the repulsive field is:

$$U_{rep}(X_i) = \begin{cases} \frac{1}{2}\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2 & \rho(X_R,X_i) \leq \rho_0 \\ 0 & \rho(X_R,X_i) > \rho_0 \end{cases} \tag{4.6}$$

where, η_x is the repulsive gain factor, $\rho(X_R, X_i)$ is the distance between robot and obstacle, ρ_0 is the obstacle influence distance, which is connected with the dimension of the robot. the repulsive force is:

$$F_{rep}(X_i) = -\nabla U_{rep}(X_i) = \begin{cases} \eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})\frac{e_{iR}}{\rho^2(X_R,X_i)} & \rho(X_R,X_i) \leq \rho_0 \\ 0 & \rho(X_R,X_i) > \rho_0 \end{cases} \tag{4.7}$$

the resultant force exerted on the robot is:

$$F = F_{att}(X) + \sum_{i=1}^{i=n} F_{rep}(X_i) \tag{4.8}$$

4.2.2 Improved Artificial Potential Field method

If the target is too close to the obstacles, the attractive force generated by the target may be smaller than the repulsive force generated by the obstacles, which causes the problem of the unreachable target. To solve this problem, the most commonly used improvement is to add the relative position of the robot and target to the repulsive function, so that the potential of the target is the global minimum.

The modified repulsive field is:

$$U_{rep}(X_i) = \begin{cases} \frac{1}{2}\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2\rho^2(X_R,X_G) & \rho(X_R,X_i) \leq \rho_0 \\ 0 & \rho(X_R,X_i) > \rho_0 \end{cases} \tag{4.9}$$

where, $\rho(X_R, X_G)$ is the distance between robot and target, when robot reach the target, repulsive field is 0.

4.2.3 Strategy of following global path

The APF method drives the robot according to the combined force of the repulsive forces generated by obstacles and the attractive force generated by the target point. In this way, the path is generated in a certain randomness and may not be the global optimal one, although with good real-time performance. For this reason, in this thesis, an artificial potential field with a local target is proposed.

Set the local target point

In the classical APF method, the attractive force driving the robot is only generated by the final target point, which means the robot will not follow the optimal path supplied by the global path planner but a straight line directing to the final target point. To solve this problem, a local target updated in real-time on the global path is set to keep the robot moving on the optimal path as much as possible.

The moving robot usually uses two coordinate systems: one is a static global coordinate system and the other is a dynamic local coordinate system, whose origin is the center of the robot, and it moves with the movement of the robot.

The transformation between the global and local coordinate system is:

$$\begin{cases} x_i = x_0 + x_i^{'} cos\theta - y_i^{'} sin\theta \\ y_i = y_0 + x_i^{'} sin\theta + y_i^{'} cos\theta \end{cases} \tag{4.10}$$

where, (x_0, y_0) is the robot pose at certain moment, (x_i, y_i) is point of global path presented in global coordinate, $(x_i^{'}, y_i^{'})$ is point of global path presented in local coordinate.

Set a square area within the visible range of the robot as a dynamic window to construct a local coordinate system. Then the intersection of the global path and the dynamic window boundary at the heading side is set as the local target point, which attracts the robot to the global optimal path in real-time, as shown in Fig. 4.3.

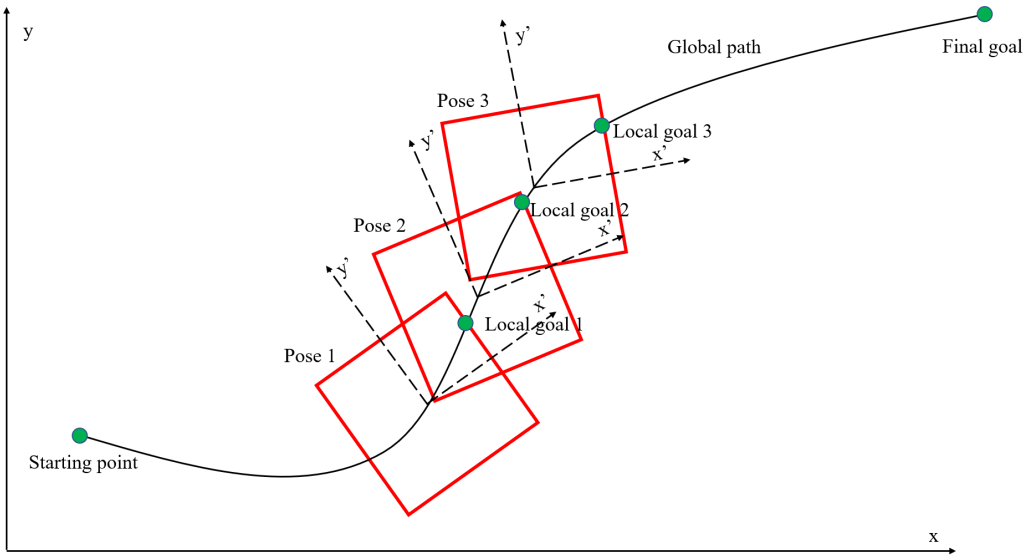


Figure 4.3: Local goal

When the robot is working in a dynamic environment, it is common to see that newly appearing obstacles block the global path, even occupying the position of the local target point, resulting in an unreachable local target. So it is not sufficient that the robot is attracted by local target. The robot should be driven under the combined contribution of global and local targets. The former makes sure the robot can reach the final goal and supply attraction continuously, even if the local target is unreachable, while the latter guarantees the robot will move following the global optimal path.

Improved attractive field

Considering the additional contribution from added local target point, the attractive field of classical APF, (4.4), should be modified correspondingly:

$$U_{att}(X) = p\rho^2(X_R, X_{PG}) + g\rho^2(X_R, X_G) \tag{4.11}$$

where, p, g is the local and global attractive gain factor. The larger the p, the closer the robot moves to the global path, the larger the g, the robot moves closer to the global target. $\rho^2(X_R, X_{PG})$ is the distance between the robot and local target point, while $\rho^2(X_R, X_G)$ is the distance between the robot and global target point.

The attractive force is:

$$F_{att}(X) = -\nabla U_{att}(X) = -2p(X_R - X_{PG}) - 2g(X_R - X_G) \tag{4.12}$$

the FBD of robot in APF with improved attractive field is:

Simulation in MATLAB

To verify the feasibility of this strategy of following the global path, MATLAB is used to simulate the robot in APF with the improved attractive field. Fig. 4.5 depicts a 2D grid map of size 20*10 without obstacle. 4.5. The blue polyline represents a simple global path, and the * symbols on it stand for local goal points. The straight red line stands for the path generated by the APF method under the only attraction from the final goal (20,10). Obviously, the robot under the only attraction from the final goal and without repulsive forces will move along the shortest path, which is the straight line leading to the final goal. In contrast, the robot will exactly follow the global path if only attracted by the local goals. The situation we want to achieve is the robot under the combined attraction of both local and global goals, as depicted by the green curve between the blue and the red line.

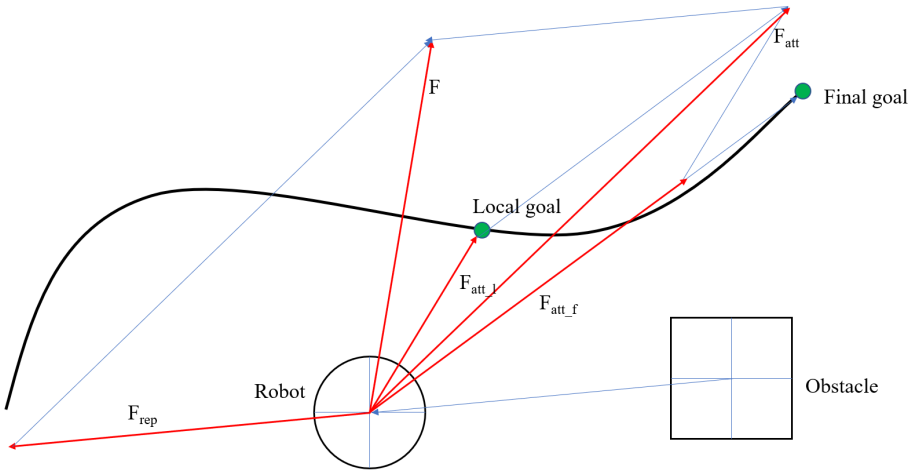


Figure 4.4: FBD of robot in APF with improved attractive field

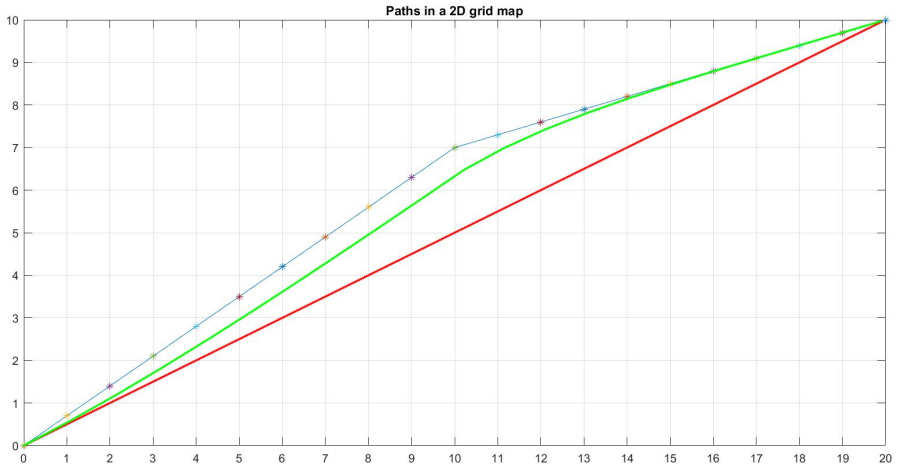


Figure 4.5: Paths in a 2D grip map

Additionally, by modifying the parameters p, g, the local and global attractive gain factors, the consistency of the global path and real path can be adjusted.

4.2.4 Dynamic Artificial Potential Field (DAPF)

The artificial potential field method detects the position of obstacles around the robot in real-time and calculates the resultant force of attractive and repulsive forces which drives the robot. Therefore, the APF is not only convenient for low-level control but also as a kind of path planning with better performance in an unknown environment. In order to overcome the shortcoming of the unreachable target of the classical APF, the improved repulsive field of (4.9) is usually adopted, and the repulsive force can be obtained by taking a negative gradient:

$$\begin{aligned} F_{rep}(X_i) = &- \nabla U_{rep}(X_i) = \eta_x(\frac{1}{\rho(X_R, X_i)} - \frac{1}{\rho_0})\frac{\rho^2(X_R, X_G)}{\rho^2(X_R, X_i)}e_{iR} \\ &+ \eta_x(\frac{1}{\rho(X_R, X_i)} - \frac{1}{\rho_0})^2(X_G - X_R) \qquad \rho(X_R, X_i) \leq \rho_0 \end{aligned} \tag{4.13}$$

Obviously, there are two components of the repulsive force: one from the obstacle points to the robot and another from the robot points to the target. The FBD of the robot in the APF with improved attractive and repulsive field is shown as:

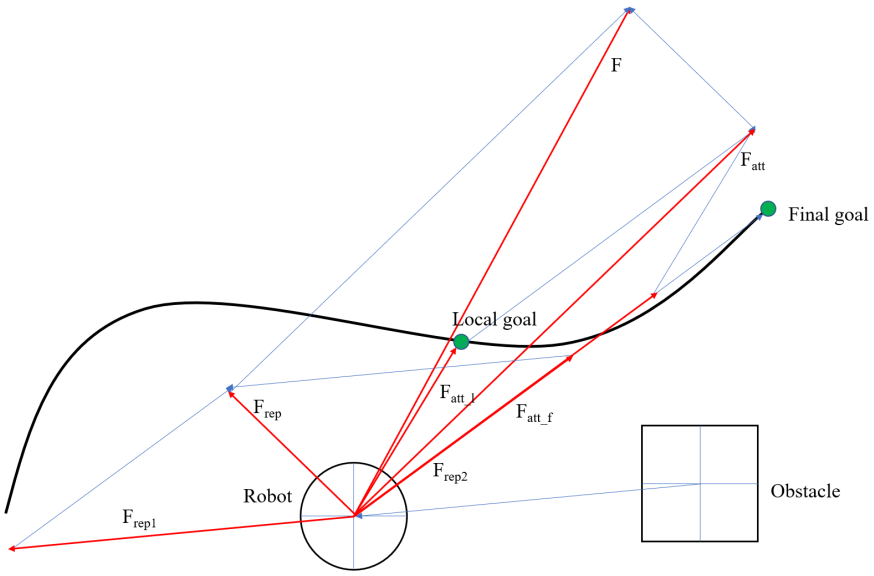


Figure 4.6: The FBD of the robot in the APF with improved attractive and repulsive field

The improved repulsive field solves the problem of unreachable target, but it only takes the relative position between the robot and obstacles without velocity

information. Adding velocity information about the robot and obstacles to the repulsive field can solve the problem of avoiding moving obstacles.

