Tubex tutorial #1

Logic programming / UE 4.1 – 2020

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

In this lesson, we will perform the state estimaton of a static robot between some landmarks. This will be an introduction to intervals, constraints and networks of contractors.

A. Installing Tubex

We first have to install Tubex and its dependencies. Tubex is used to deal with interval of trajectories and manage networks of contractors. Its dependency is the IBEX library, that has been developed to process constraints over real numbers.

1. Follow the instructions of `www.simon-rohou.fr/research/tubex-lib/02_installation` in order to install the two libraries.
2. Install and launch the visualizer VIBes if it is not already the case.
3. Go to `tubex-lib/examples/basics/` and compile/run one of the examples using CMake or directly with the `build.sh` script. You do not have to understand it, it is only to check that everything is properly installed.

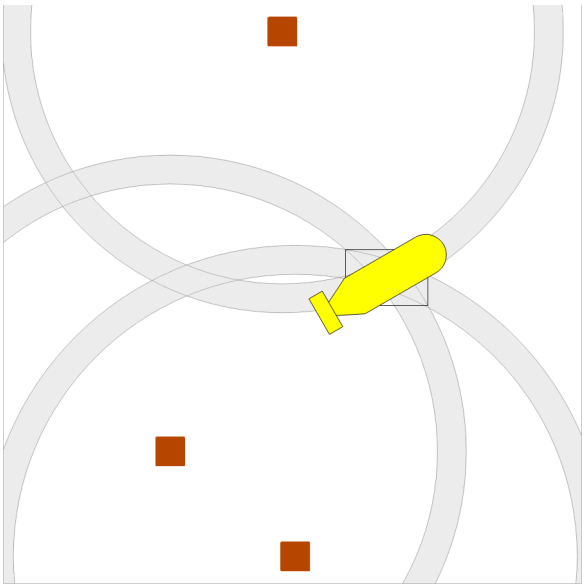

Figure 1: The final application of this lesson.

B. Start a new project

4. Start a new project as explained in `www.simon-rohou.fr/research/tubex-lib/03_basics/01_start_project.html` and verify that it is displaying information about a simple tube in the terminal.

C. Interval arithmetic, $[x]$

Tubex is using C++ objects[1] to represent intervals and boxes:

- `Interval x(double lb, double ub)` represents for instance the interval $[x]$ defined with its lower and upper bounds $[x^-, x^+]$. There exists predefined values such as `Interval x = Interval::EMPTY_SET` for $x = \varnothing$ and `Interval x = Interval::ALL_REALS` for $x = [-\infty, \infty]$, which is the default value.
- `IntervalVector x(int n)` is used for boxes. For instance, `IntervalVector x(2, Interval(-1,3))` creates the box $[\mathbf{x}] = [-1, 3]^2$. One can access vector components as we do for common vector sets: `x[1]=Interval(0,10);`

[1] The static objects, that are not time-related, are the ones defined in the IBEX library. `http://www.ibex-lib.org`

5. In your new project, compute the following simple operations on intervals:
   - $[-2,4] \cdot [1,3]$
   - $[-2,4] \sqcup [6,7]$
   - $\max([2,7],[1,9])$
   - $\max(\varnothing,[1,2])$
   - $[-1,3]/[0,\infty]$
   - $([1,2] \cdot [-1,3]) + \max([1,3] \cap [6,7],[1,2])$

   Note that $\sqcup$ is the hull union, *i.e.*, $[x] \sqcup [y] = [[x] \cup [y]]$.

6. These simple operations on sets can be extended to elementary functions such as cos, exp, tan. Create a 2d box $[\mathbf{y}] = [0,\pi] \times [-\pi/6, \pi/6]$ and display the result of $\cos([\mathbf{y}])$.

