Autonomous Path Planning using Self-Evolving Cubic Splines

A. F. Hill

Abstract

This paper introduces a novel method of autonomous path planning using self-evolving cubic splines. Firstly, this paper will discuss the theory behind smoothing splines and how they can be adapted to construct self-evolving splines with the ability to update as new information is discovered by an autonomous agent. Secondly, this paper will discuss in detail how these self-evolving splines can be utilised for new methods of path planning, in particular, how these methods can be used to develop the software in self-driving vehicles. Further to this, this paper will show how Monte Carlo methods can be used effectively for evaluating the success of the path planning system. These evaluation methods will then be used to determine the ideal smoothing parameter of the self-evolving cubic smoothing splines.

1 Introduction

Cubic splines allow for a flexible and powerful method of modelling relationships between a dependent variable and an explanatory variable. A result of this is that cubic splines have many applications, including computer graphics, image interpolation and digital filtering. Cubic splines come in many different forms (smoothing splines, B-splines, and periodic splines to name a few) and can be applied to many interesting problems. Contributing further to this pool of innovative applications, in this paper we will demonstrate the effectiveness of cubic splines at autonomous path-planning.

Path planning refers to the task of generating a path that successfully navigates an agents surroundings. *Autonomous* path planning increases the difficulty of this task by forcing the agent to make decisions on it's own with no external help along the way.

This paper will first discuss the mathematical theory behind cubic splines and cubic smoothing splines. The theory will follow the works of Y. Wang [1], and C. Constable and R. L. Parker [2]. Additionally, the method of computing cubic smoothing splines will be discussed in detail following the works of G. Rodriguez [3]. Following this, the self-evolving cubic spline will be defined and the core autonomous path-planning algorithm will be described and then given in pseudo-code. Furthermore, this paper will then discuss the methods of evaluating path-planning systems, and in particular how Monte Carlo simulations can be used to make estimations of evaluation metrics. Using these evaluation metric estimations this paper will then fine-tune the cubic smoothing splines by optimising the smoothing parameter λ.

2 Theory & Methodology

2.1 Cubic Smoothing Splines

Splines first and foremost offer a powerful method of modelling relations of the form

$$y_i = f(x_i) + \epsilon_i \ , \tag{1}$$

where the ϵ_i are independent random variables with mean 0 and variance σ^2 representing the noise in the data.

Splines are particularly effective at modelling relationships with vastly different behavior across different regions of the domain of the explanatory variable. In essense, splines are piecewise polynomials joined together smoothly at *knots*. What we consider to be smooth in this context depends upon the order of the spline.

Definition 1. *Let $a < \xi_1 < \cdots < \xi_k < b$ be fixed points called knots. Let $\xi_0 = a$ and $\xi_{k+1} = b$. Then a spline of order d is a real-valued function on $[a, b]$, $f(x)$, such that*

(i) f is a piecewise polynomial of order d on $[\xi_i, \xi_{i+1})$ $\;for\;$ $i = 0, 1, \ldots, k$

(ii) f has $d - 2$ continuous derivatives

From this definition one can see that splines are simply piece-wise polynomials with some additional constraints to preserve smoothness of the resulting function. When $d = 3$ we arrive at the *cubic spline*.

A spline of odd degree $d = 2r - 1$ is called a *natural spline* if it is a polynomial of degree $r - 1$ outside the range of the knots. Therefore, a *natural cubic spline* is a cubic spline which is linear outside the range of the data. Natural cubic spines are usually a robust choice of model if one wants to make predictions outside the range of the training data.

The method we now use to fit the piece-wise polynomials to the data will now determine what type of spline we create. A *regression spline* is the piece-wise polynomial $f(x)$ which minimises the *residual sum of squares* (RSS)

$$RSS = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f\left(x_i\right)\right)^2 \tag{2}$$

subject to the constraints in Def. (1).

The regression spline can now be shown to be just a particular case of the more general *smoothing spline*. The relationship between regression splines and smoothing splines is analogous to the relationship between linear regression and ridge regression. Just as ridge regression introduces a penalty term on the norms of the model parameters with the goals of regularisation and improved generalisation, smoothing splines add a penalty term on the norms of the derivatives of the piece-wise polynomial, with the exact same goals in mind.

