Une très brève introduction aux intervalles pour la robotique mobile

Tutoriel aux JNRR – Octobre 2023

Ce document est téléchargeable sur :
`www.codac.io/tmp/jnrr-2023/exercice_slam.pdf`

A. Introduction

We propose to apply interval tools for solving a classical state estimation problem.

A tank robot $\mathcal{R}$, described by the state vector $\mathbf{x} \in \mathbb{R}^3$ depicting its position $(x_1, x_2)^\intercal$ and its heading x_3, is evolving among a set of landmarks $\mathbf{b}^k \in \mathbb{R}^2$. It is equipped with a compass for measuring its heading x_3. The speed is assumed to be constant.

The system is described by the following state equations:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), u(t)\big) & \text{(evolution equation)} \\ d_i^k = g\big(\mathbf{x}(t_i), \mathbf{b}^k\big) & \text{(observation equation)} \end{cases} \tag{1}$$

The evolution of the state is given by:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = \begin{pmatrix} 10\cos(x_3) \\ 10\sin(x_3) \\ u \end{pmatrix}, \tag{2}$$

and the system input u (rotation rate) is expressed as:

$$u(t) = 3\sin^2(t) + \frac{t}{100}. \tag{3}$$

The observation function g is the distance function between a position $(x_1, x_2)^\intercal$ and a landmark $(b_1^k, b_2^k)^\intercal$.

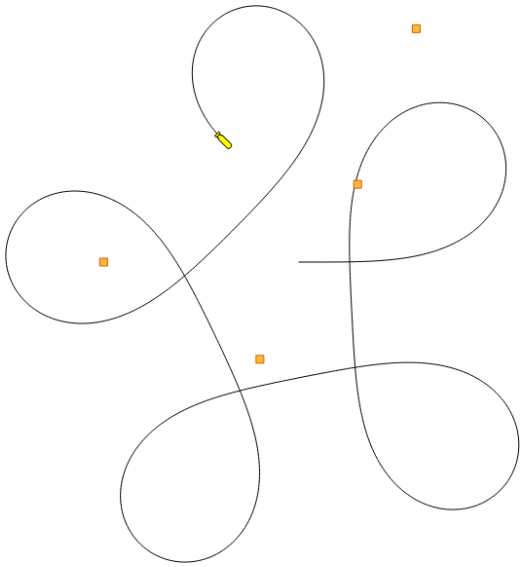


Figure 1: Trajectory of $\mathcal{R}$ between a set of landmarks.

B. Codac

1. Install the Codac library on your computer. Codac is available in C++ and Python3 (now also in Matlab), on Linux, Windows, MacOS systems. See: `http://codac.io/installation`
2. Download and launch the VIBes viewer.
   See: `http://codac.io/install/01-installation.html#graphical-tools`
3. Download the file of this lesson, available at `http://codac.io/tmp/jnrr-2023/exercice.py`. Compile and/or execute the example. You should obtain the Figure 1 in the VIBes interface, without the landmarks.

The following instructions will be given in Python, but feel free to use C++ or Matlab if you prefer.

C. Deadreckoning

The robot knows its initial state: $\mathbf{x}(0) = (0, 0, 0)^\intercal$. Deadreckoning consists in estimating the following positions of the robot without exteroceptive measurements (*i.e.* without distances from the landmarks). In this section, we will compute the set of feasible positions of $\mathcal{R}$, considering only heading measurements and the evolution function $\mathbf{f}$.

4. The set of feasible positions along time is a *tube* (interval of trajectories). We create a tube $[\mathbf{x}](t)$ using:

   ```
   x = TubeVector(tdomain, dt, 3)
   ```

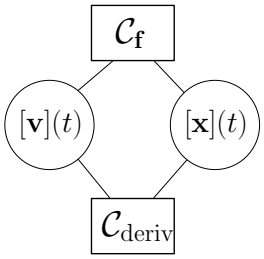
   where `tdomain` is the temporal window $[t_0, t_f]$ of the simulation, and `dt` is a discretization parameter. Last argument is the dimension of the tube.
   At this point, $\forall t \in [t_0, t_f]$, $[\mathbf{x}](t) = [-\infty, \infty]^3$: the states are completely unknown.
   This can be verified, for instance at $t = 0$, with: `print(x(0.))`