Dynamic repulsive field

Since the local target has been introduced, the attractive component of the repulsive field should be modified as the resultant attractive component of the attractive forces generated by the global and local targets. the modified repulsive field is:

$$U_{rep}(X_i) = \begin{cases} \frac{1}{2}\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2[\frac{p}{p+g}\rho^2(X_R,X_{PG}) \\ +\frac{g}{p+g}\rho^2(X_R,X_G)] & \rho(X_R,X_i) \le \rho_0 \\ 0 & \rho(X_R,X_i) > \rho_0 \end{cases} \tag{4.14}$$

correspondingly, the repulsive force is:

$$\begin{aligned} &F_{rep}(X_i) = -\nabla U_{rep}(X_i) = \\ &\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})\frac{\frac{p}{p+g}\rho^2(X_R,X_{PG}) + \frac{g}{p+g}\rho^2(X_R,X_G)}{\rho^2(X_R,X_i)}e_{iR} + \\ &\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2(\frac{p}{p+g}(X_{PG} - X_R) + \frac{g}{p+g}(X_G,X_R)) \qquad \rho(X_R,X_i) \le \rho_0 \end{aligned} \tag{4.15}$$

As mentioned above, to handle moving obstacles, the relative velocity between robot and obstacle is introduced to the repulsive field.

The modified repulsive field is then:

$$U_{rep}(X_i,V_i) = \begin{cases} \frac{1}{2}\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2[\frac{p}{p+g}\rho^2(X_R,X_{PG}) \\ +\frac{g}{p+g}\rho^2(X_R,X_G)] + \eta_V V_{Ri} \qquad \rho(X_R,X_i) \le \rho_0 \quad V_{Ri} > 0 \\ \frac{1}{2}\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2[\frac{p}{p+g}\rho^2(X_R,X_{PG}) \\ +\frac{g}{p+g}\rho^2(X_R,X_G)] \qquad \rho(X_R,X_i) \le \rho_0 \quad V_{Ri} \le 0 \\ 0 \qquad \rho(X_R,X_i) > \rho_0 \end{cases} \tag{4.16}$$

where, $V_{Ri} = (V_R - V_i)^T e_{Ri}$ is the projection of the relative velocity between robot and obstacle on the line connected robot and obstacle. η_V is the relative velocity gain factor. the repulsive force is changed correspondingly:

$$
\begin{aligned}
&F_{rep}(X_i, V_i) = -\nabla U_{rep}(X_i, V_i) = -\nabla_X U_{rep}(X_i, V_i) - \nabla_V U_{rep}(X_i, V_i) = \\
&\eta_x(\frac{1}{\rho(X_R, X_i)} - \frac{1}{\rho_0})\frac{\frac{p}{p+g}\rho^2(X_R, X_{PG}) + \frac{g}{p+g}\rho^2(X_R, X_G)}{\rho^2(X_R, X_i)}e_{iR} + \\
&\eta_x(\frac{1}{\rho(X_R, X_i)} - \frac{1}{\rho_0})^2[\frac{p}{p+g}(X_{PG} - X_R) + \frac{g}{p+g}(X_G, X_R)] + \\
&\eta_V \frac{V_{Ri\perp} e_{Ri\perp}}{\rho(X_R, X_i)} + \eta_V e_{iR} \qquad \rho(X_R, X_i) \leq \rho_0
\end{aligned}
\tag{4.17}
$$

where, $V_{Ri\perp}e_{Ri\perp} = (V_R - V_i) - V_{Ri}e_{Ri}$ is the projection of the relative velocity on the direction perpendicular to the line connected robot and obstacle. There are 5 forces in different directions. The term with factor η_X presents the relative position between robot and obstacle, while the term with η_V presents the relative velocity between robot and obstacle.

Taking the relative velocity into consideration, an extra force component F_{reqv} results from the relative velocity, driving the robot to move in the direction that the projection of the relative velocity on the line connecting robot and obstacle is decreased. As a consequence, the robot can avoid the oncoming obstacles quickly. Fig. 4.7 shows the FBD of robot in the APF with improved attractive and dynamic repulsive field.

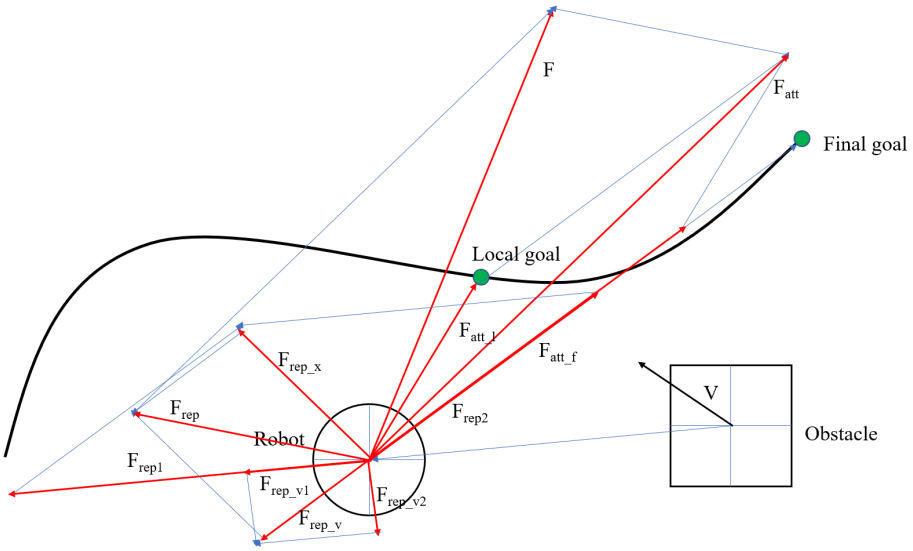


Figure 4.7: The FBD of the robot in the APF with improved attractive and dynamic repulsive field

Simulation in MATLAB

Firstly, the scenario where the robot follows the global path and avoids static obstacles is simulated. The result is shown in Fig. 4.8, where the circle is the symbol of the obstacle and the zigzag curve is the path generated by the DAPF method. The result proves that the robot can track the global path and avoid static obstacles simultaneously. There is no harm in mentioning that the path has some sharp turns, for example, at coordinates $(2.208, 1.167), (4.230, 2.542), (10.180, 6.534)$ etc. That is because, to simplify the simulation, only a limited number of the local goal points are set. When the local goal point is updated to the next one, there will be a sudden change in the attractive force, resulting in the sharp turning of the path. To be more specific, the one with the shortest distance to the robot and closer to the final target node is set as the current local goal. As the robot moves, the current local goal point is updated correspondingly. This problem will be solved when implementing the algorithm in ROS, and it will be introduced in Section 5.2.

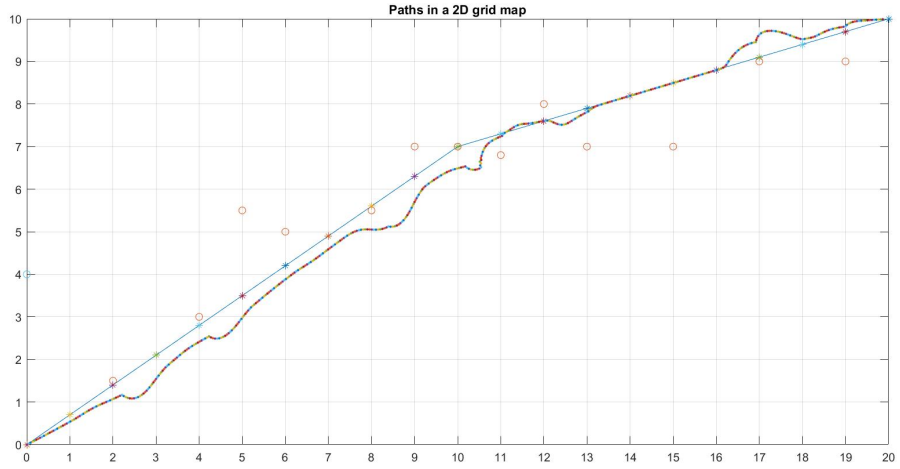


Figure 4.8: Robot follows the global path and avoids static obstacles

Then a moving obstacle is added to verify the improved DAPF.

It can be witnessed from Fig. 4.9, that a moving obstacle starts from coordinate (0,4) and moves horizontally at a constant speed of 0.041/step, then encounters the robot around coordinate (6,3.5), which triggers the mechanism of avoiding moving obstacles. The robot moves back a little bit immediately and turns to the left, avoiding the moving obstacle perfectly. The zoomed view recorded from the sequence steps can be seen in Fig. 4.10.

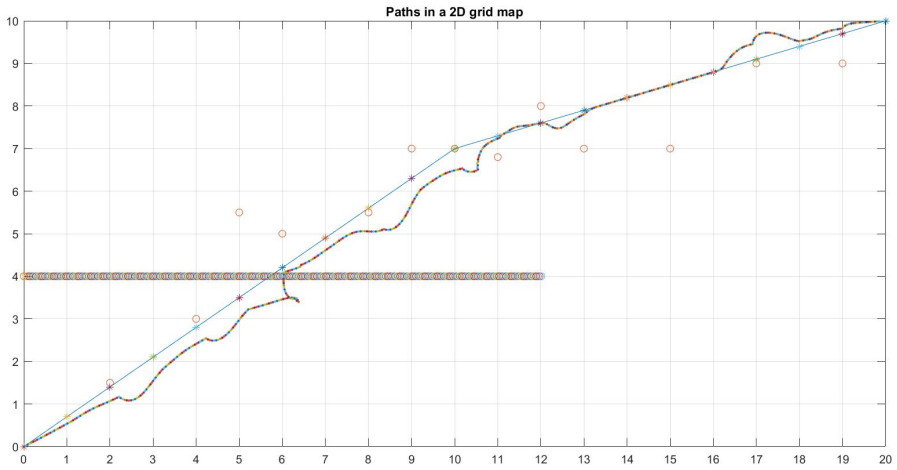


Figure 4.9: Robot follows the global path and avoids static and moving obstacles

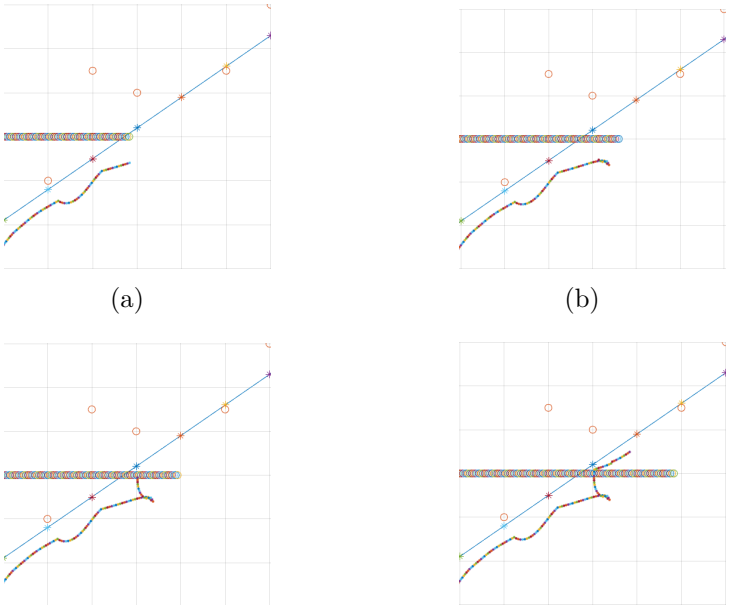


Figure 4.10: Zoomed view of avoiding moving obstacle

Above all, the feasibility of improved DAPF is verified through MATLAB, which only proves that the method can be implemented in principle. The following work is to transplant the algorithm into ROS to do some simulation. Finally, the experiments with real robot will be done. All these contributions will be introduced

in the next chapter.

