16.485: Visual Navigation for Autonomous Vehicles (VNAV) **Fall 2020**

Lecture 10: Trajectory Optimization II

Lecturer: Luca Carlone *Scribes: -*

These notes describe a Quadratic-Programming-based approach for trajectory optimization based on [1, 2].

10.1 QP-based Trajectory Optimization

In this method, the order of the polynomial is fixed and the goal is to find the polynomial coefficients by solving a Quadratic Programming (QP). If the r-th derivative has to be minimized, the polynomial should be of order at least $r+1$. However, the order of the polynomial is usually higher than $r+1$. For instance, in [1], the polynomial has order 9 for a minimum snap problem ($r=4$).

The goal is to convert the following trajectory optimization problem:

$$\begin{aligned} \underset{P(t)}{\text{minimize}} \quad & \int_0^T (P^{(r)}(t))^2 dt \\ \text{subject to} \quad & P(0) = \bar{x}_0 \\ & \vdots \\ & P^{(q)}(0) = x_0^{(q)} \\ & P(T) = \bar{x}_T \\ & \vdots \\ & P^{(q)}(T) = x_T^{(q)} \end{aligned} \tag{10.1}$$

to a Quadratic Programming (QP) problem in the form:

$$\begin{aligned} \min \quad & \boldsymbol{p}^\mathsf{T} \boldsymbol{Q} \boldsymbol{p} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{p} = \boldsymbol{b} \end{aligned} \tag{10.2}$$

where $\boldsymbol{p}$ is the vector the polynomial coefficients (as in Lecture 9), $\boldsymbol{A}$ is a known positive semi-definite matrix, and $\boldsymbol{A}\boldsymbol{p} = \boldsymbol{b}$ describes the trajectory constraints, including its derivatives. The rationale behind rewriting (10.1) as a QP, is that QP can be solved efficiently using standard libraries.

10.1.1 From Trajectory Optimization To QP

The main goal of this section is to provide intuition on how to rewrite (10.1) as a QP. Towards this goal, we consider two basic operations on polynomials: the computation of the derivatives and the square of a polynomial $P(t) = p_N t^N + p_{N-1} t^{N-1} + ... + p_2 t^2 + p_1 t + p_0$.

Derivatives of order r.

$$\begin{aligned} P^{(0)}(t) = P(t) = &\quad p_N t^N + p_{N-1} t^{N-1} + ... + p_2 t^2 + p_1 t + p_0 \\ P^{(1)}(t) = &\quad N p_N t^{N-1} + (N-1) p_{N-1} t^{N-2} + ... + 2 p_2 t + p_1 \\ P^{(2)}(t) = &\quad N(N-1) p_N t^{N-2} + (N-1)(N-2) p_{N-1} t^{N-3} + .. + 2 p_2 \\ \vdots & \\ P^{(r)}(t) = &\quad \textstyle\sum_{i=r}^{N} [\prod_{m=0}^{r-1} (i-m)] p_i t^{i-r} \end{aligned} \tag{10.3}$$

Square of a polynomial.

$$\begin{aligned} & P(t)^2 = \textstyle\sum_{i=0}^{2N} (\sum_{j=0}^{N} p_j p_{i-j}) t^i \\ & \\ & \text{example}: P(t) = p_1 t + p_0 \\ & P(t)^2 = (p_1 t + p_0)((p_1 t + p_0) = p_0\, p_0 + p_0\, p_1 t + p_1\, p_0 t + p_1\, p_1\, t^2 \end{aligned} \tag{10.4}$$

Cost functional in (10.1). By combining (10.3) and (10.4):

$$\begin{aligned} J &= \int_0^T \sum_{i=0}^{2N} (\sum_{j=r}^{N} \prod_{m=0}^{r-1} (j-m)(i-j-m) p_j p_{i-j}) t^{i-2r} dt \\ J &= \sum_{i=0}^{2N} (\sum_{j=r}^{N} \prod_{m=0}^{r-1} (j-m)(i-j-m) p_j p_{i-j}) \frac{T^{i-2r+1}}{i-2r+1} \end{aligned} \tag{10.5}$$

The terms $\prod_{m=0}^{r-1}(j-m)(i-j-m)$ and $\frac{T^{i-2r+1}}{i-2r+1}$ are independent of the coefficients. Our goal is to reformulate J as follows:

$$J = \boldsymbol{p}^\mathsf{T} \boldsymbol{Q} \boldsymbol{p} = \sum_{l=0}^{N} \sum_{k=0}^{N} Q_{lk} p_l p_k \tag{10.6}$$

In order to extract the elements of the matrix $\boldsymbol{Q}$, we use two times partial derivatives with respect to p_l and p_k $(0 \le l, k \le N)$ as follows:

$$\frac{\partial J}{\partial p_l} = \sum_{i=0}^{2N} \sum_{j=0}^{N} \prod_{m=0}^{r-1} (j-m)(i-j-m) (\frac{\partial p_j}{\partial p_l} p_{i-j} + \frac{\partial p_{i-j}}{\partial p_l} p_j) \frac{T^{i-2r+1}}{i-2r+1} = 2 \sum_{i=0}^{2N} \prod_{m=0}^{r-1} (l-m)(i-l-m) p_{i-l} \frac{T^{i-2r+1}}{i-2r+1} \tag{10.7}$$

