U09M11109: Visual Navigation for Unmanned Aerial Vehicles (VisNav) **Spring 2023**

Lecture 4: Trajectory Optimization I

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor(s). Notes based on the draft produced by Golnaz Habibi during VNAV'18.*

These notes describe a polynomial method for trajectory optimization. In particular, our goal is to generate a trajectory which is smooth enough to be traversable by a quadrotor. For this purpose we will introduce an optimal control problem and solve it using the Euler-Lagrange equation. Then, in the next lecture, we will use a quadratic programming approach.

In the *single segment* case, the goal is to compute a trajectory from a point A to a point B. In the *multi segment* case, we have intermediate waypoints, and we want to compute a trajectory passing through all of them. The latter case is more common in practice, where one would use a path planner (e.g., RRT*) to obtain a set of waypoints, and then compute a smoother *trajectory* (as opposed to a *path*) using polynomial trajectory optimization.

4.1 Trajectory Optimization for Quadrotors

4.1.1 Optimal control basics

Optimal control tries to find input functions $\boldsymbol{u}(t)$ and the corresponding state trajectories $\boldsymbol{x}(t)$ that maximize performance (or minimize a cost) under constraints. As example of optimal control problem is given below:

$$\begin{aligned} \min_{\boldsymbol{x}(t),\boldsymbol{u}(t)} \quad & J(\boldsymbol{u}(t),\boldsymbol{x}(t)) \\ \text{subject to} \quad & \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t)) \\ & \boldsymbol{x}(0) = \bar{\boldsymbol{x}}_0 \\ & \boldsymbol{x}(T) = \bar{\boldsymbol{x}}_T \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{max} \\ & \boldsymbol{x}_{min} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{max} \end{aligned} \tag{4.1}$$

where J is the *functional* (e.g., quantifying energy, path length, input usage) we want to minimize, $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$ enforces the dynamics of the system, $\bar{\boldsymbol{x}}_0$ and $\bar{\boldsymbol{x}}_T$ represent given initial and final states, and $\boldsymbol{u}_{min}$, $\boldsymbol{u}_{max}$, $\boldsymbol{x}_{min}$, $\boldsymbol{x}_{max}$ are given input and state bounds. In general, the decision variables ($\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$) are continuous functions and finding a solution to (4.1) is challenging.

Some terminology. The so-called *indirect* methods reformulate (4.1) as a two-point boundary problem using Pontryagin's maximum principle. Boundary problems are difficult to solve in general. More recently, the fast-paced advances in nonlinear programming (NLP) promoted the use of *direct* methods, where (4.1) is first discretized (or, in general, rewritten using a finite parametrization), converted to a nonlinear program, and then solved using NLP techniques. In *shooting* methods, we optimize over control command only and eliminate the states using the system dynamics (intuitively: the trajectory of $\boldsymbol{x}(t)$ is fixed for a given initial state and choice of control). In *collocation* methods, we jointly optimize over controls and states as in (4.1).

This formulation typically makes it easier to enforce constraints over the states (obstacles, waypoints, etc.). We refer the interested reader to [2] for more details.

In the following, we will use a direct method and use differential flatness to avoid optimizing over states and controls.

9.1.2 Differential flatness

Our initial objective is to generate a smooth trajectory to allow the drone to fly from an initial to a final point. In lectures 3, we have shown the drone dynamics are differentially flat, which means there is a flat output $\boldsymbol{\sigma}(t) = [x, y, z, \psi]^\mathsf{T}$ such that the state and the input can be defined as smooth functions of this flat output and its derivatives, i.e., $\boldsymbol{x}(t) = h_x(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, \ldots)$ and $\boldsymbol{u} = h_u(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, \ldots)$.

Therefore, for differentially flat systems, we can rewrite the objective in (4.1) as:

$$J(\boldsymbol{u}(t), \boldsymbol{x}(t)) = J(h_u(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, \ldots), h_x(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, \ldots)) = J(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, \ldots) \tag{4.2}$$

where, with slight abuse of notation, we also use $J(\cdot)$ to denote the cost expressed as a function of the flat output.

