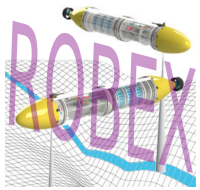
The Codac library

Simon Rohou, Benoît Desrochers

ENSTA Bretagne, Lab-STICC, UMR CNRS 6285, Brest, France

SWIM 2022
19th July 2022



Section 1

Codac in a nutshell

Codac in a nutshell

Domains (wrappers)

- for reals $x \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$: intervals $[x]$ and boxes $[\mathbf{x}]$ (IBEX)
- for trajectories $x(\cdot) : \mathbb{R} \to \mathbb{R}$: tubes $[x](\cdot)$

Codac in a nutshell

Domains (wrappers)

- for reals $x \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$: intervals $[x]$ and boxes $[\mathbf{x}]$ (IBEX)
- for trajectories $x(\cdot) : \mathbb{R} \to \mathbb{R}$: tubes $[x](\cdot)$
- for subsets $\mathbb{X} \subset \mathbb{R}^n$: thicksets $\mathbb{X} \in [\mathbb{X}] = [\mathbb{X}^-, \mathbb{X}^+]$

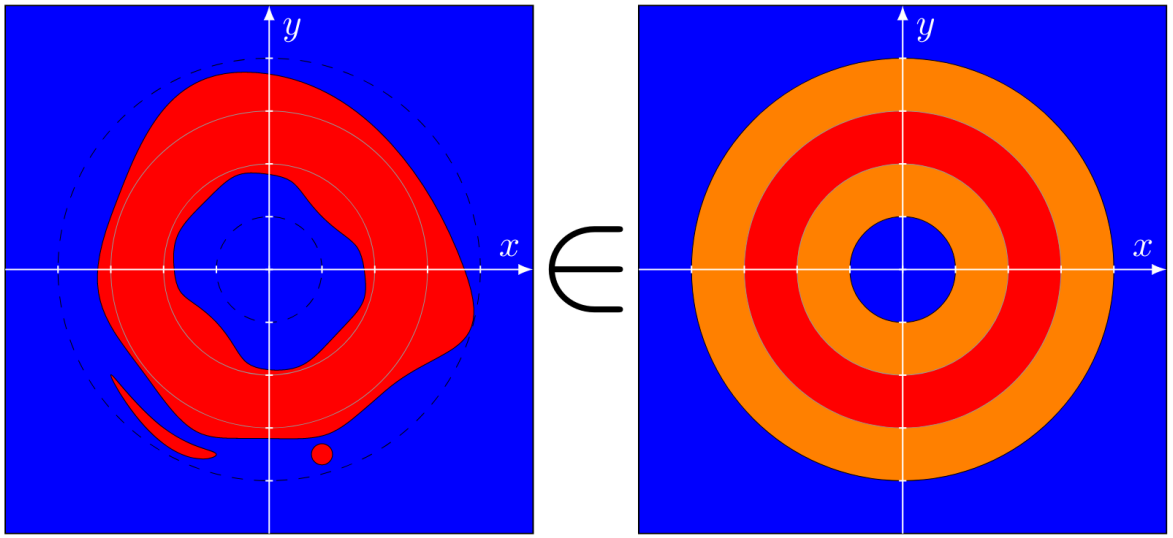


Illustration of a thickset (right-hand side)
for enclosing and uncertain red set (left-hand side)

Thick set inversion
Desrochers, Jaulin. *Artificial Intelligence. Volume 249, Issue C, Pages 1-18*, 2017

Codac in a nutshell

Domains (wrappers)

- for reals $x \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$: intervals $[x]$ and boxes $[\mathbf{x}]$ (IBEX)
- for trajectories $x(\cdot) : \mathbb{R} \to \mathbb{R}$: tubes $[x](\cdot)$
- for subsets $\mathbb{X} \subset \mathbb{R}^n$: thicksets $\mathbb{X} \in [\mathbb{X}] = [\mathbb{X}^-, \mathbb{X}^+]$
- *etc.*

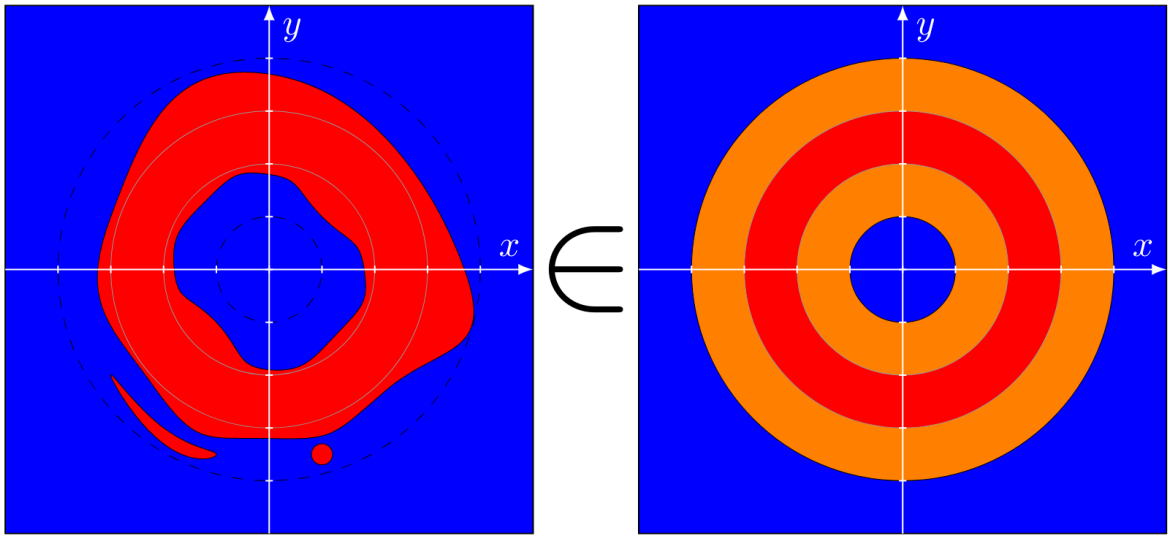
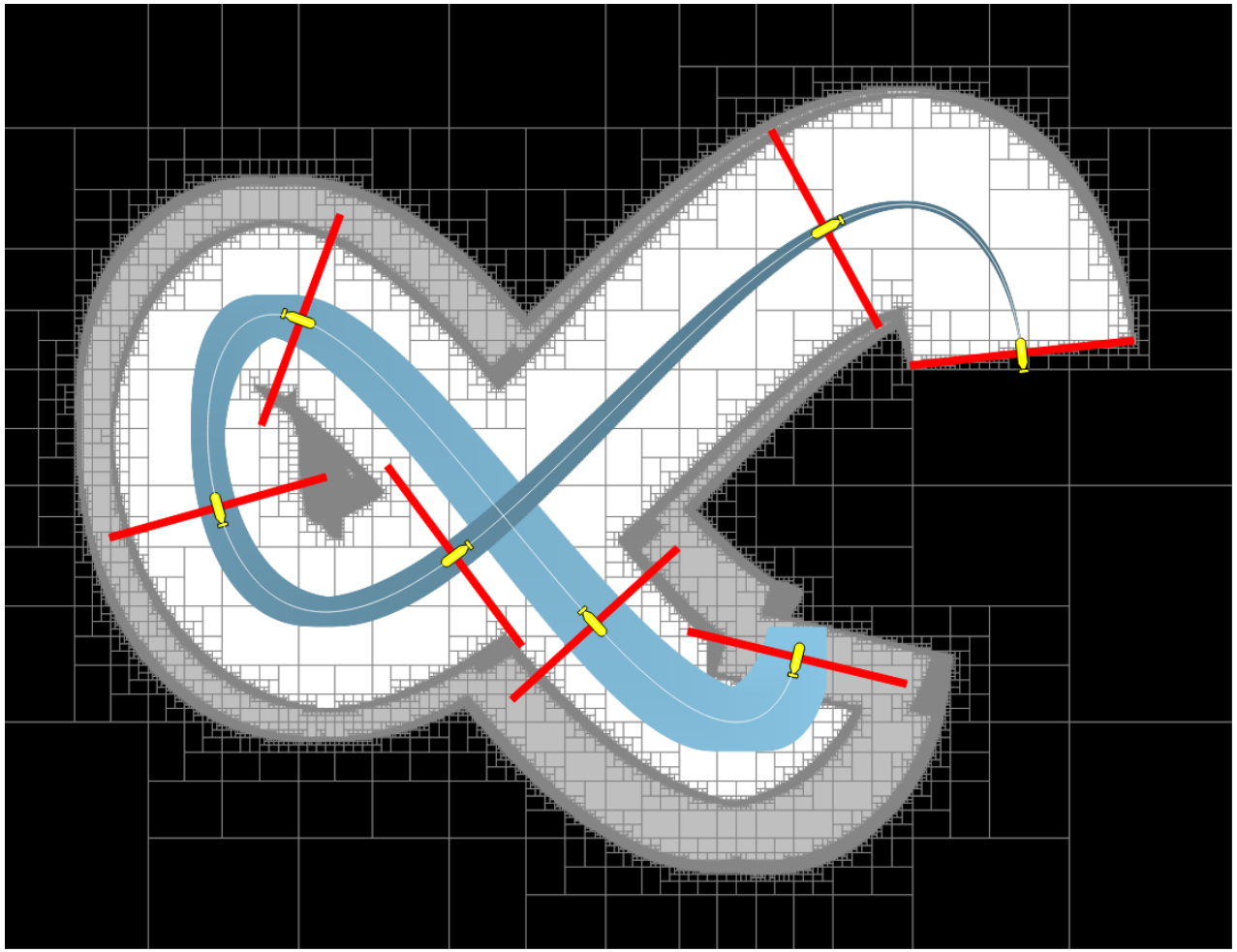


Illustration of a thickset (right-hand side)
for enclosing and uncertain red set (left-hand side)

Thick set inversion
Desrochers, Jaulin. *Artificial Intelligence. Volume 249, Issue C, Pages 1-18*, 2017

Codac in a nutshell

Example of tubes and thicksets



Computing a Guaranteed Approximation of the Zone Explored by a Robot
Desrochers, Jaulin. *IEEE Transaction on Automatic Control. Volume 62, Issue 1, pages 425-430*, 2017

Codac in a nutshell

Codac: Catalog Of Domains And Contractors

Several types of **domains**:

- `Interval`, `IntervalVector`, `IntervalMatrix` (from IBEX)
- `Tube`, `TubeVector`, `Slice`
- `Thickset`
- ...

Codac in a nutshell

Codac: Catalog Of Domains And Contractors

Several types of **domains**:

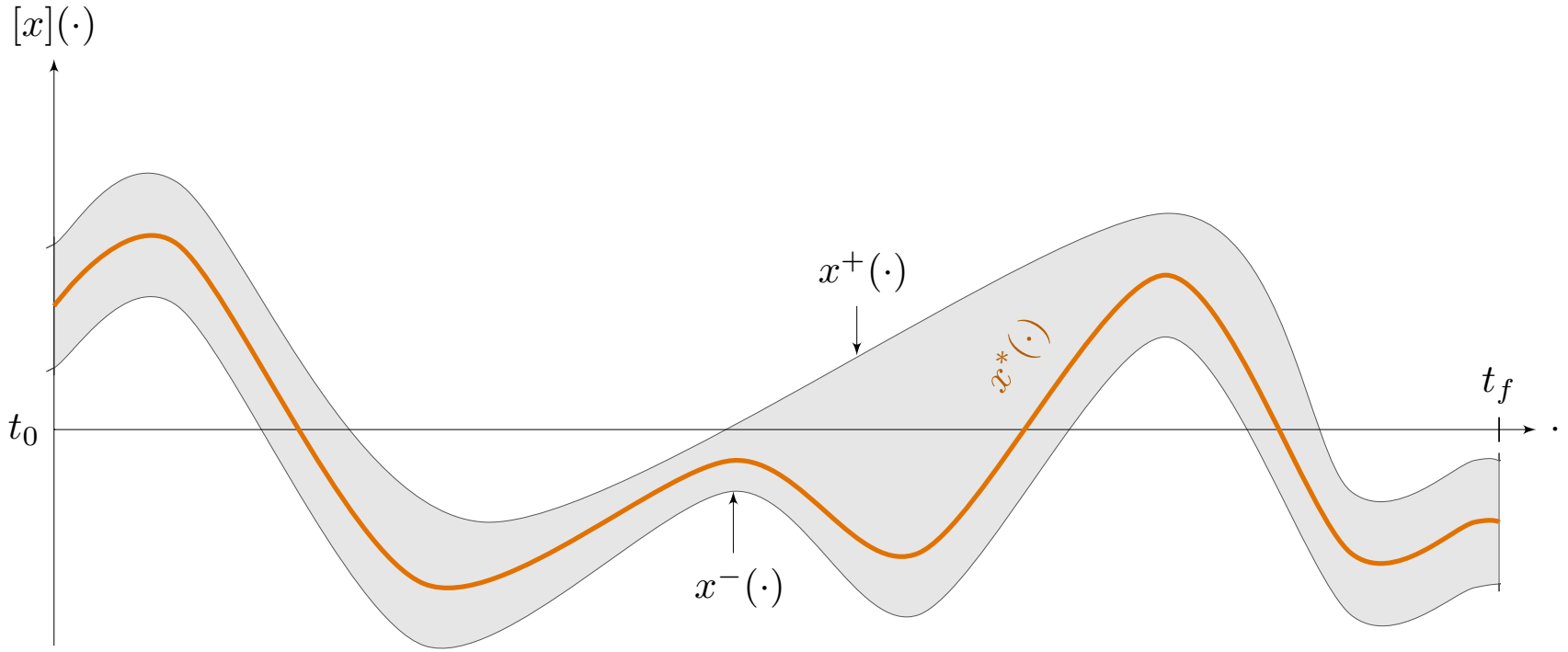
- `Interval`, `IntervalVector`, `IntervalMatrix` (from IBEX)
- `Tube`, `TubeVector`, `Slice`
- `Thickset`
- ...

Contractors for various constraints:

- non-linear constraints $\mathbf{f}(\mathbf{x}) = \mathbf{0}$
- geometric constraints: distance, polar equation, circles, ...
- differential equations: $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$
- time uncertainties: $\mathbf{y} = \mathbf{x}(t), \quad \text{with } t \in [t]$
- delays: $x(t) = y(t - \tau)$
- ...

