Guaranteed Path Planning Using Interval Analysis

Benoit DESROCHERS

ENSTA Bretagne
DGA Techniques Navales (Brest)

November 25, 2015

Outlines

Outlines

1. Problem Statement
   - Definitions
   - 2D version of the Wire Loop Game
   - Modeling
2. Tools and Concepts
3. Path Planning Algorithms

Definitions

Path Planning

Find a sequence of states in order to move a system from an initial state to a final state while avoiding collision.

$\mathbb{C}_{\text{space}}$

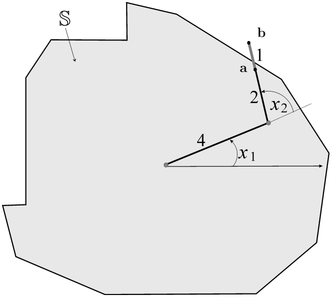
Space of all feasible states for our system.

$\mathbb{C}_{\text{free}}$

Subset of $\mathbb{C}_{\text{space}}$ that avoids collision with obstacles

2D version of the Wire Loop Game

Find a feasible path such as arms could make a full rotation without touching the loop to the wire.

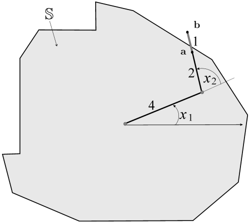


2D version of the Wire Loop Game

Here we have :

$$\mathbb{C}_{\text{space}} = \{(x_1, x_2),\ x_1 \in [0, 2\pi] \text{ and } x_2 \in [0, 2\pi]\}$$

$$\mathbb{C}_{\text{free}} = \{\mathbf{x} \in [0, 2\pi]^2,\ \mathbf{a} \in \mathbb{S} \text{ and } \mathbf{b} \notin \mathbb{S}\}$$



Set Inversion Problem

The problem is summarized with the following set inversion:

$$\mathbb{C}_{\text{free}} = f_{\mathbf{a}}^{-1}(\mathbb{S}) \cap f_{\mathbf{b}}^{-1}(\overline{\mathbb{S}})$$

where $f_{\mathbf{a}}$ (resp. $f_{\mathbf{b}}$) is a non linear function which transforms the point $\mathcal{O}(0,0)^T$ into the $\mathbf{a}$ (resp. $\mathbf{b}$).
In our case f_i is :

$$f_{\mathbf{i}}(\mathbf{x}) = 4\begin{pmatrix} \cos(x_1) \\ \sin(x_1) \end{pmatrix} + l_i \begin{pmatrix} \cos(x_1 + x_2) \\ \sin(x_1 + x_2) \end{pmatrix}$$

with $l_{\mathbf{a}} = 2$ and $l_{\mathbf{b}} = 3$ and $\mathbf{x} = (x_1, x_2)^T$.

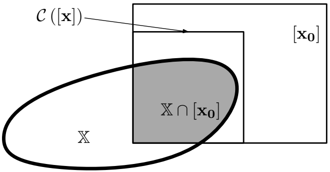
Outlines

Contractor Definition

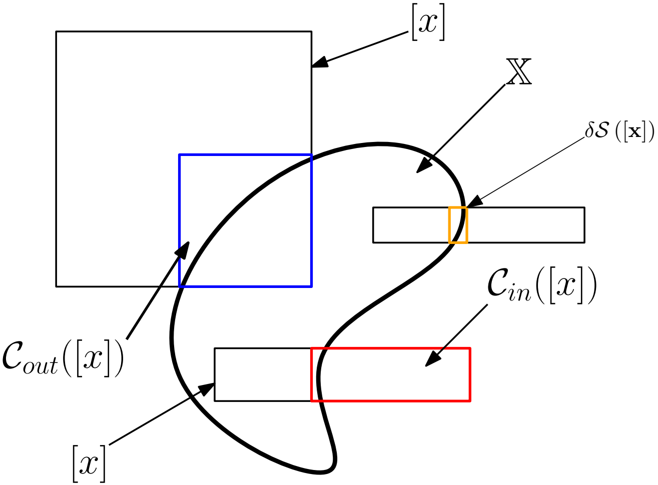
Definition

A contractor is an operator from $\mathbb{IR}^n \longrightarrow \mathbb{IR}^m$ associated to a set $\mathbb{X}$ such as:

$$\begin{array}{lll} \forall[\mathbf{x}] \in \mathbb{IR}^n & , \mathcal{C}([\mathbf{x}]) \subset [\mathbf{x}] & (\textit{contractance}) \\ \forall[\mathbf{x}] \in \mathbb{IR}^n & , \mathcal{C}([\mathbf{x}]) \cap \mathbb{X} = [\mathbf{x}] \cap \mathbb{X} & (\textit{completeness}) \end{array}$$



