



PFE - RAPPORT MI-PARCOURS
ROBOTIQUE ET SYSTÈMES EMBARQUÉS
2014/2015

Réf : DIASI / 15-351

Local Dynamic Path Planning for an Autonomous Forklift in Human Environment

Unclassified Report
Can be made public on the internet

Author:

José Magno MENDES FILHO

Promotion 2014

Supervisor - ENSTA:

David FILLIAT

Supervisor - CEA:

Éric LUCET

Internship from 05 Mars 2015 to 28 August 2015

CEA LIST Digiteo Moulon
Bât. 660 91191 GIF-SUR-YVETTE Cedex, France

1 Introduction

This document is meant to describe the work done until now as well as the partial results and the perspectives for my final year internship as an engineering student at ENSTA ParisTech and UPMC - Paris VI.

The first two months of this internship took place at the "Unité Informatique et Ingénierie des Systèmes" at ENSTA under the supervision of David Filliat. After that, from May to the present I have been working at CEA LIST Digiteo Moulon under the supervision of Eric Lucet.

1.1 Internship context

The work developed during this internship falls within the context of an applied research project on automation of a forklift truck for the effective supply of assembly lines.

Thus, the mobile robot has to be able to evolve in a partially known, shared with humans environment while being efficient with respect to time and energy spent on its tasks and preserving the workers' safety.

1.2 Objectives

The main objective of this internship is to implement, test, evaluate and improve an experimental path planning algorithm presented in details in [2] with respect to its applicability to a scenario where autonomous forklift trucks and humans share the same environment. This experimental planning algorithm consists in planning the mobile robot's path by solving a direct trajectory optimization problem [1] using B-splines for representing the system flat output [3].

As stated [2] this approach presents good advantages for multi-robots systems evolving in an uncertain environment with static obstacles over other solutions. For instance, analytic methods are inapplicable for nonholonomic systems in presence of obstacles. Cell decomposition methods have the downside of requiring a *a priori* space modeling. The dynamic window approach does not seem flexible enough to be extended to a multi-robot system.

The two main challenges that may be confronted during this work is how to guarantee real-time property for our specific application and how to generalize the algorithm in order to account for humans, i.e. dynamic obstacles.

2 Initial achievements

During the first two months of this work we focused in understanding and reproducing the trajectory generation algorithm presented in [2] going from a single robot global planning method to a multi-robot local real-time planning. During the third month we focused in the analysis of the impact of different parameters in the method performance and feasibility.

2.1 Nonlinear programming problem (NLP)

Firstly we studied how the path planning problem could be translated into a nonlinear programming problem by intelligently using the mobile robot's model flatness property and representing the trajectory by B-splines.

2.1.1 Problem Formulation

Let us briefly and without mathematical rigor present how the problem of finding a collision-free, optimized path for a mobile robot represented by a unicycle model can be written.

The equation 2.1 represents the unicycle kinematic model. Thanks to the flatness property it is possible to be only interested in planning a trajectory for the flat output variable z where $z = [x, y]^T$.

$$\begin{aligned} \dot{q} &= f(q, u) \Rightarrow \\ \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} v \cos(\theta) \\ v \sin(\theta) \\ w \end{bmatrix} \end{aligned} \quad (2.1)$$

The equations 2.2, 2.3 show how the state variables and control variables can be calculated from the flat output and its first l^{th} derivatives. This way, whenever we need to retrieve the fundamental variables we can by means of these equations.

$$\begin{aligned} \varphi_1(z(t_k), \dots, z^{(l)}(t_k)) &= \\ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} &= \begin{bmatrix} z_1 \\ z_2 \\ \arctan(\dot{z}_2/\dot{z}_1) \end{bmatrix} \end{aligned} \quad (2.2)$$

$$\begin{aligned} \varphi_2(z(t_k), \dots, z^{(l)}(t_k)) &= \\ \begin{bmatrix} v \\ \omega \end{bmatrix} &= \begin{bmatrix} \sqrt{\dot{z}_1^2 + \dot{z}_2^2} \\ \frac{\dot{z}_1 \ddot{z}_2 - \dot{z}_2 \ddot{z}_1}{\dot{z}_1^2 + \dot{z}_2^2} \end{bmatrix} \end{aligned} \quad (2.3)$$

Now let us write the NPL problem that minimizes the square of the time spend (2.4) to go from a $q_{initial}$ pose at a $u_{initial}$ velocity to a q_{final} pose at a u_{final} velocity, avoiding M static obstacles represented by circles with radius r_m ¹, having velocities in an admissible velocity set denoted by \mathcal{U} .