Chapter 5

Simulations and experimental results

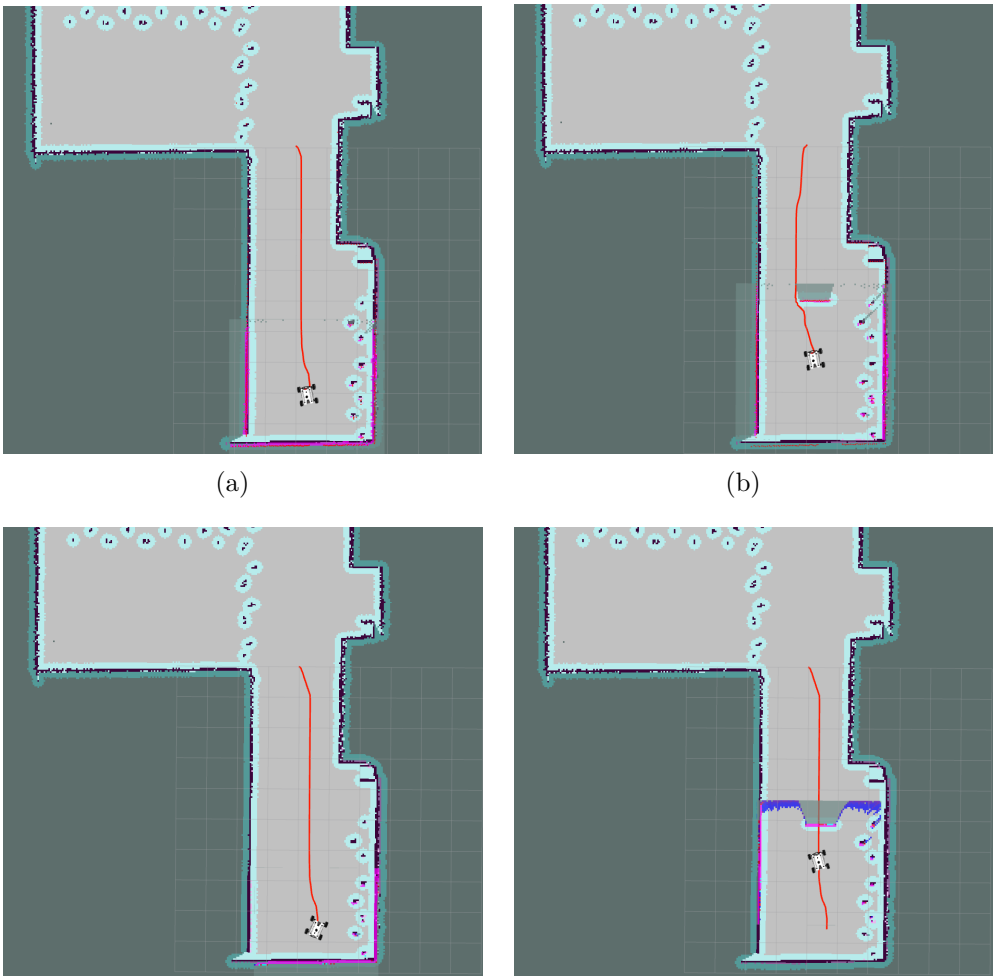
This chapter is mainly devoted to implementing the D* Lite and DAPF algorithms proposed in Section 4.2.4 in the ROS simulation environment as well as on the real robot, verifying the ability to avoid obstacles, and comparing the performance with path planning methods provided by ROS such as A* and DWA.

5.1 Comparison of A* and D* Lite Algorithms

As mentioned in Section 4.1, D* Lite has the ability to re-plan when discovering changes in the environment. To simulate this scenario, the AMR is put at one end of the corridor of the CIM4.0 laboratory, while a goal point is set at the other end. The adopted GPP algorithm generates a global path leading to the goal point. When the robot is moving towards the goal point along the global path, a new obstacle is put in its way. The different performances of the A* and D* Lite algorithms are presented in Fig. 5.1, where (a), (b) show the global path planned by D* Lite before and after the appearance of a new obstacle, while (c), (d) stand for A*. It can be seen that the global path generated by A* is not changed after the appearance of a new obstacle. However, D* Lite re-plans it to avoid the new obstacle.

5.2 Strategy of following global path

As mentioned in Section 4.2.3, a local goal point updated in real-time on the global path is set to keep the robot moving on the optimal path as much as possible. The main loop written by C++ in ROS to realise this strategy is shown as follows:



(a) (b)

(c) (d)

Figure 5.1: Comparison of A* and D* Lite algorithms

```
int i=0;
    for (it = this->global_plan.begin(); it != this->global_plan.end
    (); it++)
    {
        tf2::doTransform(*it, local_goal, goalToLocal_);
        geometry_msgs::Point tmp;
        tmp.x = local_goal.pose.position.x;
        tmp.y = local_goal.pose.position.y;
        d = this->dist(pose.pose.position,tmp);
        if (d > max_local_goal_dist)
            break;
```

```
11            local_global_goal_pub.publish(*it);
12            i++;
13        }
```

The idea is to iterate through the points that make up the global path until finding the intersection point between the global path and the boundary of a circle whose center is the robot's current position and whose radius is defined by d, a predefined parameter called *max_local_goal_dist*. Since the global path is made up of intensive nodes, the local goal point is updated without sudden change, which guarantees a smooth local path generated by DAPF.

5.3 Estimation of the position and velocity of moving obstacles

5.3.1 Build the local cost map

Differing from the global costmap, the size of the local costmap is reduced within the detection range of the on-board sensors, with a higher updating frequency. Once a dynamic window is defined in the local coordinates, a same-sized local cost map moving with the dynamic window is initialized. The Lidar detects surroundings, returning the point cloud of obstacles that the laser beam first reaches within the boundary of local cost map. The point cloud includes information about the distance and angle positions of the obstacles with respect to the robot. Then the distance in meters is transformed to pixels to update the occupancy of the 2D grid map according to the index of pixels (purple points in Figure 5.2) to mark the obstacles. Meanwhile, the transformation among three different coordinates, laser scan frame, local frame, and global frame, is handled by ROS tf.

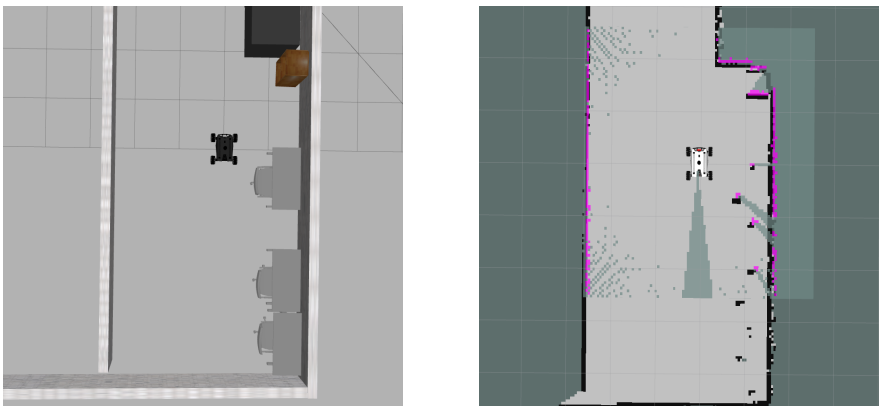


Figure 5.2: Build the local cost map

5.3.2 Add moving obstacle

In Gazebo, most of the objects in the Gazebo library are static. Although the human model of the Gazebo "actor" (animated model) can move, it is without collision property, which means all sensors in Gazebo, for example, Lidar and camera, can not detect it. And the robot will go through the actor without collision. Therefore, it is necessary to write a plugin to define a moving obstacle in the simulation that can be recognized by Gazebo simulated sensors. The Gazebo plugin is shown bellow, which defines the key frames of movement. Given the repeatability of the frames, only partial frames are listed.

```
namespace gazebo
{
  class AnimatedBox : public ModelPlugin
  {
    public: void Load(physics::ModelPtr _parent, sdf::ElementPtr /*
    _sdf*/)
    {
        this->model = _parent;
        gazebo::common::PoseAnimationPtr anim(new gazebo::common::
    PoseAnimation("test", 110, true));
        gazebo::common::PoseKeyFrame *key;

        key = anim->CreateKeyFrame(0);
        key->Translation(ignition::math::Vector3d(1.5, -4, 0));
        key->Rotation(ignition::math::Quaterniond(0, 0, 0));

        key = anim->CreateKeyFrame(5.0);
        key->Translation(ignition::math::Vector3d(1, -4, 0));
        key->Rotation(ignition::math::Quaterniond(0, 0, 0));

        key = anim->CreateKeyFrame(10.0);
        key->Translation(ignition::math::Vector3d(0.5, -4, 0));
        key->Rotation(ignition::math::Quaterniond(0, 0, 0));
          ...

        _parent->SetAnimation(anim);
    }
    private: physics::ModelPtr model;
    private: event::ConnectionPtr updateConnection;
  };
  GZ_REGISTER_MODEL_PLUGIN(AnimatedBox)
}
```

Then a box model is added in the virtual CIM4.0 laboratory world, whose movement is defined by the AnimatedBox plugin above. The piece of code can be

seen as:

```
<model name="box">
      <pose>2.5 0 0 0 0 0</pose>
      <link name="link">
        <collision name="collision">
          <geometry>
            <box>
              <size>0.5 0.5 0.5</size>
            </box>
          </geometry>
        </collision>

        <visual name="visual">
          <geometry>
            <box>
              <size>0.5 0.5 0.5</size>
            </box>
          </geometry>
        </visual>
      </link>

      <plugin name="push_animate" filename="libanimated_box.so"/>
    </model>
```

Above all, the moving obstacle can be simulated in Gazebo. It moves back and forth in the corridor depicted as Fig. 5.3.

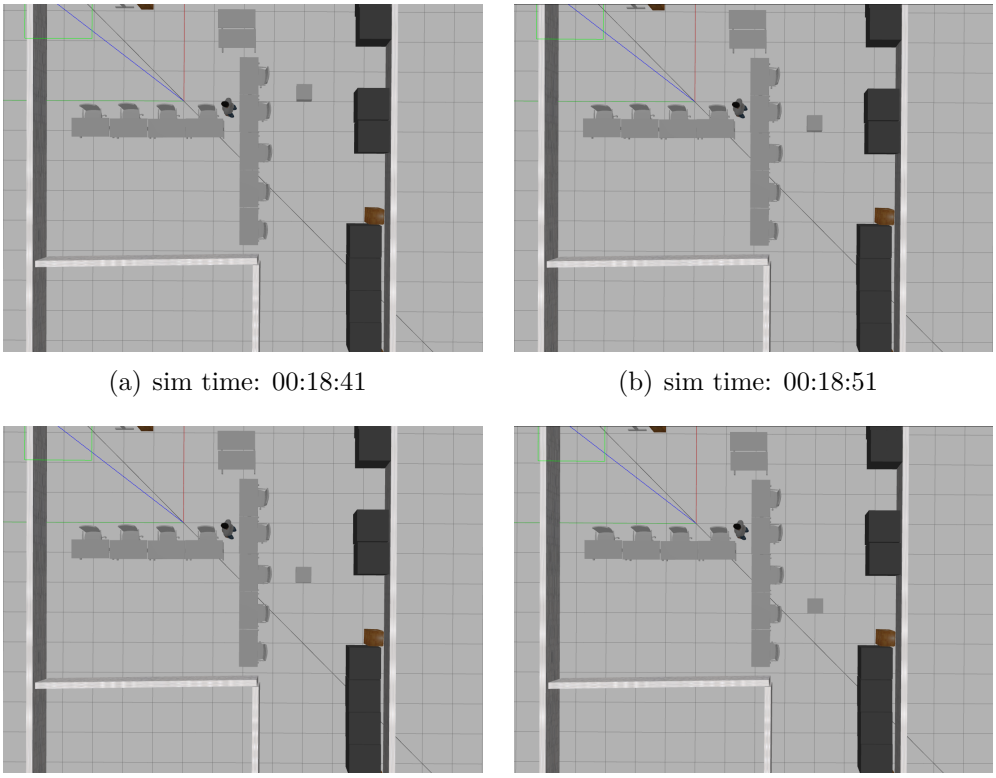
After adding the moving obstacle, the next step is to detect it. The detection of moving objects can be done by the following techniques.

