Curve Generation

In order to generate closed-loop contours, the method used is to generate 5th order interpolated splines on both *x* and *y* values. Both *x* and *y* are treated as independent functions of "time" *t*:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$
$$y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5$$

x and *y* are now parametric functions of *t*.

The interesting property of a 5th order function is that we can calculate coefficients such that we can ensure that our function will have:

- A start (t = 0) and end ($t = t_f$) position
- A start (t = 0) and end ($t = t_f$) velocity
- A start (t = 0) and end ($t = t_f$) acceleration

In order to calculate these coefficients, the following equations are used **[Line 44 of the code]**:

$$\begin{aligned}
a_0 &= x_0 \\
a_1 &= \dot{x}_0 \\
a_2 &= \frac{\ddot{x}_0}{2} \\
a_3 &= \frac{20(x_f - x_0) - (8\dot{x}_f + 12\dot{x}_0)t_f - (3\ddot{x}_0 - \ddot{x}_f){t_f}^2}{2{t_f}^3} \\
a_4 &= \frac{30(x_0 - x_f) + (14\dot{x}_f + 16\dot{x}_0)t_f + (3\ddot{x}_0 - 2\ddot{x}_f){t_f}^2}{2{t_f}^4} \\
a_5 &= \frac{12(x_f - x_0) - 6(\dot{x}_f + \dot{x}_0)t_f - (\ddot{x}_0 - \ddot{x}_f){t_f}^2}{2{t_f}^5}
\end{aligned}$$

t_f is time at end point (Also known as STEP_SIZE in code)
x_0 is x position at t = 0; x_f is x position at $t = t_f$
$\dot{x}_0$ is x velocity at t = 0; $\dot{x}_f$ is x velocity at $t = t_f$
$\ddot{x}_0$ is x acceleration at t = 0; $\ddot{x}_f$ is x acceleration at $t = t_f$

Positions, velocities and accelerations are all generated randomly, and t_f (STEP_SIZE) is a value which can be set.

Take for example these randomly generated (except for t_f) parameters:

$$\begin{aligned}
t_f &= 10 \\
x &= [276, 50, 273] \\
\dot{x} &= [22, 173, -366] \\
\ddot{x} &= [-59, -52, 58]
\end{aligned}$$

For the first coefficients, we consider:

$$x_0 = 276;\ x_f = 50$$
$$\dot{x}_0 = 22;\ \dot{x}_f = 173$$
$$\ddot{x}_0 = -59;\ \ddot{x}_f = -52$$

Plugging these values into the coefficient equations, we get:

$$a_0 = 276;\ a_1 = 22;\ a_2 = -29.5;\ a_3 = -4.25;\ a_4 = 1.361;\ a_5 = -0.06856$$

Then, we continue with the next values:

$$x_0 = 50;\ x_f = 273$$
$$\dot{x}_0 = 173;\ \dot{x}_f = -366$$
$$\ddot{x}_0 = -52;\ \ddot{x}_f = 58$$

Which produces:

$$b_0 = 50;\ b_1 = 173;\ b_2 = -26;\ b_3 = 17.19;\ b_4 = -2.8725;\ b_5 = 0.12628$$