Codac in a nutshell

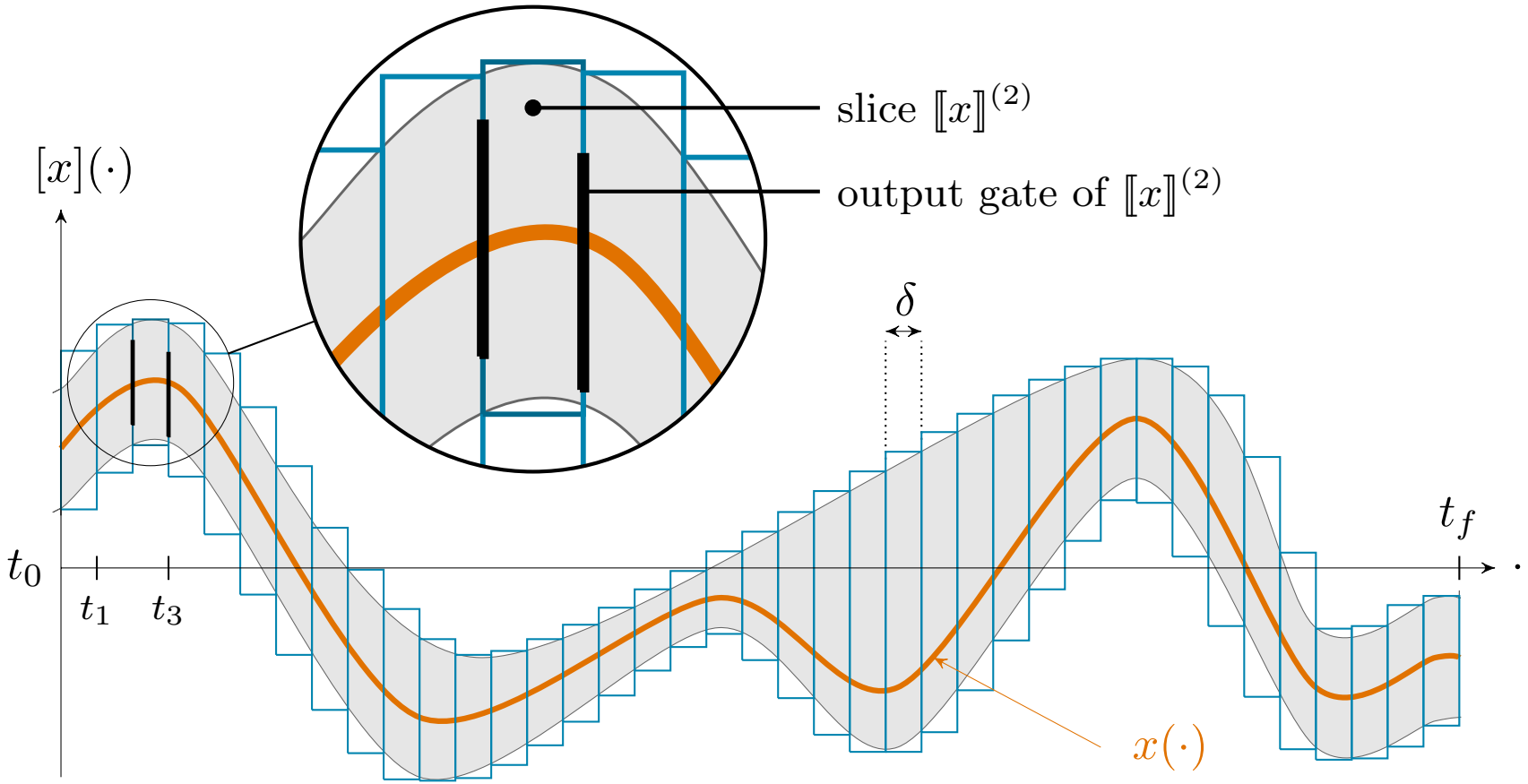
Domains for trajectories: tubes



Example of scalar tube: interval of two trajectories

Codac in a nutshell

Domains for trajectories: tubes



Computer implementation (http://codac.io)

Codac in a nutshell

Example of optimal contractors for the $\mathcal{L}_{\text{polar}}$ constraint

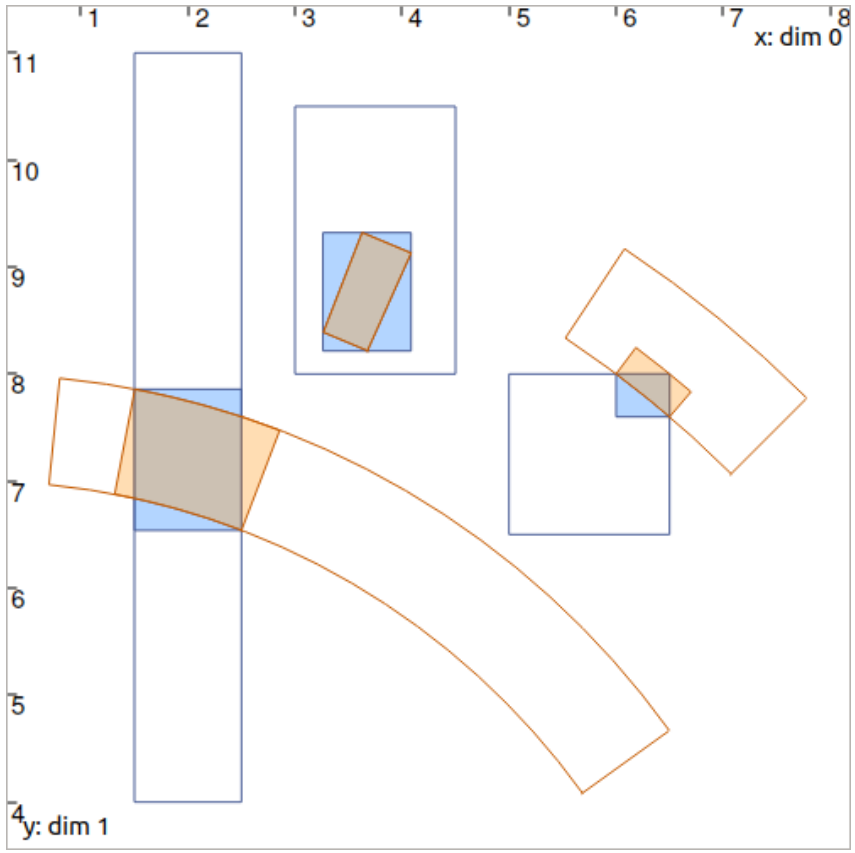
$$\mathcal{L}_{\text{polar}} : \begin{cases} x = \rho \cos\theta \\ y = \rho \sin\theta \end{cases}$$

Optimally dealt with by:

$$\mathcal{C}_{\text{polar}}\big([x],[y],[\rho],[\theta]\big)$$

Using Codac:

```
x = Interval(..)
y = Interval(..)
r = Interval(..)
theta = Interval(..)

ctc.polar.contract(x,y,r,theta)
```



A Minimal contractor for the Polar equation: application to robot localization
Desrochers, Jaulin. *Engineering Applications of Artificial Intelligence, 55(Supplement C):83–92*, 2016

Codac in a nutshell

The library is open source and available:

- in Python and C++
- on Linux, Windows, MacOS systems
- from official packages:
  Python package: `pip install codac`
  Debian in progress..: `sudo apt install libcodac`

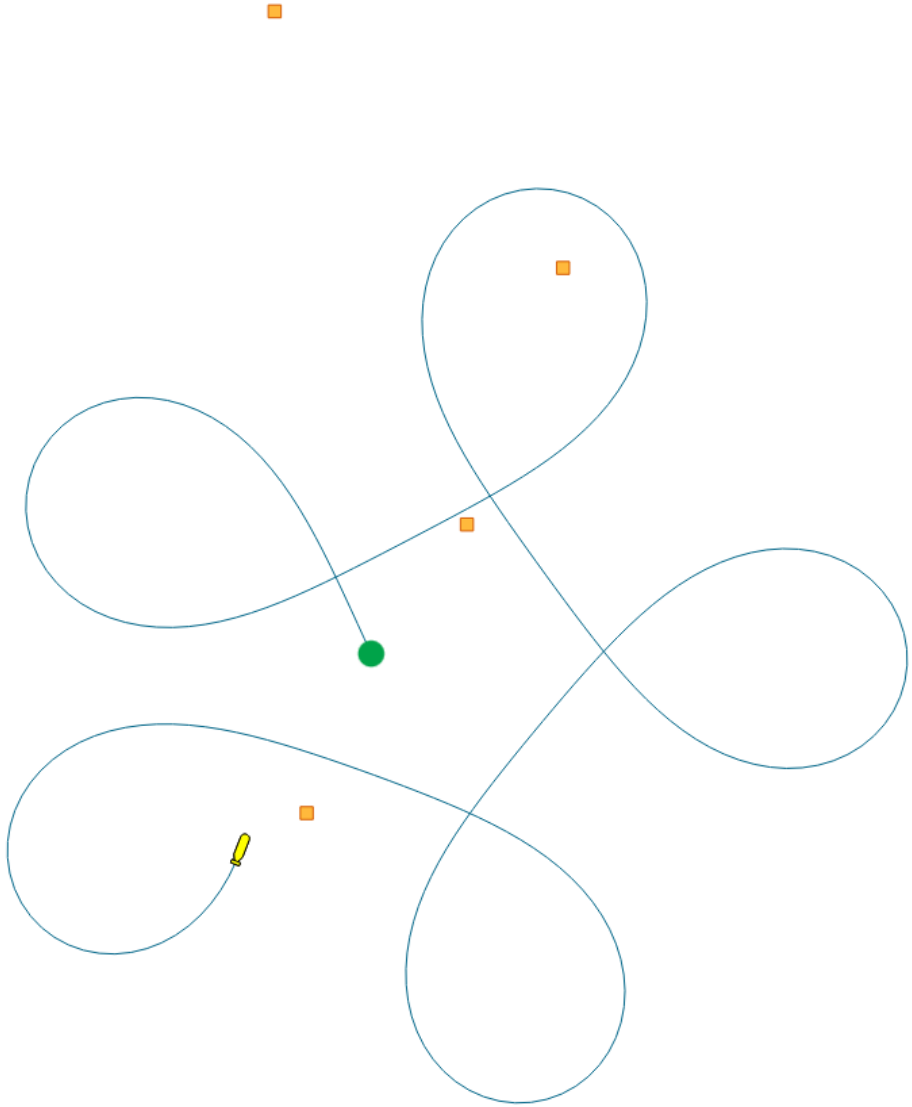
http://www.codac.io

Section 2

Application: range-only SLAM

Application: range-only SLAM

Simultaneous Localization And Mapping



Application: range-only SLAM

Formalization

SLAM: Simultaneous Localization And Mapping.
Classically, we have:

$$\begin{cases} \mathbf{x}(0) = \mathbf{0} & (\text{initial state}) \\ \dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) & (\text{evolution}) \end{cases}$$

With:

- $\mathbf{x}$: state vector (position, heading, ...)
- $\mathbf{u}$: input vector (command)
- $\mathbf{f}$: *evolution* function

Application: range-only SLAM

Formalization

SLAM: Simultaneous Localization And Mapping.
Classically, we have:

$$\begin{cases} \mathbf{x}(0) = \mathbf{0} & (\text{initial state}) \\ \dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) & (\text{evolution}) \\ y_i = g\big(\mathbf{x}(t_i), \mathbf{b}_j\big) & (\text{observations}) \end{cases}$$

With:

- $\mathbf{x}$: state vector (position, heading, ...)
- $\mathbf{u}$: input vector (command)
- $\mathbf{f}$: *evolution* function
- g: *observation* function (scalar, distance equation)
- y_i: scalar measurements (at t_i) (distance values)
- $\mathbf{b}_j$: unknown position of a landmark

Application: range-only SLAM

Formalization

SLAM: Simultaneous Localization And Mapping.
Classically, we have:

$$\begin{cases} \mathbf{x}(0) = \mathbf{0} & \text{(initial state)} \\ \dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) & \text{(evolution)} \\ y_i = g\big(\mathbf{x}(t_i), \mathbf{b}_j\big) & \text{(observations)} \end{cases}$$

With:

- $\mathbf{x}$: state vector (position, heading, ...)
- $\mathbf{u}$: input vector (command)
- $\mathbf{f}$: *evolution* function
- g: *observation* function (scalar, distance equation)
- y_i: scalar measurements (at t_i) (distance values)
- $\mathbf{b}_j$: unknown position of a landmark

Application: range-only SLAM

Involved variables and domains

Variables:

- reals: $y_i \in \mathbb{R}$
- vectors: $\mathbf{b}_j \in \mathbb{R}^2$
- trajectories: $\mathbf{x}(\cdot) : \mathbb{R} \to \mathbb{R}^n$

Application: range-only SLAM

Involved variables and domains

Variables:

- reals: $y_i \in \mathbb{R}$
- vectors: $\mathbf{b}_j \in \mathbb{R}^2$
- trajectories: $\mathbf{x}(\cdot) : \mathbb{R} \to \mathbb{R}^n$

Domains (envelopes) of the variables:

- intervals: $[y_i] \in \mathbb{IR}$
- boxes: $[\mathbf{b}_j] \in \mathbb{IR}^2$
- tubes: $[\mathbf{x}](\cdot) : \mathbb{R} \to \mathbb{IR}^n$

Application: range-only SLAM

Decomposition of the problem

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Application: range-only SLAM

Decomposition of the problem

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Elementary constraints:

- $\mathbf{v}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \rightarrow$ algebraic constraint $\rightarrow \mathcal{L}_{\mathbf{f}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$

Application: range-only SLAM

Decomposition of the problem

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Elementary constraints:

- $\mathbf{v}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \rightarrow$ algebraic constraint $\rightarrow \mathcal{L}_{\mathbf{f}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \rightarrow$ derivative constraint $\rightarrow \mathcal{L}_{\text{deriv}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$

Application: range-only SLAM

Decomposition of the problem

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Elementary constraints:

- $\mathbf{v}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \rightarrow$ algebraic constraint $\rightarrow \mathcal{L}_{\mathbf{f}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \rightarrow$ derivative constraint $\rightarrow \mathcal{L}_{\text{deriv}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\mathbf{p}_i = \mathbf{x}_{1,2}(t_i) \rightarrow$ evaluation constraint $\rightarrow \mathcal{L}_{\text{eval}}\big(t_i, \mathbf{p}_i, \mathbf{x}_{1,2}(\cdot)\big)$

Application: range-only SLAM

Decomposition of the problem

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Elementary constraints:

- $\mathbf{v}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \rightarrow$ algebraic constraint $\rightarrow \mathcal{L}_{\mathbf{f}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \rightarrow$ derivative constraint $\rightarrow \mathcal{L}_{\text{deriv}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\mathbf{p}_i = \mathbf{x}_{1,2}(t_i) \rightarrow$ evaluation constraint $\rightarrow \mathcal{L}_{\text{eval}}\big(t_i, \mathbf{p}_i, \mathbf{x}_{1,2}(\cdot)\big)$
- $y_i = g\big(\mathbf{p}_i, \mathbf{b}_j\big) \rightarrow$ distance constraint $\rightarrow \mathcal{L}_{\text{dist}}\big(\mathbf{p}_i, \mathbf{b}_j, y_i\big)$

Application: range-only SLAM

Decomposition of the problem

System:

$\mathbf{v}(\cdot)$ and $\mathbf{p}_i$ are intermediate variables

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Elementary constraints:

- $\mathbf{v}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \rightarrow$ algebraic constraint $\rightarrow \mathcal{L}_{\mathbf{f}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \rightarrow$ derivative constraint $\rightarrow \mathcal{L}_{\text{deriv}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\mathbf{p}_i = \mathbf{x}_{1,2}(t_i) \rightarrow$ evaluation constraint $\rightarrow \mathcal{L}_{\text{eval}}\big(t_i, \mathbf{p}_i, \mathbf{x}_{1,2}(\cdot)\big)$
- $y_i = g\big(\mathbf{p}_i, \mathbf{b}_j\big) \rightarrow$ distance constraint $\rightarrow \mathcal{L}_{\text{dist}}\big(\mathbf{p}_i, \mathbf{b}_j, y_i\big)$

Application: range-only SLAM

Decomposition of the problem

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

$\mathbf{v}(\cdot)$ and $\mathbf{p}_i$ are intermediate variables

Note: some symbolic solver could break down such problem automatically.

