Tubex tutorial #2

Logic programming / UE 4.1 – 2020

www.simon-rohou.fr/research/icra2020-tutorial/ensta

This second lesson will extend the previous exercise by dealing with a data association problem together with localization: the landmarks perceived by the robot are now indistinguishable: they all look alike. The goal of this exercise is to develop our own contractor in order to solve the problem.

A. Indistinguishable landmarks

Our goal is to deal with the localization of a mobile robot for which a map of the environment is available. It is assumed that:

- the map is static and made of point landmarks;
- the landmarks are indistinguishable;
- the position of each landmark is known;
- the pose of the robot is not known.

This problem often occurs when using sonars for localization, where it is difficult to distinguish a landmark from another. The challenge in this type of scenario is the difficulty of *data associations*, *i.e.* to establish feature correspondences between perceptions of landmarks and the map of these landmarks, known beforehand. This is especially the case of challenging underwater missions, for which one can reasonably assume that the detected landmarks cannot be distinguished from the other: for instance, two different rocks may have the same aspect and dimension in sonar images, as illustrated by Figure 1.

Figure 1: Different underwater rocks perceived with one side-scan sonar. These observations are view-point dependent and the sonar images are noisy, which makes it difficult to distinguish one rock from another only from image processing. These sonar images have been collected by the *Daurade* robot (DGA-TN, Shom, Brest) equipped with a Klein 5000 lateral sonar.

Due to these difficulties, we consider that we cannot compute reliable data associations that stand on image processing of the observations. We will rather consider the data association problem together with state estimation.

B. Perception of identifiable landmarks: formalism

We will first focus on the perception of rocks clearly identified: each observation is related to a known position. The problem is similar to the localization of the previous lesson, but we also know the bearing, *i.e.* the angular position of the rock with respect to the heading of the robot.

The map is known beforehand and the robot obeys to state equations. For now, the localization problem corresponds to the following observation function:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) \tag{1}$$

where $\mathbf{x}$ is the unknown state vector: the robot is located at $(x_1, x_2)^\intercal$ with a heading x_3. The output measurement vector is $\mathbf{y}$ and is made of the range y_1 and the bearing y_2, as illustrated by Figure 2. It is assumed that the measurement is bounded: $\mathbf{y} \in [\mathbf{y}]$, contrary to $\mathbf{x}$ that is unknown: $\mathbf{x} \in [\mathbf{x}] = [-\infty, \infty]^2$.

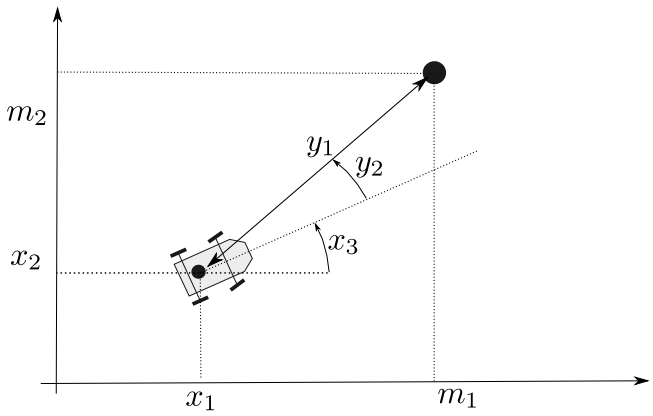

Figure 2: The landmark $\mathbf{m} = (m_1, m_2)^\intercal$ is perceived by the robot at a distance y_1 and a bearing y_2. The pose of the robot is $(x_1, x_2, x_3)^\intercal$.