D. Functions, $f([x])$

Custom functions can be defined and used on sets. For instance, to compute:

$$f(x) = x^2 + 2x - \exp(x),$$

an object `Function` can be created and evaluated over the set $[x]$:

```
Interval x(-2,2);
ibex::Function f("x", "x^2+2*x-exp(x)");
Interval y = f.eval(x);
```

The first arguments of the function (only one in the above example) are its input variables. The last argument is the expression of the output[2]. The result is the set of images of all defined inputs through the function: $[f]([x]) = [\{f(x) \mid x \in [x]\}]$. Note that we use the `Function` class of IBEX, with namespace `ibex::Function`, since Tubex has also its own functions for dynamical systems.

7. For our robotic applications, we often need to define the distance function g:

   $$g(\mathbf{x},\mathbf{b}) = \sqrt{(x_1 - b_1)^2 + (x_2 - b_2)^2},$$

   where $\mathbf{x} \in \mathbb{R}^2$ would represent for instance the 2d position of a robot, and $\mathbf{b} \in \mathbb{R}^2$ the 2d location of some landmark. Create g and compute the distance between the boxes $[\mathbf{a}] = [0,0]^2$ and $[\mathbf{b}] = [3,4]^2$. Note that in the library, the `eval()` of functions only take one argument: we have to concatenate the boxes $[\mathbf{a}]$ and $[\mathbf{b}]$ in one 4d interval-vector $[\mathbf{c}]$ and then compute $g([\mathbf{c}])$.

E. Graphics

The graphical tool VIBes has been created to Visualize Intervals and Boxes. It is compatible with simple objects such as `Interval` and `IntervalVector`. Its features have been extended in the Tubex library with objects such as `VIBesFigMap`.

8. Create a view with:

```
vibes::beginDrawing();
VIBesFigMap fig("Map");
fig.set_properties(50, 50, 400, 400); // position and size
```

[2]For full documentation about static functions, please refer to `http://www.ibex-lib.org/doc/function.html`

```
// ... draw objects here

fig.show(); // display all items of the figure
vibes::endDrawing();
```

9. Before the `show()` method, draw the boxes $[\mathbf{a}]$ and $[\mathbf{b}]$ with the `fig.draw_box(IntervalVector x)` method. Draw the computed interval range with the `fig.draw_circle(double x, double y, double rad)` method. Is the result reliable, according to the sets $[\mathbf{a}]$ and $[\mathbf{b}]$?

F. Contractors, $\mathcal{C}([x])$

In the constraint programming approach, the method consists in defining contractors on sets in order to reduce them without losing any feasible solution. In Tubex, the contractors can also be defined with C++ objects. For this lesson, we will use the `ibex::CtcFwdBwd` class to defined a contractor according to a function f. Note that the contractors aim at solving constraints in the form $f(\mathbf{x}) = 0$. This contractor can be instantiate with a reference to the `ibex::Function` defining the constraint. For instance, the simple constraint $(x + y = a)$ is expressed as $f(x, y, a) = x + y - a$, and can be implemented as a contractor $\mathcal{C}_+$ with:

```
ibex::CtcFwdBwd ctc_add(*new ibex::Function("x", "y", "a", "x+y-a"));
```

10. Define a contractor $\mathcal{C}_{\text{dist}}$ related to the distance constraint between two 2d positions $\mathbf{x}$ and $\mathbf{b}$. We will use the distance function previously defined, but in the form $f(\mathbf{x}, \mathbf{b}, d) = 0$.

The contractor is then simply used in a *contractor network* that applies constraints on different variables for solving a problem. For instance, we can use the previously defined $\mathcal{C}_+$ as:

```
Interval x(0,1), y(-2,3), a(1,20);

ContractorNetwork cn;
cn.add(ctc_add, x, y, a); // adding the C+ contractor to the network
cn.contract();

// x=[0, 1], y=[-0, 3], a=[1, 4]
```