9.1.3 Minimum Snap Trajectory Optimization

Cost functional. In lectures 3, we have seen that the drone inputs can be written in terms of the 4th derivative of the position of the drone (x, y, z) and the 2nd derivative of the drone yaw angle ψ. Therefore, to obtain a smooth trajectory (i.e., one with low input usage), we can minimize the 4th derivative of the positions (or, equivalently, the 4th derivative of the first three entry of our flat output $\boldsymbol{\sigma}(t) = [x, y, z, \psi]^\mathsf{T}$) and the 2nd derivative of the yaw angle (or, equivalently, the 2nd derivative of the last entry of $\boldsymbol{\sigma}$):

$$J(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, \ldots) = \int_0^T (x^{(4)}(t))^2 dt + \int_0^T {y^{(4)}(t)}^2 dt + \int_0^T {z^{(4)}(t)}^2 dt + \int_0^T {\psi^{(2)}(t)}^2 dt \tag{4.3}$$

The previous expression represents the functional we want to minimize to obtain a smooth trajectory.

Constraints (single segment). Here, we are mainly concerned with computing trajectories from a point A (at time 0) to a point B (at the final time T). Therefore, we only consider the following set of *end-point* constraints:

$$\begin{aligned}
\boldsymbol{\sigma}(0) &= \boldsymbol{\sigma}_0 && \text{(initial state)} \\
\boldsymbol{\sigma}^{(1)}(0) &= \boldsymbol{\sigma}_0^{(1)} && \text{(first derivative at starting point)} \\
&\vdots \\
\boldsymbol{\sigma}^{(q)}(0) &= \boldsymbol{\sigma}_0^{(q)} && \text{(q-th derivative at starting point)} \\
\\
\boldsymbol{\sigma}(T) &= \boldsymbol{\sigma}_T && \text{(final state)} \\
\boldsymbol{\sigma}^{(1)}(T) &= \boldsymbol{\sigma}_T^{(1)} && \text{(first derivative at end point)} \\
&\vdots \\
\boldsymbol{\sigma}^{(q)}(T) &= \boldsymbol{\sigma}_T^{(q)} && \text{(q-th derivative at end point)}
\end{aligned} \tag{4.4}$$

Note that the systems dynamic is omitted in this formulation, since we implicitly incorporate the relation between inputs and state using flat outputs.

9.2 Polynomial Trajectory Optimization

In order to minimize (4.3) under constraints (4.4), we adopt a direct method and assume a (finite-dimensional) polynomial parametrization of the flat output $\boldsymbol{\sigma}$. Polynomial functions are good candidates to model smooth trajectories. Moreover, they are easy to differentiate, which will be handy when dealing with the high-order derivatives in (4.3) and (4.4).

Since the cost function is the sum of 4 independent terms (and the constraints can be split accordingly), we can optimize the problem with respect to the flat outputs x, y, z, and ϕ, separately. For the rest of the lecture, we focus on a single scalar decision variable $x(t)$ and we assume it to be a polynomial function, i.e., $x(t) = P(t)$, where $P(t) = p_N t^N + p_{N-1}t^{N-1} + ... + p_2 t^2 + p_1 t + p_0$ is a polynomial of order N, and $p_i, i = 0, .., N$ are the polynomial coefficients. Moreover, we phrase the problem as the generic minimization of the r-th derivative of $x(t)$, hence the formulation can be easily applied to minimize the 4th derivative of x, y, and z, or the second derivative of ψ.

Therefore, the single-segment trajectory optimization problem for a flat polynomial output $x(t) = P(t)$ becomes:

$$\begin{aligned} \underset{P(t)}{\text{minimize}} \quad & \int_0^T (P^{(r)}(t))^2 dt \\ \text{subject to} \quad & P(0) = x_0 \\ & \vdots \\ & P^{(q)}(0) = x_0^{(q)} \\ & P(T) = x_T \\ & \vdots \\ & P^{(q)}(T) = x_T^{(q)} \end{aligned} \tag{4.5}$$

Depending on the derivative r being minimized, the problem obtains different trajectories:

- $r = 1$ minimum velocity trajectory (shortest distance);
- $r = 2$ minimum acceleration trajectory;
- $r = 3$ minimum jerk trajectory;
- $r = 4$ minimum snap trajectory.

4.3 Euler-Lagrange-based Trajectory Optimization

In this method, the order of the polynomial is not fixed and can be found by solving (4.5). We use Euler-Lagrange differential equation to find the polynomial order. Then, the end-point constraints are used to find the polynomial coefficients [1, 3].