Often, the observation equation also appears in the following implicit form:

$$\mathbf{g}(\mathbf{x}, \mathbf{y}) = \mathbf{0}. \tag{2}$$

In our case, the observation equation is defined as:

$$\mathbf{g}(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} x_1 + y_1 \cdot \cos(x_3 + y_2) - m_1 \\ x_2 + y_1 \cdot \sin(x_3 + y_2) - m_2 \end{pmatrix}. \tag{3}$$

Note that in a *range-only* case where the robot only measures the distance y to the landmark, we obtain the following relation:

$$g(\mathbf{x}, y) = (x_1 - m_1)^2 + (x_2 - m_2)^2 - y^2.$$

In the more general case, the observation function $\mathbf{g}$ is uncertain, or more precisely, it depends on some parameters that are not exactly known. The related observation equation can be written as:

$$\mathbf{g}(\mathbf{x}, \mathbf{y}, \mathbf{m}) = \mathbf{0} \tag{4}$$

In a set-membership approach the parameter vector $\mathbf{m}$, expressing the uncertain position of the landmark, is known to be bounded within an interval-vector (a box): $[\mathbf{m}]$. We emphasize that uncertainties on $\mathbf{x}$ and $\mathbf{y}$ are implicitly bounded by $[\mathbf{x}]$ and $[\mathbf{y}]$.

C. Constraint programming

The problem will be described as a set of constraints depicting Equations (4). A solver based on these relations will then allow the resolution of this simple state estimation.

In order to build the solver with a `ContractorNetwork`, we first need to break down Equation (4) into a set of elementary constraints. The range-and-bearing problem is usually related to the *polar constraint* that links Cartesian coordinates to polar ones. This constraint can appear in our decomposition, since a dedicated contractor already exists to deal with it:

$$\mathcal{L}_{\text{polar}}(d_1, d_2, \rho, \alpha): \quad \mathbf{d} = \rho \cdot \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} \tag{5}$$

1. On a sheet of paper, write a decomposition of Equation (4) involving $\mathcal{L}_{\text{polar}}$ and intermediate variables.
2. Define the initial domains of each variable.

D. Contractor network

The constraints are applied on sets by means of contractors. Using Tubex, the `ContractorNetwork` will help us to solve the problem efficiently.

3. Update your version of Tubex in order to obtain the new features:

   ```
   cd tubex-lib
   git pull origin master
   cd make ; cmake .. ; sudo make install
   ```

4. Download the sources of the exercise on the webpage of the lesson[1] and read carefully the provided code.

5. Define the three-dimensional box $[\mathbf{x}] \in \mathbb{IR}^3$ used as a domain of the state. We assume that the position of the robot is not known: $[x_1] = [x_2] = [-\infty, \infty]$. However, the heading x_3 is completely known (for instance thanks to a compass); the actual heading x_3 is represented by `truth[2]`.

6. Create the contractors related to the decomposition of Question 1. The contractor $\mathcal{C}_{\text{polar}}$ is given by the class `pyibex::CtcPolar` and is used to contract the four interval domains $[d_1]$, $[d_2]$, $[\rho]$, $[\alpha]$:

   ```
   pyibex::CtcPolar ctc_polar; // polar constraint (d1,d2,rho,alpha)
   ```

   The other contractor objects related to the elementary constraints you obtained in Question 1 can be built with the `ibex::CtcFwdBwd`, as we did in the previous Lesson #01 (Section F).

7. Create a `ContractorNetwork` (CN) to solve the problem by adding (with `cn.add(..)`) the contractors and the domains.

8. Use `cn.contract();` to solve the problem, and display the result of $[\mathbf{x}]$ with:

   ```
   fig_map.draw_box(x.subvector(0,1)); // estimated position (2d box)
   ```

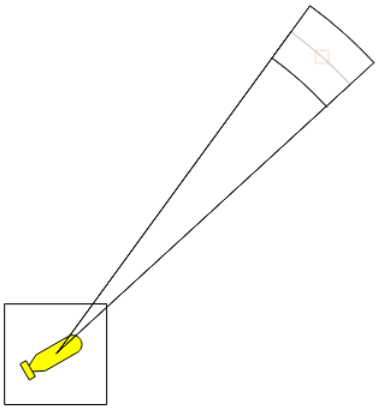

Figure 3: Range-and-bearing localization: expected result with one landmark.

