MULTI-ROBOT MOTION PLANNING: A MODIFIED RECEDING HORIZON APPROACH FOR REACHING GOAL STATES

JOSÉ M. MENDES FILHO[a, b], ERIC LUCET[a, *]

[a] *CEA, LIST, Interactive Robotics Laboratory, Gif-sur-Yvette, F-91191, France*

[b] *ENSTA Paristech, Unité d'Informatique et d'Ingénierie des Systèmes, 828 bd des Marechaux, 91762, France*

[*] corresponding author: `eric.lucet@cea.fr`

ABSTRACT.
This paper proposes the real-time implementation of a multi-robot optimal collision-free motion planner algorithm based on a receding horizon approach, for the navigation of a team of mobile robots in presence of different structures of obstacles. The method is simulated with three robots. Impact of parameters is studied with regard to computation time, obstacle avoidance and travel time.

KEYWORDS: multi-robot motion planning, nonholonomic mobile robot, decentralized planning, receding horizon.

1. INTRODUCTION

The control of mobile robots is a long-standing subject of research in the iterative robotics domain. A trending application of mobile robots systems is their use in industrial supply-chains for processing orders and optimizing products storage and distribution. Companies such as Amazon and the logistic provider IDEA Groupe employ mobile multi-robot systems (Kiva systems, Scallog respectively) for autonomously processing clients orders [1, 2]. Such logistics tasks became increasingly complex as uncertainties sources, such as human presence, are admitted in the work environment.

One basic requirement for such mobile robot systems is the capacity of motion planning, i.e., generating admissible configuration and input trajectories that connects two arbitrary states. For solving the motion planning problem, different constraints must be taken into account, in particular:

- robot's kinematic and dynamic constraints;

- geometric constraints;

The first constraints derive directly from the mobile robot architecture implying, for example, in nonholonomic constraints and sliding phenomena. Geometric constraints result from need of preventing the robot to assume specific configurations in order to avoid collisions, communication lost, etc.

We are particularly interested in solving the problem of planning a trajectory for a team of nonholonomic mobile robots in a partially known environment occupied by static obstacles being near optimal with respect to the execution time (time spent from going from initial to goal configurations).

A great amount of work towards collision-free motion planning for cooperative multi-robot systems has been proposed. That work can be split into centralized and decentralized approaches. Centralized approaches are usually formulated as an optimal control problem that takes all robots in the team into account at once. This produces more optimal solutions compared to decentralized approaches. However, the computation time, security vulnerability and communication requirements can make it impracticable specially for a great number of robots [3].

Decentralized methods based in probabilistic [4] and artificial potential fields [5] approaches, for instance, are computationally fast. However, they are inapplicable to real-live scenarios as collision avoidance is dealt as a function to be minimized which value increases as paths leading to collision are considered.

Other decentralized algorithms are based on receding horizon approaches. In [6] a brief comparison of the main decentralized receding methods is made as well as the presentation of the base approach extended in our work. In this approach each robot optimizes only its own trajectory at each computation/update horizon. In order to avoid robot-to-robot collisions and lost of communication, neighbors robots exchange information about their intended trajectories before performing the update. The intended trajectories are computed by each robot ignoring coupling constraints, that is, constraints that take the other robots into account. Identified drawbacks of this approach are the dependence on several parameters for achieving real-time performance and good solution optimality, the difficulty to adapt it for handling dynamic obstacles, the impossibility of bringing the robots to a precise goal state and the limited geometric representation of obstacles.

Therefore, in this paper, we propose a motion planning algorithm that extends the approach presented in [6] so goal states can be precisely reached and more complexes forms of obstacles handled. Furthermore, we perform a study of how the methods parameters impact a set of performance criteria. Thus, this decentralized algorithm is able to find collision-free tra-

jectories and computes the corresponding angular and longitudinal velocities for a multi-robot system in presence of static obstacles perceived by the robots as they evolve in their environment. The dynamic trajectories computed are near optimal with respect to the total time spent going from the initial configuration to the final one.

This paper is structured as follows: The second section states the problem to be resolved pointing out the cost function for motion planning and all constraints that need to be respected by the computed solution. The third explains the method to resolve the motion planning problem and gives some remarks on how to resolve the constrained optimization problems associated with the method. The forth section is dedicated to the results found using this method and the analysis of the certain performance criteria and how they are impacted by the algorithm parameters. Finally, in last section we present our conclusions and perspectives.

2. PROBLEM STATEMENT

2.1. ASSUMPTIONS

In the development of this approach, the following assumptions are made:

(1.) The motion of the multi-robot system begins at the instant t_{init} and goes until the instant t_{final}.

(2.) The team of robots consists of a set $\mathcal{R}$ of B nonholonomic mobile robots.

(3.) A robot (denoted R_b, $R_b \in \mathcal{R}$, $b \in \{0, \dots, B-1\}$) is geometrically represented by a circle of radius ρ_b centered at (x_b, y_b).

(4.) All obstacles in the environment are considered static. They can be represented by a set $\mathcal{O}$ of M static obstacles.

(5.) An obstacle (denoted O_m, $O_m \in \mathcal{O}$, $m \in \{0, \dots, M-1\}$) is geometrically represented either as a circle or as a convex polygon. In the case of a circle its radius is denoted r_{O_m} centered at (x_{O_m}, y_{O_m}).

(6.) For a given instant $t_k \in [t_{init}, t_{final}]$, any obstacle O_m having its geometric center apart from the geometric center of the robot R_b of a distance inferior than the detection radius $d_{b,sen}$ of the robot R_b is considered detected by this robot. Therefore, this obstacle is part of the set $\mathcal{O}_b$ ($\mathcal{O}_b \subset \mathcal{O}$) of detected obstacles.

