PREPRINT 2015-07-03

TRAJECTORY OPTIMIZATION

Eric Lucet[a,*], José Magno Mendes Filho[a,b]

[a] *CEA, LIST, Interactive Robotics Laboratory, Gif-sur-Yvette, F-91191, France*
[b] *ENSTA Paristech, Unité informatique et Ingénierie des Systéme, 828 boulevard des Marechaux, 91762, France*
[*] corresponding author: `eric.lucet@cea.fr`

Abstract. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nullam mauris turpis, sagittis sed mattis vel, pulvinar eu nisi.

Keywords: keyword one, another keyword, yet another keyword.

1. Introduction

All mobile robots consisting of a solid block in motion are flat systems [1].

A flat system presents the property that states and inputs can be written in terms of the flat outputs z and its derivatives. Thus, all system behavior can be expressed by the flat outputs and a finite number of its derivatives (l).

TODO reference Veeraklaew et al. in [106] firstly combines the concepts of differential flatness and sequential quadratic programming.

TODO discussion about how to find the mapping $(q, u) \to z$ (look up Millan 2003).

The approach proposed by Defoort/Milam (and others) takes advantage of the flatness property and search a solution for the nonholonomic motion planning problem in the flat space rather than in the configuration space of the mobile robot.

In addition, their approach use B-splines for representing the solution in the flat space. This provide a small local support (TODO verify and elaborate) able to represent complex trajectories.

Finally a trajectory optimization routine is done that accounts for all constraints (nonholonomic, geometric and bounded-input) finding an appropriate solution.

This approach such as just described assume full knowledge of the environment where the mobile robot is to execute its motion. Defoort adapts this method to a sliding window architecture where the motion planning problem is devised throughout time as the mobile robot evolves in its environment. By relaxing the final state constraints, changing the objective function of the NLP and planning for a fixed timespan T_p his new planner dynamically generates the trajectory as the robot moves.

Furthermore, an adaptation of this method is done in [1] for multi robot systems. Thanks to the exchange of information among the different robots they can adapt theirs trajectories to avoid robot-to-robot collisions and loss of communication in a decentralized fashion.

2. Mono-robot sliding window planning algorithm

The basis algorithm for planning for a mono robot is based on the optimization problem shown in equation **??** Two aspects of the implementation of this algorithm are neglected in [1]: the initial values for the control points and the procedure for reaching a desired final state.

$$\min_{(C_{(0,\tau_k)},\ldots,C_{(d+n_{knot}-2,\tau_k)})} J_{\tau_k} = \|\varphi_1(z(\tau_k+T_p,\tau_k),\ldots,z^{(l-1)}(\tau_k+T_p,\tau_k)) - q_{final}\|^2 \tag{1}$$

under the following constraints $\forall t \in [\tau_k, \tau_k + T_p]$:

$$\begin{cases} \varphi_1(z(\tau_k,\tau_k),\ldots,z^{(l-1)}(\tau_k,\tau_k)) & = & q_{ref}(\tau_k,\tau_{k-1}) \\ \varphi_2(z(\tau_k,\tau_k),\ldots,z^{(l)}(\tau_k,\tau_k)) & = & u_{ref}(\tau_k,\tau_{k-1}) \\ \varphi_2(z(t,\tau_k),\ldots,z^{(l)}(t,\tau_k)) & \in & \mathcal{U} \\ d_{O_m}(t,\tau_k) & \geq & \rho + r_m, \quad \forall O_m \in \mathcal{O}(\tau_k) \end{cases} \tag{2}$$

2.1. Initilization

The initialization of the solution paths used for each NLP solving is important for two reasons. A good initialization allows the optimization solver to find a better solution for a given timespan. When using a local optimization method the initialization impacts the final trajectory.

The simplest of initializations was performed in our studies. Linear spacing from current flat output value to the estimate final flat output. The estimate final output is simply the flat output computed from final states and inputs assuming a displacement of the maximum linear speed of the mobile robot times the planning horizon and assuming direction equals to the (final postion - current pos direction).

2.2. Stop condition and last NLP

As the robot evolves its state approximates to the final state since the distance between current and final state is what is to be minimized. At some point the constraints associated to the final state shall be integrated into the NLP and the timespan for performing this last step shall not be fixed and must be one of the values calculated.