Thus, the smoothing spline of order d is the piece-wise polynomial $f(x)$ which minimises the *penalised residual sum of squares* (PRSS)

$$PRSS = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f\left(x_i\right)\right)^2 + \lambda\int_a^b \left(f^{(d-1)}\right)^2 dx \tag{3}$$

where λ is the *smoothing parameter*.

The smoothing parameter has a significant qualitative effect on the resulting smoothing spline. As $\lambda \to 0$ we impose no smoothness penalty and end up with a very close fit to the data, however the resulting curve could be very noisy as it follows every detail in the data. As $\lambda \to \infty$ the smoothness penalty dominates and the solution converges to the ordinary least squares line, which is as smooth as you can get (with second derivative always 0), but may be severely underfitting. The cubic smoothing spline is for when d = 3

Surprisingly, it can be shown that minimizing the PRSS for a fixed λ over the space of all continuous differentiable functions leads to a unique solution. We will now turn our attention to how we can compute this unique solution for a cubic smoothing spline of fixed λ, following closely to the works of G. Rodriguez [3].

Implicit in the work of Reinsch is the fact that the penalty may be written as a quadratic form [4]

$$\int_a^b \left(f''(x)\right)^2 dx = \mu' K \mu$$

where $\mu = f\left(x_i\right)$ is the fit, K is an $n \times n$ matrix given by $K = \Delta' W^{-1}\Delta$, Δ is an $(n-2) \times n$ matrix of second differences, with elements

$$\Delta_{ii} = \frac{1}{h_i}, \Delta_{i,i+1} = -\frac{1}{h_i} - \frac{1}{h_{i+1}}, \Delta_{i,i+2} = \frac{1}{h_{i+1}}$$

W is a symmetric tridiagonal matrix of order $n-2$ with elements

$$W_{i-1,i} = W_{i,i-1} = \frac{h_i}{6}, w_{ii} = \frac{h_i + h_{i+1}}{3}$$

and $h_i = \xi_{i+1} - \xi_i$, the distance between successive knots. This implies that we can compute a smoothing spline as

$$\hat{f}(x) = (I + \lambda K)^{-1} y$$

Proof. : Write the penalized residual sum of squares as

$$\text{PRSS} = (y-\mu)'(y-\mu) + \lambda\mu' K \mu$$

Taking derivatives

$$\frac{\partial \text{PRSS}}{\partial \mu} = -2(y-\mu) + 2\lambda K \mu$$

Setting this to zero gives

$$y = \hat{\mu} + \lambda K \hat{\mu} = (I + \lambda K)\hat{\mu}$$

and pre-multiplying both sides by $(I + \lambda K)^{-1}$ completes the proof. $\square$

This computation using matrices K, Δ and W allows for efficient implementations of computing the smoothing splines [3]. For implementation of smoothing splines in Python the library SciPy was used, which has many well designed functions for creating smoothing splines. Similarly, in R, one can use the in-built function smooth.spline() to fit smoothing splines to data. These implementations of fitting smoothing splines were originally designed by P. Dierckx in 1975 [5–8].

Finally, with the cubic smoothing spline properly introduced we may now define the *self-evolving cubic spline*.

Definition 2. *Let S be an online system where new data points (x_i, y_i) are 'discovered' at times t_i with $0 \leq t_0 \leq \ldots \leq t_n \leq t$. Then the self-evolving cubic spline is a cubic smoothing spline, which upon the discovery of a new data point (x_i, y_i) at time t_i, re-computes the entirety of the cubic smoothing spline in light of the new information.*

The self-evolving cubic spline can be used in any online system that continuously gathers information. For the remainder of this paper we will demonstrate the effectiveness of the self-evolving cubic spline in autonomous path-planning.

