Decentralized trajectory optimization for a fleet of industrial mobile robots

Inês Sofia Baptista Silva
ines.sofia.silva@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

October 2022

Abstract

Multi-robot trajectory planning is a complex task given the need to avoid collisions with both static and dynamic obstacles, such as other moving robots. This paper presents a novel approach to efficiently generate collision-free optimal trajectories for multiple mobile robots, with time-varying obstacle constraints. The proposed method first uses a sampling-based kinodynamic trajectory planner to obtain an initial solution. Then, a non-linear optimization is applied to refine this solution into a smoother trajectory. In classic approaches that solve multi-agent trajectory optimization problems, the computational time often grows with the number of agents. As the main contribution of this work, the proposed method interpolates a set of signed distance fields to obtain the signed distance from any pose of the robot to the nearest static or dynamic obstacle. Thus, the formulated optimization problem provides to the solver a single constraint regarding collision avoidance. Therefore, this method presents to be more efficient. The algorithm was tested for a fleet of vehicles with non-linear constraints in obstacle-populated scenarios.

Keywords: Path Planning for Multiple Mobile Robots or Agents, Collision Avoidance, Optimization and Optimal Control, Multi-Robot Systems, Kinematics

1. Introduction

Many multi-robot systems require the inter-coordination of mobile agents when navigating complex environments. For instance, these systems can have military, civil or even industrial applications in areas such as surveillance, search and rescue and transportation [1–4]. These scenarios require computing collision-free trajectories from an initial to a final position for every autonomous mobile robot in the fleet.

Motion planning in multi-robot systems can be rather challenging since each robot must avoid both static and moving obstacles, in which the latter might be other vehicles in the fleet. Additionally, this becomes even more demanding when differential constraints are imposed by the physical dynamics and limited control inputs are considered. Also, it can be difficult to efficiently compute optimal collision-free routes, where the trajectory quality is determined by a cost function that, for example, minimizes the control effort or the trajectory duration. This paper presents an efficient formulation for the optimization problem that handles static and dynamic obstacles in the environment.

2. Background

When dealing with motion planning problems, it is necessary to distinguish between two designations: path and trajectory [5]. Informally, a path is a spatial construct with no time constraint. A trajectory, on the other hand, provides a time constraint, by assigning a time law to a geometric path.

There are many different path planning algorithms proposed. Decomposition graph-based methods [6] convert the motion planning problem into a graph search one, by transforming the configuration space into a finite set of regions called cells. However, these methods show high complexity in high-dimensional spaces. Additionally, sampling-based techniques, such as Probabilistic Roadmaps (PRM) [7] and Rapidly-exploring Random Tree (RRT) [8], incrementally search for a solution in the environment. These algorithms are very effective and they guarantee probabilistic completeness. However, due to their random search property, the solution obtained is not optimal. Finally, the artificial potential fields approach [9] considers the robot as a particle under the influence of an artificial potential field. This method can generate a locally optimal trajectory. However, this approach is not complete, since it is not guaranteed that a solution will be found even if it exists.

Additionally, it is not only important to consider global constraints, which are imposed by obstacles. In fact, robots often have local constraints, which can be considered as limits in velocities or accelerations at every point due to kinematic and dynamic restrictions. According to [10], this kind of problem may be more suitably formulated in the optimal control framework, in which the solution is a trajectory encoded as a sequence of states and controls that optimizes a given objective function.

According to [11], the techniques for solving optimal control problems can be divided into two categories: indirect and direct methods. An indirect method can achieve more accurate solutions, at the expense of being more difficult to construct and solve. On the other hand, a direct method discretizes the trajectory optimization problem and converts the original problem into nonlinear programming one [12]. The direct shooting and direct collocation techniques are included in the direct method family. Direct shooting methods are suitable for applications with simple control and few path constraints. Direct collocation methods are more appropriate for applications where the dynamics and control must be computed accurately and the structure of the control trajectory is not known a priori.

Many methods have been proposed for multi-robot trajectory planning. They can be divided into centralized, decentralized, and decoupled approaches. In [13, 14], centralized techniques were used, where the optimal trajectory generation problem was formulated as mixed integer quadratic programming (MIQP) or sequential convex programming (SQP) problems. The solutions generated with centralized methods are complete and optimal. Nevertheless, they are not scalable or robust.

Additionally, motion planning in multi-robot systems can be solved using decoupled techniques, such as sequential planning methods [15–17]. In sequential planning, the trajectory of each robot is decoupled and it is generated by avoiding the previously planned ones. This method is relatively fast, however, it cannot find a globally optimal solution.

