UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXIX Ciclo

GOAL-ORIENTED PATH PLANNING FOR GROUND AND AERIAL VEHICLES

Coordinator:

Prof. Marco Locatelli

Supervisor:

Prof. Alberto Broggi

Tutor:

Prof. Massimo Bertozzi

PhD student: *Andrea Signifredi*

December 2016

Fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza
–Dante Alighieri

List of contents

List of Figures

List of Tables

Introduction

Nowadays autonomous robots are used in everyday life more than ever. The idea that motivate the develop of new autonomous application is to lessen the fatigue of repetitive works and to make safer the work that are difficult if done by humans alone. Another goal of autonomous robots is to improve precision and repetitiveness in the actuation of actions.

Car makers are now showing to consider autonomous driving a ground braking functionality and one of the most important additions to their assets to maintain a sustainable competitive edge on the market in the mid-long term. Same scenario is happening in almost every technological area. To be competitive on the market, companies need to add intelligent feature to their products. For example in drone market all the top of the line product have autonomous features and companies invest a lot of effort to improve these features.

Path planning is a crucial part of the creation of autonomous systems. While motion planning is a difficult task in general, its application to autonomous vehicles and drone moving in presence of other actor imposes requirements that make it even more challenging. For example autonomous cars have to deal with other cars, pedestrians, cyclists, etc.. Moreover road going vehicles obey non-holonomic motion laws that must be accounted for during planning. Another example are drones that flying in presence of pedestrian can be a danger if not driven correctly.

Path Planning is a huge field of study. In this dissertation we focus our attention on those application where the planner has a well target. We provide an architecture and algorithms that allow the build of a general system that allows autonomous

robot to reach a target. Attention is given in the generation of behaviors that are very smooth. Moreover is given top priority to the safety of the system.

To test our algorithm we choose an agricultural application, which is an autonomous tractor that reach and collect round hay bale in a field. We also describe an automotive application such as an autonomous parking feature. Finally we demonstrate that our algorithm are also suitable for 3D planning with some drone applications.

The dissertation is structured as follows: chapter 1 provides a broad overview of the state of the art. Chapter 2 describes Ground Vehicles application, while chapter 3 describes Aerial vehicle one. Chapter 2 is divided in: first section, which describes the agricultural application, and second section, which describes the parking algorithm. Chapter 3 describes Aerial Vehicles applications.

Chapter 1

State Of The Art

There is a huge corpus of literature on autonomous robot, collision-free, time-optimal path planning. One of the first and most influential was [40] back in 1979. For most of them a high number of Degrees of Freedom (DoF) is required in the experimental platforms. Classical methods search for the mathematically proven optimal, if it exists. On the other hand, there are other traditional very widely used heuristic methods like A* ([18]), D* ([51]), Probabilistic Roadmaps ([53]), Rapidly-exploring Random Tree (RRT ([36], [37]), potential field ([23]) and also based on Pontryagin's Minimum Principle ([1], [5], [2]). There are also many examples on Genetic Algorithms, Neural Networks, Ant Colonies, Simulated Annealing, Particle Swarm Optimization and other Natural Computing optimization techniques, usually in a multi-objective fashion.

However, for this dissertation, we will focus specifically on two types of robot. Firstly we will focus on non-holonomic[1] unmanned Ground Vehicles (UGVs) path planning developments, where DoF are dramatically restricted. In other words, we will focus on automated car-like vehicles. Secondarily we will focus on aerial drone, in particular quadrotors. Most of the algorithm described in this section are for ground robots but with little work they can generalized to 3D path planning. This is also

[1] vehicles with a restricted DoF, e.g. cars. Holonomic robots are those who are assumed to have all the 6 DoF.

possible because, differently from airplane like robots, quadrotors can stay still in the air. They don't have wings that need a minimum velocity to generate enough lift force, but they use four propellers that lift them.

We divided Path planning in two category of algorithms: Explorative Path Planning and Direct Path Planning. The first one plans from point A to B, to create a close to time optimal path from origin to destination, avoiding static obstacles. The second one connects two points, normally without checking obstacles or at most it is possible to generate multiple trajectory and discard the ones that cause a crash. This class of algorithm allows to control the vehicle when the target is in sight, sending a sequence of feasible (drivable) control commands, making trajectories smooth and time optimal, and sometimes even avoiding smaller obstacles, avoiding less traversable patches.

Now we will list the most frequent methodologies used for Explorative Path Planning and Direct Path Planning.

1.0.1 Explorative Planning

For Explorative Planning, the most frequently used methodologies are as follows:

- A* ([20] [58] [30] [16] [17] [25]) combines the advantages of uniform-cost and greedy searches using a fitness function ([57]). Its fitness function is basically composed of two parts, one to calculate the cost from the origin to the current position, and the second to estimate, through an heuristic, the cost from the evaluated position to the target position. Obviously, the choice of a good heuristic is essential to have an efficient and robust method. Usually, just the linear distance is enough since it never overestimates the cost to the goal position and is computationally very efficent.

- D*, standing for Dynamic A* Search, ([50] [27] [15]) is a quite similar to A* algorithm which is believed to be faster. It expands path nodes from destination. It is called dynamic because arc costs can change during the algorithm execution. In comparison to A*, D* has the completeness and optimality properties. It can also be more efficient. However, according to [59], A* seems to

be more suited to passive or static environments, like it is in the case of the present work.

- Dijkstra ([33]) is a good path-finding algorithm to find high-quality paths. It only considers the cost from origin to each evaluated position. Hence a greedy search is needed, meaning it has a much higher computational cost than methods using heuristics and future cost estimations.

- Rapidly-Exploring Random Trees (RRTs) and relatives like RRT*, MARRT, etc. ([64], [19], [29], [38], [62]) . RRT is a data structure and algorithm that is designed for efficiently searching non-convex high-dimensional spaces by building a space-filling tree. Basically, it does a biased search into the largest Voronoi regions of a graph. It works by creating recurrently collision free branches of a expanding tree until it gets to the goal position.

There are a few other less common methodologies that can be found used as general planning methodologies or in combination with other algorithms like Probabilistic Roadmaps ([35], [49]), Potential Field ([35]), Pontryagin's Minimum Principle based methods ([2], [34]) and Natural Computing based methodologies, generally Genetic Algorithms ([4], [39], [12]) and Particle Swarm Optimization ([42]).

There is also a quite different group of methodologies we could label as Cooperative or Distributed Path Planning ([47], [65], [56]) where paths are calculated using information coming from a network of vehicles, distributed sensors, or both. We have included this category here, even if it approaches the path-planning problem in a quite different fashion. It starts from the assumption, hardly attainable in real-world outdoor scenarios, which is that a very populated sensors network is deployed in the field, so a top view of all the environment can be acquired. In [47] they use a Potential field approach over a Voronoi skeleton.

1.0.2 Direct Planning

The second step is Direct Planning. The focus is on generating a smooth curve that can be run by the control system of the vehicle. This curve should connect the vehicle

with the position of the target given by a detection system. For direct planning there are traditionally two big categories: Functional Methods and Polynomial Approximation Methods.

Functional Methods

Functional methods work by combining mathematical functions to express the path. This first category can also be divided in two, as follows:

Closed-Form Functional Methods These are functional methods whose coordinates have a closed-form expression, like spline based methodologies ([32], [61], [43], [6], [41]), quintic polynomials ([54]) and Bézier based curves ([60] [22] [63]).

According to [63] the main weaknesses of this group of methodologies are that there is no efficient way to optimize the path lengths nor the minimum radium to fit the vehicle mechanical restrictions.

Parametric Functional Methods Functional Methods whose curvature is defined as a parametric curve, which is a function of their arc length. Some frequent functions used are:

- Clothoids ([14])
- Cubic spirals ([28])
- Quintic G^2-splines ([45] [44])
- Intrinsic splines ([8])

This family of methodologies started with Dubins curves ([10], [46], [24], [13]). The original 1957's method is quite limited, since it basically combines only two kind of functions: lines and circle segments, taking the maximum rate of change of turn of the specific vehicle. The main drawback of Dubins curves' based methods is the discontinuities that may arise between two different circle segments – where a real

system would need to stop because the linear and angular velocities need to be continuous in real world. There has been a number of efforts to cancel those discontinuities, e.g. using clothoids instead of circles ([14], [31]) or even spirals ([28]).

Clothoid pairs provide smooth transitions with continuous curvature, meaning that a non-holonomic vehicle could actually follow them. They can also be optimized to minimum-length for a given limit on jerk ([41]).

The main two drawbacks of this kind of methodologies are that it is not possible to theoretically proof completeness (with the exception of [48]) and that having no closed-form expression for position along the clothoid path means that the vehicle control needs to be built on approximations to that path and/or look-up tables. Anyway, both limitations should not be a problem with modern equipments and computers.

The methodology proposed in this dissertation in the local path planning of agricultural application is in this group. In particular, we are using the method presented in [45] because it provide a flexible tool to generate simple trajectories with the possibility to impose some contraints.

Polynomial Interpolation Methods

Some argue that these methods have limitations regarding providing an optimal path for a set of random points and serious scalability problems, due to the Runge's phenomenon ([11]) and its computational cost.

A remarkable work in this category is [21], where a very interesting Quadratic Polynomial Interpolation is shared.

Chapter 2

Ground Vehicles Path Planning

In this chapter we will describe two applications. The first is an agricultural field application. An autonomous tractor has to go in a field to pick up round hay bales and take them to one side of the field. The second application is the self parking for an autonomous car.

2.1 Hay Bale Picking

In this section we describes the Hay Bale Picking applications. The first part describes the system setup. The second one describes the detail of the algorithm. In the end, results and conclusion are presented.

2.1.1 System Description

In this section we will describe the systems that work and interact with the Path Planning System.

Hardware Setup

The hardware platform is a New Holland T7000 tractor, as shown in figure 2.1. It is a tractor produced by CNH Industrial N.V. equipped with a drive by wire system that

allows a PC to control it by means of a CAN bus interface. In mount an hydraulic fork lifter used to pick up and move round hay bale.

Figure 2.1: New Holland T7000 tractor.

The tractor is equipped with sensors to perceive the environment around it. It mount a Bumblebee XB3 on the front of the roof, as shown in figure 2.2. It is a 3-sensor multi-baseline IEEE-1394b (800Mb/s) stereo camera designed for improved flexibility and accuracy. It features 1.3 mega-pixel sensors and has two baselines available for stereo processing. The extended baseline and high resolution provide more precision at longer ranges, while the narrow baseline improves close range matching and minimum-range limitations. To enable the system to remain outdoor in any circumstances the camera is protecter with a custom case that impermeabilize it, making it IP 67 certifiable.

It mount a laser scanner Sick LMS-101 on the front of the tractor, as shown in figure 2.3. It is an efficient and cost-effective 2D laser scanner for measuring ranges of up to 20 m. It guarantees outstanding performance whatever the weather, thanks to multi-echo technology and intelligent algorithms. It is rugged, compact housing with

Figure 2.2: Bumblebee XB3 with custom case. The case has the purpose of impermeabilize the camera.

enclosure rating up to IP 67, integrated heating and a temperature range from ˇ40°C and $+60°C$.

Obstacle Detection system

The Obstacle Detection system has the goal of provide to the planner a point cloud of the obstacles in front of the vehicle.

As first step, it compute a disparity map using the images from Bumblebee cameras. we can see an example of disparity map in figure 2.4.

After this step, the system convert the disparity map to a point cloud. The Obstacle Detection system run now a terrain estimator. This algorithm works by fitting a 2D Bspline that approximate the terrain surface. Then Obstacle Detection system deletes all points that belong to the terrain. An example of a 3D reconstruction of the obstacles in front of the vehicle is shown in figure 2.5. The object are converted in

Figure 2.3: Laser scanner Sick LMS-101.

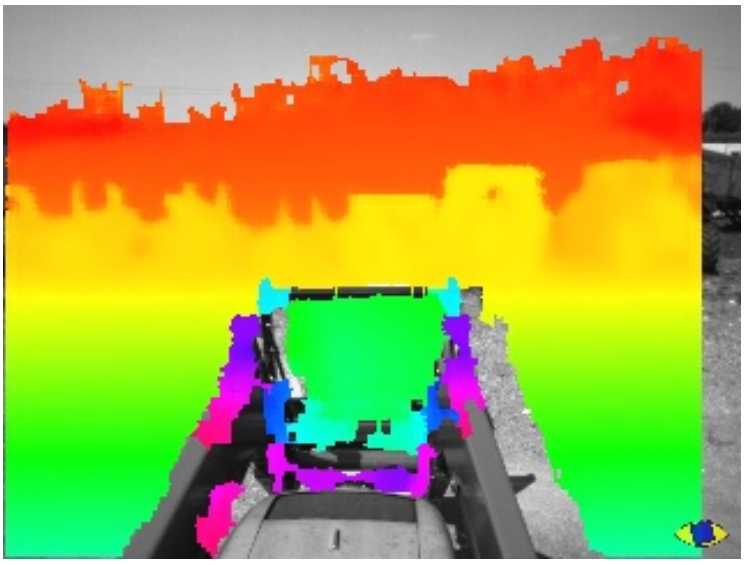


Figure 2.4: Disparity image computed by the obstacle detection system.

global position using the coordinate of the tractor and given as output of the Obstacle Detection system.

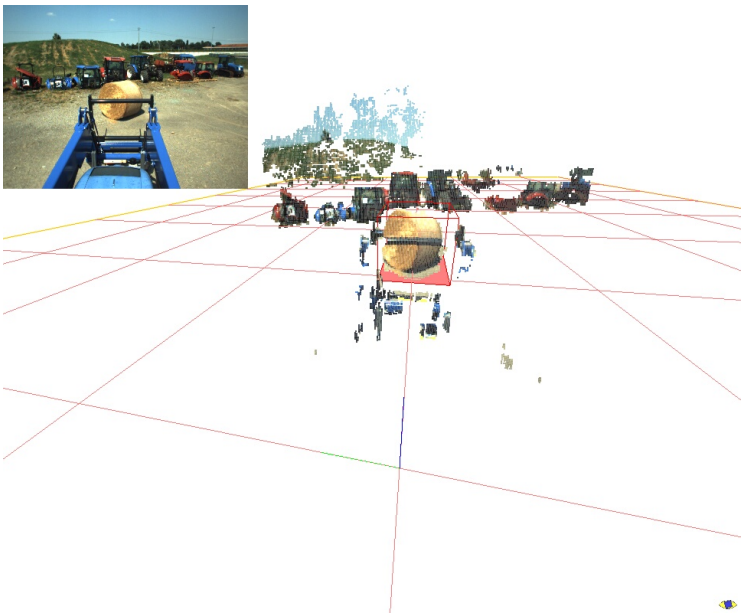


Figure 2.5: 3D reconstruction of the obstacles in front of the vehicle.

Bale Detection system

The Bale Detection system has the goal of provide to the planner the exact position and orientation of the hay bale when there is one in front of the tractor.

The point cloud of the Obstacle Detection system is given as input to the system. Using this point cloud the system divide it in separate objects. For each object, the system runs a monocular Adaboost classifier on the 2D bounding box on the image that contain all the object. This first classification steps eliminate the object that are completely different from hay bale. After this phase it tries to fit a cylinder of dimension similar to bale on each remaining objects. If the object has a low fitting error, it is classified as a round bale. To have a precise position and orientation of the bale, the position and orientation given by the cylinder fitting phase is used together with

the laser scanner data to have a better measure.

In figure 2.6 is possible to se the point clouds of two object, an hay bale and a pedestrian. The round bale is correctly classified and shows a segment pointing out from the center of it representing the model found by the cylinder fitting algorithm. The pedestrian is correctly classified as non cylinder.

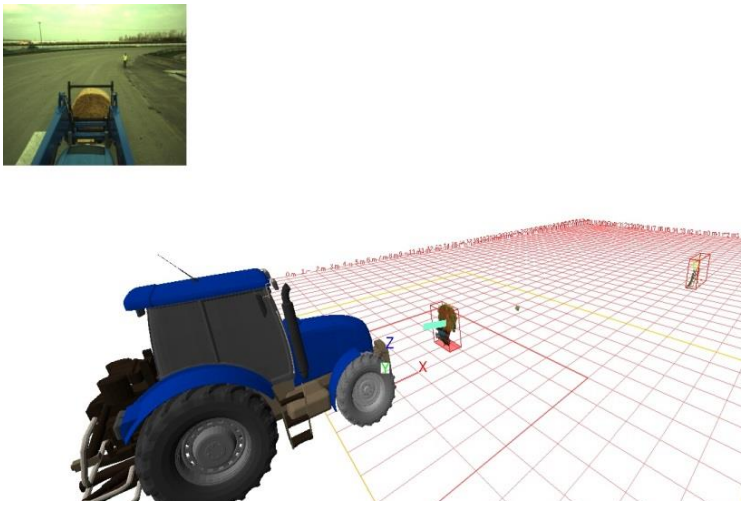


Figure 2.6: 3D reconstruction of the object in front of the vehicle. It is possible to see that the bale has a segment coming out from its center, and pedestrian don't. This is because the system recognize and classify correctly the two object.