Elementary constraints:

- $\mathbf{v}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \rightarrow$ algebraic constraint $\rightarrow \mathcal{L}_{\mathbf{f}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \rightarrow$ derivative constraint $\rightarrow \mathcal{L}_{\text{deriv}}\big(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\big)$
- $\mathbf{p}_i = \mathbf{x}_{1,2}(t_i) \rightarrow$ evaluation constraint $\rightarrow \mathcal{L}_{\text{eval}}\big(t_i, \mathbf{p}_i, \mathbf{x}_{1,2}(\cdot)\big)$
- $y_i = g\big(\mathbf{p}_i, \mathbf{b}_j\big) \rightarrow$ distance constraint $\rightarrow \mathcal{L}_{\text{dist}}\big(\mathbf{p}_i, \mathbf{b}_j, y_i\big)$

Application: range-only SLAM

Using Codac

1. Defining the domains:

```
# Creating the state tube:
dt = 0.01
tdomain = Interval(0,6)          # temporal domain
x = TubeVector(tdomain, dt, 4) # dim. 4
# etc.
```

.

Application: range-only SLAM

Using Codac

1. Defining the domains:

```
# Creating the state tube:
dt = 0.01
tdomain = Interval(0,6)          # temporal domain
x = TubeVector(tdomain, dt, 4) # dim. 4
# etc.
```

2. Defining contractors:

```
ctc.deriv   # Some contractors are objects
ctc.eval    # already defined in the library
ctc.polar
ctc.dist

# Other contractors can be built from analytical expressions:
ctc_plus = CtcFunction(Function("x","y","z","x+y-z"))
# CtcFunction is related to constraints under the form f(x,..)=0
```

Application: range-only SLAM

Using Codac

3. Build a Contractor Network

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Contractor programming:

1. $\mathcal{C}_{\mathbf{f}}\big([\mathbf{x}](\cdot), [\mathbf{v}](\cdot)\big)$
2. $\mathcal{C}_{\text{deriv}}\big([\mathbf{x}](\cdot), [\mathbf{v}](\cdot)\big)$
3. $\mathcal{C}_{\text{eval}}\big([t_i], [\mathbf{p}_i], [\mathbf{x}_{1,2}](\cdot)\big)$
4. $\mathcal{C}_{\text{dist}}\big([\mathbf{p}_i], [\mathbf{b}_j], [y_i]\big)$

Application: range-only SLAM

Using Codac

3. Build a Contractor Network

System:

$$\begin{cases} \dot{\mathbf{x}}(\cdot) = \mathbf{f}\big(\mathbf{x}(\cdot)\big) \\ y_i = g\big(\mathbf{x}_{1,2}(t_i), \mathbf{b}_j\big) \end{cases}$$

Contractor programming:

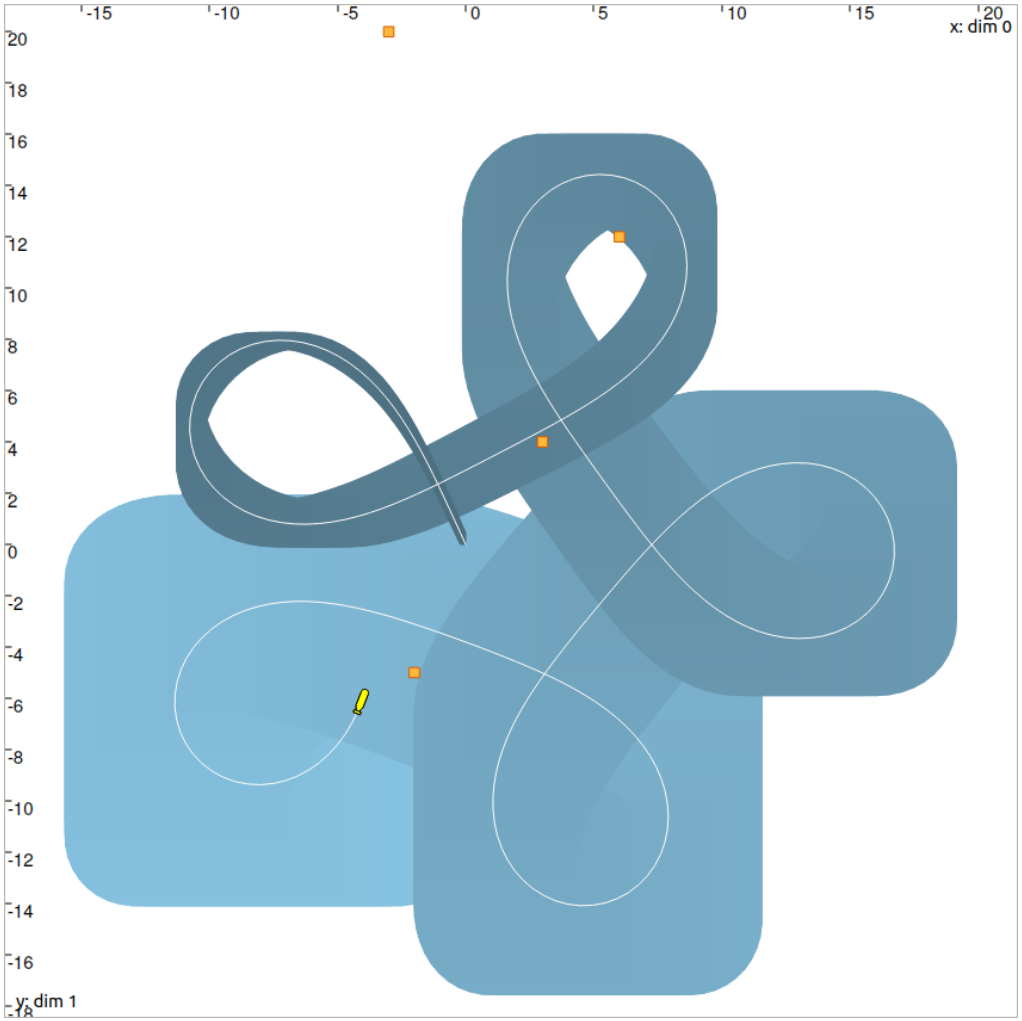
1. $\mathcal{C}_{\mathbf{f}}\big([\mathbf{x}](\cdot), [\mathbf{v}](\cdot)\big)$
2. $\mathcal{C}_{\text{deriv}}\big([\mathbf{x}](\cdot), [\mathbf{v}](\cdot)\big)$
3. $\mathcal{C}_{\text{eval}}\big([t_i], [\mathbf{p}_i], [\mathbf{x}_{1,2}](\cdot)\big)$
4. $\mathcal{C}_{\text{dist}}\big([\mathbf{p}_i], [\mathbf{b}_j], [y_i]\big)$

Using Codac:

```
cn = ContractorNetwork()

cn.add(ctc.polar, [v[0],v[1],x[3],x[2]])  #1
cn.add(ctc.deriv, [x,v])                  #2

for i in range(len(v_t)):
  pi = IntervalVector(4)
  cn.add(ctc.eval, [t[i],pi,x])           #3
  cn.add(ctc.dist, [pi,b[j],y[i]])        #4

cn.contract()                             -
```

Application: range-only SLAM

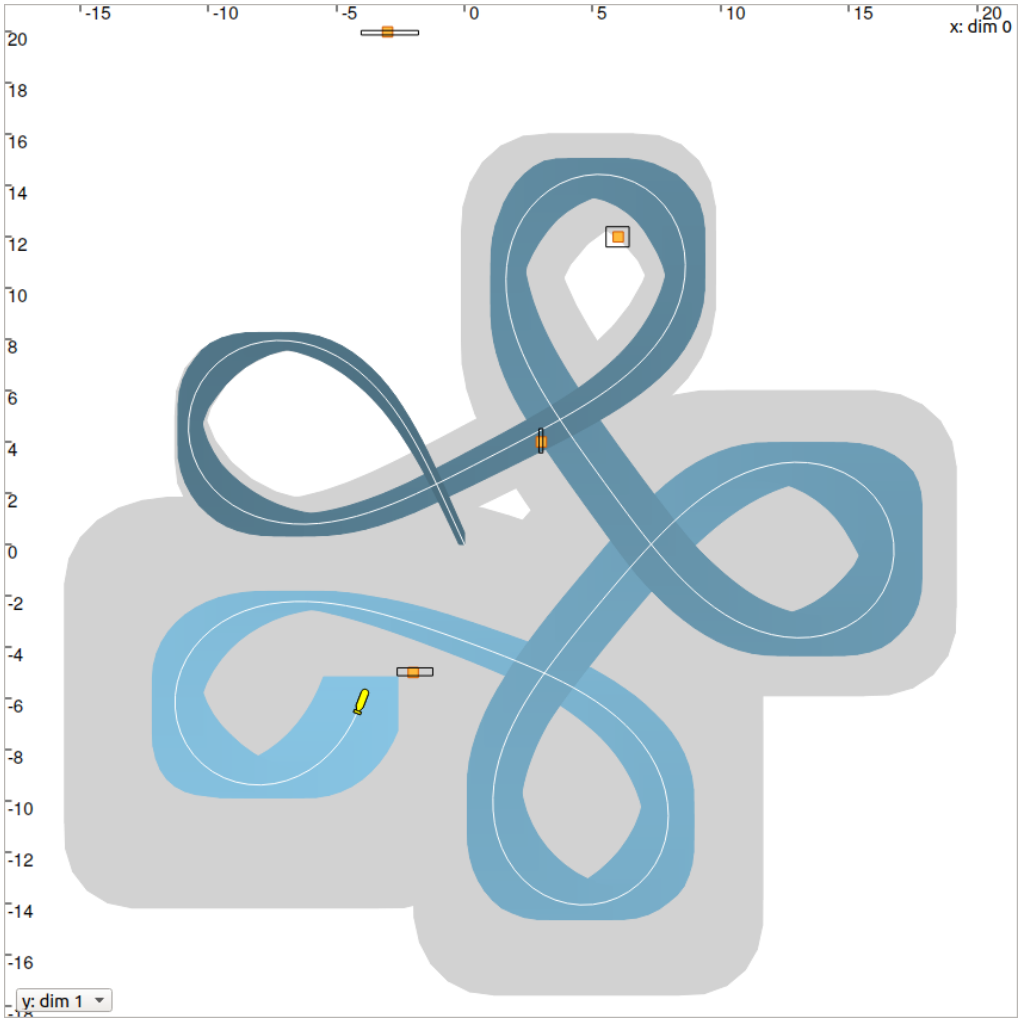
Using Codac




```
cn = ContractorNetwork()

cn.add(ctc.polar, ..
cn.add(ctc.deriv, ..

for i in range(len(v_t)):
  pi = IntervalVector(4)
  cn.add(ctc.eval, ..
  cn.add(ctc.dist, ..
```

Application: range-only SLAM

Using Codac




```
cn = ContractorNetwork()

cn.add(ctc.polar, ..
cn.add(ctc.deriv, ..

for i in range(len(v_t)):
  pi = IntervalVector(4)
  cn.add(ctc.eval, ..
  cn.add(ctc.dist, ..

cn.contract()
```

2399 contractors, 2410 dom.
Computation time: 0.25s

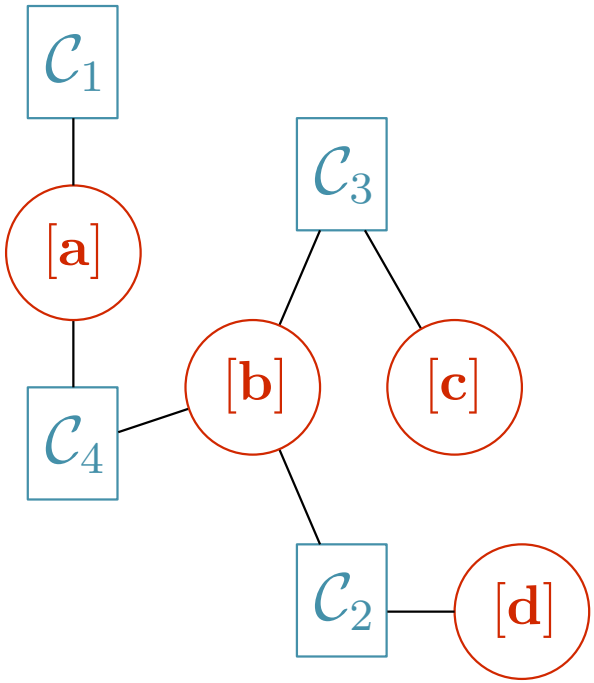
Section 3

Contractor Networks

Contractor Networks

Propagation

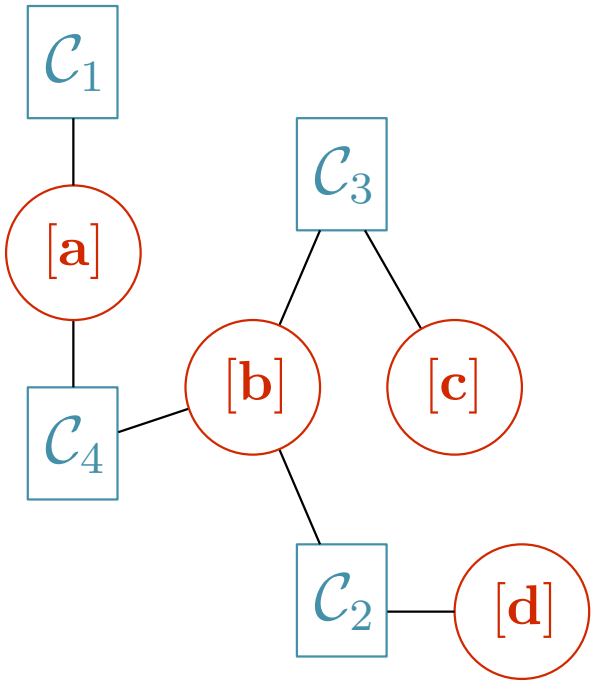
- Contractor Network = graph of Domains and Contractors
- The graph is possibly directed
- The network allows accurate propagation of the contractions
- When the stack of contractor calls is empty, a fixed point is reached



Contractor Networks

Propagation

- Contractor Network = graph of Domains and Contractors
- The graph is possibly directed
- The network allows accurate propagation of the contractions
- When the stack of contractor calls is empty, a fixed point is reached