(7.) A robot has precise knowledge of the position and geometric representation of a detected obstacle, i.e., obstacles perception issues are neglected.

(8.) A robot can access information about any robot in the team by using a wireless communication link.

(9.) Latency, communication outages and other problems associated to the communication between robots in the team are neglected.

(10.) Dynamics was neglected.

(11.) The input of a mobile robot R_b is limited.

2.2. CONSTRAINTS AND COST FUNCTIONS

After introducing the motion planning problem in Section 1 and giving the assumptions in the previous Subsection, we can define the constraints and the cost function for the multi-robot navigation.

(1.) The solution of the motion planning problem for the robot R_b represented by the pair $(q_b^*(t), u_b^*(t))$ – $q_b^*(t) \in \mathbb{R}^n$ being the solution trajectory for the robot's configuration and $u_b^*(t) \in \mathbb{R}^p$ the solution trajectory for the robot's input – must satisfy the robots kinematic model equation:

$$\dot{q}_b^*(t) = f(q_b^*(t), u_b^*(t)), \quad \forall t \in [t_{init}, t_{final}]. \quad (1)$$

(2.) The planned initial configuration and initial input for the robot R_b must be equal to the initial configuration and initial input of R_b:

$$q_b^*(t_{init}) = q_{b,init}, \quad (2)$$

$$u_b^*(t_{init}) = u_{b,init}. \quad (3)$$

(3.) The planned final configuration and final input for the robot R_b must be equal to the goal configuration and goal input for R_b:

$$q_b^*(t_{final}) = q_{b,goal}, \quad (4)$$

$$u_b^*(t_{final}) = u_{b,goal}. \quad (5)$$

(4.) Practical limitations of the input impose the following constraint: $\forall t \in [t_{init}, t_{final}]$, $\forall i \in [1, 2, \cdots, p]$,

$$|u_{b,i}^*(t)| \leq u_{b,i,max}. \quad (6)$$

(5.) The cost for the multi-robot system navigation is defined as:

$$L(q(t), u(t)) = \sum_{b=0}^{B-1} L_b(q_b(t), u_b(t), q_{b,goal}, u_{b,goal}) \quad (7)$$

where $L_b(q_b(t), u_b(t), q_{b,goal}, u_{b,goal})$ is the integrated cost for one robot motion planning (see [6]).

(6.) To ensure collision avoidance with obstacles, the euclidean distance between a robot and an obstacle (denoted $\mathrm{d}(R_b, O_m) \mid O_m \in \mathcal{O}_b, R_b \in \mathcal{B}$) has to satisfy:

$$\mathrm{d}(R_b, O_m) \geq 0. \quad (8)$$

For the circle representation of an obstacle the distance $\mathrm{d}(R_b, O_m)$ is defined as:

$$\sqrt{(x_b - x_{O_m})^2 + (y_b - y_{O_m})^2} - \rho_b - r_{O_m}.$$

For the convex polygon representation, the distance was calculated using three different definitions, according to the Voronoi region [7] R_b is located.

Figure 1 shows an quadrilateral $ABCD$ representation of an obstacle where three of the nine regions were distinguished.

For these three regions, the distance robot-to-quadrilateral is computed as follows:

(a) If robot in region 1:

$$\sqrt{(x_b - x_A)^2 + (y_b - y_A)^2} - \rho_b$$

which is simply the distance of the robot to the vertex A.

(b) If robot in region 2:

$$\mathrm{d}(s_{DA}, (x_b, y_b)) - \rho_b$$

where

$$\mathrm{d}(s_{DA}, (x_b, y_b)) = \frac{|a_{s_{DA}} x_b + b_{s_{DA}} y_b + c_{s_{DA}}|}{\sqrt{a_{s_{DA}}^2 + b_{s_{DA}}^2}}.$$

The distance $\mathrm{d}(s_{DA}, (x_b, y_b))$ represents the distance from the robot to the side DA.

(c) If robot in region 3:

$$- \min\left(\mathrm{d}(s_{AB}, (x_b, y_b)), \cdots, \mathrm{d}(s_{DA}, (x_b, y_b))\right) - \rho_b$$

which represents the amount of penetration of the robot in the obstacle.

The distance computation for other regions can be easily inferred from equations in items ((6.).a.) and ((6.).b.).
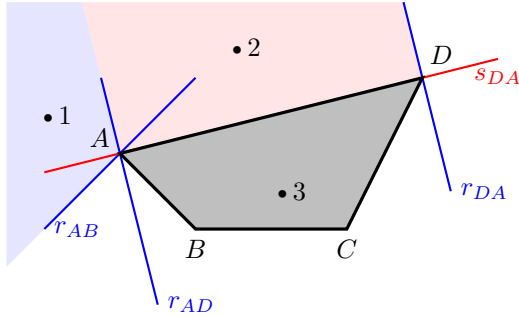

FIGURE 1. Voronoi regions used for case differentiation.

(7.) In order to prevent inter-robot collisions, the following constraint must be respected: $\forall \, (R_b, R_c) \in \mathcal{R} \times \mathcal{R}, b \neq c, c \in \mathcal{C}_b$,

$$\mathrm{d}(R_b, R_c) - \rho_b - \rho_c \geq 0 \qquad (9)$$

where $\mathrm{d}(R_b, R_c) = \sqrt{(x_b - x_c)^2 + (y_b - y_c)^2}$ and $\mathcal{C}_b$ is the set of robots that present a collision risk with R_b.