5.3.3 Background Subtraction

The process of creating a foreground mask, specifically, a binary image comprising the pixels that correspond to moving objects in the scene by utilizing static cameras is known as Background Subtraction (BS). It determines the foreground mask by subtracting the current frame from a background model that contains the static portion of the image or, more generally, everything that may be regarded as background given the features of the observed scene, as described in Fig. 5.4.

Background modeling consists of two main steps: background initialization and Background update, which means an initial model of the background is generated in the first phase, and it is updated in the second step to account for potential changes to the scene.

For the application of this thesis, the local cost map is used as the input of this BS technique to create a foreground mask with moving obstacles. Despite the



(a) sim time: 00:18:41 (b) sim time: 00:18:51

(c) sim time: 00:19:01 (d) sim time: 00:19:11

Figure 5.3: Simulated moving obstacle in Gazebo

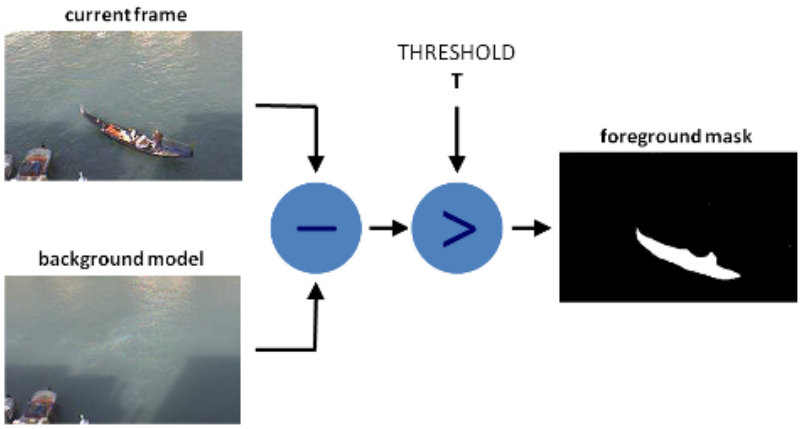


Figure 5.4: Background Subtraction scheme [90]

fact that the moving object's pixels are included in the foreground mask, they are discrete points, which cannot be treated as obstacles. Hence, blob detection is used

to organize these discrete pixels.

5.3.4 Blob detection

A blob is a collection of related pixels in an image that have some properties in common (e.g., grayscale value). For example, blob detection is performed to recognize and indicate the dark connected regions in Fig. 5.5 as blobs.

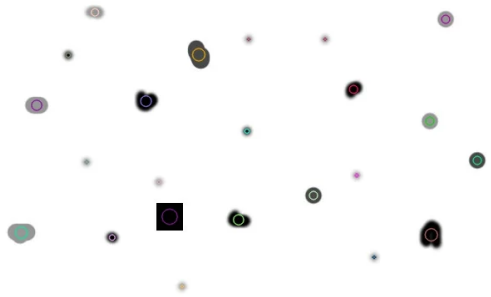


Figure 5.5: Blob detection [91]

Blobs can be easily found and filtered using OpenCV's SimpleBlobDetector, based on steps outlined below:

1. Thresholding: several binary images are converted from the input image by thresholding, with a series of thresholds starting from the minimum one and increasing by certain step length until reaching the maximum value.

2. Grouping: Connected white pixels are grouped together in each binary picture, which is defined as binary blobs.

3. Merging: The centers of the binary blobs in the binary images are calculated. And blobs with a distance between centers smaller than $minDistBetweenBlobs$ are merged.

4. Center and Radius Calculation: The resulting amalgamated blobs' centers and radius are calculated.

Furthermore, blobs can be filtered using SimpleBlobDetector parameters, as shown in Fig. 5.6. Generally, it can be filtered:

By color: filtering by color is validated by setting $filterByColor = 1$. Set blobColor = 0 to select darker blobs, and blobColor = 255 for lighter blobs.

By Size: filtering by color is validated by setting the parameters $filterByArea = 1$, and appropriate values for $minArea$ and $maxArea$, e.g., $minArea = 100$ will filter out all the blobs that have less then 100 pixels.

By Shape: shape has three different parameters:

1. Circularity : This gauges how closely the blob resembles a circle. For instance, a regular hexagon has higher circularity than a square. To filter by circularity, set $filterByCircularity = 1$, as well as $minCircularity$ and $maxCircularity$. In detail, Circularity is defined as $\frac{4*\pi*Area}{perimeter^2}$, which infers that a circle has a circularity of 1, circularity of a square is 0.785, and so on.

2. Convexity : Convexity is defined as the $Areaoftheblob/Areaofit'sconvexhull$. Convex hull of a shape is the tightest convex shape that completely encloses the shape. To filter by convexity, set $filterByConvexity = 1$, followed by setting $0 \leq minConvexity \leq 1$ and $maxConvexity \leq 1$.

3. Inertia Ratio : This measures how elongated a shape is. E.g., for a circle, this value is 1, for an ellipse it is between 0 and 1, and for a line it is 0. To filter by inertia ratio, set $filterByInertia = 1$, and set $0 \leq minInertiaRatio \leq 1$ and maxInertiaRatio ≤ 1 appropriately.

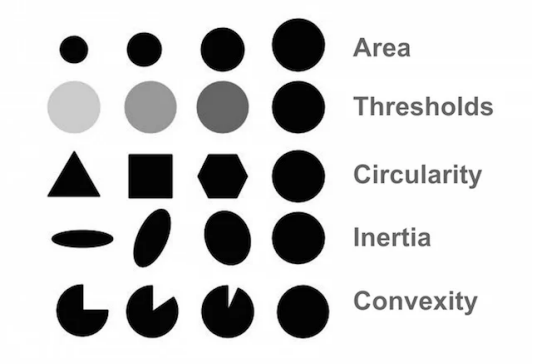


Figure 5.6: Parameters of blob detection

Apply the blob detection to the front ground mask of the local cost map generated by BS. In order to return the contour of the moving obstacle as shown in Fig. 5.7 (green part):

5.3.5 Tracking

As the blob detector returns the blobs with centers, the Kalman filter is used to estimate the velocities of centers, i.e., the velocities of moving obstacles. By estimating a joint probability distribution over the variables for each time frame, Kalman filtering, also known as Linear Quadratic Estimation (LQE), is an algorithm that uses a series of measurements observed over time, including statistical noise and other inaccuracies, to produce estimates of unknown variables that are typically more accurate than those based on a single measurement alone. The algorithm works through a two-phase process. The Kalman filter generates estimates of the current state variables and their uncertainty for the prediction phase. These estimates are

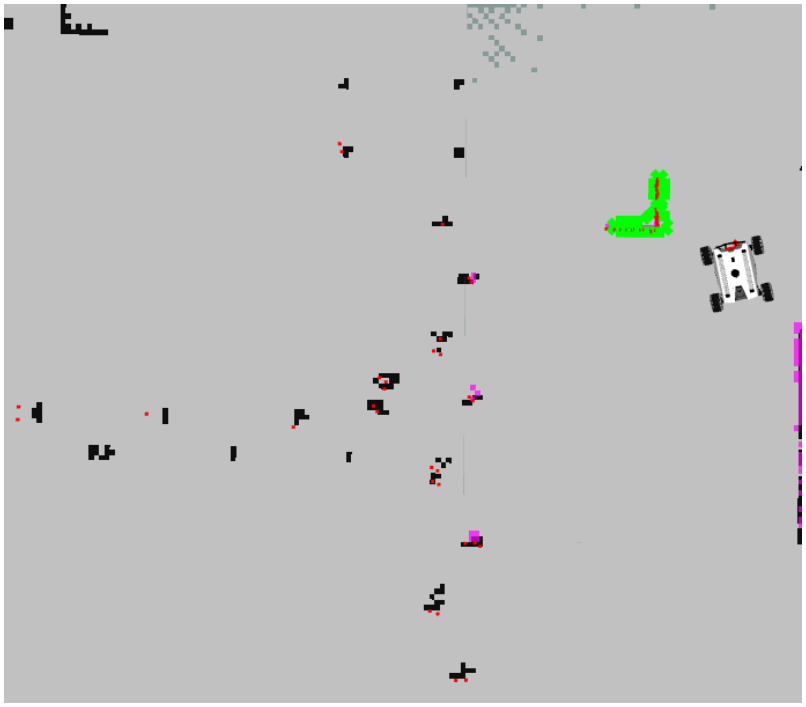


Figure 5.7: Contour of moving obstacle detected by blob detector

updated using a weighted average, with more weight given to estimations with more certainty, after the result of the next measurement—inevitably tainted with some mistake, including random noise—is detected. The algorithm repeats itself. It is capable of operating in real time with just the current input measurements, the previously determined state, and its uncertainty matrix; no extra historical data is needed.

The tracking of a moving obstacle is shown in Fig. 5.9, where the red arrow represents the velocity. Above all, the position and velocity of the moving obstacle are obtained.

5.4 Simulations

5.4.1 Simulation of avoiding static obstacles

In order to simulate the scenario of the AMR avoids the static obstacles that newly appear in the environment, four obstacles with different shapes, two boxes, one cylinder, and a sphere, are added randomly in the corridor of the CIM4.0 laboratory as shown in Fig. 5.10.

Then set a goal point at the left end of the corridor; the robot moves to it under

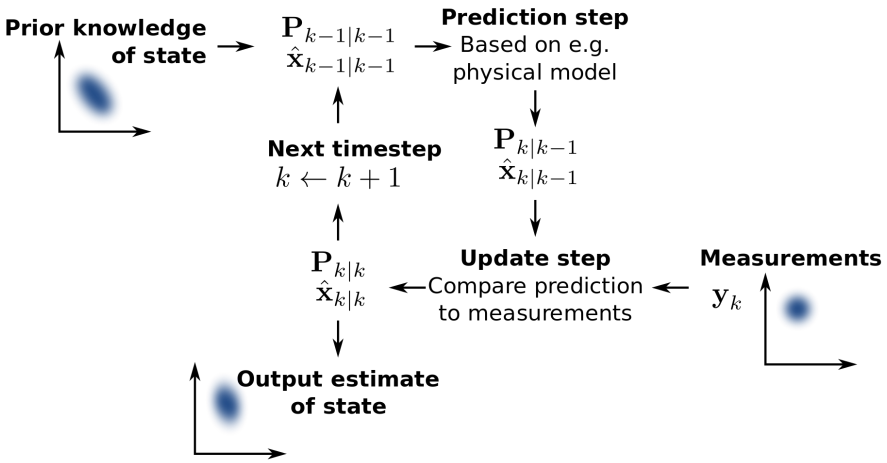


Figure 5.8: Basic concept of Kalman filter

Figure 5.9: Tracking of the moving obstacle

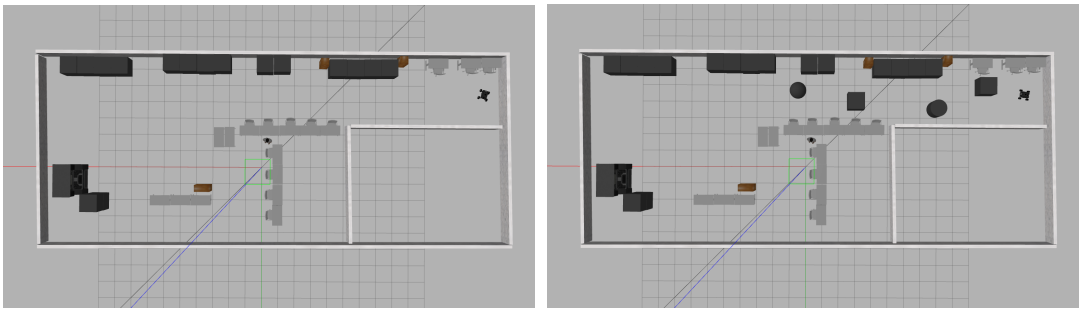
the contributed effort of global and local planning. The autonomous navigation can be presented as Fig. 5.11, from which the capability of path planning methods proposed by this thesis to avoid newly appearing obstacles is proved.

