Motion planning with TinySpline and OpenCV

Duncan Freeman

Q: Why do you even want to use splines anyway?
A: Because they're *cool*

Contents

1 Introduction/Why **2**

2 Libraries used **2**

3 Setting up the spline **2**
3.1 Background 2
3.2 Control Points 2

4 Generating paths **3**
4.1 Some notes about the notation 3
4.2 Wheel positions 3
4.3 Wheel velocities 4
4.4 Reversing 4
4.5 Incrementing over spline indices 4
4.6 Real velocities of each wheel 5
4.7 Accumulative distances 5

5 Special Thanks **5**

1 Introduction/Why

But seriously, we use splines because they give us an optimal path to a target without having to waste time stopping and turning. We turn *while* we move. This is important because it saves time during the autonomous period in FRC, as well as giving us more margin in teleop. This document details how we accomplish this.

2 Libraries used

Tiny spline is a library for calculating an n-dimensional paramatric function (spline) given a number of control points. Details about their software are available at https://tinyspline.org/ . We also use an open source computer vision library called OpenCV to find our goal, available at https://opencv.org

3 Setting up the spline

3.1 Background

TinySpline expects a set of control points to influence the shape and trajectory of the spline it creates. When the vision code detects the goal, it should return both it's relative position and it's orientation. The bot's position is always at the origin $(0, 0)$, while the goal is at some offset.

3.2 Control Points

Because our goals have both a position and a direction, we use a combination of 3 control points for each goal. The first control point, located exactly at our goal's location, determines where we want to end up. The second, located at an offset and from our goal's position based off of the goal's orientation, makes sure that the robot ends oriented at our goal. The third, located at *double* the offset distance from the goal's position, makes sure that we have a margin to finish turning at our desired location. If the location of the goal is (x, y), the direction of the goal θ (In radians), and the distance we want to move out from the goal d (usually the distance between the wheels of the bot), then the locations of the 3 control points necessary are as follows (in order):

(x, y)
$(\cos\theta * d, \sin\theta * d) + (x, y)$
$(\cos\theta * d * 2, \sin\theta * d * 2) + (x, y)$

Note: The terms above only detail the first 3 points of the spline, the next 3 are the similar but in reverse order and located around the *end* position and direction.

4 Generating paths

4.1 Some notes about the notation

TinySpline allows us access to index positions in n dimensions, the first two of which are $x(j)$ and $y(j)$. Note that j does not represent a real distance in space, but an arbitrary index over the spline as a whole.

When we express $y'(j)$, or $x'(j)$, we have to keep in mind that these are real distances over a change in index j where $j \in (0, 1)$:

$$y'(j) = \frac{dy}{dj} y(j) \text{ and } x'(j) = \frac{dx}{dj} x(j) \tag{1}$$

4.2 Wheel positions

To find the position of a wheel at index j along a spline, we first need to find the direction of the current point, given by $(x'(j), y'(j))$. We can then find the perpendicular to the vector by rotating it by 90°:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x'(j) \\ y'(j) \end{bmatrix} = \begin{bmatrix} -y'(j) \\ x'(j) \end{bmatrix} = \{-y'(j), x'(j)\} \tag{2}$$

We then normalize the point by dividing it by the length of the original vector:

$$\frac{\{-y'(j), x'(j)\}}{\sqrt{x'(j)^2 + y'(j)^2}} \tag{3}$$

Adding the vector to the position along the curve gives us the final position of the wheel:

$$\{x(j), y(j)\} \pm d\left(\frac{\{-y'(j), x'(j)\}}{\sqrt{x'(j)^2 + y'(j)^2}}\right) \tag{4}$$

Note: Either adding or subtracting the vector to the position gives us two distict posibilities; These are the left and right wheels, respectively.

4.3 Wheel velocities

To find the velocity of each wheel, we can derive the position of each like so:

$$\{x'(j), y'(j)\} \pm d\frac{\{-y''(j), x''(j)\} * \sqrt{x'(j)^2 + y'(j)^2} - \{-y'(j), x'(j)\} * \frac{y''(j)y'(j)+x''(j)x'(j)}{\sqrt{x'(j)^2+y'(j)^2}}}{x'(j)^2 + y'(j)^2} \tag{5}$$

4.4 Reversing

Because the rate of change in velocity must be positive for each motor, we have no intrinsic way of knowing if a given wheel should be in reverse to compensate for a tight corner. Luckily, there is a solution; we can find the rate of change in angle of center line, and measure if it's above/below 2π. This works because if the rate of change is above 2π we're making a turn sharper than if one motor was at 100% and the other at 0%. We can find the derivate of the change in angle like so:

$$\frac{d}{dj}\tan^{-1}\left(\frac{y'(j)}{x'(j)}\right) = \frac{1}{1 + \left(\frac{y'(j)}{x'(j)}\right)^2} * \frac{y''(j)x'(j) - x''(j)y'(j)}{x'(j)^2} \tag{6}$$

4.5 Incrementing over spline indices

To approximate how far along the spline we should move (in index units j) to reach our next time point, we first need to find the distance we predict we will travel for j units of change; this can be acquired with the lengths of the velocity vectors for both wheels, selecting the largest one to make sure that no wheel goes faster than the max velocity. This gives us $\frac{dr}{dj}$. We specify what $\frac{dr}{dt}$ is in code, manually, as well as dt.

Where r = position, t = time, and j is an index in the spline:
$\frac{dr}{dj}$ = Largest velocity of the two sides over dj

$\frac{dr}{dt}$ = Maximum velocity we allow

$\left(\frac{dr}{dj}\right)^{-1} * \frac{dr}{dt} = \frac{dj}{dt}$

$\frac{dj}{dt} * dt = dj$ required

4.6 Real velocities of each wheel

$< Unfinished >$

4.7 Accumulative distances

$< Unfinished >$

5 Special Thanks

1. To Andrew Jones and Grace Carroll, for helping me derive the length of the speed vector in equation 3

2. To Paul Buchanan, for helping me derive the inverse tangent in equation 6 and for putting up with my pesky questions.