PFE - RAPPORT MI-PARCOURS

ROBOTIQUE ET SYSTÈMES EMBARQUÉS

2014/2015

Réf : DIASI / 15-351

Local Dynamic Motion 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

The main objective of this internship is to implement, test, evaluate and improve an experimental path planning algorithm for mobile robots with respect to its applicability to a scenario where autonomous forklift trucks and humans share the same environment.

The base planning algorithm is presented in details in [2]. It 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].

This approach have presented good results for multirobots systems evolving in an uncertain environment with static obstacles.

The main challenge that may be confronted during this work is 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 multirobot 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 in a nonlinear programming problem by intelligently using the mobile robot's model flatness property and representing the trajectory by B-splines.

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 $z = [x,\ y]^T$.

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

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.

$$\begin{gathered} \varphi_1(z(t_k), \ldots, z^{(l)}(t_k)) = \\ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \arctan(\dot{z}_2/\dot{z}_1) \end{bmatrix} \end{gathered} \tag{2.2}$$

$$\varphi_2(z(t_k), \ldots, z^{(l)}(t_k)) = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{z}_1^2 + \dot{z}_2^2} \\ \dfrac{\dot{z}_1\ddot{z}_2 - \dot{z}_2\ddot{z}_1}{\dot{z}_1^2 + \dot{z}_2^2} \end{bmatrix} \tag{2.3}$$

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 $\mathcal{U}$.

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

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

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

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 a 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.

A 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).

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

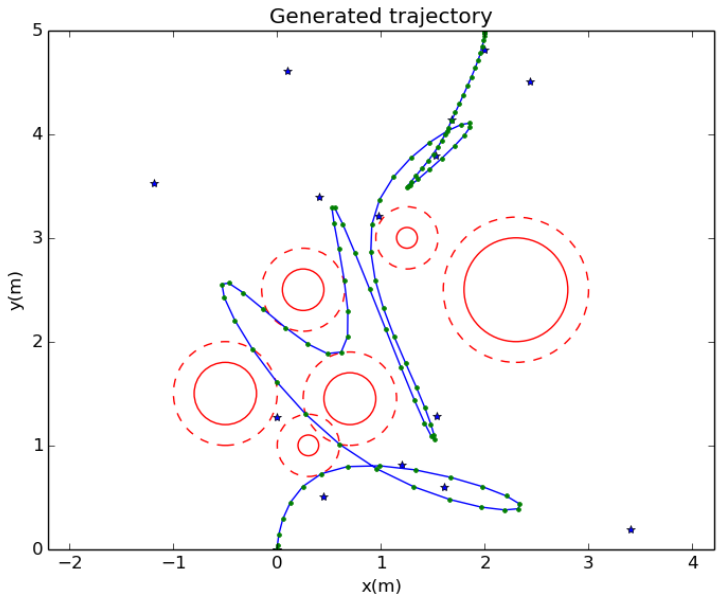


Figure 1 – Bad first guess.

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 multirobot 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.

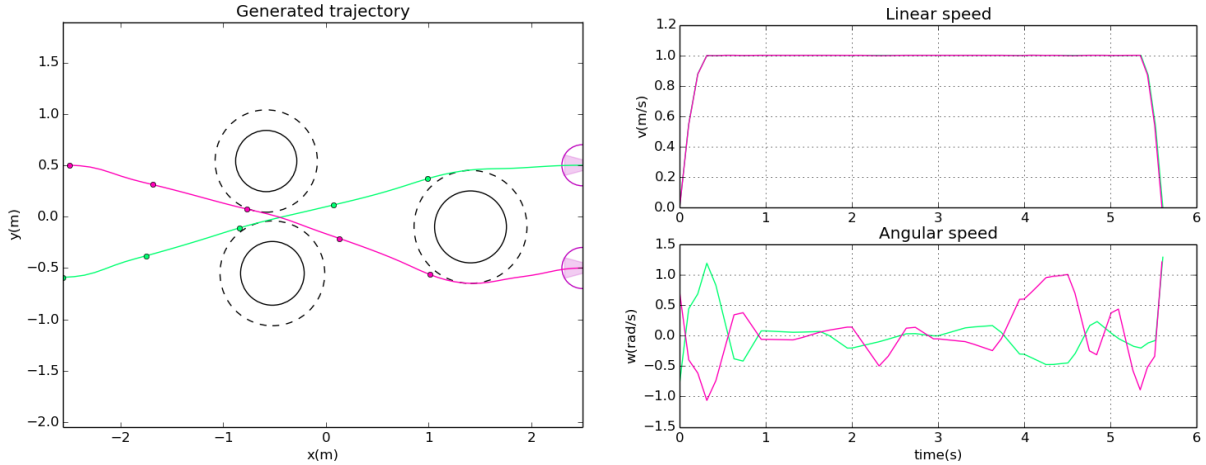
Using a sliding window in time the new planner produced a "per robot" intended trajectory meant to be valid within a planning horizon that was locally optimal with respect to a new objective function and collision-free only with respected to the static obstacles.