Inner / Outer Contractor



Seperators

A separator $\mathcal{S}$ associated with the set $\mathbb{X}$ is an application such as:

$$\begin{array}{rccc} \mathcal{S}: & \mathrm{I\!R}^n & \longrightarrow & \mathrm{I\!R}^m \times \mathrm{I\!R}^m \\ & [\mathbf{x}] & \longmapsto & ([\mathbf{x_{in}}], [\mathbf{x_{out}}]) \end{array}$$

with following properties:

$$\begin{array}{lccc} (i) & [\mathbf{x}] & = & [\mathbf{x_{out}}] \cup [\mathbf{x_{in}}] \\ (ii) & [\mathbf{x_{out}}] \cap \mathbb{X} & = & [\mathbf{x}] \cap \mathbb{X} \\ (iii) & [\mathbf{x_{in}}] \cap \overline{\mathbb{X}} & = & [\mathbf{x}] \cap \overline{\mathbb{X}} \end{array}$$

Operations with separators

The Separator algebra is a direct extension of contractor algebra.
With $\mathcal{S}_i = \left\{\mathcal{S}_i^{in}, \mathcal{S}_i^{out}\right\}$ we define :

$$
\begin{array}{rcll}
\mathcal{S}_1 \cap \mathcal{S}_2 & = & \left\{\mathcal{S}_1^{\text{in}} \cup \mathcal{S}_2^{\text{in}}, \mathcal{S}_1^{\text{out}} \cap \mathcal{S}_2^{\text{out}}\right\} & \text{(intersection)} \\
\mathcal{S}_1 \cup \mathcal{S}_2 & = & \left\{\mathcal{S}_1^{\text{in}} \cap \mathcal{S}_2^{\text{in}}, \mathcal{S}_1^{\text{out}} \cup \mathcal{S}_2^{\text{out}}\right\} & \text{(union)} \\
\mathcal{S}_1 \backslash \mathcal{S}_2 & = & \mathcal{S}_1 \cap \overline{\mathcal{S}_2}. & \text{(difference)}
\end{array}
$$

With $\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ the transformation of a separator is:

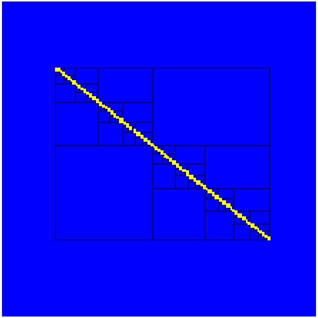
$$
\mathbf{f}\left(\mathcal{S}_1\right) \stackrel{\text{def}}{=} \left\{\mathbf{f} \circ \mathcal{S}_1^{\text{in}} \circ \mathbf{f}^{-1}, \mathbf{f} \circ \mathcal{S}_1^{\text{out}} \circ \mathbf{f}^{-1}\right\}
$$

PyIbex Example

Point inside Segment

A polygon P is an union of segments
Point on segment verifies:

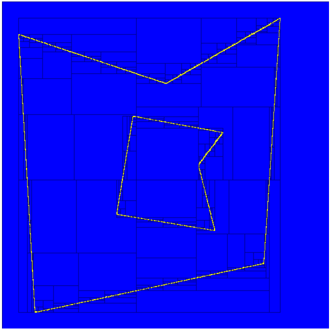
$$\begin{cases} \det(\mathbf{b}-\mathbf{a},\mathbf{a}-\mathbf{m}) = 0 \\ \min(\mathbf{a},\mathbf{b}) \leq \mathbf{m} \leq \max(\mathbf{a},\mathbf{b}). \end{cases}$$



Point on the border of a Polygon

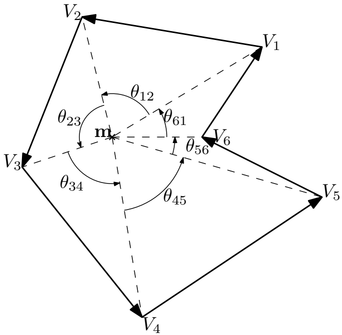
$\mathcal{C}_{a_i,b_i}$ as a contractor for the segment $[\mathbf{a_i}, \mathbf{b_i}]$, the contractor for $\Delta\mathcal{P}$ is:

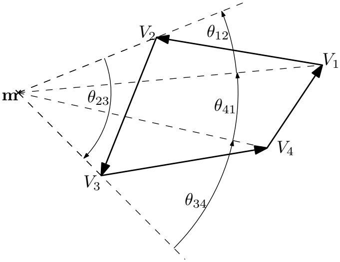
$$\mathcal{C}_{\Delta\mathcal{P}} = \bigcup_{i=1}^{N} \mathcal{C}_{a_i,b_i}$$