11. Define a contractor network with the $\mathcal{C}_{\text{dist}}$ object and apply it on some boxes $[\mathbf{b}^i]$.
    Check the results with $\mathcal{C}_{\text{dist}}([x_1], [x_2], [b_1^i], [b_2^i], [r])$ and
    - $[\mathbf{x}] = [0, 0]^2$, $[\mathbf{b}^1] = [-\infty, \infty]^2$, $[r] = [7, 8]$,
    - $[\mathbf{x}] = [0, 0]^2$, $[\mathbf{b}^2] = [6, 9]^2$, $[r] = [7, 8]$,
    - $[\mathbf{x}] = [0, 0]^2$, $[\mathbf{b}^3] = [7.2, 7.8]^2$, $[r] = [7, 8]$,
    - $[\mathbf{x}] = [0, 0]^2$, $[\mathbf{b}^4] = [7.5, 12]^2$, $[r] = [7, 8]$,
    - $[\mathbf{x}] = [0, 0]^2$, $[\mathbf{b}^5] = [-1, 1] \times [7, 10]$, $[r] = [7, 8]$,

    Draw the $[\mathbf{b}^i]$ boxes before and after the contractions, in order to assess the contracting effects.

G. Static range-only localization

We now have all the material to compute a solver for state estimation. We deal with the problem of localizing a robot of state $\mathbf{x} \in \mathbb{R}^2$ that measures distances $\rho^{(k)}$ from three landmarks $\mathbf{b}^{(k)}$, $k \in \{1,2,3\}$. We consider uncertainties on both the measurements and the location of the landmarks: $\rho^{(k)} \in [\rho^{(k)}]$ and $\mathbf{b}^{(k)} \in [\mathbf{b}^{(k)}]$. This corresponds to the constraint network presented in the slides of this lesson:

$$\begin{cases} \textbf{Variables: } \mathbf{x},\ \mathbf{b}^{(1)},\ \mathbf{b}^{(2)},\ \mathbf{b}^{(3)},\ \rho^{(1)},\ \rho^{(2)},\ \rho^{(3)} \\ \textbf{Constraints:} \\ \quad 1.\ \mathcal{L}_g^{(1)}\left(\mathbf{x}, \mathbf{b}^{(1)}, \rho^{(1)}\right) \\ \quad 2.\ \mathcal{L}_g^{(2)}\left(\mathbf{x}, \mathbf{b}^{(2)}, \rho^{(2)}\right) \\ \quad 3.\ \mathcal{L}_g^{(3)}\left(\mathbf{x}, \mathbf{b}^{(3)}, \rho^{(3)}\right) \\ \textbf{Domains: } [\mathbf{x}],\ [\mathbf{b}^{(1)}],\ [\mathbf{b}^{(2)}],\ [\mathbf{b}^{(3)}],\ [\rho^{(1)}],\ [\rho^{(2)}],\ [\rho^{(3)}] \end{cases}$$

Where $\mathcal{L}_g^{(k)}$ is a distance constraint from a landmark $\mathbf{b}^{(k)}$, that links a measurement $\rho^{(k)}$ to the state $\mathbf{x}$:

$$\mathcal{L}_g^{(k)}:\ \rho^{(k)} = \sqrt{\left(x_1 - \mathbf{b}_1^{(k)}\right)^2 + \left(x_2 - \mathbf{b}_2^{(k)}\right)^2}.$$

The following code provides a simulation of random landmarks and related range-only measurements:

```
// Truth (unknown pose)
Vector truth(3,0.);
truth[2] = M_PI/6.; // heading

// Creating random map of landmarks
const int nb_landmarks = 3;
const IntervalVector map_area(2, Interval(-8.,8.));
vector<IntervalVector> v_b = DataLoader::generate_landmarks_boxes(map_area, nb_landmarks);

// Generating range-only observations of these landmarks
vector<Interval> v_range(v_b.size());
for(int i = 0 ; i < v_b.size() ; i++)
{
  v_range[i] = sqrt(pow(truth[0]-v_b[i][0],2) + pow(truth[0]-v_b[i][1],2));
  v_range[i].inflate(0.1); // adding uncertainties
}
```