2.2 Path Planning

Path planning refers to the task of generating a path that successfully navigates an agents surroundings. There are many ways to simulate an agent in space with the objective of successfully navigating it's surroundings. In this paper we will demonstrate a particular case of path planning; simulating a self-driving car travelling around a 2 dimensional track, where the boundary of the track is indicated by cones.

The self-driving car will be simulated to only have a limited field of view, both in distance and angle. This way, information about the track will only be available to the car once the it gets close enough to it and is pointing in the right direction. The car's field of view can be seen in Figure (1).

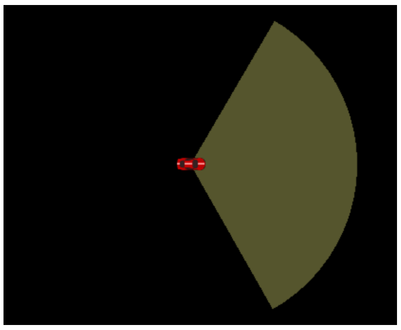


Figure 1: The field of view (in yellow) of the simulated self-driving car with the objective of path planning

This agent coupled with the mechanics of autonomous driving can be seen as simply an online system where new data points (cones) are discovered as the car traverses the path that it generates for itself. The autonomous driving mechanics which allow the car to follow the generated path were implemented in Python and can be found in the following repository on Github `https://github.com/alex21347/Cubic_Spline_Path_Planning`.

The path planning system generates the path by first collecting a list of visible cones, and then constructing a list of midpoints between left cones and right cones (corresponding to the left and right sides of the track). In general, this set of midpoints will form a point cloud in the middle of the track. The agent can now compute a cubic smoothing spline through this point cloud of midpoints. This spline can now be evaluated at equidistant points along the curve to generate a set of 'targets' for the car to follow, a process referred to as *thinning*. This system can be seen below in pseudo-code.

Algorithm 1 Path planning

```
$cone\ list \leftarrow \emptyset$
$target\ list \leftarrow \emptyset$
$cone\ detected \leftarrow False$
$Autonomous\ driving \leftarrow True$
while $Autonomous\ Driving = True$ do
    if $target\ list \neq \emptyset$ then
        $Drive\ through\ targets\ in\ targets\ list$
    if $new\ cone\ detected$ then
        $cone\ detected \leftarrow True$
    if $cone\ detected = True$ then
        $midpoints\ list \leftarrow \{midpoints\ of\ all\ left/right\ cones\}$
        $New\ spline \leftarrow Compute\ Spline\ (midpoints\ list\ ,\ \lambda)$
        $Target\ list \leftarrow Thinned\ list\ (New\ spline)$
        $cone\ detected \leftarrow False$
```

There are many subtleties in the path planning system not included in the above pseudo-code for the sake of clarity and brevity (for example the mechanism that governs which targets have been reached by the agent and thus need to be removed from the target list), however, all of the key elements are addressed. Additionally, to estimate the boundaries of the track, cubic splines can also be used to interpolate the cones themselves, as can be seen in the final simulations. To see a live simulation of the path planning system in action, the reader is strongly encouraged to visit the github repository : `https://github.com/alex21347/Cubic_Spline_Path_Planning`.

Now that the self-evolving cubic splines are at the core of the autonomous path planning system, we must somehow decide which smoothing parameter λ is optimal in practice. To do this, we will now formulate a quantitative method of evaluating the path-planning system.

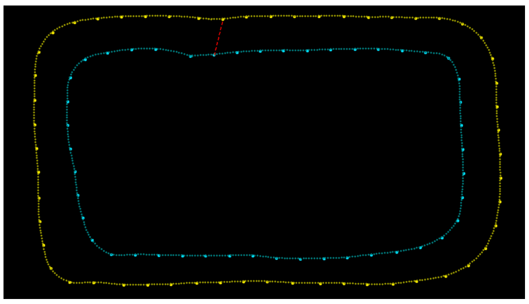
3 Evaluation & Fine-Tuning

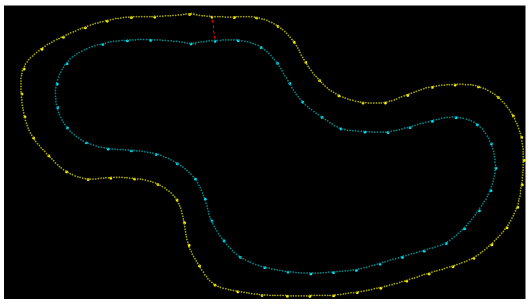
To fine-tune the path-planning, we will now investigate which smoothing parameter λ is the most effective/successful for constructing the cubic smoothing splines. To evaluate the smoothing parameter we need a set of evaluation metrics that indicate the success of an agents path-planning, and a method of estimating these metrics for different smoothing parameters.