5. The headings $x_3(t)$ are measured with some uncertainties known to be bounded in $[-0.03, 0.03]$. We set these bounded measurements in the last component of the tube vector $[\mathbf{x}](t)$:

   ```
   # Heading measurement with bounded uncertainties
   x[2] = Tube(x_truth[2], dt) + Interval(-0.03,0.03)
   ```

6. The initial state $\mathbf{x}(0)$ (position and heading) is known, which can be implemented in the tube with:

   ```
   x.set([0,0,0], 0.) # setting a vector value at t=0
   ```



Now that a domain (a tube) has been defined for enclosing the estimates together with their uncertainties, it remains to define contractors for narrowing their bounds. In deadreckoning, only Eq. (2) is considered: $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$. This can be processed with two contractors, one for dealing with $\mathbf{v}(t) = \mathbf{f}(\mathbf{x}(t))$, and one for $\dot{\mathbf{x}}(t) = \mathbf{v}(t)$. The new tube $[\mathbf{v}](t)$ will enclose the feasible derivatives of the possible states in $[\mathbf{x}](t)$.

7. As for $[\mathbf{x}](t)$, create another `TubeVector` for $[\mathbf{v}](t)$, called `v`.

8. Create a contractor[1] for $\mathbf{v}(t) = \mathbf{f}(\mathbf{x}(t))$:

   ```
   ctc_f = CtcFunction(Function("x[3]", "v[3]", "(v[0]-10*cos(x[2]) ; v[1]-10*sin(x[2]))"))
   # This contractor expresses the constraint under the form f(x,v)=0
   ```

9. Create a contractor[2] for $\dot{\mathbf{x}}(t) = \mathbf{v}(t)$:

   ```
   ctc.deriv # object already instanciated in the library, nothing to do
   ```

Contractors are algorithms for reducing sets of feasible values (intervals, boxes, tubes..). A network of contractors can be created using a `ContractorNetwork` (CN), that will manage the contractors automatically.

10. Combine the contractors in a CN with:

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

11. At this point, you have implemented an algorithm for deadreckoning.
    The state estimation can now be triggered with:

    ```
    cn.contract(True)
    ```

[1] `CtcFunction`, see more: `http://codac.io/manual/04-static-contractors/01-ctc-function.html`
[2] `CtcDeriv`, see more: `http://codac.io/manual/05-dynamic-contractors/01-ctc-deriv.html`

12. The contracted tube $[\mathbf{x}](t)$ can be displayed with:

```
fig_map.add_tube(x, "[x](t)", 0, 1)
fig_map.show(1.)
```

You should obtain the following result:

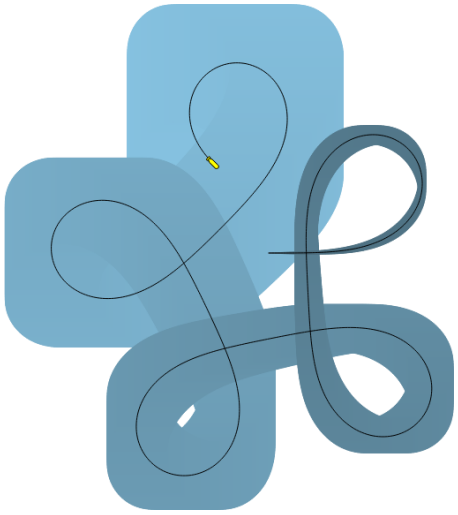


Figure 2: In blue: tube $[\mathbf{x}](t)$ enclosing the estimated trajectories of the robot. The actual but unknown trajectory is depicted in black. With interval methods, the computations are guaranteed: the actual trajectory cannot be outside the tube. However, the tube may be large in case of poor localization, as it is the case up to now without state observations.

D. Range-only localization

The obtained tube grows with time, illustrating a significant drift of the robot. We now rely on distances measured from known landmarks for reducing this drift. This amounts to a non-linear state estimation problem: function g of System (1) is a distance function. Non-linearities can be difficult to solve with conventional methods.