Finally, the graphical functions are given:

```
vibes::beginDrawing();
VIBesFigMap fig_map("Map");
fig_map.set_properties(50, 50, 600, 600);
for(const auto& iv : v_b)
  fig_map.add_beacon(Beacon(iv), 0.2);
for(int i = 0 ; i < v_range.size() ; i++)
{
  fig_map.draw_circle(v_b[i][0].mid(), v_b[i][1].mid(), v_range[i].lb(), "gray");
  fig_map.draw_circle(v_b[i][0].mid(), v_b[i][1].mid(), v_range[i].ub(), "gray");
}
fig_map.draw_box(x.subvector(0,1)); // estimated position
fig_map.draw_vehicle(truth, 1.);
fig_map.show();
vibes::endDrawing();
```

12. Before the graphical part, compute the state estimation of the robot by contracting the box $[\mathbf{x}] = [-\infty, \infty]^2$ with a contractor network:

   - $[\mathbf{x}]$ represents the unknown position of the robot
   - `vector<Interval> v_range` is the set of measurements $\{[\rho^{(1)}], [\rho^{(2)}], [\rho^{(3)}]\}$
   - `vector<IntervalVector> v_b` is the set of landmarks $\{[\mathbf{b}^{(1)}], [\mathbf{b}^{(2)}], [\mathbf{b}^{(3)}]\}$

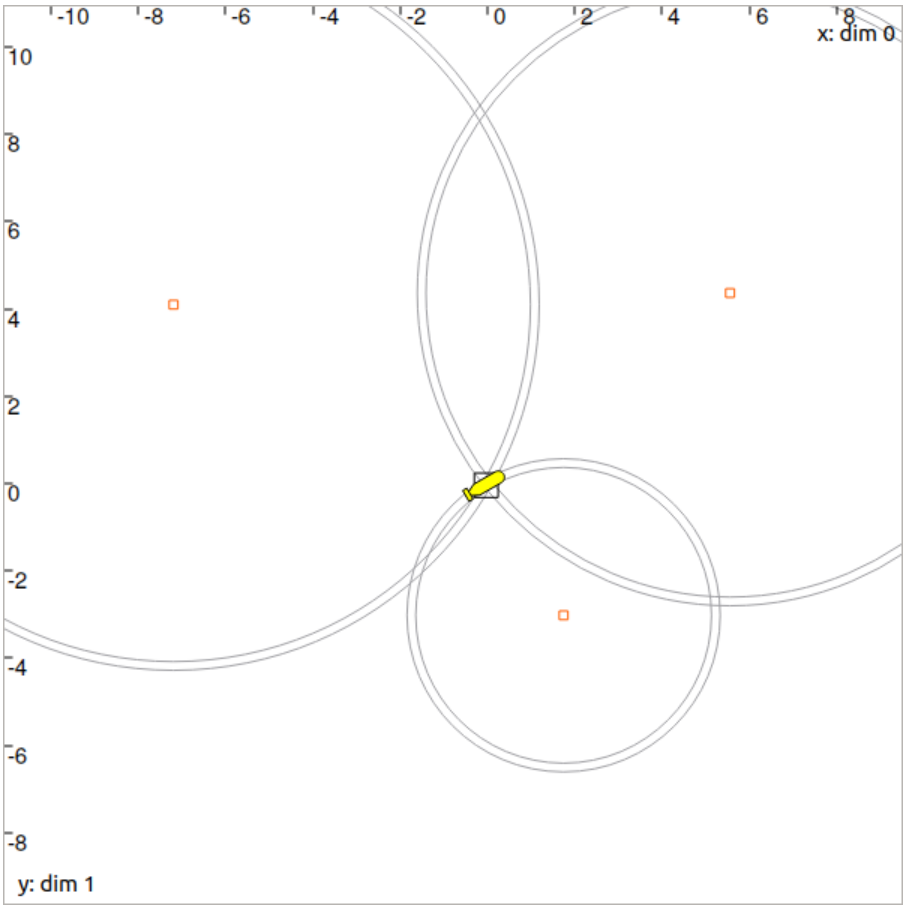

Figure 2: Range-only localization: expected result.