The criterion used to pass from the NLP used during for the initial and intermediates steps to the last step NLP is define below in the equation **??**:

$$d_{rem} \geq d_{min} + T_c \cdot v_{max} \tag{3}$$

This way we insure that the last planning section will be done for at least a d_{min} distance from the robot's final position. This minimal distance is assumed to be sufficient for the robot to reach the final state.q

Algorithm 1 Sliding window planning algorithm

procedure Plan
  $knots \leftarrow$ GenKnots$(t_p, d_{spl}, n_{knots})$
  $time \leftarrow$ LineSpacing$(0, t_p, n_s)$
  $q_{latest} \leftarrow q_{initial}$
  $d_{rem} \leftarrow |$Pos$(q_{final})-$Pos$(q_{latest})|$
  while $d_{rem} \geq d_{min} + T_c \cdot v_{max}$ **do**
    $q_{latest} \leftarrow$ PlanSec
    $d_{rem} \leftarrow |$Pos$(q_{final})-$Pos$(q_{latest})|$
  end while
  $s \leftarrow$ Min$(\frac{d_{rem}}{v_{max} \cdot t_p}, 1.0)$
  $n_{knots} \leftarrow$ Max(Round$(s \cdot n_{knots}), d_{spl})$
  $n_s \leftarrow$ Max(Round$(s \cdot n_s), n_{knots} + d_{spl})$
  $\Delta t \leftarrow$ PlanLastSec
end procedure

3. Decentralized multi robot sliding window planning algorithm

A straight forward extension of the previous algorithm can be done in order to support a multi robot system. The sliding window algorithm presented before remains virtually the same. The changes are done within the PlanSec PlanLastSec routine.

After solving the NLP stated before each robot will have generated an intended trajectory that would be valid if we were dealing with a mono robot system. For the multi robot system some exchange of information among the robots and possibly some replanning has to be done.

Right after solving the standalone NLP a given robot represented by the index i computes a conflict list that is based on all robots' positions as of when they started to plan their intended trajectories. This conflict list contains the indexes of the robots in the fleet that can possibly cause some conflict. Conflict here is understood as a collision or a loss of communication between robots in the fleet.

Notice that the i robot can compute its conflict list as soon as it finishes its planning even though other robots may still be doing so.

Conflict detection is computed TODOThe additional constraints associated to the multi robot system TODO

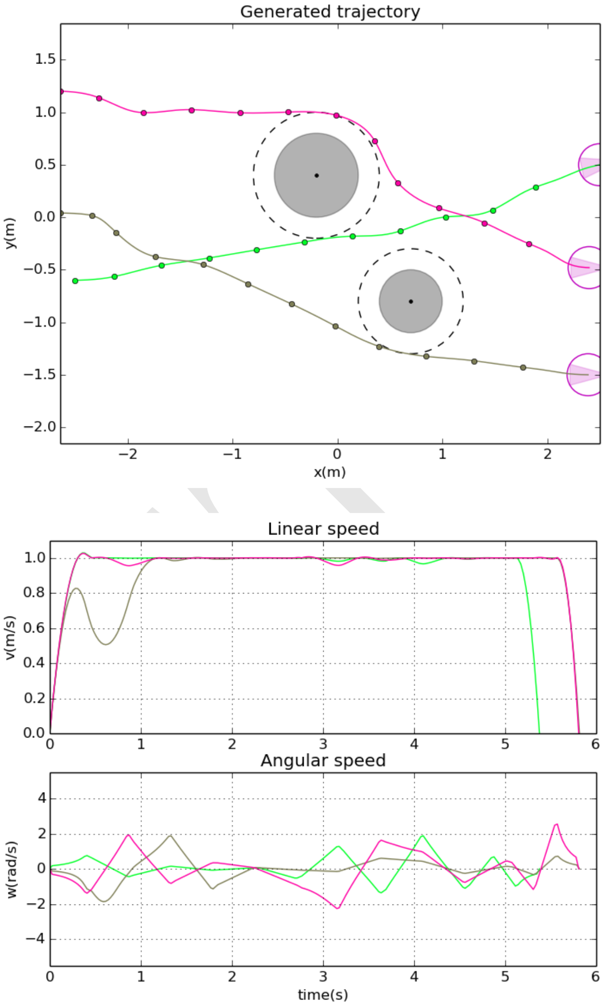

Figure 1. Our results: black box (top) and black box (bottom).

For the next step of replanning all robots involved in a conflict have to be done computing the first standalone planning. This is needed simply because all intended trajectories will be taken into account on the replanning part.

Using the intended trajectory as the initialization of the optimization parameters a new NLP is solved where collision avoidance between robots and keeping communication are translated into constraints.

After solving this second NLP, the trajectories are updated and the planning goes on to the next section.

3.1. Conflict detection

Conflict detection is computed TODO

3.2. Aditional constraints

The additional constraints associated to the multi robot system TODO

4. Parameters' impact analyses

4.1. Maximum computation time over computation horizon MCT/T_c

4.2. Obstacle penetration P

4.3. Mission total time T_{tot}

5. Conclusions

References

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

[2] DOI Foundation. *DOI Handbook*, 2014. `http://www.doi.org/hb.html`, DOI:10.1000/182.