[1] `http://simon-rohou.fr/research/icra2020-tutorial/doc/02/tuto_02_src.zip`

E. The problem of data association

Up to now, we assumed the perceived landmark was identified without ambiguity. However, when several landmarks $\mathbf{m}_1, \ldots, \mathbf{m}_\ell$ exist, the observation data may not be *associated*: we do not know to which landmark the measurement $\mathbf{y}$ refers. We now consider several measurements denoted by $\mathbf{y}^i$. Hence the observation constraint has the form:

$$\begin{cases} \mathbf{g}(\mathbf{x}, \mathbf{y}^i, \mathbf{m}^i) = \mathbf{0}, \\ (\mathbf{m}^i \in [\mathbf{m}_1]) \vee \cdots \vee (\mathbf{m}^i \in [\mathbf{m}_\ell]), \end{cases} \tag{6}$$

with $\mathbf{m}^i$ the unidentified landmark of the observation $\mathbf{y}^i$. The $\vee$ notation expresses the logical disjunction *or*. Equivalently, the system can be expressed as:

$$\begin{cases} \mathbf{g}(\mathbf{x}, \mathbf{y}^i, \mathbf{m}^i) = \mathbf{0}, & \text{(7a)} \\ \mathbf{m}^i \in \mathbb{M} = \{[\mathbf{m}_1], \ldots, [\mathbf{m}_\ell]\}, & \text{(7b)} \end{cases}$$

where $\mathbb{M}$ is the bounded map of the environment: the set of all landmarks represented by their bounded positions. In other words, we do not know the right parameter vector $\mathbf{m}_1, \ldots, \mathbf{m}_\ell$ associated with the observation $\mathbf{y}^i$. Equation (7b) represents this problem of data association. Figure 4 illustrates $\mathbb{M}$ with the difficulty to differentiate landmarks in underwater extents.

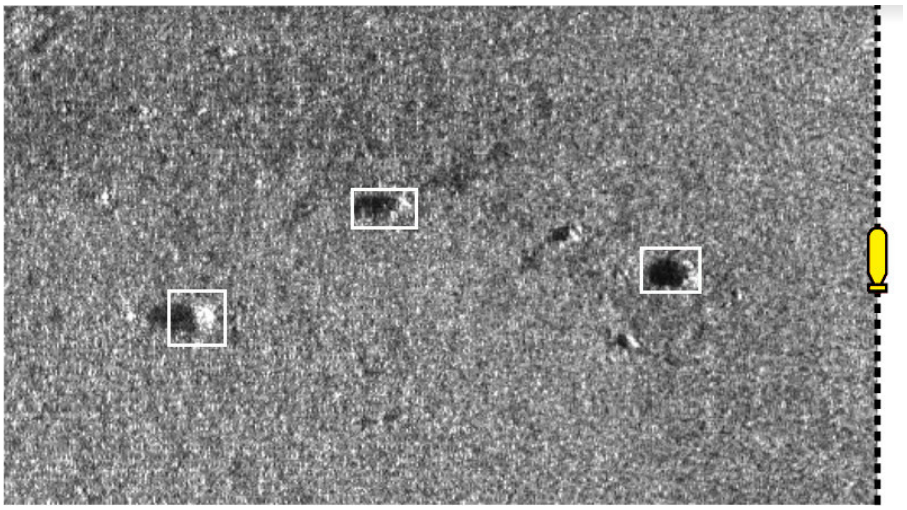

Figure 4: The yellow robot, equipped with a side-scan sonar, perceives at port side some rocks $\mathbf{m}^i$ lying on the seabed. The rocks, that can be used as landmarks, are assumed to belong to small georeferenced boxes $[\mathbf{m}^i]$ enclosing uncertainties on their positions. The robot is currently not able to make any difference between the rocks, as it is typically the case in underwater extents when acoustic sensors are used to detect features.

