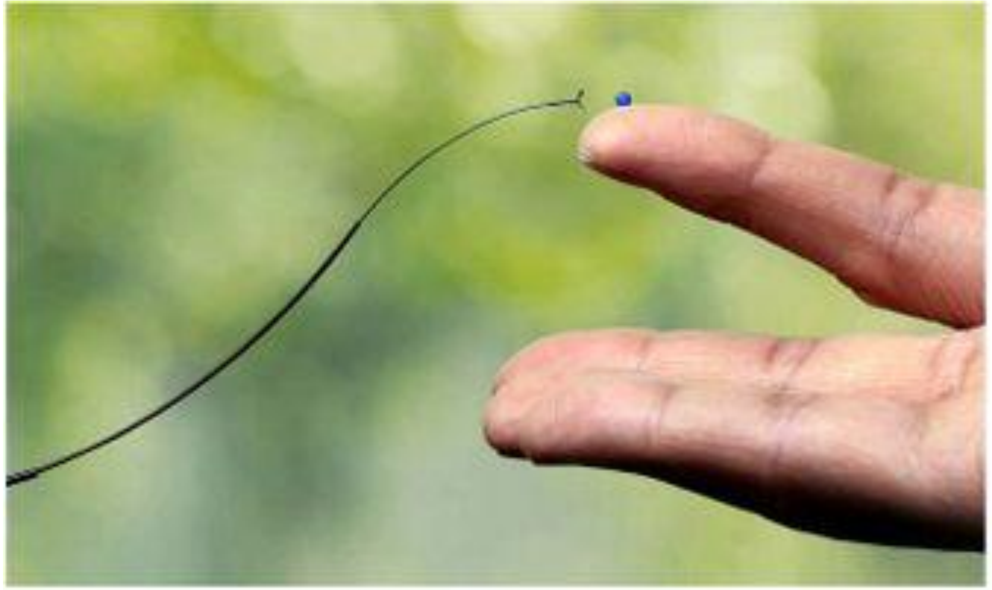
B-Splines for Continuum Robotic Modeling and Kinematics

Isabelle Byrne

What are continuum robots?

What are continuum robots?



Closed-Loop Feedback Control

1. Sensing
   - Sensors collect real-time data about the robot
2. State Estimation
   - Data is processed and model provides an estimate of the current config.
3. Error Computation
   - Difference between the current estimated state and the desired config

Forward Kinematics

4. Control Algorithm
   - Process error to generate appropriate control actions

Inverse Kinematics

5. Actuation
   - Apply the new state to the robot via commands to actuators

B-Splines

- Piecewise polynomial function
- Constructed from a set of basis functions, control points and a knot vector. The spline does not interpolate the control points, but rather is “pulled” by them
  - **Basis Functions**: Constructed via the de Boor algorithm. They have local support and determine how each control point contributes to the shape.
  - **Control Points**: These are the “handles” that when adjusted change the curve.
  - **Knot Vector**: Sequence of parameter values that determines where the polynomial pieces join together, and also influences the smoothness of the curve. In this project I use a clamped uniform knot vector.

Why B-Splines

- Smoothness and Continuity
  - C^2
- Local Control
  - Individual robotic segments
- Computational Efficiency
  - Few parameters

Mathematical Formulation of B-Splines

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i$$

Where:

- $u \in [0,1]$ is the parametric variable; ‘slider’
- $N_{i,p}(u)$ are the B-spline basis functions; ‘weights’
- $\mathbf{P}_i \in \mathbb{R}^d$ ($d = p = 3$ for 3D robotics)

This curve will allow us to describe the backbone of variable curvature robots!