Euler-Lagrange equation. In calculus of variations, the Euler-Lagrange equation is a second-order partial differential equation. Its solutions are the functions that minimize/maximize a given functional $J(x, \dot{x}, t) = \int_0^T \mathcal{L}(x, \dot{x}, t)dt$:

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}} = 0 \tag{4.6}$$

Note: The Euler-Lagrange equation is a *necessary* condition for a stationary point of $J(x, \dot{x}, t)$.

The Euler-Lagrange equation also extends to the general case $J(x, \dot{x}, \ddot{x}, ..., x^{(r)}(t), t) = \int_0^T \mathcal{L}(x, \dot{x}, \ddot{x}, ..., x^{(r)}(t), t)dt$ –as in (4.5)– leading to the following general equation:

$$\sum_{k=0}^{r} (-1)^k \frac{d^k}{dt^k} \left(\frac{\partial \mathcal{L}}{\partial x^{(k)}} \right) = 0 \tag{4.7}$$

Example. Assuming a polynomial trajectory $x(t) = P(t)$, minimum snap trajectory optimization ($r = 4$ and $\mathcal{L} = (P^{(4)}(t))^2$) leads to the following Euler-Lagrange equation:

$$\frac{\partial \mathcal{L}}{\partial P} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{P}} \right) + \frac{d^2}{dt^2}\left(\frac{\partial \mathcal{L}}{\partial \ddot{P}} \right) - \frac{d^3}{dt^3}\left(\frac{\partial \mathcal{L}}{\partial \dddot{P}} \right) + \frac{d^4}{dt^4}\left(\frac{\partial \mathcal{L}}{\partial P^{(4)}} \right) = 0 \tag{4.8}$$

Since our $\mathcal{L} = (P^{(4)}(t))^2$ only depends on the 4-th derivative, all the terms $\frac{\partial \mathcal{L}}{\partial P^{(i)}}$ are zero for $i \neq r$. Therefore, the previous expression simplifies to:

$$\frac{d^4}{dt^4}\left(\frac{\partial \mathcal{L}}{\partial P^{(4)}} \right) = 0 \Rightarrow \frac{d^4}{dt^4}\left(2P^{(4)}(t) \right) = 0 \Rightarrow \frac{d^8}{dt^8} P(t) = 0 \tag{4.9}$$

The solution of differential equation above is nothing but a polynomial with order of $N = 7$

$$P(t) = p_7 t^7 + p_6 t^6 + p_5 t^5 + p_4 t^4 + ... + p_1 t^1 + p_0 \tag{4.10}$$

To find the coefficients $p_i, i = 0, ..., 7$, the constraints in (9.5) can be written as linear equations of coefficients:

$$\mathbf{A_{8\times8}}\, \boldsymbol{p} = \mathbf{b_{8\times1}} \tag{4.11}$$

Where, $\boldsymbol{p}$ is the vector of coefficients $\boldsymbol{p} = [p_0, p_1, ..., p_7]^\mathsf{T}$. For instance, for a trajectory with one segment (one start and one ending point), we can assume the drone to start and end at a specific location (2 linear constraints), and we can assume the drone is stationary at the start and end points and its velocity, and acceleration are zero (*i.e.* $P^{(i)}(t) = 0$, $i = 1, 2, 3$), resulting in 6 constraints (3 at the start, and 3 at the end point). This gives us a total of 8 constraints which allow finding all the polynomial coefficients.

General case. In general, for $\mathcal{L} = (P^{(r)}(t))^2$, the solution of Euler Lagrange equation is a polynomial with order $N = 2r - 1$, which –in the single-segment case– requires $2r$ constraints.

4.3.1 Trajectory with waypoints

In practice, it is convenient to break down long trajectories into multiple segments, where the end-point of each segment corresponds to a waypoint we want the robot to follow. We refer to this setup as *multi-segment* trajectory optimization. As in the previous section, we assume a polynomial parametrization for the trajectory in each segment. For instance, assume a trajectory with 3 segments:

$$P(t) = \begin{cases} P_1(t), & t_0 \leq t \leq t_1 \\ P_2(t), & t_1 \leq t \leq t_2 \\ P_3(t), & t_2 \leq t \leq t_3 \end{cases} \tag{4.12}$$

Then the cost functional becomes:

$$J = \int_{t_0}^{t_1} \left(P_1^{(r)}(t) \right)^2 dt + \int_{t_1}^{t_2} \left(P_2^{(r)}(t) \right)^2 dt + \int_{t_2}^{t_3} \left(P_3^{(r)}(t) \right)^2 dt \tag{4.13}$$

An example for the case of $r = 2$ (minimum acceleration) is given in Fig. 4.1.