Additionally, by setting the attractive gain factor k and repulsive gain factor η_x properly, the AMR can pass narrow space as reported in Fig. 5.12.

5.4.2 Avoid moving obstacle

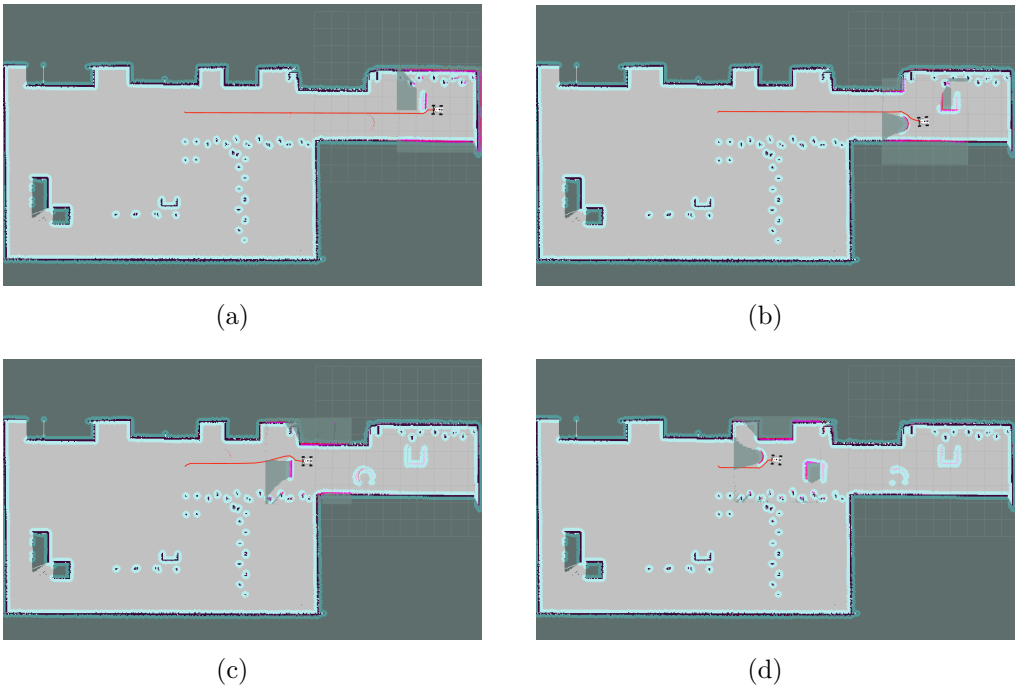
When there is a moving obstacle, the improved DAPF will detect the position and velocity of it, and take this information into consideration to update the repulsive force. As a result, the robot immediately moves to the side and avoids the moving obstacle successfully. The process is shown in Fig. 5.13.

Above all, the functionality of the proposed path planning algorithm is verified in



(a) Before setting static obstacles (b) After setting static obstacles

Figure 5.10: Newly added static obstacles

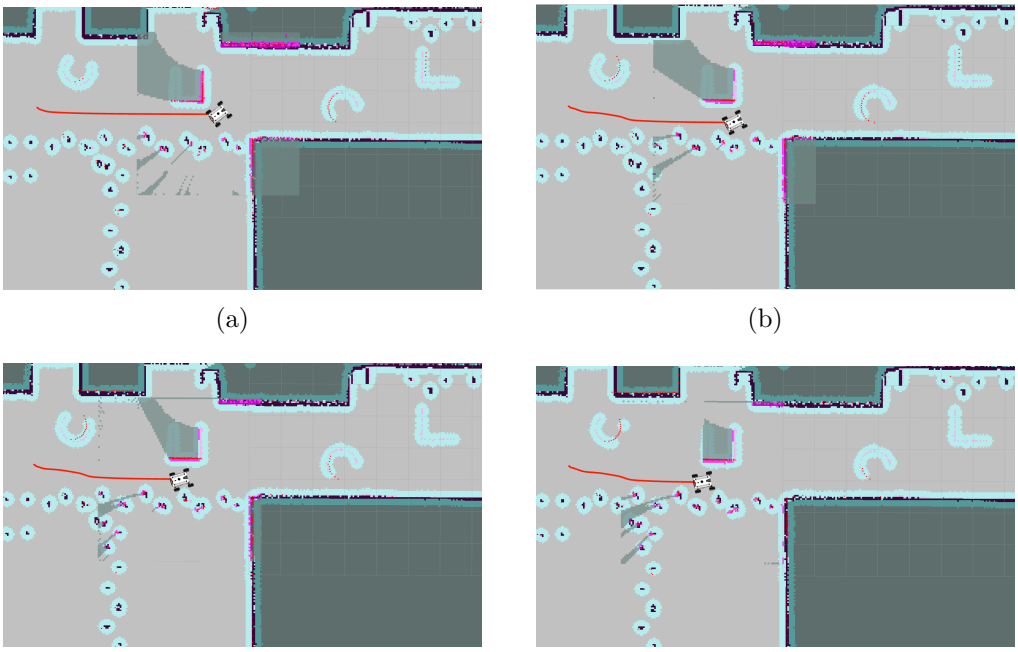


(a) (b)

(c) (d)

Figure 5.11: Simulation of avoiding static obstacles

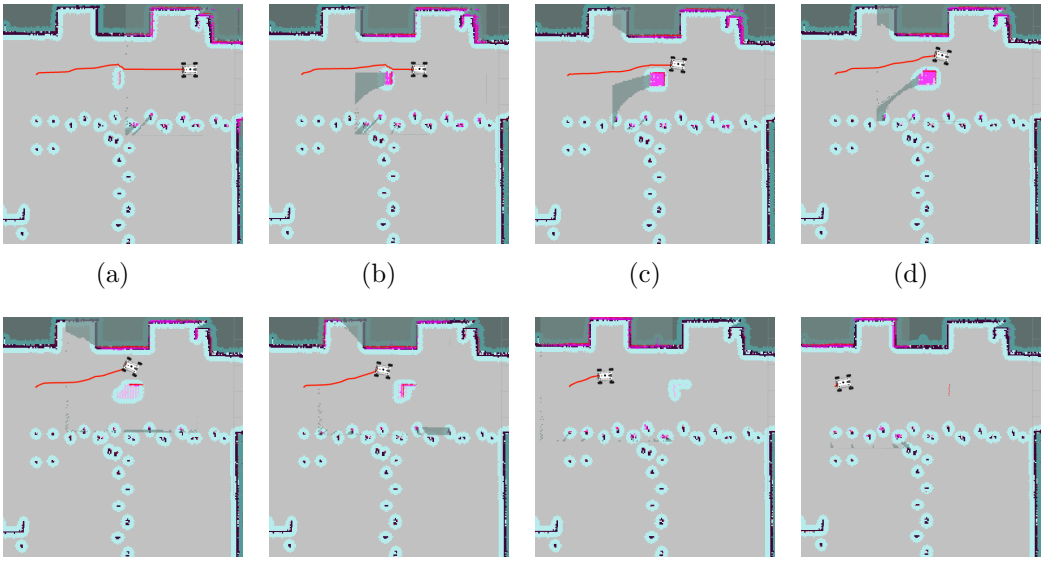
ROS simulation. However, even though Gazebo provides good simulation for users, there are always differences with respect to the real robot system and environment. More specifically, the real on-board sensors are less accurate compared to the simulated ones. Thus, sensor fusion is applied to improve the overall performance of the perception system.



(a) (b)

(c) (d)

Figure 5.12: Passage of narrow space



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.13: Simulation of avoiding moving obstacle

5.5 Sensor fusion

Sensor fusion is the process of merging sensor data or data obtained from other sources so that the final information has less uncertainty than would be feasible if these sources were used separately. It is not required that the data sources for a fusion process come from identical sensors. There are three types of fusion: direct, indirect, and fusion of the outputs of the first two. While indirect fusion relies on data sources including a priori environmental knowledge and human input, direct fusion combines sensor data from a variety of heterogeneous or homogeneous sensors, soft sensors, and sensor data historical values.

5.5.1 Odometry and IMU fusion

As mentioned in Section 3.3.4, the odometry will accumulate errors during navigation. A common way to correct this error is to fuse odometry with IMU data. For our project, odometry is provided by the encoders of Scout mini, and the IMU data comes from Intel Realsence. Firstly, the raw IMU data is filtered by $imu\_filter\_madgwick$, which is used to filter and fuse raw data from IMU devices. It fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into an orientation quaternion, and publishes the fused data on the $imu/data$ topic. The filtered IMU data, along with the odometry data, is then sent to the $ukf\_localization\_node$ to generate filtered odometry. An unscented Kalman filter is implemented by $ukf\_localization\_node$. It uses a set of carefully selected sigma points to project the state through the same motion model that is used in the EKF, and then uses those projected sigma points to recover the state estimate and covariance. This eliminates the use of Jacobian matrices and makes the filter more stable. Lastly, the filtered odometry is used by AMCL as an auxiliary to localize. The process is show in Fig. 5.14:

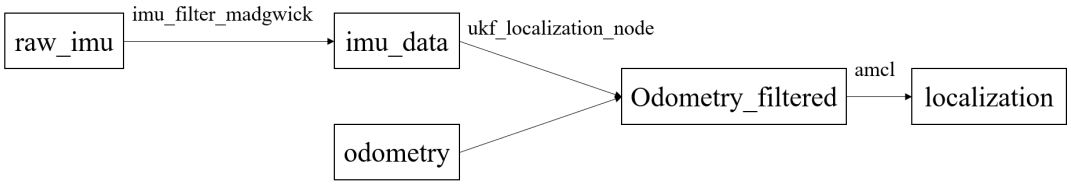


Figure 5.14: Odometry and IMU fusion

5.5.2 Two Lidars fusion

Considering the overall structure of the FIXIT as shown in Fig. 1.1, the extended components on the application layer will block part of the view of the Lidar that is installed at the side hub. Therefore, at least two Lidar should be used to cover a 360° view. Data from these two Lidars is filted first to remove the point clouds from the blocked view, then merged together. Fig. 5.15 (a) depicts the filtering of the front Lidar, with the red point cloud representing the data before filtering and the green point cloud representing the data after filtering; similarly, (b) depicts the filtering of the rear Lidar; by merging these two filtered Lidar data (green point clouds), a full view of the surroundings is obtained, as depicted in (c).

5.5.3 Lidar and camera fusion

As mentioned in Section 3.4.3, a 2D Lidar is not enough to detect all the obstacles at different heights. To solve this problem, data from a camera that has a wider field of view in the vertical direction is fused. The ROS package *depthimage_to_laserscan* generates a *laserscan* message by cutting the depth image provided by the camera at a specific height. From Fig. 5.16, it can be seen that a box below the detection height of the Lidar can be detected by the camera, and the corresponding *laserscan* (the red point cloud) is converted.

5.6 ROS distributed system

Distributed computing is a key component of ROS' design. A well-written node does not presuppose where it will be situated in the network, enabling processing to be moved around at runtime to match the available resources. As described in Section 3.4.2, the higher-level controller of the AMR is designed as a distributed system. To build a ROS distributed system, multiple machines should be connected to the same local network. By adding the IP address and the host name in the */etc/hosts* file to each machine, bi-directional connections among them can be constructed. Then one of them is set as master while the others are slaves. *roscore* is run by the master, and ROS_MASTER_URI is configured on all the machines. With all these steps, a ROS distributed system is validated.