In this exercise, the data association problem is solved together with state estimation, without image processing. Equation (7a) has been solved in Section D., it remains to deal with the constraint related to Equation (7b).

F. Association constraint

Equation (7b) refers to what we will call the *association constraint*. This last part of the exercise consists in implementing a new contractor $\mathcal{C}_{\text{asso}}$, that will allow us to solve the data association problem.

Let us consider a constellation of ℓ landmarks $\mathbb{M} = \{[\mathbf{m}_1], \ldots, [\mathbf{m}_\ell]\}$ of $\mathbb{IR}^2$ and a box $[\mathbf{a}] \in \mathbb{IR}^2$. Our goal is to compute the smallest box containing $\mathbb{M} \cap [\mathbf{a}]$. In other words, we want to contract the box $[\mathbf{a}]$ so that we only keep the best envelope of the landmarks that are already included in $[\mathbf{a}]$. An illustration is provided by the Figures 5.

The definition of the contractor is given by:

$$\mathcal{C}_{\text{asso}}([\mathbf{a}]) = \bigsqcup_j \left([\mathbf{a}] \cap [\mathbf{m}_j] \right), \tag{8}$$

where $\sqcup$, called *squared union*, returns the smallest box enclosing the union of its arguments.

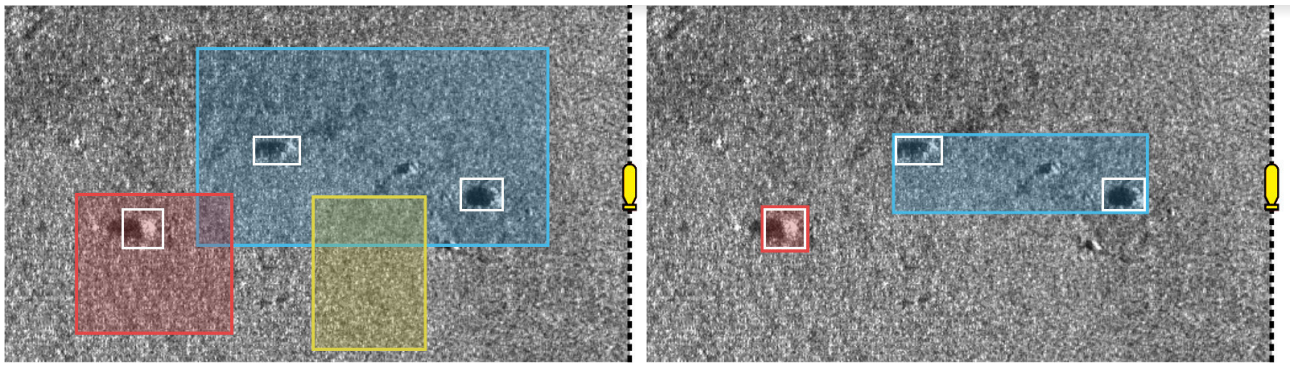

Figure 5: Illustration of the association contractor, before and after the contraction step. The set $\mathbb{M}$ of landmarks is depicted by white boxes. This operator is reliable as it does not remove any significant rock. In this example, the red perception leads to a reliable association since the box contains only one item of $\mathbb{M}$.