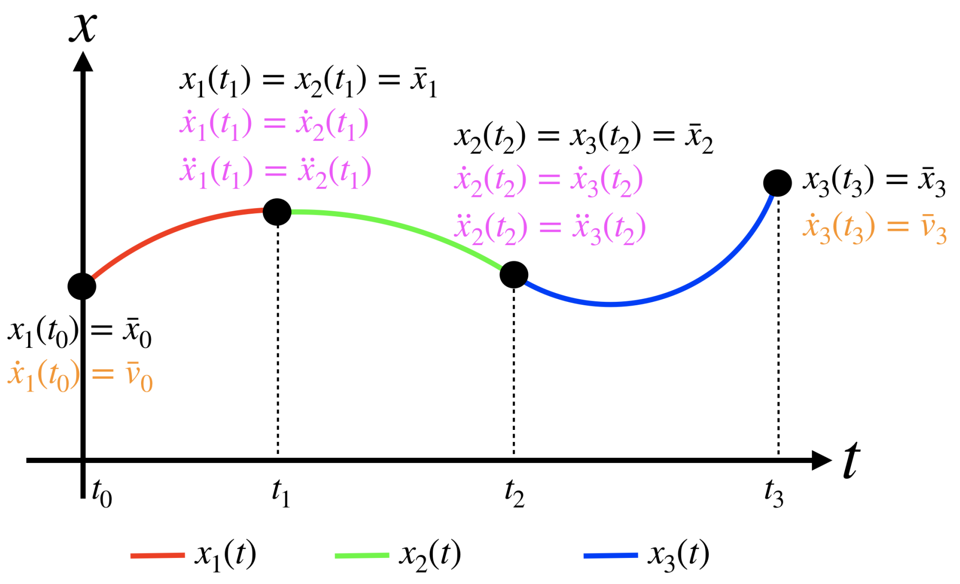
x
$x_1(t_1) = x_2(t_1) = \bar{x}_1$
$\dot{x}_1(t_1) = \dot{x}_2(t_1)$
$\ddot{x}_1(t_1) = \ddot{x}_2(t_1)$
$x_2(t_2) = x_3(t_2) = \bar{x}_2$
$\dot{x}_2(t_2) = \dot{x}_3(t_2)$
$\ddot{x}_2(t_2) = \ddot{x}_3(t_2)$
$x_3(t_3) = \bar{x}_3$
$\dot{x}_3(t_3) = \bar{v}_3$
$x_1(t_0) = \bar{x}_0$
$\dot{x}_1(t_0) = \bar{v}_0$
t
t_0
t_1
t_2
t_3
$x_1(t)$
$x_2(t)$
$x_3(t)$

Figure 4.1: Multi-segment minimum acceleration ($r = 2$) trajectory optimization with 3 segments.

If each polynomial has degree N, we now have $3(N+1)$ unknown coefficients we need to compute. Therefore, we need to ensure we have enough constraints to allow computing a unique choice of coefficients. We have 3 type of constraints:

- **waypoint constraints**: by design we want to constrain the polynomial to pass through the specified waypoints ($\bar{x}_0$, $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$ in Fig. 4.1). The corresponding constraints are shown in black in the figure (6 constraints in this example).
- **free derivatives**: in order to have a continuous and smooth trajectory, the trajectory and its derivatives should be continuous. For the trajectory with 3 segments, the continuity condition for the first and second derivative are shown in magenta in Fig. 4.1 (4 constraints in this example). In general, for a polynomial of order N, the continuity of derivatives up to $N-1$ can be enforced in the constraints.
- **fixed derivatives**: since the constraints introduced so far might not be enough to compute all the coefficients, we can introduce extra constraints on the derivatives at the initial point and the final point. For instance, we can enforce that the drone starts and ends with zero velocity. These constraints are shown in orange in Fig. 4.1 (2 constraints in this example)

References

[1] J. Brett. Practical methods for optimal control and estimation using nonlinear programming. *SIAM*, 2010.

[2] M. Kelly. An introduction to trajectory optimization: how to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[3] V. Kumar V, M. Zefran, and J. Ostrowski. *Motion planning and control of robots*. New York: Wiley, 1999.