Piecewise Polynomial Path Planner (P4)

Tucker Haydon

Spring 2019

1 Introduction

This document describes the underlying theory for the piecewise polynomial path planner.

2 Polynomial Theory

2.1 Problem Statement

Given a set of waypoints and corresponding arrival times, construct a piecewise-continuous polynomial trajectory through these waypoints that minimizes the norm of a specified derivative. Furthermore, the trajectory — or any derivative of the trajectory — may be subject to linear inequality constraints.

Mathematically, the problem may be stated so: given a set of m waypoints

$$\mathbf{W} = \begin{bmatrix} \vec{w}_0 & \vec{w}_1 & \vec{w}_2 & \dots & \vec{w}_m \end{bmatrix} \in \mathcal{R}^a \tag{1}$$

and a set of times to arrive at these waypoints

$$\vec{T} = \begin{bmatrix} t_0 & t_1 & t_2 & \dots & t_m \end{bmatrix} \tag{2}$$

determine a set of n'th-order piecewise-continuous polynomials that smoothly connect these waypoints

$$\vec{p}(t) = \begin{cases} p_0(t) = \sum_{k=0}^{n} p_{0,k} \cdot (\frac{1}{k!} t^k) & t_0 \leq t < t_1 \\ p_1(t) = \sum_{k=0}^{n} p_{1,k} \cdot (\frac{1}{k!} t^k) & t_1 \leq t < t_2 \\ \vdots & \\ p_m(t) = \sum_{k=0}^{n} p_{m,k} \cdot (\frac{1}{k!} t^k) & t_{m-1} \leq t < t_m \end{cases} \tag{3}$$

while minimizing the squared norm of the r'th derivative

$$\min \int_{t_0}^{t_m} \frac{d^r}{dt^r} ||\vec{p}(t)||^2 dt \tag{4}$$

subject to continuity and path constrains (see sections **??** & **??**).

This problem construction and the following solution follow those of Mellinger and Kumar [1].

2.2 Polynomial Basis

Trajectories for quadcopters may be specified with no more than four independent dimensions: x, y, z, and yaw. Any more dimensions (roll, pitch) introduce constraints that may render certain specified trajectories infeasible. For example, a quadcopter cannot be simultaneously stationary and have a non-zero roll angle [1].

Importantly, the x, y, z, and yaw dimensions may be specified *independently*: each dimension of the polynomial basis functions may be considered alone, permitting scalar analysis. The following section uses a 3rd-order polynomial, but the analysis generalizes to any order polynomial.

Consider a 3rd-order polynomial representing the first section of trajectory.

$$p_0(t) = \frac{1}{0!}t^0 x_0 + \frac{1}{1!}t^1 v_0 + \frac{1}{2!}t^2 a_0 + \frac{1}{3!}t^3 j_0$$

This expression may be divided into two components: a vector of coefficients and a vector of basis functions.

$$p_0(t) = \underbrace{\begin{bmatrix} x_0 & v_0 & a_0 & j_0 \end{bmatrix}}_{\text{coefficients}} \underbrace{\begin{bmatrix} \frac{t^0}{0!} \\ \frac{t^1}{1!} \\ \frac{t^2}{2!} \\ \frac{t^3}{3!} \end{bmatrix}}_{\text{basis functions}}$$

Or in vector form

$$p_0(t) = \vec{x}^T \vec{t}$$

2.3 Polynomial Optimization

For each section in the trajectory, the r'th derivative must be minimized.

$$\begin{aligned} x_i &= \arg\min_{\vec{x}} \int_{t_i}^{t_{i+1}} \left[\frac{d^r p(\tau)}{d\tau^r}\right]^2 dt \\ &= \arg\min_{\vec{x}} \int_{t_i}^{t_{i+1}} \left[\frac{d^r p(\tau)}{d\tau^r}\right]^T \left[\frac{d^r p(\tau)}{d\tau^r}\right] dt \\ &= \arg\min_{\vec{x}} \int_{t_i}^{t_{i+1}} \vec{x}^T \frac{d^r \vec{t}(\tau)}{d\tau^r}^T \frac{d^r \vec{t}(\tau)}{d\tau^r} \vec{x} dt \\ &= \arg\min_{\vec{x}} \vec{x}^T \int_{t_i}^{t_{i+1}} \frac{d^r \vec{t}(\tau)}{d\tau^r}^T \frac{d^r \vec{t}(\tau)}{d\tau^r} dt \vec{x} \end{aligned}$$

By defining

$$\mathbf{Q} = \int_{t_i}^{t_{i+1}} \frac{d^r \vec{t}(\tau)}{d\tau^r}^T \frac{d^r \vec{t}(\tau)}{d\tau^r} dt$$

and noting that the time-varying components will be integrated out by the definite integral, the optimization problem may be written as

$$x_i = \arg\min_{\vec{x}} \vec{x}^T \mathbf{Q} \vec{x}$$

The structure of the integrand of $\mathbf{Q}$ is revealed by first computing the initial couple derivatives of $\vec{t}(\tau)$.

