



**POLITECNICO**  
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE

# Development of a modified A-Star Algorithm for path planning

TESI DI LAUREA MAGISTRALE IN  
AUTOMATION AND CONTROL ENGINEERING - INGEGNERIA  
DELL'AUTOMAZIONE

Author: **Francesco Desiderio**

Student ID: 10740771

Advisor: Prof. Matteo Matteucci

Co-advisors: none

Academic Year: 2021-22



*'A smooth sea never made a skilled sailor.'*



# Abstract

Path planning plays an essential role in mobile robot navigation, and the A\* algorithm is one of the best-known path planning algorithms. However, the conventional A\* algorithm and the subsequent improved algorithms still have some limitations in terms of robustness and efficiency. These limitations include slow algorithm efficiency, weak robustness and collision. In this work of thesis is proposed a modified implementation of A\* algorithm, this algorithm is chosen after the study of the state of the art. The new algorithm introduces the expansion distance, a new method to construct the OPEN list and heuristic function optimization. The expansion distance extends a certain distance from obstacles to improve path robustness by avoiding collisions. Heuristic function optimization designs a new heuristic function to replace the traditional heuristic function.

Furthermore another improvement is performed to solve the problem of slow search speed and low efficiency of the algorithm. When a set of nodes contains obstacle, this node is defined as an untrusted point, it is pre-inserted in the CLOSED list, and will not be searched or evaluated. The rest of the thesis activity concerns the simulation of autonomous driving, aimed to reproduce the behavior of a work vehicle able to carry out its activity in autonomous or semi-autonomous way. In this case the vehicle has to move from its starting position to a certain point of arrival, avoiding collisions with any type of obstacle. To carry out the simulation are used Simulink and ROS, and two type of kinematics: differential drive and four-wheel steering, that is then approximated with the bicycle model.

**Keywords:** path planning, A\* algorithm, algorithm implementation, modified A\*, expansion distance, heuristic function, autonomous driving, differential drive, bicycle model



# Abstract in lingua italiana

La pianificazione del percorso svolge un ruolo essenziale nella navigazione dei robot mobili e l'algoritmo A\* è uno degli algoritmi di pianificazione del percorso più noti. Tuttavia, l'algoritmo A\* convenzionale e i successivi algoritmi migliorati presentano ancora alcune limitazioni in termini di robustezza ed efficienza. Queste limitazioni includono la bassa efficienza dell'algoritmo, debole robustezza e la collisione. In questo lavoro di tesi viene proposta un'implementazione modificata dell'algoritmo A\*, tale algoritmo viene scelto dopo una fase di studio dello stato dell'arte. Il nuovo algoritmo introduce la distanza di espansione, un nuovo metodo per costruire la lista OPEN e l'ottimizzazione della funzione euristica. La distanza di espansione introduce una certa distanza dagli ostacoli per migliorare la robustezza del percorso evitando collisioni. Ottimizzazione della funzione euristica vuol dire progettare una nuova funzione euristica per sostituire quella tradizionale.

Inoltre viene eseguito un altro miglioramento per risolvere il problema della bassa velocità di ricerca e della bassa efficienza dell'algoritmo. Quando un insieme di nodi contiene un ostacolo, questo nodo viene definito come un punto non attendibile, viene preinserito nella CLOSED list e non verrà ricercato o valutato. Il resto dell'attività di tesi riguarda la simulazione e la guida autonoma, volta a riprodurre il comportamento di un mezzo da lavoro in grado di svolgere la propria attività in modo autonomo o semi-autonomo. In questo caso il veicolo deve spostarsi dalla sua posizione di partenza ad un certo punto di arrivo, evitando collisioni con qualsiasi tipo di ostacolo. Per effettuare la simulazione vengono utilizzati Simulink e ROS, e due tipi di cinematica: trazione differenziale e modello bicicletta, derivato dal più complesso modello a 4 ruote sterzanti.

**Parole chiave:** pianificazione del percorso, algoritmo A\*, implementazione algoritmo, A\* modificato, espansione della distanza, funzione euristica, guida autonoma, trazione differenziale, modello bicicletta





# Contents

<b>Abstract</b>	<b>iii</b>
<b>Abstract in lingua italiana</b>	<b>v</b>
<b>Contents</b>	<b>vii</b>
<b>Thesis Summary</b>	<b>1</b>
<b>Introduction</b>	<b>3</b>
<b>1 Motion Planning</b>	<b>5</b>
1.1 Perception . . . . .	5
1.2 Planning . . . . .	6
1.2.1 Occupancy space . . . . .	7
1.3 Controlling . . . . .	8
<b>2 Path planning - State of the art</b>	<b>9</b>
2.1 Graph-based algorithm . . . . .	9
2.1.1 Dijkstra . . . . .	9
2.1.2 EBS-A* . . . . .	10
2.1.3 EBSHA* . . . . .	10
2.1.4 IQACA . . . . .	10
2.1.5 Improved Eight-way A* . . . . .	10
2.1.6 SBMPC . . . . .	11
2.2 Sample-based algorithm . . . . .	12
2.2.1 ENPS RRT* . . . . .	12
2.2.2 ECRE . . . . .	12
2.2.3 Improved RRT . . . . .	13
2.2.4 RRT plus DWA . . . . .	13
2.2.5 Sweeping RRT . . . . .	14

2.2.6	PRM*	14
2.3	Potential field method	15
2.3.1	APF	15
2.3.2	TPP	16
2.3.3	IAPF	17
2.4	Numerical Optimization	17
2.4.1	4-step path planning	17
2.4.2	RHP plus sigmoid function	18
2.4.3	Frenet plus quintic polynomials	19
2.4.4	Q-Learning	20
<b>3</b>	<b>Path planning algorithm: design and implementation</b>	<b>21</b>
3.1	The A* algorithm	21
3.1.1	The working principles	22
3.2	How to improve	24
3.2.1	First improvement	24
3.2.2	Second improvement	24
3.2.3	Third improvement	24
3.3	Implementation	26
3.3.1	Map inflated	28
3.3.2	Comparison with Matlab planner	30
<b>4</b>	<b>Simulink and ROS simulation</b>	<b>35</b>
4.1	Robot Operating System	35
4.2	Path following strategy	36
4.2.1	Pure Pursuit block	37
4.2.2	Vector field histogram	39
4.3	Overall scheme	42
4.3.1	Process input	43
4.3.2	Compute Velocity and Heading for Path Following	44
4.3.3	Adjust Velocities to Avoid Obstacles	44
4.3.4	Outputs block	45
4.3.5	Kinematics	46
4.4	Configuration and run of the Model	50
4.4.1	Differential drive robot	51
4.4.2	Ackermann steering vehicle	54
<b>5</b>	<b>Conclusion and future developments</b>	<b>59</b>

**Bibliography**

**61**

**Acknowledgements**

**63**



# Thesis Summary

The goal of the research is studying the state of the art of the path planning algorithm and implementing a modified version of the best algorithm in the market.

In the first chapter is introduced the concept of motion planning and are illustrated the three main part, i.e. Perception , Planning and Controlling.

In chapter 2 is illustrated the state of the art. The main path planning algorithms are explained, they can be divided into multiple classifications according to different ways. In particular are described four different type of algorithm: Graph-based, Sample-based, Potential field method and Numerical optimization.

The chapter 3 was the core of the thesis and is dedicated to the design and implementation of the algorithm. There are introduced three main improvements in order to make the A\* more efficient and robust. Then the implemented algorithm is compared with two Matlab planner showing the differences and the similarities.

After that are carried out a lot of simulation in Simulink environment, illustrated in chapter 4. This chapter will illustrate in details the way in which the path following and obstacle avoidance strategies have been implemented. The goal of this phase of simulation is to combine the aforementioned planning with an obstacle avoidance system so that they operate in parallel and in real time. The ROS toolbox is used to take track of the current position of the robot and to take into account the kinematics, in particular the differential drive. Then, with another simulink scheme, is tested the system with the bicycle kinematic model. The conclusive chapter 5 sums up the improvements and the main results that the whole work of thesis has produced. In the end some future developments are mentioned.



# Introduction

In the last few years, more and more often, we hear about self-driving cars, opening in the minds of drivers scenarios of unmanned cars and the possibility of being able to travel without worrying about what is happening around them. The reality is obviously different and before seeing driverless cars we will have to wait few years, while models are already on the market that can potentially drive autonomously in certain situations.

To clarify the subject, the Society of Automotive Engineers, an association with over 100,000 engineers engaged in the transport sector, has made a distinction that divides the systems present on a car into five levels of automation [19]. It is important to be clear that at the moment the Highway Code prohibits traveling without hands on the wheel. This clarification underlines how the driver must keep in mind that technology is not a means to be distracted on board but, on the contrary, a valid assistant during every transfer. It is necessary to keep in mind the distinctions between the levels of autonomous driving, in fact today the maximum technological expression is represented by Level 2, where we can find partial automation on board. In this specific driving mode, the system controls all the dynamic aspects of the car, using information on the driving environment. The driver checks the remaining aspects. Steering and speed control can be entrusted to the system, while the surrounding environment can be controlled. In particular with level 2 of automation, the car cannot circulate in a semi-autonomous form in every situation but only under certain conditions, such as motorway sections. The on-board systems allow you to take your hands off the wheel for a few moments, as the system controls steering and speed. With level 3 the automated system can have control of the steering, speed and the surrounding environment. A strong technological leap will come with level 4. Road tests have already been carried out, such as the 190-kilometer journey made by a fleet of Hyundai, including the new Nexa in a “laboratory” version, capable of driving in completely autonomous mode on the motorway at  $110\text{km/h}$  [1]. Upon entering the highway, the vehicles traveled in response to the natural flow of traffic, making lane changes and overtaking maneuvers.

While levels 3 and 4 will still take some time, level 2 is already on a long list of smart models on the market. Level 5 means full autonomous vehicles, which can drive based

on the perception of surrounding environmental conditions, just like human drivers. In recent years, research on autonomous driving has become a hot spot. Advanced Driving Assistant System (ADAS) has developed rapidly, and some technologies have been applied in mass production. The autonomous driving system contains a wide range of technologies, including multi-sensor fusion technology, signal processing, communication technology and artificial intelligence. Autonomous driving technology can be summarized as: “Identify the surrounding environment through a variety of on-board sensors, and make analysis and judgments based on the obtained environmental information, thereby controlling the movement of the vehicle, and ultimately achieving autonomous driving” [18].

In this field the path planning algorithms play an essential role, they are the core of the motion planning phase alongside with the perception and controlling.



# 1 | Motion Planning

The autonomous driving technology mainly includes three parts: perception, planning and controlling .

## 1.1. Perception

Perception of the surroundings is a core building block for autonomous driving. In practice, perception is implemented by collecting data from various sensors, such as lidars, radars, cameras and ultrasound sensors, and processing these observations in the autonomous vehicle's compute unit. Thanks to the readings coming from various sensor, the system is able to reconstruct the surrounding environment and to be aware of the road limits and the obstacles. The results of these computations then allow the AI to drive the vehicle safely and efficiently. In order to predict what other road users are doing, the first step is to identify where they are. This step is called object detection, and it is solved by combining observations from both cameras and lidar sensors, using various computer vision and sensor fusion algorithms, such as deep neural networks. As a result, we get estimates of where nearby cars, pedestrians and other road users are, relative to the autonomous vehicle [17].

The next step is to estimate how the other road users are currently moving (object tracking), and predict how they will likely move in the future (movement prediction). This can be done using a sequence of observations of their locations, and a model that predicts how a car or a pedestrian will most likely move in the future. We can take advantage of our knowledge of the traffic participant type in this prediction, as pedestrians can change their direction of movement much faster than cars, and cars tend to move at a much higher velocity than pedestrians.

## 1.2. Planning

Autonomous navigation is a valuable asset for mobile robots. It helps to mitigate their dependency on human intervention. However, it also entails many tasks or problems to solve, like path planning. This task lies in finding the best course of action to make a robot reach the desired state from its current one. For example, both states could be, respectively, the goal and the initial position. This course of action comes in the form of a path or route, it guides the robot through the desired target state. However, there may be numerous possible paths, given the free space in which the robot can move. Path planning algorithms generally try to obtain the best path or at least an admissible approximation to it. The best path here refers to the optimal one, in the sense that the resulting path comes from minimizing one or more optimization functions. For instance, the optimal path may be the one entailing the least amount of time. This is critical in missions such as those of the search-and-rescue field : victims of a disaster may ask for help in life-or-death situations. Another optimization function to consider could be the energy of the robot. In the case of planetary exploration, this is critical since rovers have limited energetic resources available. At the same time, the path generated by the planner must follow any imposed restrictions. These may come from the limitations in the adaptability of the robot to certain terrains. The locomotion of the robot and the characteristics of the existing terrain limit the kind of manoeuvres that can be performed, this consequently reduces the number of feasible paths that the path planner can generate.

In the literature there are a vast number of path planning approaches and this number has continued to increase over the years. For this reason, selecting the most appropriate approach given certain requirements (for instance, the aforementioned locomotion restrictions) can be a challenging task.

Planning is the key to autonomous driving. It is necessary to make reasonable judgments on the scene to ensure the safety of autonomous vehicles. It acts as a decision maker. On the basis of satisfying safe driving, the comfort of the vehicle must also be considered to realize different plans for different types of driving scenarios. Path planning is the determination of a collision-free path in a given environment, which may often be cluttered in the real world. The planned path determines whether a mobile robot can achieve reliable and efficient autonomous navigation. Therefore, path planning plays an essential role in mobile robot navigation. With the widespread application of mobile robots, research on path planning is becoming increasingly popular [18].

### 1.2.1. Occupancy space

As stated before, a basic motion planning problem is to compute a continuous path that connects a start configuration  $S$  and a goal configuration  $G$ , while avoiding collision with known obstacles. The robot and obstacle geometry are described in a 2D or 3D workspace, while the motion is represented as a path in (possibly higher-dimensional) configuration space.

**The Configuration space  $C$**  is the set of all possible configurations, for example: if the robot is a 2D shape that can translate and rotate, the workspace is 2-dimensional; if the robot is a solid 3D shape that can translate and rotate, the workspace is 3-dimensional [15]

**The free space  $C_{free}$**  is the set of configurations that avoids collision with obstacles. The complement of  $C_{free}$  in  $C$  is called obstacle or forbidden region. Often, it is prohibitively difficult to explicitly compute the shape of  $C_{free}$ . However, testing whether a given configuration is in  $C_{free}$  is efficient. First, forward kinematics determine the position of the robot's geometry, and collision detection tests if the robot's geometry collides with the environment's geometry.

**Target space** is a subspace of free space which denotes where we want the robot to move to. In global motion planning, target space is observable by the robot's sensors. However, in local motion planning, the robot cannot observe the target space in some states. To solve this problem, the robot goes through several virtual target spaces, each of which is located within the observable area (around the robot). A virtual target space is called a sub-goal.

**Obstacle space  $C_{obs}$**  is a space that the robot cannot move to. Obstacle space is not opposite of free space.



## 2 | Path planning - State of the art

Path planning algorithms can be divided into multiple classifications according to different ways [7]. Graph search-based planners include the Dijkstra algorithm, the A-star algorithm (A\*), the state lattice algorithm etc. Sampling-based planners include rapidly-exploring random trees (RRT), interpolating curve planners including lines and circles, clothoid curves, polynomial curves, Bezier curves, spline curves; function optimization planners, including the genetic algorithm, swarm particle optimization, etc. Other path planning methods include vision-based path planning, artificial intelligence path planning, and others. For application scenarios such as warehousing and logistics, path planning in a static environment assumes that the robot perceives the environment and uses local path planning algorithms when the environmental information is not fully grasped.