(8.) Finally, the need of a communication link between two robots (R_b, R_c) yields to the following constraint:

$$\mathrm{d}(R_b, R_c) - \min(d_{b,com}, d_{c,com}) \leq 0 \qquad (10)$$

with $d_{b,com}, d_{c,com}$ the communication link reach of each robot and $\mathcal{D}_b$ is the set of robots that present a communication lost risk with R_b.

3. Distributed motion planning

3.1. Receding horizon approach

Since the environment is progressively perceived by the robots and new obstacles may appear as time passes, planning the whole motion from initial to goal configurations at a moment previous to the beginning of the motion is not a satisfying approach. Planning locally and replanning is a more suitable approach for taking new information, as they come, into account. Besides, the computation cost of finding a motion plan using the first approach may be prohibited high if the planning complexity depends on the distance between start and goal configurations.

Therefore, an on-line planner that computes the configuration and input trajectories as the multi-robot system evolves in the environment is proposed. One possible way to do so is to use a receding horizon control approach [8].

Two fundamental concepts of this approach are the planning horizon T_p and update/computation horizon T_c. T_p is the timespan for which a solution will be computed and T_c is the time horizon during which a plan is executed while the next plan, for the next timespan T_p, is being computed. The problem of producing a motion plan during a T_c time interval is called here a receding horizon planning problem.

For each receding horizon planning problem the following is done:

Step 1. All robots in the team compute an intended solution trajectory (denoted $(\hat{q}_b(t), \hat{u}_b(t))$) by solving a constrained optimization problem where coupling constraints are ignored, that is, constraints 9 and 10 that involve other robots in the team.

Step 2. Robots involved in a potential conflict (i.e. risk of collision or lost of communication) update their trajectories computed during Step 1 by solving another constrained optimization problem that additionally takes into account coupling constraints (9 and 10). This is done by using the other robots' intended trajectories computed in the previous step as an estimate of those robots' final trajectories. If a robot is not involved in any conflict, Step 2 is not executed and its final solution trajectory is identical to the one estimated in Step 1.

All robots in the team use the same T_p and T_c for assuring synchronization when exchanging information about their positions and intended trajectories.

For each of these steps and for each robot in the team one constrained optimization problem is resolved. The cost function to be minimized in those optimization problems is the distance of a robot's current configuration to its goal configuration. This assures that the robots are driven towards their goal configurations.

This two step scheme is explained in details in [6, 9] where constrained optimization problems associated to the receding horizon optimization problem are formulated.

However, constraints related to the goal configuration and goal input of the motion planning problem are neglected in the receding horizon scheme presented in [9]. Constraints 4 and 5 are left out in the receding horizon scheme. For taking them into account, a termination procedure is proposed in the following that enables the robots to reach their goal state.

3.2. Motion planning termination

As the robots evolve in the environment using the decentralized receding horizon planning, their configuration approximate to the goal configuration. But simply stopping the motion planner as the robots are in the neighbourhood of their final configuration is not a satisfying approach (it leaves constraints 4 and 5 unsatisfied).

Therefore, after stopping the receding horizon planning algorithm, we propose a termination planning that considers those constraints associated to the goal state and that plans for unknown, to be determined planning horizon. This enables the robots to reach their goal states.

The criterion used to pass from the receding horizon planning to the termination planning is based on the distance between goal and current position of the robots and is defined in the equation 11:

$$d_{rem} \geq d_{min} + T_c \cdot v_{max} \qquad (11)$$

This condition ensures that the termination plan will be planned for at least a d_{min} distance from the robot's goal position. This minimal distance is assumed to be sufficient for the robot to reach the goal configuration.

After stopping the receding horizon planning, we calculate new parameters for the solution representation and computation taking into account the estimate remaining distance and the typical distance traveled for a T_p planning horizon. This is done in order to rescale the configuration intended for a previous planning horizon not necessarily equal to the new one. Potentially, this rescaling will decrease the computation time for the termination planning.

The following pseudo code 1 summarizes the planning algorithm and the Figure 2 illustrates how plans would be generated through time by the algorithm.

In the pseudo code, we see the call of a PlanSec procedure. It corresponds to the resolution of the receding horizon planning problem as defined in subsection 3.1.

PlanLastSec is the procedure solving the termination planning problem. This problem is similar to the receding horizon planning problems. It also has the two steps presented before for computing an intended plan and for updating it, if need be, so conflicts are avoided. The difference consists in how the optimization problems associated to it are defined. The optimization problem defined in equations 12 and 13 is the problem solved at the first step which generates an intended plan. The optimization problem associated with the update step is defined in equations 14 and 15 and produces the final solution $(q_b^*(t), u_b^*(t))$. Besides, in both new constrained optimal problems, the planning horizon is not a fixed constant as before, instead it is a part of the solution to be found.