9. We will build our own C++ contractor. To do it, we have to derive the class `ibex::Ctc`. Insert in your code (possibly in separated files) the new class:

```
class CtcAssociation : public ibex::Ctc
{
  public:

    CtcAssociation(const std::vector<ibex::IntervalVector>& map)
      : ibex::Ctc(2), m_map(map)
    {

    }

    void contract(ibex::IntervalVector& a)
    {
      // todo...
    }

  protected:

    const std::vector<ibex::IntervalVector> m_map;
};
```

   With:

   - `const std::vector<ibex::IntervalVector>& map`, the set $\mathbb{M}$
   - `ibex::IntervalVector& a`, the box $[\mathbf{a}]$ to be contracted

10. Propose a simple implementation of the method `void contract(ibex::IntervalVector& a)`. Note that you cannot change the definition of this method. The IBEX documentation may help you to compute operations on sets such as unions or intersections: `http://www.ibex-lib.org/doc/interval.html`

11. Test your contractor with:

    - a set $\mathbb{M} = \{([2,3],[2,3])^\intercal\ ,\ ([5,6],[4,5])^\intercal\ ,\ ([9,10],[2,3])^\intercal\}$
    - two boxes to be contracted: $([0,4],[0,4])^\intercal$ and $([3.5,11],[1,6])^\intercal$

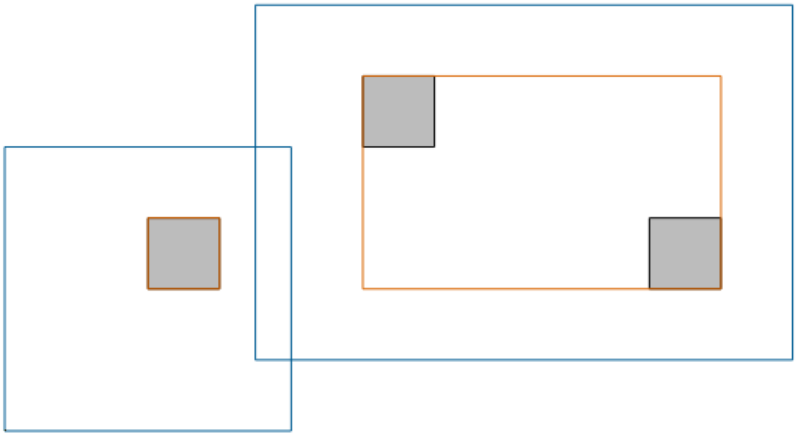

Figure 6: Expected result for Question 11. Blue boxes are initial domains before contraction. Red boxes are contracted domains. The left one is *identified* as it contains only one item of $\mathbb{M}$.

G. Localization with data association

We now have all the material to solve the full problem of state estimation with data association. The contractor $\mathcal{C}_{\text{asso}}$ will be added to the CN of Section D.

12. In your code, change the value of `nb_landmarks` to deal with 3 landmarks.

13. Now, the vectors `v_obs` and `v_b` contain three items. Update the CN in order to deal with several observations. For this, we will need to use *intermediate variables*.

    The case of intermediate variables. The following function:

    ```
    IntervalVector& d = cn.create_var(IntervalVector(2));
    ```

    creates an intermediate 2d variable with a 2d box domain: $[\mathbf{d}] = [-\infty, \infty]^2$. The method `create_var()` returns a C++ *reference* (in the above example: `IntervalVector&`). The reference is an alias on the intermediate variable, that is now inside the contractor network. It can be used in the same way as other variables.

    Why use intermediate variables instead of using directly domains, as we did in the first lesson?
    The reason is that if we create a CN and add contractors and domains in another block of code (for instance during an iteration), then the C++ variables are destroyed at the end of the block and so the CN lose the reference to them. For instance:

    ```
    ContractorNetwork cn;
    ibex::CtcFwdBwd ctc_plus(*new ibex::Function("a", "b", "c", "b+c-a"));
    Interval x(0,1), y(-2,3);

    if(/* some condition */)
    {
      Interval a(1,20);
      cn.add(ctc_plus, x, y, a); // constraint x+y=a
    } // 'a' is lost here

    cn.contract(); // segmentation fault
    ```

    Instead, we create the variable inside the CN in order to keep it outside the block:

    ```
    ContractorNetwork cn;
    ibex::CtcFwdBwd ctc_plus(*new ibex::Function("a", "b", "c", "b+c-a"));
    Interval x(0,1), y(-2,3);
    ```

```
if(/* some condition */)
{
  Interval& a = cn.create_var(Interval(1,20));
  cn.add(ctc_plus, x, y, a);
} // 'a' is not lost

cn.contract();
```

In our case, because there are several landmarks, we will have to use intermediate variables in the iteration of each observation i. You can change the `nb_landmarks` variable in the code in order to add more observations, if you want.