3. Contributions

Based on the above considerations, this paper proposes a multi-agent motion planning approach that generates safe, dynamically feasible and locally optimal trajectories for multiple mobile robots with nonlinear constraints. The proposed method will follow a sequential planning approach to deal with asynchronous tasks. Note that the generated trajectories will not guarantee a globally optimal solution. In fact, a new trajectory is assigned with the goal of avoiding the already planned ones.

In classic multi-robot trajectory optimization approaches, there is a restriction for each pair of robots that limits the distance between them. Thus, the computational complexity scales with the number of agents. The novelty of this method is that the optimization problem of each trajectory is formulated having only one constraint regarding obstacle avoidance, which includes both static and dynamic obstacles. This approach makes a tradeoff between computational effort and allocated memory. To build this single constraint regarding obstacle avoidance is necessary to precompute and store a set of signed distance fields, that maps the signed distance from a generic point to the nearest obstacle. Nevertheless, with this implementation, the number of restrictions does not increase with the number of robots and, consequently, it is possible to reduce the problem complexity.

4. Problem Formulation

Consider a multi-robot system with N robots, $\mathcal{R} = \{R^1, ..., R^N\}$, that move in an 2-D environment, $\mathcal{W} \in R^2$ populated by a set of M obstacles $\mathcal{O} = \{O^1, ..., O^M\}$. The set of all possible states of agent $i = \{1, ..., N\}$ are defined by the the state space $\mathcal{X}^i$, while the set of the allowable control input space is represented by $\mathcal{U}^i$.

The set of states that satisfies the global constraints are defined as $\mathcal{X}_{free}$. The dynamic constraints, that should be considered in this planning problem, are expressed in a differential state model, $\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t))$, defined for every state $\mathbf{x} \in \mathcal{X}$ and input $\mathbf{u} \in \mathcal{U}$.

For each robot R^i, a task s is assigned at time t^i_{init}. Every task should guarantee that the respective robot generates a feasible trajectory from its initial state, $\mathbf{x}_{init} \in \mathcal{X}^i$, to a goal state, $\mathbf{x}_{goal} \in \mathcal{X}^i$. Hence, the trajectory of robot R^i can be written as $\pi^i : [t^i_{init}, t^i_{goal}] \to \mathcal{X}_{free}$. Each trajectory is defined by: its duration, T^i; the control input along the trajectory, $\mathbf{u} : [t^i_{init}, t^i_{goal}] \to \mathcal{U}^i$; and the corresponding states, $\mathbf{x} : [t^i_{init}, t^i_{goal}] \to \mathcal{X}^i$.

In a multi-robot system, given the workspace $\mathcal{W}$ and the tasks $\mathcal{S} = \{s^1, ..., s^K\}$ for the robot set $\mathcal{R}$, it is necessary to find a set of trajectories $\mathcal{P} = \{\pi^1, ..., \pi^K\}$ that execute the K tasks. Note that π^i, π^j of every two different robots R^i, R^j must be conflict free. The tasks are assigned sequentially, i.e, the trajectory π^i is planned by avoiding the already defined ones: $\pi^1, ..., \pi^{i-1}$.

5. Methodology

In order to generate each robot trajectory, firstly, a sampling-based method is used to create an initial trajectory that satisfies the task assigned to a robot R^i. Then, the trajectory is refined into a smoother and shorter one by solving a trajectory optimization problem. In both stages, the non-linear motion model of the robots is directly considered to guarantee the feasibility of the trajectories.

5.1. Trajectory Planning

To solve the planning problem and to generate an initial solution, an extension of the RRT algorithm, the Kinodynamic RRT by La Valle [18], is implemented. Both global and local constraints are going to be considered when searching for a feasible trajectory.

Finally, to show the random property of the Kinodynamic RRT, the algorithm was run fifty times and, as expected, fifty different trajectories were generated for the same motion planning problem. Figure 1 reports the trajectories obtained. The inherent randomness of the Kinodynamic RRT algorithm reflects on the low-quality path. In fact, the solutions provided by this method are not smooth and may contain unnecessary maneuvers to achieve the goal region. This technique is good at delivering an initial solution, but it must be complemented with more sophisticated methods to return higher quality solutions.

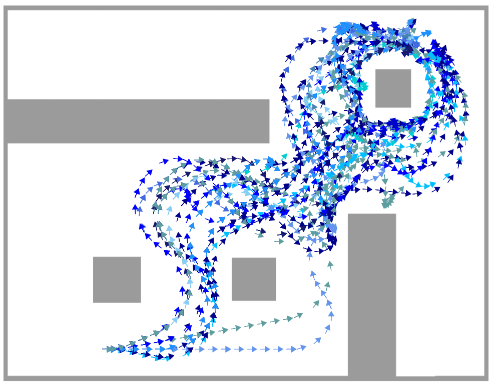


Figure 1: Example of fifty different trajectories generated with the Kinodynamic RRT for the same motion problem.