Then, for generating the intended plan the following is resolved:

$$\min_{\hat{q}_b(t), \hat{u}_b(t), T_f} L_{b,f}(\hat{q}_b(t), \hat{u}_b(t), q_{b,goal}, u_{b,goal}) \qquad (12)$$

under the following constraints for $\tau_k = kT_c$ with k the number of receding horizon problems solved before the termination problem:

$$\begin{cases} \dot{\hat{q}}_b(t) = f(\hat{q}_b(t), \hat{u}_b(t)), \quad \forall t \in [\tau_k, \tau_k + T_f] \\ \hat{q}_b(\tau_k) = q_b^*(\tau_{k-1} + T_c) \\ \hat{u}_b(\tau_k) = u_b^*(\tau_{k-1} + T_c) \\ \hat{q}_b(\tau_k + T_f) = q_{b,goal} \\ \hat{u}_b(\tau_k + T_f) = u_{b,goal} \\ |\hat{u}_{b,i}(t)| \leq u_{b,i,max}, \quad \forall i \in [1,p], \forall t \in (\tau_k, \tau_k + T_f) \\ d(R_b, O_m) \geq 0, \quad \forall O_m \in \mathcal{O}_b, t \in (\tau_k, \tau_k + T_f) \end{cases}$$
$$(13)$$

And for generating the final solution:

$$\min_{q_b^*(t), u_b^*(t), T_f} L_{b,f}(q_b^*(t), u_b^*(t), q_{b,goal}, u_{b,goal}) \qquad (14)$$

under the following constraints:

$$\begin{cases} \dot{q}_b^*(t) = f(q_b^*(t), u_b^*(t)), \quad \forall t \in [\tau_k, \tau_k + T_f] \\ q_b^*(\tau_k) = q_b^*(\tau_{k-1} + T_c) \\ u_b^*(\tau_k) = u_b^*(\tau_{k-1} + T_c) \\ q_b^*(\tau_k + T_f) = q_{b,goal} \\ u_b^*(\tau_k + T_f) = u_{b,goal} \\ |u_{b,i}^*(t)| \leq u_{b,i,max}, \quad \forall i \in [1,p], \forall t \in (\tau_k, \tau_k + T_f) \\ d(R_b, O_m) \geq 0, \quad \forall O_m \in \mathcal{O}_b, \forall t \in (\tau_k, \tau_k + T_f) \\ d(R_b, R_c) - \rho_b - \rho_c \geq 0, \quad \forall R_c \in \mathcal{C}_b, \forall t \in (\tau_k, \tau_k + T_f) \\ d(R_b, R_d) - \min(d_{b,com}, d_{d,com}) \geq 0, \quad \forall R_d \in \mathcal{D}_b, \\ \qquad \qquad \forall t \in (\tau_k, \tau_k + T_f) \\ d(q_b^*(t), \hat{q}_b(t)) \leq \xi, \quad \forall t \in (\tau_k, \tau_k + T_f) \end{cases}$$
$$(15)$$

A possible definition for the $L_{b,f}$ cost function present in the equations above can be simply T_f. The sets $\mathcal{O}_b$, $\mathcal{C}_b$ and $\mathcal{D}_b$ are functions of τ_k.

3.3. Strategies for solving the constrained optimization problems

3.3.1. Flatness property

As explained in [9], all mobile robots consisting of a solid block in motion can be modeled as a flat system. This means that a change of variables is possible in a way that states and inputs of the kinematic model of the mobile robot can be written in terms of a new variable, called flat output (z), and its lth first derivatives. The value of $l \mid l \leq n$ depends on the kinematic model of the mobile robot. Therefore, the flat output can completely determine behavior of the system.

Algorithm 1 Motion planning algorithm

1: **procedure** PLAN
2:     $q_{latest} \leftarrow q_{initial}$
3:     $d_{rem} \leftarrow |\text{POS}(q_{final}) - \text{POS}(q_{latest})|$
4:     **while** $d_{rem} \geq d_{min} + T_c \cdot v_{max}$ **do**
5:         INITSOLREPRESENTATION($\cdots$)
6:         $q_{latest} \leftarrow$ PLANSEC($\cdots$)
7:         $d_{rem} \leftarrow |\text{POS}(q_{final}) - \text{POS}(q_{latest})|$
8:     **end while**
9:     RESCALEREPRESENTATION($\cdots$)
10:     $T_f \leftarrow$ PLANLASTSEC($\cdots$)
11: **end procedure**
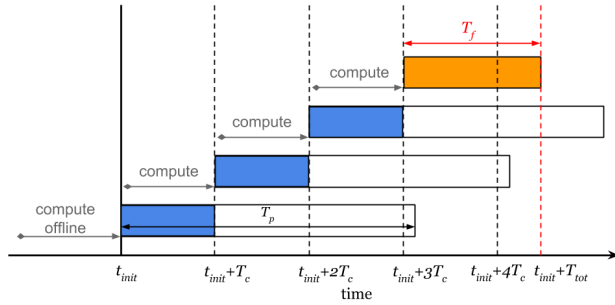
FIGURE 2. Receding horizon scheme with termination plan. The timespan T_f represents the duration of the plan for reaching the goal configuration.

Searching for a solution to our problem in the flat space rather than in the actual configuration space of the system presents advantages. It prevents the need for integrating the differential equations of system (constraint 1) and reduces the dimension of the problem of finding an optimal admissible trajectory. After finding (optimal) trajectories in the flat space, it is possible to retrieve back the original configuration and input trajectories.

3.3.2. PARAMETRIZATION OF THE FLAT OUTPUT BY B-SPLINES

Another important aspect of this approach is the parametrization of the flat output trajectory. As done in [10], the use of B-spline functions present interesting properties:

- It is possible to specify a level of continuity C^k when using B-splines without additional constraints.

- B-spline presents a local support – changes in parameters values have a local impact on the resulting curve.

The first property is very well suited for parametrizing the flat output since its lth first derivatives will be needed when computing the system actual state and input trajectories. The second property is important when searching for an admissible solution in the flat space; such parametrization is more efficient and well-conditioned than, for instance, a polynomial parametrization [10].

This choice for parameterizing the flat output introduces a new parameter to be set in the motion planning algorithm which is the number of non-null knots intervals (denoted simply N_{knots}). This parameter plus the l value determines how many control points will be used for generating the B-splines.