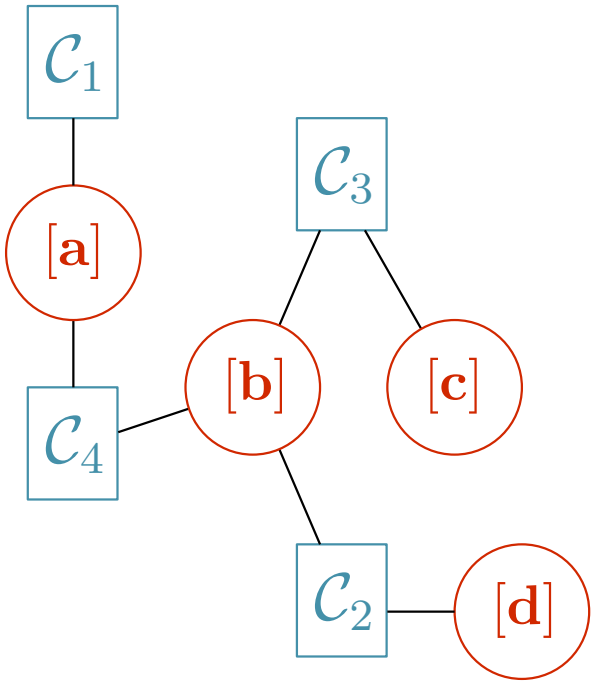
Propagation sequence:

1. $[\mathbf{c}]$ contracted/triggered
2. adding $\mathcal{C}_3$ in stack
3. calling $\mathcal{C}_3$
4. $[\mathbf{b}]$ contracted
5. adding $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$ in stack
6. calling $\mathcal{C}_4$
7. $[\mathbf{b}]$ contracted
8. adding $\mathcal{C}_4$ in stack
9. calling $\mathcal{C}_2$
10. $[\mathbf{d}]$ contracted
11. calling $\mathcal{C}_3$, $\mathcal{C}_4$
12. fixed point

Contractor Networks

Propagation

- Contractor Network = graph of Domains and Contractors
- The graph is possibly directed
- The network allows accurate propagation of the contractions
- When the stack of contractor calls is empty, a fixed point is reached



Propagation sequence:

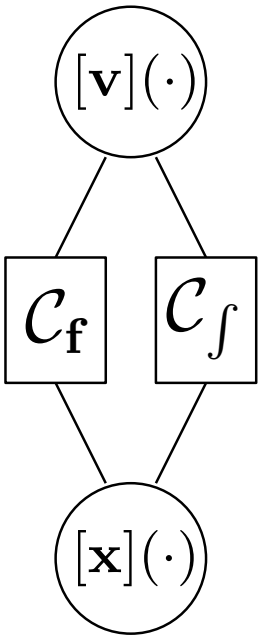
1. $[\mathbf{c}]$ contracted/triggered
2. adding $\mathcal{C}_3$ in stack
3. calling $\mathcal{C}_3$
4. $[\mathbf{b}]$ contracted
5. adding $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$ in stack
6. calling $\mathcal{C}_4$
7. $[\mathbf{b}]$ contracted
8. adding $\mathcal{C}_4$ in stack
9. calling $\mathcal{C}_2$
10. $[\mathbf{d}]$ contracted
11. calling $\mathcal{C}_3$, $\mathcal{C}_4$
12. fixed point

$\Rightarrow$ **heuristic of propagation**

Contractor Networks

Contractor Network involving Tubes

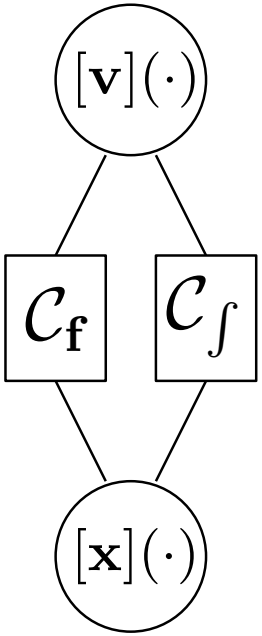
Example, dealing with: $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \iff \dot{\mathbf{x}} = \mathbf{v}\ ,\ \mathbf{v} = \mathbf{f}(\mathbf{x})$

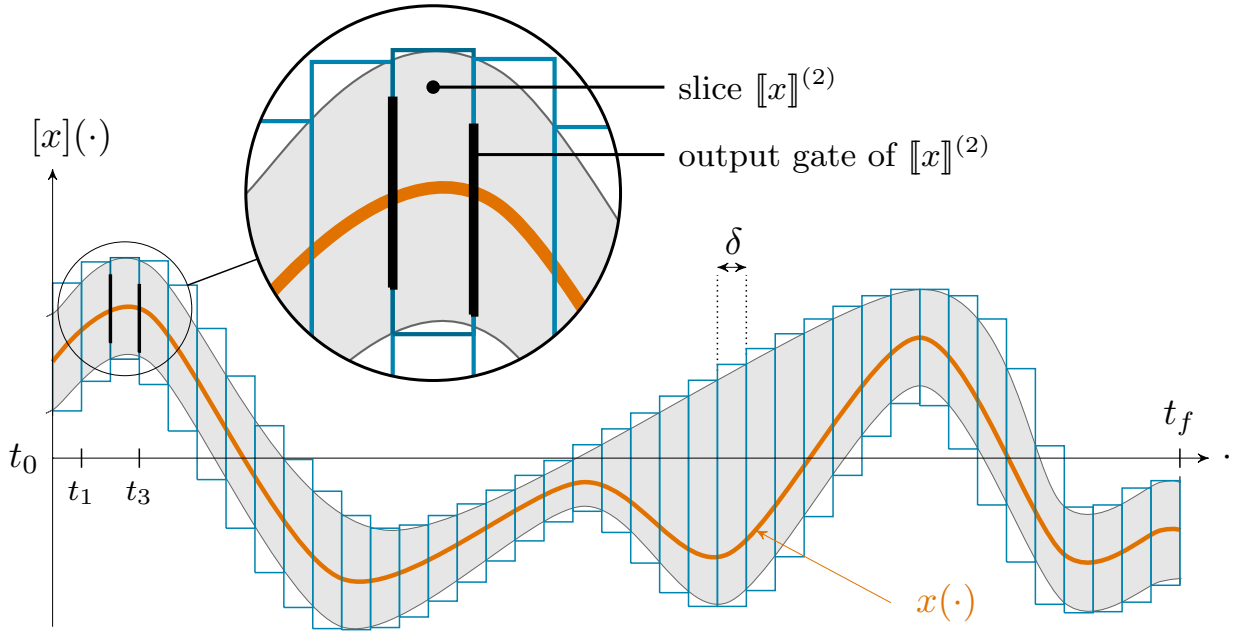


Contractor Networks

Contractor Network involving Tubes

Example, dealing with: $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \iff \dot{\mathbf{x}} = \mathbf{v}\ ,\ \mathbf{v} = \mathbf{f}(\mathbf{x})$

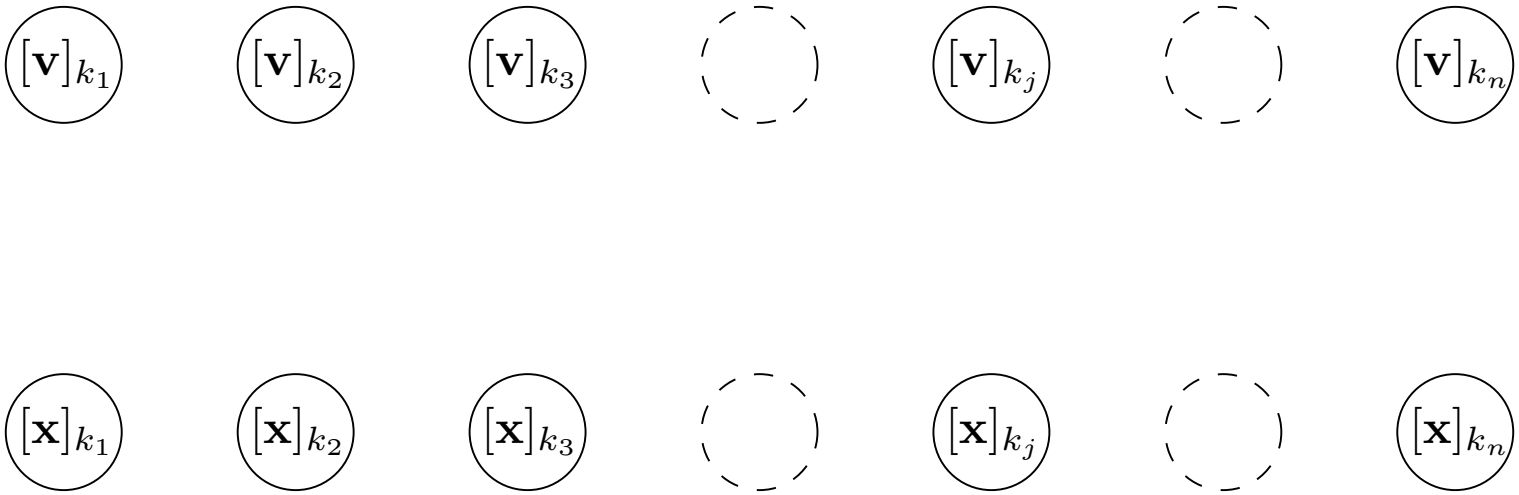




- break down contractors into *micro-contractors* at the level of slices
- $\Rightarrow$ strong densification of the graph

