Smooth Autonomous Paths on Tank Drive using Splines

Lyra Solomon, Citrus Circuits

Spring 2018

1 Why Splines?

Getting from point to point efficiently is key to writing an effective autonomous routine. Before using splines, we designed our paths by sequencing semicircles. This has two primary issues. Most visibly, where each drive action ends up depends on where it started, so a change near the beginning of a routine means the entire rest of the routine has to be retuned. Less visibly, the transition between semicircles is imperfect because angular velocity is not continuous. This year, we began using splines, or two-dimensional polynomials, to define smooth curves between arbitrary states. This allowed us to do more in autonomous by getting around more efficiently and tuning faster.

2 Notation Used in this Paper

$a_{\perp b}$ The rejection of a onto b, $a - a_{\parallel b}$.

$a_{\parallel b}$ The projection of a onto b, $(a \cdot \hat{b})\hat{b}$.

$\hat{a}$ The unit vector in the direction of a, $\frac{a}{|a|}$.

$a^{\perp}$ The perpendicular of a, $\begin{bmatrix} -a_1 \\ a_0 \end{bmatrix}$.

A^{+} The pseudoinverse of A.

cap A function to ensure a value is in a given range, $\text{cap}(x, min, max) = \min(max, \max(min, x))$.

3 Definitions of Symbols

$d_0, d_f : \mathbb{R}$ The additional mostly straight initial and final distances, specified by the user to avoid obstacles or better handle nonzero ω_0 and ω_f.

$k : \mathbb{R}_+$ The approximate curvature of the path.

$k_r : \mathbb{R}_+$ The radius of the wheel base, half the distance between left and right wheels.

$r(s) : [0, 1] \to \mathbb{R}^2$ A curve which defines the shape of the path. s is an arbitrary variable that doesn't correspond exactly to time.

$\Delta r : \mathbb{R}^2$ The total displacement $r(1) - r(0)$.

$v_0, v_f : \mathbb{R}$ The initial and final velocity of the robot.

$\omega_0, \omega_f : \mathbb{R}$ The initial and final angular velocity of the robot.

$u_{cap} : \mathbb{R}_+$ The maximum voltage for feedforward control. Should be less than the battery voltage to allow feedback corrections.

$a_{cap} : \mathbb{R}_+$ The maximum acceleration. This is limited since reversing direction at full power would result in the robot tipping or skidding.

4 Paths

The path the robot must follow is defined by six constraints - the initial and final position, tangent (which closely corresponds to velocity), and normal (which closely corresponds to acceleration). Therefore, the path must be a fifth-degree polynomial since that is the minimum degree with six coefficients. Since $r(s) = a_0 + a_1 s + a_2 s^2 + a_3 s^3 + a_4 s^4 + a_5 s^5$, the constraints are $r(0) = a_0$, $r'(0) = a_1$, $r''(0) = 2a_2$, $r(1) = a_0 + a_1 + a_2 + a_3 + a_4 + a_5$, $r'(1) = a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5$, and $r''(1) = 2a_2 + 6a_3 + 12a_4 + 20a_5$. Solving for the coefficients in terms of the

constraints yields the following equation for $r(s)$:

$$\begin{aligned} r(s) =& r(0)+ \\ & r'(0)\cdot s+ \\ & \frac{1}{2}r''(0)\cdot s^2+ \\ & (-10r(0)-6r'(0)-\frac{3}{2}r''(0)+\frac{1}{2}r''(1)-4r'(1)+10r(1))\cdot s^3+ \\ & (15r(0)+8r'(0)+\frac{3}{2}r''(0)-r''(1)+7r'(1)-15r(1))\cdot s^4+ \\ & (-6r(0)-3r'(0)-\frac{1}{2}r''(0)+\frac{1}{2}r''(1)-3r'(1)+6r(1))\cdot s^5 \end{aligned} \tag{1}$$

The positions $r(0)$ and $r(1)$ are two-dimensional vectors representing the initial and final position of the robot, respectively.

The tangents $r'(0)$ and $r'(1)$ are used primarily to define the shape of the curve. They are two-dimensional vectors with direction defined by the robot's heading. Usually, their magnitude is set to the endpoint-to-endpoint length of the curve. This has the advantage of being very predictable, since scaling the endpoints simply scales the curve, and being very quick to drive if the robot starts and ends from rest. However, if the robot has a high enough initial or final velocity, the path will curve too quickly for the robot to follow. To fix this, we add an estimate of the extra stopping distance, $\frac{1}{2}v^2k^2$, where k is the estimated curvature.

$$k = \frac{2\cdot|\Delta r_{\perp r'(0)}| - \Delta r_{\perp r'(0)}\cdot\widehat{r'(1)}}{|\Delta r|} \tag{2}$$

This estimates the curvature by taking twice the sideways distance minus the amount of sideways distance accounted for by ending at a different angle, as a proportion of the total distance. An additional value is added by the user if necessary to extend one end of the curve to avoid an obstacle. Experimentally, it was found that to drive mostly straight for an additional distance d, approximately $5d$ should be added to the tangent. The complete equation for $r'(0)$ is as follows ($r'(1)$ works the same way):

$$r'(0) = (|\Delta r| + \frac{1}{2}v_0^2k^2 + 5d_0)\cdot\widehat{r'(0)} \tag{3}$$

The normals $r''(0)$ and $r''(1)$ are used to ensure that the angular angular velocity at the endpoints is correct. The equations $a = v^2/r$ and $r = v/\omega$ give the following equation for $r''(0)$ ($r''(1)$ works the same way):

$$r''(0) = r'(0)^{\perp}\cdot|r'(0)|\cdot\frac{\omega_0}{v_0} \tag{4}$$

5 Reparameterization

For the robot to follow the path, it must be given position and velocity as a function of time. First, the polynomial is broken into a finite list of samples

containing x, y, and heading at $s = 0, \frac{1}{n}, \ldots, \frac{n-1}{n}$. From this, the drivetrain state is calculated at each sample using the following equations:

$$\Delta x_{l,i} = \Delta r_i \cdot T_i - k_r \Delta \theta_i \tag{5}$$

$$\Delta x_{r,i} = \Delta r_i \cdot T_i + k_r \Delta \theta_i \tag{6}$$

where T is the unit tangent vector at the first sample and k_r is the radius of the wheel base.