Note that one contractor can be added to the CN several times together with different domains.

14. Create an object of your own contractor $\mathcal{C}_{\text{asso}}$ with:

    ```
    CtcAssociation ctc_asso(v_b);
    ```

    and add it to the network for each observation, by using for instance:

    ```
    cn.add(ctc_asso, v_m[i]);
    ```

    where `v_m` would be a `vector` of 2d boxes representing all the $\mathbf{m}^i$ of Equation (7b).

15. Test the state estimation with one call to the `contract()` method.
    Run several simulations: are the landmarks all identified?

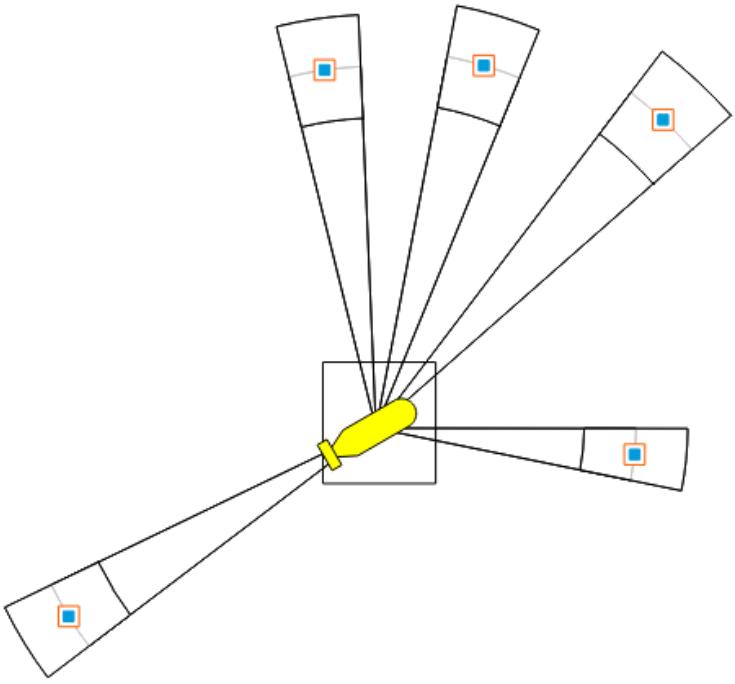

Figure 7: Localization with data association: example of expected result. The blue boxes depict the landamrks (red boxes) that have been identified. $[\mathbf{x}]$ is represented by the black box.

H. The robot moves (optional)

The movement of the robot is described by a differential equation: we now have to deal with a dynamical system. The state equations are:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) & \text{(evolution equation)} \\ \mathbf{g}\big(\mathbf{x}(t_i), \mathbf{y}^i, \mathbf{m}^i\big) = \mathbf{0} & \text{(observation equation)} \\ \mathbf{m}^i \in \mathbb{M} & \text{(data association)} \end{cases} \tag{9}$$

where $\mathbf{x}(t_i)$ corresponds to the time when the observation $\mathbf{y}^i$ has been made.

Tubex offers several tools to deal with dynamical systems such as this one. The same framework of constraint programming can be extended to deal with differential equations while keeping the advantages of simplicity and genericity of the approach. We can now expand the formalism of Contractor Networks by introducing a new kind of variable: *trajectories*, denoted by $x(t)$, or $\mathbf{x}(t)$ is the vector case. Trajectories are sets of values evolving with time. A new kind of variable comes with a new kind of domain, that will allow constraints computations around this variable. Hence, we also introduce *tubes*, that are intervals of trajectories. We denote them by $[x](t)$, or $[\mathbf{x}](t)$ for *tube vectors*. Tubex offers functionalities to handle tubes, such as basic operations on sets (intersection of tubes, additions, bisections, ...) and contractors to apply constraints on trajectories.