5.2. Trajectory Optimization

Due to the possible non-linearity of the system dynamics, the trapezoidal collocation method was implemented [10]. By using trapezoidal quadrature, the continuous problem can be converted into a discrete approximation. This is done by representing the continuous state, control and system dynamics by their values at specific points in time, known as collocation points.

5.2.1 Dynamic Function

In direct collocation methods, the differential state model can be considered as a constraint on the dynamics of the vehicle. With trapezoidal quadrature, the first-order differential constraints can be converted as equality constraints:

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \frac{1}{2} h \left(f_k + f_{k+1} \right) \tag{1}$$

where $h = t_{k+1} - t_k$ with $k \in \{0, ..., C-1\}$ and C is the number of collocation points. This approximation is then applied between every pair of collocation points, where the discrete time nodes along the trajectory are defined as $t_k = kh$. Consequently, $\mathbf{x}_k = \mathbf{x}(t_k)$ is defined as the state at point k, $\mathbf{u}_k = \mathbf{u}(t_k)$ the control at point k, and $\mathbf{f}_k = \mathbf{f}(t_k, \mathbf{x}_k, \mathbf{u}_k)$ the systems dynamics at point k.

5.2.2 State and Control bounds

The physical limits of the system must be taken into consideration when defining the optimization problem. These constraints are applied throughout the entire trajectory to the state and control variables. They are given by $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$ and $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$, with $k \in \{0, ..., C-1\}$.

5.2.3 Boundary Constraints

Boundary constraints satisfy the requirements of the initial and final states. They are given by some function $g(t_{init}, t_{goal}, \mathbf{x}_0, \mathbf{x}_{C-1}) \leq 0$.

5.2.4 Obstacle Avoidance Constraint

Let the cartesian coordinates of a robot R be denoted as $\mathbf{p}(t)$, for all $t \in [t_{init}, t_{goal}]$. A trajectory is considered to be collision-free if the distance from every vehicle position along the trajectory to the nearest obstacle is greater than a safe margin, $d_{safe} > 0$.

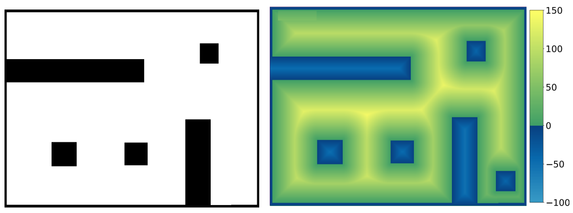
When considering static obstacles, it is advantageous to simply precompute a signed distance field (SDF) which stores the signed distance from any point $\mathbf{p}(t)$ to the boundary of the nearest obstacle. Values for the signed distance are negative inside obstacles, positive outside and zero at the boundary [19]. To create the SDF based on a binary occupancy grid map that represents the environment it is necessary to compute the Euclidean Distance Transform (EDT) for both the grid map and its logical complement. The SDF is then given by the difference between the two EDT values.

When generating an optimized trajectory it is necessary to ensure that it does not collide with any of the static or dynamic obstacles in the environment. The novelty of this method lies in this obstacle avoidance constraint. In the following results the dynamic obstacles are considered as other moving robots. However, any dynamic object, such as people, can be avoided using with this method, if provided with a predictive movement model.

The distance between every position $\mathbf{p}^i(t)$ of agent R^i and all static obstacles and moving robots must respects the safe margin:

$$sd(\mathbf{p}^i(t), \mathbf{p}^j(t), O^m) \geq d_{safe}, \quad \forall i, j \in \{1, ..., N\}, \forall m \in \{1, ..., M\}. \tag{2}$$

Let $sdf(t)$ be a function that represents the SDF of the environment at a time instant t, where static obstacles and positions of vehicles with already defined trajectories are considered. In Fig 2a is illustrated an example of an occupancy grid map, where black represents the obstacles and white are unoccupied areas. Fig. 2b presents the SDF computed from the known map, where the darker blue represents the obstacle boundaries, light blue the area inside the objects, and green and yellow the areas outside the obstacles (yellow being the furthest from an obstacle). Additionally, at the right bottom corner is represented a dynamic square object.



(a) Occupancy gridmap (b) SDF

Figure 2: Example of occupancy gridmap (a) and SDF with dynamic obstacle (right bottom corner) (b).