3.3.3. OPTIMIZATION SOLVER

There exists a variety of numerical optimization packages implemented in many different programming languages available for solving optimization problems [11].

Not all of those implementations are suitable for solving the particular kind of optimization problems presented before. The need of a solver that supports nonlinear equality and inequality constrains restricts the number of possible choices.

For our initial implementation of the motion planning algorithm, the SLSQP optimizer stood out as a good option. Besides being able to handle nonlinear equality and inequality constrains, its availability in the minimization module of the open-source scientific package Scipy helps to facilitate the motion planner implementation.

However, an error was experienced using this optimizer which uses the SLSQP Optimization subroutine originally implemented by Dieter Kraft [12]. As the cost function value becomes too high (typically for values greater than 10^3), the optimization algorithm finishes with the "Positive directional derivative for linesearch" error message. This appears to be a numerical stability problem experienced by other users as discussed in [13].

For working around this problem, we proposed a change in the objective functions of the receding horizon optimization problems. This change aims to keep the evaluated cost of the objective function around a known value when close to the optimal solution instead of having a cost depending on the goal configuration (which can be arbitrarily distant from current position).

We simply exchanged the goal position point in the cost function by a new point computed as follows:

$$p_{b,new} = \frac{p_{b,goal} - p_b(\tau_{s-1} + T_c)}{\text{norm}(p_{b,goal} - p_b(\tau_{s-1} + T_c))} \alpha T_p v_{b,max}$$

Where $p_{b,goal}$ and $p_b(\tau_{s-1} + T_c)$ are the positions associated with configurations $q_{b,goal}$ and $q_b(\tau_{s-1} + T_c)$ respectively, $\alpha \mid \alpha \geq 1, \alpha \in \mathbb{R}$ is a constant for controlling how far from the current position the new point is placed, the product $T_p v_{b,max}$ the maximum possible distance covered by R_b during a planning horizon and $s \mid s \in [0, k), s \in \mathbb{N}$ the current receding horizon problem index.

Numerically solving the constrained optimization problems presented before (which includes constraints equations that must hold for a time interval) introduces a new parameter in the algorithm: the time sampling for optimization N_s.

How this parameters is set impacts on the solution quality and computation time. We discuss that in the following section.

4. SIMULATION RESULTS

Here we show the results and analyses founded for the motion planner presented in the previous sections.

The trajectory and velocities shown in the Figures 3 and 4 illustrate a motion planning solution found for a team of three robots. The robots move in an environment where three static obstacles are present. Each point along the trajectory line of a robot represents the beginning of a T_c computation horizon.

It is possible to see on those figures how the planner generates configuration and input trajectories satisfying the constraints associated to the goal state.

In particular, in Figure 3 we show the resulting plan when coupling constraints are ignored (Step 2 is never performed). In Figure 4 we have a collision-free solution. The blue zones in Figure 4 are in the same position as the red ones in Figure 3. Specially near these regions a change in the trajectory is present. Complementary, changes in the robots velocities across charts in both figures can be notice. Finally, the bottom charts show that the collisions were indeed avoid: inter-robot distances in Figure 4 are greater then or equal to zero all along the simulation.

For performing these two previous simulations a reasonable number of parameters have to be set. These parameters can be categorized into two groups. The **algorithm related** parameters and the **optimization solver related** ones. Among the former group, the most important ones are:

- The number of sample for time discretization (N_s);

- The number of internal knots for the B-splines curves (n_{knots});

- The planning horizon for the sliding window (T_p);

- The computation horizon (T_c).

The latter kind depends on the numeric optimization solver adopted. However, since most of them are iterative methods, it is common to have at least the two following parameters:

- Maximum number of iterations;

- Stop condition.

This considerable number of parameters having influence on the solution and/or on the time for finding a solution makes the search for a satisfactory set of parameters' values a laborious task.

Therefore, it is important to have a better understanding of how some performance criteria are impacted by the changes in algorithm parameters.

4.1. PARAMETERS' IMPACT ANALYSES

Three criteria considered important for the validation of this method were studied. We tested different parameters configuration and scenario in order to
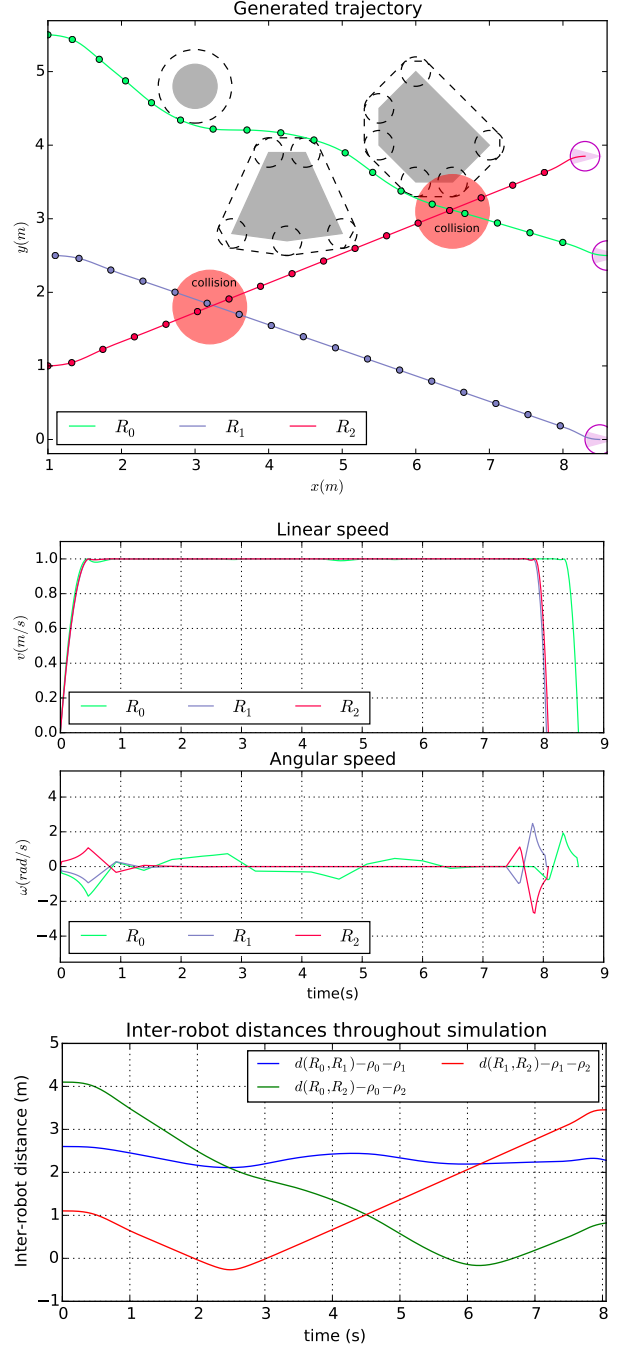