1. the own robot geometry is here represented by a circle of radius ρ

$$\min_{(t_{final}, C_0, \dots, C_{d+n_{knot}-2})} J = (t_{final} - t_{initial})^2 \quad (2.4)$$

under the following constraints $\forall k \in \{0, \dots, N_s - 1\}$ and $\forall m \in 0, \dots, M - 1$:

$$\begin{cases} \varphi_1(z(t_{initial}), \dots, z^{(l-1)}(t_{initial})) &= q_{initial} \\ \varphi_1(z(t_{final}), \dots, z^{(l-1)}(t_{final})) &= q_{final} \\ \varphi_2(z(t_{initial}), \dots, z^{(l)}(t_{initial})) &= u_{initial} \\ \varphi_2(z(t_{final}), \dots, z^{(l)}(t_{final})) &= u_{final} \\ \varphi_2(z(t_k), \dots, z^{(l)}(t_k)) &\in \mathcal{U} \\ d_{O_m}(t_k) &\geq \rho + r_m, \quad \forall O_m \in \mathcal{Q}_{occupied} \end{cases} \quad (2.5)$$

2.1.2 Implementation of the solution

Once we were able to write the problem as above the subsequent step was to implement this planning method using some programming language. We kept in mind that a high level language provided with some NLP solver package would be preferable.

We decided to use Python language and the Scipy package. Within the Scipy module many minimization methods can be found. For this specific optimization problem, only the method SLSPQ was appropriate. It was the only one to handle constrained minimization where the constraints could be equations as well as inequations.

Since the SLSPQ is a local optimization method the first guess used for initializing the solver had an impact on the time of convergence as well as on the found solution. A bad first guess can prevent the solver for converging at all, as shown in the figure 1.

Besides the bad first guess we notice another problem that could cause the Scipy implementation of the SLSQP solver to not converge. A too big cost value for the objective function (for instance, for values greater then 10^6) could also prevent the convergence of the solver.

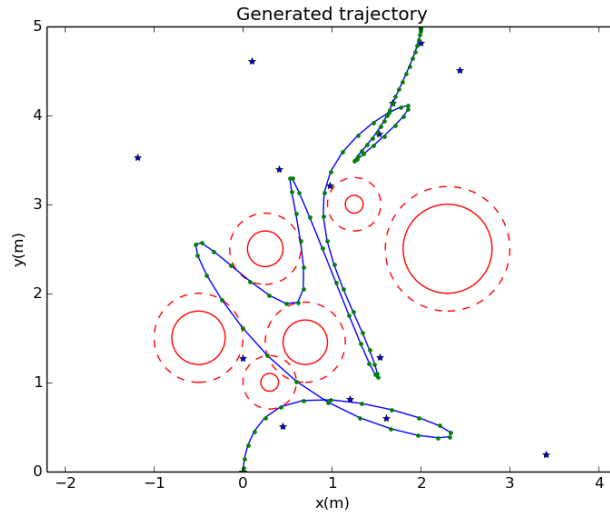


Figure 1 – Path resulting from a bad initialisation.

An initialization algorithm was then proposed along some changes in the objective function evaluation so better solutions could be achieved quicker.

The initialization algorithm is a simple one that interactively changes the positions of the B-splines control points in order to prevent the initial trajectory guess to pass between two obstacles that are too close together (distance inter-obstacle smaller than the robot diameter).

2.2 Evolving from the previous NLP to the sliding window multi-robot decentralized approach

Solving the problem as stated in the previous subsection is only worth considering as a base initial solution to our problem.

Following the work done in [2] we built over the first implementation so to have a strongly decentralized planner for multi-robot fleet that is collision-free with respect to the robots in the fleet as well as to static obstacles as before. This new approach was suitable for real-time implementation as well. We chose a strongly decentralized approach over a centralized one so no central supervisor is needed and so we can keep the computation complexity of solving the trajectory optimization problem close to the one in a single robot system.

Using a sliding window in time the new planner produced a "per robot" intended trajectory meant to be valid within a planning horizon that was locally optimal with respect to a new objective function and collision-free only with respect to the static obstacles.

The presence of other robots is taken into account in a second moment: the intended trajectory is updated after the robots involved in a possible future conflict exchange their intended trajectories so no conflict (collision or lost of communication) occur for the corrected new trajectory.

Figures 2 and 3 show two multi-robot local planning without and with collision handling.