When formulating the trajectory optimization problem of R^i, a set of $sdf(t)$ is generated with a defined time interval, $t_q \in \{t_{init}, ..., t_{final}\}$, where $t_{final} = max(t^i_{goal}), \forall i \in \{1, ..., N\}$. Let us define a tuple that attaches the time step to the respective generated $sdf(t_q)$, $\{(t_q, sdf(t_q))\}, \forall q$. The set of all generated tuples is given by $\mathcal{SD} = \bigcup_{t_q \in \{t_{init},...,t_{final}\}} \{(t_q, sdf(t_q)\}$.

By interpolating $\mathcal{SD}$, it is possible to obtain a function that provides as an output the signed distance from R^i to the nearest obstacle or vehicle, given a time t and the respective robot pose $\mathbf{p}^i(t)$. This function can be defined as $d(t, \mathbf{p}^i(t))$.

5.2.5 Cost Function

Without loss of generality, the cost function was chosen to minimize the total duration of the vehicle's trajectory. By using direct collocation, this can be achieved by minimizing the time step, i.e, $J(h, \mathbf{x}_k, \mathbf{u}_k) = h$. Notwithstanding, it could also be applied a cost function regarding, for example, the control effort.

5.2.6 Proposed Formulation

Finally, the overall trajectory optimization problem of robot R^i can be defined as follow:

$$\begin{aligned}
\text{Minimize} \quad & J(h, \mathbf{x}_k, \mathbf{u}_k) \\
\text{w.r.t.} \quad & h, \mathbf{x}_k, \mathbf{u}_k, \quad \forall k \in \{0, ...C-1\} \\
\text{subject to} \quad & \mathbf{x}_{k+1} - \mathbf{x}_k = \frac{1}{2} h(f_k + f_{k+1}), \forall k && (3) \\
& d(hk, \mathbf{p}^i(hk)) \geq d_{safe}, \quad \forall k && (4) \\
& g(t_{init}, t_{goal}, \mathbf{x}_0, \mathbf{x}_{C-1}) \leq 0 && (5) \\
& \mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}, \quad \forall k && (6) \\
& \mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}, \quad \forall k && (7)
\end{aligned}$$

5.2.7 Initial Guess

It is critical to have a good initial guess when solving a trajectory optimization problem, as this allows the solver to quickly arrive at the optimal solution. In this paper, the trajectory generated by the Kinodynamic RRT, as specified in Section 5.1, serves as the initial guess.

6. Results

This implementation was experimented on a fleet of differential drive robots. However, this method can be applied to any vehicle with dynamic constraints expressed as a differential state, as defined in Section 4.

6.1. Non-linear Differential Drive Model

The state vector of a differential drive agent with two wheels can be described by $\mathbf{x} = [x_r, y_r, \theta_r, t, v_R, v_L]^T \in \mathcal{X}$, where x_r and y_r are the cartesian coordinates of the robot, θ is the orientation of the robot with respect to the x-axis, t denotes the time when each state is reached and, finally, v_R and v_L are the linear velocities of the right and left wheels, respectively. Additionally, the control vector can be defined as $\mathbf{u} = [v, w, a_R, a_L]^T$, where v is the vehicle linear velocity, ω is the angular velocity and a_R and a_L represent the linear acceleration of the right and left wheels, respectively.

The kinematic model of the system is given by the following nonlinear equations of motion:

$$\begin{gathered}
\dot{x}_r = v \cos\theta; \quad \dot{y}_r = v \sin\theta; \quad \dot{\theta}_r = \omega; \\
\dot{t} = 1; \quad \dot{v}_R = a_R; \quad \dot{v}_L = a_L.
\end{gathered} \tag{8}$$

The vehicle has a constant acceleration between two consecutive collocation points. The velocities and accelerations of both wheels are limited to physically admissible values at every time step, i.e, $|v_{Rk}| \leq v_{max} \wedge |v_{Lk}| \leq v_{max}$ and $|a_{Rk}| \leq a_{max} \wedge |a_{Lk}| \leq a_{max}$, with $k \in \{0, ..., C-1\}$. Additionally, the linear velocity of the vehicle cannot be a negative value, since it cannot perform a reverse manoeuvre, $v_k \geq 0, k \in \{0, ..., C-1\}$.

Furthermore, the angular variation of the vehicle every collocation point, should insure that the robot only moves within the angular range of the sensor, β, i.e, $|\omega_k| \geq \frac{\beta/2}{h_k}, k \in \{0, ..., C-1\}$.

Boundary constraints, for the specific case where the vehicle starts and finishes an assigned task at rest, can be given by $\mathbf{x}_0 = \left[x_{rinit}, y_{rinit}, \theta_{rinit}, 0, 0\right]^T$ and $\mathbf{x}_{N-1} = \left[x_{rgoal}, y_{rgoal}, \theta_{rgoal}, 0, 0\right]^T$.