Ex. weighted sum

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i$$

For some point u along the curve:

- $N_{0,p}(u) = 0.7$
- $N_{1,p}(u) = 0.3$
- $\mathbf{C}(u) = 0.7\mathbf{P}_0 + 0.3\mathbf{P}_1$

Knot Vector

- For a clamped B-spline (forces curve to pass through the first and last control points), the knot vector has the form

$$\mathbf{U} = [\underbrace{0, 0, \ldots, 0}_{p+1 \text{ times}}, u_{p+1}, u_{p+2}, \ldots, u_n, \underbrace{1, 1, \ldots, 1}_{p+1 \text{ times}}]$$

$$\mathbf{C}(0) = \sum_{i=0}^{n} N_{i,p}(0)\mathbf{P}_i = \mathbf{P}_0$$

- Internal knots are calculated via:

$$u_j = \begin{cases} 0 & \text{if } j \leq p, \\ \frac{j-p}{n-p+1} & \text{if } p < j < n+1, \\ 1 & \text{if } j \geq n+1. \end{cases}$$

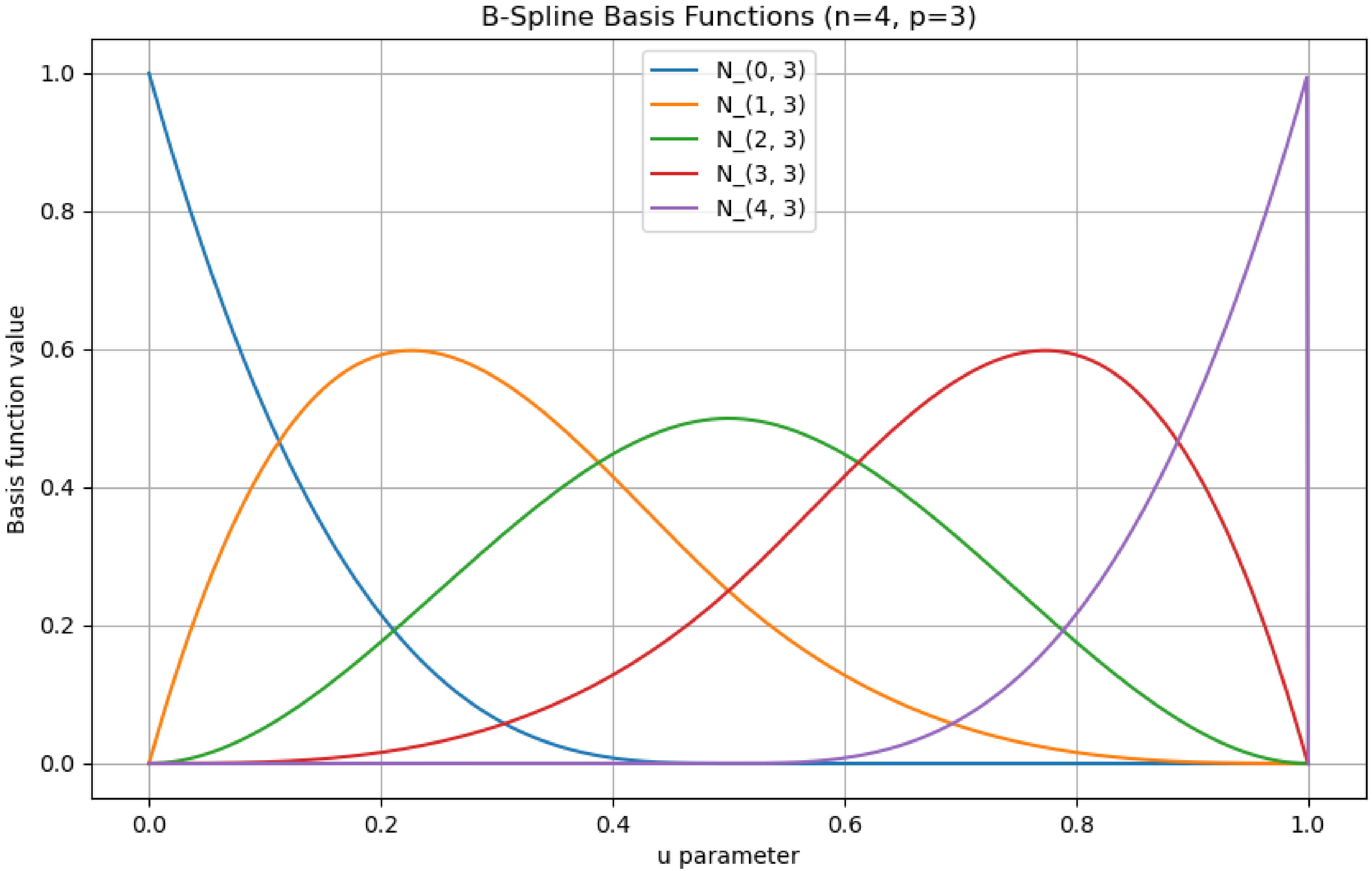
de Boor Algorithm

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u < u_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u),$$

This is computationally expensive

- Repeats calculations due to recursive nature – implement a memory/cache
- Still computes extra terms for knots outside of the span of u – implement non-recursive algorithm
  will show this in code later


B-Spline Basis Functions (n=4, p=3)
N_(0, 3)
N_(1, 3)
N_(2, 3)
N_(3, 3)
N_(4, 3)
Basis function value
u parameter
0.0
0.2
0.4
0.6
0.8
1.0

Forward Kinematics

- Mapping the parameters of the B-Spline model to the spatial positions and orientation along the robot's backbone.
- Position: Directly obtained from the B-spline curve

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i$$

- Orientation: Derived from the derivative:

$$\mathbf{C}'(u) = \sum_{i=0}^{n} N'_{i,p}(u)\,\mathbf{P}_i \qquad \mathbf{T}(u) = \frac{\mathbf{C}'(u)}{||\mathbf{C}'(u)||}$$

Transformation Matrix