FIGURE 3. Motion planning solution without collision handling

understand how they influence those criteria. The three criteria defined for a given robot R_b are:

- *Maximum computation time* during the planning over the computation horizon (MCT/T_c ratio).

- Obstacle penetration area (P).

- Travel time (T_{tot}).

4.1.1. MAXIMUM COMPUTATION TIME OVER COMPUTATION HORIZON MCT/T_c

The significance of this criterion lays in the need of assuring the real-time property of this algorithm. In a
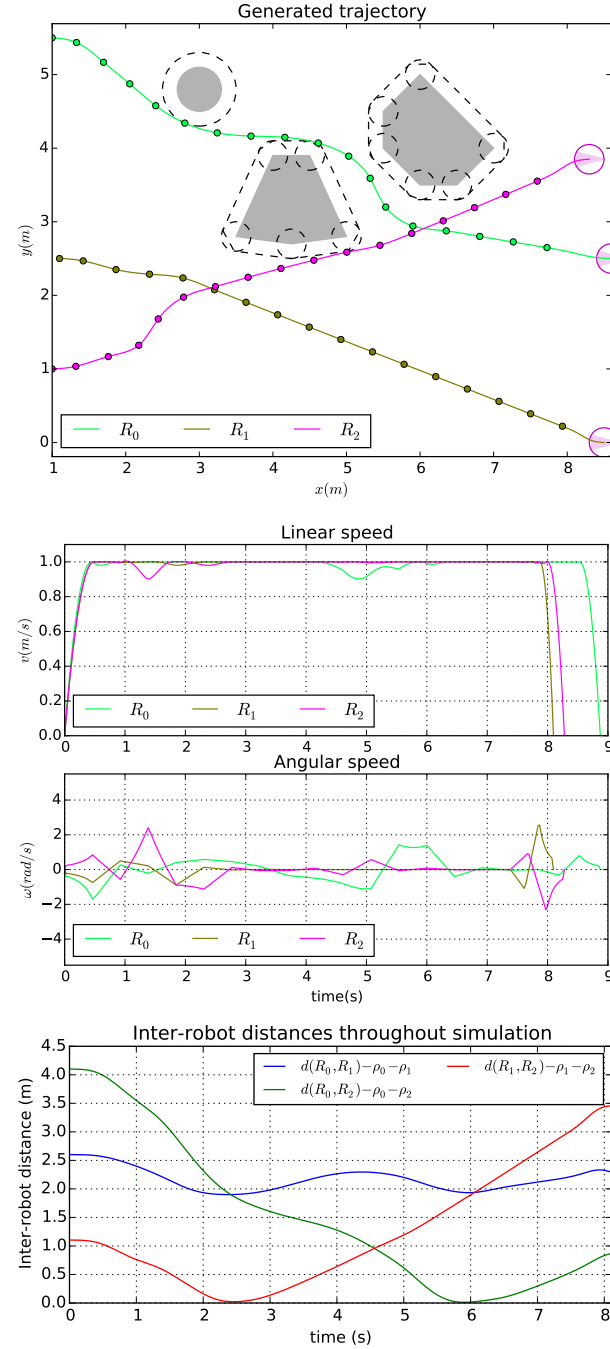
FIGURE 4. Motion planning solution with collision handling

real implementation of this approach the computation horizon would have always to be superior than the maximum time took for computing a plan.

Several simulations for different scenarios with different sets of parameters were performed.

For instance, Table 1 summarizes one of the scenarios studied for a single robot. In Figure 5 we present the results obtained from simulation in that scenario for different parameters set.

Each dot along the curves corresponds to the average of MCT/T_c along different T_p's for a given value of $(T_c/T_p, N_s)$.

The absolute values observed in the charts depend on the processing speed of the machine where the algorithm is run. Those simulations were run in an Intel Xeon CPU 2.53GHz processor.

But rather than observing the absolute values, it is interesting to analyze the impact of changes in the parameters values. It is possible to see that an increasing number of N_s increases MCT/T_c for a given T_c/T_p. Similarly, we see an increasing of MCT/T_c as the number of internal knots N_{knots} increases from chart 5a to 5c.