5.7 Experimental results

5.7.1 Experiments of avoiding static obstacles

Firstly, the cost map of the CIM4.0 laboratory is built by gmapping as shown in Fig. 5.17.

Then, similar to the simulation, several boxes are put in the corridor randomly to test the ability of the AMR to to avoid static obstacles that are newly appearing in the environment. As a comparison, both A* with DWA and D* Lite with DAPF are implemented. The results are shown in Fig. 5.18 and Fig. 5.19:

Starting from the same location and navigating to the same goal, both of them are set with a maximum linear velocity of 0.3m/s.

1. It takes 0.35s for D* Lite and DAPF to reach the goal point, while it takes 50s for A* and DWA to reach the last obstacle and stop there without accomplishing the task.

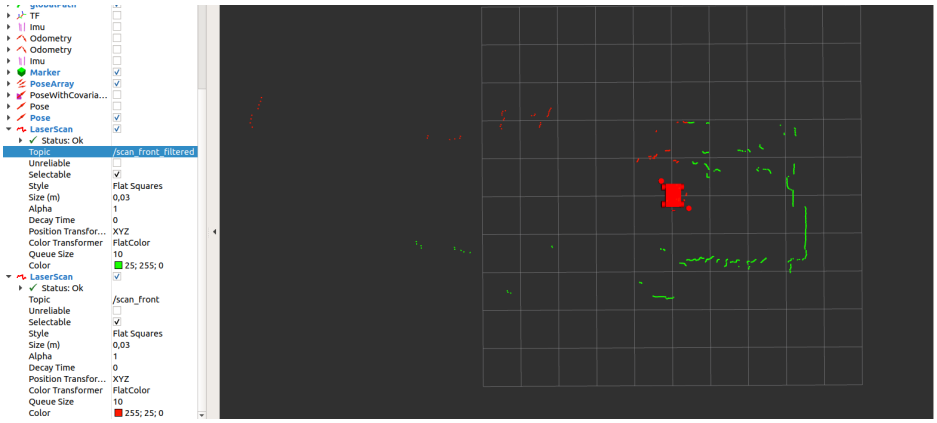
2. DAPF moves the robot smoothly because the driving force generated by the field is continuous and the local goal is updated in real-time thanks to the strategy proposed in Section 4.2.3. The movement of DWA is with pauses since DWA plans a short local path one after another within the dynamic windows, which cannot be connected perfectly considering the change of velocity and acceleration in the robot's state. This phenomenon can be witnessed, especially when turning.

3. DWA can not take full advantage of the OMR even if the parameter *holonomic_robot* is set as true. It takes 30s to pass the first passage between two boxes, while DAPF takes 7s. Besides, DWA fail to plan a path in front of the last obstacle, but actually, there is enough space for the rover to pass, which indicates the poor performance of DWA in narrow spaces.

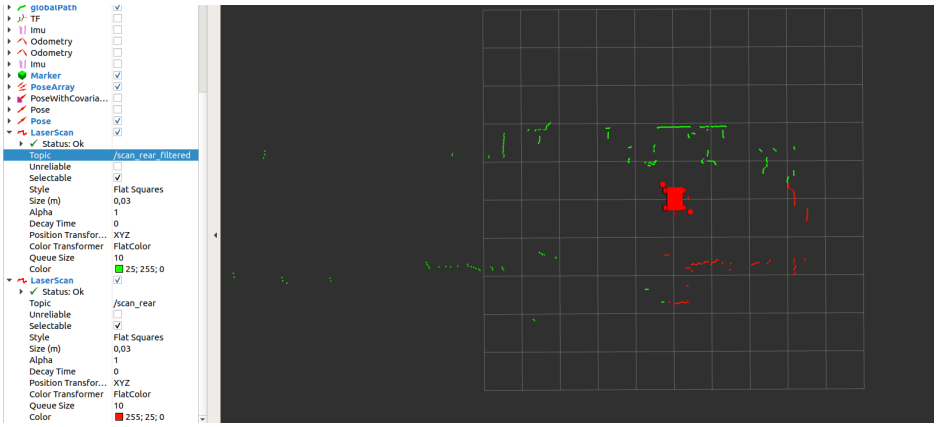
5.7.2 Experiment of avoiding moving obstacle

To test the ability of DAPF to avoid moving obstacles, a tester sitting on a swivel armchair moves heading to the robot to perform as a moving obstacle. The result of the experiment is shown in FIg. 5.20:

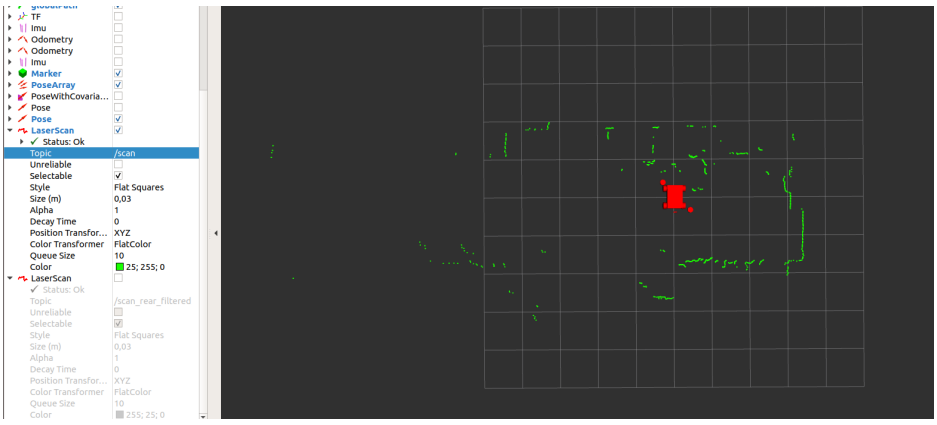
Taking the white line on the floor as reference, it can be seen that, at the beginning, the AMR and the swivel wheelchair are moving towards each other along the same line. When the wheelchair enters the detection region of the AMR, it immediately moves to the side to give way to the wheelchair. As analysed in Section 4.2.4, the extra force component F_{reqv} results from the relative velocity plays the most important role in avoiding moving obstacles, which drives the robot to move in the direction that the projection of the relative velocity on the line connecting robot and obstacle is decreased. All these processes happened within 7s, proving the fast reaction of the AMR when encountering a moving obstacle.



(a) filtered front Lidar data

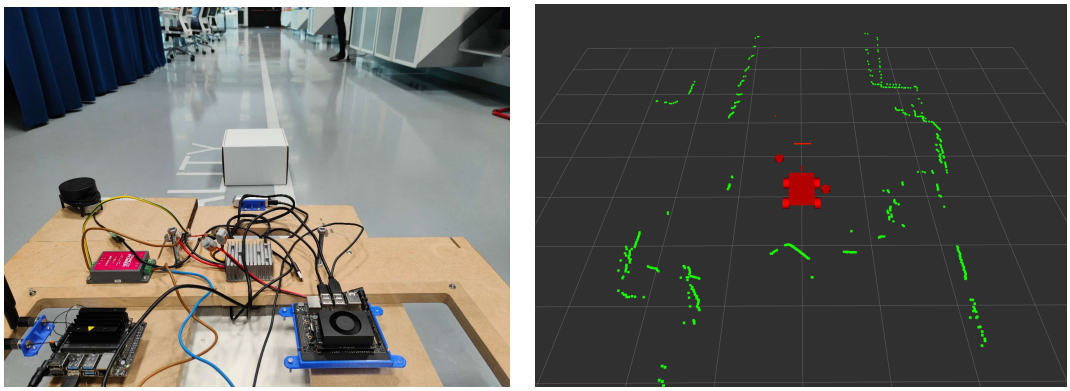


(b) filtered rear Lidar data



(c) merged Lidar data

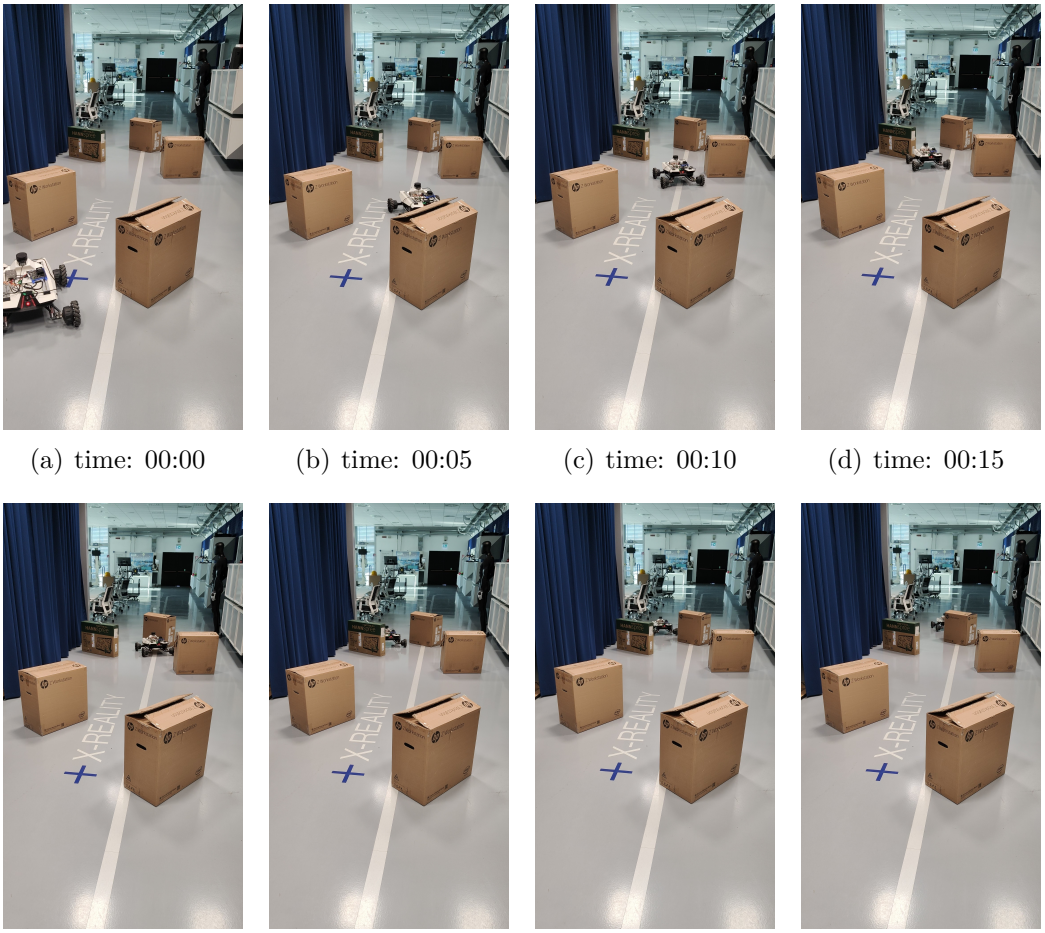
Figure 5.15: Fusion of two Lidars



(a) (b)