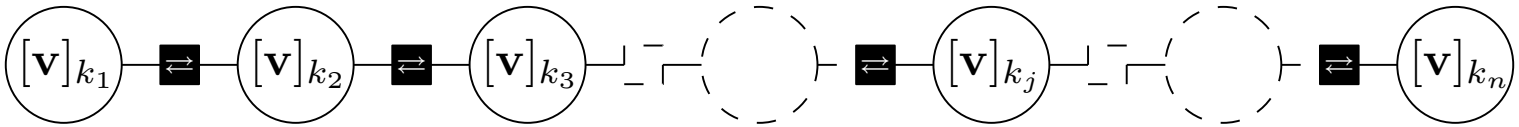
Contractor Networks

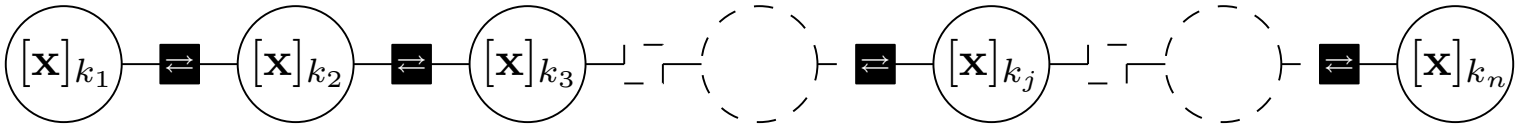
Contractor Network involving Tubes



Contractor Networks

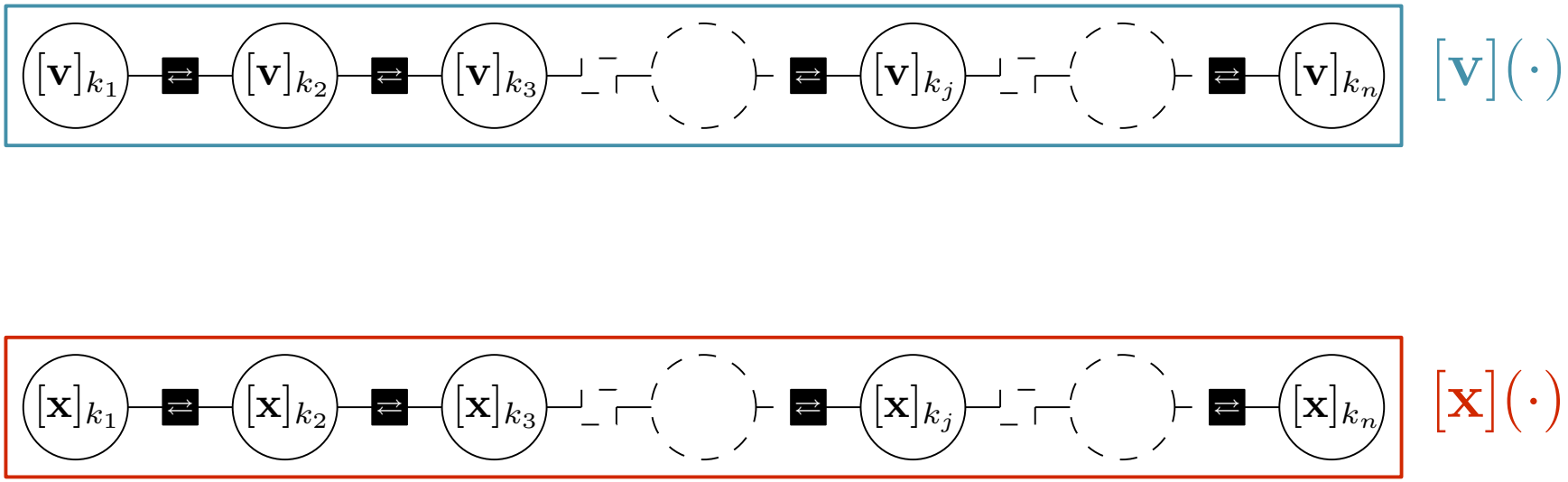
Contractor Network involving Tubes





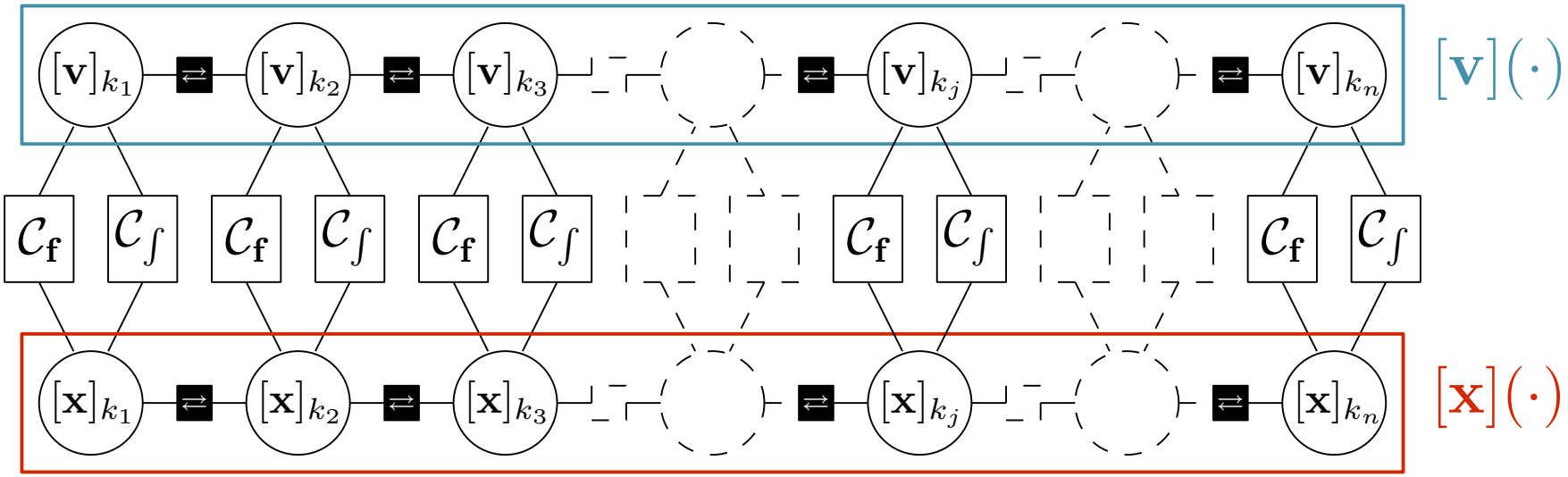
Contractor Networks

Contractor Network involving Tubes



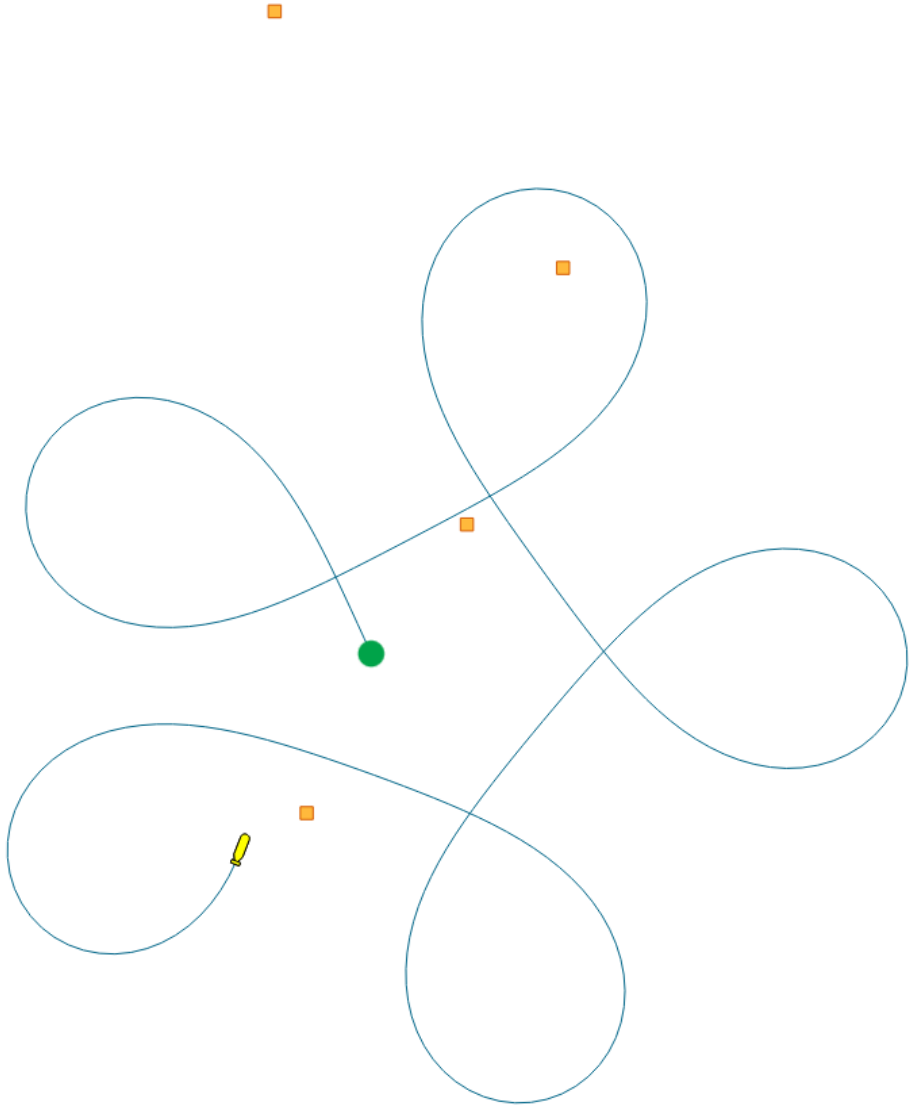
Contractor Networks

Contractor Network involving Tubes



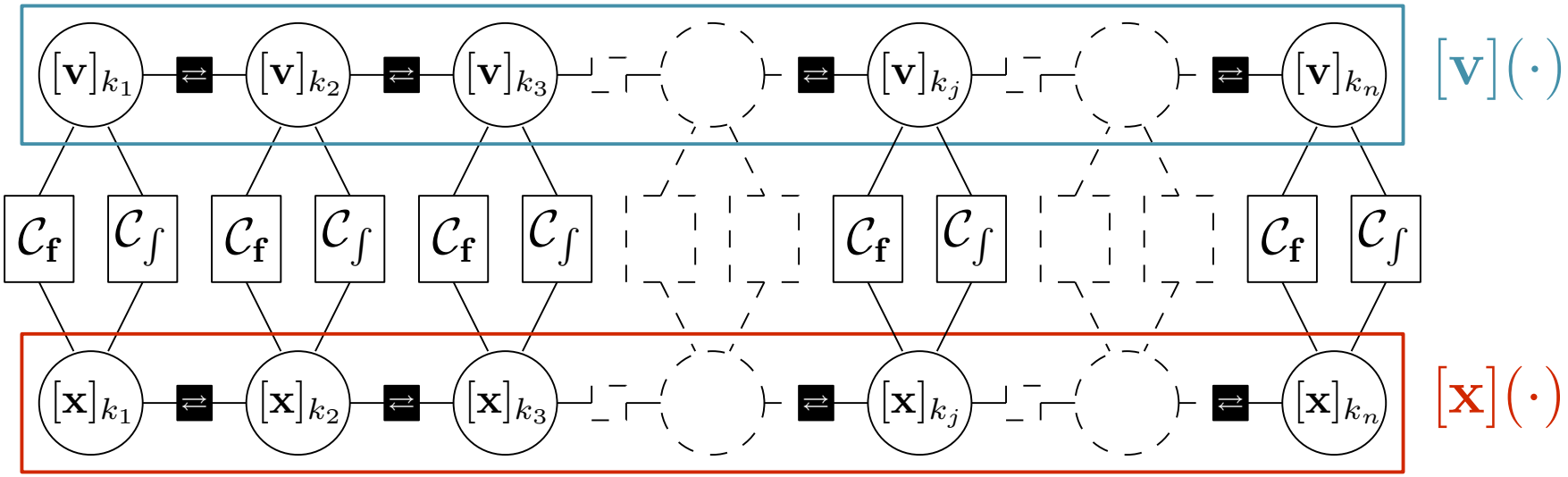
Contractor Networks

Contractor Network for SLAM: SLAM-CN



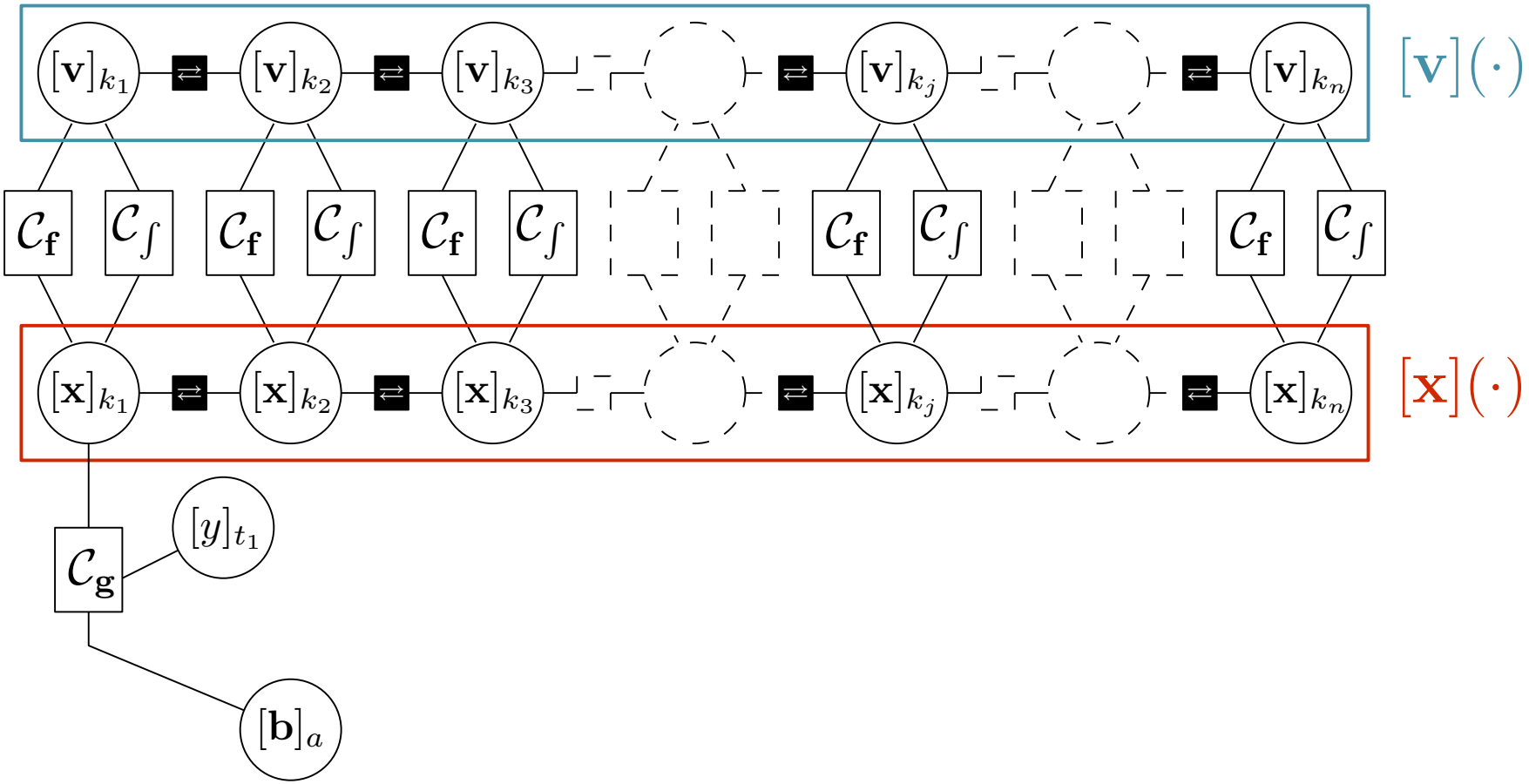
Contractor Networks

Contractor Network for SLAM: SLAM-CN



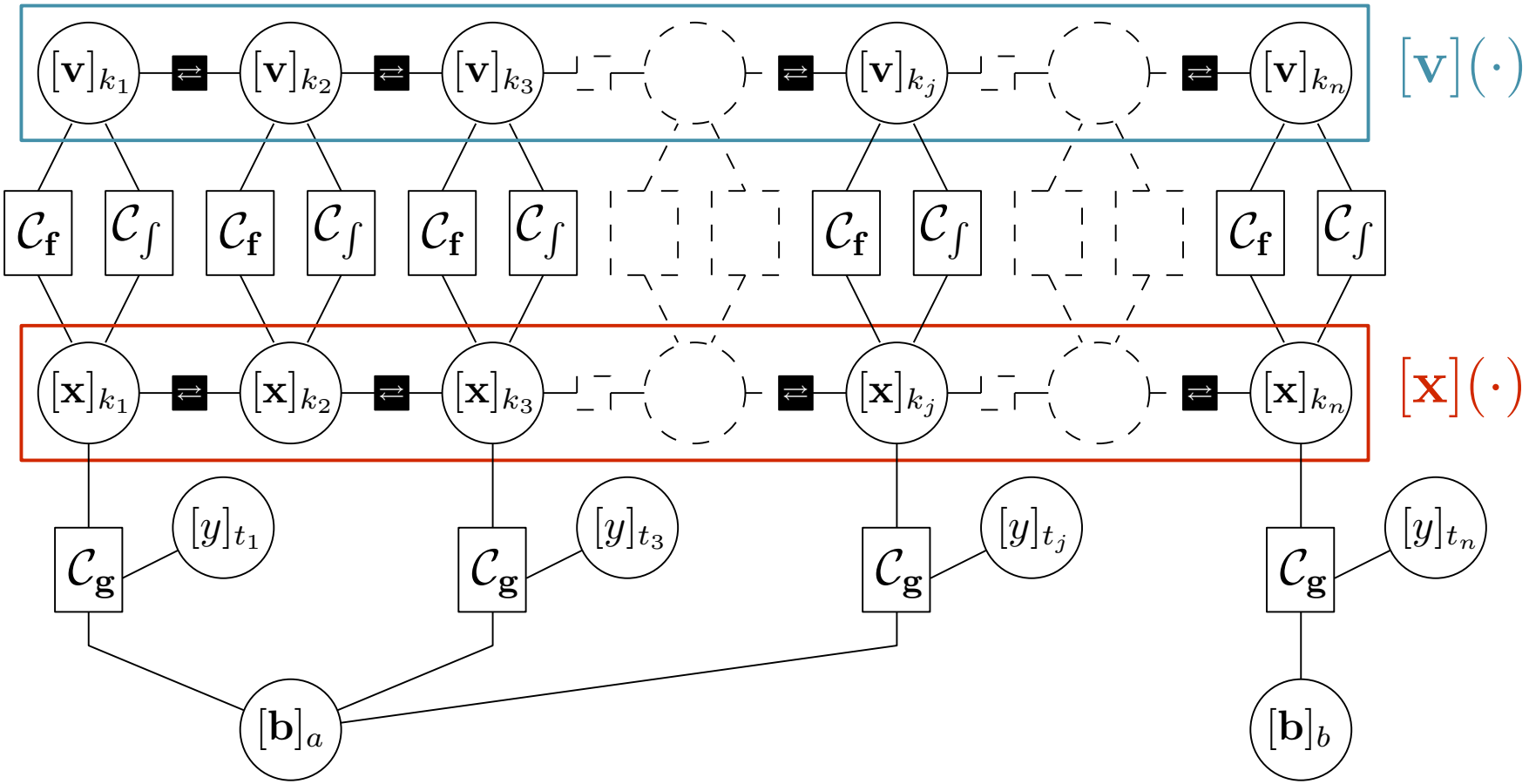
Contractor Networks

Contractor Network for SLAM: SLAM-CN



Contractor Networks

Contractor Network for SLAM: SLAM-CN



Contractor Networks

Programming a SLAM-CN

1. Define domains:
   - intervals, boxes, tubes, ...
   - related to measurements or initialized as $[-\infty, \infty]$

..

Contractor Networks

Programming a SLAM-CN

1. Define domains:
   - intervals, boxes, tubes, ...
   - related to measurements or initialized as $[-\infty, \infty]$
2. Define contractors:
   - use/configure already existing contractors from the library
   - or build own contractors for specific constraints

..

Contractor Networks

Programming a SLAM-CN

1. Define domains:
   - intervals, boxes, tubes, ...
   - related to measurements or initialized as $[-\infty, \infty]$
2. Define contractors:
   - use/configure already existing contractors from the library
   - or build own contractors for specific constraints
3. Build the Contractor Network:

..