One can refer to the documentation of Tubex for more information about basic operations on tubes:
`http://simon-rohou.fr/research/tubex-lib/03_basics/04_tubes.html`

Equations (9) can be broken down into the following set of elementary constraints:

$$\begin{cases} (i) & \mathbf{v}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) \\ (ii) & \dot{\mathbf{x}}(t) = \mathbf{v}(t) \\ (iii) & \mathbf{p}^i = \mathbf{x}(t_i) \\ (iv) & ... \\ (v) & ... \\ (vi) & ... \\ (vii) & \mathcal{L}_{\text{asso}}(\mathbf{m}^i) \end{cases} \tag{10}$$

You have to complete the above decomposition with your own elementary constraints of Question (1). Note that these constraints involve new intermediate variables introduced for ease of decomposition:

- a vector $\mathbf{p}^i \in \mathbb{R}^3$, representing the state at time t_i (which is equivalent to the static state $\mathbf{x}$ of previous sections);
- a trajectory $\mathbf{v}(t) \in \mathbb{R} \to \mathbb{R}^3$ that depicts the derivative of the state. In practice, $\mathbf{v}(t)$ can be given by velocity sensors. In our case, we will assume that $\mathbf{v}(t)$ is at hand, enclosed in a tube $[\mathbf{v}](t)$, and we will not consider the Constraint (i).

There exists a catalog of contractors for differential constraints, available in Tubex. We will use them to solve our problem, without creating new operators:

- (iii): $\mathbf{p}^i = \mathbf{x}(t_i)$ is a constraint that links the vector $\mathbf{p}^i$ to the evaluation of the trajectory $\mathbf{x}(t)$ at time t_i. The constraint seems trivial, but it has to be considered carefully in the context of interval computations. A dedicated contractor $\mathcal{C}_{\text{eval}}$ already exists to deal with it. It will allow the correction of the positions of the robot.
  We will apply[2]: $\mathcal{C}_{\text{eval}}\big([t_i], [\mathbf{p}^i], [\mathbf{x}](t), [\mathbf{v}](t)\big)$.
  The $\mathcal{C}_{\text{eval}}$ contractor is provided in the class `CtcEval`.
- (ii): $\dot{\mathbf{x}}(t) = \mathbf{v}(t)$ is the simplest differential constraint that binds a trajectory $\mathbf{x}(t)$ to its derivative $\mathbf{v}(t)$. The related contractor $\mathcal{C}_{\frac{d}{dt}}$ exists and allows contractions on the tube $[\mathbf{x}](t)$ to preserve only trajectories consistent with the derivatives enclosed in the tube $[\mathbf{v}](t)$.
  We will apply $\mathcal{C}_{\frac{d}{dt}}\big([\mathbf{x}](t), [\mathbf{v}](t)\big)$.
  The $\mathcal{C}_{\frac{d}{dt}}$ contractor is provided in the class `CtcDeriv`.

[2] note that the contractor needs the derivative information: $[\mathbf{v}](t)$.

I. Expanding the CN with contractors on tubes (optional)

16. Download new sources on the webpage of the lesson:
`http://simon-rohou.fr/research/icra2020-tutorial/doc/02/tuto_02_src_tubes.zip`

    Read carefully the provided code. The tubes are already created for the problem. Note that the vectors of observations are now of dimension 3: they include time in their first component. The robot moves and describes a Lissajous curve ∞ from $t_0 = 0$ to $t_f = 6$.

17. Complete the code with the contractors you defined in the previous sections. Do not forget to add your own implementation of $\mathcal{C}_{\text{asso}}$.