Figure 5.16: Lidar and camera fusion

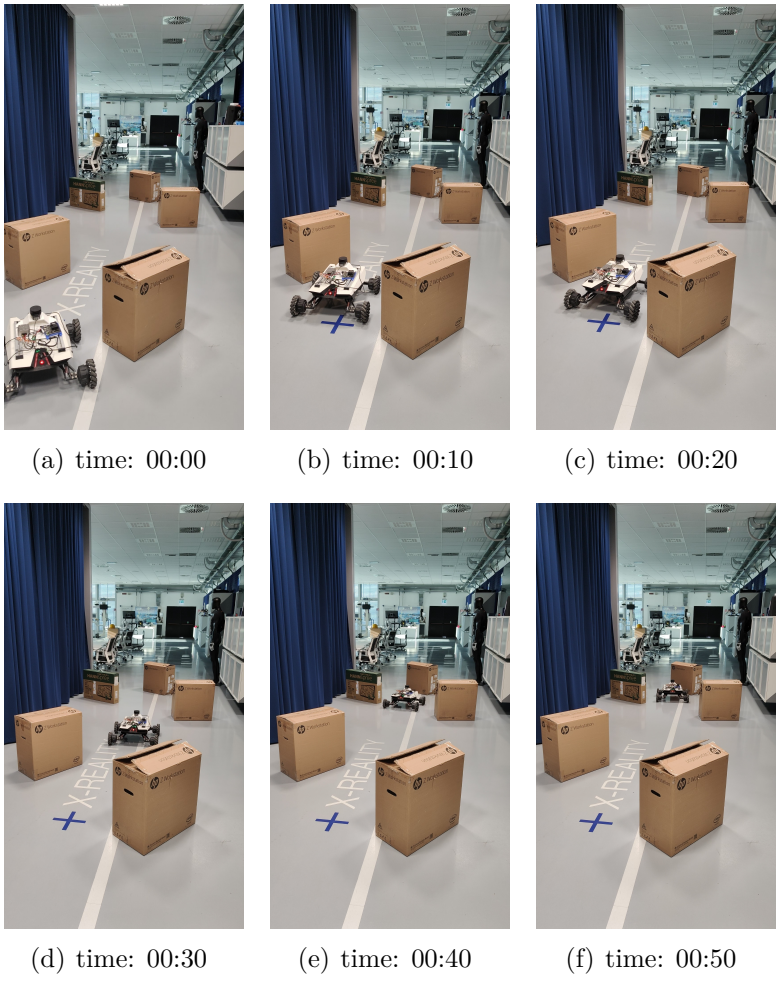
Figure 5.17: Cost map of the CIM4.0 laboratory



(a) time: 00:00 (b) time: 00:05 (c) time: 00:10 (d) time: 00:15

(e) time: 00:20 (f) time: 00:25 (g) time: 00:30 (h) time: 00:35

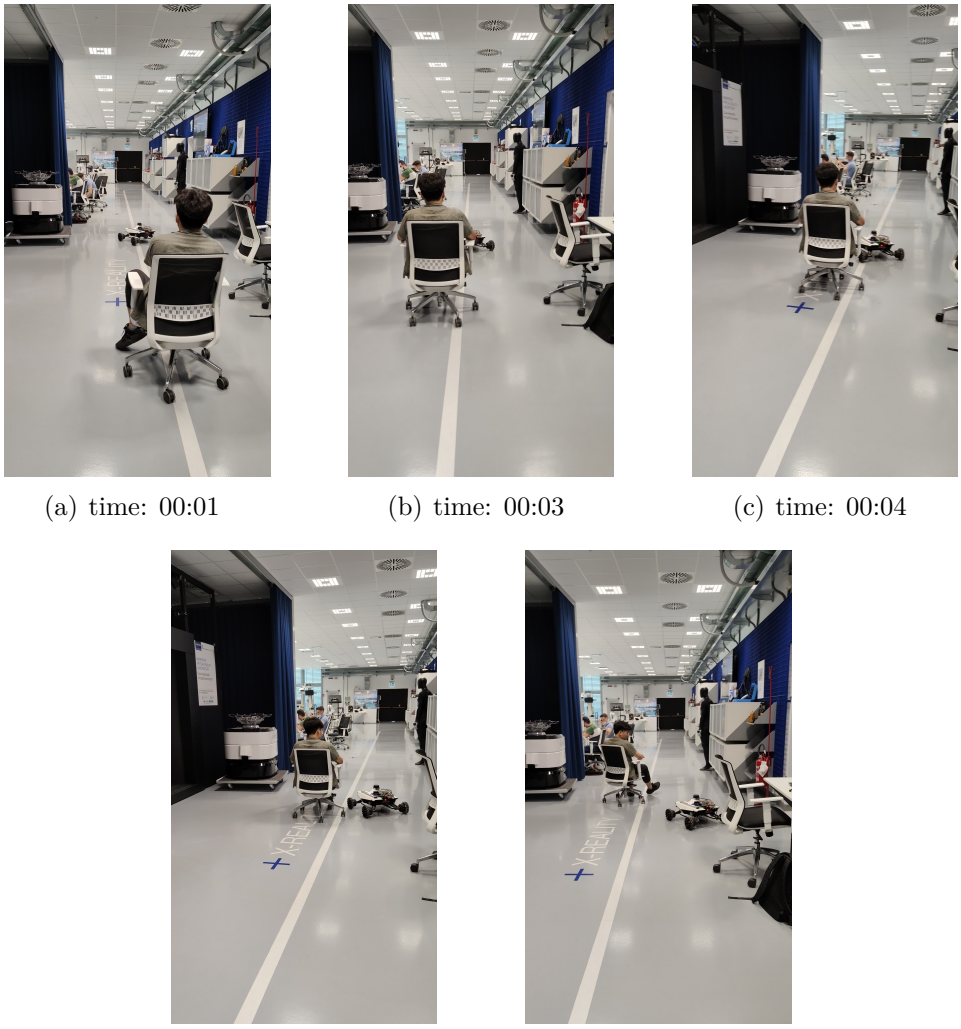
Figure 5.18: Experimental of avoiding static obstacles by D* Lite and DAPF



(a) time: 00:00 (b) time: 00:10 (c) time: 00:20

(d) time: 00:30 (e) time: 00:40 (f) time: 00:50

Figure 5.19: Experimental of avoiding static obstacles by A* and DWA



(a) time: 00:01 (b) time: 00:03 (c) time: 00:04

(d) time: 00:05 (e) time: 00:07

Figure 5.20: Experiment of avoiding moving obstacle

Chapter 6

Conclusion and future work

The main purpose of this thesis is to develop and implement an obstacle avoidance algorithm to be executed on the AMR of FIXIT. Based on the comprehensive analysis of state of the art of path planning algorithms, a Dynamic Artificial Potential Field (DAPF) method is proposed as the Local Path Planning (LPP) to make up the drawbacks of traditional APF, with the ability to avoid moving obstacles. Combined with the D* Lite as Global Path Planning (GPP), the performance of the collision avoidance is verified by the ROS simulation and experiments on the designed real robot.

1. The URDF model of the robot and the environment model are built according to the real ones, and they are simulated by Gazebo and visualized by RViz.

2. A real robot with a three-layer mechanical system: chassis, control, and application layers; and a two-layer control system: a higher-level distributed computer system and a lower-level motion control system, is designed based on ROS to perform the path planning algorithms.

3. For the disadvantage that the traditional APF generates the path with a certain randomness and may not be the global optimal one, a strategy of following the global path by setting the local goal point in real time is proposed. Furthermore, A DAPF is proposed as the LPP algorithm, which takes into consideration the moving obstacle detected by OpenCV tools such as background subtraction and blob detection and tracked by the Kalman filter, estimating its position and velocity.

The proposed algorithm is not only verified by the experiments but also compared to the ROS provided A* and DWA methods, which stress the improvement of the navigation efficiency and the performance of avoiding moving obstacles. Meanwhile, there are several aspects of the work in this thesis that can be further improved:

1. Considering the fact that the AMR can move in omnidirection, although two Lidars and two depth cameras are fused, it is not sufficient to cover all the obstacles at different heights with 360° view. A 3D Lidar can be used to solve this problem if the budget permits.

2. The moving object detection technique provided by OpenCV is not accurate enough. Certain static objects are recognized as moving ones during navigation due to the movement of the robot, which disturbs the AMR to implement tasks.

3. The FIXIT project is a cooperation between the rover and the drone. But until now, the autonomous navigation is performed without utilizing the data from the drone, and vice versa. Developing the rover and drone as a unified system could be a interesting future work.

Bibliography

[1] Yi Wang, Hai-Shu Ma, Jing-Hui Yang, and Ke-Sheng Wang. «Industry 4.0: a way from mass customization to mass personalization production». In: *Advances in Manufacturing* 5.4 (2017), pp. 311–320 (cit. on p. 1).

[2] Kyriakos Manousakis, Tony McAuley, Raquel Morera, and John Baras. «Using multi-objective domain optimization for routing in hierarchical networks». In: *2005 International Conference on Wireless Networks, Communications and Mobile Computing*. Vol. 2. IEEE. 2005, pp. 1460–1465 (cit. on p. 4).

[3] Biwei Tang, Zhanxia Zhu, and Jianjun Luo. «Hybridizing particle swarm optimization and differential evolution for the mobile robot global path planning». In: *International Journal of Advanced Robotic Systems* 13.3 (2016), p. 86 (cit. on p. 4).

[4] Purushothaman Raja and Sivagurunathan Pugazhenthi. «Optimal path planning of mobile robots: A review». In: *International journal of physical sciences* 7.9 (2012), pp. 1314–1320 (cit. on p. 4).

[5] Zexuan Zhu, Fangxiao Wang, Shan He, and Yiwen Sun. «Global path planning of mobile robots using a memetic algorithm». In: *International Journal of Systems Science* 46.11 (2015), pp. 1982–1993 (cit. on p. 4).

[6] Peng Yang, Ke Tang, Jose A Lozano, and Xianbin Cao. «Path planning for single unmanned aerial vehicle by separately evolving waypoints». In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1130–1146 (cit. on p. 4).

[7] N Buniyamin, N Sariff, WAJ Wan Ngah, and Z Mohamad. «Robot global path planning overview and a variation of ant colony system algorithm». In: *International journal of mathematics and computers in simulation* 5.1 (2011), pp. 9–16 (cit. on p. 5).

[8] Jared Giesbrecht. *Global path planning for unmanned ground vehicles*. Tech. rep. DEFENCE RESEARCH and DEVELOPMENT SUFFIELD (ALBERTA)¦ 2004 (cit. on p. 5).

[9] Christos Alexopoulos and Paul M Griffin. «Path planning for a mobile robot». In: *IEEE Transactions on systems, man, and cybernetics* 22.2 (1992), pp. 318–322 (cit. on p. 5).

[10] Edsger W Dijkstra et al. «A note on two problems in connexion with graphs». In: *Numerische mathematik* 1.1 (1959), pp. 269–271 (cit. on p. 5).

[11] Anthony Stentz. «Optimal and efficient path planning for partially known environments». In: *Intelligent unmanned ground vehicles*. Springer, 1997, pp. 203–220 (cit. on p. 5).

[12] Sven Koenig and Maxim Likhachev. «D^* lite». In: *Aaai/iaai* 15 (2002), pp. 476–483 (cit. on pp. 5, 55, 58).

[13] Sven Koenig and Maxim Likhachev. «Incremental a». In: *Advances in neural information processing systems* 14 (2001) (cit. on p. 5).

[14] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. «Probabilistic roadmaps for path planning in high-dimensional configuration spaces». In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580 (cit. on p. 5).

[15] Panagiotis G Zavlangas and Spyros G Tzafestas. «Motion control for mobile robot obstacle avoidance and navigation: a fuzzy logic-based approach». In: *Systems Analysis Modelling Simulation* 43.12 (2003), pp. 1625–1637 (cit. on p. 5).

[16] Howard Li, Simon X Yang, and Mae L Seto. «Neural-network-based path planning for a multirobot system with moving obstacles». In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 39.4 (2009), pp. 410–419 (cit. on p. 5).

[17] Hong Qu, Simon X Yang, Allan R Willms, and Zhang Yi. «Real-time robot path planning based on a modified pulse-coupled neural network model». In: *IEEE Transactions on Neural Networks* 20.11 (2009), pp. 1724–1739 (cit. on p. 5).

[18] Xiao-Ping Zeng, Yong-Ming Li, and Jian Qin. «A dynamic chain-like agent genetic algorithm for global numerical optimization and feature selection». In: *Neurocomputing* 72.4-6 (2009), pp. 1214–1228 (cit. on p. 5).

[19] Yong Zhang, Dun-wei Gong, and Jian-hua Zhang. «Robot path planning in uncertain environment using multi-objective particle swarm optimization». In: *Neurocomputing* 103 (2013), pp. 172–185 (cit. on p. 5).

[20] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136 (cit. on p. 6).

[21] František Duchoň, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. «Path planning with modified a * algorithm for a mobile robot». In: *Procedia Engineering* 96 (2014), pp. 59–69 (cit. on p. 6).