Winding Number

$$\mathbf{wn}(\mathbf{m}, P) = \frac{1}{2\pi}\sum_{i=1}^{n} \theta_i$$

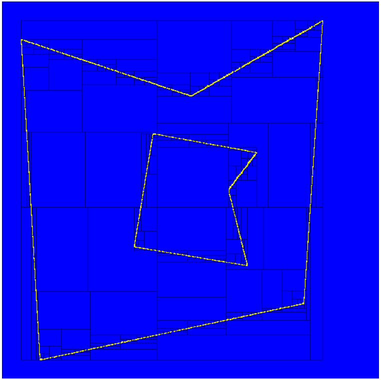


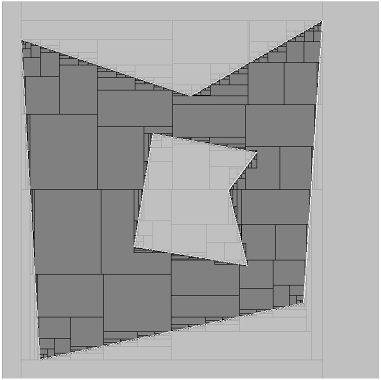


Point Inside a Polygone

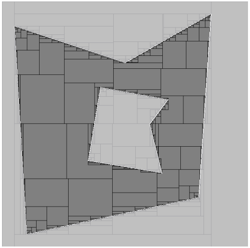
We define the separator for the border of P as follows:

$$\text{Separator } \mathcal{S} = \{\mathcal{C}_{\Delta\mathcal{P}}, \mathcal{T}_{\mathbf{wn}}\}$$

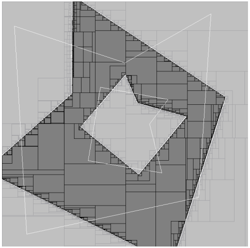




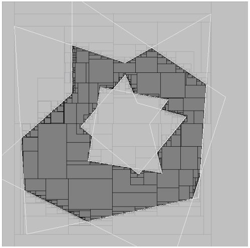
Separator Transformation



$\mathbb{S}$



$\mathbb{S}_R = Rot_{\frac{\pi}{2}}(\mathbb{S})$



$\mathbb{S}_R \cap \mathbb{S}$

Outlines

$\mathbb{C}_{\text{free}}$ Space as a Graph

Using a paver the $\mathbb{C}_{\text{free}}$ space :

$$\mathbb{C}_{\text{free}} = f_{\mathbf{a}}^{-1}(\mathbb{S}) \cap f_{\mathbf{b}}^{-1}(\overline{\mathbb{S}})$$

can be enclose between two sets :

$$\mathbb{C}^{-} \subset \mathbb{C}_{\text{free}} \subset \mathbb{C}^{+}$$

A graph is build while bisecting and contracting boxes:

- node represents box
- when boxes have over-lapping face, an edge joints vertices.

Cameleon's algorithm (L. Jaulin 2001)

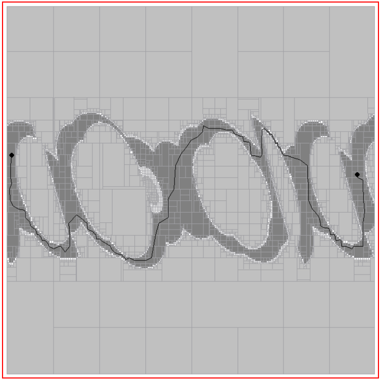
If the goal is to find a solution, the cameleon's algorithm can be used to speed up the computation.

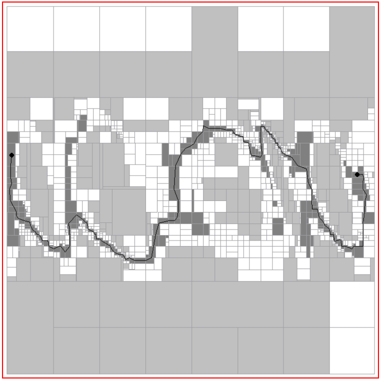
Cameleon's Algorithm

- Find a path through $\mathbb{C}^+$ from x_0 to x_{goal}
- Bisect and separate all boxes which belong to $\Delta\mathbb{C}_{free}$
- Repeat until the path is completely included in $\mathbb{C}^-$

The cost of a bisection is huge compare to Shortest Path Algorithm. Very efficient when no solution exist.

Results





Performances

Algorithm	time (ms)	number of bisection	graph's size	epsilon
SIVIA	15490	33703	29304	0.01
SIVIA	6183	7169	6137	0.05
Cameleon	1151	1266	1438	0.01

Conclusion

The wire loop game example:

- illustrates the possibility of doing path planning with intervals
- demonstrates how simple it could be using pyIbex/Python.

However it needs to be compared with existing methods on higher dimensional space.

Questions ?