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 multirobot local planning without and with collision handling.

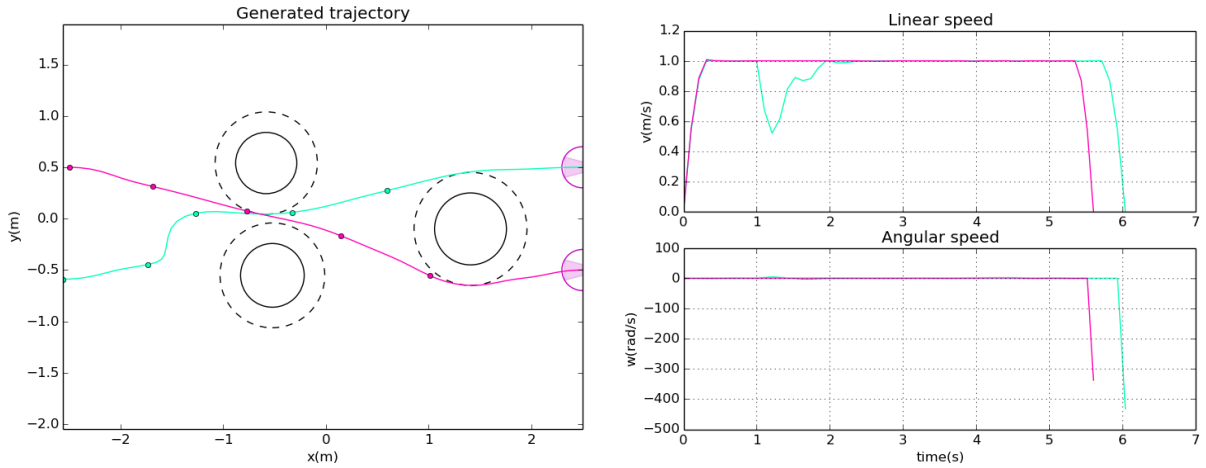
2.3 Analyzing the parameters impact on real-time feasibility and solution fitness

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: the number of sample for time discretization



(a) Generated paths (b) Velocities

Figure 2 – multirobot path generation without conflict handling.



(a) Generate paths (b) Velocities

Figure 3 – multirobot path generation with conflict handling.

(N_s), the number of internal knots for the B-splines curves (n_{knots}), and the planning and computation time horizons for the sliding windows (T_p and T_c respectively). The latter kind depends on the optimization solver adopted but since most of them are iterative methods is common to have at least a "maximum number of iterations" and a "stop condition" parameters.

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.

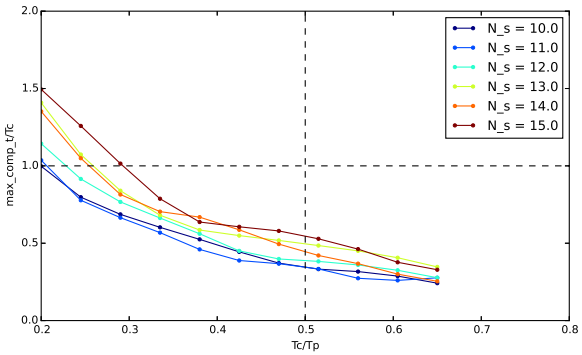
We attempted nevertheless to extract some quantitative knowledge about how these parameters impact the generated solution based on several simulations run with different parameters configurations. The main objective here is to be able to support the feasibility of a real-time motion planner based on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

Omitting some details about the simulations conditions we present one of the many data used in our analyze in the figures.