In figure 2.7 is shown the displacement in meter and the difference in degrees respect to the front of the tractor. This measure is computed using both the data from stereo system and laser scanner data.

2.1.2 Method Overview

In this section we will explain the detail of the algorithm. First part describes the Global Planning part, used to reach the proximity of the bales and perceive the exact position of it. Second part describes the Local Planning part, used to compute the exact maneuver to pick up the bail after seeing it.

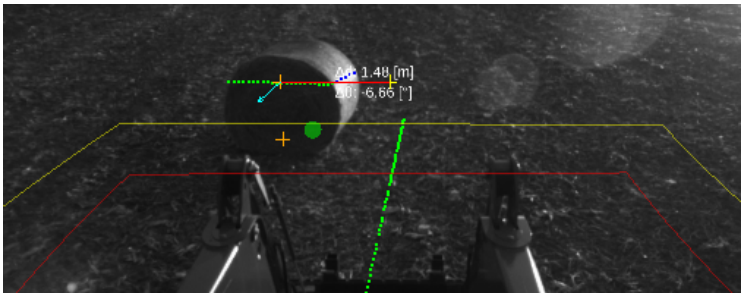


Figure 2.7: Output of the Bale Detection system. It is shown the displacement in meter and the difference in degrees respect to the front of the tractor. The cross in the center of the bale with a vector pointing out is the central point and orientation of the bale detected by using the laser scanner fused with the stereo camera system.

Global Planning

As said before, the global method for planning aims at solving the problem of estimating a trajectory to navigate from the current position of the vehicle to a specific GPS configuration (position and orientation) of the place of interest, before that its exact position and orientation is being observed and calculated by the detection system.

We based the first part of our work on [9] because A* is one of the most commonly used search techniques in motion planning, and the variation proposed by Dolgov et al. permits to search generating feasible steering actions. This allows to generate trajectory almost ready to be actuated.

The output of the system is a trajectory that neither collides with any fixed obstacle nor goes outside the working area. It must have a maximum curvature, restricted by the vehicle's maximum feasible curvature – taking into consideration the non-holonomic nature of the vehicle. Furthermore, the trajectory has to be as short as possible for obvious reasons.

As in the implementation of Dolgov et al. [9], the input of the algorithm is an obstacle map, that contains all the position of known obstacles. This map is implemented as a grid divided into cells. At first, each cell with a part of an obstacle on it,

is set as occupied. After all the obstacles have been evaluated, we perform a morphological dilation, taking into account the space occupied by the vehicle. The dimension of the vehicle is used to expand the obstacles. After this phase we can consider the vehicle as a point.

Another input of our algorithm is the configuration that have to be reached. This data is collected via GPS for the position and with a magnetometer for the orientation. We must take into consideration that this input may be affected by noise such as drift for GPS, bias related to temperature, etc., and there may be also noise due to the fact that the data has been collected from another vehicle or by a human operator with devices that may have a slightly different calibration from the ones mounted on our vehicle.

The algorithm is a variant of the Hybrid-state A* Search proposed in the first part of the work of Dolgov et al. [9]. Each node of our implementation has only three continuous coordinates (x,y,θ) which represent position and orientation of the node in the world. We choose not to take into consideration the direction of the motion of the vehicle (forward or reverse) for several reasons. One reason is that the system must be able to respect real-time specifications without neither the use of high speed processor nor dedicated computer. Through reducing the coordinates from four to three, we reduced the computational cost of the algorithm. Another very important reason is that the vehicle has a vision obstacle detector and a laser scanner obstacle detector that are mounted on the front of the vehicle. For this reason we opt for limiting the reverse manoeuvres. We go in reverse only when we have recently seen the area where we have to go through, for example when local planner is running and we are too close to the goal and we cannot calculate a good trajectory.

For these and some other reasons we also modify the Analytic Expansion part of the algorithm described by Doglov et al. We decided to use Dubins curves ([10]) instead of the curves based on Reed–Shepp model. These two curves are very similar: both are segment of circle with maximum curvature and straight line. The difference is that Reed–Shepp curves contemplate reverse movements (e.g. a Reed–Shepp curve can be composed by circle to the right, reverse circle to the left and a circle to the right again, on the other hand Dubins curve are composed only by forward movements).

Another difference with Doglov's implementation of Analytic Expansion is that when the distance between the node and the goal is under a chosen threshold, we randomly try to connect the node that have to be expanded with the goal by a Dubins curve. We didn't pick any of the possible Dubins curves that connect the two positions. Instead we took out all the curves that pass on a obstacle and we also rejected the curves with circle parts that exceed $\pi/2$ each. This last condition is important, especially when we are near the goal, because matematically the positions can be reached anyway but with a long circular path that is uncomfortable and inefficient.

We also changed the non-holonomic-without-obstacles heuristic using the Dubins curves ([10]). We calculate online the length of the shortest Dubins curve that link the node with the goal and use that measure as the heuristic. We left unchanged the holonomic-with-obstacles heuristic.

A required feature of the trajectory is that the last segment has to be straight in order to give enough space to the local planner to calculate a new smooth trajectory after the goal is detected. This space allows the vehicle to reach the goal without going in reverse when the position and orientation error is low. This feature is required because the detection system is based mainly on a stereo system which sees only on a visual cone in front of it.

This feature is difficult to be fulfilled in the standard implementation of A* and in Hybrid-state A*. To fulfil it, our algorithm starts to expand the A* search from the goal toward the current position of the vehicle. For the first N meters of the tree, a weight is summed to the costs of each node that is not straight. The weight decreases as we get away from the first node. With this modification, the search will try to go straight for N meters and, if it can't (e.g. to avoid the collision with an obstacle), it will go straight as much as it can and then turn. This change doesn't affect the speed of convergence of the algorithm if the weights are big enough, because it works as a choice of the best node from where the normal execution starts.

We left unchanged the rule to expand the tree. From each node we can go straight, turn max curvature to the right or turn max curvature to the left. In fig. 2.8 we can see an example of computed trajectory.

To follow the generated trajectory we use a Pure Pursuit tracker [7]. We decided

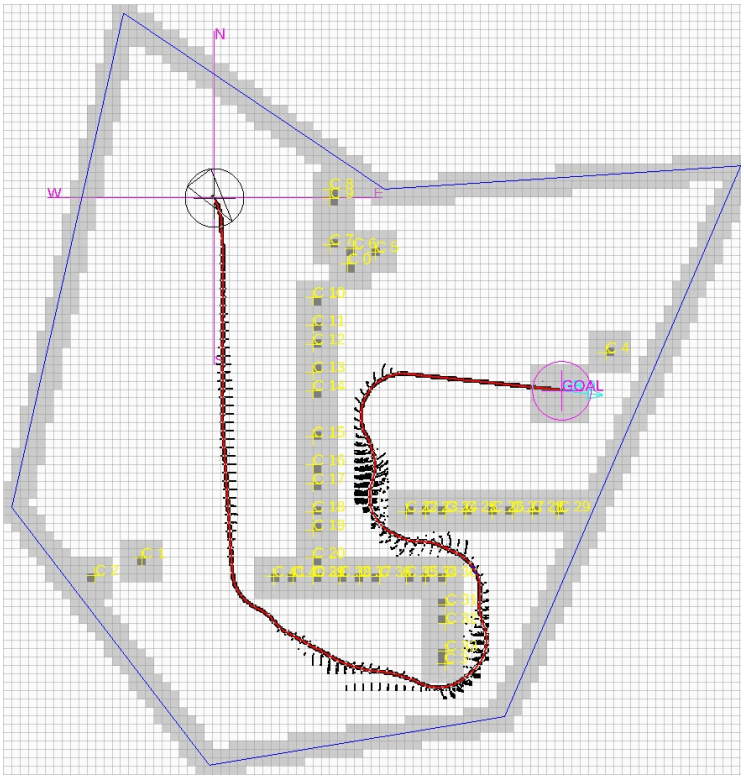


Figure 2.8: Result of global planning. The background contains the grid, and the grey cells represent obstacles and perimeter. The triangle represents the vehicle. The trajectory is shown in red while the black dots show the states explored by our algorithm.

to use it because it is easy to understand and simple to implement and moreover it is robust and reliable. Furthermore its computational complexity is appropriate for real-time applications. It has proved to be able to follow the trajectory correctly, rejecting the error caused by position sensor errors and by delays.

As we can see in fig. 2.9, another effect of using a Pure Pursuit tracker is that the final path has a continuous curvature, even though our trajectory has a discontinuous curvature. This is a necessary requirement for real system because the curvature is

directly proportional to the steering angle which can not change instantaneously.

In this implementation, when the vehicle finds a dynamic or unexpected obstacle, it stops and it waits that the path is freed.

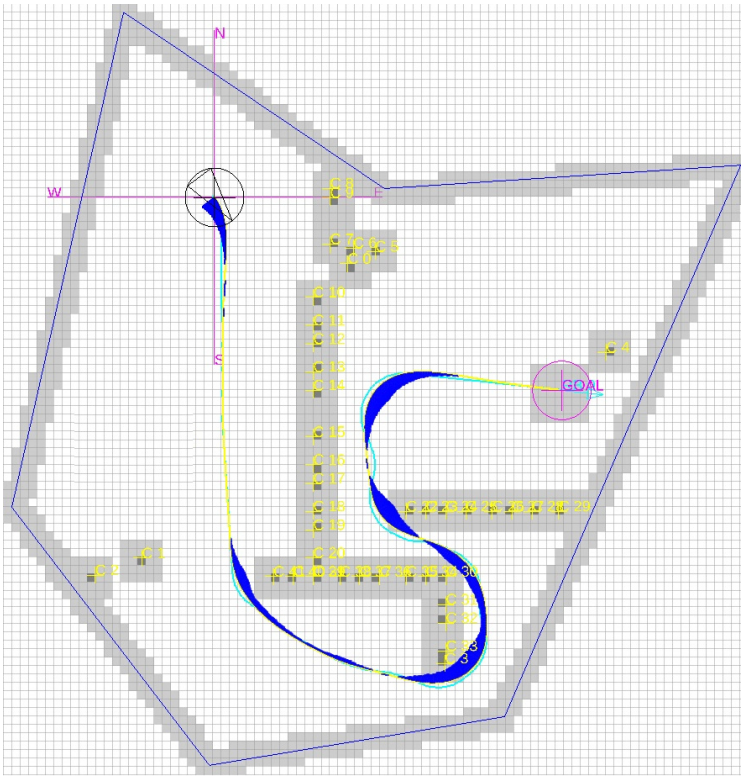


Figure 2.9: Simulation of the tracking system. In yellow the simulated path and in blue a measure of the simulated curvature setpoint. The thickness is proportional to the curvature.

Local Planning

The local method for planning aims at solving the problem of estimating a feasible trajectory to navigate from the current position of the vehicle and the position and orientation of the target place observed and calculated by the detection system.

We based the second part of our work on [45] because we need a motion planning primitive with overall second-order geometric (G^2) continuity. Piazzi et al. present a method that permits the generation of trajectories that satisfy all the conditions and restrictions of our problem.

The detection system gives the relative coordinates of the goal respect to its center. The rototraslation that converts from the coordinate system centered in the detection system to the GPS system is known. First, we convert the point in GPS coordinates in order to consistently store data even if the detection system misses some frames or to cope with the navigation system working at a different frequency, which is expected. If everything works perfectly we should have a not moving point in GPS coordinates because the area detected is not moving. Obviously, this is not a realistic situation because the measure given by the detection system is subject to errors due to miscalculations or approximations. The GPS position and the orientation are also affected by the noise coming from the GPS and the magnetometer.

Taking into account these problems we implemented an algorithm that is focused on robustness, reliability and noise tolerance. Moreover, the algorithm should provide a smooth navigation through the goal.

The input of the algorithm is the GPS position of the vehicle, the current curvature and the GPS position of the goal. The system tries to compute a correct trajectory each time the vehicle moves or the goal position is updated.

The algorithm is composed by two parts: the first calculates a trajectory to the goal and the second part follows this trajectory and implements a controller to reject errors.

The first part calculates a quintic spline trajectory defined as follows:

$$\begin{cases} x = a*u^5 + b*u^4 + c*u^3 + d*u^2 + e*u + f \\ y = g*u^5 + h*u^4 + i*u^3 + l*u^2 + m*u + n \end{cases}$$

As we can see, we have a maximum of twelve constraints to be applied because the number of constrains is equal to the number of free variables in the system.

The main constraints are:

- Start position equal to current vehicle position (two constraints)

- Start orientation equal to current vehicle orientation (one constraint)
- End position equal to detected goal position (two constrains)
- End orientation equal to detected goal orientation (one constraint)
- Start curvature equal to current curvature (two constraints)
- End curvature equal to zero (two constraints)

The formula of the curvature is:

$$K = \frac{x' * y'' - y' * x''}{(x'^2 + y'^2)^{\frac{3}{2}}}$$

To set the curvature in the beginning and in the end, we calculate the first derivatives. Then we set one of the two second derivatives to zero. In this way, after having reversed the equation, we have the formula to set the curvature.

The first six constraints make the trajectory go from current position to goal considering the orientation. We forced the starting curvature to be equal to the current one to avoid any curvature discontinuity. The path must be curvature continue, in order to provide a smoother and more comfortable control. Otherwise, the vehicle should have to stop to maintain angular and linear momentum.

The curvature at the end of the trajectory is forced to zero because we want that the vehicle completes the bend before reaching the goal. In this way the curvature, as well as the angle of the turning wheels, changes continuously, reaching zero in the end.

With this configuration we have ten constraints and twelve free parameters, allowing us to vary the last two parameters to generate many feasible trajectories.Between these trajectories, we choose the one that has the best combination of these features:

- The maximum curvature less then or equal to the maximum curvature feasible by the vehicle
- It must not touch any obstacles

- Minimize the trajectory length
- Minimize the maximum derivative of the curvature

The first two features are mandatory. We reject immediately any trajectory that breaks one of these two constraints without checking the other two. Out of the surviving trajectories, we choose the one that minimizes both the length and the maximum derivative of the curvature. In fig. 2.10 it is possible to see an example of output.

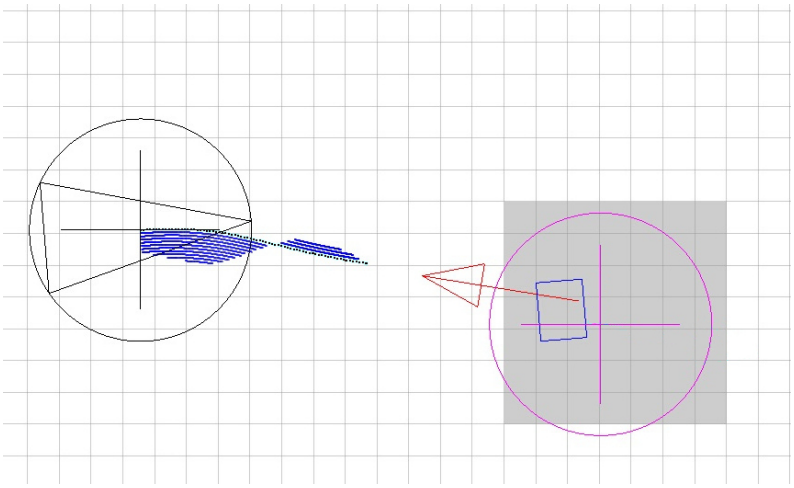


Figure 2.10: Result of local planning. In black the trajectory and in blue a measure of the curvature. The thickness is proportional to the curvature. The arrow represents the output of the detection system. The trajectory ends in front of the goal point because we have to reach the goal with the front of the vehicle.

Path following Once provided the trajectory calculated at the previous step and a GPS position and orientation, we developed an algorithm that has as output a set-point of curvature that permits the vehicle to navigate on the trajectory or, in case the vehicle deviate from it, to get close to the trajectory.

Unfortunately, it is not possible to stay perfectly on the trajectory for multiple reasons: wheel slips, delay in the actuators, delay in the communication modules.

Another source of errors is the detection system, because during the path its output may change both in position and orientation. This can be caused by bad interpretation of the object seen, or because the measure is imprecise in the distance and when the vehicle get close the measure change because it becomes more accurate.

These errors can be grouped in two effects to be corrected:

- The vehicle is not on the planned trajectory, so we want to bring it back on it
- The end of the trajectory is not on the detected point, so if we continue on it we miss the target

We calculate the combination of these errors and correct them simultaneously. The basic idea is that if the errors are both zero we can reach the goal by applying as output the curvature of the trajectory in the point where we are. We simulate the path that we will follow if we don't use any corrections. Starting from the current position, the simulator moves the vehicle by applying the curvature of the nearest point of the trajectory to the simulated position. When the nearest position is the end of the trajectory, we assume that the simulator has reached the position that the vehicle will reach without correction. We use the euclidean distance between the points just calculated and the last point given by the detection system as the error of the tracking system.

The output curvature of the path follower is the result of the combination of: the curvature of the nearest point of the trajectory as feed-forward component and the output of a PID controller that takes as input the calculated error.

2.1.3 Results

In this section we present some qualitative results.