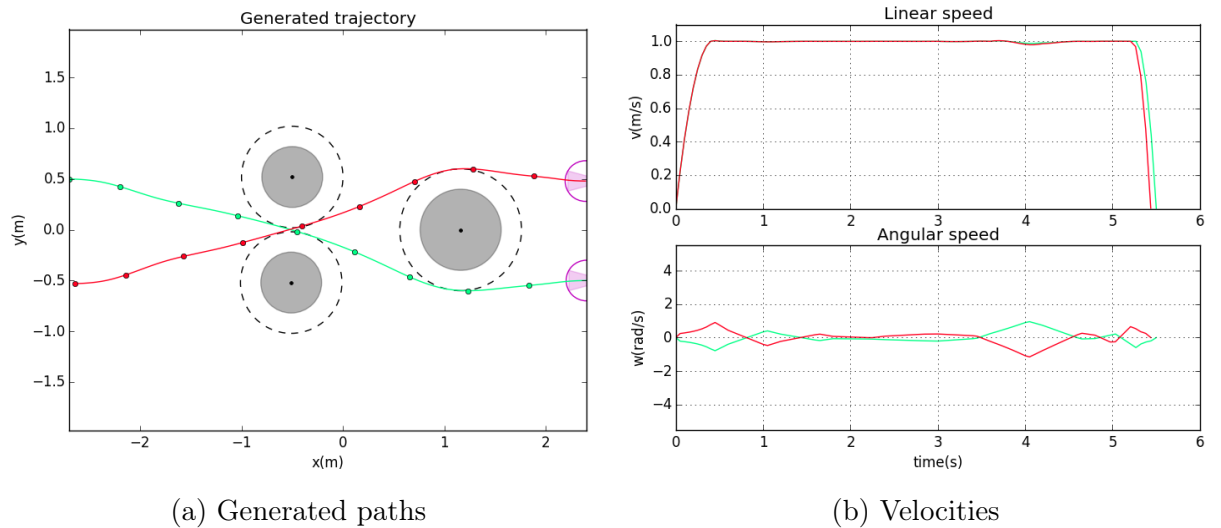


Figure 2 – Multi-robot path generation without conflict handling.

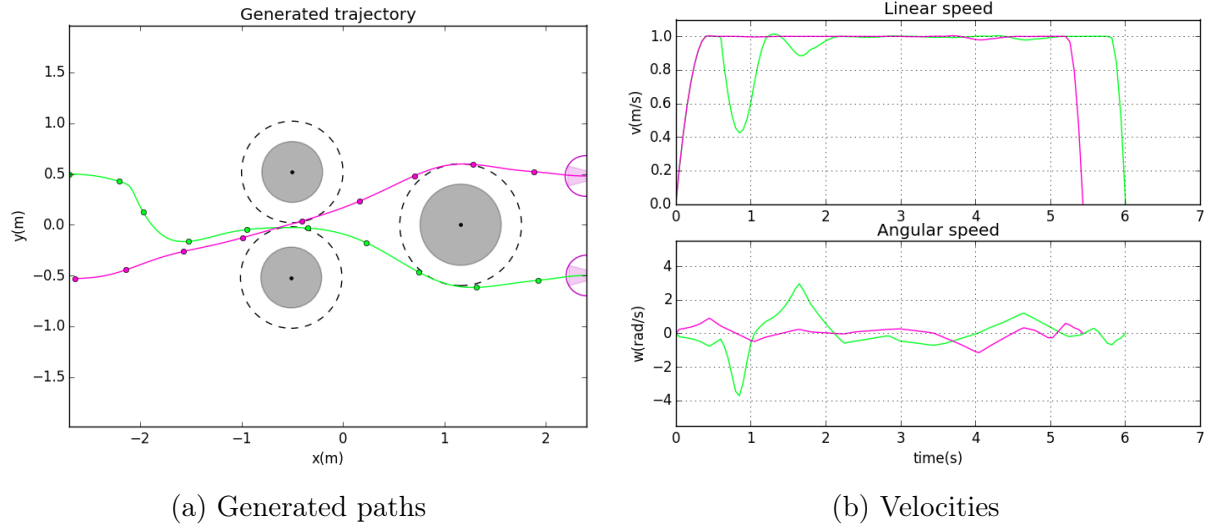


Figure 3 – Multi-robot path generation with conflict handling.