### 2.1. Graph-based algorithm

The general idea of a graph-based algorithm is to generate a discretized representation of the environment, then build a graph with these points and search for the graph with the optimal solution [7]. In the following are described the main graph-based algorithm found in the literature.

#### 2.1.1. Dijkstra

The most known algorithm is the Dijkstra [5]. As a first step, this algorithm takes a node, it can be the initial or the final one, and then proceeds to propagate information to neighborhood nodes. The information reported could be either the value of the cost required to get to the beginning or what remains to get to the goal. Iteratively the algorithm visits the neighbors of the nodes already visited. The cost amount of the information is propagated and the algorithm assigns a parent node to each visited node. If the environment allows it, that is, no obstacle is isolating neither the end nor the beginning, the algorithm ends up visiting both. At this point, the path is retrieved by visiting the parent nodes backwards. In other words, the path starts from the last visited node and goes back through the parent nodes.

### 2.1.2. EBS-A\*

The A\* plans the shortest route, but needs to visit all nodes of the route and select the minimum cost. To do this, it requires a great deal of computing power and a long computing time, so the algorithm's efficiency decreases as the map size increases.

For these reasons, EBS-A\* [4] is implemented which improves the previous algorithm with distance expansion, bidirectional search and smoothing.

The basic idea of smoothing, in particular, is to decompose a 90-degree curve into smaller angles to make the path less edgy. This technique shortens the length of the path by reducing the number of turns with 90 degree angles that would otherwise reduce the overall efficiency of the vehicle.

### 2.1.3. EBSHA\*

The EBHSA\* [5] algorithm introduces the expansion distance, bidirectional search, heuristic function optimization and smoothing into path planning. The expansion distance extends a certain distance from obstacles to improve path robustness by avoiding collisions. Bidirectional search is a strategy that searches for a path from the start node and from the goal node at the same time. Heuristic function optimization designs a new heuristic function to replace the traditional heuristic function. Smoothing improves path robustness by reducing the number of right-angle turns.

### 2.1.4. IQACA

The IQACA is a new optimization algorithm that combines quantum inspired computing with ant colony optimization algorithm [2]. It is based on quantum code and a quantum rotation gate from quantum-inspired computing. The quantum bit (Q-bit) is the basic unit in quantum computing. In the IQACA, the quantum pheromone is obtained by encoding the pheromone left by the ants on the path in the ACA by the Q-bits. The transfer direction of the ants is selected by the quantum pheromone concentration on the path.

### 2.1.5. Improved Eight-way A\*

This algorithm overcomes the problem of the right angle inflection point in the path and plans the shortest path [26]. The crossing nodes and the Floyd algorithm are used for the smoothing process. These two methods need to trace backwards all the nodes of the path one by one, starting from the origin or the end, to determine if they intersect obstacles.

If they do not intersect, the intermediate node will be discarded while, if they intersect, the previous node is kept. These two methods combined together optimize the original path.

Specifically, Floyd's algorithm works as follows:

- **Step 1** Delete the processed nodes from the eight-way-A\* and perform the second analysis back from the end node.
- **Step 2** Connect all inflection points one by one from the end point. Non-inflection point nodes are ignored during the connection process.
- **Step 3** When the points before meeting the obstacle are directly connected, keep looking for the inflection point backwards. At this point there will be a conflict between the path planning and the obstacle. Then we look for the nodes of the no-inflection point that connect the points before the obstacle to those after the obstacle.
- **Step 4** If there is still a conflict with the obstacle while searching for the node, continue with step 3 up to the non-conflicting node
- **Step 5** At this point, connect the start node directly with the last node not in conflict with the obstacle and return to step 2, the process ends when there is a path that connects the start node with the end node without conflicts with the obstacle

In addition, to ensure the safe distance from the obstacle, two methods are used. One is to expand the pixels occupied by the robot's body inside the map pixels and render it consistent with the actual map-to-pixel ratio. The other is to extend the perimeter of the obstacle. With these two methods combined together the safety distance can be set arbitrarily and intuitively without increasing the difficulty of the algorithm, and can effectively reduce the number of crossing grids and search time.

### 2.1.6. SBMPC

The algorithm contains three main functions: SBMPC, SBMPO and Neighbor Generation. The main one, the SBMPC, operates on a direct dynamic graph which is a set of all the nodes and arcs currently in the graph [23]; the component that optimizes is the SBMPO which determines the optimal cost (i.e. the shortest path, the shortest time, minimum energy, etc.) from an initial node to an end node. To facilitate fast replanning, it does not make each node locally consistent after a cost change, but instead uses a heuristic function  $h(\tau, \tau_{goal})$  to focus the search so that it only updates the current cost  $g(\tau)$  and

the nodes needed to get the shortest path. This heuristic function, together with the initial distance estimate, are used to rank the priority queue containing locally inconsistent nodes so that all of them are updated and made locally consistent. Finally, the Neighbor Generation determines the successor nodes to the current node. In the input space, a set of quasirandom samples are generated and used by the robot motion predictor model to compute a new set of outputs. The branching factor determines the number of paths that will be generated. The path is represented by a sequence of states  $\tau(k)$ . The set of states that do not violate any state or no constraint is said to be  $\tau_{free}$ . If  $\tau(k)$  belongs to  $\tau_{free}$ , then the new neighbour node  $\tau_{new}$  and the connecting edge can be added to the graph. If  $\tau_{new}$  already belongs to the graph, then the node currently exists and only the new path to reach the existing node needs to be added.

## 2.2. Sample-based algorithm

Sample-based planning works in a continuous space and guarantees probabilistic completeness, i.e. the planner will surely find the solution, if it exists, using random sampling [7]. The generation of an achievable trajectory is done through collision checking of the nodes and sides.

The algorithm that correspond to the aforementioned category will be listed.

### 2.2.1. ENPS RRT\*

The Enzymatic numerical P systems (ENPS) [6] are an extension of NPS, introduced for modeling and simulation of membrane controller for autonomous mobile devices. The main difference is the enzyme concept which is used to write program-related conditions. The membrane structure is designed to organize programs and variables in modules. All variables are considered global, meaning any program can read or write a variable regardless of the compartment in which the variable is defined. The ENPS is used to simulate the behavior of the RRT\* algorithm and therefore to support it in the search for nodes, making the search more efficient and faster.

### 2.2.2. ECRE

The RRT method is computationally expensive compared to the elliptical path generation methods, the advantage of the Enhanced combined RRT ellipse (ECRE) is that the result is as accurate as that of the RRT and requires far fewer iterations to find the solution. multiple solutions and the planner can choose the best one.



In numerical terms, this algorithm manages to reach the target in 10 iterations, while the RRT needs at least 10,000 of these. This technique has been used for the path planning of worm-shaped robots [25], but it has been seen that some movements are very similar to the movement of a vehicle when it has to park.

### 2.2.3. Improved RRT

The improved RRT takes advantage of Hadoop MapReduce parallel computing acceleration [24].

The approach consists of 4 phases: entry phase, map phase, map reduction phase and exit phase. The master node sends the work done by the RRT to each slave node and manages the workload. The slave nodes iteratively perform each activity of the RRT. First the metric space  $X$  is partitioned into  $n$  subspaces, and each slave node is assigned a partial tree. Then each slave builds its own intermediate tree in an iterative way. During execution, if a worker node reaches the  $X_{obs}$  obstacle set, it sends a message to the master node. This allows in some cases to stop the calculation assuming that there is no feasible solution, for example, this basic idea is very useful for the formal verification of the correctness of the provided algorithms. When a node finishes creating its subtree, it waits for the last node to complete its activities, and so begins the reduction phase which consists of rebuilding the final tree. Each gearbox chains the intermediate shafts from the map phase, which share the same boundary. The final extremity of the  $(i - 1)$ -th subtree becomes the starting end of the  $i$ -th subtree, and so on up to the last subtree.

### 2.2.4. RRT plus DWA

The RRT algorithm aims to find border points and viable routes on the map made up of various cloud points.

Since the RDT will generate too many border points that are extremely close together, it needs to discard the redundant ones. To do this, a filter called frontier filter detection is used [20].

First, the border points are grouped by the moving average algorithm without assigning the number of clusters. The border point closest to the center of each cluster is selected, the others are removed. Sometimes, at the intersection of paths, multiple desired border points are found. The coordinates of the border points relative to the position of the robot are used as selection criteria, since the coordinates provide the orientation information of the border points. During the decision-making process, preferential conditions will determine which border point will be chosen.

The key part is entrusted to DWA, a speed-based local planner. This approach directly calculates the optimal speed of a robot to reach the target without collisions.

The main objectives of the DWA-based planning algorithm are to calculate a valid velocity in the search space, and then to select the optimum speed command for the robot. The allowed speed is calculated according to the kinematics of the robot, which also ensures continuous motion in the robot in the working environment. The set of speeds generates safe trajectories, which allow the robot to stop before the collision. The set of speeds can be achieved in the next time interval given the dynamics of the robot, i.e. in the dynamic window. The optimal speed is chosen to maximize the speed function, the direction closest to the end and the overlap between the straight line of the goal and the possible trajectories.

### 2.2.5. Sweeping RRT

The sweeping RRT approach was designed for robots with legs. It arises from the need to decouple the planning of the body from the planning of the legs, to achieve that it combines global and local planning together [9]. In this way, the overall configuration space does not need to include states associated with the leg joints. Instead, the planning of the robot body movement is performed using already known and larger obstacles which could collide with the robot's body. While the leg step planning is done locally, taking into account the smallest obstacles detected in real time. The combined method allows for rapid global rescheduling thanks to information on obstacles from local path planning. In Sweeping RRT, the obstacles known a priori are taken into account in the use of fast global planning, while the small unknown obstacles are ignored. Local planning takes care of validating the global plan in the presence of small obstacles and, if it is deemed not feasible, it sends information to re-plan the route. In this type of algorithm the global plan is recalculated only in two cases: when the robot is unable to stop in neutral position, or when the local planning fails.

### 2.2.6. PRM\*

In general the PRM works with static obstacles but recent developments extend the change also to dynamic environments with moving obstacles [11]. The PRM constructs a roadmap of the configuration environment without collisions, it can be used to find multiple routes/queries from start to finish. Its biggest advantage is the fact that it keeps the original structure and reuses it to connect two points. This planner has two simultaneous phases: Create the RoadMap and build the route.

The first step is to create the roadmap: a configuration is randomly selected from the configuration space, then it checks for collisions and repeated this step until the selected configuration is free from collisions. The sampling of the configuration space can take place in a deterministic or random way, the generated sample, if it is free from collisions, is linked to the previous configuration in the roadmap by a local planner.

The second step is the path creation phase: to find a path between the starting node and the arrival node, the algorithm tries to connect the two configurations to the roadmoap and searches within it for a local path that connects the starting point with the end point.

The PRM, however, does not reach the optimum, which is why the PRM \* was developed, that is the optimized version of the PRM.

## 2.3. Potential field method

The algorithms presented here rely on defining how the robot reacts at each instant to the presence of obstacles. This reaction can be defined according to a formulation that addresses the location of existing obstacles. A common feature of the different formulation approaches is the low computational requirements needed to produce the reaction, usually in the form of a steering or velocity command. Since this formulation lacks global information, these techniques are commonly used as Local Planners. The formulation in question may be based on the use of fields to tackle the location of obstacles, the detection of obstacle boundaries to circumvent them, or the production of a velocity command after evaluating the available free space or the speed of moving obstacles.

The main algorithm that belongs to this category are presented below.

### 2.3.1. APF

In the Artificial Potential Fields (APF) the movement of the robot is the result of the sum of all the virtual forces that are created by external elements, such as obstacles [13]. It is assigned to them a repulsive force, in this way the robot moves away from them and avoids collisions. On the contrary, an attractive force is assigned to the end point. The main cons of this is that the robot can get stuck in the local minimum points, for this reason works have been developed that have led to combine the APF with genetic methods (evolutionary algorithm). Another can be to combine the algorithm with a PSO or, again, add a repulsive potential field around the robot.

### 2.3.2. TPP

The TPP (Tide-Inspired Path Planning ) algorithm is inspired by the phenomenon of the tides [3]. Tides are regular changes in sea level, these are caused by the gravitational force of the Moon, which creates what is called a tidal bulge, where the water closest to the moon is pulled towards it, while the water further away from the moon is affected by the centrifugal force and is directed in the opposite direction, leading to move away from the Moon.

In a similar way, in the proposed algorithm, the agent (water) is attracted to the arrival (Moon), while obstacles direct the agent away from the arrival. Here the idea is to give the cells near the obstacles a very high cost; so the agent will tend to move away from them, and the cells that lead to the arrival will be given a very low cost.

The surrounding environment is represented with a grid made up of discrete cells, the start and end points and obstacles are generated randomly at each run of the algorithm.

The TPP algorithm consists of 3 major components: obstacles neighbors incrementor, path finder and cell value calculator. Once the start and finish cells and obstacles have been determined, the obstacles neighbors incrementor increases the values of all cells immediately next to the obstacle by one. Every obstacle it has eight neighbors, that is, the cells immediately to its left, right, up, down and diagonally. Therefore, the cost of cells next to an obstacle will be higher than in other cells. Next, the path finder searches for an optimal path from start to finish, it keeps a list of visited cells and initializes the starting point as the current cell. While the list is not empty, the cell visited with the minimum cost is checked to understand if it is the end point, if it is true, then the sequence of cells visited from the starting point to the current point will be returned as the final path. Otherwise the cell value calculator will update the value of all cells close to the current cell.

After updating the value of the cells and calculating the cost, the path finder removes the visited cells from the list, adds the neighbors to the list of cells, marks the neighbors as visited and continues to search for the arrival point by checking all the neighbors that minimize the total cost. This process is repeated until a path is found or until all cells are checked.

### 2.3.3. IAPF

The IAPF combines the potential already described of the APF with the characteristics of the driver [18]. There are different types of driver: cautious, normal and aggressive. In relation to that the potentials around obstacles change their intensity.

The IAPF is a type of path planning based on the characteristics of the driver and the APF method, it considers the differences in the repulsive potential field generated by the different types of drivers. In fact, in the path planning process of an autonomous vehicle, manually driven cars are considered obstacles and their repulsive field varies from car to car. To complete the analysis, road boundaries are also considered obstacles. In traditional APF the function for the potential field is fixed, so the planning is the same for different situations, as in the case of different drivers, which explains the poor adaptability of the APF. The IAPF, on the other hand, reserves different potentials for different drivers: when a normal driver is encountered, the schedule will be similar to that of the APF; with an aggressive driver, a larger lateral distance will be reserved; with a more sedate driver, the lateral distance will be reduced. The result will be a smoother and more natural course, adapted in relation to the different obstacles encountered.

## 2.4. Numerical Optimization

In the numerical optimization approach, by introducing an optimality objective, the motion planning problem for a non holonomic system is reformulated as an open-loop optimal control problem and solved numerically [7].

Those type of algorithm are described in the following.

### 2.4.1. 4-step path planning

The novelties, with respect to the old algorithm, of the 4-step path planning are [10]:

- The path is constructed by using a new four-phase algorithm. In phase 1 and 2, a reverse path is obtained by retrieving a virtual parked-car from the parking space and steering it to a target line in the drive aisle. In phase 3 a forward path is generated by steering a virtual car from the initial pose to the target line. In phase 4 the two paths are connected along the target line.
- Instead of finding a path with sufficient width to accommodate the car or computing the free-space, the obstacles are padded with a buffer zone for collision avoidance, thereby reducing the feasible path to a single line, which greatly improved the

computational efficiency.