Figure 2.8 and figure 2.9 show that the planner is able to plan a trajectory in complex environments. Moreover, it is able to follow this trajectory generating a smooth sequence of curvature set-points. During the experiments the system successfully reached the goal avoiding the fixed obstacles.

The local planner has proven to be able to reach the target with high precision. We obtained good results regarding both position and orientation. We tested a scenario where the vehicle had to reach a box shaped target with the front of the vehicle. In this way when the vehicle leans against the goal, it is possible to measure the errors. During the experiment we obtained a mean error of approximately ten centimeters in regards to position and ten degrees in regards to orientation.

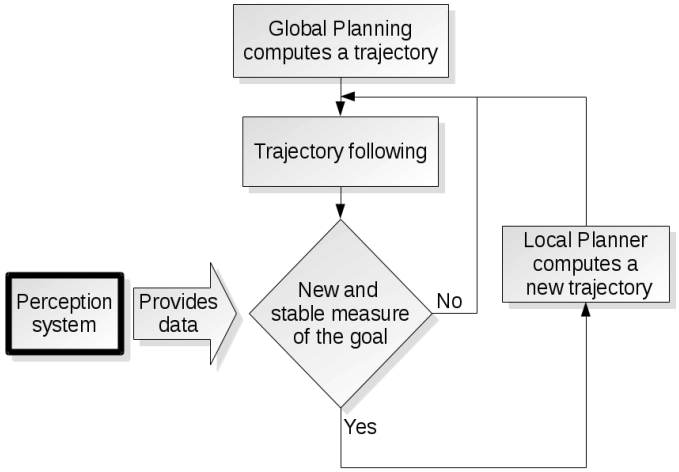


Figure 2.11: Flowchart of the planning system.

As shown in figure 2.11, the switching between the two systems is done when the detection system provides a stable measure. When the information about the goal is provided for some consecutive cycles, the local planner try to calculate a trajectory. When the trajectory is ready, the control system seamlessly switches between global and local planner. That switching is jerk free. There are no curvature discontinuities because the starting curvature of the local planner trajectory is set equal to the last curvature actuated.

In table 2.1 are shown the average computational time of each module described before on a pc with a i7-4800MQ. For global pianification we considered only non-trivial configuration for the time measurement.

Tests have been made also during night using the headlamps of the tractor, as

Table 2.1: Average omputational times of each module.

Task	Average computational time
Global Pianification	532ms
Pure pursuit tracker	89μs
Local Pianification	14ms
PID tracker	77μs

shown in figure 2.12.

Figure 2.12: Tests done during night time.

2.1.4 Conclusion And Future Works

This chapter describes a new secure method to approach a target and reach it with high precision using the information provided by a detection system composed by a stereo vision system and a laser system. The chapter firstly presents a global planning algorithm to go near the target and then it describes a local planning method that takes

control of the vehicle when the target is detected.

Through experiments on real application, we can see that the combination algorithms proposed in this chapter satisfies the requirement of real-time navigation. Furthermore it generates a G^2 control sequence. This feature becomes crucial in transporting people or dangerous goods providing high comfort and safety.

A new scenario where this algorithm will be applied is the path planning for the vehicle shown in figure 2.13. This is an electric vehicle equipped with sensors and actuators, already used for autonomous driving tasks. We will use the system proposed in this chapter to add some interesting tasks:

- Reaching a platform to let people get inside the vehicle. For example to transport old people or people with disability.
- Reaching an autonomous battery chargeing station, where the vehicle can go and charge its batteries or replace them if needed.
- Driving the vehicle in a parking lot after being utilised. After the passengers get off the autonomous vehicle, the vehicle will autonomously park itself, if no one else need it.

2.2 Parking

In this section we describes the Parking applications. The first part describes the system setup. The second one describes the detail of the algorithm. In the end, results and conclusion are presented.

2.2.1 System Description

In this section we will describe the hardware systems on which the Path Planning System works. Images and descriptions are from [3].

The platform car used is a 2013 Audi A4 2.0T FWD multitronic which is depicted in figure 2.14.

Figure 2.13: Autonomous Piaggio Porter. It mounts several stereo vision systems, with different baselines, that look in different directions. It also has four laser scanner that cover the area all around the vehicle

Figure 2.14: Deeva autonomous vehicle.

The car already has off-the-shelf actuators, however, since CAN specifications are proprietary, an AEVIT RPV driving control system has been installed. AEVIT RPV is derived from equipment for impaired people, and already used during the DARPA challenges; it uses the same control points of human drivers. This control

module has its own CAN bus where the Autonomous Driving System must provides messages for primary actuators (i.e.: acceleration/brake and steering) and optionally for secondary actuators (such as wipers and indicators). Also the engine status, and the shifter can be controlled through the control system using secondary actuators. The system integrates safety features and is redundant. Mainly the control module must receive primary messages from the driving system every 30 ms; if this does not happen the control module triggers a user-defined emergency maneuver. The control system cannot be overridden but is connected to an emergency button called Inhabited Take Over (ITO) which is positioned in the middle of the central tunnel and allows the driver to force back to manual control. A remote control is connected to the XBW system to trigger the emergency maneuver.

Sensors are installed all around the car. There are three main sets of sensors: cameras, laserscanners, GPS/IMU.

Cameras: Cameras are the main perception sensor. There are 13 stereo pairs on-board covering all the 360 o surrounding the car, at different distances. The cameras used are UI-5242LE from IDS-imaging, all color version except four near infrared for the far rearview mirrors. They are divided in the following two systems:

- Near: used during maneuvers. It is made of four stereo systems located in the front grille (focal length is 2.5 mm, horizontal field of view is 117.5°), in the upper part of the trunk coverage (same specifications) and under each rear-view mirrors respectively. The stereo pairs in the rear-view mirrors belonging to this subsystem feature two fisheye lenses with the aim of covering the whole side of the car. In this way with 8 cameras all the surroundings of the car are covered with 3D reconstruction.

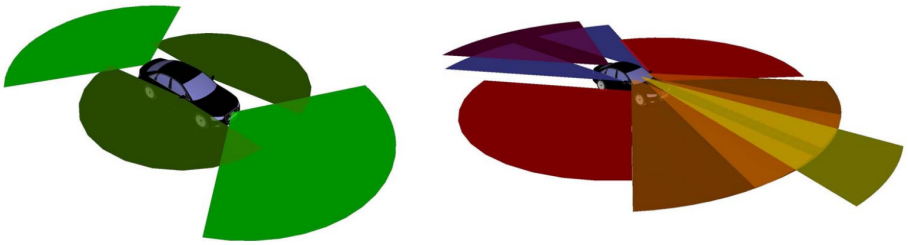
- Far: used during normal driving. Fields of view are longer in the driving direction, shorter but wider on the sides. This subsystem counts nine stereo pairs. Four of them are located behind the front windshield. One mounting a 16 mm telephoto lenses, and offering an horizontal field of view of 24.93° , and a baseline of 0.5 m, is used for detection in far forward looking direction; other three, each featuring 8.0 mm lenses, a 0.3 m baseline, and looking straight, and

22.5° left and right provides a comprehensive 90° field of view, result of the overlapping of the three stereo pairs.

The front bar still maintain the ability to slightly orient the stereo pair separately; locking screws have been placed in order to avoid unwanted movements due to vibrations. Another stereo pair, featuring 6.0 mm lenses and an horizontal field of view of 59.00° is used for detection in the far back area; with a baseline of 0.4 m can detect obstacles in the medium range.

One stereo pair is located, in addition to the one for the near system, in each rear-view mirror. Every rear-view mirror contains therefore four cameras; the integration has been done by removing the interior OEM devices, cutting the internal metal frame and the plastic shell, creating a new modified, 3D printed, shell part capable of containing the four cameras and the wires. Two stereo pairs are finally located in the mudguard and oriented vertically. They features 2.5 mm lenses and a 0.3 m baseline allowing to detect obstacles from 3.0 to 30.0 m with an horizontal aperture of 104.0° . For this system each of the two cameras is rolled by 90° in order to take advantage from they wider horizontal aperture.

In image 2.15 are shown where the camera are positioned and their field of view.



(a) (b)

Figure 2.15: Deeva autonomous vehicle cameras. Image (a) represent the near system and image (b) represent the far system.

Laserscanners: An additional perception system based on laserscanner technology has been installed. This system is used to capture ground truth data and as

safety/backup system. One eight-layer Lux 8L unit is integrated in the middle of the front bumpers. Other two, four-layer, Lux HD are integrated one at each side of the front bumper and another one behind the middle of the back bumper. These units allow to cover almost all the 360° around the car (with the exception of two small areas at the back side of the car). The bumpers have been modified with custom, handcrafted, finished with fiberglass, and repainted windows. The system also includes a fusion unit taking as input the raw laserscanners data, some car inertial data, such as the speed and the yaw rate, and GPS NMEA information and PPS. The output produced by the fusion unit is a list of tracked obstacles that may be recorded and used by the higher level software layers. the fusion unit output is provided both through Ethernet and CAN. A gateway has been placed between the fusion unit and the Comfort CAN bus of the car to filters outgoing IBEO messages to avoid flooding the this car bus.

GPS/IMU: An Oxford Technical solutions RT-3040 has been installed under the central tunnel inside the cabin. A Novatel GNSS+L-Band high performance antenna is integrated in the car roof, providing the GPS signal to the RT-3040 and to the Grand Master Clock. The open roof is still fully operational. The RT-3040 is used as backup and ground truth check unit, since the main sensor for odometry will be vision.

2.2.2 Method Overview

In this section we will describe the algorithm used to compute the trajectory that allows the car to park itself.

The proposed algorithm is a variation of the one described in section 2.1.2.

It implements a Continuous state A*, but differently from the version described before it doesn't use a grid to approximate obstacles. Each obstacle vertex is stored as a 2D coordinate. To take into account that the car is not a point, while before we did a cell expansion, now we approximate the car with three circles, as shown in figure 2.16. Now we don't do anymore cell expansion for two reasons. the first one is that we decided to not using a grid anymore. The second one is the same reason we choose to not use a grid. The reason is that the passage in which we want to park the car are very narrow, and the discretization introduced by a grid can close

them. Also if padding has to be done it has to take into account the orientation of the car, and this will introduce more noise than benefits. Using three circumference for collision check is easy and fast. We compute the minimum distance from the center of each circle to every obstacles. This distance minus the radius of the circumference represent the distance between the car and any obstacles. If this value is negative, obviously the car state in exam is colliding with an obstacle and has to be discarded.

To improve performance, we insert this distance inside the node during A* search. At each search step, when a collision check has to be done, we will recompute the distance value only if the stored distance is negative. Every time the car state is moved by a certain space, the stored distance value will be decrease by the same quantity. The stored value represent a lower bound of the exact distance from obstacles. Geometrically, after the computaion of the distance, it represent the distance from obstacle as if the car is always going toward the nearest obstacle. The reason expressed with other words, is that if car is at a certain distance from the obstacles, it has to travel at least that distance to collide with them. This trick allows to save a lot of time.

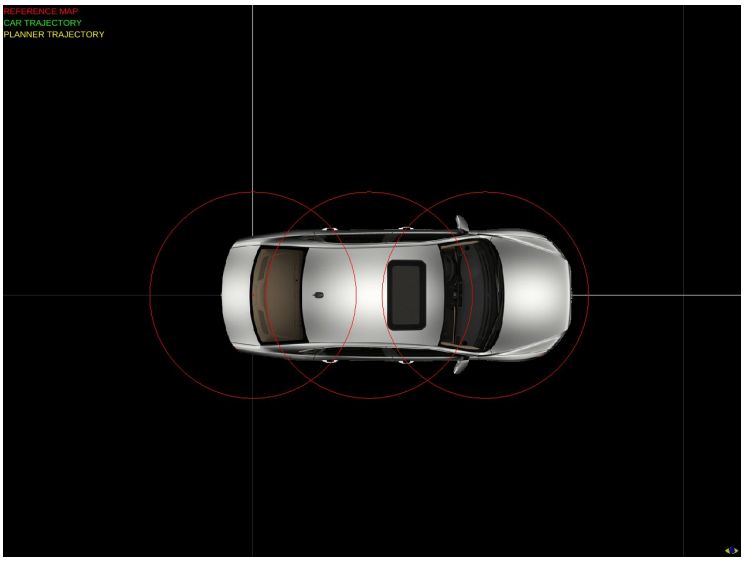


Figure 2.16: Space occupied by the car is approximated with three circles.

In this application the exploration nodes that are used by A* during the search contain the last direction of the motion, forward or backward. We choose to generate maneuvers that contains only one change of direction because almost in every situation is possible to park two maneuvers. This is not a critical constrain, as the algorithm accepts a parameter to specify the number of direction change. We set it to one anyway because the computational time improve a lot and the advantage of using three maneuvers are limited only in narrow parallel parking.

Two version of tree exploration have been tested:

- The first one is the same as the one described before, which from each state it evolves with a straight line, a circular segment to left or a circular segment to right. The circular segments have radius equal to the maximum curvature radius of the car. The olny difference is that the segment are shorter to enables more precise maneuvers.

- The second one add in the state of the nodes the curvature. Evolving the state, the possible next state are obtained maintaining the same curvature, decreasing it or increasing it. The resulting trajectory is the composition of straight lines, circle segments with various radius less or equal to the maximum curvature radius of the car and clothoid arcs to change from a circle segments with different radius.

The second approach has the advantage of using clothoids to have very smooth curvature transition. A clothoid is a curve whose curvature changes linearly with its curve length. An object traveling on a circular path experiences a centripetal acceleration. When a vehicle traveling on a straight path suddenly transitions to a tangential circular path, it experiences a sudden centripetal acceleration starting at the tangent point; and this centripetal force acts instantly causing much discomfort (causing jerk spike). The use of a clothoid arc permit to variate the centripetal force continuously creating jerk steps which amplitude is proportional to the derivative of curvature chosen for the clothoid. On the other side using to a too low derivative of curvature causes to spend more time in the transitions.

The continuity of the curvature guaranteed by the second approach allows the control to stay exactly on the trajectory. This is possible because the trajectory is a second-order geometric (G^2) continuity curve.

This feature and the improved comfort created by it make us decide to use the second approach. Another reason to switch to the second approach is that its problem of being computationally more intensive is less important because the parking path planner can work at a lower frequency than other real time automotive systems. The reason is that the parking case studio has not very dynamic obstacles and if one moves on the trajectory is not too much annoyance to stop the car.

A modification to add clothoids in the A* search regards the heuristic used. For the approach without clothoids we used Dubin's curve length as heuristic. In that case the heuristic approximates very well the final trajectory not taking into account only the obstacles and the discretization of the space between a change of curvature and the other. In the new version Dubin's curve length is still an admissible heuristic, but it doesn't take in account the clothoids neither. We choose to use the length of another class of functions as new heuristic. The functions implemented are the Continuous Curvature Paths proposed by Fraichard et al. [55]. Continuous Curvature Paths are a generalization of Dubin's curves. Similar to Dubin's curves they are composed by two Continuous Curvature Turn connected by a straight line. Each Continuous Curvature Turn is composed by a Clothoid arc followed by a circle segment with radius equal to max curvature radius, followed by another Clothoid arc. The curvature profile is shown in figure 2.17.

To follow the trajectory generated a variation of the Pure Pursuit algorithm is used. The modification regards mainly the management of the changes of direction. As first step the nearest point to the car on the trajectory is found. In the second step, when we have to find a point looking ahead on the trajectory two case are possible. If the look ahead point is on a segment of the trajectory with the same direction of the nearest point, we follow it normally. Else we find the change of direction and from that point, we extend the trajectory with a straight line. The look ahead point is taken on that line. When the nearest point is a change direction point the previous part of the trajectory is ignored in the next control steps. This avoid to fall on point already

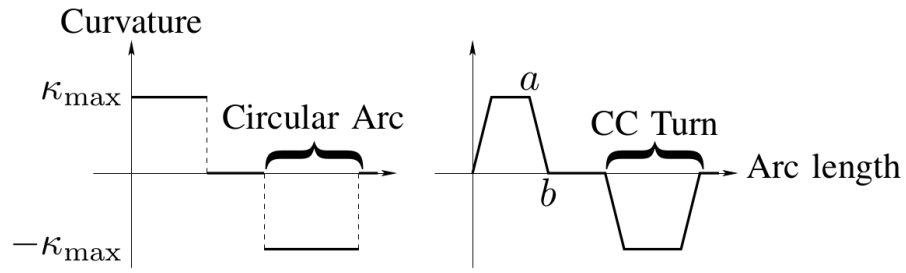


Figure 2.17: Curvature profile of a Dubin's curve on left and the one of Continuous Curvature Path on the right.

visited by the control.

2.2.3 Results

In this section we present some qualitative results.

The car successfully park itself. In figures 2.18, 2.20, 2.19 and 2.21 are shown some example of parking maneuvers. In figures 2.18 and 2.20 are shown a perpendicular park and a parallel park respectively with emphasized the intermediate positions that the car will assume during the maneuver. In figures 2.19 and 2.21 are shown a perpendicular park and a parallel park respectively with highlighted the space occupied by the car during the maneuver.