13. Define five landmarks with:

```
b = [(6,12),(-2,-5),(-3,20),(3,4),(-10,0)]
```

14. In a loop, for each $t_i \in \{0, 1, \ldots, 15\}$, compute the distance measured from a random landmark:

```
for ti in np.arange(0,15):
  k = random.randint(0,len(b)-1) # a random landmark is perceived
  d = sqrt(sqr(x_truth(ti)[0]-b[k][0])+sqr(x_truth(ti)[1]-b[k][1]))
```

15. The measurements come with some errors (not computed here) that are known to be bounded within $[-0.03, 0.03]$. We use intervals for enclosing the observations:

```
...
  d += Interval(-0.03,0.03)
```

16. In the same `for` loop, we add contractors in the ContractorNetwork in order to improve the localisation of the robot: the state observations are added to the set of constraints by means of distance contractors:

```
...
  pi = IntervalVector(3)                              # will represent the state at time ti
  cn.add(ctc.eval, [ti,pi,x,v])                       # constraint pi=x(ti), dot(x)=v
  cn.add(ctc.dist, [pi[0],pi[1],b[k][0],b[k][1],d])   # constraint d=g(pi,b)
```

The `cn` object that you have implemented is now the following:

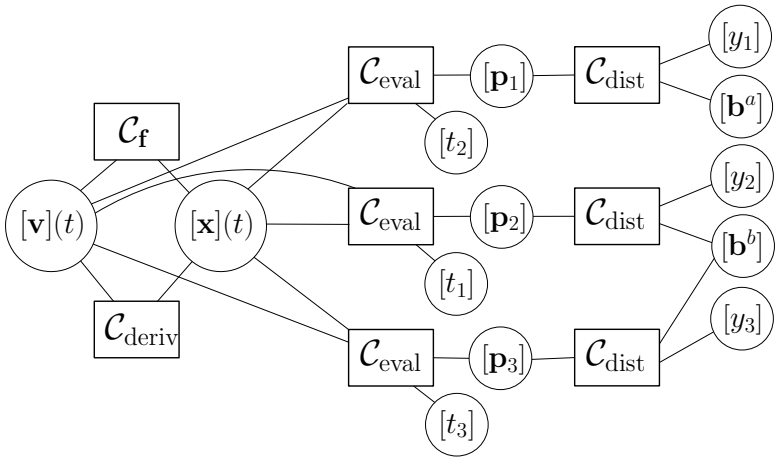


Figure 3: Example of CN related to System (1), involving three measurements and two landmarks. Boxes are contractor operators, circles are related domains: intervales, boxes and tubes.

17. Outside the iterations, trigger again the ContractorNetwork and display the results:

```
cn.contract(True)
fig_map.add_landmarks(b, 0.4)
fig_map.show(1.)
```

You should obtain the following result:

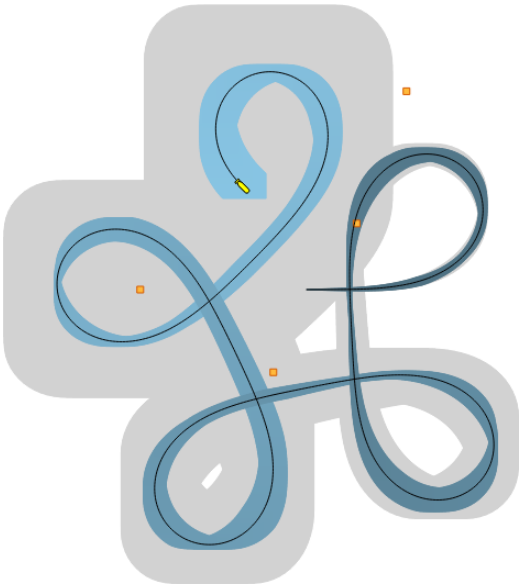


Figure 4: In gray: the former tube of Figure 2. In blue, the new contracted tube, considering distance measurements from the landmarks.

18. *What if we have no knowledge about the initial position of the robot?*
    Try to remove the condition set in Question 6, and zoom towards the initial position.

E. Challenge: Simultaneous Localization and Mapping (SLAM)

Update the script in order to deal with a SLAM problem, in which the position of the landmarks is estimated together with the positions of the robot. Note that in SLAM, the initial condition set in Question 6 is mandatory.