- A feasible path is constructed using the line-of-sight (LOS) pure-pursuit guidance technique, which produces a feasible path segments that are natural and smooth
- The problem of retrieving a virtual parked-car from the parking space and the problem of steering a virtual car to a target line are formulated as a 1st-order and a 3rd-order nonlinear switching control problem respectively. For each problem, an effective switching control law is designed. This is the first time that the switching control theory is used to solve the path planning problem.
- The forward path and the backward path are linked together by a fifth-order polynomial function

### 2.4.2. RHP plus sigmoid function

The algorithm presented is based on Rolling Horizon Planning and on the sigmoid, using a configurable version of the latter it is possible to provide an efficient and achievable path in a given prediction horizon [21]. The generated path takes into consideration the position and motion of the vehicle or the detected obstacle.

The algorithm consists of different parts capable of reacting to any obstacle on the road. The initial reference for the vehicle route can be provided by a database containing a digital map or by a previously registered map. The inputs are the position of the vehicle and those of the obstacles, which represent the part of perception and localization. The local path is then calculated according to the desired smoothing parameter and the two parameters of safety distance with respect to obstacles. It is necessary to maintain a high level of passenger safety and comfort, and it is also important to remain consistent with the dynamics of the vehicle when performing a maneuver.

The safety distance when avoiding an obstacle is configurable taking into account the speed of the vehicle. Way-points are generated which represent the position to reach to avoid the obstacle.

The main goal is to calculate the next local path with the RHP, this path depends on the area perceived by the vehicle sensors. In the calculation of the Horizon planning approach the next navigation area is automatically indicated while the vehicle is moving forward. Thanks to the level of perception, the position of the obstacles is defined step by step. When considering moving obstacles, the next trajectory is recalculated to be adapted to the new position of the obstacle. The length of the prediction horizon depends on the detection range of the sensors and is editable.

At the first iteration, the first horizon vector is calculated, which represents the next trajectory to be planned.

When the vehicle reaches the horizon step distance, the second horizon for the next trajectory is calculated. A part of each vector is recalculated at each iteration, so as to update the trajectory and take into account the moving obstacles.

Obviously it is important to choose a correct horizon step, it affects the distance beyond which the trajectory must be recalculated and updated. The relationship between the length of the horizon and the step is represented by the number of points that must be calculated in each horizon vector. Clearly if the length of the step is small, the position vector contains many more points and the generation of the trajectory will be more precise.

### 2.4.3. Frenet plus quintic polynomials

The quintic polynomials are used for the generation of the longitudinal parts of the path. The new path is calculated based on the relative position to the reference path. The goal is to take the vehicle on a path between two limited corridors, and since the vehicle is driven in a limited environment, the reference path is set in the middle of the road [8].

The planning of the route is divided into the following steps:

- Calculate the start values with the Frenet coordinates, in relation to the current position of the vehicle
- Plan different paths with different longitudinal and transversal variations with respect to the reference path
- Transform Frenet coordinates into Cartesian coordinates
- Calculate the cost for each route and select the best one

For the path planning algorithm it is not necessary to know the absolute coordinates of the planned route, but it is sufficient to have those relating to the position of the vehicle. Frenet coordinates indicate the distance of one point from the reference path to the vehicle position itself.

Using Frenet coordinates allows separate optimization of longitudinal and transverse route planning. In each planning step, the initial state described in the Environmental Sensor Coordinate System (ESCS) is transformed into the Frenet space to then calculate a direct path to the target point also described in the Frenet space. The result is then converted back into the ESCS. Since the computational effort of the transformation is quite high, some approximations are used.

### 2.4.4. Q-Learning

The Q-Learning algorithm [14] requires a discrete simulation environment and a discrete action space as well. It attempts to estimate a value known as Q, which symbolizes the expected reward for attempting a certain action for a certain state. The higher is the value of Q, the higher is the expected reward.

Q is learned through the exploration of the environment and the trained algorithm, and returns in feedback in the form of positive or negative reward. Positive reward is synonymous with good behavior, that is what the algorithm must learn, vice versa the negative reward is a symptom of a behavior to be avoided.

The rewards are set in such a way as to join the longest routes, thus encouraging the fastest solutions. In particular, the reward for reaching the arrival point is set to the same value as that assigned to bad behavior, in absolute value. If a higher reward were given upon reaching the target, the vehicle could learn to reach the goal even at the cost of some collision.

The algorithm has 3 hyperparameters alpha ( $\alpha$ ), gamma ( $\gamma$ ), and epsilon  $\epsilon$ :

- $\alpha$  is the learning rate of the algorithm which controls how big of a change the reward is going to affect the new value of Q.
- $\gamma$  is the discount factor of the expected future reward for taking the next best action.
- $\epsilon$  is the ratio of exploration over exploitation meaning, how often does the algorithm try new things versus using the information it has already learned. An epsilon of 0.1 means that the algorithm will try new things 10% of the time and will exploit its known values 90% of the time.

The Q-Value is calculated based on the following equation:

$$Q_{new}(s_t, a_t) = Q_{old}(s_t, a_t) + \alpha[r_t + \gamma \cdot \max(Q(s_{t+1}, a)) - Q_{old}(s_t, a_t)] \quad (2.1)$$

- $Q$  is the estimated expected value/reward for performing an action for a given state.
- $Q_{new}$  is the new estimated value of  $Q$ .
- $Q_{old}$  is the value of  $Q$  from the previous time-step.
- $s_t, a_t, r_t$  are the state, action, and reward for a certain timestep  $t$
- $\max(Q(s_{t+1}, a))$  is the maximum expected reward of the next state for the current action.



# 3 | Path planning algorithm: design and implementation

The core of the thesis is the implementation of the path planning algorithm. After some researches it was chosen the A\* among all, because it is one of the bestknown path planning algorithms, moreover it has a wide range of usage and it is highly flexible. It is an heuristic search algorithm, which aims to find a path from the start node to the goal node with the smallest cost by searching among all possible paths. Heuristic information related to the characteristics of the problem is utilized to guide its performance, meaning that it is superior to other blind search algorithms.

## 3.1. The A\* algorithm

A-star (also referred to as A\*) is one of the most successful search algorithms to find the shortest path between nodes or graphs. It is an informed search algorithm, as it uses information about path cost and also uses heuristics to find the solution. A\* achieve optimality and completeness: two valuable property of search algorithms. This algorithm is a popular graph traversal path planning algorithm. The A\* algorithm is similar to Dijkstra's because it is a Dijkstra algorithm modification, in a sense. A\* aims to find a shortest path and guides its search towards the shortest path states by a heuristic function. Therefore, the efficiency of that algorithm is better than Dijkstra's.

Many works of literature[4, 5, 22] show that the A\* algorithm has been widely used in many fields, such as transportation industry, and with the development of artificial intelligence, it has been improved, including automated guided vehicle (AGV) or unmanned surface vehicle (USV) path planning and robot path planning . The A\* is simpler and uses less search node, which means less computation than some other path planning algorithms, so it lends itself to more constrained scenarios.

The A\* algorithm needs to search for the shortest path in a map and through a heuristic function for guidance. The heuristic function mainly includes Manhattan distance (3.1),

Euclidean distance (3.2), Chebyshev distance (3.3) and Diagonal distance (3.4).

$$h_n = |x_a - x_b| + |y_a - y_b| \quad (3.1)$$

$$h_n = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \quad (3.2)$$

$$h_n = \max(|x_a - x_b|, |y_a - y_b|) \quad (3.3)$$

$$h_n = |x_a - x_b| + |y_a - y_b| + (\sqrt{2} - 2)\min(|x_a - x_b|, |y_a - y_b|) \quad (3.4)$$

The Manhattan distance is mostly used for a 4-direction search. In fact in the case of an 8- direction search there are too many nodes to traverse. When the Euclidean distance is used, the estimated function value is very similar when the path is close to the goal node, and the node selection may be unclear. When the diagonal distance is used, if there are few obstacles, it is not easy to obtain the shortest path near the goal node. When the Chebyshev distance is used, only the maximum value of the abscissa difference or the ordinate difference is used, and the path always starts from a straight line and then a diagonal. If the search path encounters an obstacle, the shortest path is not obtained [5].

### 3.1.1. The working principles

Each time A\* enters a node, it calculates the cost,  $f(n)$  ( $n$  being the neighboring node), to travel to all of the neighboring nodes, and then enters the node with the lowest value of  $f(n)$  (3.5). Here we will solve a maze problem. We have to find the shortest path in the maze from a start node to the end node. For this problem, there are four moves (left, right, up, and down) from a maze position provided a valid step is available. Referring to the Figure 3.1, in red square positions no movement is allowed, indeed, in start position, only down motion is available since up and left move are blocked by the wall, while for the right is a red square position thus no movement allowed.

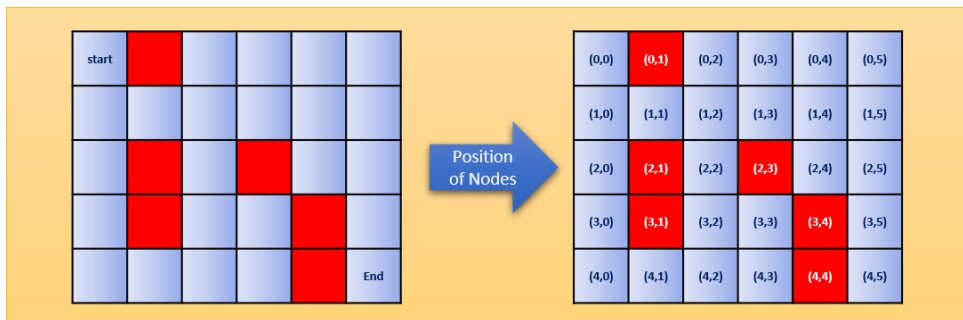


Figure 3.1: the maze problem

Entering in the details, we consider a square grid having many obstacles and a starting cell and a target cell. We want to reach the target cell (if possible) from the starting as quickly as possible. Here A\* Search Algorithm comes to the rescue. What it does is that at each step it picks the node according to a value 'f'(3.5) which is a parameter equal to the sum of two other parameters 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and process that node/cell.

In particular g is the cost to move from the starting point to a given square on the grid, following the path generated to get there and h, instead, is the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don't know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.).

$$f_n = g_n + h_n \quad (3.5)$$

In the equation (3.5) g(n) is the cost of the path from the start node to the current node n, h(n) is the cost of the path from node n to the goal node through the selected sequence of nodes, and h(n) is the heuristic function of the A algorithm. f(n) is denoted as the evaluation function of node n. This sequence ends in the actually evaluated node. Each adjacent node of the actually reached node is evaluated by f(n). The node with the minimum value of f(n) is chosen as the next node in the sequence. The advantage of the A\* algorithm is that other distances can be adopted, modified, or added as standard distances. The searching process has two list, OPEN and CLOSED list. In the OPEN list we put the node that are not yet evaluated, once we evaluate the node, it is put in the CLOSED list. At the end this list is full of all the evaluated node, so we choose the node with the minimum cost and, proceeding backwards, we can construct the optimal path. Starting from this type of algorithm, have been made some improvements which will be described in the rest of the chapter.

## 3.2. How to improve

### 3.2.1. First improvement

A first improvement for the algorithm is proposed to solve the problem of slow search speed and low efficiency of the algorithm. In the rasterized maps, a set of nodes centered on a certain point is constructed. When the set of nodes contains obstacle nodes, this node is defined as an untrusted point, it is pre-inserted in the CLOSED list, and will not be searched or evaluated. This improvement is used to improve the efficiency of the A\* algorithm.

### 3.2.2. Second improvement

Another improvement is how is constructed the OPEN list: when a node that is already in the OPEN list is evaluated, i.e. a node that was traversed before, if this node has a lower cost than the previous with the same coordinates, the node with the higher cost was deleted and it is subscribed with the new node, This process speed the search for the optimal path, because at the end of the creation of the OPEN list the algorithm traverse backwards and search in the OPEN list the parent nodes of the current node. It starts from the goal node and it select iteratively the parent of the current node with the minimum cost until the starting point is reached. All this point are inserted in an array to graphic the path and draw it on the map.

### 3.2.3. Third improvement

The path generated by the traditional A\* algorithm may be very close to the obstacle. If the path is adopted by a mobile robot as the planning path, the mobile robot will have a high risk of collision with obstacles. Therefore, it is indispensable to consider maintaining an appropriate distance from obstacles during path planning. The appropriate distance is intended as expansion distance, it means keeping an extra space around the obstacles as the safe distance during path planning. Robots adapt rasterized maps as path planning maps. Expansion distance adopts a grid as the basic unit to expand outward around obstacles. Expansion distance is the shortest distance that the path is allowed to approach obstacles. The value of the expansion distance is determined by parameters such as the speed of the robot, the size of the robot model, and the number of grids. The nodes of expanded distance will not be visited together with the obstacles during the path planning. Expansion distance will be used as a “collision buffer” between the robot and the obstacles, which can effectively reduce the collision risk of the robot during the travel

process. Therefore, expansion distance can increase the robustness of the algorithm. The superiority of the expansion distance is not limited to enhancing the robustness of the algorithm, it is also effective in improving the efficiency of the algorithm. Since the expanded nodes are no longer visited by the algorithm, expansion distance is equivalent to reducing the map scale in some sense. The total number of nodes that the algorithm needs to traverse is reduced accordingly[4]. Therefore, expansion distance can improve the efficiency of the algorithm. The schematic diagram of expansion distance is shown in Figure 3.2.