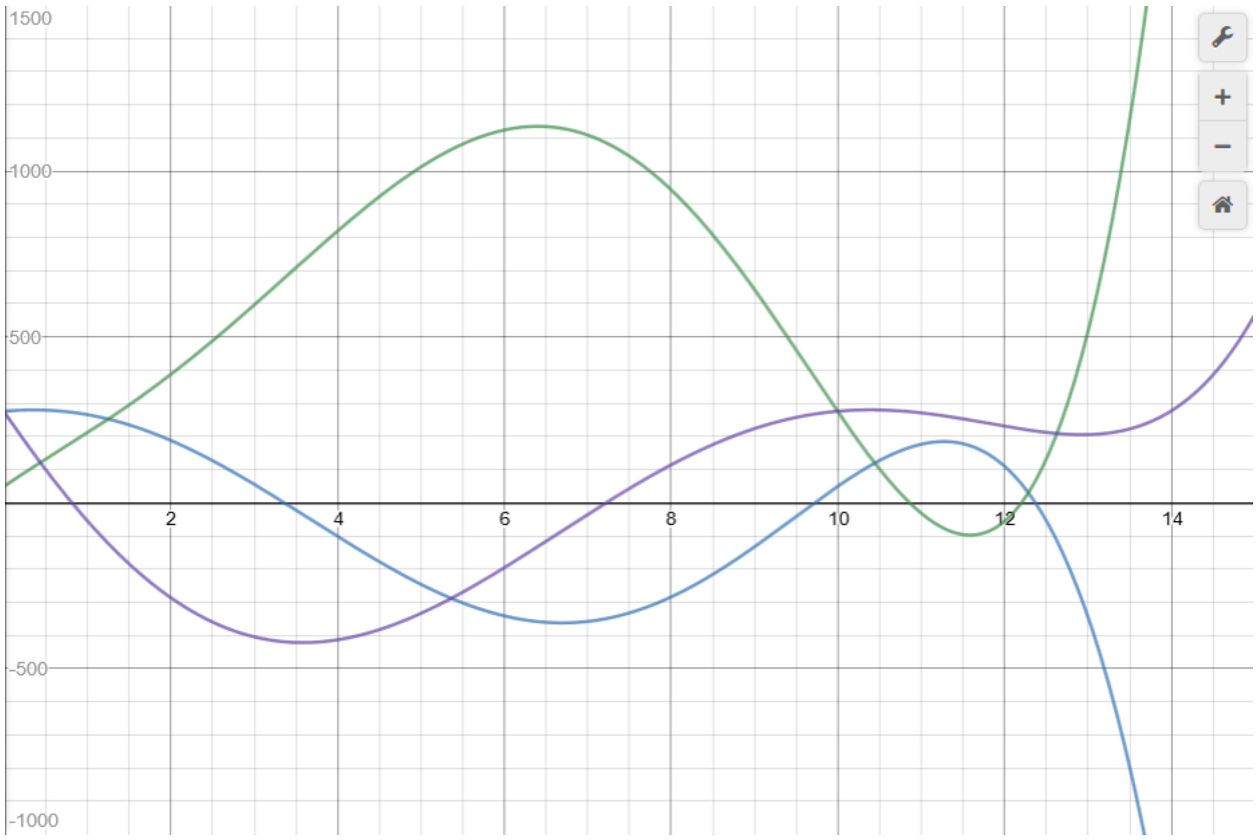
For the very last values, there are no next values, so instead, we wrap around to the very first value. This operation essentially closes the loop continuously because the function that is generated will have the very first position, velocity, and acceleration as it's end position, velocity and acceleration:

$$x_0 = 273;\ x_f = 276$$
$$\dot{x}_0 = -366;\ \dot{x}_f = 22$$
$$\ddot{x}_0 = 58;\ \ddot{x}_f = -59$$

This produces:

$$c_0 = 273;\ c_1 = -366;\ c_2 = 29;\ c_3 = 9.46;\ c_4 = -1.3185;\ c_5 = 0.04488$$

Now, we have all the coefficients required to generate functions to randomly move the *x* coordinate. The function *x*(*t*) looks like this:



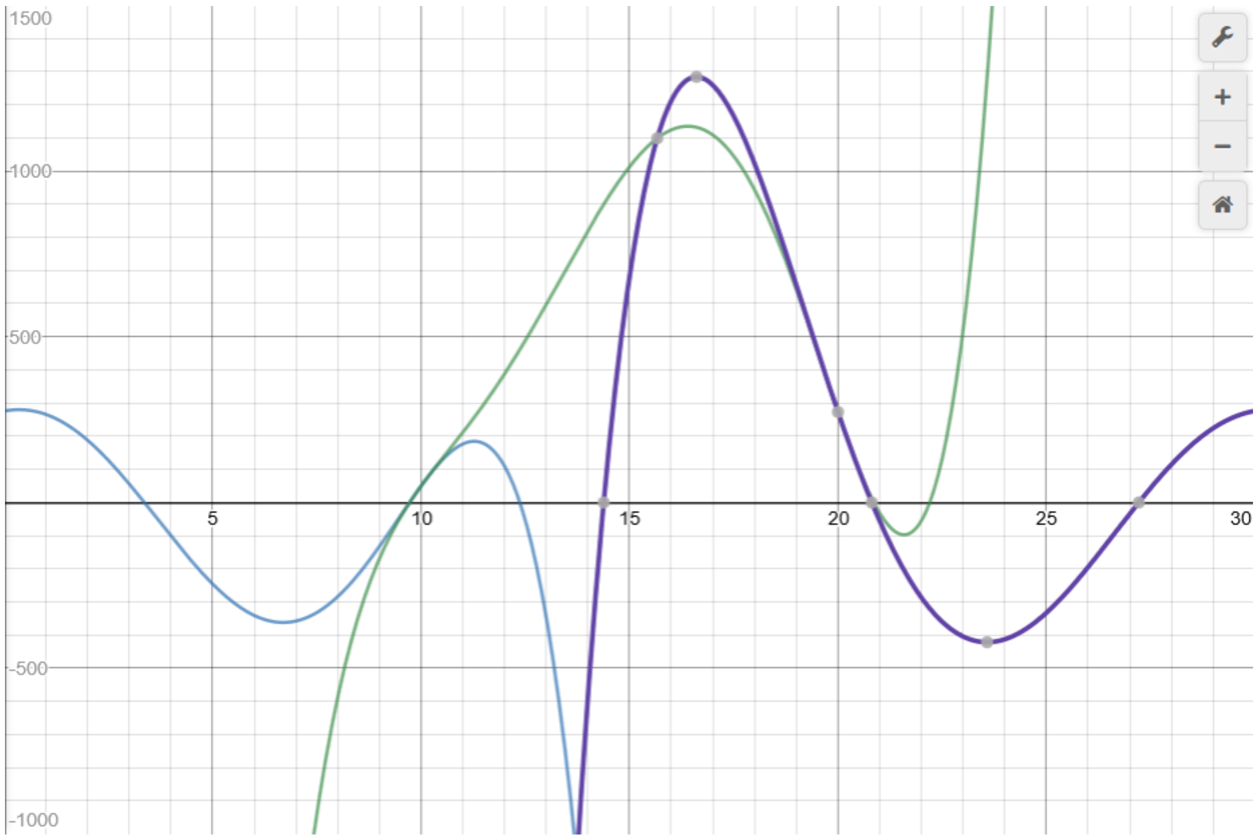
What does this graph mean? Well, not much right now. As you can see, our formulas do not connect with each other. The reason for this is that right now they're all centered at $t = 0$. We need to shift the formulas to their appropriate section:

- For the first formula, no shifting required
- For the second formula, t must be shifted by t_f
- For the last formula, t must be shifted by $2t_f$.

This can be done as such **[Line 99 of the code]**:

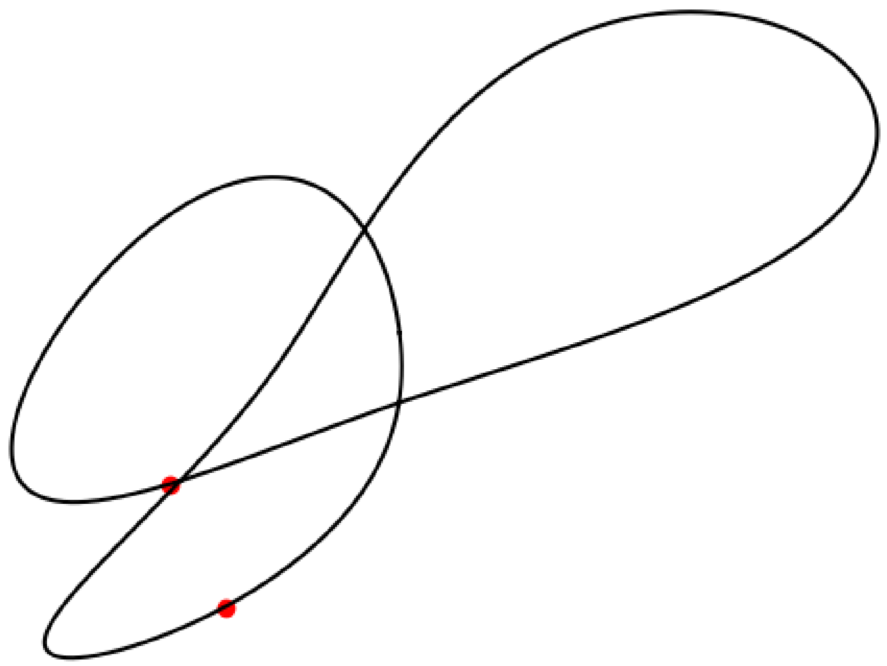
$$x(t) = \begin{cases} x_a(t); when\ 0 < t \leq t_f \\ x_b(t - t_f); when\ t_f < t < 2t_f \\ x_c(t - 2t_f); when\ 2t_f < t < 3t_f \end{cases}$$

This piecewise function with shifting produces the following graph:



At *t* = 10 and *t* = 20, the functions blend in a continuous manner. At *t* = 30, the position, velocity, and acceleration match the initial position, velocity and acceleration.

Now, if we apply this to the *y* coordinate as well, and then we plot both x and y, we get:



Random Point Generation

Due to the nature of these parametric functions, the probability distribution for picking a random point on the contour is non-uniform. Random points which are near to each other generate “denser” sections which have higher probabilities for being picked. To remedy this, the whole contour must be normalized.

To do this, the total length of the contour must be calculated. This is done through an approximation.

While the *x* and *y* values are being generated **[Line 89 of the code]**, the length between each intermediate point is calculated using Euclid’s distance formula **[Line 116 of the code]**:

$$d_{ab} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

Adding up the intermediate point distances produces a total distance (*dist*) **[Line 118 in code]**. All intermediate distances are also saved (*t_dist*[]) **[Line 116 in code]**.

This value can be multiplied by a uniform-random floating-point number from [0, 1) to produce a uniform random length (*r_dist*) **[Line 123 in code]**.

Lastly, we iterate through all intermediate distances (*t_dist*[]) while decrementing the random length (*r_dist*). Once the random length falls under 0, a point is plotted at the current index **[Line 132 in code]**.

FINAL NOTES

My code is sometimes confusing, gonna write notes on things that look weird. Ask away if there's something else you don't understand.

There is sometimes a need to wrap around to the very first value. Take for example line 48 in the code:

```
((20*(xPos[(index + 1) % VIA_POINT_COUNT] - xPos[index]) -
```

(index + 1) % VIA_POINT_COUNT will allow me to wrap around to 0 if (index + 1) is greater than or equal to VIA_POINT_COUNT.

VIA_POINT_COUNT = 5	Number of points that the contour will visit. Increasing increases the complexity of the contour, so mess around with it as you please.
RAND_POINT_COUNT = 2	Number of random points that will be drawn on the curve.
STEP_SIZE = 10	"Time" it takes to reach each via point. If you choose a really small number (< 0.5), the curvy-ness disappears. You really don't need to make this too large (>500), as there's no real added benefit. Going over 1000 runs p slow.
MIN_X_POS = 0; MAX_X_POS = 800	Minimum and maximum x values for via points. The curve can still go out of these bounds, just fyi.
MIN_Y_POS = 0; MAX_Y_POS = 600	Minimum and maximum y values for via points. The curve can still go out of these bounds, just fyi.
VEL_THRESH = 500	Velocities for *x* and *y* will be randomly picked between -VEL_THRESH and VEL_THRESH.
ACC_THRESH = 100	Accelerations for *x* and *y* will be randomly picked between -ACC_THRESH and ACC_THRESH.