Extensive simulation have been done and the system has been integrated into the car. Initial batch of test is completed and has been successful. Those tests are been made using a GPS position taken manually as goal because the system that has the purpose of find the parking slot is not ready yet. Further test will be made to test the interaction between those system, given the difference of a fake point selected manually with the output of a parking slot detection system, which is affected by noise, delays, etc..

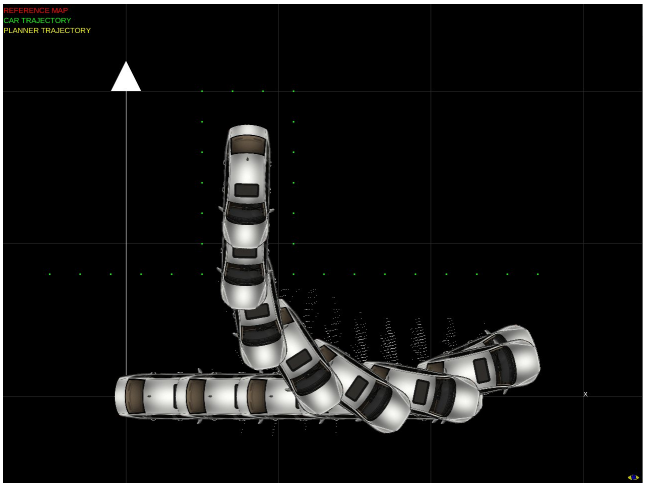


Figure 2.18: Simulation of an L park. The intermediate car position are shown.

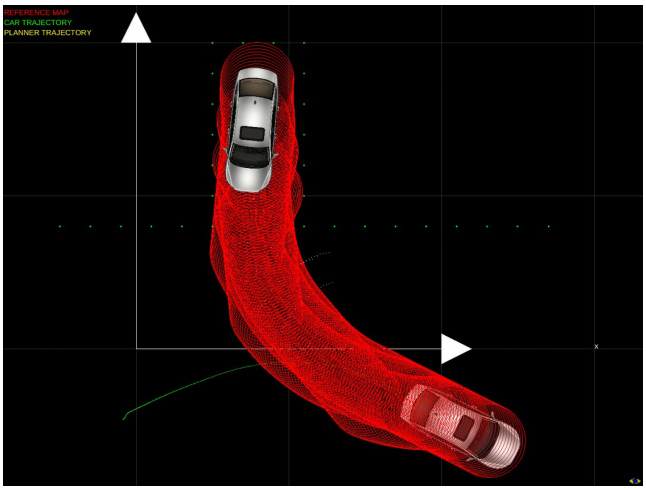


Figure 2.19: Simulation of an L park. The space occupied during the maneuver is shown.

2.2.4 Conclusion And Future Works

This section proposes a novel method to generate trajectories that allow a car to park itself. The proposed method is suitable for real-time automobile systems since it has

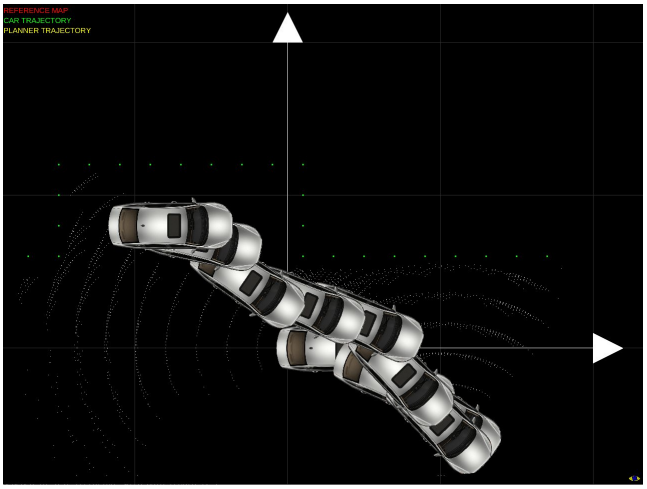


Figure 2.20: Simulation of an parallel park. The intermediate car position are shown.

Figure 2.21: Simulation of an parallel park. The space occupied during the maneuver is shown.

low computational time. We succeeded in creating a system that is able to park itself using only the initial car position and the goal position, without the use of any prior

knowledge of how the maneuvers should be.

The developed system described has proven to be able to generate behaviors that are very smooth. Through experiments on real application, it has proven to satisfy the safety measures that we impose to be run on a car in a real environment. The attention paid in the creation of comfortable behaviors, without brusque movements, have the user feel trust toward the system.

As already mentioned, next works will concern the integration of the Path Planning system with a Parking slot detection. An example of the input expected by the path planner is shown in figure 2.22. For example, the parking slot detector developed by Suhr et al. [52] gives the position of each parking slot in the surrounding of the vehicle. Fusing this result with the data given by the obstacle detector is possible to find all the free slot around the car. After an algorithm will select the best one using some politics, and then our algorithm will park the car.

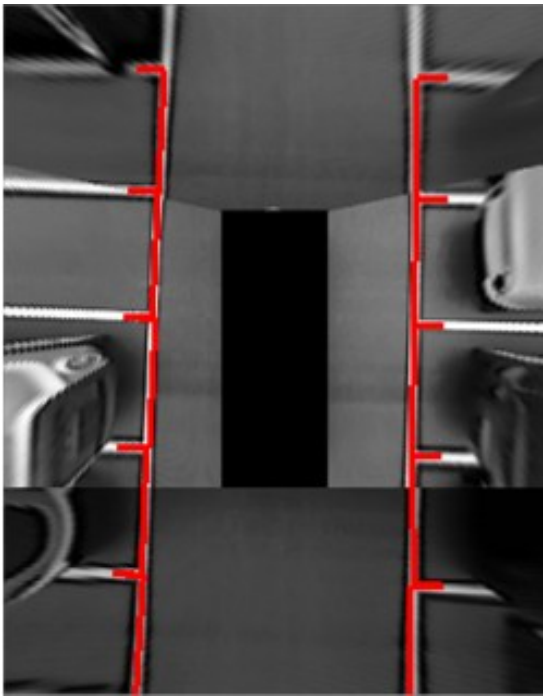


Figure 2.22: Parking Slot Detection algorithm output. in red are shown the slots. The image is taken the paper of Suhr et al. [52].

Chapter 3

Aerial Vehicles Path Planning

In this chapter we will describe the applications for the drone. The first one is a target tracking feature. The goal of the application is to enable the drone to follow a generic target selected on a image. The second one is the shape in the sky feature. The goal, similarly to target tracking application is to enable the drone to circle around a generic target selected on a image. The last application is Enhanced Radio Control. This application allows the drone operator to control the drone using a joystick avoiding obstacles.

The first section describes all the systems that work and interact with the Path Planning System and are common in all the application. The second one describes the Control system developed. The third, fourth and fifth sections describe target tracking, shape in the sky and enhanced RC algorithms respectively. Finally the last two sections collect results and conclusions.

3.1 System Description

In this section we will describe the systems that work and interact with the Path Planning System.

3.1.1 Hardware Setup

The hardware platform is a DJI Matrice 100. As shown in figure 3.1, this drone is a quadcopter from DJI (Dà-Jiāng Innovations Science and Technology Co. Ltd) a chinese technology company headquartered in Shenzhen, Guangdong. The quadcopter specification are shown in table 3.1. DJI provides an SDK to send commands to and receive data from the drone. To control the drone multiple mode are available. It can be controlled setting GPS position, speeds or directly pitch and roll. The control system presented in this dissertation uses the latter. From the drone, the sdk send GPS position, velocity, orientation and other information about the state of the drone, like if the drone is on the ground etc..

Figure 3.1: DJI Matrice 100 drone.

The drone is equipped with a gimbal, which is shown in figure 3.2. The Purpose of the gimbal is to keep the gimbal camera during flight maneouvers, even when the drone is moving avay from a target and it has to look in the opposite direction. The reason is that, as we will explain better in the next paragraph, the Obstacle detection System doesn't cover all the 360° around the drone. For this construction constraint, the drone has to look, with some tolerance, in direction of its own movement. Another

Table 3.1: Hardware specification of DJI Matrice 100 Drone.

Weight	$2355g$
Max Takeoff Weight	$3400g$
Max Tilt Angle	30°
Ascendent Speed	$5m/s$
Max Wind Resistance	$10m/s$
Max Speed Atti	$22m/s$
Hovering Time (No payload)	$22min$
Hovering Time (500g payload)	$17min$
Hovering Time (1Kg payload)	$13min$

Figure 3.2: DJI Zenmuse Z3 gimbal system.

important reason why we use a gimbal, is that the cameras used for obstacle detection have a lower quality, and cannot look straight toward the target. This problem is caused by the fact the camera for obstacle detection are fixed with the frame of of

the drone. So, if we hypothetically want to keep the target in the center of the image, the drone cannot change its own pitch and roll to move. The gimbal system gives the addictional dregrees of freedom, required to follow a generic target and record an high quality video.

To receive the images from the gimbal camera we use a DJI Manifold, shown in figure 3.3. Right now we use this device only as an interface to resend the images, but in future we plan to move the elaboration of target tracking system on this board. The specification of the DJI Manifold are shown in table 3.2.

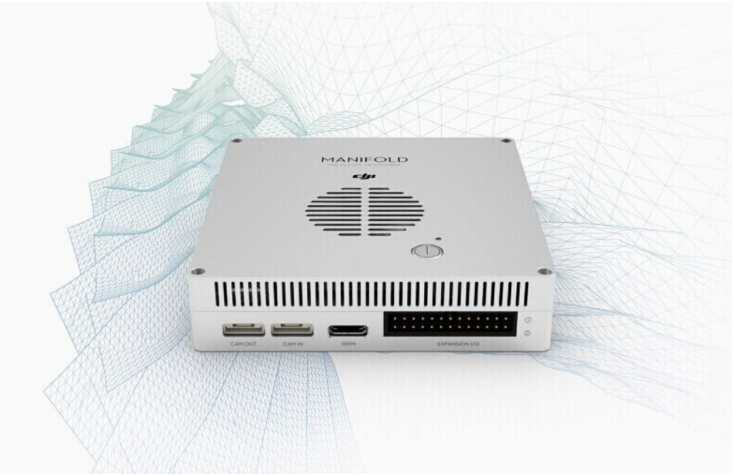


Figure 3.3: DJI Manifold system.

Mounted on the drone, there is a Pico-ITX motherboard. This is the board that do all the processing from Path Planning system to perception tasks. It run an LUbuntu 14.04 (Trusty Tahr). The board is shown in figure 3.4 and its specification are listed in table 3.3. This board is very fitted for our task because it provides excellent CPU, graphics, media performance, flexibility, and enhanced security. Furthermore it has a very small factor and low weight, that, combined with low consuption power, make this board perfect for drone application where battery duration is critical.

Table 3.2: Hardware specification of DJI Manifold.

Weight	197 g
Dimension	110 mm×110 mm×26 mm
Processors	Quad-core, 4-Plus-1 ARM Cortex-A15 MPcore Processor Low-power NVIDIA Kepler-based GeForce graphics processor Image-signal processor Ultra low-power audio processor
Memory	2GB DDR3L system RAM 16 GB eMMC 4.51 storage
Network	10/100/1000BASE-T Ethernet
Audio	Combo audio jack(mic/headphone)
USB	USB 3.0 Type-A Host connector×2 USB 2.0 Type-A Host connector×2 Micro-B USB connector (host/slave mode) Extended USB connector with DJI M-series Multicopter×2
I/O	Mini-HDMI connector Half mini-PCIe expansion slot UART port(3.3V)×2 Micro SD card connector I/O expansion headers (26pins)
Input Voltage	14 V 26 V
Operating Temperature	-10 °C 45 °C
Power Consumption	5 w 15 w

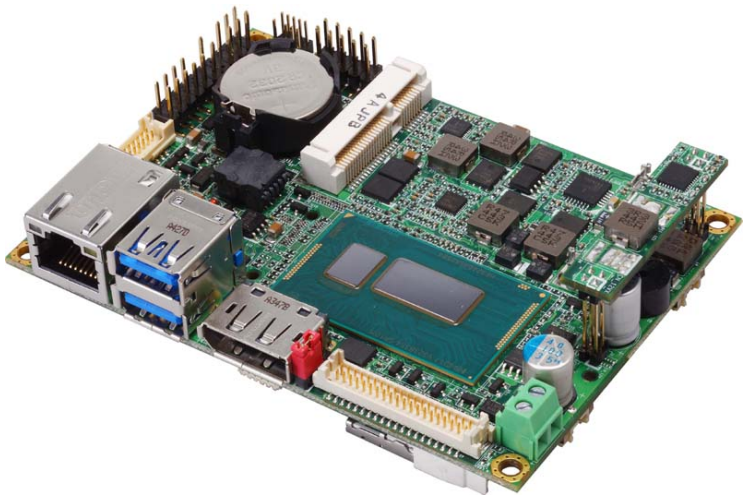


Figure 3.4: Pico-ITX motherboard.

3.1.2 Obstacle Detection System

In this section we will describe the Obstacle Detection System, one of most important system that interact with Path Planning System.

The goal of the Obstacle Detection System is to create a 3D occupancy map of the environment around the drone. This map will be used by the Path Planner to plan a safe trajectory. The Obstacle Detection System continuously update the map with sensors information.

The eyes that see the surrounding of the drone are Vislab 3DV-E intelligent cameras. As shown in image 3.5 they are mounted on a custom mount. After testing various configuration we choose to have a stereo camera pointing downward and another one looking in front of the drone pitched thirty degrees. In this configuration is possible to see the ground under the drone and the obstacle in front of it. In this way safety is guaranteed. It is possible to navigate seeing obstacles even when the quadrirotor is pitching to accelerate and to stop. Looking downward permits to compute the altitude with better precision than GPS, because GPS altitude in pretty inaccurate and don't take into account slope of the terrain. It also permits to compute a valid landing point

Table 3.3: Hardware specification of Pico-ITX motherboard.

CPU	Intel® Broadwell I7 5650U Core™ U-series Processor
Memory	One DDR3L (support 1.35V) 1333/1600 SO-DIMM up to 8GB
Integrated Graphics	Intel® 5th/4th Gen integrated HD Graphics
LVDS interface	Onboard 24-bit dual channel LVDS connector with +3.3V / +5V supply
Serial ATA	Support 1 x SATA3 with 600MB/s (6Gb/s) transfer rate
Audio	Realtek ALC888 HD Audio
LAN Interface	Intel® I218-LM Gigabit LAN
Extended interface	One PCIE mini card socket or mSATA.
Internal I/O	1 x SATA3, 1 x DVI-D, 1 x LVDS, 1 x LCD inverter, 1 x PS/2, 2 x RS232, 1 x LPC, 1 x SMBUS, 1 x Audio, 2 x USB2.0, 1 x DC Out
Rear I/O	1 x Display Port, 1 x RJ45, 2 x USB 3.0
Power	9-24V input or DC 12V(Optional)

whether with an artificial landing platform or simply to validate if the ground under the drone is a flat surface.

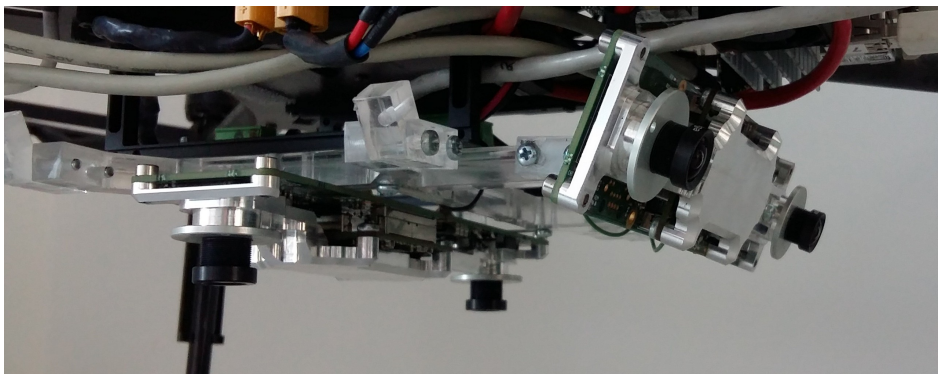


Figure 3.5: 3DV-E system with custom mount.

Each 3DV-E system have two camera. On board it mounts an FPGA that processes a disparity image. An example of a disparity image is shown in image 3.6. A disparity image or disparity map is a representation of the world where each pixel

of the image has a value equal to its displacement between right and left images. From this value is possible to compute the distance from the camera. Therefore from the disparity image given by each 3DV-E it is possible to calculate a slice of the surrounding world.