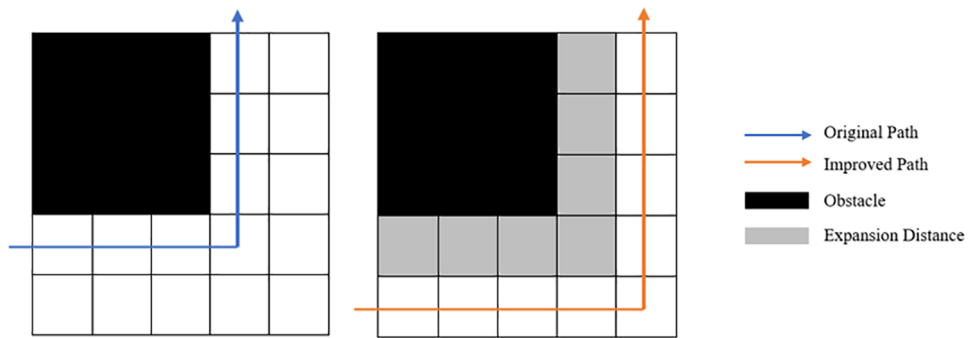


Figure 3.2: expansion distance

So, the expansion distance is applied to the warehouse map that is used for the simulation of the algorithm

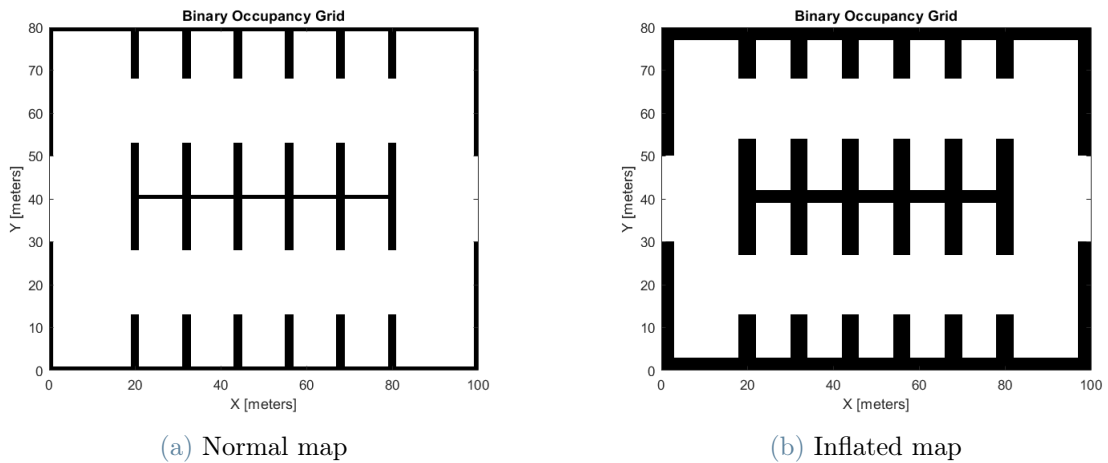


Figure 3.3: The map before and after the expansion distance

### 3.3. Implementation

The algorithm is implemented in Matlab, it is tested with different map, start and goal location and with the adding of some obstacles. In the following is presented the pseudocode of the algorithm

---

#### Algorithm 3.1 A Star modified

---

```

1: Input: Start and goal node, map
2: Output: Optimal path, Evaluated nodes
3: Initialize OPEN, path cost, g and h function
4: insert the node obstacles in the CLOSED list
5: while Current node  $\neq$  Target Node do
6:   Update OPEN list with the successors
7:   subscribe if there is a node with a minor cost
8:   if not improved any node then
9:     add a new element in OPEN
10:  end if
11:  Find node with smallest cost
12:  Update g and f
13:  Move the node in the CLOSED list
14: end while
15: Reconstruct backward Optimal Path from CLOSED list

```

---

As previous discussed the three improvements makes the A-star more efficient and faster, the results, shown in table (3.1), are obtained comparing the new algorithm with a basic a star with a warehouse map like:

**A-basic vs Modified**

	A* basic	A* modified
<b>Path lenght [m]</b>	102.11	103.11
<b>Node searched</b>	1630	1585
<b>Path Node</b>	95	95
<b>Elapsed time [s]</b>	0.000125	0.000036

Table 3.1

It can be seen that, while the node searched are reduced by a small amount, i.e. not so significant, the elapsed time of the modified algorithm is about 4 times less. This means an algorithm that is almost 4 times faster than the normal one.

Figure 3.4 and Figure 3.5 show the optimal path draw on two different map and with an obstacle on the route.

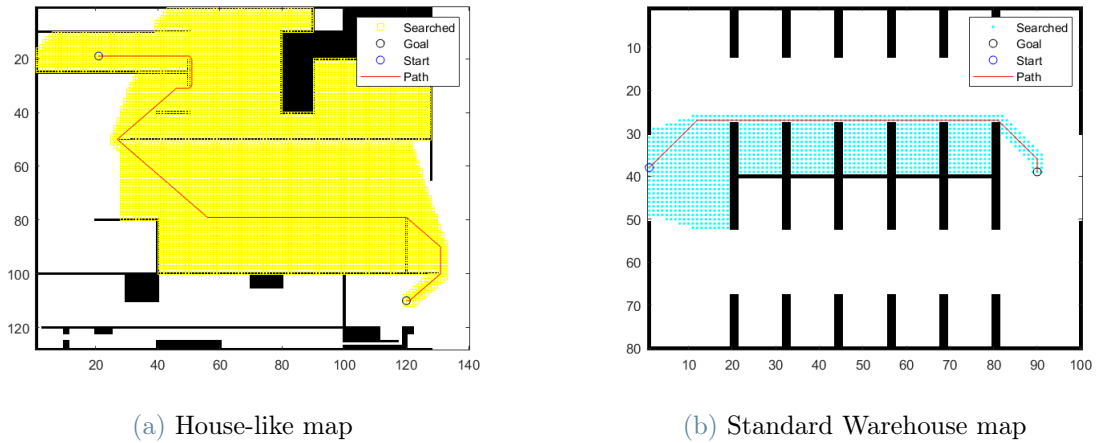


Figure 3.4: Two different map

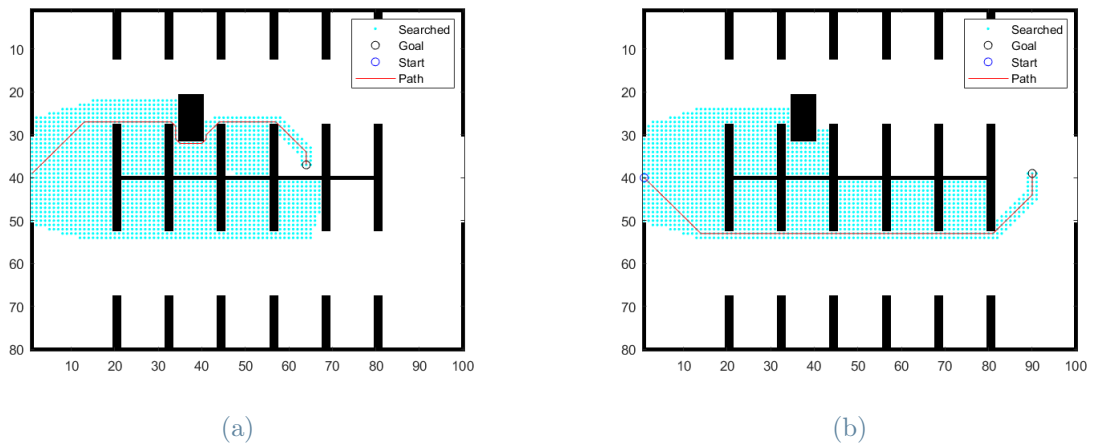


Figure 3.5: Two different goal

### 3.3.1. Map inflated

A second phase of simulation was carried out with the map inflated, as represented in the Figure 3.6a and Figure 3.6b, with an obstacle on the route. These figure show how the expansion distance is made graphically.

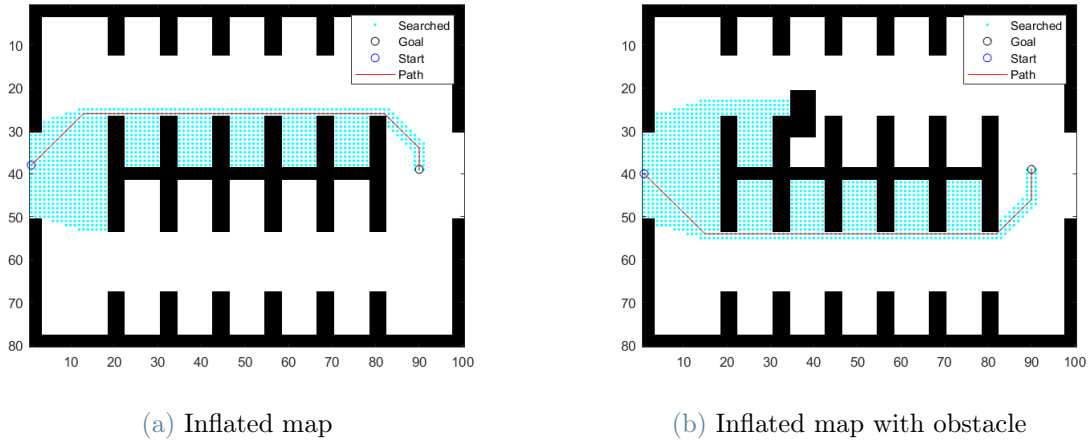


Figure 3.6

If we simulate an action, like picking some furniture in a lot, it seems that the classical heuristic function used before is not enough, we can do better. Infact changing the  $h(n)$  and the  $g(h)$  we have an increasement in terms of efficiency: we evaluate much less node reaching the goal with less computational burden.

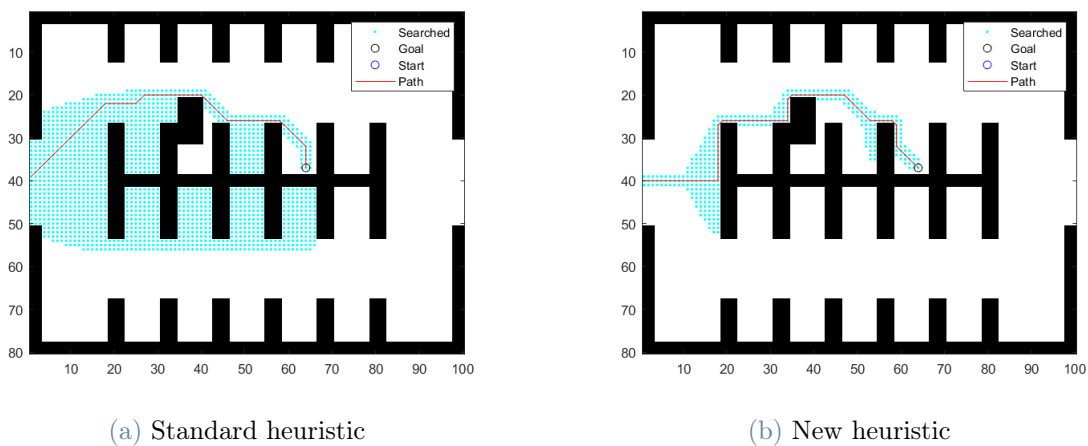


Figure 3.7: Two different map



To reach this result it is combined a  $g(n)$  function with a new  $h(n)$ , keeping the  $g(n)$  as before (3.1). Based on the analysis of the heuristic function and the experience of repeated experiments performed by Huanwei Wang et al. [5], it is proposed a new heuristic function, as shown in Formula (3.6). The optimized heuristic function is composed of the Euclidean distance and Chebyshev assignment, according to different weights. This heuristic function not only takes into account the advantages of the Euclidean distance and Chebyshev distance, but also weakens their disadvantages. The first part of the heuristic function ensures that the path is searched along the diagonal direction, and the latter part effectively reduces the number of nodes that are searched around after encountering obstacles in the search process.

$$h_n = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} + \max(|x_a - x_b|, |y_a - y_b|) \cdot 2 \quad (3.6)$$

	Inflated Map	Inflated Map New h(n)	Normal Map
<b>Path lenght [m]</b>	90.97	92.63	103.11
<b>Node searched</b>	1343	436	1585
<b>Path Node</b>	97	112	95
<b>Elapsed time [s]</b>	0.00034	0.00030	0.000036

Table 3.2: Normal vs Inflated

The table (3.2) summarize the main evaluation parameter and it is referred to Figure 3.7. As it can be seen in the two inflated case the path lenght is reduced with respect to the normal case. Talking about the different heuristic used in the inflated case, the "hybrid" heuristic makes the path slightly longer and thus it utilized more node to construct the path but it evaluated much more nodes working in a smarter way, it reaches the goal utilizing almost one hundred nodes less, that is a relevant number in terms of computational effort.

### 3.3.2. Comparison with Matlab planner

The efficiency of the implemented algorithm was tested by comparison with two Matlab planner, `plannerAStarGrid` and `plannerHybridAStar`. The first performs A\* search on the occupancy map and finds obstacle-free shortest path between the given start grid and goal grid guided by heuristic cost. The Hybrid A\* Path Planner, instead, generates a smooth path in a given 2-D space for vehicles with nonholonomic constraints. Thus the latter takes in consideration also the orientation of the vehicle, it is a more completed and complex algorithm. The implemented algorithm does not take in consideration any constraints or kinematics of the vehicles, thus is more similar to the A\* Grid planner. The three planner were tested with the same map, warehouse like, adding an obstacle and changing the goal location, Figure 3.8.

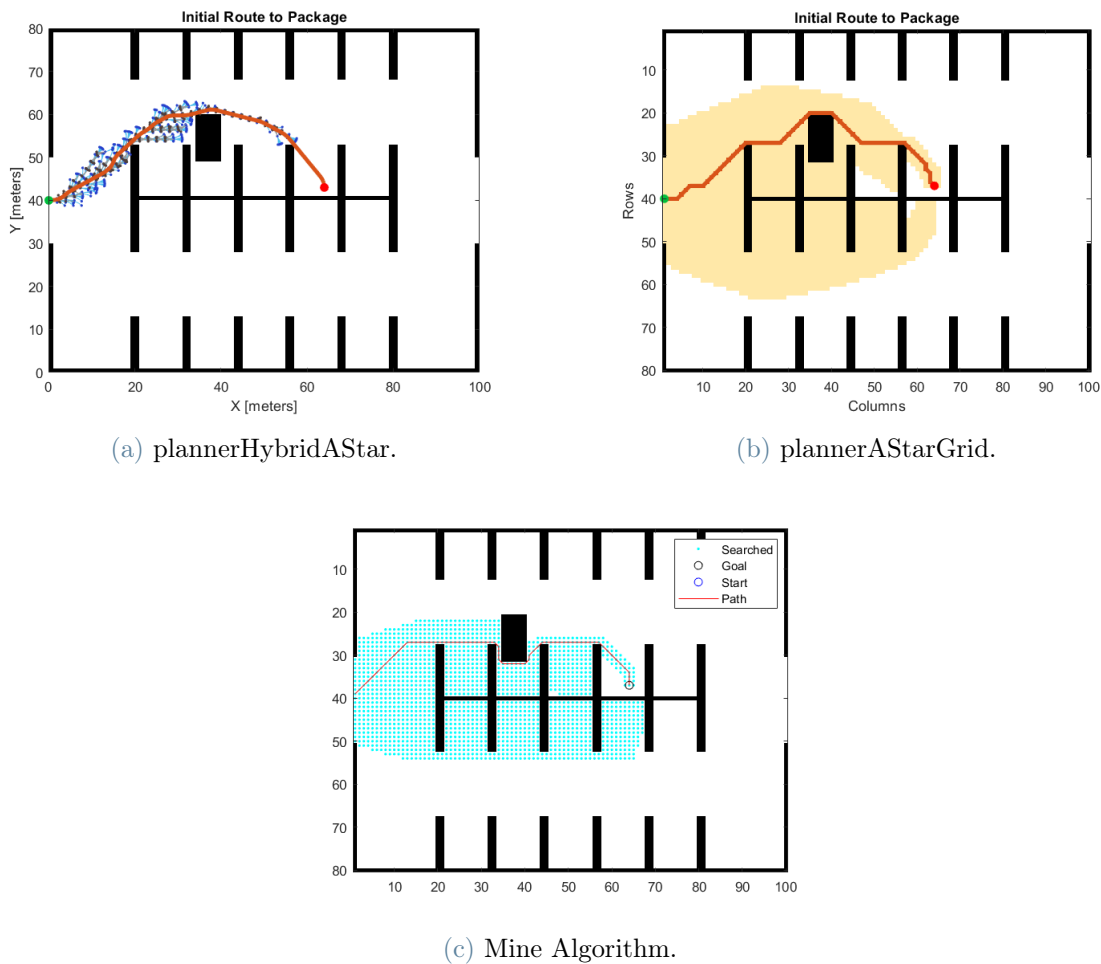


Figure 3.8: Comparison with Matlab planners



	Hybrid	Grid	Mine	Mine Inflated
<b>Path length [m]</b>	81.100	80.083	80.355	91.799
<b>Path Node</b>	82	67	71	87
<b>Searched Node</b>	1038	1742	1631	367
<b>Elapsed time [s]</b>	0.000130	0.000057	0.000041	0.000033

Table 3.3: Comparison between planner

The algorithm is tested, as discussed before, with different goal location, in the Figure 3.10 there are the outcome of the simulation performed with the two Matlab planner and with the implemented algorithm, Figure 3.10c and Figure 3.10d.