6.2. Optimization Solver Integration

The algorithm was formulated using Python's symbolic framework CasADi [20] and was solved with the open-source and highly efficient Interior Point Optimizer (IPOPT) solver [21] . This solver finds local optimum solutions of large-scale non-linear optimization problems using an interior-point method.

6.3. Multi-Robot Problem

For a set of six vehicles $\mathcal{R} = \{R^1, R^2, R^3, R^4, R^5, R^6\}$, asynchronous tasks are assigned at time instants $t^{1,2}_{init} = 0s$, $t^{3,4}_{init} = 5s$ and $t^{5,6}_{init} = 10s$ for the pairs of robots $\{R^1, R^2\}$, $\{R^3, R^4\}$ and $\{R^5, R^6\}$, respectively. The trajectories are generated sequentially, i.e, the first and last trajectories to be calculated are π^1 and π^6, respectively. To each vehicle i was assigned an initial pose $\mathbf{x}^i_{init}$ and a goal region $\mathbf{X}^i_{goal}$.

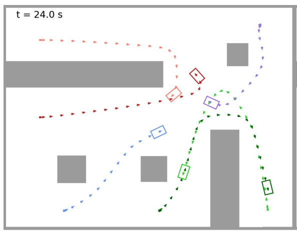
Since the trajectory of R^1 is the first one to be generated it does not depend on any other vehicle. Therefore, it is treated as a single-agent problem. When generating trajectory π^2, the position of R^1 through time is described by a set of SDFs. This set is interpolated and used in the optimization problem to reveal information regarding the signed distance from R^2 to the nearest obstacle or vehicle. Similarly to π^2, the trajectory of R^3 is calculated avoiding the already existing ones, π^1 and π^2. The remain trajectories were generated following the same line of thought as the previous ones.

Finally, some strategic frames that validate the implementation are depicted in Figure 3. It is possible to verify that all vehicles swerve to avoid collisions.

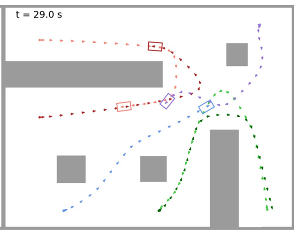
6.4. Alternative Architectures

Typically, Trajectory Planning (TP), with the Kinodynamic RRT presented in Section 5.1, does not take moving obstacles into account, i.e., when the tree expands, the other agent positions are ignored.

Figure 4a illustrates the architecture followed to obtained each previously presented trajectory. In this approach, only the Trajectory Optimization (TO) module considers the other vehicles trajectories and, consequently, is the only responsible for the moving obstacles avoidance. The TP takes as inputs the static obstacles of the environment and



(a) $t = 24s$ R_3 and R_4 avoid colliding with each other.

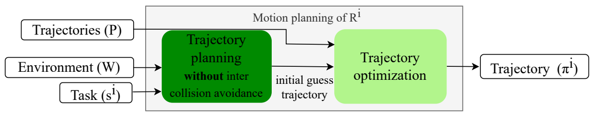


(b) $t = 29s$ R_5 and R_6 avoid colliding with each other.

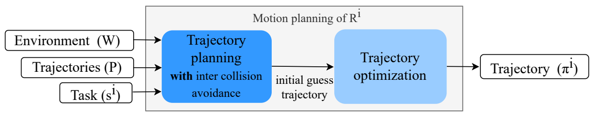
Figure 3: Trajectories of 6 vehicles with inter vehicle avoidance, at different time instants.

the task to be assigned to vehicle R^i, i.e, the initial and goal poses to plan the route.

However, the Kinodynamic RRT collision avoidance module can be modified to take moving obstacles into account as well. Essentially, before adding a new node to the tree, the distance from that node to the nearest obstacle, static or not, can be determined using $d(t, \mathbf{p}^i(t))$, where t and $\mathbf{p}(t)$ can be the time instant and cartesian coordinates which describe the vehicle at that specific node. With this approach, both the TP and TO modules have information regarding other vehicles trajectories. Figure 4b represents the architecture of approach B.



(a) Approach A



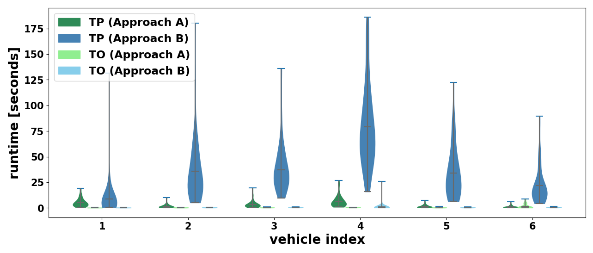
(b) Approach B

Figure 4: Architecture of approaches A and B to solve the motion planning and optimization of a robot R^i. In (a) the TP only considers static obstacles and does not avoid moving ones. In (b) the TP considers both static and moving obstacles.