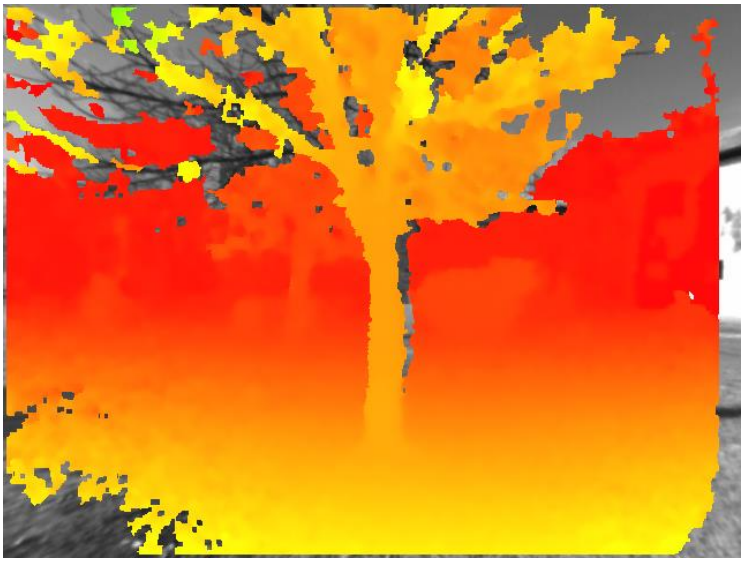


Figure 3.6: Example of disparity image.

Each disparity map is converted in a grid representation. A spherical coordinate system is chosen as shown in the left part of figure 3.7. The advantage of the use of a spherical representation in a stereo camera system are:

- "True" world representation: the coordinates system reflects how data are acquired from the physical device.
- Better definition of occluded zones, precision and uncertainty.
- A column in image coordinates has a unique azimuth angle, and a row a unique elevation. For this reasons, a lot of operations can be done (and parallelized) easily in image coordinates.

The 3d point cloud is stored in a discretized spherical map, as shown on right in figure 3.7. This representation has few advantage. Managing the discretization step on θ and φ is possible choose a optimal trade off between precision and computational complexity, with consequent control over the quantity of memory usage. The output in this phase is a Lidar-like representation of the world, with one distance per beam. Each beam is identified by a discrete θ and a discrete φ. Each beam or cell in the spherical map correspond to rectangle in image coordinate.

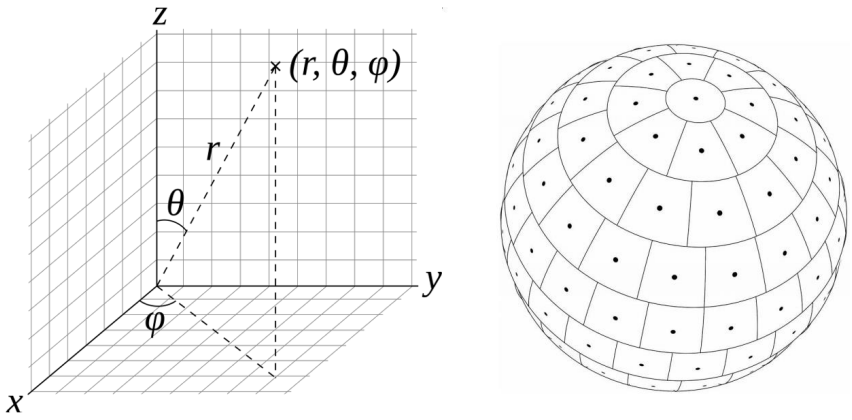


Figure 3.7: On left spherical coordinate system, r represent radial distance, θ zenith angle (elevation), φ azimuth angle. On right a discrete version of a full 3D spherical map.

Each local obstacle detection system take as input a disparity image 12.5 times per second and create a discretized 3d spherical map, and then send it to the Data Fusion module. An example of this output is shown in figure 3.8.

The Data Fusion module takes in input various spherical grids from different points of view and use them to update a global map. Internally Data Fusion module has a global map that describe all the surroundings that are already been explored. This global map is a cartesian 3d map. It is georeferenced, with abscissa that point toward Nord and ordinate that point toward West. Each cell of the map contain a probability of being part of an obstacle. Data Fusion module uses information from spherical grids to update such probability. The steps for updating the global map

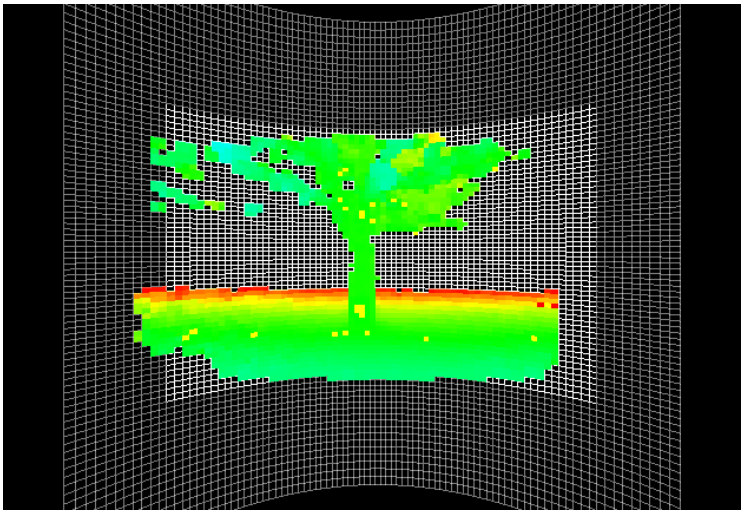


Figure 3.8: Example of discretized 3d spherical map.

when a new spherical grid is given are:

- Compute segment from the center of the spherical grid to each occupied cell.
- For each segment, lower the probability of being occupied of each cell in the global map that touch that segment.
- For each segment, raise the probability of being occupied of each cell in the global map that are at the end of the segment.
- For each non occupied cell of the spherical grid, lower the probability of being occupied of each cell in the global map that touch a segment that start from the center of the spherical grid and it is cast with length equal to the depth of field of view of the sensor.

Data Fusion module send an occupational grid to Path Planner at a rate of $10Hz$. This grid is a portion of the global grid centered on the drone. Each cell of the occupational grid is set to free or occupied applying a threshold to the probability of the correspondent cell in the global grid. In figures 3.9 and 3.10 is possible to see two

examples of the resulting occupational grid, seen from the perspective of the drone and from above.

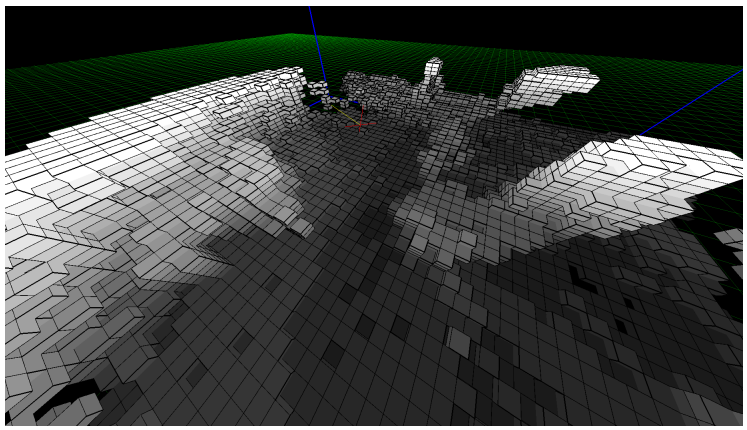


Figure 3.9: Example of Data Fusion Module Output (FPV perspective). The red cross in the center represent the drone.

3.2 Control System

In this section we will describe the control system that permits the drone to fly following a trajectory and to stay still in air.

The main feature that is required by the control is to generate smooth control commands, that prefer to move the drone softy respect to exact positioning. The main reason is that the application taken into account in this dissertation always put the quadrotor in close contact with human operators so it is mandatory to transmit a strong sense of confidence avoiding sudden direction changes. Another constraint of the control system is to have low computational requirement. The target platform where the control will run is an low power ARM CPU, with very low computational power. For this reason, we choose to implement simple and efficient control method.

There are two main mode in which the drone can be:

- Hovering: the drone will remain still in the air.

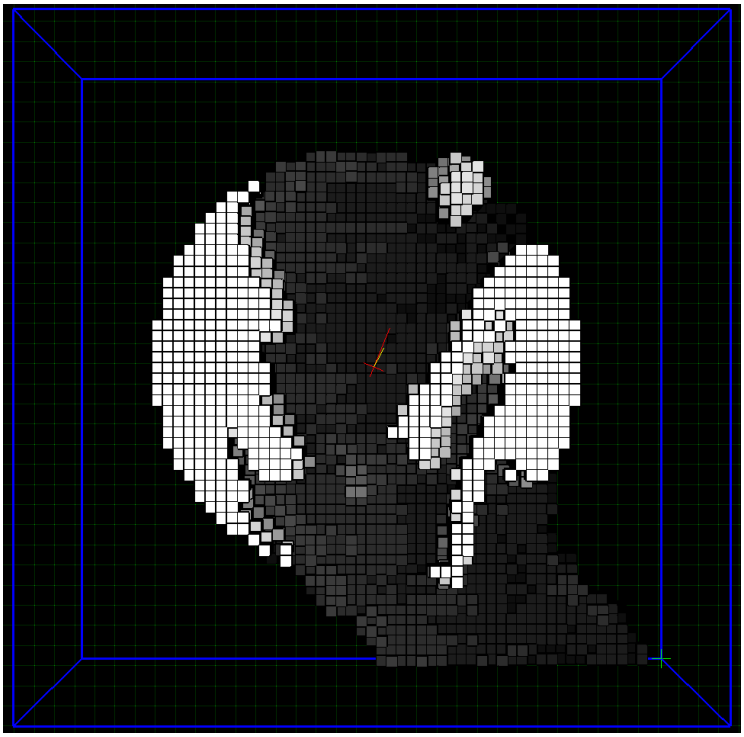


Figure 3.10: Example of Data Fusion Module Output (Bird eye view perspective). The red cross in the center represent the drone.

- Trajectory Following: the drone will follow a trajectory computed by the Path Planner System.

Hovering is required during initialization when the Path Planner System is not ready to provide valid trajectory. Other times, the drone has to be stable in the air, like during Target Tracking Feature when target is not moving. In figure 3.11 is shown the diagram of the control system. It takes in input the position setpoint, that can be the first position of the drone when the control is turned on or the static position provided by the Path Planner. Then it uses the current position of the drone as feedback to compute a positional error. The control is decoupled and controls separately along x and y. Two PID controller are used to compute a speed setpoint from the position

error.

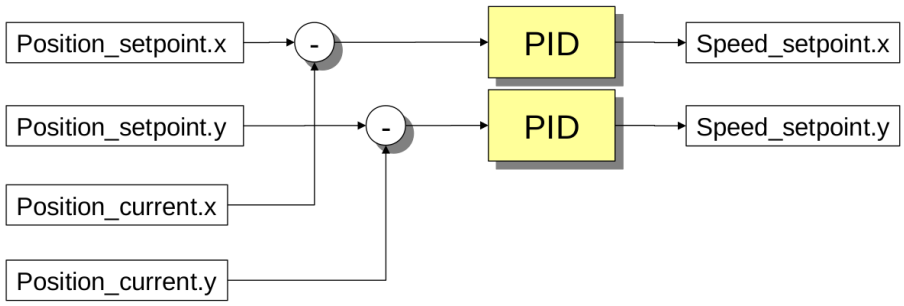


Figure 3.11: Diagram of the longitudinal position control during hovering.

During Trajectory Following mode, the input of the controller is a trajectory. In figure 3.12 is shown the principle of operation of the controller. First the closest point of the trajectory to the drone is found. From this point another point ahead on the trajectory is chosen. The distance between the closest point and the Look-ahead point depends on the speed of the drone. The faster the drone is, the farther the controller will look on the trajectory. The vector that connect the actual drone position with the Look-ahead point determines the direction of the speed setpoint outputted. The norm of the speed is given by the Path Planner as additional information in each point of the trajectory. This type of controller works in the same mode of Pure Pursuit Tracker used on cars. It doesn't use integrative control to reach a stable configuration, but ensure smooth output that stabilize the drone on the trajectory.

When the drone reach the end of the trajectory, the controller switch to Hovering mode using the end of the trajectory as the position goal. This switch permits to take advantage of the PID controllers to stabilize the drone and reject possible disturbance, like wind.

The Speed setpoints computed by the previous passages are given as input to the Speed Controller. It converts them from global coordinate to a local coordinate system oriented as the drone. This change of coordinate permits to simplify the next step of speed control. This simplification occur because in local coordinate the x axis

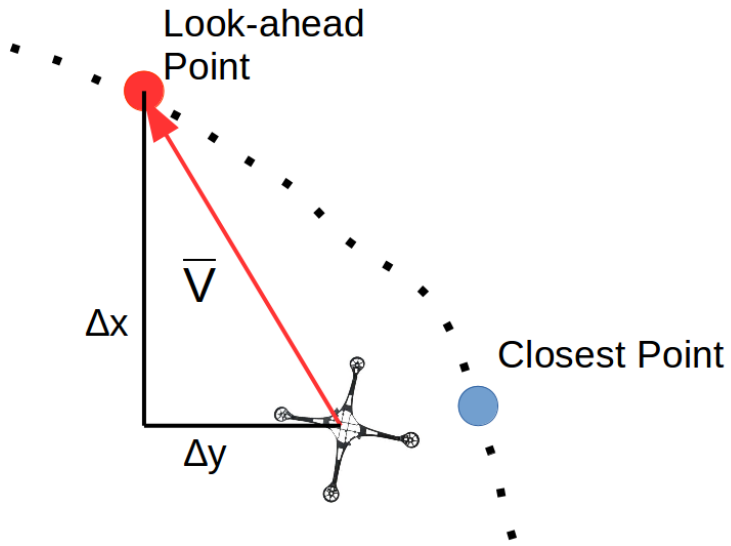


Figure 3.12: Diagram of the longitudinal position control when following a trajectory.

is the direction where the pitching action act, and the y is the one for rolling. At this point the controller uses the current speed of the drone as feedback to compute a speed error. The x and y components of speed error are given to two PID controller that compute directly the pitch and roll directly.

Diagrams of altitude control are shown in figures 3.14 and 3.15. For the control of the altitude, we read the altitude setpoint from the nearest point on the trajectory or we take the requested altitude from the hovering position. Afterwards the altitude error is calculate from this setpoint and current altitude. We choose to use a PID controller to compute vertical speed setpoint.

Using the current vertical speed, we calculate a vertical speed error and with a PID controller we obtain a vertical acceleration setpoint. To this acceleration is added gravity acceleration to take in account that when we want to have no acceleration, there is gravitational force acting on the drone that we should compensate.

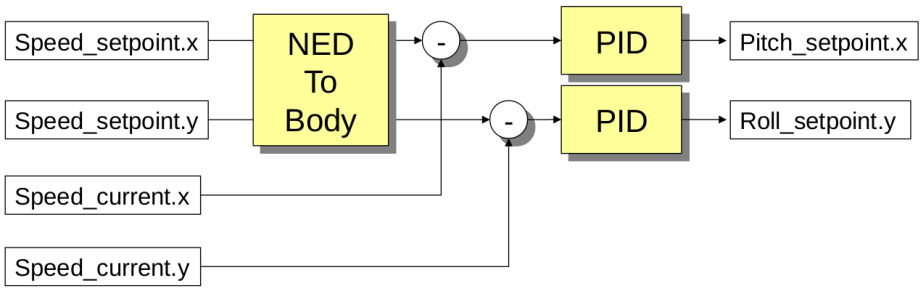


Figure 3.13: Diagram of the longitudinal speed control.

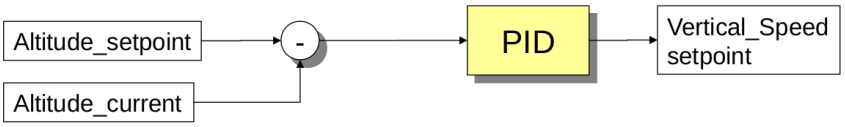


Figure 3.14: Diagram of the altitude position control.

Another element taken into account and compensated is a additive effect proportional to the battery level. When battery is fully charged the drone motors have more responsiveness than when the battery level is low. The acceleration obtained is then multiplied by the mass of the drone to convert it to the force setpoint. Finally the force is multiplied by $\frac{1}{\cos(Pitch)*\cos(Roll)}$. This multiplicative component takes into account the rotation of the drone respect to the vertical axis. If the drone is perfectly horizontal all the force generated by the rotors is used to maintain its height. However, when the drone has pitch and roll different from zero for moving or correcting its position, part of the force partially point laterally and this need to be taken into account. After this last correction the thrust is ready to be sent to the flight controller.

Lastly a simple heading controller has been implemented as shown in figure 3.16. This controller is useful in some application like Target tracking to follow the target with the heading of the drone when it is at the right distance from it or to follow the trajectory when moving. When following a trajectory the drone will point its headind to look ahead position described in longitudinal control. Whether with a target or with

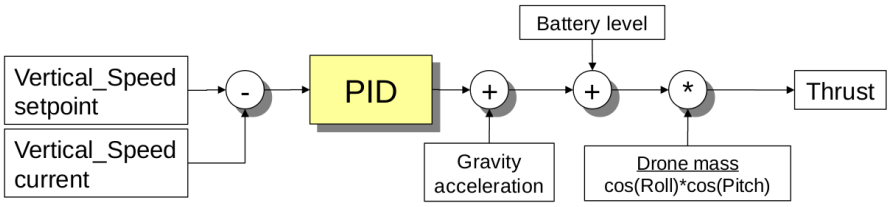


Figure 3.15: Diagram of the altitude speed control.

trajectory, after an heading setpoint is chosen heading error is calculated using current heading. This error is reduced to zero using PID controller, that gives as output the yaw rate requested by the flight controller.