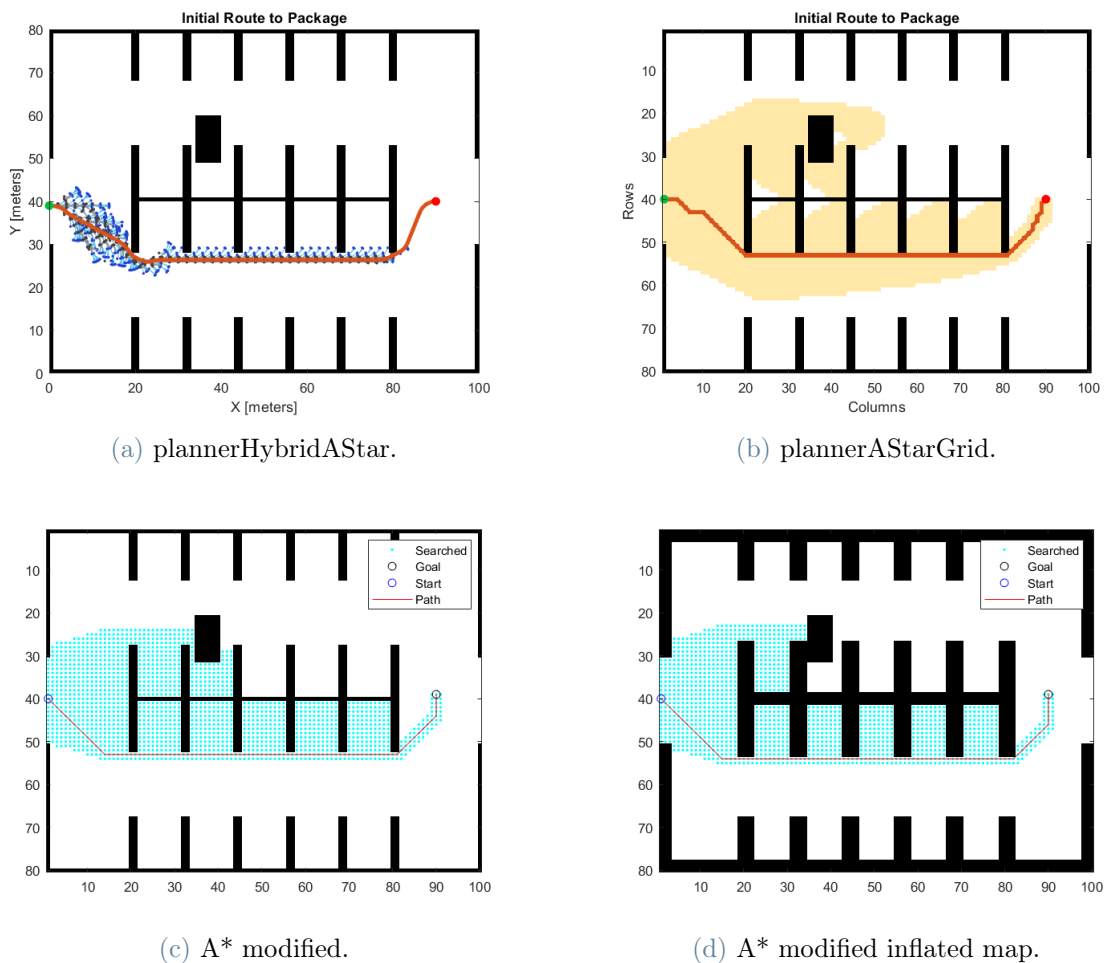


Figure 3.10

	Hybrid	Grid	Mine	Mine Inflated
<b>Path lenght [m]</b>	104.14	102.53	103.11	105.11
<b>Path Node</b>	102	95	95	97
<b>Searched Node</b>	1207	1725	1586	1343
<b>Elapsed time [s]</b>	0.00028	0.000043	0.000032	0.000027

Table 3.4: Comparison between planner, different goal

After all the simulation is clear that the implemented algorithm is comparable with the planner AStarGrid because they are similar and both doesn't take in consideration of the kinematics of the vehicle. Infact there are no mention to the type of vehicle, the turning radius and other parameter like linear and angular velocity. From both the tables (3.3) and (3.4) turns out that the improvements performed on the algorithm give a visible advantage in terms of computational cost and the elapsed time. Infact the elapsed time with the map inflated is the shorter and the node evaluated are about one hundred less than the other planner.

To ensure that the algorithm takes into account also the kinematics, it is necessary to make some considerations. The next chapter takes care of this problem simulating the algorithm in a Simulink-ROS environment, thanks to that it is introduced the type of vehicle we want to simulate and also a system to avoid unexpected obstacle on the route.



# 4 | Simulink and ROS simulation

This chapter will illustrate in details the way in which the path following and obstacle avoidance strategies have been implemented. The goal of this phase is to develop the aforementioned logics so that they can operate in parallel and in real time. What is already known is the results from the path planning phase, i.e. the ideal trajectory to be followed consisting of a series of points in the plane characterized by the coordinates  $x$ ,  $y$ ,  $\theta$ . Now, instead, what we intend to obtain are the longitudinal speed and the yaw speed of the vehicle, from which it is possible to obtain the commands to be given to the vehicle. Furthermore, from this moment on, the implementation will take place directly in the Simulink environment with the support of the ROS toolbox.

## 4.1. Robot Operating System

Robot Operating System (ROS or ros) is an open-source robotics middleware suite. Although ROS is not an operating system (OS) but a set of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages [12]. ROS has two parts:

**The operating system side** , which provides standard operating system services such as:

- hardware abstraction
- low-level device control
- implementation of commonly used functionality
- message-passing between processes

- package management

**A suite of user contributed packages** that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc. In ROS are available single-purpose executable programs like sensor driver(s), actuator driver(s), mapper, planner, UI, etc. It has nodes written using a ROS client library `roscpp`, the C++ client library and `rospy`, python client library. Nodes can publish or subscribe to a Topic and can also provide or use a Service. A topic is a name for a stream of messages with a defined type for example, data from a laser range-finder might be sent on a topic called `scan`, with a message of type `LaserScan` Nodes communicate with each other by publishing messages to topics.

ROS provides the tools, libraries, and capabilities that one need to develop robotics applications. It's open-source, and isn't exclusive, there is no need to choose between ROS or some other software stack; ROS easily integrates with existing software to bring its tools to the problem. Infact ROS is Multi domain ready for use across a wide array of robotics applications, from indoor to outdoor, home to automotive, underwater to space, and consumer to industrial.

Software in the ROS Ecosystem can be separated into three groups: language- and platform-independent tools used for building and distributing ROS-based software; ROS client library implementations such as `roscpp`, `rospy`, and `roslisp`; packages containing application-related code which uses one or more ROS client libraries.

## 4.2. Path following strategy

Initially, the tracking strategy was implemented using the 'Robotics System Toolbox: Mobile Robot Algorithms' section in the Simulink library, selecting in particular the 'Pure Pursuit' block, Figure 4.1 below.



### 4.2.1. Pure Pursuit block

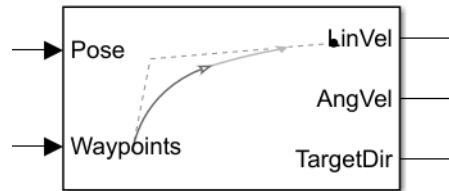


Figure 4.1: Pure pursuit block

This, in particular, derives the linear and angular velocity commands to track a given path, it takes in input a set of waypoints and the current position of the robot, in this particular case the current pose is given by ROS messages, Figure 4.2.

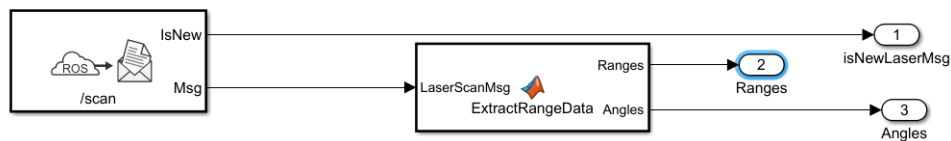


Figure 4.2: laser scan block

The 'Pure Pursuit' object is a 'controller' which then updates the speed output, based on updated information about the position that receives in input. The characteristics of the robot, on the other hand, are represented in this case by two properties to set in the Simulink block interface, namely the 'DesiredLinearVelocity' and 'MaxAngularVelocity'. The first, in  $m/s$ , implies a constant linear velocity of the medium, independent of the angular velocity; the second, in  $rad/s$ , instead constitutes a limit on the yaw rate of the vehicle cannot overcome. Another property, the 'LookaheadDistance', then determines a look-ahead point on the route and how far it is positioned, constituting a local target for the robot, and the angular velocity comes determined on the basis of that point. As the Figure 4.3 shows, the actual path does not match the direct line between waypoints.

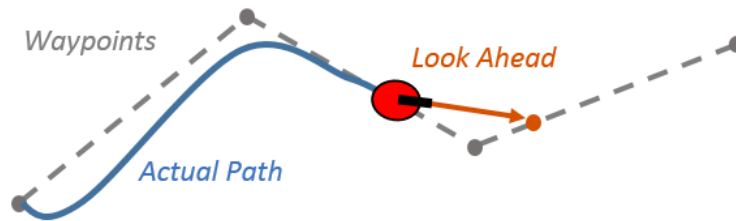


Figure 4.3: lookahead distance

Changing the viewing distance has a significant impact on performance of the algorithm: a greater LookaheadDistance (Figure 4.4) produces a more trajectory uniform for the robot, but can cause the vehicle to cut corners along the way. Conversely, a low value of this property can generate fluctuations, as can be seen in the upper part of Figure 4.4. Thus when the path is plotted the unstable behavior arises.

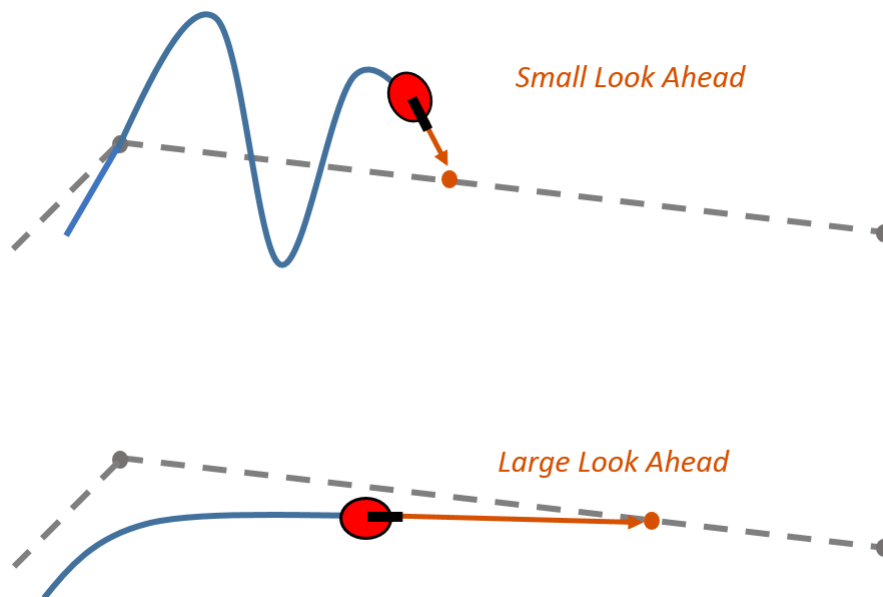


Figure 4.4: example of small and large look ahead

It is therefore clear that the 'Pure Pursuit' controller has some limitations: first everything does not allow to exactly follow the profile of the desired trajectory between two points, but you need to tune the parameters to improve performance and converge to the ideal path over time. It also does not allow the robot to be stabilized exactly at the end point, but an appropriate threshold distance must be set that entails stopping in the immediate vicinity of the target.

### 4.2.2. Vector field histogram

As already mentioned, in parallel with the path follower, a logic has been developed that made possible to avoid any obstacles encountered along the route in real time when they were not initially foreseen. Also at this juncture, from the 'Robotics System Toolbox: Mobile Robot Algorithms' from the Simulink library has been downloaded on 'Vector Field Histogram' block, Figure 4.5.

The vector field histogram (VFH) algorithm computes obstacle-free steering directions for a robot based on range sensor readings. Range sensor readings are used to compute polar density histograms to identify obstacle location and proximity. Based on the specified parameters and thresholds, these histograms are converted to binary histograms to indicate valid steering directions for the robot. The VFH algorithm accounts for robot size and turning radius to output a steering direction for the robot to avoid obstacles and follow a target direction. To calculate steering directions, you must specify information about the robot size and its driving capabilities. The VFH algorithm requires only four input parameters for the robot. These parameters are properties of the controllerVFH object: RobotRadius, SafetyDistance, MinTurningRadius, and DistanceLimits.

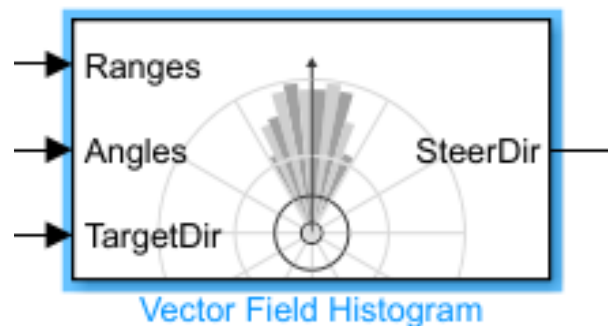


Figure 4.5: VFH block

## VFH properties

**RobotRadius** specifies the radius of the smallest circle that can encircle all parts of the robot, see Figure 4.6. This radius ensures that the robot avoids obstacles based on its size.



Figure 4.6: Robot radius

**SafetyDistance** optionally specifies an added distance on top of the RobotRadius (Figure 4.7). You can use this property to add a factor of safety when navigating an environment.

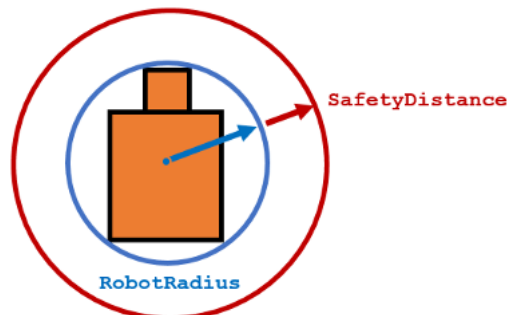


Figure 4.7: Safety distance

**MinTurningRadius** specifies the minimum turning radius for the robot traveling at the desired velocity, Figure 4.8. The robot may not be able to make sharp turns at high velocities. This property factors in navigating around obstacles and gives it enough space to maneuver.

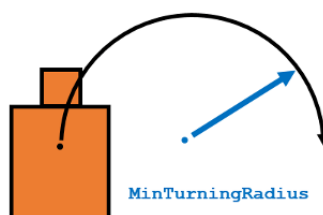


Figure 4.8: Minimum turning radius

**DistanceLimits** specifies the distance range that you want to consider for obstacle avoidance. You specify the limits in a two-element vector, [lower upper]. The lower limit is used to ignore sensor readings that intersect with parts on the robot, sensor inaccuracies at short distances, or sensor noise. The upper limit is the effective range of the sensor or is based on your application. You might not want to consider all obstacles in the full sensor range.

### Sensors parameter

All information about the range sensor readings assumes that your range finder is mounted in the center of your robot. If the range sensor is mounted elsewhere, transform your range sensor readings from the laser coordinate frame to the robot base frame.