[22] Antti Autere et al. *Extensions and Applications of the A* Algorithm*. Helsinki University of Technology, 2005 (cit. on p. 7).

[23] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984 (cit. on p. 7).

[24] John H Reif. «Complexity of the mover's problem and generalizations». In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE Computer Society. 1979, pp. 421–427 (cit. on p. 8).

[25] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. «Anytime motion planning using the RRT». In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1478–1483 (cit. on p. 8).

[26] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. «Real-time motion planning with applications to autonomous urban driving». In: *IEEE Transactions on control systems technology* 17.5 (2009), pp. 1105–1118 (cit. on p. 9).

[27] Steven M LaValle et al. «Rapidly-exploring random trees: A new tool for path planning». In: (1998) (cit. on p. 9).

[28] Gregory Antonovsky. *Bidirectional RRT* FND algorithm*. 2018. URL: `https://github.com/onlinex/bidirectionalRRTStarFND.git` (cit. on p. 9).

[29] Sertac Karaman and Emilio Frazzoli. «Sampling-based algorithms for optimal motion planning». In: *The international journal of robotics research* 30.7 (2011), pp. 846–894 (cit. on p. 10).

[30] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. «Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2997–3004 (cit. on p. 10).

[31] Steven M LaValle, James J Kuffner, BR Donald, et al. «Rapidly-exploring random trees: Progress and prospects». In: *Algorithmic and computational robotics: new directions* 5 (2001), pp. 293–308 (cit. on p. 10).

[32] James J Kuffner and Steven M LaValle. «RRT-connect: An efficient approach to single-query path planning». In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001 (cit. on p. 10).

[33] Bing Kang, XH Wang, and Fu Liu. «Path planning of searching robot based on improved ant colony algorithm [J]». In: *Journal of Jilin University (Engineering and Technology Edition)* 44.4 (2014), pp. 1062–1068 (cit. on p. 10).

[34] MAO Lin-bo, LIU Shi-rong, and YU Jin-shou. «An improved ant colony algorithm for mobile robot path planning». In: 8 (2006), pp. 997–1001 (cit. on p. 10).

[35] Thomas Stützle and Holger Hoos. «Improvements on the ant-system: Introducing the max-min ant system». In: *Artificial neural nets and genetic algorithms*. Springer. 1998, pp. 245–249 (cit. on p. 10).

[36] Bing Shuang, Jiapin Chen, and Zhenbo Li. «Study on hybrid PS-ACO algorithm». In: *Applied Intelligence* 34.1 (2011), pp. 64–73 (cit. on p. 10).

[37] De-Lin Luo and Shun-Xiang Wu. «Ant colony optimization with potential field heuristic for robot path planning». In: *Systems Engineering and Electronics* 32.6 (2010), pp. 1277–1280 (cit. on p. 10).

[38] Nicolas Monmarché, Frédéric Guinand, and Patrick Siarry. *Artificial ants*. Wiley-ISTE Hoboken, 2010 (cit. on p. 10).

[39] Christian Blum. «Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling». In: *Computers & Operations Research* 32.6 (2005), pp. 1565–1591 (cit. on p. 10).

[40] Marco Dorigo and Luca Maria Gambardella. «Ant colony system: a cooperative learning approach to the traveling salesman problem». In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66 (cit. on p. 11).

[41] URL: `https://complex-systems-ai.com/en/algorithms-desaims/ant-colony/` (cit. on p. 11).

[42] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. «The dynamic window approach to collision avoidance». In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33 (cit. on p. 13).

[43] Oliver Brock and Oussama Khatib. «High-speed navigation using the global dynamic window approach». In: *Proceedings 1999 ieee international conference on robotics and automation (Cat. No. 99CH36288C)*. Vol. 1. IEEE. 1999, pp. 341–346 (cit. on pp. 13, 14).

[44] URL: http://wiki.ros.org/dwa_local_planner (cit. on p. 13).

[45] Zhang Hong, Sun Chun-Long, Zheng Zi-Jun, An Wei, Zhou De-Qiang, and Wu Jing-Jing. «A modified dynamic window approach to obstacle avoidance combined with fuzzy logic». In: *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*. IEEE. 2015, pp. 523–526 (cit. on p. 14).

[46] Omer Ali Abubakr, Mohammed Abdel Kareem Jaradat, and Mamoun Adel Hafez. «A reduced cascaded fuzzy logic controller for dynamic window weights optimization». In: *2018 11th International Symposium on Mechatronics and its Applications (ISMA)*. IEEE. 2018, pp. 1–4 (cit. on p. 14).

[47] Sean Quinlan and Oussama Khatib. «Elastic bands: Connecting path planning and control». In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 802–807 (cit. on p. 14).

[48] Thorsten Brandt and Thomas Sattel. «Path planning for automotive collision avoidance based on elastic bands». In: *IFAC Proceedings Volumes* 38.1 (2005), pp. 210–215 (cit. on p. 14).

[49] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. «Timed-Elastic-Bands for time-optimal point-to-point nonlinear model predictive control». In: *2015 European Control Conference (ECC)*. 2015, pp. 3352–3357. DOI: 10.1109/ECC.2015.7331052 (cit. on p. 14).

[50] URL: https://epsavlc.github.io/2019/08/06/teb_g2o.html (cit. on p. 15).

[51] Kristin Glass, Richard Colbaugh, David Lim, and Homayoun Seraji. «Real-time collision avoidance for redundant manipulators». In: *IEEE transactions on robotics and automation* 11.3 (1995), pp. 448–457 (cit. on p. 15).

[52] Chia-Chia Kao, Chih-Min Lin, and Jih-Gau Juang. «Application of potential field method and optimal path planning to mobile robot control». In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, pp. 1552–1554 (cit. on p. 16).

[53] Jean Bosco Mbede, Xinhan Huang, and Min Wang. «Fuzzy motion planning among dynamic obstacles using artificial potential fields for robot manipulators». In: *Robotics and autonomous Systems* 32.1 (2000), pp. 61–72 (cit. on p. 16).

[54] J Randolph Andrews and Neville Hogan. «Impedance control as a framework for implementing obstacle avoidance in a manipulator». MA thesis. M. I. T., Dept. of Mechanical Engineering, 1983 (cit. on p. 16).

[55] Daniel Koditschek. «Exact robot navigation by means of potential functions: Some topological considerations». In: *Proceedings. 1987 IEEE International Conference on Robotics and Automation*. Vol. 4. IEEE. 1987, pp. 1–6 (cit. on p. 16).

[56] Elon Rimon and Daniel E Koditschek. «The construction of analytic diffeomorphisms for exact robot navigation on star worlds». In: *Transactions of the American Mathematical Society* 327.1 (1991), pp. 71–116 (cit. on p. 16).

[57] Pradeep Khosla and Richard Volpe. «Superquadric artificial potentials for obstacle avoidance and approach». In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE. 1988, pp. 1778–1784 (cit. on p. 16).

[58] Murat Köseoğlu, Orkan Murat Çelik, and Ömer Pektaş. «Design of an autonomous mobile robot based on ROS». In: *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE. 2017, pp. 1–5 (cit. on p. 17).

[59] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. «ROS: an open-source Robot Operating System». In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. on pp. 17, 20).

[60] URL: `http://library.isr.ist.utl.pt/docs/roswiki/ROS%5C%282f%5C%29Concepts.html` (cit. on p. 19).

[61] ChangLong Ye, ShuGen Ma, and Li Hui. «An omnidirectional mobile robot». In: *Science China Information Sciences* 54.12 (2011), pp. 2631–2638 (cit. on p. 20).

[62] URL: `https://en.wikipedia.org/wiki/Ackermann_steering_geometry` (cit. on p. 20).

[63] Ioan Doroftei, Victor Grosu, and Veaceslav Spinu. *Omnidirectional mobile robot-design and implementation*. INTECH Open Access Publisher London, UK, 2007 (cit. on p. 21).

[64] Chao Ren and Shugen Ma. «Dynamic modeling and analysis of an omnidirectional mobile robot». In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 4860–4865 (cit. on p. 21).

[65] Florentina Adăscăliţei and Ioan Doroftei. «Practical applications for mobile robots based on mecanum wheels-a systematic survey». In: *The Romanian Review Precision Mechanics, Optics and Mechatronics* 40 (2011), pp. 21–29 (cit. on p. 22).

[66] Stephen L Dickerson and Brett D Lapin. «Control of an omni-directional robotic vehicle with Mecanum wheels». In: *NTC'91-National Telesystems Conference Proceedings*. IEEE. 1991, pp. 323–328 (cit. on p. 23).

[67] URL: https://en.wikipedia.org/wiki/Mecanum_wheel#cite_note-5 (cit. on p. 24).

[68] Mahmood Reza Azizi, Alireza Rastegarpanah, and Rustam Stolkin. «Motion planning and control of an omnidirectional mobile robot in dynamic environments». In: *Robotics* 10.1 (2021), p. 48 (cit. on p. 24).

[69] URL: http://library.isr.ist.utl.pt/docs/roswiki/urdf(2f)XML.html (cit. on p. 28).

[70] URL: https://articulatedrobotics.xyz/ready-for-ros-7-urdf/ (cit. on p. 30).

[71] URL: http://wiki.ros.org/urdf/XML/joint (cit. on p. 31).

[72] Tully Foote. «tf: The transform library». In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373 (cit. on pp. 32, 33).

[73] Patricio Castillo-Pizarro, Tomás V Arredondo, and Miguel Torres-Torriti. «Introductory survey to open-source mobile robot simulation software». In: *2010 Latin American Robotics Symposium and Intelligent Robotics Meeting*. IEEE. 2010, pp. 150–155 (cit. on p. 34).

[74] URL: http://wiki.ros.org/move_base (cit. on p. 37).

[75] URL: http://wiki.ros.org/costmap_2d (cit. on p. 38).

[76] URL: https://groups.csail.mit.edu/drl/courses/cs54-2001s/odometry.html (cit. on p. 39).

[77] Gang Peng, Wei Zheng, Zezao Lu, Jinhu Liao, Lu Hu, Gongyue Zhang, and Dingxin He. «An improved AMCL algorithm based on laser scanning match in a complex and unstructured environment». In: *Complexity* 2018 (2018) (cit. on p. 40).

[78] URL: http://wiki.ros.org/nav_core (cit. on p. 41).

[79] Rachael N Darmanin and Marvin K Bugeja. «Autonomous Exploration and Mapping using a Mobile Robot Running ROS.» In: *ICINCO (2)*. 2016, pp. 208–215 (cit. on p. 43).

[80] URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ (cit. on pp. 48, 49).

[81] URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/ (cit. on pp. 48, 49).

[82] URL: https://www.slamtec.com/en/Lidar/A1 (cit. on p. 50).

[83] URL: https://www.slamtec.com/en/Lidar/A1Spec (cit. on p. 51).

[84] URL: https://www.intelrealsense.com/depth-camera-d435i/ (cit. on p. 52).

[85] Ganesan Ramalingam and Thomas Reps. «An incremental algorithm for a generalization of the shortest-path problem». In: *Journal of Algorithms* 21.2 (1996), pp. 267–305 (cit. on p. 54).

[86] Sven Koenig, Maxim Likhachev, and David Furcy. «Lifelong planning A». In: *Artificial Intelligence* 155.1-2 (2004), pp. 93–146 (cit. on p. 54).

[87] Anthony Stentz et al. «The focussed dˆ* algorithm for real-time replanning». In: *IJCAI*. Vol. 95. 1995, pp. 1652–1659 (cit. on p. 55).

[88] Kaili Xie, Jie Qiang, and Haitao Yang. «Research and optimization of d-*t lite algorithm in track planning». In: *IEEE Access* 8 (2020), pp. 161920–161928 (cit. on pp. 55, 56).

[89] Yang Zhaofeng and Zhang Ruizhe. «Path planning of multi-robot cooperation for avoiding obstacle based on improved artificial potential field method». In: *Sensors & Transducers* 165.2 (2014), p. 221 (cit. on p. 59).

[90] URL: https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html (cit. on p. 75).

[91] URL: https://learnopencv.com/blob-detection-using-opencv-python-c/ (cit. on p. 76).