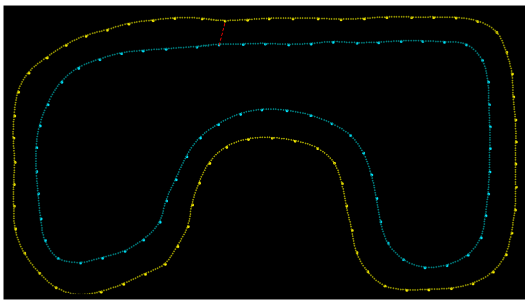
3.1 Monte Carlo Simulation

To make estimates of the different evaluation metrics one can use Monte Carlo simulations. The agent will be simulated hundreds of times and data will be retrieved from the simulations to make Monte Carlo estimations of the metrics.

The path planning system needs to be tested in such a way that the track that we choose to simulate the agent on does not greatly bias the results. To achieve this, the agent was simulated driving around four unique tracks over multiple laps. The four tracks can be seen in Figure (2).


(a) Track 1


(b) Track 2


(c) Track 3

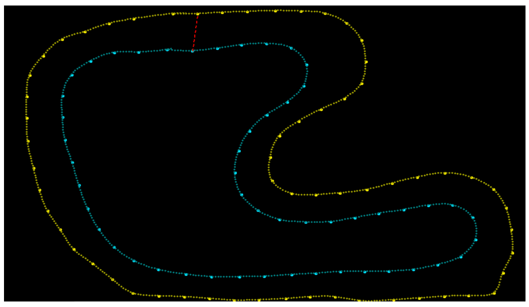

(d) Track 4

Figure 2: Four test tracks used to evaluate the autonomous path planning system.

3.2 Evaluation Metrics

3.2.1 Success Rate

The success rate of the agents journey is the most important evaluation metric. Specific characteristics of the agents journey cease to matter if the agent cannot successfully complete the journey in the first place. A simulation was deemed a success if the agent was able to complete 3 laps of the track without crashing. A failure was automatically assigned to a simulation if the agent:

(a) Hits the boundary of the track (crashes)

(b) A full lap is left uncompleted

For each of the four test tracks, 100 simulations were carried out, and then the final Monte Carlo estimate of the success rate is given as the total number of successes divided by the total number of simulations.

3.2.2 Average Lap Time

The average lap time is the evaluation metric that gives insight into the speed at which the agent is able to traverse the tracks. The speed at which the agent travels is fixed for the simulations, thus the lap time is governed exclusively by the path taken by the agent, with a shorter path offering a faster average lap time.

For each of the four test tracks, 100 simulations were carried out, and then the average lap time is given by the mean lap time over the four tracks. There is a small nuance involved with this evaluation metric as the longer tracks will have a greater effect on the average lap time due to having a larger magnitude. To mitigate this, we can normalise the lap times with respect to each track when taking averages.

3.2.3 Journey Smoothness

Lastly, the journey smoothness can be evaluated using the successive positions of the agent over time. In any journey, the aspect that describes the smoothness or bumpiness of a journey is the acceleration one feels during the journey. The velocity has no effect on the agent regarding the inertial force it experiences, it is only the *change* in velocity that one can feel.

Therefore by retrieving the positions of the agent at each iteration of the simulation, the velocity can be estimated simply by using the difference between subsequent positions. Similarly, the agents acceleration can then be estimated using the difference between subsequent velocity estimates at each iteration. The overall journey smoothness can then be evaluated as the average amount of acceleration the agent experiences per lap. To turn this average acceleration per lap into a journey smoothness score, we can then apply Min-Max normalisation and subtract this from 1. This results in a journey smoothness score in the interval $[0, 1]$ where a greater score indicates a smoother journey as desired.