**Cost Function Weights** are used to calculate the final steering directions. The VFH algorithm considers multiple steering directions based on your current, previous, and target directions. By setting the `CurrentDirectionWeight`, `PreviousDirectionWeight`, and `TargetDirectionWeight` properties, you can modify the steering behavior of your robot. Changing these weights affects the responsiveness of the robot and how it reacts to obstacles. To make the robot head towards its goal location, we have to set `TargetDirectionWeight` higher than the sum of the other weights. This high `TargetDirectionWeight` value helps to ensure the computed steering direction is close to the target direction. Depending on the application, the weights have to be tuned accordingly.

**Histogram Properties** The VFH algorithm calculates a histogram based on the given range sensor data. It takes all directions around the robot and converts them to angular sectors that are specified by the `NumAngularSectors` property. This property is non-tunable and remains fixed once the `controllerVFH` object is called. The range sensor data is used to calculate a polar density histogram over these angular sectors.

Using a small `NumAngularSectors` value can cause the VFH algorithm to miss smaller obstacles. Missed obstacles do not appear on the histogram. The `HistogramThresholds` property is a two-element vector that determines the values of the masked histogram, specified as [lower upper]. Polar obstacle density values higher than the upper threshold are represented as occupied space (1) in the masked histogram. Values smaller than the lower threshold are represented as free space (0). Values that fall between the limits are set to the values in the previous binary histogram, with the default being free space (0). The masked histogram also factors in the `MinTurningRadius`, `RobotSize`, and `SafetyDistance`.

### 4.3. Overall scheme

At this point all that remains is to make a connection between the path strategy following and obstacle avoidance, so that the overall system operates in parallel. This is possible by enabling the 'TargetDir' exit from the 'Pure Pursuit' block and connecting it directly to the 'Vector Field Histogram' block: the objective direction determined by the path follower constitutes in fact one of the three inputs of the algorithm for obstacle avoidance. Therefore the block diagram thus described is depicted in Figure 4.9:

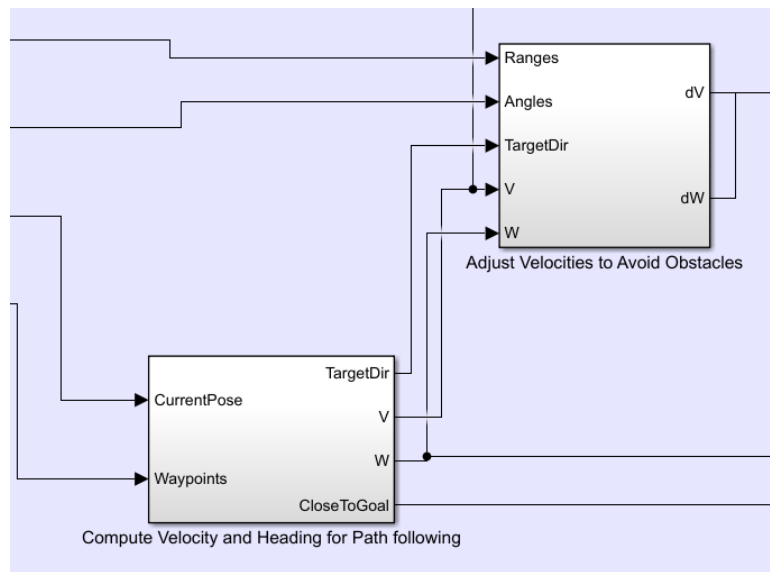


Figure 4.9

The two blocks presented in Figure 4.9 are connected with the input block and the output block, as it can be seen in the complete scheme, see Figure 4.17. This scheme is a ROS-based simulator for a differential-drive robot. The simulator receives and sends messages on the following topics:

- It receives velocity commands, as messages of type `geometry_msgs/Twist`, on the `/mobile-base/commands/velocity` topic
- It sends ground truth robot pose information, as messages of type `nav_msgs/Odometry`, to the `/ground-truth-pose` topic
- It sends laser range data, as messages of type `sensor_msgs/LaserScan`, to the `/scan` topic

This model is used to implement the path following with obstacle avoidance algorithm. The model is divided into four subsystems:

### 4.3.1. Process input

The 'Inputs' subsystem ( Figure 4.10) processes all the inputs to the algorithm. There are two subscribers to receive data from the simulator. The first subscriber receives messages sent on the /scan topic. The laser scan message is then processed to extract scan ranges and angles. The second subscriber receives messages sent on the /ground-truth-pose topic. The (x,y) location and Yaw orientation of the robot is then extracted from the pose message. The path is specified as a set of waypoints. This example uses a 3x2 constant input. You can specify any number of waypoints as an Nx2 array. To change the size of the path at run-time, you can either use a variable sized signal or use a fixed size signal with NaN padding. This example uses a fixed size input with NaN padding for the waypoints that are unknown.

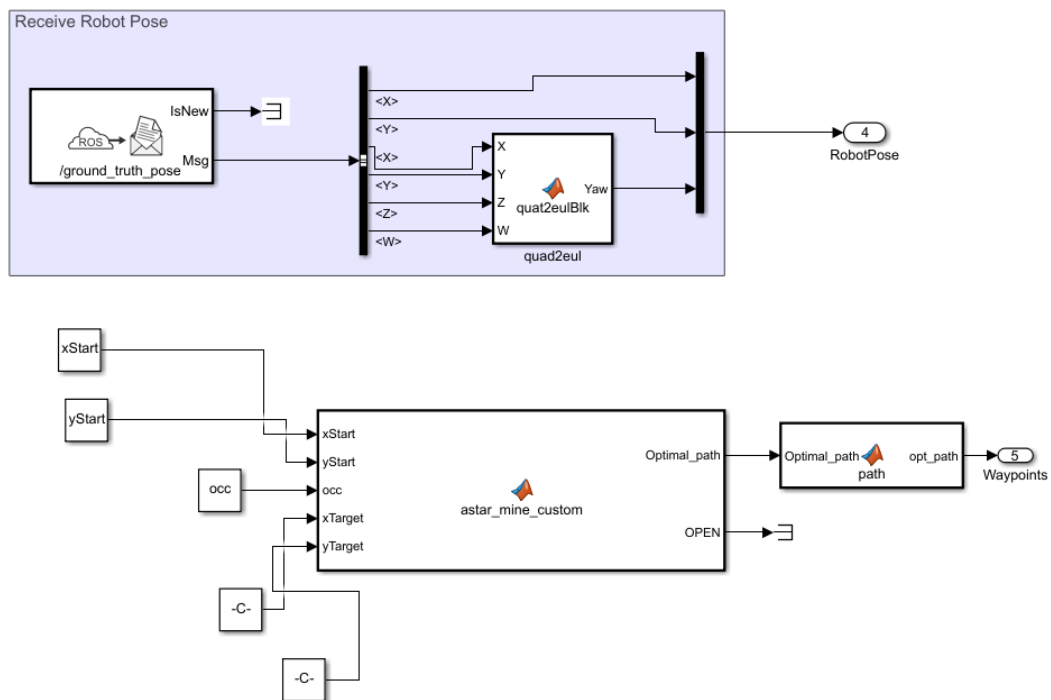


Figure 4.10: inside input block

### 4.3.2. Compute Velocity and Heading for Path Following

The 'Compute Velocity and Heading for Path Following' subsystem (Figure 4.11) computes the linear and angular velocity commands and the target moving direction using the Pure Pursuit block. This subsystem also compares the current robot pose and the goal point to determine if the robot is close to the goal.

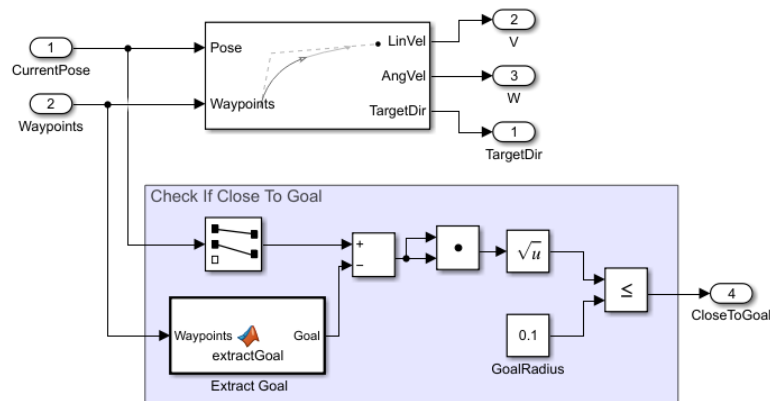


Figure 4.11: inside the block

### 4.3.3. Adjust Velocities to Avoid Obstacles

The 'Adjust Velocities to Avoid Obstacles' subsystem computes adjustments to the linear and angular velocities computed by the path follower. The Vector Field Histogram block uses the laser range readings to check if the target direction computed using the Pure Pursuit block is obstacle-free or not based on the laser scan data. If there are obstacles along the target direction, the Vector Field Histogram block computes a steering direction that is closest to the target direction and is obstacle-free. The Vector Field Histogram block is also located in the Mobile Robot Algorithms sub-library. The steering direction is NaN value when there are no obstacle-free directions in the sensor field of view. In this case, a recovery motion is required, where the robot turns on-the-spot until an obstacle-free direction is available. Based on the steering direction, this subsystem computes adjustments in linear and angular velocities, scheme in Figure 4.12.



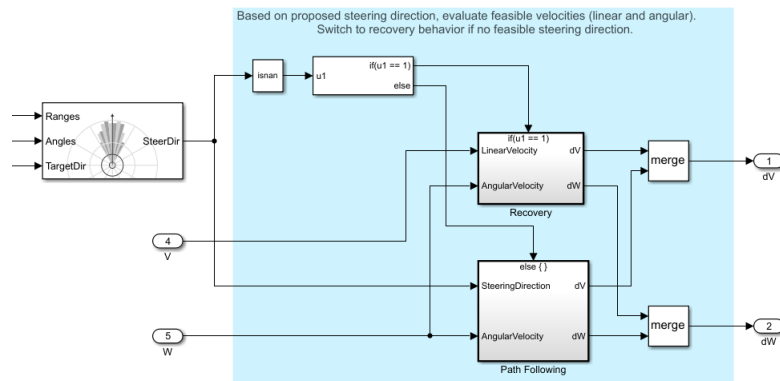


Figure 4.12: Recovery/following scheme

### 4.3.4. Outputs block

The 'Outputs' subsystem (Figure 4.13) publishes the linear and angular velocities to drive the simulated robot. It adds the velocities computed using the Pure Pursuit path following algorithm with the adjustments computed using the Vector Field Histogram obstacle avoidance algorithm. The final velocities are set on the geometry\_msgs/Twist message and published on the topic /mobile-base/commands/velocity. This is an enabled subsystem which is triggered when new laser message is received. This means a velocity command is published only when a new sensor information is available. This prevents the robot from hitting the obstacles in case of delay in receiving sensor information.

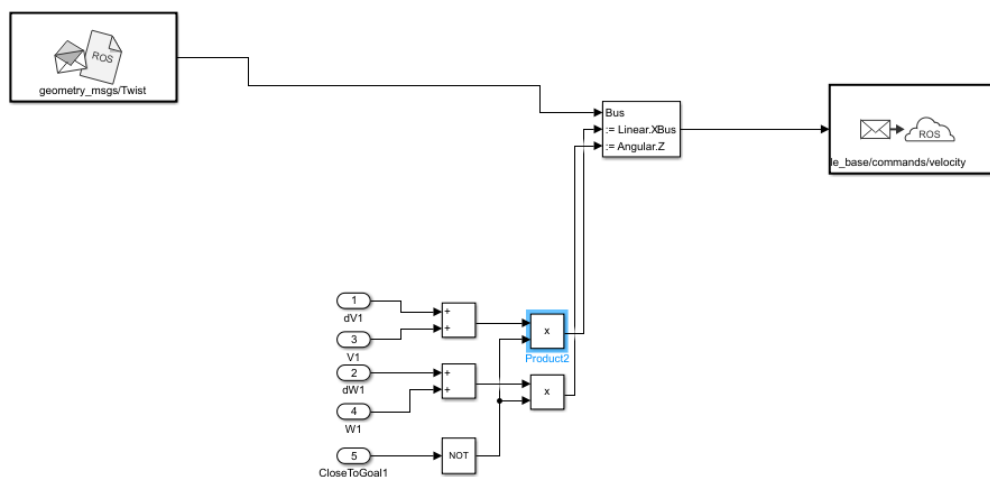


Figure 4.13: Inside output block

### 4.3.5. Kinematics

After having defined an appropriate control logic, all that remains is to develop a model that can simulate the dynamics of the vehicle in a realistic way. In this sentence it is expected to obtain in real time the response to the commands given by the logic control, i.e. how these influence the progress and orientation of the vehicle. The outputs of the robot  $x, y, \theta$  are update in real time, necessary for the previous phase to determine the new values of longitudinal velocity 'v' and angular 'w', always trying to follow the desired trajectory. At first all this was implemented in Simulink, starting first from a simplified model of vehicle with differential drive, later replaced with a more complicated one. To best mimic the vehicle behaviour it was implemented an ackerman-steering model.

### Differential drive kinematics

According to this model, the vehicle can be traced back to a vehicle with two wheels independent used to control longitudinal and angular velocity. This therefore implies that the two wheels can travel with different angular speeds.

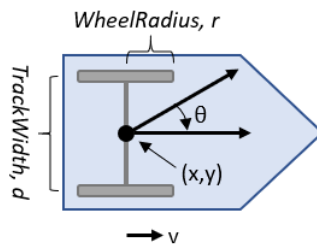


Figure 4.14: differential drive robot

By changing the speed of the wheels to have different movements, the robot rotates around to a point lying along the common axis of the two wheels: it is known as the center of instantaneous rotation (CIR). The angular velocity  $\Omega$  produced by this rotation is known and the following equations hold:

$$\begin{cases} \Omega(D + \frac{d}{2}) = V_r \\ \Omega(D - \frac{d}{2}) = V_l \end{cases} \quad (4.1)$$

where  $D$  is the distance from the center of instantaneous rotation to the midpoint between the wheels,  $d$  is the track width,  $V_r$  and  $V_l$  are respectively the linear speeds of the right wheel and of the left wheel. For each instant of time it is therefore possible to obtain:

$$\begin{cases} D = \frac{d(V_l+V_r)}{2(V_l-V_r)} \\ \Omega = \frac{(V_r-V_l)(V_r-V_l)}{d} \end{cases} \quad (4.2)$$

For the system thus defined, the trajectory of the vehicle depends solely on the speeds of the wheels: 1. if  $V_l = V_r$  the motion will be linear; 2. if  $V_l = -V_r$  the robot rotates around its own central axis; 3. if the speeds are not equal the resulting trajectory will be the result of one combination of translational and rotational motion. To better understand what has just been said, let's assume that the vehicle is in one given position  $(x, y)$  forming an angle  $\theta$  with the X axis. After a certain interval of time the new position will be  $(x', y')$  and the new angle will be  $\theta'$ . Thus:

$$CIR = [x - D\sin(\theta), y + D\cos(\theta)] \quad (4.3)$$

In this case, the reference block taken from the MATLAB library 'Mobile Robotics Simulation Toolbox', returns in output the current pose of the robot, but requires the angular speeds of the two wheels as input. What is known however from the Processing phase of the commands are the longitudinal speed and yaw rate of the vehicle, ideally applied at the center of mass. It is therefore necessary to apply the first inverse kinematics equations to derive wheel speeds; in the following are reported the direct and inverse kinematics relationships between the various speeds in question.