Two passes are needed to find the point in time at which each sample occurs - one which steps forwards using the maximum acceleration, and one which steps backwards using the maximum deceleration.

The robot is assumed to follow the state-space equation $\dot{x} = Ax + Bu$. On each pass, the holding voltage $V_{holding} = -B^+ Ax$ is calculated at each sample. On the forwards pass, this is subtracted from the minimum and maximum voltage to give the amount of voltage available for acceleration. On the backwards pass, it is added instead, since the output state occurs before the input state.

$$u_{max} = \begin{bmatrix} u_{cap} \\ u_{cap} \end{bmatrix} - B^+ Ax * \text{timedirection} \tag{7}$$

$$u_{min} = \begin{bmatrix} -u_{cap} \\ -u_{cap} \end{bmatrix} - B^+ Ax * \text{timedirection} \tag{8}$$

For each side of the drivetrain, the maximum acceleration is calculated. In most cases, the segment can be approximated as an arc, in which the acceleration on each side is proportional to the segment length. However, this approximation fails when one side needs to travel much further than the other. In that case, when computing the acceleration for the shorter side, treat the other side as being shorter than it is. A reasonable cutoff is to cap the opposite side at three times the length. From the acceleration ratio, the voltage ratio is calculated to find the maximum.

$$u_{accel} = B^+ \begin{bmatrix} a_{same} \\ a_{opposite} \end{bmatrix} k \tag{9}$$

$u_{accel,max}$: select largest k such that no component of u_{accel} is greater than its matching component in u_{max}.
$u_{accel,min}$: select smallest k such that no component of u_{accel} is less than its matching component in u_{max}.

When the distance is forwards, select the maximum acceleration, when the distance is backwards, select the minimum acceleration. In either of these cases, $a = \text{cap}(Bu_{accel}, -a_{cap}, a_{cap})$, and $v_f = \sqrt{v_0^2 + 2ad}$, but with the sign set to that of the d.

In the case that the direction changes, take the next distance as well and assume the samples are evenly spaced in time, forming a quadratic equation where $x(0) = 0$, $x(1) = d_1$, and $x(2) = d_1 + d_2$, $x(s) = \frac{d_2 - d_1}{2} s^2 + ks$. If the total

time from $s = 0$ to $s = 2$ is T, $x(t) = 2\frac{d_2-d_1}{T^2}t^2 + kt$ and $a = 4\frac{d_2-d_1}{T^2}$. Selecting $a = a_{cap}$ and solving gives $T = 2\sqrt{\frac{d_2-d_1}{a}}$, and $v_f = v(\frac{T}{2}) = \frac{d_2+d_1}{T}$.

Due to approximations and capping, the two velocities may not agree. The time each side of the drivetrain requires is calculated as $t = \frac{2d}{v_0+v_f}$. The time required for the segment is $\max(t_l, t_r)$, unless one side crosses $v = 0$ during that segment, in which case the time from the other side is used to prevent possible floating point precision issues. The final velocity for each side is multiplied by $\frac{t_{side}}{t}$ so both sides are balanced. The balancing is done on the final velocity instead of the average velocity to prevent errors from passed on to the next segment.

6 Interpolation

The times that the robot needs to know the goal state are not the same as the sample times. Because of this, the goal state must be calculated from the nearest samples. We maintain the index of the segment the robot is currently following, and the time elapsed since the robot began that segment. When the time elapsed exceeds the segment time, the time elapsed is decreased by that amount and the index is incremented. The path is considered complete when the segment index is equal to the number of segments.

The goal position and velocity follow the equations $x = x_i + v_i t + \frac{1}{2}a^2$ and $v = v_i + at$, where x_i and v_i come from the sample at the start of the segment, t is the time elapsed in the current segment, and $a = \frac{v_{i+1}-v_i}{t_i}$, where v_{i+1} comes from the sample at the end of the segment and t_i is the segment time.

The goal angle $\theta = \omega t$, where $\omega = \frac{\theta_{i+1}-\theta_i}{t_i}$.

The goal Cartesian position has $p_x = p_{x,i} + d\frac{cos\theta+cos\theta_i}{2}$ and $p_y = p_{y,i} + d\frac{sin\theta+sin\theta_i}{2}$, where the distance traveled this segment $d = \frac{x_l+x_r-x_{l,i}-x_{r,i}}{2}$.

7 Implementation

Our implementation of splines is in the directory `/third_party/frc971/control_loops/paths/` of our robot code repository. The files `path.*` contain the implementation of the paths section, and the files `trajectory.*` contain the implementation of the reparameterization and interpolation sections.

When implementing or modifying spline code, important edge cases to test are driving backwards and turns sharp enough that one side needs to reverse direction.