2.3 Analyzing the parameters impact on real-time feasibility and solution adequacy

The performance of the motion planning algorithm previously presented depends on several parameters. For starters these parameters can be split into two groups. The **algorithm related** parameters and the **optimization solver related** ones. Among the former group the most important ones are:

- *bullet* 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 optimization solver adopted but since most of them are iterative methods it is common to have at least the two following parameters:

- Maximum number of iterations;
- Stop condition.

The task of searching for a satisfactory set of parameters' values with regard to a performance metric (e.g. total time to complete the mission) is quite laborious.

We attempted nevertheless to extract some quantitative knowledge about how these parameters impact the generated solution based on several simulations run with different parameters configurations. The main objective here is to be able to support the feasibility of a real-time motion planner based on this algorithm.

Omitting some details about the simulations conditions we present one of the many data used in our analyze in the figures 5 and 4. Figure 4 shows the solution (path and velocities) for one of the scenarios that were simulated for a given set of parameters listed in the table 1. Keeping the scenario and varying the parameters we were able to plot the charts on figure 5. We can notice an impact of the number of samples (N_s) and number of non-null internal knots (n_{knots}) on the *maximum computation time*/ T_c ratio. The greater the n_{knots} or the N_s the greater is the *maximum computation time*/ T_c . This behavior is the one expected since the number of constraints and the number of arguments for the cost function to be minimized depend on these two parameters respectively. Keep in mind that

the *maximum computation time* $/T_c$ ratio has to be inferior to one so real-time planning is possible.

We were also able to characterize the influence of the number of obstacles seen at once on the computation time and path quality. As expected the computation time increases with the number of obstacles seen at once.

Finally we proposed some metric to characterize the adequacy of a found solution based on the total time spend going from the initial to the final pose and on the robot to the obstacles distance.

Table 1 – Motion planner main parameters

T_p	2.40 s
T_c	0.48 s
N_s	11
n_{knots}	4
v_{max}	1.00 m/s
ω_{max}	5.00 rad/s
$q_{initial}$	$[-0.05 \ 0.00 \ \pi/2]^T$
q_{final}	$[0.10 \ 7.00 \ \pi/2]^T$
u_{final}	$[0.00 \ 0.00]^T$
u_{final}	$[0.00 \ 0.00]^T$
O_0	$[0.55 \ 1.91 \ 0.31]$
O_1	$[-0.08 \ 3.65 \ 0.32]$
O_2	$[0.38 \ 4.65 \ 0.16]$

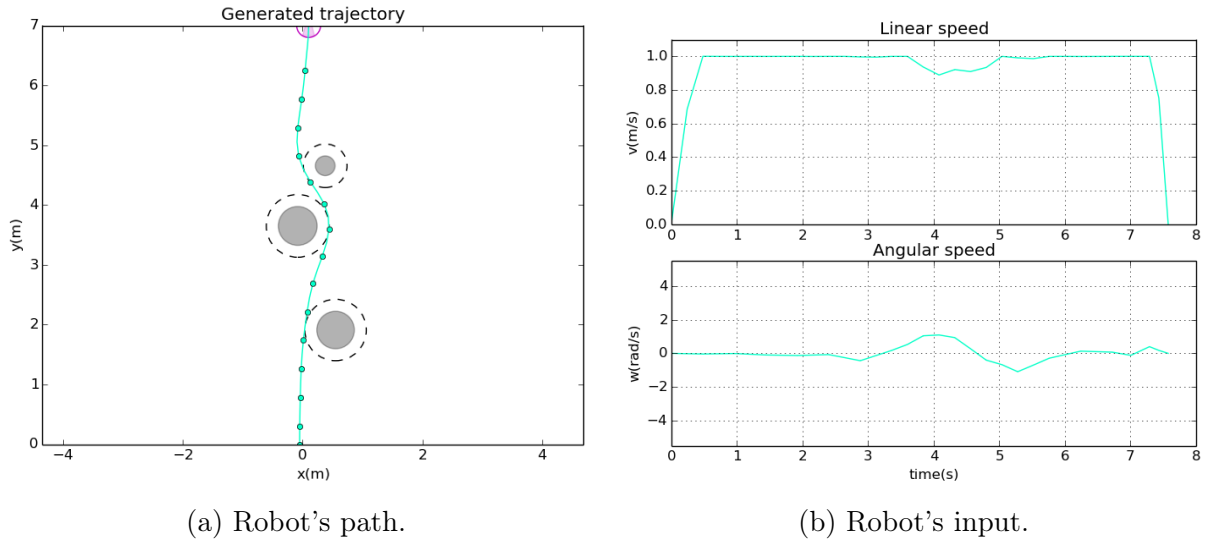
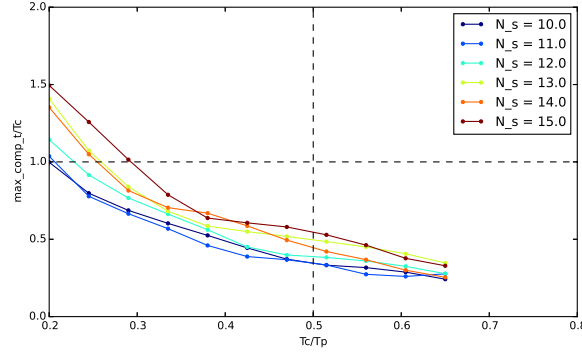
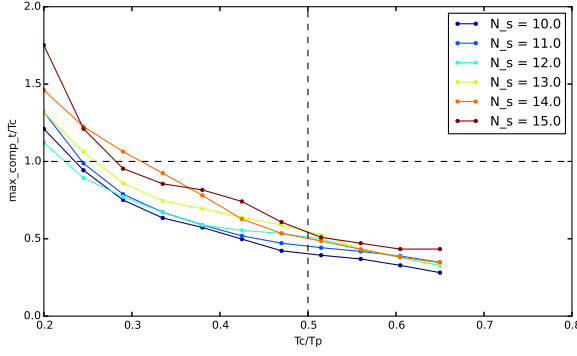