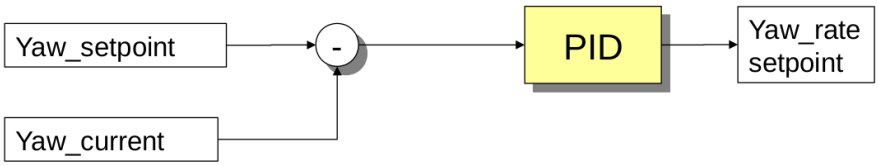


Figure 3.16: Diagram of the heading control.

3.3 Target Tracking Planner

Target tracking system has the goal of following a target. The target can be a pedestrian, a bike, a car, a boat, or anything that can be seen by a camera. During the movements done to follow the target the drone has to avoid obstacles on his path.

The architecture of the system is shown in figure 3.17.

Target Detection System takes in input the image of the gimbal system. As first step the human operator selects the target. After the target is correctly selected, the Gimbal Control System will point the camera toward the target in order to maintain it in the center of the image. The Gimbal Control System continuously controls the position of the gimbal taking into account the position, the velocity and the angular

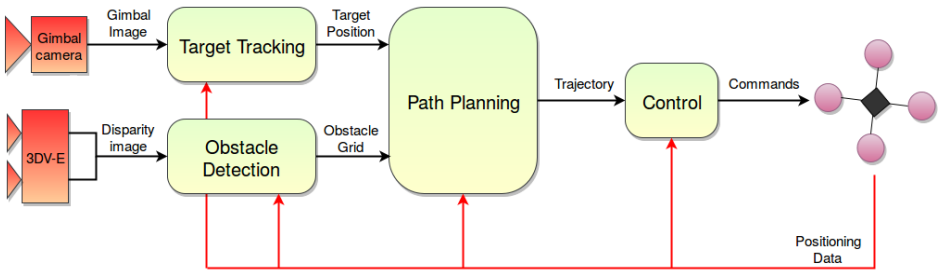


Figure 3.17: Architecture of Target Tracking System.

velocity to provide a stable image centered on the target. It uses the movements and rotations of the drone as feedforward action, to maintain the camera focused on the target when the drone is moving, rotating the gimbal in the opposite direction. It controls with two PI controller the pitch and the yaw of the camera. The vertical and horizontal displacement in pixels is given as input of the controllers.

Obstacles Detection System is described in section 3.1.2.

The algorithm used to compute the trajectory is a multy-goal A*. As classic A* it explore over the grid taken from the perception using an heuristic that measure the distance from the goal. In this A* variation, there is not only one goal but a set of goals, as shown in green in figure 3.18.

In this application we choose to use circles but the algorithm works with any shapes, for example is possible to use ellipses placing one focus point on the target and the other one in the direction of the movement of the target with a focal distance proportional to the speed of the target.

To allow the algorithm to converge over multiple goals we modify the heuristic as follows:

$$H(n, goal[]) = \min_i \left\{ H(n, goal[i]) \right\}$$

Where $H(state, goal[])$ is the heuristic of state n to set of goals, $H(n, goal[i])$ is the heuristic of state n to the *i-th* goal. The new heuristic, $H(n, goal[i])$, is admissible so it never overestimates the cost of reaching the goal, if the basic heuristic is admissible. Following the proof:

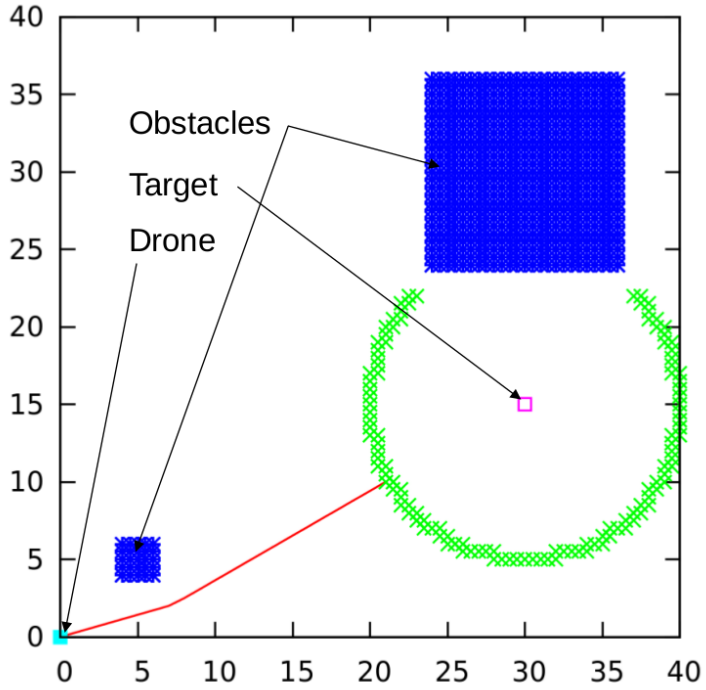


Figure 3.18: Example of Target Tracking algorithm. Obstacles are shown in blue. Cyan dot is the drone position. Empty cyan dot represent target position. Green x are the point of the circumference around the target that are the goals for the A* algorithm. Red line is the computed trajectory.

if $H(n, goal[i]) \leq Real\_cost(n)$ *then*

$$H(n, goal[]) = \min_i \left\{ H(n, goal[i]) \right\} \leq \min_i \left\{ Real\_cost(n) \right\} \leq Real\_cost(n)$$

Another modification introduced to switch to multiple goal is that for each node selected from the open state space, which are the nodes selected to be possible solution nodes, as neighbor of already visited nodes, but yet to be visited by A* search, we should control if it correspond with any goal of the set.

This two modification slightly increase the computational proportionally to number of goals. To maintain the performance adequate, we introduce some modification to improve computational time.

One modification is that we use a bounded relaxation of the heuristic admissibility criterion. An admissible heuristic guarantees an optimal solution path, but it forces A* to examine all equally meritorious paths to find the optimal path. It is possible to speed up the search at the expense of optimality by relaxing the admissibility criterion. Oftentimes we want to bound this relaxation, so that we can guarantee that the solution path is no worse than ε times the optimal solution path. This new guarantee is referred to as ε-admissible. We choose to use a Static Weighting method. If $H(n)$ is an admissible heuristic function, in the weighted version of the A* search one uses $H_{weighted}(n) = \varepsilon * H(n), \varepsilon >$ as the heuristic function, and perform the A* search as usual. The path hence found by the search algorithm can have a cost of at most ε times that of the least cost path in the graph. However the exploration will be faster than using $H(n)$ since fewer nodes are expanded. In our tests we used $\varepsilon = 1.5$, which is a good compromise between goodness of trajectories and acceptable computational times. Raising ε too much cause the search algorithm to become to much "greedy" and expanding only nodes near the goals never evaluating the early part of the exploration. In figure 3.19 it is possible to se an example of the effect of weighting the heuristic. Classic A* search expand a lot more nodes before reach the goal, but with weighted heuristic the path goes straight and do a 90 degrees turn, which is obviously non optimal. Most of this non optimal point of the trajectory are solved by the Ray Casting algorithm that will be explained later in this section.

Another modification to improve performance involve the moment when the check if new node are already in the open set is done. In classic implementation, when the best node in the open set is evaluated, if it doesn't collide with any obstacle, each node in its neighborhood is inserted in the open set if not already present. We choose to eliminate this control and, instead to add a control during the extraction of the best node from open set. If the best node is already in the closed set, it is discarded immediately. From the point of view of the logic of the algorithm, this modification doesn't change anything. Computationally as two effect:

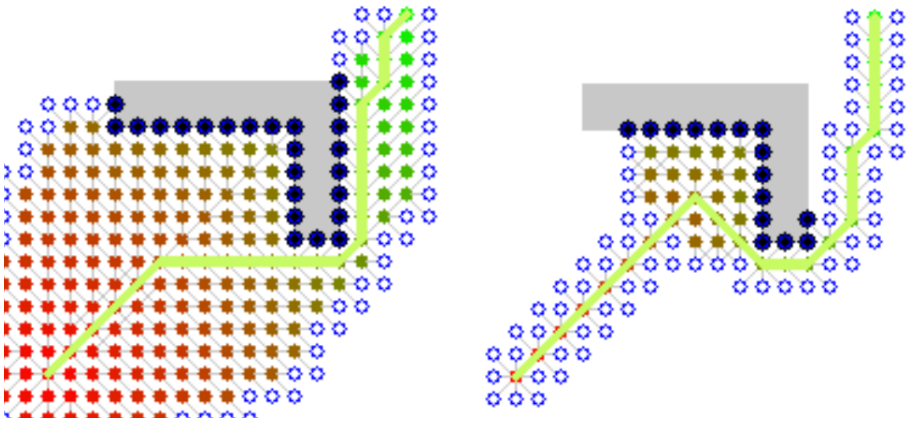


Figure 3.19: Example of problem introduced by weighting the heuristic. On the left the solution found by classic A*. On right solution found with $\varepsilon = 5$. The empty circles represent the nodes in the open set, those that remain to be explored, and the filled ones are in the closed set.

- In classic A*, it is checked if new nodes are in open set. In our implementation best nodes of open set is checked if present in closed set. In most cases open set is much larger than closed set, so the first check is much slower.
- In classic A*, neighbor nodes are not inserted in open set if already in there, in our implementation yes. Therefore in our implementation open set grows bigger that classic A*.

For our system, computational time is a more important bottle neck than memory usage so we choose to apply this change.

Another change in classic A*, is the shape in the neighborhood as shown in figure 3.20. This change permits to find better solution because it distinguish the action of going straight and than 45 degrees diagonal form going directly in the cell 26 degrees diagonally, even if its the same cell. Moving to this cell cost $1+\sqrt{2} \simeq 2.414$ in classic solution but only $\sqrt{1^2+2^2} = \sqrt{5} \simeq 2,236$. This difference can let discern the algorithm from different path that otherwise will have the same cost.

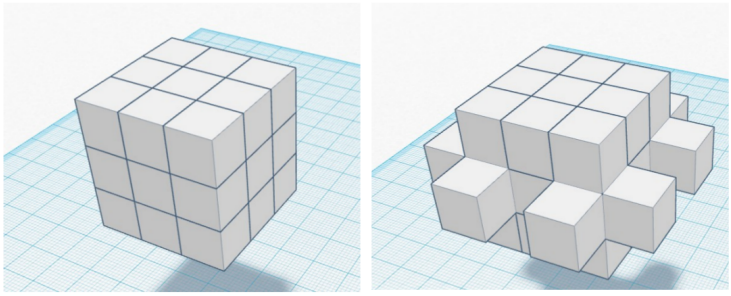


Figure 3.20: Neighborhood shapes. On the left the classic cube with 26 neighbors (27 cells minus the central one). On the right the modified cube with 26 neighbors plus 8 diagonal neighbors.

After a trajectory is found by the algorithm, two step smoothing is applied. The first smoothing algorithm is named Ray Casting. An example of the action of this phase is shown in figure 3.21. The main goal of this phase is to eliminate the unnatural path generated by A*. As we can see in the example, which represent a simplified environment, A* create a 45 degrees path followed by a straight line. This is caused by the fact that A* can move the exploration on the grid only on 8 direction (in our implementation we have more direction, but the problem remains). Ray Casting algorithm resolve this problem removing cell of the solution that are not essential, that are the ones that if they are deleted and the remaining cells are connected with lines, those lines don't collide any obstacles. The algorithm iteratively select couple of non consecutive solution cells and if the line connecting the two cells doesn't collide with any obstacle, all the solution cells in between are deleted. As mentioned before this smoothing step allows to improve the solution even when the solution is not optimal (see weighted heuristic above), allowing movement not feasible while moving on a grid.

The second smoothing step is the application of a spline fitting algorithm. This step allows to eliminate sudden changes in the direction of the trajectory. Before this step the trajectory is sequence of lines, whereas after the trajectory become a curve with continuous direction and piecewise continuous curvature. The trajectory is fitted

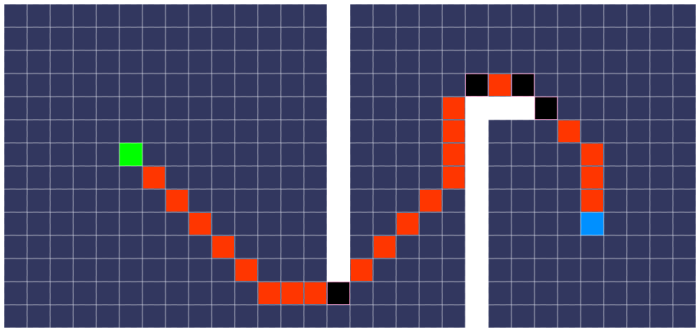


Figure 3.21: Example of Ray Casting algorithm on simplified 2D grid. Green square is starting position and cyan is the goal. In red is shown the input trajectory. Black square are the output of Ray Casting algorithm.

with Bspline with degrees 3. In addition of having smoother trajectory this step allow the trajectory to have a curvature different from zero as before spline fitting. This will be used to compute lateral acceleration, that will be used in computing the speed setpoint.

After the smoothing steps, the speed profile is computed. This speed setpoint will be used by the Control to have a speed to reach. In some configuration, when the drone has to look where is moving, every control cicle the control will slow down the drone when the drone in not looking in the direction of the speed.

Following the passage to determine the speed of each point of the trajectory.

$$speed[i] = min\left\{ max\_speed \ , \ \sqrt{\frac{max\_lateral\_acceleration}{curvature[i]}} \right\}$$

For each point, speed is initialized to the minimum between the maximum speed read from configuration and the square root of maximum lateral acceleration read from configuration divided by the curvature in the point.

$$speed[0] = min\left\{ speed[0] \ , \ current\_speed \right\}$$

$$speed[i] = min\left\{ speed[i] \, , \, \sqrt{speed[i+1]^2 + 2 * acceleration * step} \right\}$$

First point speed is set to current drone speed. For each other point, starting from second point, speed is propagated forward using the acceleration chosen in the configuration options. Both this settings are done only if new speed is lower than the one already calculated.

$$speed[end] = min\left\{ speed[end] \, , \, target\_speed \right\}$$

$$speed[i] = min\left\{ speed[i] \, , \, \sqrt{speed[i+1]^2 + 2 * acceleration * step} \right\}$$

End of trajectory point speed is set to current target speed. For each other point, starting from second to last point, speed is propagated backward using the deceleration chosen in the configuration options. Both this settings are done only if new speed is lower than the one already calculated.

In figure 3.22 is shown an example of speed profile. The acceleration ramp and the deceleration ramp are section of parabola because on the abscissa there is the space traveled, if it is converted in a time version the parabola sections will become straight lines.

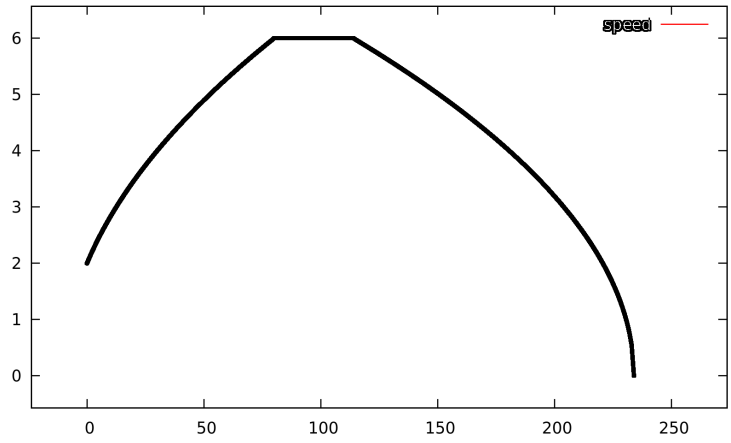


Figure 3.22: Example of speed profile. On abscissa is represented the space on trajectory and on ordinate the speed in meter per second.

The algorithm described is always used when the drone has to approach the circumference. When the drone is on the circumference, small movements of the target cause the drone to apply little correction to remain exatly on the circumference. To provide behaviors less shaky, an hysteresis feature is added. when the drone is at the exact distance from the target, the drone change to holding position control mode. To exit from this condition and start to follow again the target, the drone has to exit from an annulus centered on the target.

Another improvement used to enhance the smoothness of the drone movement, helping the control system, is the choose of the starting point. There are two different case that are treated different. The first case is when the drone is still or moving slowly under a threshold. In this case, if is first run of the algorithm all the planning start from the current position. Else if the drone is sufficiently near the trajectory, the planning will start from the nearest point on the trajectory else it will start from current position. This choice allow to maintain the trajectory more stable. Without this modification when the drone has a trajectory following error caused by wind, control errors, etc., the planned trajectory will move causing even more error that will not be compensated by the control system. The second case is when the drone is moving fast. In this case, if the planning start from current drone position, the planner can calculate a trajectory that start with a direction different from the speed of the drone itself. If this happens, it is impossible for the control system to stay on the trajectory but it will at least overshoots causing the drone to go in an unplanned position. To eliminate this problem, planning will start from a future position of the drone, that is the current position translated using the current speed multiplied by a time constant. In our experimentation we choose as prediction time one second.