18. The contractors $\mathcal{C}_{\text{eval}}$ are already added in the provided code. Add the contractor $\mathcal{C}_{\frac{d}{dt}}$ (from the class `CtcDeriv`) to the CN. You should obtain a result similar to Figure 8.

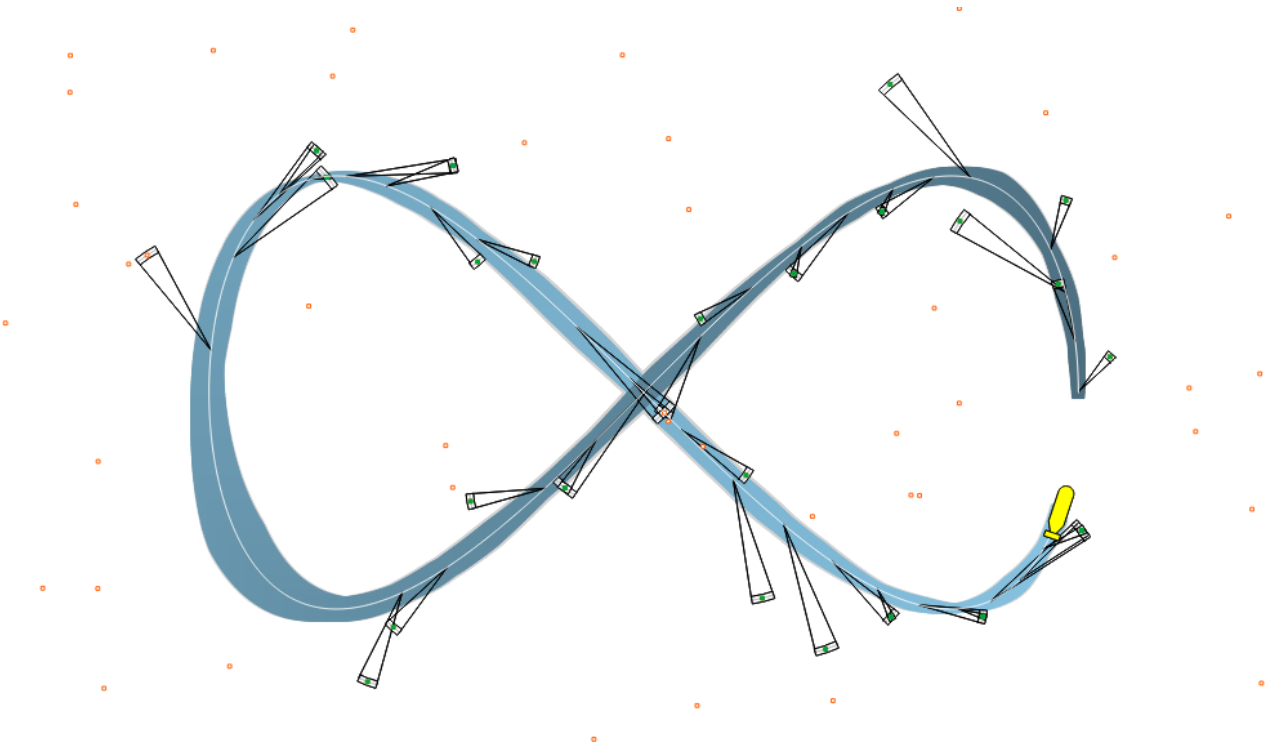

Figure 8: Dynamic localization with data association: example of expected result. The identified landmarks are green painted. The tube is depicted in blue and the actual (but unknown) trajectory is plotted in white.

Note that it is not necessary to know the initial position of the robot $\mathbf{x}(t_0)$ to be able to solve this problem. This is one of the advantages of constraint programming coupled with interval analysis, that take into account any information without a specific temporal order. Other conventional methods would hardly solve such problem.