4 Results & Discussion

The three previously discussed evaluation metrics were estimated for 7 distinct smoothing parameters in $[0, 1.5]$. The results can be seen below in Table (1).

Smoothing Parameter	**Success Rate**	**Average Lap Time**.	**Journey Smoothness**
0	63%	11.61s	0.236
0.25	69%	11.83s	0.688
0.5	81%	11.59s	0.558
0.75	75%	11.64s	0.644
1	75%	11.74s	0.900
1.25	**100%**	11.36s	0.903
1.5	94%	**11.35s**	**0.981**

Table 1: Success Rates, Average Lap Times, and Journey Smoothness values retrieved through Monte Carlo simulations for different Smoothing Parameters. Most successful results for each evaluation metric are in bold.

Firstly, it is clear that the success rate is much higher when the smoothing parameter λ is increased. This can be attributed to the fact that smoother splines are less sensitive to small quirks in the data. The smoother splines were more robust to small perturbations and this lead to a more stable and successful path planning system. For $\lambda = 1.25$ the car was able to successfully complete 3 full laps 100% of the time which provides a strong indication of the advantages of using cubic smoothing splines in autonomous path planning.

Additionally, it can be seen in Table (1) that the greater the smoothing parameter λ, the smaller the average lap time. This also makes intuitive sense as a journey that makes many small positional changes will be longer than a journey that 'cuts through' the small changes. Therefore the smoother the cubic smoothing spline, the faster the agent. Lastly, the journey smoothness metric increases as the smoothing parameter increases. This result is a good sanity check that a smoother cubic smoothing spline actually does result in a smoother journey for the agent.

The most important of the three evaluation metrics is the success rate, therefore the final fine-tuned cubic smoothing spline for autonomous path planning will use the optimal smoothing parameter of $\lambda = 1.25$.

5 Conclusion

In this paper we constructed a novel method of autonomous path planning using self-evolving cubic splines. Modifying the smoothing spline we were able to create self-evolving splines with the ability to automatically update as time progresses and more data is collected by an agent. Through Monte Carlo estimation we found that the optimal smoothing parameter was $\lambda = 1.25$. With this fine-tuned path planning system the simulated car was able to attain a perfect 100% success rate over the four test tracks, whilst also providing the second fastest average lap time, and second smoothest journey.

For further research, it would be beneficial to experiment with and compare different types of self-evolving cubic splines. In this paper we used only cubic smoothing splines to construct self-evolving splines, however, in theory other types of splines can also be used for autonomous path planning. Additionally, it would be beneficial to study the autonomous path planning system in different (and potentially more complex) contexts.

References

[1] Y. Wang, *Smoothing splines: methods and applications*. CRC Press, 2011.

[2] C. Constable and R. Parker, "Smoothing, splines and smoothing splines; their application in geomagnetism," *Journal of Computational Physics*, vol. 78, no. 2, pp. 493–508, 1988.

[3] G. Rodrıguez, "Smoothing and non-parametric regression," tech. rep., Working paper, 2001.

[4] C. H. Reinsch, "Smoothing by spline functions," *Numerische mathematik*, vol. 10, no. 3, pp. 177–183, 1967.

[5] P. Dierckx, "An algorithm for smoothing, differentiation and integration of experimental data using spline functions," *Journal of Computational and Applied Mathematics*, vol. 1, no. 3, pp. 165–184, 1975.

[6] P. Dierckx, "An improved algorithm for curve fitting with spline functions," *TW Reports*, 1981.

[7] P. Dierckx, "A fast algorithm for smoothing data on a rectangular grid while using spline functions," *SIAM Journal on Numerical Analysis*, vol. 19, no. 6, pp. 1286–1304, 1982.

[8] P. Dierckx, *Curve and surface fitting with splines*. Oxford University Press, 1995.