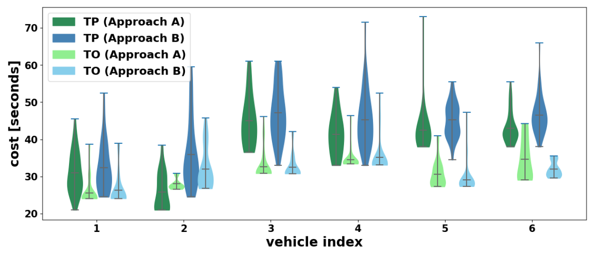
In order to compare approaches A and B, the problem in Subsection 6.3, regarding the six vehicles set $\mathcal{R}$, was solved fifty times with both approaches. The results are depicted in Figures 5a-c. The data regarding the computational time[1] (Figure 5a) and the trajectory duration (Figure 5b) of each module (TP and TO), for both approaches, was gathered for each one of the six agents. Finally, the frequency of

[1]The simulations were performed in a desktop with an Intel Core i7 U-Processor Processor 1.80GHz, 16GB installed RAM and an Intel(R) UHD Graphics. The operative system used was Microsoft Windows 11 Pro.

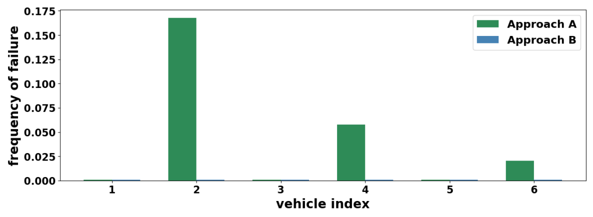
failed trajectory generations is plotted in Figure 5c. A failure occurs when the solver of the TO module cannot find a locally optimal trajectory given the initial guess provided from TP.



(a) Computational time



(b) Trajectory duration (cost)



(c) Frequency of failure

Figure 5: Violin plots regarding computational time (a) and trajectory duration (b) for the TP and TO algorithms of each vehicle. Frequency of failure for both approaches (c).

Firstly, concerning approach A, it can be verified that the TP module is computationally faster. In fact, the Kinodynamic RRT tree is significantly slower to reach the goal region when dealing with moving obstacles. However, approach A's TO cannot always optimize the trajectory given by the TP, since it can be infeasible when considering moving obstacles. This lead to a higher frequency of failures. Regarding approach B, the altered TP is computationally slower, but the solver can always find a solution. Thus, it can be concluded that there are both advantages and disadvantages when using either approach A or B.

Consequently, a third method was design, represented in Figure 6, that contemplates both approaches. When generating the trajectory of vehicle R^i using approach C, it is obtained an initial guess using the TP without inter collision avoidance, which is provided to the TO. If the solver cannot achieve an optimized trajectory, a new initial trajectory is computed using the TP with inter collision avoidance and, subsequently, sent to the TO. In this paper, the following results will be obtained using approach C.

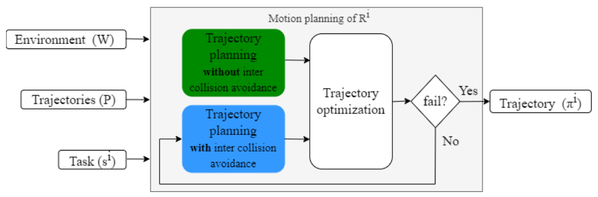


Figure 6: Architecture of approach C to solve the motion planning and optimization of a robot R_i. Initially, the TP only considers static obstacles. If the TO cannot find a feasible solution, the TP is reran, considering now both static and moving obstacles.

6.5. Scalability Study

Next, the algorithm's behaviour to overcrowded spaces and how many routes can be built on this specific map is examined. To do that, the initial and final poses of each trajectory were generated randomly inside the defined scenario, ensuring that all initial poses were separated with a distance greater than a safe margin d_{safe}. Also, all trajectories were assigned at the same time instant, i.e, $t_{1,\dots,N} = 0s$. For an agent R^i, the motion planning module is executed based on approach C. If a trajectory can be successfully generated, then the process is repeated for a new agent, R^{i+1}. If a trajectory π^i cannot be found, the number of fails is increased, until a maximum value, n_{max}, is reached. In this case, the algorithm stops the search since it cannot add any more agents to the environment.