3.4 Shape In The Sky Planner

Another application developed for the drone is the Shape in the sky Planner. Its goal is to compute a path that permits the drone to do shapes around the target. I can be seen as a variation of Target Tracking Planner but with few differences. It has the same interaction with Target Tracking System and Obstacles Detection System, so

for further information see section 3.3. For what it concerns the functionality of the system the main difference is that in Target Tracking application the drone stop when it is at the right distance from the target, while in Shape in the sky mode it start to move around it. As in Target Tracking, the shape that the drone will "draw" in the air is generic, but in our experimentation we choose to circle around the target. in figure 3.23 is shown an example of the output of the algorithm.

The implementation works with a multiple step approach. Firstly a multy goal A* is executed. Its working principles are described in Target Tracking Planner (3.3) so we will skip them here. After this phase the algorithm has knowledge of the nearest goal taking into account all obstacles and the best path to reach it. To that path starts the procedure to circle around the target. Starting from the nearest goal, we compute the path from a goal to the next on the circumference. Every path is computed using the A* algorithm described before but with only one goal. Then it connect each path to each other. The Ray Casting algorithm is applied with the modification of don't delete goals point on the trajectory.

At this point of the algorithm we have a trajectory that circles around the target but its not guaranteed that its the best one. For example if the algorithm output this trajectory, it can happen that the drone is on the circumference and the nearest goal in the opposite direction of the one we want to circle. The trajectory will be going in one direction until it reach the first goal and than change direction to circle around the target. This is obviously very uncomfortable. Another problem occurs when the drone is outside the circumference. With the hypothesis of field without obstacles, the resulting trajectory will be a straight line toward the target and, when reaching the circumference, a 90 degrees turn to start circling around the target. Again this is obviously very uncomfortable.

To solve this problems, we compute more trajectory and choose the best. The other trajectory with which we compare the one already computed, will be the ones that anticipate the circles around the target. So the ones that are not pointing to the nearest goal but look ahead on the circumference. To compute the first one, a path to connect the current drone position to the goal x meters ahead the nearest goal is calculated. This path is then connected with the path to next goal and so on. Next

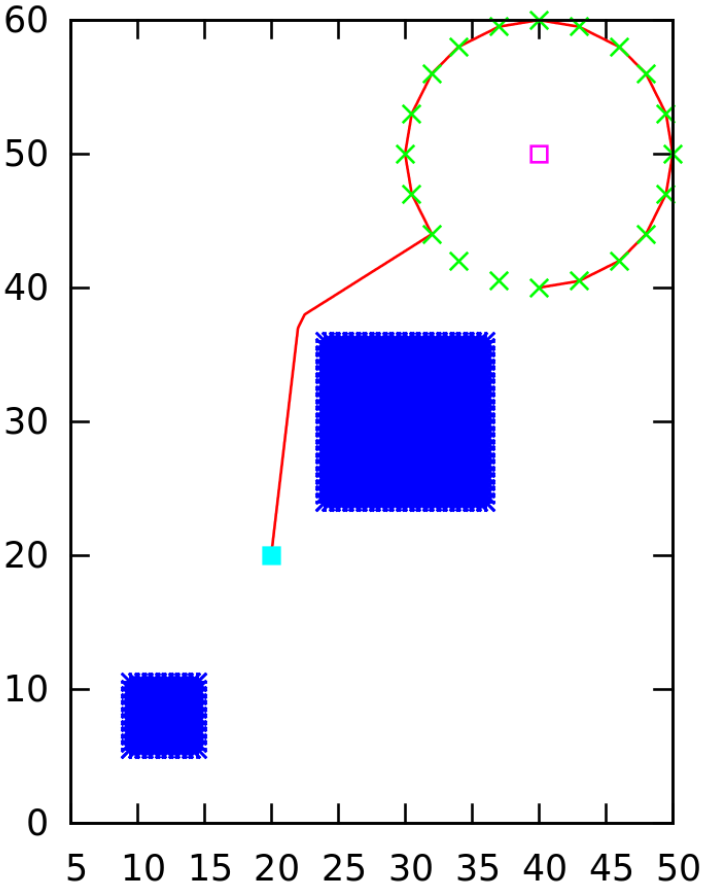


Figure 3.23: Example of Shape In The Sky Planner algorithm. Obstacles are shown in blue. Cyan dot is the drone position. Empty cyan dot represent target position. Green x are the point of the circumference around the target that are the goals for the A* algorithm. Red line is the computed trajectory.

trajectory is computed in the same way choosing as first goal the one $2 * x$ meters ahead the nearest goal is calculated, and so on for the other trajectory. In our experimentation we choose to use x equal to 1 meter.

To choose the best trajectory, a cost function is created. This function has to take

into account the length of the trajectory and its smoothness. The choose function is the time needed to travel along the trajectory. The function is as follows:

$$Cost = \sum_{for\ each\ point\ i} \left\{ \frac{trajectory\_step}{\min\left\{ max\_speed, \sqrt{\frac{max\_lateral\_acceleration}{curvature[i]}} \right\}} \right\}$$

This is a very good function to be used. It doesn't need the tuning of any parameters, its easy to understand and easy to implement.

Using this cost function, the trajectory with lower cost is chosen.

Another difference with Target Tracking Path Planning is the use of two different speed limit. One is for the drone when is approaching the circumference, and the other is for when the drone is already circling on the trajectory. The final speed profile will have an acceleration ramp to reach speed limit outside the circumference, a deceleration ramp to reach the speed limit on the circumference, and a constant circling speed.

3.5 Enhanced RC Planner

Enhanced RC Planner is an application developed for the quadrirotor that has the goal to enable an human operator to control the drone with a radio controller in a safe mode. With safe mode, we mean that the operator con give any command but he will not be able to have the drone collide with any obstacle, even if he want and try to do it.

In figure 3.24 it is shown the architecture of Enhanced Radio Control System. The architecture is similare to the one shown for Target Tracking Planner. It takes in input an occupancy grid from Obstacle Detection system. For further information on Obstacle Detection system see 3.1.2. Another input is the state of the Radio Control Joystick. From the DJI SDK we take as input four values. One for yaw rate, one for altitude change, one for X speed and one for Y speed. Those four values are the state of the left and right lever of the joystick. Those values vary from -100 to 100. The output of the planner is a trajectory and is given in input to control system.

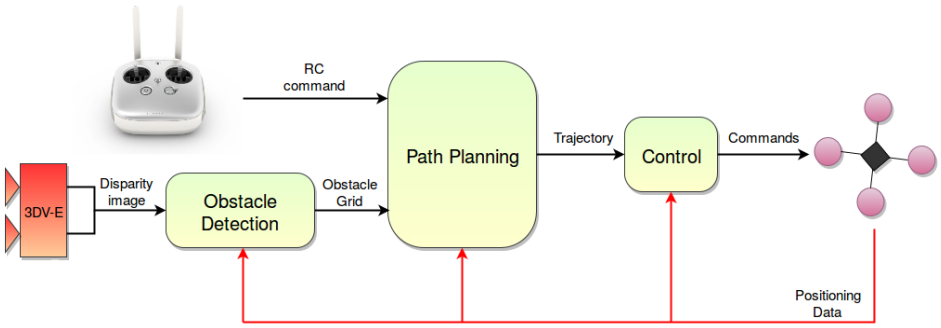


Figure 3.24: Architecture of Enhanced Radio Control System.

The value for yaw rate is send directly to the control because it is not possible to crash the drone by only rotating it.

The other three values are used to compute a requested speed. This speed is a 3d vector that has a and a maximum vertical speed. The abscissa and ordinate of the speed is normalized so that when both values are at 100 the planar speed has norm equal to the maximum planar speed chosen in the configuration. Also vertical speed is normalized so that value 100 corresponds to maximum vertical speed. To improve control of the drone the change of value near zero influence less the change in speed. This feature is implemented using a non linear normalization. Near zero, when non normalize speed norm is less than 20%, we use a smaller normalization factor. Otherwise we use a bigger normalization factor and an additive offset to maintain continuity in the speed. This feature improve a lot the maneuverability of the drone, avoiding error in the commands caused by over sensibility of the radio control commands.

The Radio control planning system can be in two state, as shown in figure 3.25. The two state are Hovering and Navigation.

During Hovering mode the drone will stay still maintaining the last goal computed by navigation. When the Enhanced Radio Control System is turned on, it take the first drone position as first goal. The system remains in Hovering mode until the Requested speed become greater than a configuration threshold.

During Navigation mode the drone will follow a trajectory that represent the re-

sult of elaboration that has as input the Requested speed. The Enhanced Radio Control System will compute this trajectory using not only the Requested speed but also the current drone speed. In the next paragraph we will explain more detail about this method. When the requested speed and the current speed are lower that a threshold, the Radio Control System change back to Hovering mode. The goal for hovering is computed as follows:

$$Hovering\_goal = current\_position + \frac{current\_speed^2}{2 * deceleration}$$

The term $\frac{current\_speed^2}{2*deceleration}$ represents the distance traveled by the drone that is stopping with constant deceleration. This allows to have a very smooth transition form Navigation to Hovering because the control has to decelerate constantly to reach the Hovering goal.

Moreover, to improve the smoothness of the movements and to permit safe maneuvers during the stopping phases, the switch to Hovering state happens only when the drone has already reduced its speed under a certain threshold. Until that moment, when the Requested speed is under the threshold, the Radio Control planner use as goal speed direction the current drone speed. In this way if an obstacle is seen after the control levers are released, the drone can gently stop avoiding it decreasing its speed slowly.

After a requested speed is computed during Navigation state, two mode of establishing the goal for the planner have been tested.

- Variable distance: in this case the goal is projected from current drone position using requested speed multiplied by a maneuver time. The goal represent the position of the drone maneuver time second in the future if it moves at the requested speed.

- Fixed distance: in this case the goal is projected by a fixed distance chosen in the configuration in the direction of the requested speed. The goal has no relation with time but represent only the desired direction by the operator.

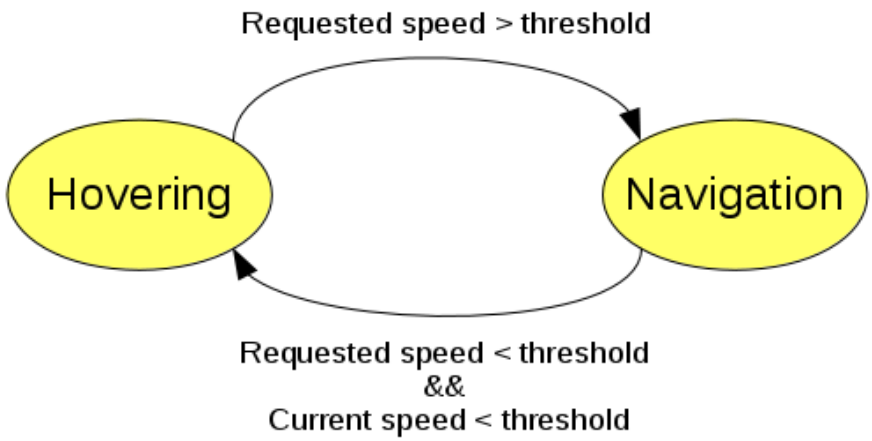


Figure 3.25: State diagram of Enhanced Radio Control System.

The variable distance speed is the first method we tried but has some disadvantage that made us change to fixed distance. The biggest problem is that when the operator want to go very slow the goal is very near to the drone position so the planned trajectory is very short. This fact is a problem because if the operator point in the direction of an obstacle the drone will go slowly toward the obstacle and at the last moment it avoid it. This is a problem because it is not comfortable because, from the point of view of the operator that don't see at the obstacle map, it seems that the drone is not seeing the obstacles. In Fixed distance mode the drone has always a long trajectory that is followed at requested speed, both if it is low or if it is high. In this case if the operator point an obstacle the drone start to avoid it at the distance of goal projection, generating a smooth trajectory that avoid the obstacles gently.

As mentioned in Target Tracking planning system, the beginning of the planning don't always start from drone position. For further information see the end of section 3.3.

3.6 Results

In this section we present some qualitative results.

The drone successfully complete all the tasks given. Starting from Target Tracking, the drone correctly follows the target with a very smooth behavior. The designed system allows the drone to automatically follows the target in any circumstances, with almost any kind of obstacles. As shown in figure 3.26, the Target Tracking platform has been tested extensively following pedestrian. In the figure is shown a difficult scenario because there are a lot of tree, there are a lot of marked shadows, horizontal branches and so on. A difficulty for Target Detection system is that the target change its appearance a lot when passing from a dark area to one with direct sun. Even with this complex environment, the system has been able to follow correctly the target avoiding obstacles.

Figure 3.26: Example of result of Target Tracking Algorithm. In this case the drone is following a pedestrian in a wood.

The system has been also tested in the following of cyclists as shown in figure 3.27. For the Path Planning system, the main difference with pedestrians is that bicycles go faster but with a more predictable path. In the image is shown a bike that travel on a road. In the chosen frame the bike is correctly passing between a truck and a wall. In this test the bike has reached a maximum velocity of 30 kilometers per hour.

The drone has been able to follow correctly the bike avoiding obstacles maintaining an high speed.

Figure 3.27: Example of result of Target Tracking Algorithm. In this case the drone is following a cyclist on a road with some cars.

The system has been also tested in the following of vehicles as shown in figure 3.28. As with the transition from pedestrians to cyclists, the main difference from bicycles is that vehicles go even faster but with a even more predictable path. In the chosen frame is possible to see one of the biggest and unresolved problem of the system that are nets. In this case, the obstacle detection system con partially see the net because we are almost tangent to it, but when facing it straight the net is almost invisible. This is a big problem that has be investigated in the future works. Apart from the higher velocity, the tests with vehicles are working good because the vehicles don't change direction of movement as fast as pedestrians and bicycles so the planning part is a little bit easier.

As mentioned before one of the main constrains of the planning system is to be computationally very efficient. In table 3.4 is shown the computational time of the developed multy goal A*. Obviously the time depends on various factors:

- The distance from the target: the further the target is, the more time will be required for the computation.

Figure 3.28: Example of result of Target Tracking Algorithm. In this case the drone is following a vehicle on an untarmacked road.

- The complexity of the environment: the more complex the environment is, the more curvy the solution will be and the more time will be required for the computation.
- The imprecision factor: the higher the imprecision factor is, the less time will be required for the computation at the cost of having a trajectory that will be at max one plus the imprecision factor longer than the optimal trajectory.

In the table is possible to see the computational time calculated on a desktop PC that has almost the same computational power of the board on the drone. We also compared this results with the computational time calculated on an low power ARM CPU that has fewer computational power. In both cases there is a configuration that allows the path planner to run in real time at at least 20 Hz.

In our experimentation, usually, the drone is pretty near to the circumference. We used a imprecision factor of 0.2.

For further tests on the ARM CPU we will use a imprecision factor of 0.5.

In tables 3.5 and 3.6 are shown the computational time of Ray Casting algorithm and BSpline smoothing algorithm. Compared to A* search time, this two computational times are much shorter. They depend only on the length of the path, the longer

Table 3.4: Multy-goal A* computational time table. Times are represented in milliseconds. First column shows the path length and corresponding test number as shown in figure 3.29. Other columns contains the computational times divided by imprecision factor. The imprecision factor is the value used to weight the heuristic. The weighted heuristic is equal to the admissible heuristic multiplied by one plus the imprecision factor. For each value of the imprecision factor are shown computational time on a desktop PC with CPU Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, and on a single core of a low power ARM processor.

Path length	impr. = 0.0		impr. = 0.1		impr. = 0.2		impr. = 0.5	
10.3 m (1)	2.5	15	0.75	4.6	0.45	3.1	0.45	3.1
19.8 m (2)	44.5	305	2	14.1	0.9	6.2	0.9	6.2
26.8 m (3)	142.5	1020	3.5	24.8	2.2	14.2	2.1	10.5
41.7 m (4)	665	4550	98.5	730	80.3	584	65.2	465

the slower the two algorithms are. As before, for each case both computational time on desktop PC CPU and on ARM CPU are shown.

In table 3.7 are shown the maximum memory required during the A* search. The memory and the number of nodes grow together with the length of the path and with the complexity of the environment because if the algorithm has to try more routes to reach the goal, it has to maintain memory of multiple possibilities during exploration.

3.7 Conclusion And Future Works

The developed system described in this chapter has proven to be able to generate behaviors that are very smooth. Through experiments on real application, it has proven to satisfy the safety measures that we impose to enable the use of the system to inexpert users. The attention paid in the creation of comfortable behaviors, without brusque movements, have the user feel trust toward the system.

The whole system architecture has proven to be flexible and adequate to the application developed. The architecture has high modularity, as each system that com-

Table 3.5: Ray Casting smoothing time table. Times are represented in milliseconds. First column shows the path length and corresponding test number as shown in figure 3.29. Second column contain the computational times on a desktop PC with CPU Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, and third column on a single core of a low power ARM processor.