$$\begin{aligned}
t(\vec{\tau}) &= \begin{bmatrix} \frac{\tau^0}{0!} & \frac{\tau^1}{1!} & \frac{\tau^2}{2!} & \dots \end{bmatrix} \\
\frac{dt(\vec{\tau})}{d\tau} &= \begin{bmatrix} 0 & \frac{\tau^0}{0!} & \frac{\tau^1}{1!} & \dots \end{bmatrix} \\
\frac{d^2t(\vec{\tau})}{d\tau^2} &= \begin{bmatrix} 0 & 0 & \frac{\tau^0}{0!} & \dots \end{bmatrix}
\end{aligned}$$

Notice that successive derivatives pre-pad zeros and shift the vector to the right.

The integrand of $\mathbf{Q}$ is found by taking the outer product of the derivative of the time vector with itself.

$$\begin{aligned}
t(\vec{\tau})^T t(\vec{\tau}) &= \begin{bmatrix} \frac{t^0}{0!}\frac{t^0}{0!} & \frac{t^0}{0!}\frac{t^1}{1!} & \frac{t^0}{0!}\frac{t^2}{2!} & \dots \\ \frac{t^1}{1!}\frac{t^0}{0!} & \frac{t^1}{1!}\frac{t^1}{1!} & \frac{t^1}{1!}\frac{t^2}{2!} & \dots \\ \frac{t^2}{2!}\frac{t^0}{0!} & \frac{t^2}{2!}\frac{t^1}{1!} & \frac{t^2}{2!}\frac{t^2}{2!} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \\
\frac{dt(\vec{\tau})}{d\tau}^T \frac{dt(\vec{\tau})}{d\tau} &= \begin{bmatrix} 0 & 0 & 0 & \dots \\ 0 & \frac{t^0}{0!}\frac{t^0}{0!} & \frac{t^0}{0!}\frac{t^1}{1!} & \dots \\ 0 & \frac{t^1}{1!}\frac{t^0}{0!} & \frac{t^1}{1!}\frac{t^1}{1!} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \\
\frac{d^2t(\vec{\tau})}{d\tau^2}^T \frac{d^2t(\vec{\tau})}{d\tau^2} &= \begin{bmatrix} 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & \dots \\ 0 & 0 & \frac{t^0}{0!}\frac{t^0}{0!} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}
\end{aligned}$$

Notice that the outer product of successive derivatives pre-pads columns and rows with zeros and shift rows down and columns to the right. This admits a convenient way to compute the integrand of $\mathbf{Q}$: take the outer product of $\vec{t}(\tau)$ with itself and shift the rows and columns by the power of the derivative.

$$\frac{d^r t(\vec{\tau})}{d\tau^r}^T \frac{d^r t(\vec{\tau})}{d\tau^r} = \text{shift}(\vec{t}(\tau)^T \vec{t}(\tau), r)$$

To compute $\mathbf{Q}$, the matrix integrand must be integrated. Integrating a matrix is as easy as integrating its components element-wise. The polynomial structure of the components of $\mathbf{Q}$ admit and easy-to-compute integral formula:

$$\int \frac{\tau^i}{i!}\frac{\tau^j}{j!}d\tau = \frac{\tau^{i+j+1}}{i!j!(i+j+1)!}$$

Finally, a careful observer might note that the structure of problem permits one to swap the shift and integrate operations. Instead of shifting all elements and then integrating, one could integrate all elements and then shift. This simplifies implementation in a for-loop-based subroutine.

In summary, to compute $\mathbf{Q}$:

1. Compute the outer product of $\vec{t}(\tau)$ with itself
2. Integrate the outer product
3. Shift the rows and columns of the integrated matrix by the order of the derivative

2.4 Relaxing Arrival Time

Until this point, one very important topic has been overlooked: timing. Users may specify arbitrary arrival times for waypoints. When times are too large, too small, or a combination of the both, numerical accuracy problems occur and many optimizers struggle to find sufficiently accurate solutions. To prevent this from occurring, segment time is normalized to $t \in [0, 1]$. Stated otherwise, every segment in a trajectory is assumed to start at $t = 0$ and end at $t = 1$.

This new timing scheme is referred to as the *computational time* while the original timing scheme is referred to as the *physical time*. Transforming the problem from physical time to computational time requires one to determine the segment index a particular time is, subtracting the start time, and normalized by the total segment time. The inverse transform is defined in a similar manner.