Direct Kinematics:

$$\begin{cases} v = \frac{r}{2}(\omega_R + \omega_L) \\ \omega = \frac{r}{d}(\omega_R - \omega_L) \end{cases} \quad (4.4)$$

Inverse Kinematics:

$$\begin{cases} \omega_L = \frac{1}{r}(v - \frac{\omega d}{2}) \\ \omega_R = \frac{1}{r}(v + \frac{\omega d}{2}) \end{cases} \quad (4.5)$$

### ‘Four-Wheel Steering’ Kinematics

The theory behind this model refers to a vehicle with four wheels that they can all be driven and steered independently. For simplicity we assume the so-called ‘Ackermann steering’ (Figure 4.15a), so that the medium can be approximated to a system having two wheels, commonly known as the ‘bicycle model’ (Figure 4.15b).

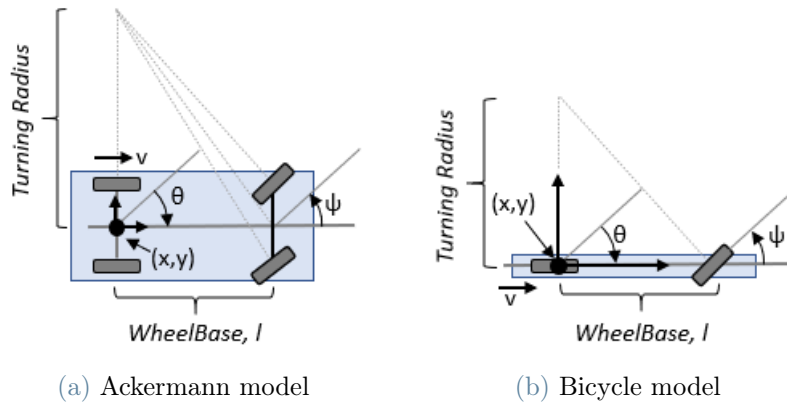


Figure 4.15

To describe the problem we also assume the validity of the assumptions of planar motion, rigid body and non-slip wheels; the vehicle ideally moves on a plane horizontal where an X-Y coordinate system is fixed, the model in the global frame is depicted in Figure 4.16.

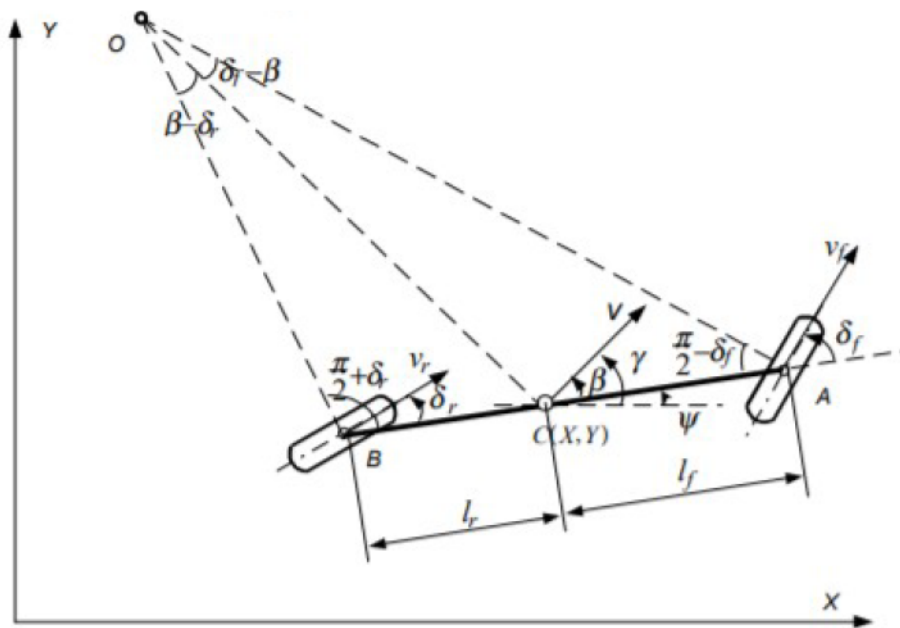


Figure 4.16: Bicycle model in a global frame

Its position is identified by the center of mass C of coordinates (X,Y), in corresponding to which its velocity  $v$  is also defined. The angle  $\Psi$  is the angle between the X axis and the longitudinal axis of the medium AB;  $\gamma$  instead is the one identified between the axis X and the direction of the velocity vector  $v$ .  $\beta$  represents the drift angle, defined between the longitudinal axis AB and the direction of vehicle speed. Also note the radius steering angle  $r$ , i.e. the distance between the center of instantaneous rotation O and the center of mass C. Finally, for the purposes of the problem, it is necessary to identify the steering angles of the wheel front and rear, which from this simplified formulation can be traced respectively to the steering angles of the wheels of each axle of the vehicle a four wheels. So you need to first specify the front wheel speed  $v_f$ , defined at the intersection between the mean plane of the front wheel and the its axis A, and the speed of the rear wheel  $v_r$ , defined analogously with the plane center of the wheel and its axis B. At this point the front steering angle  $\delta_f$  is the angle between the longitudinal axis AB and the direction of  $v_f$ , as well as the steering angle posterior  $\delta_r$  is defined between AB and the direction of  $v_r$ . The kinematic model '4WS' can be expressed as follows.

$$\begin{cases} \dot{X} = v \cos(\Psi + \beta) \\ \dot{Y} = v \sin(\Psi + \beta) \\ \dot{\Psi} = \frac{v \cos \beta (\tan \delta_f + \tan \delta_r)}{l_f + l_r} \end{cases} \quad (4.6)$$

Where:

$$\begin{cases} \beta = \arctan\left(\frac{l_f \tan \delta_r + l_r \tan \delta_f}{l_f + l_r}\right) \\ v = \frac{v_f \cos \delta_f + v_r \cos \delta_r}{2 \cos \beta} \end{cases} \quad (4.7)$$

Here the direct kinematics

$$\begin{cases} v_X = \frac{R}{2} (\omega_f \cos \phi_f + \omega_r \cos \phi_r) \\ v_Y = \frac{R}{2} (\omega_f \sin \phi_f + \omega_r \sin \phi_r) \\ \omega = \frac{R}{L_f + L_r} \omega_f \sin \phi_f - \omega_r \sin \phi_r \end{cases} \quad (4.8)$$

The inputs in this case are the angular speeds of the front and rear wheels  $[\omega_f; \omega_r]$  in  $rad/s$  and the steering angles  $[\phi_f; \phi_r]$  in  $rad$ . The outputs instead are the speeds linear  $v_x$  and  $v_y$  in  $m/s$  and the yaw velocity of the medium  $\omega$  in  $rad/s$ . Given the characteristics of the 'coupled' kinematics envisaged by the bicycle model, MATLAB Toolbox and Simulink propose three different approaches of inverse kinematics:

- 1 Zero sideslip: in the first case it is assumed that the front and rear wheels steer with opposite angles to minimize slippage. The inputs are the speed of feed  $v_x$  and the angular velocity  $\omega$ .

$$\begin{cases} \omega = \frac{v_x}{R \cos \phi} & \omega_r = \omega_f = \omega \\ \phi = \arctan\left(\frac{\omega(L_f + L_r)}{v_x}\right) & \phi_r = -\phi_f = \phi \end{cases} \quad (4.9)$$

- 2 Parallel steering: steering angles are assumed to be equal both at the front and at the rear, so that the vehicle moves without rolling. The inputs are the linear velocity  $v_x$  and  $v_y$ .

$$\begin{cases} v = \frac{\text{sign}(v_x)}{R} \sqrt{v_x^2 + v_y^2} & \omega_r = \omega_f = \omega \\ \phi = \arctan\left(\frac{v_y}{v_x}\right) & \phi_r = -\phi_f = \phi \end{cases} \quad (4.10)$$

- 3 Front steering: in this case  $\phi_r$  is zero, therefore only the front axle steers. As for the zero sideslip model, the inputs are the longitudinal speed  $v_x$  and the yaw rate  $\omega$ .

$$\begin{cases} \omega_f = \frac{v_x}{R \cos \phi_f} & \omega_r = \frac{v_x}{R} \\ \phi = \arctan\left(\frac{\omega(L_f + L_r)}{v_x}\right) & \phi_r = 0 \end{cases} \quad (4.11)$$

For the simulink simulation we have utilized the front steering equation (4.11)

## 4.4. Configuration and run of the Model

In this model the parameters can be changed while the simulation is still running, for example ones can change the velocity of the robot, or can be changed the weights of the obstacle avoidance block and see how to robot change his behaviour.

The waypoints are computed in Matlab with the implemented algorithm, thanks to that is computed the optimal path and it is given in input to the pure pursuit control. In that way we have our global planner, we have the precomputed route that the robot has to follow to reach the goal. The algorithm (waypoints) is combined with the obstacle avoidance system, it is intended as local planner, so the robot can take care of the obstacle that are not presented in the map, thanks to the sensor reading the robot can detect them and replan accordingly his route, after passing the obstacle it can go back on track and continue to follow the precomputed path. In Figure 4.17 is represented the complete Simulink scheme.



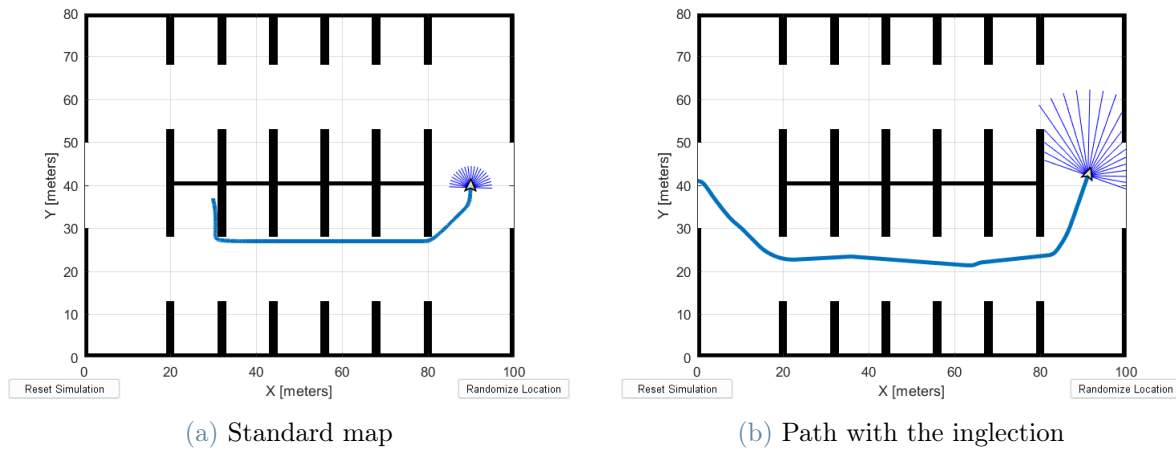


Figure 4.18

Then it is added an obstacle by changing the map in the matlab script that runs the simulation. In that way the precomputed route collides with the obstacle, but thanks to the detection the robot sees the obstacle and avoid it. In the Figure 4.19 this is clear.

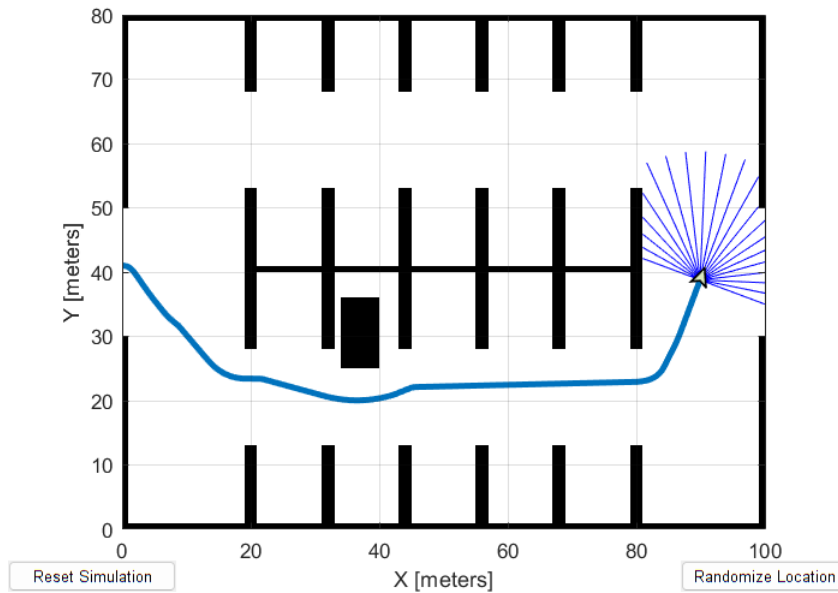


Figure 4.19

Further simulation were performer in order to test more the robustness of algorithm, adding new obstacle and changing their size.



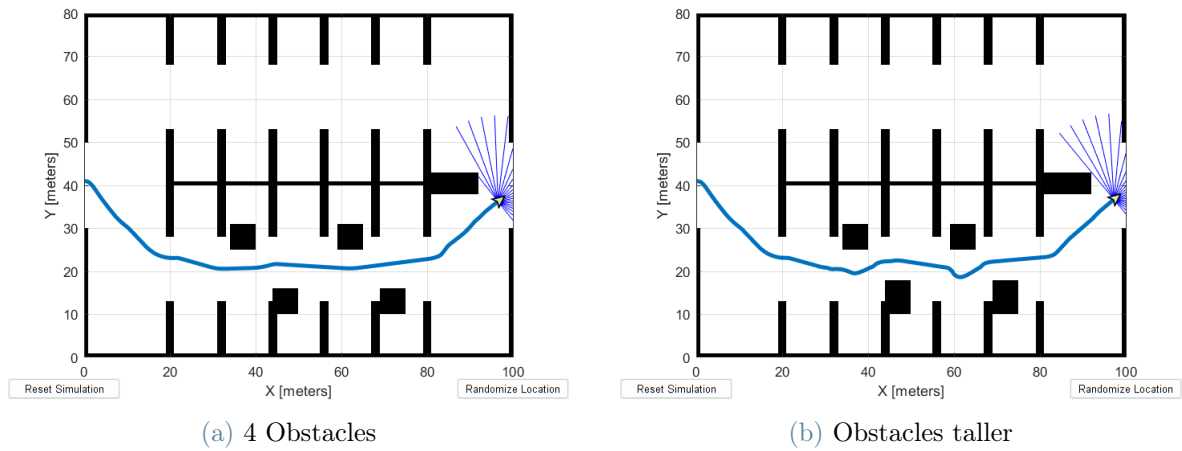


Figure 4.20

The difference of the route that the robot follows in Figure 4.19 and Figure 4.20a is just a bit different. That is because in the algorithm the map is inflated, thus the trajectory is passing sufficiently away from the walls also if there aren't other obstacles in the route. In the Figure 4.20b, instead, the obstacles are taller and the obstacle avoidance algorithm is challenged more, in fact the robot trajectory is less direct and has different turn and changes of direction. Table (4.1) summarizes the main parameter used for the simulation that have been found after different attempts.

	Simulation parameter
Desired linear velocity [ $m/s$ ]	1.5
Maximum angular velocity [ $rad/s$ ]	0.1
Lookahead distance [ $m$ ]	10

Table 4.1: Pure pursuit parameter

The linear velocity is chosen in order to not harm people in case of collision and hazard.

	Simulation parameter
Range distance limits [m]	[3.5,20]
Vehicle radius [m]	0.5
Safety distance [m]	0.3
Minimum turning radius [m]	1.3
Target direction weight	5
Current direction weight	2
Previous direction weight	2

Table 4.2: VFH parameter

The last 3 voices of the table (4.2) are the cost function weights of the vector field histogram block. It is decided to give at the target direction more weight than the sum of the other two voices, in that way the robot is following strictly the route that is given by the modified A\*.

