

Filtering Unique Curves

SLAPStack

June 3, 2025

We want to find a minimal set of curves S_C such that every curve in the search space S can be obtained from some curve in S_C by applying a translation and/or rotation. In the context of the code implementation, the problem can be restated to: Given an array of curve objects S , find the minimum set of curves S_C , $S_C \subseteq S$, such that every curve in S can be obtained from some curve in S_C by applying a translation and/or rotation.

1 Establishing the mathematics

Assume we are given a collection of 2D curves:

$$\mathcal{C} = \{C^{(1)}, C^{(2)}, \dots, C^{(m)}\}$$

where each curve is a finite ordered list of points in \mathbb{R}^2 :

$$C^{(k)} = [p_1^{(k)}, p_2^{(k)}, \dots, p_n^{(k)}], p_i^{(k)} \in \mathbb{R}^2$$

and each curve consists of exactly n points. Also, the first segment of a curve is not zero in length (i.e. $p_1 \neq p_2$). We say two curves C_1 and C_2 are equivalent if:

$$\exists R \in SO(2), t \in \mathbb{R}^2 \text{ such that } C_2 = RC_1 + t$$

where $SO(2)$ is the Special Orthogonal Group in \mathbb{R}^2 , or, the set of all 2D rotation matrices.

2 An algorithmic approach

2.1 Base filtering

First filter curves based on the location of the maneuver, i.e. whether the curve occurs in a lane, intersection, or dock. Now we have three non-overlapping arrays of curves: `lane_curves`, `intersection_curves`, `dock_curves`. Since a curve is implemented as an array of points, we can further filter based on array length. An example of the previous filtering results are

- `lane_curves_5`

- `lane_curves_6`
- `intersection_curves_5`
- `intersection_curves_7`
- `dock_curves_6`
- `dock_curves_9`

where, for example, `lane_curves_5` represents curves in `lane_curves` with array length of 5. We now apply the following normalization procedure to each of the filtered arrays above.

2.2 Centering the curves

For each curve $C = [p_1, p_2, \dots, p_n]$, move it to the origin by subtracting the first point from every point. So

$$\tilde{C} = [0, p_2 - p_1, \dots, p_n - p_1] = [0, \tilde{p}_2, \dots, \tilde{p}_n].$$

2.3 Rotation normalization

For each \tilde{C} , let the first segment vector be $v = \tilde{p}_2 - 0 = \tilde{p}_2$. We find the angle between v and the x-axis:

$$\theta = -\arctan2(v_y, v_x)$$

Note: `arctan2(y, x)` is a built in function in numpy that helps prevent division by zero and other quadrant problems.

Define the rotation matrix:

$$R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

Rotate all points:

$$\hat{p}_i = R(\theta) \cdot \tilde{p}_i$$

So

$$\hat{C} = [0, R(\theta) \cdot \tilde{p}_2, \dots, R(\theta) \cdot \tilde{p}_n]$$

2.4 Matching

After applying the previous steps, all arrays `lane_curves_5`, `lane_curves_6`, ..., `dock_curves_9` should be translationally and rotationally normalized.

Now use numpy's `unique()` method. For each array, apply the `unique()` method and add the result to S_C . Return S_C and we are done.