Path length	Desktop PC CPU	ARM processor
10.3m (1)	0.02ms	0.10ms
19.8m (2)	0.09ms	0.66ms
26.8m (3)	0.20ms	1.50ms
41.7m (4)	0.45ms	3.90ms

Table 3.6: BSpline smoothing time table. Times are represented in milliseconds. First column shows the path length and corresponding test number as shown in figure 3.29. Second column contain the computational times on a desktop PC with CPU Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, and third column on a single core of a low power ARM processor.

Path length	Desktop PC CPU	ARM processor
10.3m (1)	0.17ms	1.47ms
19.8m (2)	0.22ms	2.50ms
26.8m (3)	0.31ms	3.61ms
41.7m (4)	0.47ms	5.45ms

Table 3.7: Multy-goal A* maximum memory usage table. First column shows the path length and corresponding test number as shown in figure 3.29. Second column contain the maximum memory occupied during computation, and third column the max number of nodes in the open set.

Path length	max memory	max number of nodes
10.3 m (1)	3768 Byte	314 nodes
19.8 m (2)	8k2 Byte	684 nodes
26.8 m (3)	16k5 Byte	1377 nodes
41.7 m (4)	390k Byte	32488 nodes

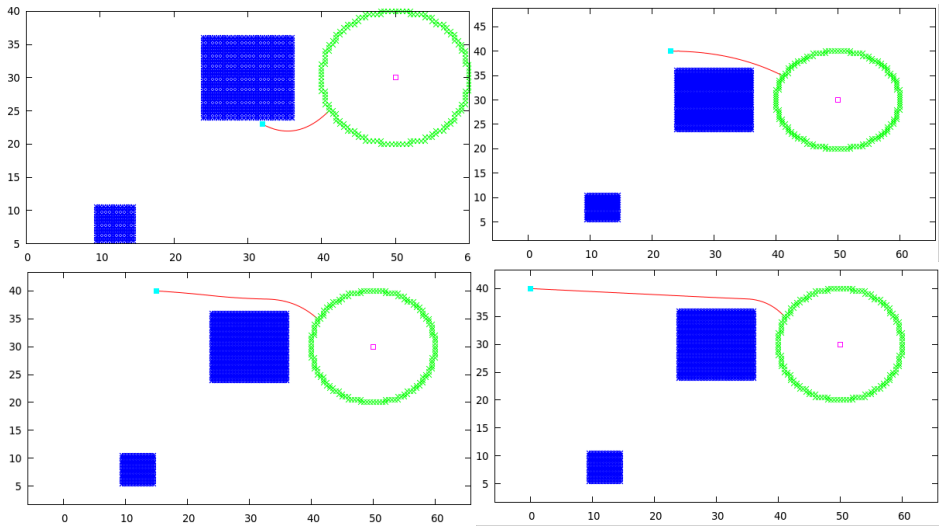


Figure 3.29: Test fields used for compute computational performances. From left to right and from up to down the are named (1)(2)(3)(4). The obstacles are shown in blue, the target is the empty square, the goal circumference is shown in green, the drone is the cyan square and the trajectory is the red line.

pose it are independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality. For this reason developing new applications and behaviors is simple and of fast deployment.

More improvement will be done in the obstacle detection system, focusing on improve the performance in the detection of "difficult" obstacles. These kind of obstacle are small branches and object that can be seen through, like nets. To improve the detection, the perception system, the 3DV-E, will be substitute with a better one. The 3DV-E has relatively low resolution so we plan to mount a 4k stereo system that look forward to improve the precision of the detection. Another improvement, is to add sonar sensors to add redundancy to the detection. The sonar system work good with nets and will give additional info that will be fused with stereo images.

A new application that will be developed for this flying platform is autonomous landing. The goal of this feature is to recognize a landing point and land on it. This landing point can be a photo of the place from which we took off. For example, the drone take off, do some video of the location around it and the user can call back the drone and it land in the same starting point. Another possibility is to make the drone on some kind of target board, like an ArUco marker board as shown in figures 3.30 and .

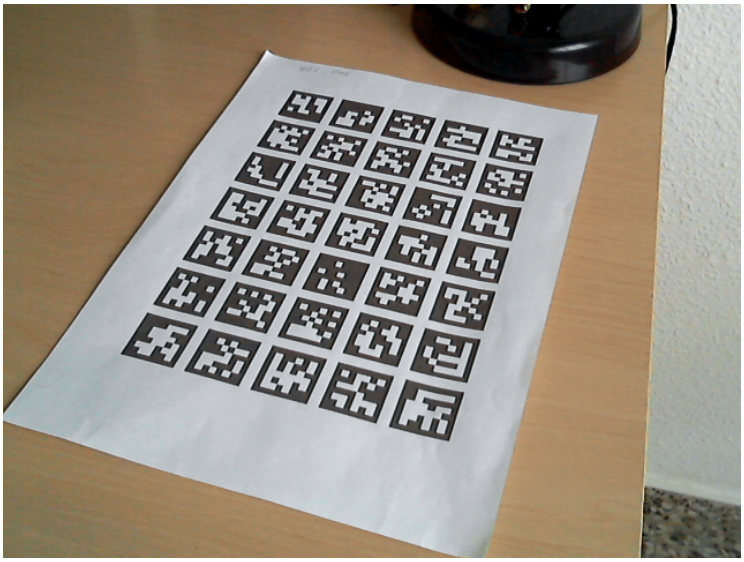


Figure 3.30: Example of ArUco marker board.

Figure 3.31: Example of drone landing on a mobile board. Reference to [26].

An example of the use of this application are numerous. One of them is to ship packages for online shopping site. The costumer can order a something online and give the shopping site a GPS position. The shopping site will send a unique landing marker to print and to be put outside where the drone can land. The drone will be able to reach the GPS position given, find the marker, land, release the package and finally go home. Another example of application for the autonomous landing on marker is to put a marker on the roof of a car. When the car will reach a parking lot, the drone can take off to find free parking slots, and communicate the free positions to the car to reach them. It will then come back to the car and land on top of it. As reference for the algorithm that will be implemented we used the survey written by Jin et al. [26].

Bibliography

[1] M. Athans and P. L. Falb. *Optimal control: an introduction to the theory and its applications*. Courier Dover Publications, 2006.

[2] D. J. Balkcom and M. T. Mason. Extremal trajectories for bounded velocity mobile robots. volume 2, pages 1747–1752 vol.2, 2002.

[3] A. Broggi, S. Debattisti, P. Grisleri, and M. Panciroli. The deeva autonomous vehicle platform. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 692–699, June 2015.

[4] M. W. Chen and A. M. S. Zalzala. Dynamic modelling and genetic-based trajectory generation for non-holonomic mobile manipulators. *Control Engineering Practice*, 5(1):39–48, 1997.

[5] Y. Chen and A. Desrochers. Structure of minimum-time control law for robotic manipulators with constrained paths. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 971–976 vol.2, May 1989.

[6] J. Connors and G. Elkaim. Analysis of a spline based, obstacle avoiding path planning algorithm. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 2565–2569. IEEE, 2007.

[7] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Pittsburgh, PA, January 1992.

[8] H. Delingette, M. Hebert, and K. Ikeuchi. Trajectory generation with curvature constraint based on energy minimization. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 206–211. IEEE, 1991.

[9] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[10] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, pages 497–516, 1957.

[11] J. F. Epperson. On the runge example. *Amer. Math. Monthly*, 94(4):329–341, 1987.

[12] G. Erinc, G. Erinc, and S. Carpin. A genetic algorithm for nonholonomic motion planning. pages 1843–1849. IEEE, 2007.

[13] T. Fraichard. Smooth trajectory planning for a car in a structured world. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 318–323. IEEE, 1991.

[14] T. Fraichard and A. Scheuer. From reeds and shepp's to continuous-curvature paths. *Robotics, IEEE Transactions on*, 20(6):1025–1035, Dec 2004.

[15] S. Francis, S. Anavatti, and M. Garratt. Dynamic model of autonomous ground vehicle for the path planning module. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 73–77, Dec 2011.

[16] S. Francis, S. Anavatti, and M. Garratt. Online incremental and heuristic path planning for autonomous ground vehicle. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pages 233–239, Nov 2011.

[17] Y.-M. Han, J.-B. Jeong, and J.-H. Kim. Quadtree based path planning for unmanned ground vehicle in unknown environments. In *Control, Automation and*

Systems (ICCAS), 2012 12th International Conference on, pages 992–997, Oct 2012.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[19] H. He, Y. Li, J. Song, W. Wang, and X. Xu. Triple rrts: An effective method for path planning in narrow passages. *Advanced Robotics*, 24(7):943–962, 2010.

[20] V.-D. Hoang, D. Hernandez, J. Hariyono, and K.-H. Jo. Global path planning for unmanned ground vehicle based on road map images. In *Human System Interactions (HSI), 2014 7th International Conference on*, pages 82–87, June 2014.

[21] U.-Y. Huh and S.-R. Chang. A g2 continuous path-smoothing algorithm using modified quadratic polynomial interpolation. *International Journal of Advanced Robotic Systems*, 11(1), 2014.

[22] J.-H. Hwang, R. C. Arkin, and D.-S. Kwon. Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1444–1449. IEEE, 2003.

[23] Y. Hwang and N. Ahuja. A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on*, 8(1):23–32, Feb 1992.

[24] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Nonholonomic Motion Planning*, pages 271–342. Springer, 1993.

[25] K. Jiang, L. D. Seneviratne, and S. W. E. Earles. Time-optimal smooth-path motion planning for a mobile robot with kinematic constraints. *Robotica*, 15(5):547–553, 1997.

[26] S. Jin, J. Zhang, L. Shen, and T. Li. On-board vision autonomous landing techniques for quadrotor: A survey. In *2016 35th Chinese Control Conference (CCC)*, pages 10284–10289, July 2016.

[27] S.-W. Jo, S.-M. Park, and J.-H. Kim. Unmanned ground vehicle for driving based global path is lateral avoidance path planning. In *Control, Automation and Systems (ICCAS), 2014 14th International Conference on*, pages 1648–1651, Oct 2014.

[28] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. pages 1265–1270 vol.3, 1989.

[29] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. pages 5041–5047. IEEE, 2013.

[30] C.-K. Kim, Y.-M. Han, B.-H. Bae, and J.-H. Kim. The research of path planning algorithm considering vehicle's turning radius for unmanned ground vehicle. In *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*, pages 754–756, Oct 2011.

[31] T. Kito, J. Ota, R. Katsuki, T. Mizuta, T. Arai, T. Ueyama, and T. Nishiyama. Smooth path planning by using visibility graph-like method. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3770–3775. IEEE, 2003.

[32] K. Komoriya and K. Tanie. Trajectory design and control of a wheel-type mobile robot using b-spline curve. In *Intelligent Robots and Systems '89. The Autonomous Mobile Robots and Its Applications. IROS '89. Proceedings., IEEE/RSJ International Workshop on*, pages 398–405, Sep 1989.

[33] P. Konkimalla and S. M. Lavalle. Efficient computation of optimal navigation functions for nonholonomic planning. In *In Proc. First IEEE Workshop on Robot Motion and Control*, pages 187–192, 1999.

[34] M. H. Korayem, M. Nazemizadeh, and H. R. Nohooji. Optimal point-to-point motion planning of non-holonomic mobile robots in the presence of multiple obstacles. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 36(1):221–232, 2014.

[35] F. Lamiraux and J.-P. Lammond. Smooth motion planning for car-like vehicles. *Robotics and Automation, IEEE Transactions on*, 17(4):498–501, Aug 2001.

[36] S. M. LaValle. Rapidly-exploring random trees a new tool for path planning. 1998.

[37] S. M. LaValle. From dynamic programming to rrts: Algorithmic design of feasible trajectories. In *Control Problems in Robotics*, pages 19–37. Springer, 2003.

[38] J. Li, S. Liu, B. Zhang, and X. Zhao. Rrt-a motion planning algorithm for non-holonomic mobile robot. pages 1833–1838. SICE, 2014.

[39] Y. Linquan, L. Zhongwen, T. Zhonghua, and L. Weixian. Path planning algorithm for mobile robot obstacle avoidance adopting bezier curve based on genetic algorithm. pages 3286–3289. IEEE, 2008.

[40] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[41] W. Nelson. Continuous-curvature paths for autonomous vehicles. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 1260–1264 vol.3, May 1989.

[42] K. Pereida and J. Guivant. Hybrid dijkstra-pso algorithm for motion planning of non-holonomic multiple-trailer platforms in dense contexts. In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pages 13–18, July 2013.

[43] A. Piazzi, C. Bianco, and M. Romano. η^3 -splines for the smooth path generation of wheeled mobile robots. *Robotics, IEEE Transactions on*, 23(5):1089–1095, Oct 2007.

[44] A. Piazzi and C. G. L. Bianco. Quintic g 2-splines for trajectory planning of autonomous vehicles. In *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, pages 198–203. IEEE, 2000.

[45] A. Piazzi, C. G. Lo Bianco, M. Bertozzi, A. Fascioli, and A. Broggi. Quintic g2-splines for the iterative steering of vision-based autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 3(1):27–36, 2002.

[46] J. REEDS and L. SHEPP. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[47] A. Reina, L. M. Gambardella, M. Dorigo, and G. A. Di Caro. zeppelin: Distributed path planning using an overhead camera network. *Int J Adv Robot Syst*, 11:119, 2014.

[48] A. Scheuer and T. Fraichard. Collision-free and continuous-curvature path planning for car-like robots. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pages 867–873 vol.1, Apr 1997.

[49] T. Simeon, S. Leroy, and J. P. Laumond. Computing good holonomic collision-free paths to steer nonholonomic mobile robots. volume 2, pages 1004–1010 vol.2, 1997.

[50] A. Stentz. The focussed dˆ* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[51] A. Stentz. Optimal and efficient path planning for partially known environments. In M. Hebert, C. Thorpe, and A. Stentz, editors, *Intelligent Unmanned Ground Vehicles*, volume 388 of *The Springer International Series in Engineering and Computer Science*, pages 203–220. Springer US, 1997.

[52] J. K. Suhr and H. G. Jung. Fully-automatic recognition of various parking slot markings in around view monitor (avm) image sequences. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 1294–1299, Sept 2012.

[53] L. E. K. P. Svestka and J.-C. L. M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces.

[54] A. Takahashi, T. Hongo, Y. Ninomiya, and G. Sugimoto. Local path planning and motion control for agv in positioning. In *Intelligent Robots and Systems '89. The Autonomous Mobile Robots and Its Applications. IROS '89. Proceedings., IEEE/RSJ International Workshop on*, pages 392–397, Sep 1989.

[55] A. S. Thierry Fraichard. From reeds and shepp's to continuous-curvature paths. pages 1025–1035. IEEE, 2004.

[56] Y. Wang, P. Chen, and Y. Jin. Trajectory planning for an unmanned ground vehicle group using augmented particle swarm optimization in a dynamic environment. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 4341–4346, Oct 2009.

[57] C. W. Warren. Fast path planning using modified a* method. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 662–667. IEEE, 1993.

[58] L. Weng and D. Song. Path planning and path tracking control of unmanned ground vehicles (ugvs). In *System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on*, pages 262–266, March 2005.

[59] H. J. Woo, H. J. Woo, S. B. Park, S. B. Park, J. H. Kim, and J. H. Kim. Research of the optimal path planning methods for unmanned ground vehicle in darpa urban challenge. pages 586–589. IEEE, 2008.

[60] J. wung Choi, R. Curry, and G. Elkaim. Path planning based on bezier curve for autonomous ground vehicles. In *World Congress on Engineering and Computer*

Science 2008, WCECS '08. Advances in Electrical and Electronics Engineering - IAENG Special Edition of the, pages 158–166, Oct 2008.

[61] M. Yamamoto, M. Iwamura, and A. Mohri. Quasi-time-optimal motion planning of mobile platforms in the presence of obstacles. volume 4, pages 2958–2963 vol.4, 1999.

[62] K. Yang, S. Moon, S. Yoo, J. Kang, N. Doh, H. Kim, and S. Joo. Spline-based rrt path planner for non-holonomic robots. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 73(1-4):763–782, 2014.

[63] K. Yang and S. Sukkarieh. An analytical continuous-curvature path-smoothing algorithm. *Robotics, IEEE Transactions on*, 26(3):561–568, June 2010.

[64] J. Yoon and C. Crane. Path planning for unmanned ground vehicle in urban parking area. In *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*, pages 887–892, Oct 2011.

[65] H. Yu, K. Meier, M. Argyle, and R. Beard. Cooperative path planning for target tracking in urban environments using unmanned air and ground vehicles. *Mechatronics, IEEE/ASME Transactions on*, 20(2):541–552, April 2015.

Thanks

I want to thank my family for all the support that gave to me during the long years of this path. I thank my brother Marco that thanks to him i continued to push forward to be an example.

My thanks go also the people of the Vislab, in particular professors Alberto Broggi and Massimo Bertozzi.

A special mention to my great friend Luca Castangia that walk this path together with me since a long time.