Contractor Networks

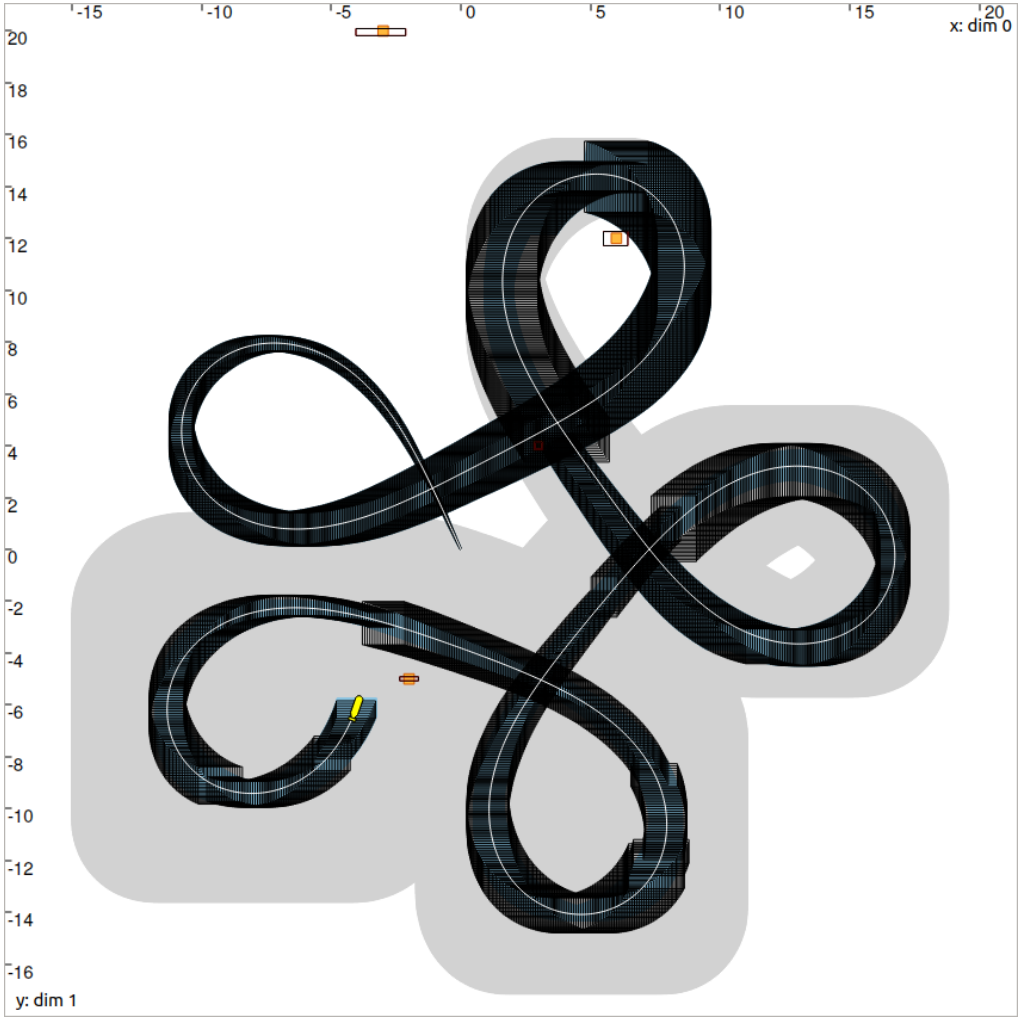
Programming a SLAM-CN

1. Define domains:
   - intervals, boxes, tubes, ...
   - related to measurements or initialized as $[-\infty, \infty]$
2. Define contractors:
   - use/configure already existing contractors from the library
   - or build own contractors for specific constraints
3. Build the Contractor Network:

```
cn = ContractorNetwork()
cn.add(ctc_f, [x,v])
cn.add(ctc.deriv, [x,v])

for i in range(len(v_t)):
  pi = IntervalVector(4)
  cn.add(ctc.eval, [t[i],pi,x]
  cn.add(ctc.dist, [y[i],pi,b[i]])

cn.contract()
```

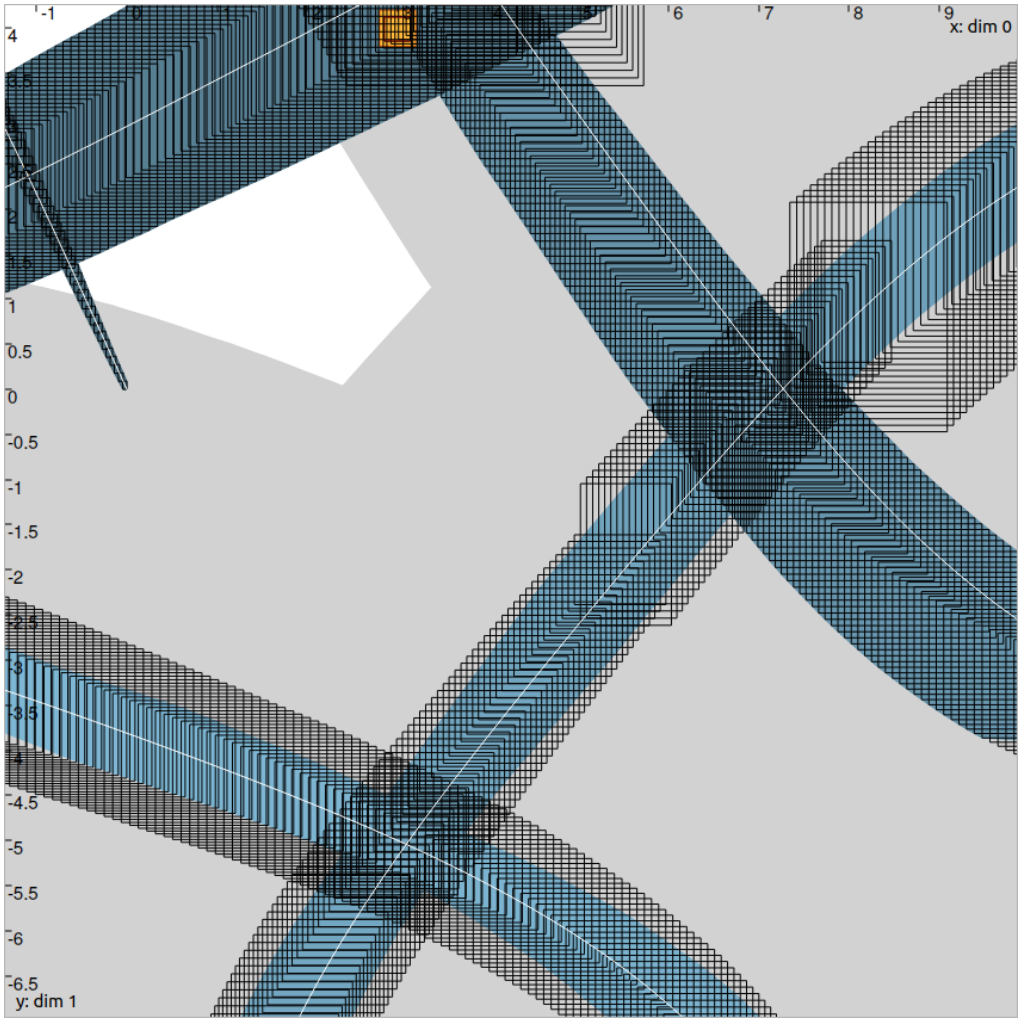
Contractor Networks

SLAM-CN: realtime application



Contractor Networks

SLAM-CN: realtime application



Section 4

Conclusions

Conclusions

Conclusion

Assets of constraint programming coupled with interval analysis:

- **simplicity** of the approach

Conclusions

Conclusion

Assets of constraint programming coupled with interval analysis:

- **simplicity** of the approach
- **reliability** of the results: no solution can be lost

Conclusions

Conclusion

Assets of constraint programming coupled with interval analysis:

- **simplicity** of the approach
- **reliability** of the results: no solution can be lost
- focus on **the *what* instead of the *how***

Conclusions

Conclusion

Assets of constraint programming coupled with interval analysis:

- **simplicity** of the approach
- **reliability** of the results: no solution can be lost
- focus on **the *what* instead of the *how***
- **complex systems** easily handled

Conclusions

Conclusion

Assets of constraint programming coupled with interval analysis:

- **simplicity** of the approach
- **reliability** of the results: no solution can be lost
- focus on **the *what* instead of the *how***
- **complex systems** easily handled

Future work related to Codac:

- extend the *catalog* of contractors
- implement other domains and intermediate wrappers
- ..towards real-time implementations

Conclusions

Conclusion

Assets of constraint programming coupled with interval analysis:

- **simplicity** of the approach
- **reliability** of the results: no solution can be lost
- focus on **the *what* instead of the *how***
- **complex systems** easily handled

Future work related to Codac:

- extend the *catalog* of contractors
- implement other domains and intermediate wrappers
- ..towards real-time implementations

Codac library: open-source C++/Python library providing tools for constraint programming over reals, trajectories and sets

http://www.codac.io

Conclusions

Conclusion

Thanks to the current (and soon) contributors!

- Benoît Desrochers
- Luc Jaulin
- Gilles Chabert
- Auguste Bourgois
- Julien Damers
- Thomas Le Mézo
- Raphael Voges
- Cyril Bouvier
- Andreas Rauh
- Fabrice Le Bars
- Quentin Brateau
- Damien Massé
- Bertrand Neveu
- Peter Franek
- Gilles Trombettoni
- Aaronkumar Ehambram
- Verlein Radwan
- Mohamed Saad Ibn Seddik
- Jonas Soueidan

Section 5

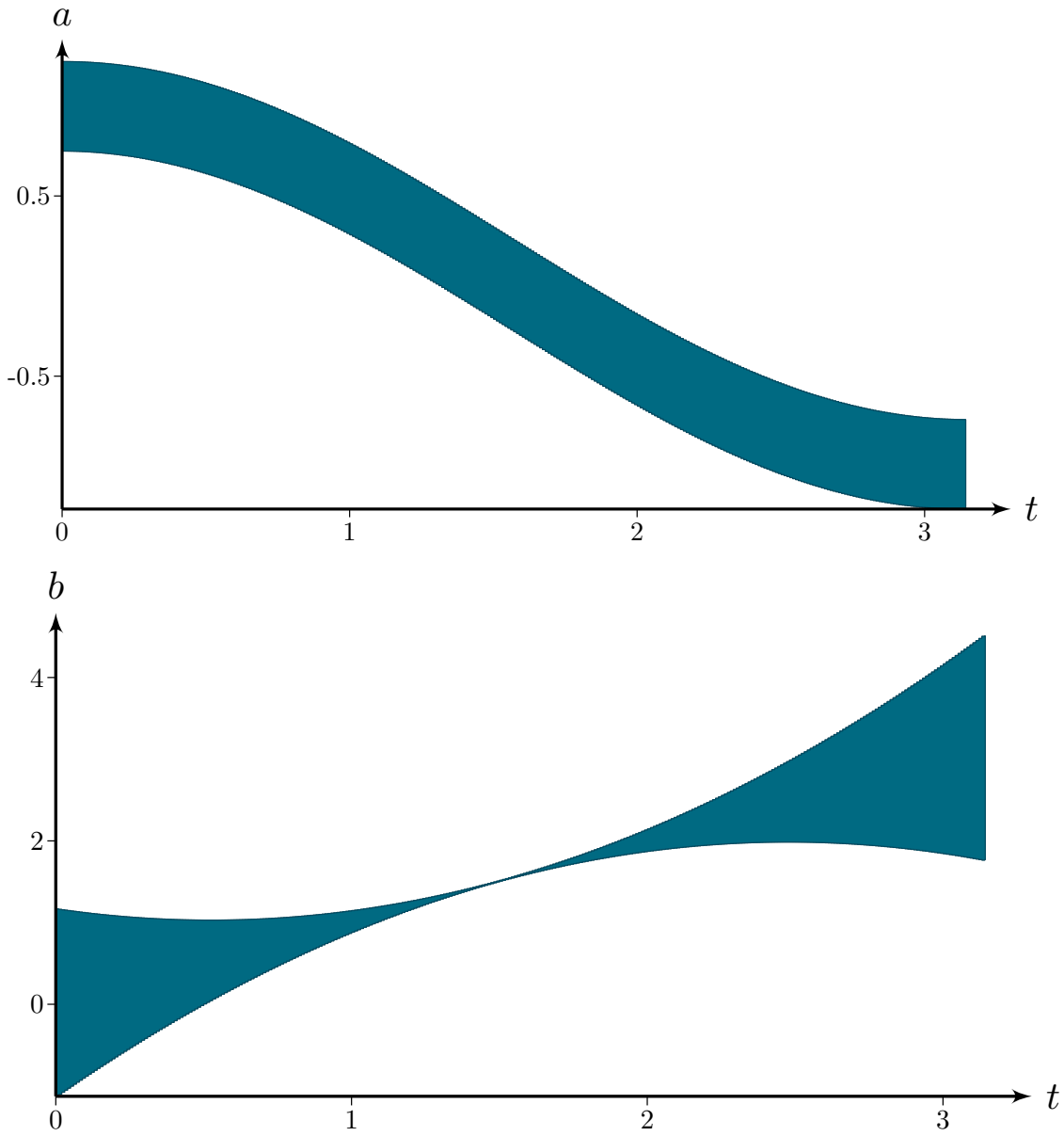
Appendices

Appendices

"Static" constraints

Static constraint:

- $\forall t,\ f(a(t), b(t), \dots) = 0$
- non differential
  (not in the form
  $\dot{a}(t) = b(t)$)
- non inter-temporal
  (not in the form
  $a(t+1) = b(t)$)