$$T(u) = \begin{bmatrix} \mathbf{R}(u) & \mathbf{C}(u) \\ \mathbf{0}^\top & 1 \end{bmatrix}$$

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i$$

$$\mathbf{R}(u) = \begin{bmatrix} \mathbf{x}(u) & \mathbf{y}(u) & \mathbf{z}(u) \end{bmatrix}$$

$$\mathbf{z}(u) = \frac{\mathbf{T}(u)}{\|\mathbf{T}(u)\|}$$

$$\mathbf{x}(u) = \frac{\mathbf{r} \times \mathbf{z}(u)}{\|\mathbf{r} \times \mathbf{z}(u)\|}$$

$$\mathbf{y}(u) = \mathbf{z}(u) \times \mathbf{x}(u)$$

$$\mathbf{T}(u) \approx \frac{\mathbf{C}(u+\delta) - \mathbf{C}(u)}{\|\mathbf{C}(u+\delta) - \mathbf{C}(u)\|}$$

Where $\boldsymbol{r} = [\mathbf{0}, \mathbf{0}, \mathbf{1}]$ or another vector if $\boldsymbol{z}(u)$ is nearly collinear

Inverse Kinematics

- The goal is to find the control points $\{\mathbf{P}_1, \ldots, \mathbf{P}_\mathrm{n}\}$ (with base point $\mathbf{P}_0$ fixed) so that the end-effector located at $\mathbf{C}(1)$ reaches a specified target $\mathbf{T}_{\mathrm{target}}$.
- Solved using optimization over a cost function:

$$J(\mathbf{P}_1, \ldots, \mathbf{P}_n) = \lambda_d \left\|\mathbf{C}(1) - \mathbf{T}_{\mathrm{target}}\right\|^2 + \lambda_{\mathrm{smooth}} \sum_{i=1}^{n} \left\|\mathbf{P}_i - \mathbf{P}_{i-1}\right\|^2 + J_{\mathrm{collision}}$$

- Used L-BFGS-B solver, a quasi-Newton method that can handle bound constraints for the robotic movement

Code

- Let’s look at the code and visualization

Summary

- Overview on continuum robotics
- Introduction to B-splines
- Using B-splines to model variable curvature continuum robots
- Performing forward kinematics with this model
- Solving the inverse kinematics problem
- Robots are so cool!!

Questions? Comments? Concerns? Grievances?

- Thanks!
- I hope you learned something or were inspired to learn more about robotics :D