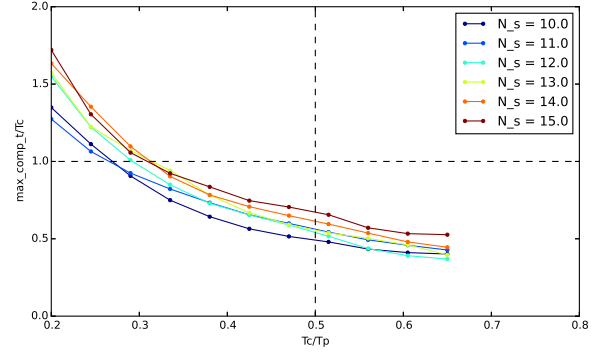
Figure 4 – Three obstacles scenario simulation example where the *maximum computation time* was about 84% of T_c and the mission total time equals to 7.57 s.



(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.

2.4 From Python to XDE simulator

XDE is a physics simulation software environment fully developed by CEA-LIST that can handle a variety of physical aspects such as deformable bodies, multibody systems with kinematic constraints and contacts, and fluids.

Its utilization presents though some rough edges and a steep learning curve.

During the third month and beginning of the fourth while producing the analysis referenced by the latest subsection we begin porting the implementation done in python to the XDE environment.

The objective is to get much closer to a real physical system being able to implement dynamic behavior in the simulated environment which was previously neglected.

3 Discussions

We implemented and did some minor improvements on the solution proposed by [2] and we gather a good understanding of the impact of the algorithm's parameters in the computation cost and in the quality of the solution.

The implemented algorithm has the quality of dealing with multi-robot systems in a robust manner, handling collision as well as loss of communication conflicts without greatly increasing the computation time.

Besides, thanks to the reduction of the solution's search space by creating a parametric representation of the trajectory, this algorithm can respect real-time constraints for a multi-robot system evolving in a unknown environment where static obstacles are present.

The biggest drawback of this approach is how to choose a set of parameters (algorithm and numerical solver related) well adapted to a given scenario. We were able to identify the parameters that has a bigger influence on the solution adequacy and computation time. The number of samples (N_s) is the parameter that greatly impacts the computing time followed by the number of obstacles detected at once by the robot (defined by the detection radius and the obstacles positioning with respect to the robot). In the other hand, the solution adequacy depends highly on the N_s/T_p ratio and on the number of internal knots of the B-spline representation n_{knots} .

4 Perspectives

During the months to come we hope to finish the implementation of the C++ code in the XDE simulator and develop the current approach so it can address dynamic obstacles.

We plan to write a paper focusing on the impact of the algorithm's parameters on the computation cost and on the solution quality for submitting to the International Workshop on On-line Decision-Making in Multi-Robot Coordination.

References

- [1] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.
- [2] Michael Defoort. Contributions à la planification et à la commande pour les robots mobiles coopératifs. *Ecole Centrale de Lille*, 2007.
- [3] Mark B Milam. *Real-time optimal trajectory generation for constrained dynamical systems*. PhD thesis, California Institute of Technology, 2003.