Appendices

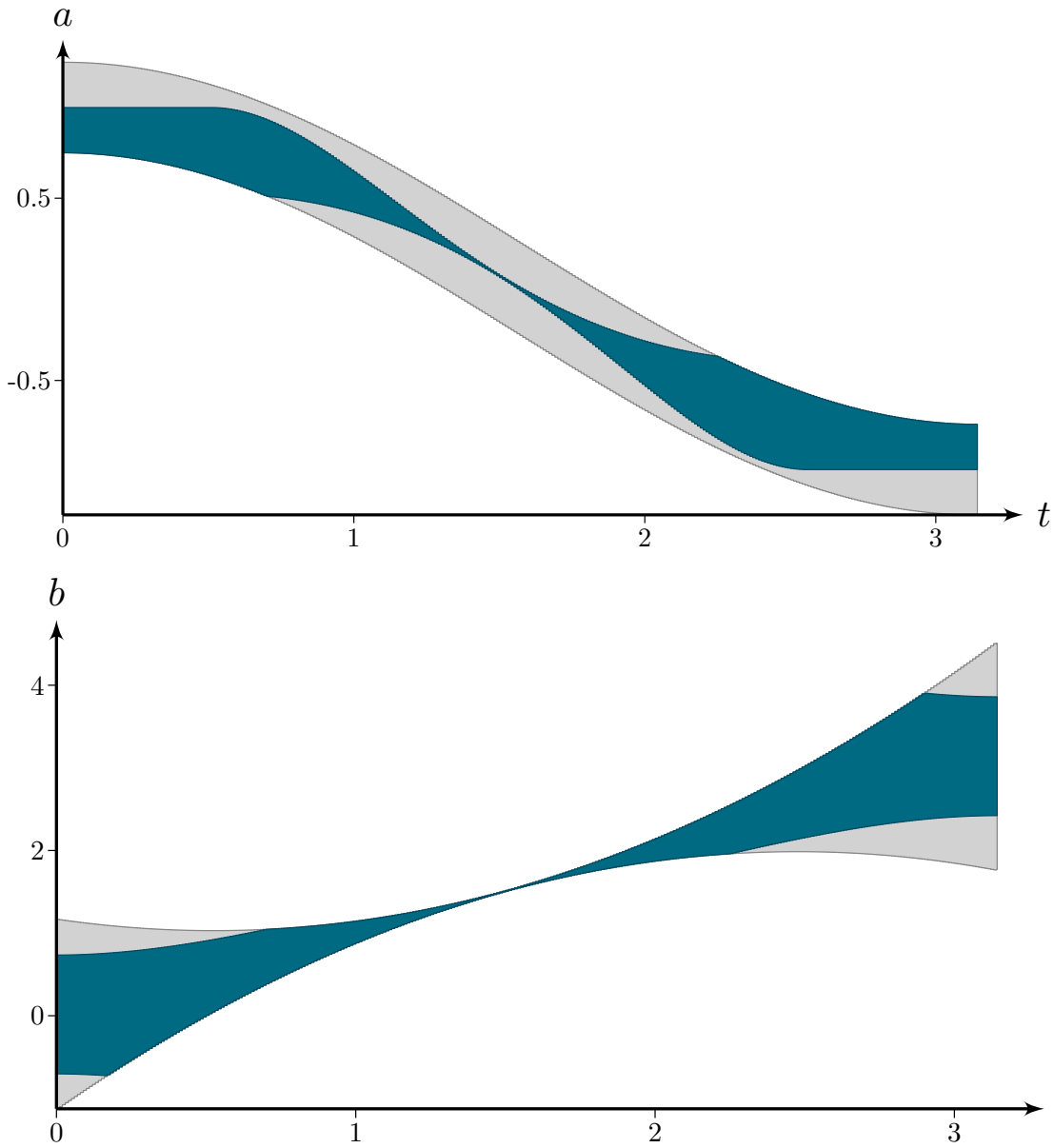
"Static" constraints

Static constraint:

- $\forall t,\ f(a(t), b(t), \dots) = 0$
- non differential
  (not in the form
  $\dot{a}(t) = b(t)$)
- non inter-temporal
  (not in the form
  $a(t+1) = b(t)$)

Example with $a(\cdot) = \cos\big(b(\cdot)\big)$:

$$\mathcal{C}_{\cos}\big([a](\cdot), [b](\cdot)\big)$$



Appendices

"Static" constraints

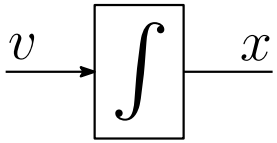
A definition of the $\mathcal{C}_+$ operator for tubes for the constraint $a(\cdot) = x(\cdot) + y(\cdot)$:

$$\mathcal{C}_+\big([a](\cdot),[x](\cdot),[y](\cdot)\big)$$

$$\begin{pmatrix} [a](t) \\ [x](t) \\ [y](t) \end{pmatrix} \xmapsto{\mathcal{C}_+} \begin{pmatrix} [a](t) \cap \big([x](\cdot) + [y](\cdot)\big) \\ [x](t) \cap \big([a](\cdot) - [y](\cdot)\big) \\ [y](t) \cap \big([a](\cdot) - [x](\cdot)\big) \end{pmatrix}$$

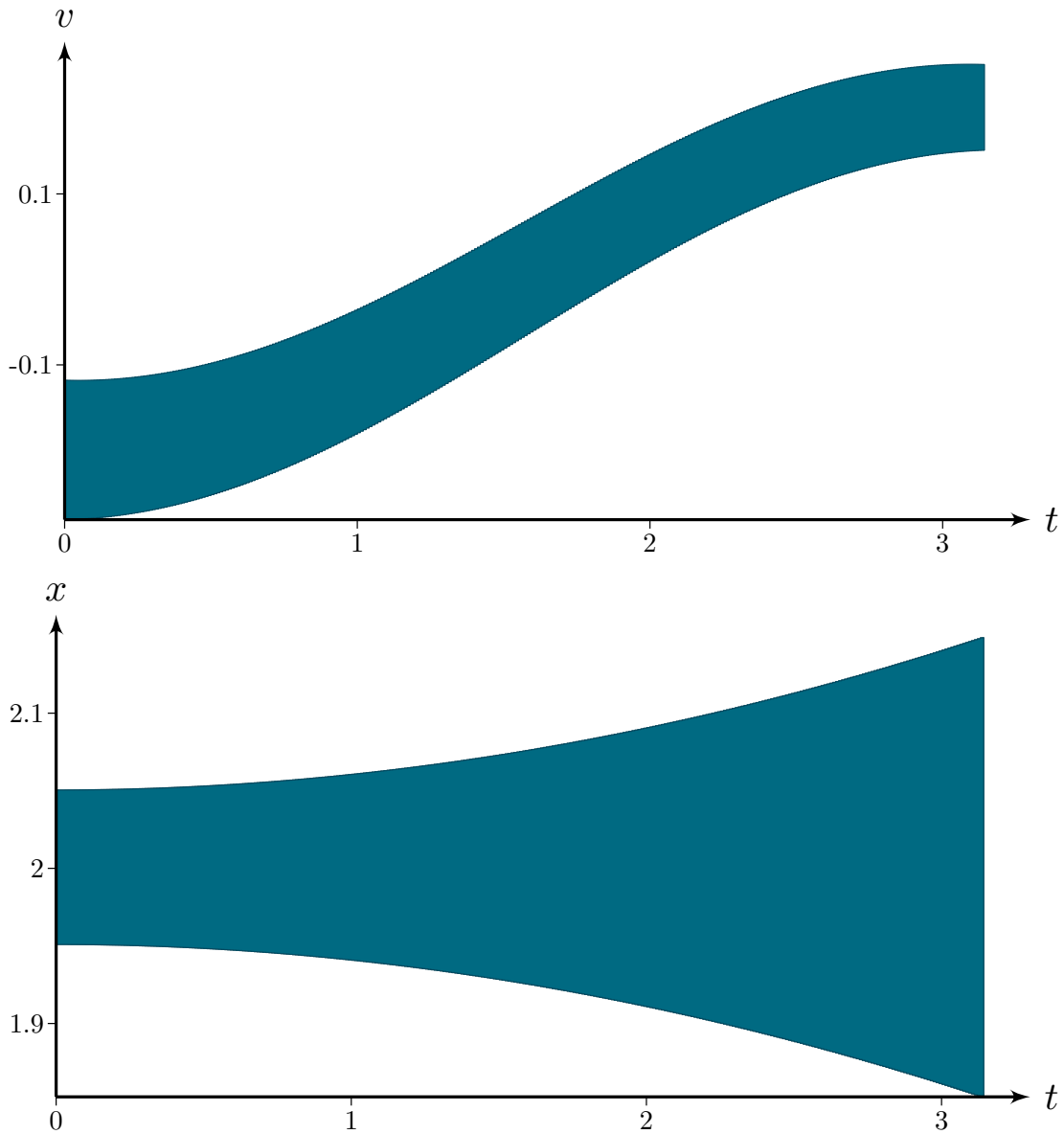
Appendices

Derivative constraint



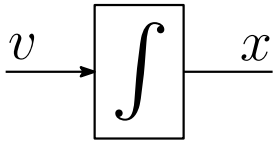
Differential constraint:

- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot)$
- one trajectory and its derivative



Appendices

Derivative constraint

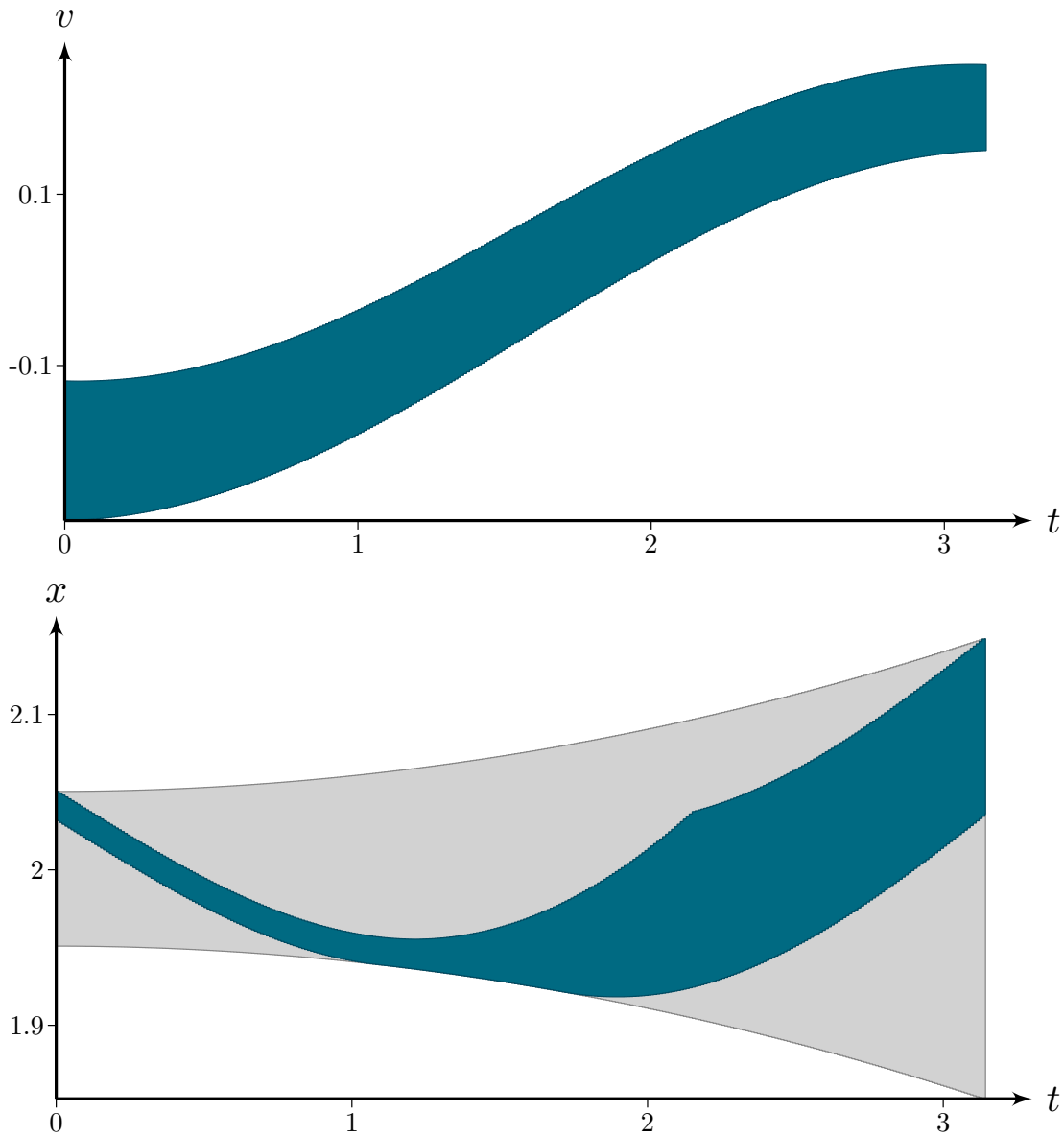


Differential constraint:

- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot)$
- one trajectory and its derivative

Contractor on tubes:

$$\mathcal{C}_{\frac{d}{dt}}\left([\mathbf{x}](\cdot), [\mathbf{v}](\cdot)\right)$$



Guaranteed computation of robot trajectories

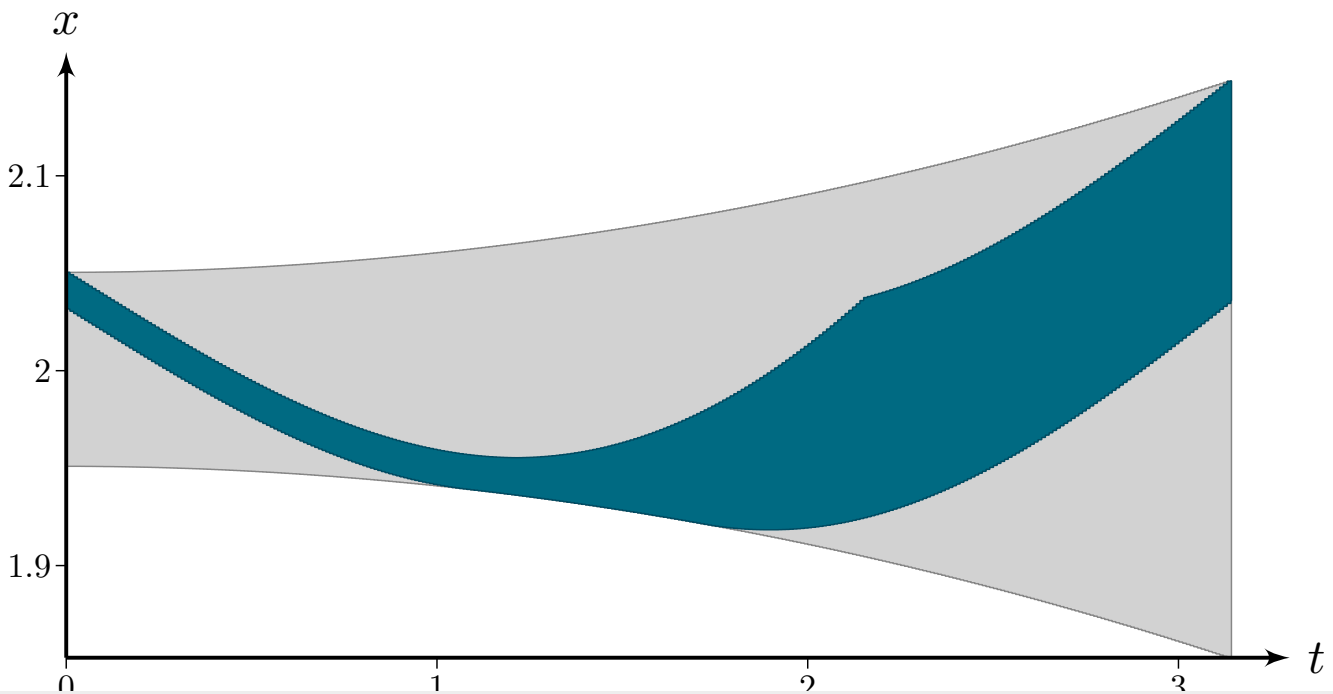
Rohou, Jaulin, Mihaylova, Le Bars, Veres