Note: the summation $\sum_{j=0}^{N}(\cdot)$ is simplified to only two terms $j = l$ and $i - j = l$, where $0 \le l \le N$

$$\frac{\partial J}{\partial p_l \partial p_k} = 2 \sum_{i=0}^{2N} \prod_{m=0}^{r-1} (l-m)(i-l-m) \frac{\partial p_{i-l}}{\partial p_k} \frac{T^{i-2r+1}}{i-2r+1} = 2 \prod_{m=0}^{r-1} (l-m)(k-m) \frac{T^{l+k-2r+1}}{l+k-2r+1} \tag{10.8}$$

Note: the summation $\sum_{i=0}^{2N}(\cdot)$ is simplified to only one term $i - l = k$ where $0 \le k \le N$.

Comparing (10.8) with (10.6), the elements of the matrix $\boldsymbol{Q}$ can be written as:

$$Q_{lk} = \begin{cases} \prod_{m=0}^{r-1} (l-m)(k-m) \frac{T^{l+k-2r+1}}{l+k-2r+1} & l \ge r \wedge k \ge r \\ 0 & l < r \vee k < r \end{cases}, \quad \text{and } Q_{kl} = Q_{lk} \tag{10.9}$$

Therefore we reformulated the cost as a quadratic function of the polynomial coefficients.

Constraints (single segment). We have two sets of constraints involving the two endpoints and their derivatives. The derivatives are usually set to zero at the endpoints. It is possible to show that the constraints can be written as a linear function of the polynomial coefficients:

$$\boldsymbol{A}\boldsymbol{p} - \boldsymbol{b} = 0 \tag{10.10}$$

where, $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_T \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_T \end{bmatrix}$, where $\boldsymbol{A}_0, \boldsymbol{b}_0$ define the constraints induced by the starting point, while $\boldsymbol{A}_T, \boldsymbol{b}_T$ define the constraints induced by the ending point. For more details to how to construct this constraint equation, see [1].

10.1.2 Trajectory with waypoints

As mentioned in the previous lecture, it is common to consider trajectories including multiple segments, where each segment is assigned a polynomial trajectory. We then optimize the whole trajectory by jointly optimizing all the polynomial segments. For a trajectory with k segments, we form a long vector $\boldsymbol{p}$ by concatenating the coefficients at each segment $[\boldsymbol{p}_1 \quad \boldsymbol{p}_2 \quad ...\boldsymbol{p}_k]$ and $\boldsymbol{Q}$ is a block-diagonal matrix describing the joint cost including all segments $i = 1, ..., k$.

Similar to the discussion in Lecture 9, we must also include constraints that enforce the continuity (of the trajectory and its derivatives) at the waypoints. These constraints that can be written as:

$$[\boldsymbol{A}_\tau^i \quad -\boldsymbol{A}_0^{i+1}] \begin{bmatrix} \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \end{bmatrix} = 0 \tag{10.11}$$

10.1.3 Further considerations

10.1.3.1 Unconstrained QP

The method we described in this lecture is a constrained QP [2], in which the derivatives in each constraints are given. This method works well for polynomials with low degree and a small number of waypoints. For higher-order polynomials and problems with many waypoints, this formulation becomes ill-conditioned and numerically unstable.

To address this limitation, [1] converts the QP (10.2) to an unconstrained QP where one optimizes for the free endpoints derivatives instead of the polynomial coefficients. Once the optimal waypoint derivatives are found, the coefficients of the polynomial corresponding to each segments can be computed [1]. Such an unconstrained formulation can be accurately solved in milliseconds.

10.1.3.2 Time Allocation

Fixing the time for each segment may not be efficient in practice. For instance, if the duration T is too large, the drone flies very slowly. On the other hand, the drone control may be saturated for a small T. In the latter case, the desired command cannot be executed and the drone fails to follow the desired trajectory.

Intuitively, we can tune T such that, if it is too large, we make it smaller and continue until we reach the desired performance. It is possible to automate finding a suitable T by adding this parameter to the cost function:

$$J = J_r + \sum_1^k c_k T_k \tag{10.12}$$

where, J_r is the usual cost function (minimizing the r-th derivative), and T_k is the time associated with the segment k. We can optimize the time by using a gradient decent algorithm [1].

10.1.3.3 Feasibility of the trajectory

After the trajectory is generated, it is reasonable to check that it does not collide with any obstacle. Indeed, although the waypoints are obstacle free by construction (they are generated by a path planning algorithm), the trajectory can collide with obstacle.

If a collection is detected, we can introduce an intermediate (collision-free) waypoint and solve the polynomial optimization again. The new waypoint is sampled along the path produced by a path planning algorithm, hence it is guaranteed to be obstacle free. This process continues until the trajectory is obstacle free.

References

[1] A. Bry, C. Richter, A. Bachrach, and N. Roy. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *Intl. J. of Robotics Research*, 37(7):969–1002, 2015.

[2] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2520–2525, 2011.