#### 4.4.2. Ackermann steering vehicle

The simulation with the ackermann-like vehicle is performed with a different simulink scheme with respect to differential drive, and it is not used ROS but the "Mobile Robot Toolbox" that Simulink provides. In Figure 4.21 is depicted the complete scheme.

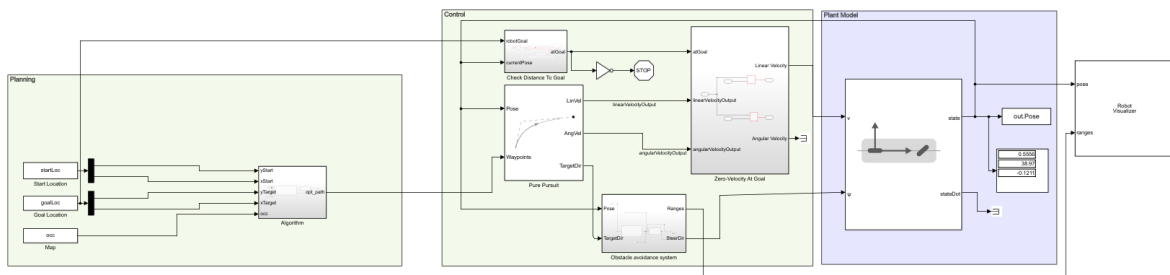


Figure 4.21: Complete scheme of Ackermann steering

Some of the blocks are the same with respect to the previous simulation, the different blocks are the one for the kinematics model ( Figure 4.22a) and, as said, the block from "Mobile Robotics Toolbox", i.e. the lidar scan (Figure 4.22b) and the robot visualizer.

This last block permits to visualize the real trajectory that the robot performs on the map.

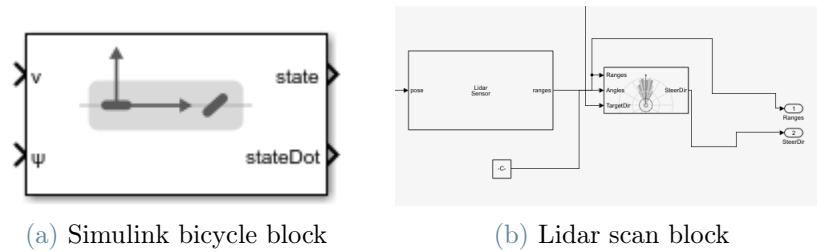


Figure 4.22

The simulink block about the bicycle model permits to select among 2 options, both concerning the inputs, one can select to give as input linear and angular velocity or linear velocity and steering angle. The choice is to inputs the steering angle that the Vector Field Histogram generates, so the model is reactive and can avoid the obstacles.

Figure 4.23a depicts the trajectory without any obs; when an obstacle is added, Figure 4.23b, the robot decides completely to take a different route in order to avoid the obstacle.

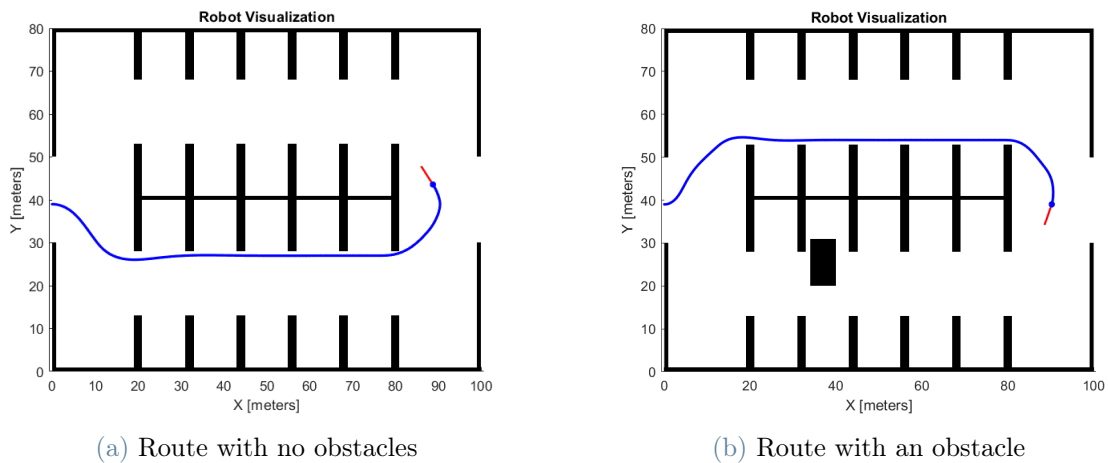


Figure 4.23

The Figure 4.24 shows the simulation taken with the same map and different obstacles. The start and goal location are the same with respect to previous runs.

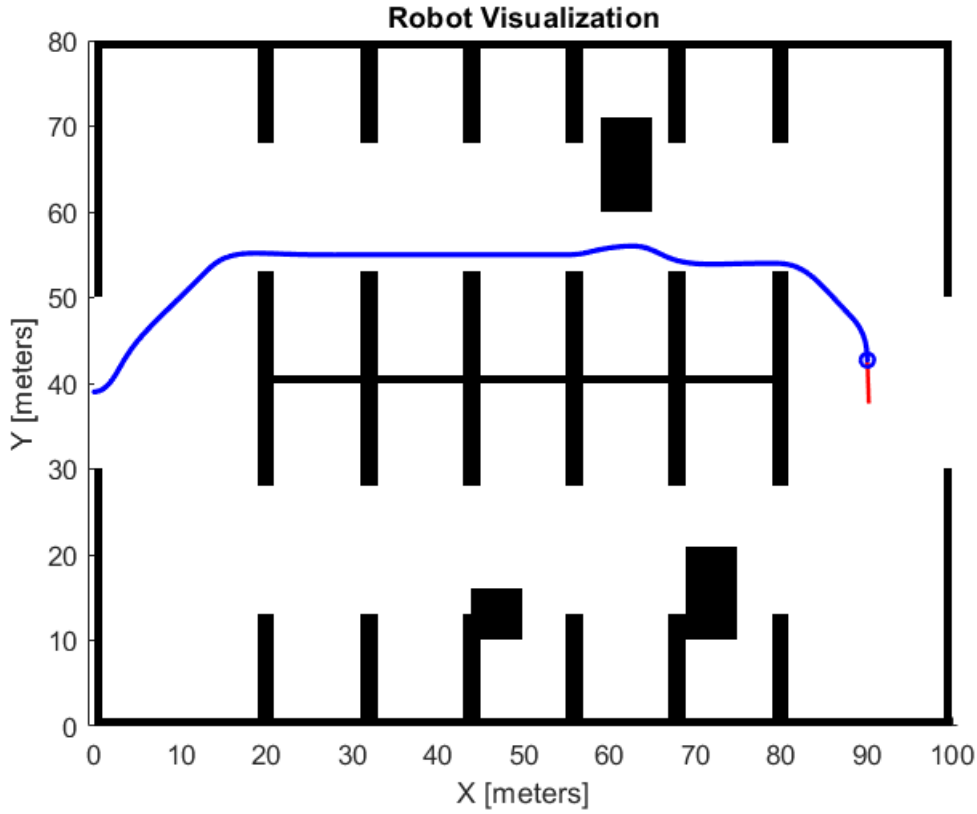


Figure 4.24: Trajectory with 3 obstacles in the map

The simulations with the bicycle model have been surely more challenging than the differential drive model ones. The importance of the parameter here is noticeable, they have been tuned and setted as it is after much trial and error session.

At the end the parameter used for the right simulation are summarized in Table (4.3), Table (4.4) and Table (4.5)

	Simulation parameter
Desired linear velocity [ $m/s$ ]	7
Maximum angular velocity [ $rad/s$ ]	2
Lookahead distance [ $m$ ]	5

Table 4.3: Pure pursuit parameter bicycle simulation

	Simulation parameter
Range distance limits [ $m$ ]	$[3.5,20]$
Vehicle radius [ $m$ ]	1.5
Safety distance [ $m$ ]	0.75
Minimum turning radius [ $m$ ]	3.5
Maximum steering angle [ $rad$ ]	$-\pi/4$
Target direction weight	5
Current direction weight	2
Previous direction weight	2

Table 4.4: VFH parameter bicycle simulation

	Simulation parameter
Scan angles [ $rad$ ]	$[-\pi/4,0, \pi/4]$
Maximum range [ $m$ ]	10

Table 4.5: Lidar parameter



# 5 | Conclusion and future developments

In this work of thesis is presented an A\* modified algorithm for path planning which aims to enhance the algorithm robustness by avoiding collisions and improving efficiency by reducing expansion nodes and run time. Since the A\* algorithm was invented for shortest path determination without consideration for robustness, it were presented three major improvements. These strategies are a smarter way to construct the OPEN list, the pre-inserted obstacle in the CLOSED list, a new heuristic function and the expansion distance, i.e the inflation of the map. These strategies should be combined because they have great potential in solving the problems of A\* algorithm.

It was proven that the modified A\* has an elapsed time lower than the basic one: the difference is about one order of magnitude. Then compared to the Matlab planner the implemented algorithm searches also much less node: it is a major efficiency enhancement. Finally it was merged with an obstacle avoidance system in order to complete the path planning environment, in that way the global planning given by the implemented algorithm is put together with the local planning that helps to detect and avoid the obstacles. Two type of kinematics were tested: the differential drive, that is the most used among the warehouse robot, and the the bicycle model, a simplified way to mimic a car-like robot. The algorithm can be further improved by introducing the bidirectional search and smoothing as in [5], also the obstacle avoidance system can be enhanced maybe with the use of the potential field method.





## Bibliography

- [1] R. Grasso. Hyundai, primo test di auto a guida autonoma con alimentazione a idrogeno, 2018. URL [https://auto.hwupgrade.it/news/tecnologia/hyundai-primo-test-di-auto-a-guida-autonoma-con-alimentazione-a-idrogeno\\_74012.html](https://auto.hwupgrade.it/news/tecnologia/hyundai-primo-test-di-auto-a-guida-autonoma-con-alimentazione-a-idrogeno_74012.html).
- [2] B. Z. C. L. Guoqing Xia, Zhiwei Han and X. Wang. Global path planning for unmanned surface vehicle based on improved quantum ant colony algorithm. pages 1–7, 2019.
- [3] H. E. H. Q. B. A. L. A.-S. Heba Kurdi, Shaden Almuhalhel and G. Almoaiqel. Tide-inspired path planning algorithm for autonomous vehicles. 2021.
- [4] J. J. Y. W. W. L. T. L. Huanwei Wang, Shangjie Lou. The ebs-a\* algorithm: An improved a\* algorithm for path planning. 2022.
- [5] S. L. J. J. . H. H. Huanwei Wang, Xuyan Qi and W. Liu. An efficient and robust improved a\* algorithm for path planning. pages 1–15, 2021.
- [6] G. Z. F. N. Ignacio Pe ´rez-Hurtado a, Miguel A. Martinez-del-Amor and M. J. Perez-Jimenez. A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. pages 1–5, 2018.
- [7] C. J. P.-d.-P. José Ricardo Sánchez-Ibáñez and A. García-Cerezo. Path planning for autonomous mobile robots: A review. pages 1–3, 2021.
- [8] F. H. M. E. B. Jörg Fickenscher, Sandra Schmidt and J. Teich. Path planning for highly automated driving on embedded gpu. *Low Power Electronics and Applications*, 2018.
- [9] O. J.-R. Keenan Albee, Alejandro Cabrales Hernandez and A. T. Espinoza. Real-time motion planning in unknown environments for legged robotic planetary exploration. 77 Massachusetts Avenue, 2020.
- [10] J. Z. Letian Lin. Path planning for autonomous car parking. Atlanta, Georgia, USA, 2018.

- [11] A. S. S. P. J. B. P. Manasvi Sagarkar, Shreyas Damodar More. Perception and planning in autonomous car. 2020.
- [12] MathWorks. Robot operating system (ros), 2021. URL <https://it.mathworks.com/help/ros/gs/robot-operating-system-ros.html>.
- [13] N. A. R. Mohammad Hamdan Garibeh, Mohammad Abdel Kareem Jaradat. A potential field simulation study for mobile robot path planning in dynamic environments. The address of the publisher, 2019.
- [14] M. F. Mohammed M.S. Ibrahim, Mostafa Rostom Atia. Autonomous vehicle path planning using q-learning. *Journal of physics*, pages 1–7, 2021.
- [15] P. Rocco. Control of industrial robots motion planning, 2020.
- [16] ROS. Base local planner, 2020. URL [http://wiki.ros.org/base\\_local\\_planner](http://wiki.ros.org/base_local_planner).
- [17] Sensible.
- [18] T. W. Y. Z. L. Z. Shaobo Wang, Fen Lin and Y. Deng. Autonomous vehicle path planning based on driver characteristics identification and improved artificial potential field. 2022.
- [19] M. C. Simonluca Pini. Guida autonoma, rivoluzione che procede a piccoli passi. 2018.
- [20] B. S. Taiping Zeng. Mobile robot exploration based on rapidly-exploring random trees and dynamic window approach. 2019.
- [21] J.-P. L. R. O. Wael Ben-Messaoud, Michel Basset. Smooth obstacle avoidance path planning for autonomous vehicles. 2020.
- [22] Z. Z.-L. Z. Wang Yijing, Liu Zhengxuan. Local path planning of autonomous vehicles based on a\* algorithm with equal-step sampling. 2018.
- [23] L.-L. Wang<sup>1</sup> and L.-X. Pan. Research on sbmpc algorithm for path planning of rescue and detection robot. 2020.
- [24] B. A. W.Y.H.Adoni, T.Nahhal. Rapidly-exploring random trees based on hadoop mapreduce concept. In *Journée Doctorale en Informatique pour l'Aide à la Décision*, 2018.
- [25] A. K. Z. L. Yifan Wang, Prathamesh Pandit and K. A. Daltorio. Rapidly exploring random tree algorithm-based path planning for worm-like robot. pages 1–5, 2020.
- [26] T.-y. Z. X.-n. Z. L. L. Yong-tao Liu, Rui-zhi Sun and G. qing Shi. Warehouse-oriented optimal path planning for autonomous mobile fire-fighting robots. 2020.

## Acknowledgements

Ringrazio il professor Matteucci per l'attenzione, il tempo e il supporto che mi ha fornito durante tutto il percorso di tesi. Ringrazio la Kineton per avermi dato l'occasione di svolgere il tirocinio presso la loro azienda ed avermi proposto un'attività così interessante e stimolante. In particolare ringrazio la dottoressa Tufo per avermi affiancato e guidato in tutta la durata di questo percorso. Infine ringrazio il Politecnico di Milano per tutti gli strumenti e l'attenzione che fornisce ogni anno a tutti i suoi studenti.

Il "grazie" più grande e significativo va alla mia famiglia, ai miei genitori e a mia sorella, mi hanno permesso di intraprendere questo bellissimo percorso e non mi hanno fatto mai mancare l'amore e il sostegno, sarò sempre grato a loro per come mi hanno cresciuto e fatto diventare la persona che sono oggi.

Ringrazio tutte le persone che ho conosciuto in questo nuovo percorso, su tutti Gianluca e Pasquale che ho conosciuto all'inizio della Magistrale e che sono stati fondamentali per me.

Infine ringrazio tutte le persone che in un modo o nell'altro hanno contribuito a questo traguardo, anche gli amici lontani mi hanno fatto sentire la loro vicinanza, sempre.