Further analyses of those data show that finding the solution using the SLSPQ method requires $O(N_{knots}^3)$ and $O(N_s)$ time. Although augmenting N_{knots} can yield to an impractical computation time, typical N_{knots} values used for the B-spline representation did not need to exceed 10 in our simulation, which is a sufficiently small value.

| v_{max}	1.00 m/s
ω_{max}	5.00 rad/s
q_{inital}	$[-0.05\ 0.00\ \pi/2]^T$
q_{final}	$[0.10\ 7.00\ \pi/2]^T$
$u_{initial}$	$[0.00\ 0.00]^T$
u_{goal}	$[0.00\ 0.00]^T$
O_0	$[0.55\ 1.91\ 0.31]$
O_1	$[-0.08\ 3.65\ 0.32]$
O_2	$[0.38\ 4.65\ 0.16]$

TABLE 1. Values for scenario definition

Another parameter that has direct impact on the MCT/T_c ratio is the detection radius of the robot's sensors. As the detection radius of the robot increases more obstacles are seen at once which, in turn, increases the number of constraints in the optimization problems. The impact of increasing the detection radius d_{sen} in the MCT/T_c ratio can be seen in the Figure 6 for a scenario where seven obstacles where present. The computation time stops increasing as soon as the robot sees all obstacles present in the environment.

4.1.2. OBSTACLE PENETRATION P

The obstacle penetration area (P) gives a metric for obstacle avoidance and consequently for solution quality.

Ideally we would have $P = 0$. Because time sampling is performed and non-collision constraints are evaluated for precise instants, that is not the case and $P > 0$. A way of assuring $P = 0$ would be to increase the obstacles radius computed by the robot's perception system by the maximum distance that the robot can run within the time spam T_p/N_s. However simple, this approach represents a loss of optimality and is not considered in this work.
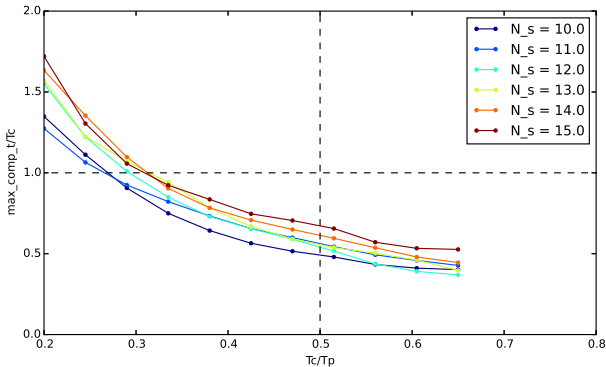
We seek then to observe the impact of the algorithm parameters in the obstacle penetration area.
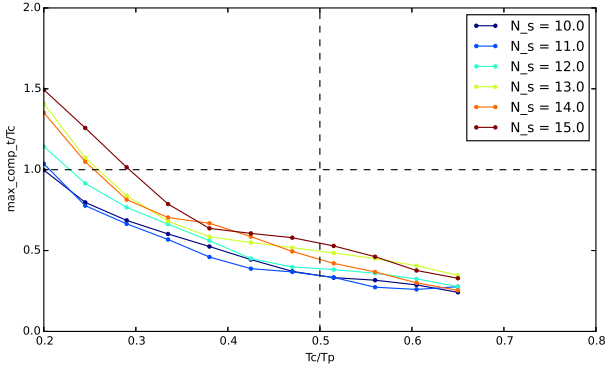
(A) . Four internal knots. Average variance between lines is 1.047×10^{-2}
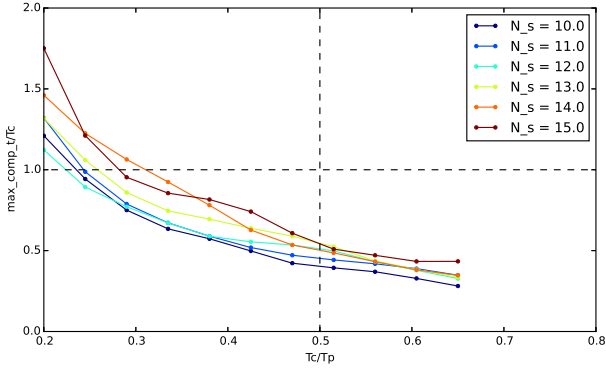
(B) . Five internal knots. Average variance between lines is 0.972×10^{-2}

(C) . Six internal knots. Average variance between lines is 0.587×10^{-2}

FIGURE 5. Three obstacles scenario simulations

T_c/T_p ratio, N_{knots} and d_{sen} impact on this criteria is only significant for degraded cases, that is to say that around typical values those parameters does not change P significantly. However, time sampling N_s has a more important impact on P. Figure 7 shows the penetration area rapidly decreasing as the number of samples increases.

4.1.3. TRAVEL TIME T_{tot}

Another complementary metric for characterizing solution quality would be the travel time T_{tot}. Analyses of data from several simulations show a tendency that for a given value for N_{knots}, N_s and T_c the travel time decreases as the planning horizon T_p decreases. This
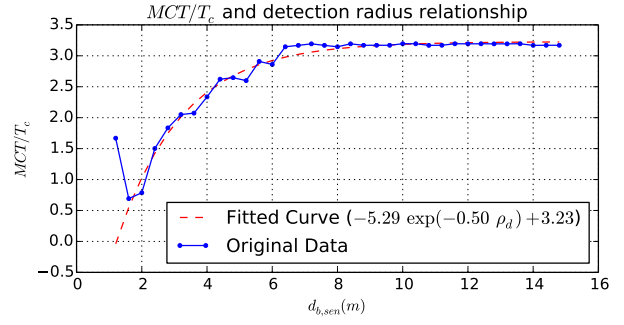