Appendices

Derivative constraint

Definition of the $\mathcal{C}_{\frac{d}{dt}}$ operator:

$$\begin{pmatrix} [x](t) \\ [v](t) \end{pmatrix} \xmapsto{\mathcal{C}_{\frac{d}{dt}}} \begin{pmatrix} \bigcap_{t_1=t_0}^{t_f} \left([x](t_1) + \int_{t_1}^{t} [v](\tau) d\tau \right) \\ [v](t) \end{pmatrix} \quad (1)$$



Appendices

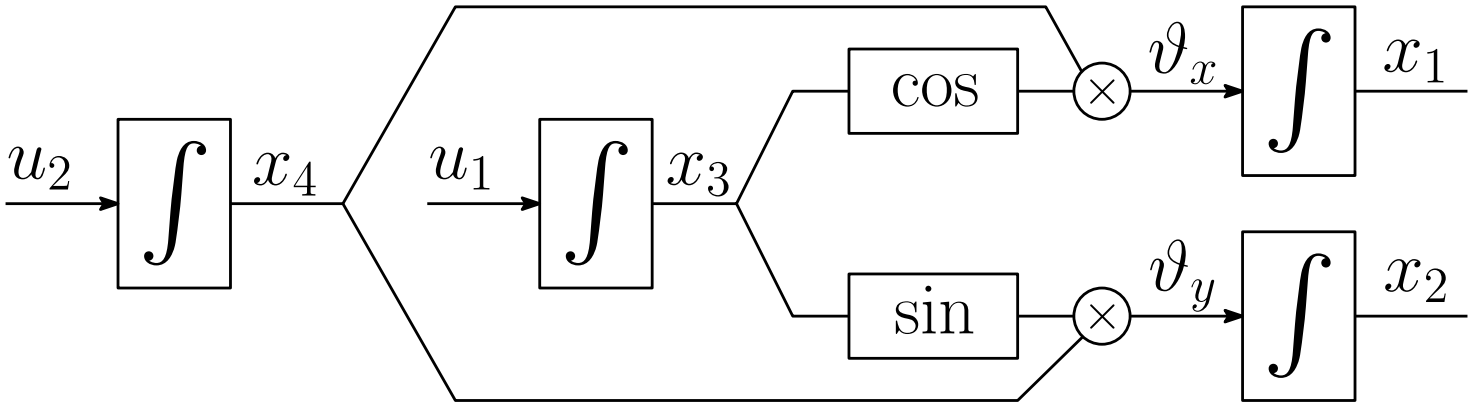
Decomposition of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$

Appendices

Decomposition of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$

State equation:

$$\begin{cases} (i) & \dot{x}_1 = x_4 \cos(x_3) \\ (ii) & \dot{x}_2 = x_4 \sin(x_3) \\ (iii) & \dot{x}_3 = u_1 \\ (iv) & \dot{x}_4 = u_2 \end{cases}$$

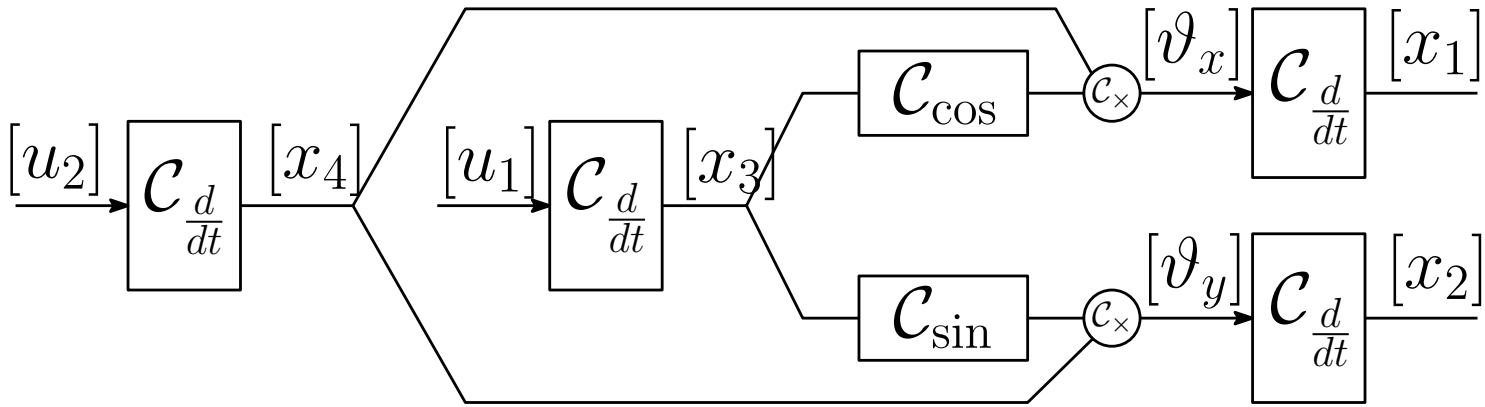


Appendices

Decomposition of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$

State equation:

$$\begin{cases} (i) & \dot{x}_1 = x_4 \cos(x_3) \\ (ii) & \dot{x}_2 = x_4 \sin(x_3) \\ (iii) & \dot{x}_3 = u_1 \\ (iv) & \dot{x}_4 = u_2 \end{cases}$$

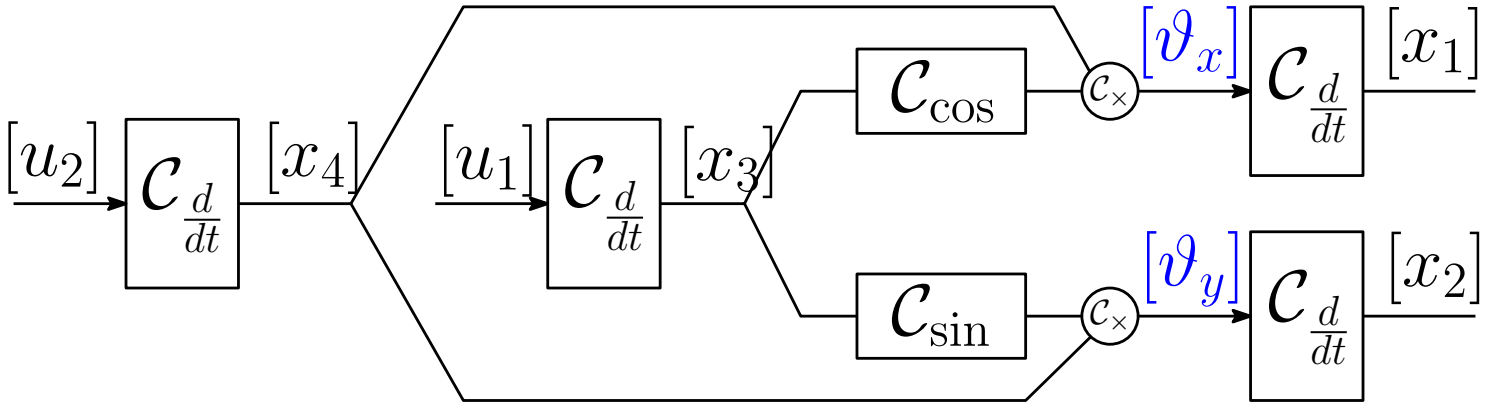


Appendices

Decomposition of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$

State equation:

$$\left\{\begin{array}{rlcll} (i) & \dot{x}_1 & = & x_4\cos(x_3) & \Leftrightarrow \ \{\dot{x}_1 = \vartheta_x \ ; \ \vartheta_x = x_4\cos(x_3)\} \\ (ii) & \dot{x}_2 & = & x_4\sin(x_3) & \Leftrightarrow \ \{\dot{x}_2 = \vartheta_y \ ; \ \vartheta_y = x_4\sin(x_3)\} \\ (iii) & \dot{x}_3 & = & u_1 & \\ (iv) & \dot{x}_4 & = & u_2 & \end{array}\right.$$

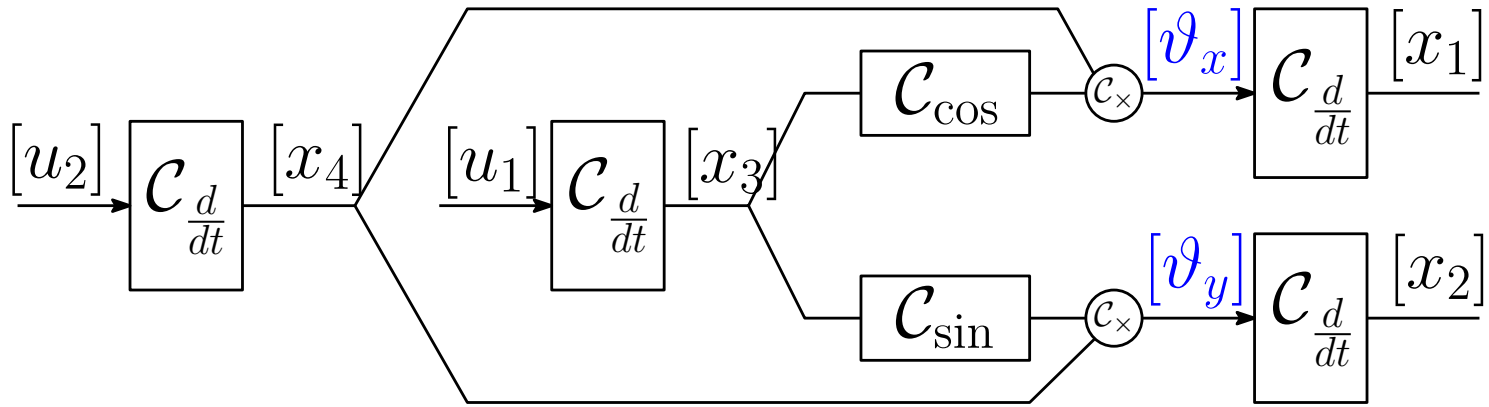


Appendices

Decomposition of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$

State equation:

$$\left\{\begin{array}{rlcll} (i) & \dot{x}_1 & = & x_4\cos(x_3) & \Leftrightarrow \ \{\dot{x}_1 = \vartheta_x \ ; \ \vartheta_x = x_4\cos(x_3)\} \\ (ii) & \dot{x}_2 & = & x_4\sin(x_3) & \Leftrightarrow \ \{\dot{x}_2 = \vartheta_y \ ; \ \vartheta_y = x_4\sin(x_3)\} \\ (iii) & \dot{x}_3 & = & u_1 & \\ (iv) & \dot{x}_4 & = & u_2 & \end{array}\right.$$



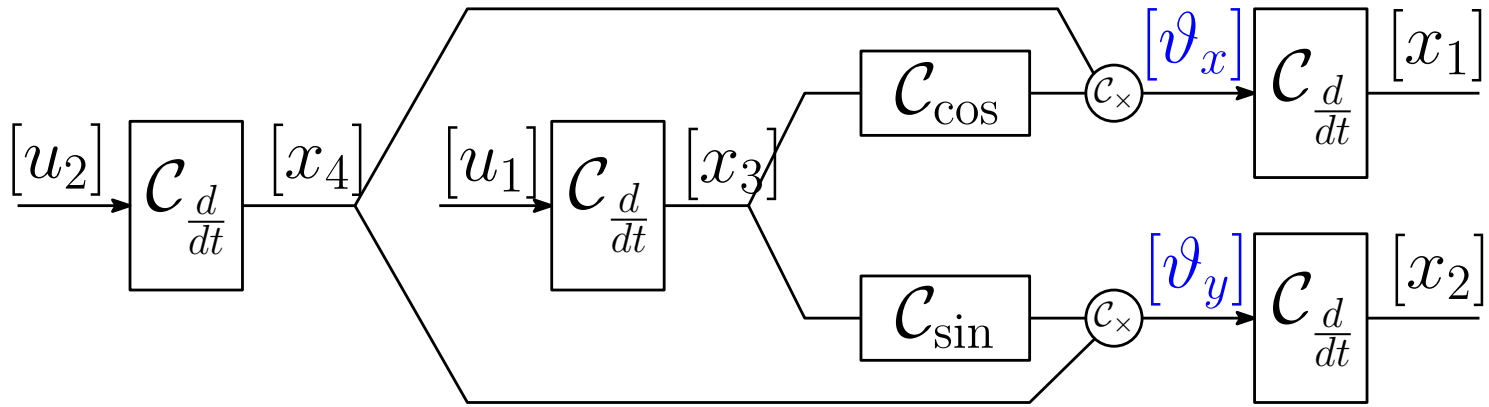
Involved operators: $\mathcal{C}_{\times}$, $\mathcal{C}_{\cos}$, $\mathcal{C}_{\sin}$, $\mathcal{C}_{\frac{d}{dt}}$

Appendices

Decomposition of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$

State equation:

$$\left\{\begin{array}{rlcll} (i) & \dot{x}_1 & = & x_4\cos(x_3) & \Leftrightarrow \ \{\dot{x}_1 = \vartheta_x \ ; \ \vartheta_x = x_4\cos(x_3)\} \\ (ii) & \dot{x}_2 & = & x_4\sin(x_3) & \Leftrightarrow \ \{\dot{x}_2 = \vartheta_y \ ; \ \vartheta_y = x_4\sin(x_3)\} \\ (iii) & \dot{x}_3 & = & u_1 & \\ (iv) & \dot{x}_4 & = & u_2 & \end{array}\right.$$



Involved operators: $\mathcal{C}_{\times}$, $\mathcal{C}_{\cos}$, $\mathcal{C}_{\sin}$, $\mathcal{C}_{\frac{d}{dt}}$

Involved sets: $[\mathbf{x}](\cdot)$, $[\mathbf{u}](\cdot)$, $[\vartheta_x](\cdot)$, $[\vartheta_y](\cdot)$

Appendices

Decomposition and wrapping effects

Appendices

Decomposition and wrapping effects

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}) \iff \left\{ \begin{array}{l} \mathbf{v} = \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \dot{\mathbf{x}} = \mathbf{v} \end{array} \right. \implies \left\{ \begin{array}{l} \mathcal{C}_{\mathbf{f}}\big([\mathbf{v}](\cdot),[\mathbf{x}](\cdot),[\mathbf{u}](\cdot)\big) \\ \mathcal{C}_{\frac{d}{dt}}\big([\mathbf{x}](\cdot),[\mathbf{v}](\cdot)\big) \end{array} \right.$$

Appendices

Decomposition and wrapping effects

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}) \iff \left\{ \begin{array}{l} \mathbf{v} = \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \dot{\mathbf{x}} = \mathbf{v} \end{array} \right. \implies \left\{ \begin{array}{l} \mathcal{C}_{\mathbf{f}}\big([\mathbf{v}](\cdot),[\mathbf{x}](\cdot),[\mathbf{u}](\cdot)\big) \\ \mathcal{C}_{\frac{d}{dt}}\big([\mathbf{x}](\cdot),[\mathbf{v}](\cdot)\big) \end{array} \right.$$

See also: more efficient contractors without decomposition, *e.g.* $\mathcal{C}_{\text{Lohner}}$ for dealing with $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$

- Safe and collaborative autonomous underwater docking

  Auguste Bourgois *PhD thesis*, 2021