MPC cheatsheet

December 17, 2018

1 Model Predictive Control

1.1 MPC setup

We deal with a **discrete** or discretized problem considered over a **N steps horizon**. Every step has a dt duration.

$$\min_{u_t, u_{t+1}, \ldots, u_{t+N-1}} \sum_{k=t}^{t+N-1} l(x_k, u_k)$$

$$\text{subj. to } \begin{cases} x_{k+1} = f(x_k, u_k) & k = t, \ldots, t+N-1 \\ u_k \in \mathcal{U} & k = t, \ldots, t+N-1 \\ x_k \in \mathcal{X} & k = t, \ldots, t+N-1 \\ x_t = x(t) & k = t, \ldots, t+N-1 \end{cases}$$

With:

- u a control command e.g. $u_k = [a_x, a_y]^T$ at time step k or $u_k = [a_k, \delta_k]^T$
- x_k a state vector e.g. $\mathbf{x}_k = [x_k, y_k, x'_k, y'_k]^T$ or $x_k = [x, y, \theta, v]^T$
- $x_{k+1} = f(x_k, u_k)$ usually corresponds to our vehicle dynamics model e.g. $x_{k+1} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \end{bmatrix} u_k = Ax_k + Bu_k = f(x_k, u_k)$
- l is a cost function we want to optimize: examples include jerk (to optimize comfort), difference with a planned trajectory, time to goal or a combination of objectives

We are solving $\min_{u_{t:t+N-1}} \sum_{k=t}^{t+N-1} l(x_k, u_k)$ and not $\min_{u_{[t:t+N-1]}} \int_t^{t+N} l(x(\tau), u(\tau) d\tau$!!! Soon or later you need to discretize anyways

The algorithm will run at every time steps:

- At time t:

  - Measure (or estimate) the current state
  - Find the optimal input sequence $U^* = \{u_t^*, u_{t+1}^*, u_{t+2}^*, \ldots, u_{t+N-1}^*\}$
  - Apply only $u(t) = u_t^*$ and discard $u_{t+1}^*, u_{t+2}^*, \ldots, u_{t+N-1}^*$
- Repeat the same procedure at time $t+1$
  - Even assuming perfect model and no disturbances: the predicted open-loop trajectories are DIFFERENT than the closed-loop trajectories

This algorithm has the following characteristics: multivariate, model based, nonlinear, constraints based, predictive. At each sampling time, starting at the current state, an open-loop optimal control problem is solved over a finite horizon

1.2 MPC requirements

In order to solve a MPC problem we need:

- A discrete-time model of the system
- A state observer
- To setup an Optimization Problem
- To solve an Optimization Problem (Matlab Optimization toolbox, NPSOL, cvxgen, C++ Ipopt/CppAd tools)
- Verifiy that the closed-loop system performs as desired (avoid infeasibility, unstability)
- Make sure it runs in real-time

1.3 MPC with Continuous Time models

Use e.g. Euler Discretization of Nonlinear Models. Given a CT model

$$\begin{cases} \frac{d}{dt}x(t) & = f(x(t), u(t), t) \\ y(t) & = h(x(t), u(t), t) \end{cases}$$

We approximate with finite differences:

- $\frac{d}{dt}x(t) \approx \frac{x(t+\Delta t)-x(t)}{\Delta t}$
- Δt is the sampling time

Use Euler discretization or better discretization approaches (cf matlab: help c2d) and check the performance of your model: data vs simulated.

2 Optimization

2.1 Convex optimization problems

- A set $\mathcal{S}$ is convex if $\lambda z_1 + (1-\lambda) z_2 \in \mathcal{S}$ for all $z_1, z_2 \in \mathcal{S}, \lambda \in [0,1]$
- A function $f : \mathcal{S} \to \mathbb{R}$ is convex if
  - $\mathcal{S}$ is convex
  - $f(\lambda z_1 + (1-\lambda) z_2) \leq \lambda f(z_1) + (1-\lambda) f(z_2)$ for all $z_1, z_2 \in \mathcal{S}, \lambda \in [0,1]$
- A function $f : \mathcal{S} \to \mathbb{R}$ is concave if $\mathcal{S}$ is convex and $-f$ is convex
- Some operations preserve convexity: intersection of convex sets, $\sum_{i=1}^{N} \alpha_i f_i$, $f(Ax+b)$ ...