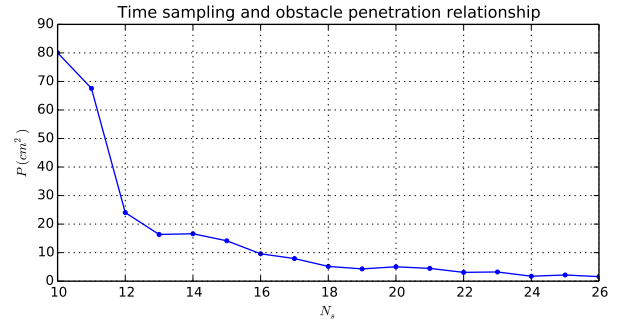FIGURE 6. Increasing of detection radius and impact on a MTC/T_c ratio

FIGURE 7. Obstacle penetration decreasing as sampling increases

can be explained by the simple fact that for a given T_c a more optimal solution (in terms of travel time) can be found if the planning horizon T_p is smaller. Another relevant observation is that the overall travel time decreases as the sampling number N_s decreases. This misleading improvement in the solution quality hides the fact that the fewer the samples the greater will be the obstacle penetration area as shown previously in Figure 7.

Furthermore, the Figure 8 shows how the travel time is invariant for changes in the detection radius far from degraded values that are too small. This points out that a local knowledge of the environment provides enough information for finding good solutions.
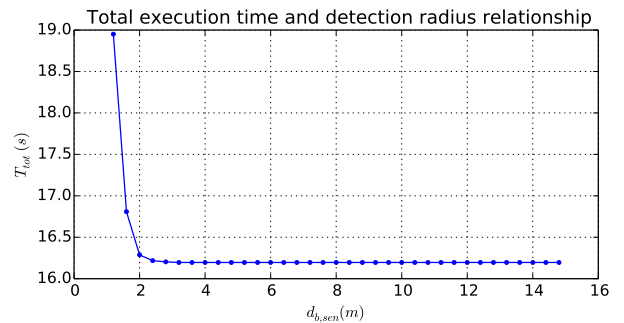

FIGURE 8. Increasing of detection radius and impact on T_{tot}

5. Conclusions

A decentralized motion planner based on a receding horizon approached modified for taking into account termination constraints was proposed. Near the goal configuration neighborhood the receding horizon approach is finished and a termination planning problem is solved for bring the robots to their precise final state. The problem is stated as constrained optimization problem which minimizes the time for reaching a goal configuration through a collision-free trajectory that secures communication between robots. Our planner can handle circle and convex polygon representation of obstacles. Key techniques for implementing the motion planner are: system flatness property, B-spline parameterization of the flat output and SLSQP optimizer. Finally, solutions using this planner for different scenarios were generated in order to validate the method analyzing how different parameters can impact the computation time and quality of the solution. Future work may be done for testing the planner in a more realistic simulation environment where dynamics is taken into account as well as sensors models and communication latency.

References

[1] S. Robarts. Autonomous robots are helping to pack your Amazon orders. `http://www.gizmag.com/amazon-kiva-fulfillment-system/34999/`. Accessed: 2015-07-22.

[2] Idea Groupe met en place Scallog pour sa prÃĺparation de commandes. `http://supplychainmagazine.fr/NL/2015/2085/`. Accessed: 2015-07-22.

[3] F. Borrelli, D. Subramanian, a.U. Raghunathan, L. Biegler. MILP and NLP Techniques for centralized trajectory planning of multiple unmanned air vehicles. *2006 American Control Conference* pp. 5763–5768, 2006. DOI:10.1109/ACC.2006.1657644.

[4] G. Sanchez, J.-C. Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *The International Journal of Robotics Research* **21**(1):5–26, 2002. DOI:10.1177/027836402320556458.

[5] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous Robot Vehicles*, pp. 396–404. Springer Science and Business Media, 1986. DOI:10.1007/978-1-4613-8997-2_29.

[6] M. Defoort, A. Kokosy, T. Floquet, et al. Motion planning for cooperative unicycle-type mobile robots with limited sensing ranges: A distributed receding horizon approach. *Robotics and Autonomous Systems* **57**(11):1094–1106, 2009. DOI:10.1016/j.robot.2009.07.004.

[7] C. Ericson. *Real-Time Collision Detection*. M038/the Morgan Kaufmann Ser. in Interactive 3D Technology Series. Taylor & Francis, 2004.

[8] T. Keviczky, F. Borrelli, G. J. Balas. Decentralized receding horizon control for large scale dynamically decoupled systems. *Automatica* **42**(12):2105–2115, 2006. DOI:10.1016/j.automatica.2006.07.008.

[9] M. Defoort. Contributions à la planification et à la commande pour les robots mobiles coopératifs. *Ecole Centrale de Lille* 2007.

[10] M. B. Milam. *Real-time optimal trajectory generation for constrained dynamical systems*. Ph.D. thesis, California Institute of Technology, 2003.

[11] R. E. Perez, P. W. Jansen, J. R. R. A. Martins. pyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. *Structures and Multidisciplinary Optimization* **45**(1):101–118, 2012. DOI:10.1007/s00158-011-0666-3.

[12] D. Kraft. *A software package for sequential quadratic programming*. DLR German Aerospace Center âĂŞ Institute for Flight Mechanics, Koln, Germany, 1988.

[13] Runtime errors for large gradients. `http://comments.gmane.org/gmane.science.analysis.nlopt.general/191`. Accessed: 2015-07-27.