This procedure that analyzes the scalability of the method was repeated 20 times. The histogram presenting the frequency of the maximum number of agents allowed in the environment can be found in Figure 7. The case with the least amount of agents was obtained with 16 vehicles. On the other hand, the situation in which was able to add more agents to the scenario was reached with 25 vehicles.

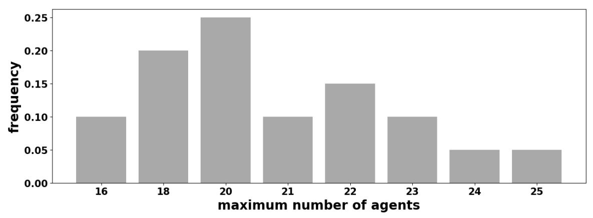


Figure 7: Histogram with frequency of maximum number of agents that can be added to the studied scenario.

6.6. Fifty-Agent Problem

A set of fifty vehicles, $\mathcal{R} = \{R^1, ..., R^{50}\}$, was used to analyse the computational complexity of the TO module with an increasing number of agents. To do that, all trajectories were assigned at the same time instant. The initial and final poses of each trajectory were generated randomly inside the defined map. Once again, it was verified if all initial poses were at a distance larger than d_{safe}. Additionally, was attempted to create trajectories with similar lengths. To do that, in the random generation process, was ensured that the distance between initial and final poses was inside a defined range.

Unlike the previous case, in which was studied the behavior of the proposed method in an overcrowded environment, now the purpose is to analyze the computational complexity with an increasing number of agents. Therefore, the map of the environment was enlarged and the boundaries were removed. Figure 8 represents the enlarged map, as well as a solution for the fifty-agent motion planning problem.

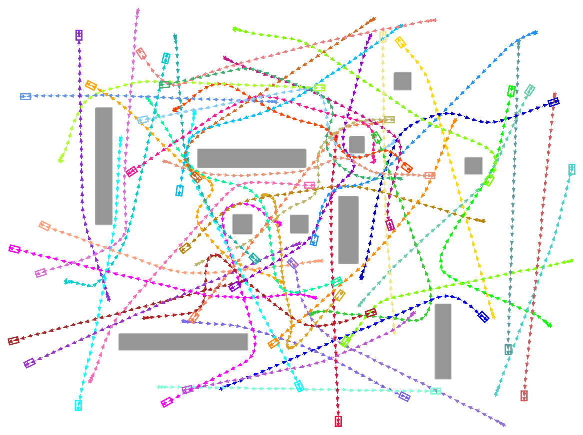
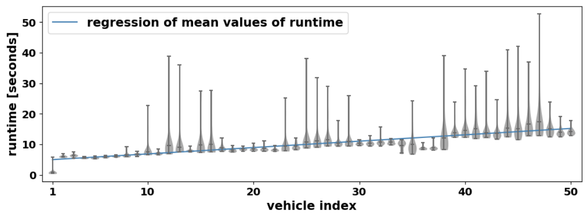


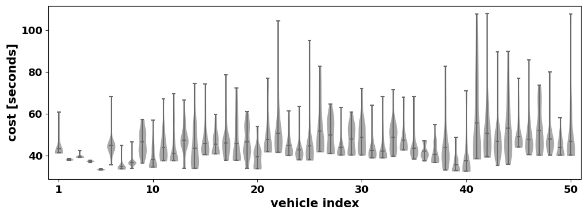
Figure 8: Solution for the fifty-multi agent problem. The trajectories of all vehicles were overlaid. The assigned initial pose of each robot is represented by its respective rectangular shape.

In addiction, the fifty-agent motion planning problem was solved fifty times for each agent. The data regarding the TO computational time of each agent was collected and presented in Figure 9a. As previously mentioned, the first trajectory generated is treated as a single-robot problem. This leads to a visibly faster computation compared to the remaining agents. Generally, the computational time of the TO increases slightly with the addition of new vehicles. In fact, the environment becomes progressively more crowded and the space for the vehicles to move more scarce. Figure 9b depicts the data collected regarding the trajectory duration of the generated routes for each vehicle.

Furthermore, a regression analysis of the mean values of computational time of each vehicle was made to measure the relationship between the complexity and the number of agents. The resulting regression line, presented in Figure 9a, enabled us to estimate a growth rate of 0.208 seconds per agent. This result indicates that the complexity scales slower than an unitary linear rate with the number of vehicles.



(a) Data distribution regarding computational time of the Trajectory Optimization module for every vehicle.



(b) Data distribution regarding trajectory duration obtained for every vehicle.

Figure 9: Data collected regarding the Trajectory Optimization of each vehicle, generated fifty times.