2.5 Time Scaling

Let $t \in [0, t_f]$ represent the physical time along a segment and $\tau \in [0, 1]$ represent the computational time along a segment. The two are related by a scaling factor.

$$t = \alpha\tau \tag{5}$$

Path constraints are specified using physical time, but the polynomial functions and optimization use computational time. These two timing schemes must be reconciled. Let a constraint be defined as:

$$\frac{dp(\tau)}{dt} < \kappa$$

Change of basis using $\tau = t/\alpha$ and $d\tau = dt/\alpha$. Substitute and rearrange:

$$\frac{dp(\tau)}{d\tau} < \alpha\kappa \tag{6}$$

Constraints on higher derivatives are derived using the same process.

$$\frac{dp^n(\tau)}{dt^n} < \kappa \rightarrow \frac{dp^n(\tau)}{d\tau^n} < \alpha^n\kappa \tag{7}$$

Continuity constraints (discussed in the next section) merit special consideration. These constraints require that the nth derivative of the end of a segment match the nth derivative of the following segment. Since the time scaling of each segment may be different, the constraint manifests so:

$$\frac{dp_k^n(\tau)}{d\tau^n}\frac{1}{\alpha_k^n} = \frac{dp_{k+1}^n(\tau)}{d\tau^n}\frac{1}{\alpha_{k+1}^n} \tag{8}$$

2.6 Continuity Constraints

Most piecewise-continuous problems require some sort of continuity — the position, velocity, or acceleration must not instantaneously change at endpoints of the piecewise trajectory. Typically this is motivated by real-life constraints: a quadcopter cannot instantaneously teleport or change its velocity from 1 m/s to -3 m/s.

Consider the first two segments of a scaler third-order piecewise polynomial trajectory. Let the optimization vector contain the coefficients for both segments of the trajectory.

$$\vec{x} = \begin{bmatrix} x_0 & v_0 & a_0 & j_0 & x_1 & v_1 & a_1 & j_1 \end{bmatrix}^T$$

The new quadratic matrix is a block-diagonal of the segment quadratic matrices.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_0 & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_1 \end{bmatrix}$$

The new optimization function is:

$$J = \vec{x}^T\mathbf{Q}\vec{x}$$

For the function to be continuous, the first segment must be equal to the second segment at the boundary point.

$$p_0(t=1) - p_1(t=0) = 0$$

Written in matrix form:

$$\underbrace{\begin{bmatrix} 1/0! & 1/1! & 1/2! & 1/3! & -1/0! & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_0 \\ v_0 \\ a_0 \\ j_0 \\ x_1 \\ v_1 \\ a_1 \\ j_1 \end{bmatrix}}_{\vec{x}} = \vec{0}$$

Or equivalently

$$\vec{l} \leq \mathbf{A}\vec{x} \leq \vec{u}$$

where both $\vec{l}$ and $\vec{u}$ are $\vec{0}$ in this case. Continuity constraints for arbitrary derivatives may be specified in a similar manner by leveraging the fact that the derivative of $\vec{t}$ is a shifted version of itself. The derivatives may be formulated in the same $\vec{l} \leq \mathbf{A}\vec{x} \leq \vec{u}$. Without further justification, all continuity and waypoint constraints may be formulated as bounded linear constraints.

2.7 Path Constraints

Path constraints are necessary for any non-trivial path. For example, quadcopters often have input constraints and can't fly faster than or accelerate faster than a certain limit. Alternatively, if obstacles are present, the position of the path should be constrained to be outside of obstacles.

In coordination with the continuity constraints, path constraints can be formulated in the form:

$$\vec{l} \leq \mathbf{A}\vec{x} \leq \vec{u}$$

Unfortunately, these constraints are linear — they cannot be used to describe non-linear constraints. For example, if a quadcopter is limited to a total acceleration of 0.5 m/s^2, then the two-norm magnitude of its acceleration should be bounded. The two-norm magnitude is a non-linear constraint and cannot be expressed. However, one could approximate it by specifying an arbitrary number of planes tangent-to or inscribed-in the hypersphere that delimits the two-norm. It's not perfect, but it enables the quadratic programming solver to find a solution fast.

Finally, path constraints introduce one additional complication: violation of dimensional independence. Previously, I claimed that the dimensions are independent and thus can be considered standalone as scalar polynomials. In the presence of path constraints, this is not necessarily true. Two dimensions can be linked together by a path constraint. For example, consider approximating a two-norm magnitude by requiring:

$$\begin{aligned} x + y &< 1 \\ x - y &< 1 \\ -x + y &< 1 \\ -x - y &< 1 \end{aligned}$$

Clearly these two dimensions are now linked. Despite this dependency, the scalar analysis still holds. The dependency is managed by the quadratic programming solver.

One last note: if there were no interdimensional dependence, a multi-dimensional problem could be broken down into multiple scalar dimensional problems and solved simultaneously by multiple solvers distributed over multiple threads. However, it's usually the case that at least one path constraint introduces interdimensional dependence and this technique cannot be leveraged.

2.8 Quadratic Programming

The problem of finding these piecewise polynomials can be cast as a quadratic programming problem:

$$\begin{aligned} \min_{\vec{x}} \quad & \frac{1}{2}\vec{x}^T\mathbf{Q}\vec{x} \\ \text{s.t.} \quad & \vec{l} \leq \mathbf{A}\vec{x} \leq \vec{u} \end{aligned} \tag{9}$$

OSQP is used to solve the QP problem [2].

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2520–2525.

[2] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *ArXiv e-prints*, Nov. 2017.