An optimization problem is said to be convex if the cost function f is convex over a convex set. A fundamental property of convex optimization is that local optimizers are also global optimizers. It suffices to compute a local minimum to determine its global minimum. Non-convex problems can be transformed into convex problems through a change of variables and manipulations on cost and constraints.

2.2 Optimality conditions

$$\min_{z} f(z)$$

$$\text{such that } \begin{cases} g_i(z) \leq 0 & \text{for } i = 1, \ldots, m \\ h_i(z) = 0 & \text{for } i = 1, \ldots, p \\ z \in \mathcal{Z} \end{cases}$$

- In general, an analytical solution does not exist
- In general, global optimality is not guaranteed
- Solutions are usualy computed by **recursive algorithms** which start from an initial guess z_0 and at step k generate a point z_k such that $\{f(z_k)\}_{k=0,1,2,\ldots}$ converges to J^*
- **Gradient descent algorithms**
- For unconstrained optimization problems, optimality conditions are:
  - $\nabla f(z^*) = 0$ is a necessary condition for z^* to be a local minimizer
  - If $\nabla f(z^*) = 0$ and Hessian of $H_f(z^*)$ is positive definite then z^* is a local minimizer
  - If f is convex differentiable at z^* then z^* is a global minimizer iff $\nabla f(z^*) = 0$

- For constrained optimization problems, optimality KKT conditions are:
  - $\nabla f(z^*) + \sum_{i=1}^{m} u_i^* \nabla g_i(z^*) + \sum_{j=1}^{p} v_j^* \nabla h_i(z^*) = 0$
  - cf Lagrange Multipliers:

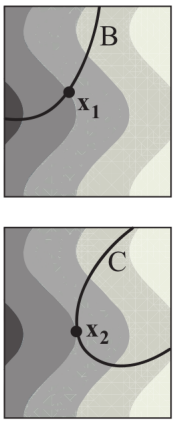
$$\min f(x,y) \text{ s.t. } g_1(x,y) = 0, \ldots, g_k(x,y) = 0$$

$$\text{solve } \nabla f(a) = \sum_{i=1}^{k} \lambda_i \nabla g_i(a)$$

2 eqs (partial derivatives) + k eqs g_i(x,y)=0 with 2+k unknowns $(x^*, y^*, \lambda_1, \ldots, \lambda_k)$

In the first picture, B crosses a level curve for f at the point $\mathbf{x_1}$. Moving along B to the right of $\mathbf{x_1}$ decreases the value of f and moving along B to the left of $\mathbf{x_1}$ increases the value of f. This shows that $\mathbf{x_1}$ is neither a local maximum nor a local minimum.

In the second picture, C is tangent to a level curve for f at $\mathbf{x_2}$. Moving away from $\mathbf{x_2}$ along C decreases the value of f, so $\mathbf{x_2}$ is a local maximum for the restriction of f to C. In this case, the gradient vector $\nabla f(\mathbf{x_2})$ is perpendicular to the tangent line to C at $\mathbf{x_2}$. Since $\nabla g_C(\mathbf{x_2})$ is also perpendicular to this tangent line, $\nabla f(\mathbf{x_2})$ and $\nabla g_C(\mathbf{x_2})$ are parallel.



  - $u_i^* g_i(z^*) = 0$
  - $u_i^* \geq 0$
  - $g_i(z^*) \leq 0$
  - $h_j(z^*) = 0$

2.3 Optimization algorithm: overall intuitive presentation

The algorithms are explained in an easy way here:

- Constrained Optimization explained to HighSchool with Calculus AB

What you need to know: computing Gradients, Jacobian, Hessian on simple functions, roots finding, following the gradient

Example Gradient James B. was located at $(1, 1)$. To choose his new location he calculates the gradient of the function $f(x, y) = x^2 + y^2 - 3xy + x$ at his current position and moves in the direction that is given by its calculated gradient at the point and distance traveled is the length of the gradient. Where he will find himself after two movements?

Example Jacobian Compute the Jacobian of $f : \mathbb{R}^2 \to \mathbb{R}^2$ with $y_1 = x_1 x_2, y_2 = x_1 + x_2$

Example Hessian Compute the Hessian of $f(x, y) = x^3 - 2xy - y^6$ at the point $(1, 2)$

2.4 Optimization tools

MPC examples in Python and Matlab

- NPSOL
- CVXGEN
- Matlab Optimization toolbox: fmincon, cvx
- C++ Ipopt with CppAd (CppAd performs automatic differentiation when gradient, Jacobian, Hessian are needed)