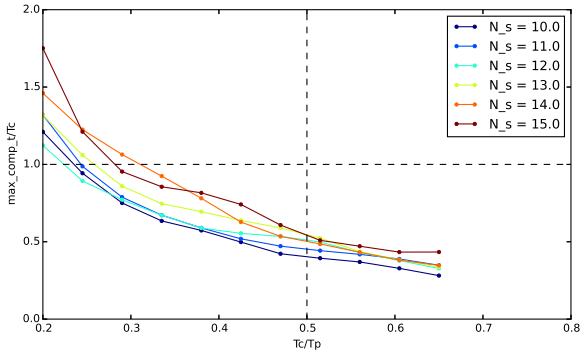
We can notice a impact of the number of samples (N_s) and number of non-null internal knots (N_{knots}). The greater the N_{knots} or the N_s the greater is the *maximum computational time*/T_c [2]. This behavior is the one expected since the number of constraints and the number of arguments for the cost function to be minimized depend on these two parameters respectively.

We were also able to characterize the influence of the number of obstacles seen at once in the computation time and path quality.

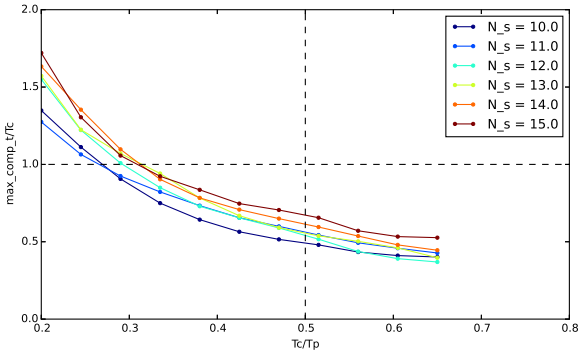
Finally we proposed some metric to characterize the fitness of a found solution based on the total time spend going from the initial to the final pose and on the proximity to the obstacles.



(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}

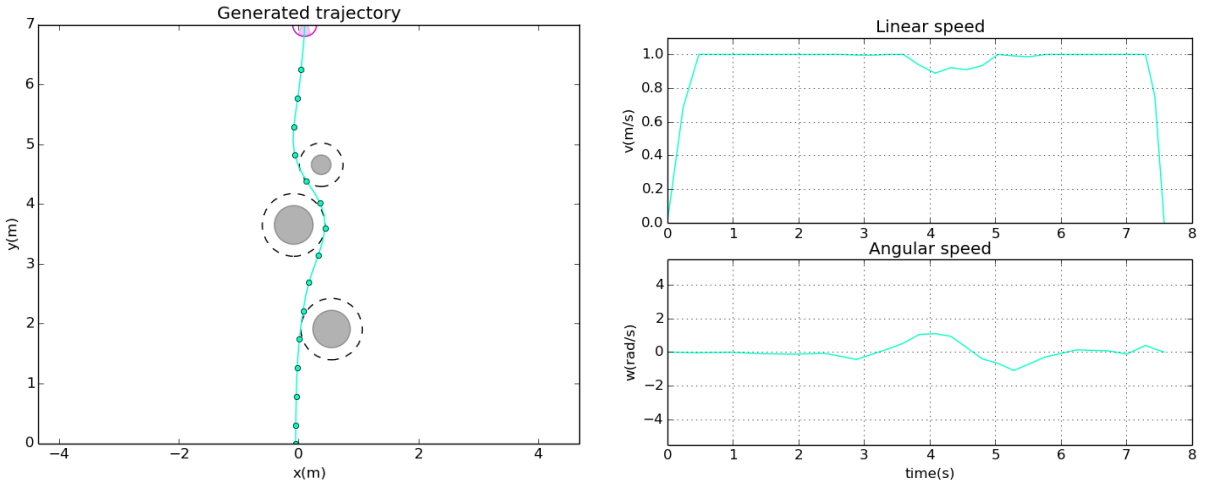
Figure 4 – Three obstacles scenario.

2.4 From Python to C++ using the 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.

2. in order to quarantine real-time this ratio has to be inferior to one



(a) Robot's path.

(b) Robot's input.

Figure 5 – Three obstacle scenario simulation example where the *maximum computational time* was about 84% of T_c and the mission total time equals to 7.57 s.

During the third month and beginning of the forth 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 in the simulated environment notions neglected at the first months. For instance the obstacle detection can be based on real sensors models carrying uncertainties instead of assuming the absolute knowledge of an obstacle position as soon as it enters within the robot's detection radius.

3 Conclusion

The work done until now is promising. We implemented and did some minors improvements on the solution proposed by [2], we gather a good understanding of the impact of the algorithm's parameters in the computation cost and in the quality of the solution and we started the porting of the algorithm to a more realistic simulation environment.

4 Perspectives

We hope being able to write a paper focused 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.

During the months to came we also hope to develop the current approach so it can address dynamic obstacles.

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.