

Nonlinear predictive control: trajectory optimization for an unicycle robot

Lecture notes for robotics/control engineering students

Antonio Sala, Leopoldo Armesto
Universitat Politècnica de València

Videos:

<http://personales.upv.es/asala/YT/V/mpcunic1EN.html> (these slides: the big picture),

<http://personales.upv.es/asala/YT/V/mpcunic2EN.html> (actual code).



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Outline

Motivation:

We wish to move a robot from configuration “A” to configuration “B” in an “optimal” way... easy, isn't it?.

Objectives:

Consider the big picture apart from solving a “toy” predictive control problem.

Contents:

Problem statement, analysing the solution, implementation issues.



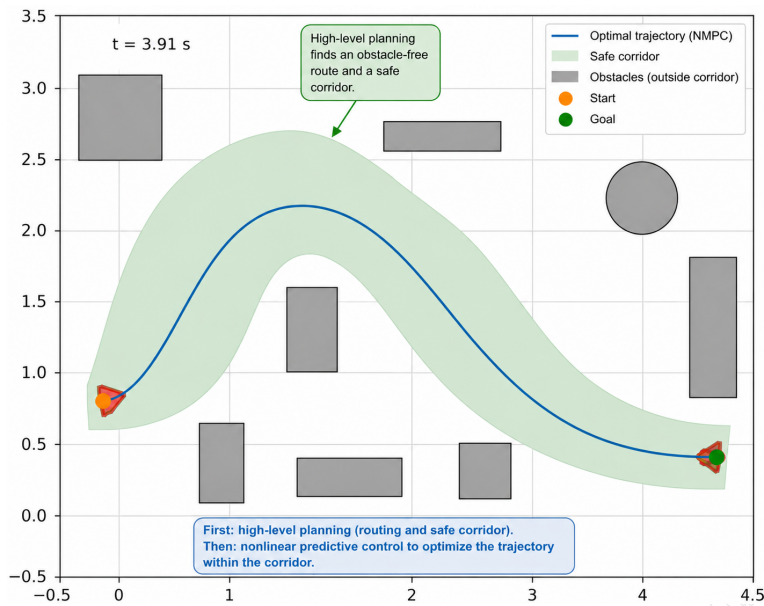
Overall methodology

(too generic to be meaningful)

- A Correctly state the problem to be solved:
practical requirements → mathematical optimization
- B Solve the problem
- C Analyze the solution
- D Implement and prepare for the unexpected



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



A. Stating the problem

- 1 Set up generic problem-independent constraints (max speeds, joint limits, ...)
- 2 Set start configuration
- 3 Set goal configuration (equality) or goal "region"
- 4 Set a desired cost index to optimize (say, minimize) reflecting your "practical" goals on how the motion should be in a single number
 - Maybe you wish to minimize time taken to carry out the task
 - or, say, you want smooth maneuvers (minimizing the squared derivatives, second-derivatives or third-derivatives of some quantities), nice velocity, acceleration, jerk...
 - or a combination of both ideas (smooth operation but take less than 12 seconds)



B. Solve the problem

- 1 High-level planning... In a “maze”-type problem you need binary decisions (take left route?). You need to solve a “routing” problem. Randomized algorithms, maybe (RRT, A*, PRM, ...) or mixed-integer programming, combinatorial optimization, graph search ...
- 2 Find a “safe corridor”, translate it to problem-dependent constraints.
- 3 Devise an initial guess for all decision variables (maybe infeasible), possibly from step 1 above.
- 4 Solve numerically (usually, gradient-based, automatic differentiation, ..., but you may encounter solutions based on **calculus of variations**, say, in space operations)
- 5 Check that the found solution is actually feasible



C. Analyze the solution

- 1 Check that solution is feasible, and, additionally, that it is not an “awkward”, spurious, local minimum.
- 2 Maybe we hit a local minimum: can you think on sensible additional initial guesses (or, well, can you just repeat with a random initial guess)? If so, repeat and check if a better solution is obtained.
- 3 Plot state and control action trajectories. Are they “smooth” and “intuitively sensible”?
- 4 In robotics: animate if possible



D. Implementation

MPC result is an “**open-loop**” state and control action trajectory.

In a **robotic arm** with precise encoders and low-level controllers, replaying the computed motion might be a viable solution, particularly in kinematic setups (if robot software handles “waypoint-to-waypoint-at-given-velocity” commands).

However, in “**mobile**” robotics or aerospace applications, we may sense a position/orientation different to the predicted one, wheels may slide, etc. so we need “closed loop” implementations to compensate for these (hopefully small) deviations.



Closed-loop implementation

Real-time MPC or just follow trajectory waypoints?

- real-time nonlinear MPC is the best option in theory, but it may be computationally costlier, and may get stuck in unfeasible/spurious minima.
- As an alternative, we might follow the NMPC solution trajectory with a simple error-based (say, PID plus velocity/acceleration feedforward) controller.
- In between, we may linearize the trajectory (**time-varying linearization**) or even easier, linearize around a fixed operating point if trajectory close to it (**time-invariant linearization**) and solve a Linear MPC problem (via, say, quadratic programming); if we repeat said linearization, we get a nonlinear MPC solution via sequential quadratic programming (which, well, might not be suited for fast real-time operation).



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

E. Prepare for the unexpected

The previous slide was considering “small” deviations from the predicted trajectory which, somehow might be “expected”... but we might have “large” unexpected issues:

- Your map isn't as accurate as you thought: wall is 1 meter closer than you thought, corridor is too narrow, door is closed.
- Your localization on said map isn't as accurate as your thought (sound/gps reflections say you crossed a wall, interferences...)
- You find an "unexpected-but-fixed" obstacle (a chair)
- You find an "unexpected and moving" obstacle (another robot, a pedestrian).

E.2) What to do?

- **FULL replanning:** costly, and it might require predicting “uncertain” future positions of moving obstacles, feasibility not guaranteed (moreover if uncertainty accumulates over time). It may also require **SLAM** (simultaneous localization and mapping) to amend an incorrect map.
- Hope for the best: create an **artificial “repulsion force”** from obstacles (gradient of a “potential field”) and apply it jointly with your prior control action. If you enter an “U”-shaped region, you are doomed but, well, we don’t expect U-shaped pedestrians, do we?.



