Motion Planner – Python library overview

for any questions, please contact stephen-monnet@hotmail.com

1. Main goals

This library is based on the mpc_motion_planner project (C++), which aims to generate a trajectory $\boldsymbol{x}(\cdot) = \{\boldsymbol{q}(\cdot), \dot{\boldsymbol{q}}(\cdot), \ddot{\boldsymbol{q}}(\cdot)\}$ between two arbitrary points $\boldsymbol{x_0} = \{\boldsymbol{q_0}, \dot{\boldsymbol{q}}_0, \ddot{\boldsymbol{q}}_0\}$ and $\boldsymbol{x_f} = \{\boldsymbol{q_f}, \dot{\boldsymbol{q}}_f, \ddot{\boldsymbol{q}}_f\}$ for the Franka Panda robot with 7 degrees of freedom. This trajectory can be generated either (a) using ruckig or (b) using polympc with ruckig as the starting solution. It should be noted that (a) only allows for considering box constraints on velocity, acceleration, and jerk, while (b) allows for adding non-linear constraints such as the minimum height of the tool or the maximum permissible torque.

In addition to making the functionalities of mpc_motion_planner accessible in Python, this library extends its capabilities to the Kuka iiwa7 and Kuka iiwa14 robots, offering numerous possibilities for manipulating and analyzing the generated data.

2. Model

a. Dynamic

For polympc, the robot's dynamics are simply modelled as a double integrator between the input $\boldsymbol{u}$ and the joint positions $\boldsymbol{q}$ (thus, $\boldsymbol{u} = \ddot{\boldsymbol{q}}$):

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \ddot{\boldsymbol{q}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0}_7 & \boldsymbol{I}_7 \\ \boldsymbol{0}_7 & \boldsymbol{0}_7 \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{0}_7 \\ \boldsymbol{I}_7 \end{pmatrix} \cdot \boldsymbol{u}$$

b. Objective Function (Polympc)

The objective of the NOCP (Nonlinear Optimal Control Problem) is to minimize the total duration of the trajectory that connects the initial state $\boldsymbol{x_0}$ to the final state $\boldsymbol{x_f}$. For more information on the formulation, refer to Jennings problem.

c. Ruckig constraints :

$\dot{\boldsymbol{q}}_{min} \le \dot{\boldsymbol{q}}(\cdot) \le \dot{\boldsymbol{q}}_{max}$: **Joint Velocity**

$\ddot{\boldsymbol{q}}_{min} \le \ddot{\boldsymbol{q}}(\cdot) \le \ddot{\boldsymbol{q}}_{max}$: **Joint Acceleration**

$\dddot{\boldsymbol{q}}_{min} \le \dddot{\boldsymbol{q}}(\cdot) \le \dddot{\boldsymbol{q}}_{max}$: **Joint Jerk**

d. Polympc constraints :

$\boldsymbol{q}_{min} \le \boldsymbol{q}(\cdot) \le \boldsymbol{q}_{max}$: **Joint Position**

$\dot{\boldsymbol{q}}_{min} \le \dot{\boldsymbol{q}}(\cdot) \le \dot{\boldsymbol{q}}_{max}$: **Joint Velocity**

$\ddot{\boldsymbol{q}}_{min} \le \ddot{\boldsymbol{q}}(\cdot) \le \ddot{\boldsymbol{q}}_{max}$: **Joint Acceleration**

$\boldsymbol{\tau}_{min} \le \boldsymbol{\tau}(\cdot) \le \boldsymbol{\tau}_{max}$: **Joint Torque**

$x_{ee}(\cdot) = [0 \quad 0 \quad 1] \cdot \boldsymbol{FK}_{xyz}(\boldsymbol{q}(\cdot)) \ge \boldsymbol{h}_{min}$: **End Effector Height**

3. Main Architecture of the Framework

The main task handled by the mpc_motion_planner block is to set up the Nonlinear Optimal Control Problem (NOCP) in a form that polympc can solve. It is responsible for evaluating the state of constraints and defining the dynamics of the system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$.