7. Conclusions

This paper proposes a method that generates collision-free and kinodynamic feasible trajectories for a fleet of autonomous vehicles with non-linear constraints in non-convex scenarios. The motion planning problem was decoupled into two modules. The first module performs a sampling-based Kinodynamic RRT trajectory search to find a collision-free route, while the second one uses trapezoidal direct collocation to optimize the previous solution into a smooth and collision-free trajectory. This formulation presented only one restriction regarding static and moving obstacle avoidance. As the main novelty of this paper, a set of signed distance fields was used to describe the environment and the dynamic obstacles through time. This renders the solver performance independent from the number of agents.

First, the proposed method was validated for a 6 agent problem. Then, the algorithm scalability was studied. Finally, a 50 agent motion optimization problem was analyzed and, it was verified that, the computational complexity had an increase with a rate less than unitary, with the number of vehicles.

References

[1] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. In *Journal of Field Robotics*, volume 29, pages 315–378. Wiley Online Library, 2012. doi : 10.1002/rob.20414.

[2] R. Saber, W. Dunbar, and R. Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *American Control Conference*, volume 3, pages 2217–2222. IEEE, 2003. doi : 10.1109/ACC.2003.1243403.

[3] W. Meng, Z. He, R. Su, A. Shehabinia, L. Lin, R. Teo, and L. Xie. Decentralized control of multi-UAVs for target search, tasking and tracking. In *IFAC Proceedings Volumes*, volume 47, pages 10048–10053. Elsevier, 2014. doi : 10.3182/20140824-6-ZA-1003.02665.

[4] Y. Zhao, Z. Zheng, and Y. Liu. Survey on computational-intelligence-based UAV path planning. In *Knowledge-Based Systems*, volume 158, pages 54–64. Elsevier, 2018. doi : 10.1016/0021-9991(88)90002-2.

[5] A. Wolek and C. Woolsey. Model-based path planning. In *Sensing and Control for Autonomous Vehicles*, page 183–206. Springer, 2017. doi : 10.1007/978-3-319-55372-69.

[6] J. C. Latombe. *Robot Motion Planning*, chapter 2-3. Kluwer Academic Publishers, USA, 1991.

[7] L. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE Transactions on Robotics and Automation*, volume 12, pages 566 – 580, September 1996. doi : 10.1109/70.508439.

[8] S. M. LaValle. *Planning algorithms.* Cambridge university press, 2006.

[9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE International Conference on Robotics and Automation*, 2:500–505, 1985. doi : 10.1109/ROBOT.1985.1087247.

[10] J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming.* Society for Industrial and Applied Mathematics, 2nd edition, 2010.

[11] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. In *SIAM Review*, volume 59, pages 849–904. SIAM, 2017. doi: 10.1137/16M1062569.

[12] J. T. Betts. Survey of numerical methods for trajectory optimization. In *Journal of guidance, control, and dynamics*, volume 21, pages 193–207, 1998. doi: 10.2514/2.4231.

[13] D. Mellinger, A. Kushleyev, and V. Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *IEEE international conference on robotics and automation*, pages 477–483. IEEE, 2012. doi: 10.1109/ICRA.2012.6225009.

[14] F. Augugliaro, A. P. Schoellig, and R. D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1917–1922, 2012. doi: 10.1109/IROS.2012.6385823.

[15] M. Čáp, P. Novák, A. Kleiner, and M. Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. In *IEEE Transactions on Automation Science and Engineering*, volume 12, pages 835–849. IEEE, 2015. doi: 10.1109/TASE.2015.2445780.

[16] Y. Chen, M. Cutler, and J. P. How. Decoupled multiagent path planning via incremental sequential convex programming. In *IEEE International Conference on Robotics and Automation*, pages 5954–5961. IEEE, 2015. doi: 10.1109/ICRA.2015.7140034.

[17] D. R. Robinson, R. T. Mar, K. Estabridis, and G. Hewer. An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments. In *IEEE Robotics and Automation Letters*, volume 3, pages 1215–1222. IEEE, 2018. doi: 10.1109/LRA.2018.2794582.

[18] S. M. LaValle and J. Kuffner Jr. Randomized kinodynamic planning. In *The international journal of robotics research*, volume 20, pages 378–400. SAGE Publications, 2001. doi: doi.org/10.1177/027836401220674.

[19] T. Chan and Wei Z. Level set based shape prior segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1164–1170. IEEE, 2005. doi: 10.1109/CVPR.2005.212.

[20] J. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. In *Mathematical Programming Computation*, volume 11, pages 1–36. Springer, 2019. doi: 10.1007/s12532-018-0139-4.

[21] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. In *Mathematical programming*, volume 106, pages 25–57. Springer, 2006. doi: 10.1007/s10107-004-0559-y.