Path planning from scratch

Team 955 (Duncan Freeman)

May 18, 2018

Contents

1 The problem **2**

2 Going from A to B (splines) **3**
2.1 Interpolation 3
2.2 Splines 4
2.3 Derivation 5

3 Path planning for *any* parametric **5**
3.1 Wheel positions 6
3.2 Wheel velocities 7
3.3 Reversing 7
3.4 Incrementing over spline indices 7

1 The problem

In FRC, we always have an autonomous period with an objective that requires us to move around. In many cases, the movement required is complex, must be generated on the fly (I.E. the field is different each time or the robot's position varies), and must have a precise result. There are a few ways to acheive this; Splines have the advantage of being easy to define, being fairly numerically stable, and relatively easy to generate.

Because we need such precise control, the Talon SRX (or it's derivatives) is a fine choice of motor controller, if not only for it's motion planning capabilities. Motion planning is essentially a method through which a PID loop can be controlled at a high sample rate (precision) using equiptment that may have high latency but adequate processing power. The Talon SRX usually receives commands for velocity or position control exactly at the moment they are required; When put in Motion Profile mode they allow the user to push a certain volume of closed loop commands and time intervals, and then they execute the result. Additionally, this buffer may be pushed to while the profile is in operation, allowing for extensive paths to be executed with extremely precise control. Usually, we just need profile outputs as velocities for left and right motors, as such:

t	V_{left}
0	2.3
1	2.1
2	2.9
3	3.4

t	V_{right}
0	-1.9
1	-2.1
2	-2.9
3	-1.9

Table 1: Motion profile example data

In conclusion: We want motion profile data, and we're going to create it in real time using splines.

2 Going from A to B (splines)

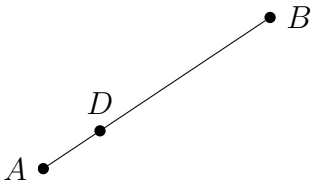
2.1 Interpolation

The basic element of path planning: We are at point A, we need to be at point B. Note that points A and B can be in any number of dimensions, but for the purpose of FRC this is mainly just 2 (x and y).

The simplest way to represent some sort of progression between A and B is through an *interpolation*. For some $0 \leq k \leq 1$, an interpolation between an A and B of any number of dimensions is:

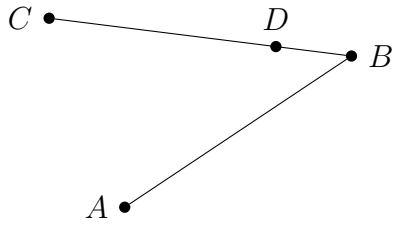
$$D = A(1 - k) + kB$$

where D is the point we are currently at. Note that k doesn't have to represent time, and in the case of path planning most likely won't.



$k = 0.25$

Now what if we needed to have multiple paths? What if we need to move in any other motion than a straight path? What if we needed to start at one direction and end in another? Well, some of this can solved using multiple interpolations.

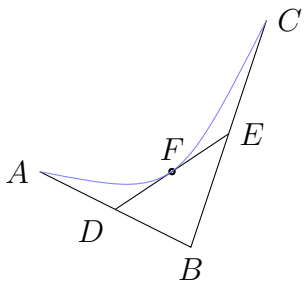


$k = 1.25$, $line = B \rightarrow C$

This *does* work, and some teams do use it. You need to turn at point B, but that's not hard; It's just less flexible than splines.

2.2 Splines

Let's start with a simple spline with only 3 control points (The simplest spline has 2, and it's actually the interpolation you've already learned!):

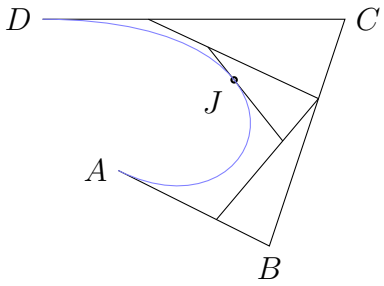


$$D = A(1-k) + kB$$
$$E = B(1-k) + kC$$
$$F = D(1-k) + kE$$

$$k = 0.5$$

In summary:
Go k interval from A to B and call the point D. Go k interval from B to C and call that point E. Do the same for D and E and call it F. As we increase k from 0 to 1, F moves along our spline (marked in blue).

3 control points is fine for some cases, but we want to be able to connect two splines end-to-end to easily form a larger spline with C_1 continuity (Meaning that there are no sudden turns that have to be handled individually. We can write a cubic spline with 4 control points like so:



$$k = 0.65$$

$$E = A(1-k) + kB$$
$$F = B(1-k) + kC$$
$$G = C(1-k) + kD$$

$$H = E(1-k) + kF$$
$$I = F(1-k) + kG$$

$$J = H(1-k) + kI$$

... phew! That was a lot.

2.3 Derivation

If we write the whole expression out and simplify, it becomes this:

$$J = (1-k)^3 A + 3(1-k)^2 kB + 3(1-k)k^2 C + k^3 D$$

It's somewhat uglier, but it's derivable. And when we do derive the expression, we get:

$$\frac{dJ}{dk} = 3(1-k)^2(A-B) + 6(1-k)k(C-B) + 3k^2(D-C)$$

... and the second derivative:

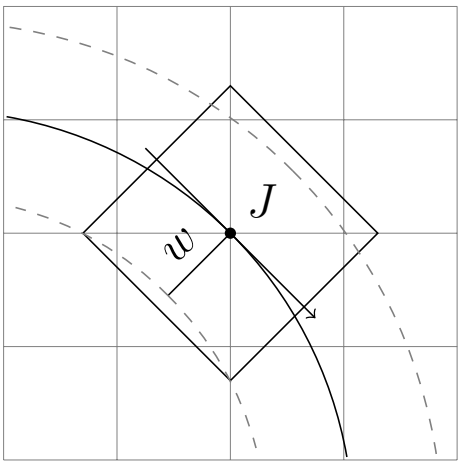
$$\frac{d^2J}{dk^2} = 6(1-k)(C-2B+A) + 6k(D-2C+B)$$

Now that we know how to create the curve and it's derivatives, it's time to create the path(s).

3 Path planning for *any* parametric

For this section, we will assume that the spline has been defined using 2 dimensional control points, and that $\frac{dx}{dk}$, $\frac{dy}{dk}$ are represented as x' and y'.

3.1 Wheel positions



As shown in the diagram above, the spline defines where the *center* of the robot will be. Our end goal is to find the rotational velocity of the individual wheels, so we need to figure out the position of each. w represents the distance from the center of the robot to each of the wheel wells that define our physical robot.

To find the position of a wheel at index k along the spline, we need to find the perpendicular unit vector to our current motion (x', y'). Mulitplying it by $\pm w$ and adding it to position J gives us the actual position of each wheel.

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

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

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

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

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

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

3.2 Wheel velocities

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

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

3.3 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:

$$\frac{d}{dk} \tan^{-1}\left(\frac{y'(k)}{x'(k)}\right) = \frac{1}{1 + \left(\frac{y'(k)}{x'(k)}\right)^2} * \frac{y''(k)x'(k) - x''(k)y'(k)}{x'(k)^2}$$

3.4 Incrementing over spline indices

To approximate how far along the spline we should move (in index units k) 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 $\frac{dr}{dk}$. 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:
$\frac{dr}{dk}$ = Largest velocity of the two sides over dk

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

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

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