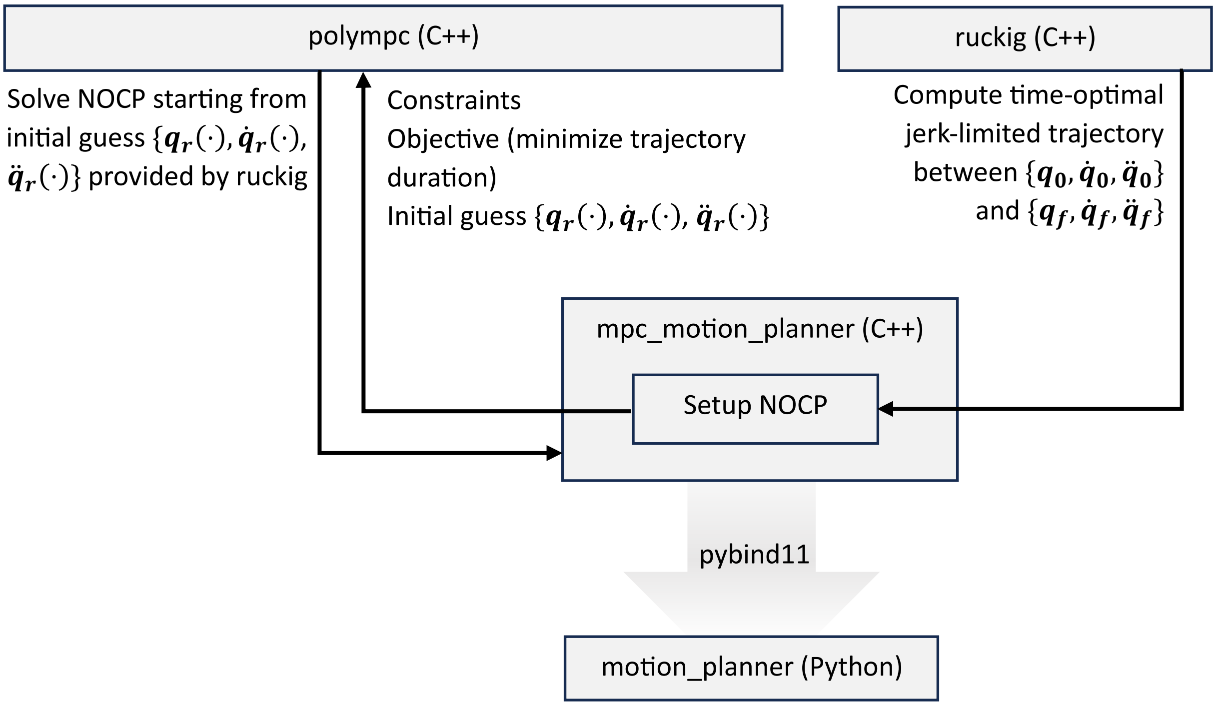

Figure 1 : Framework Architecture

4. Constraints Margin

For practical reasons, it is important to incorporate a safety margin on certain constraints. For example, for a state constraint $s_{min} \leq s \leq s_{max}$, new bounds will be calculated as $s_{min} \leq s'_{min} \leq s \leq s'_{max} \leq s_{max}$:

$$s'_{min} = \frac{s_{min} + s_{max}}{2} - \frac{(s_{max} - s_{min})}{2} \cdot \eta = s_{min} + \frac{(1-\eta)\cdot(s_{max} - s_{min})}{2}$$

$$s'_{max} = \frac{s_{min} + s_{max}}{2} + \frac{(s_{max} - s_{min})}{2} \cdot \eta = s_{max} - \frac{(1-\eta)\cdot(s_{max} - s_{min})}{2}$$

Note that if $s_{max} = -s_{min}$, then :

$$s'_{min} = s_{min} \cdot \eta$$

$$s'_{max} = s_{max} \cdot \eta$$

In the case of symmetrical constraints such as velocity, acceleration, or torque constraints. Please mind the notation, which might be a bit confusing, where η is called “margin” while it is usually defined in the literature as $(1-\eta)$.

5. Classes

The Python library is organized with two main classes:

a. Trajectory

A trajectory object stores mainly 5 vector values (time t, joint position $\boldsymbol{q}$, joint velocity $\dot{\boldsymbol{q}}$, joint acceleration $\ddot{\boldsymbol{q}}$, joint torque $\boldsymbol{\tau}$) and provides convenient methods to assess constraint satisfaction.

b. MotionPlanner

A MotionPlanner object is used to interface with the C++ class that corresponds to the desired robot (Panda, Kuka-iiwa7, Kuka-iiwa14). It provides methods to set the constraint limits and boundary conditions ($\{\boldsymbol{q_0}, \dot{\boldsymbol{q}}_{\boldsymbol{0}}, \ddot{\boldsymbol{q}}_{\boldsymbol{0}}\}$ and $\{\boldsymbol{q_f}, \dot{\boldsymbol{q}}_{\boldsymbol{f}}, \ddot{\boldsymbol{q}}_{\boldsymbol{f}}\}$), solve a trajectory using ruckig and/or polympc and compute inverse / forward kinematics.

For more information about how to use these objects, please refer to the GitHub repository : mpc_motion_planner/pyMPC/howToUse.ipynb