A CSP Approach to Design CPS

Alexandre Chapoutot

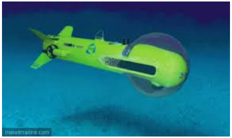
joint work with Julien Alexandre dit Sandretto and Olivier Mullier, Adina M. Panchea, *et al.*
U2IS, ENSTA ParisTech, Palaiseau, France

SHARC
June 30, 2017

Context

Robot's behavior

A mobile robot



Thanks to Google...

- moves $\Rightarrow$ Continuous-time dynamical system
- depending on parameters $\Rightarrow$ (bounded) uncertainties
- using sensors $\Rightarrow$ (bounded) uncertainties
- and actuators $\Rightarrow$ control input

We want to do something with...

Global property: safety

Avoid obstacles, respect actuator limits, etc.

A goal

Reach an objective, perform a mission, etc.

Some requirements

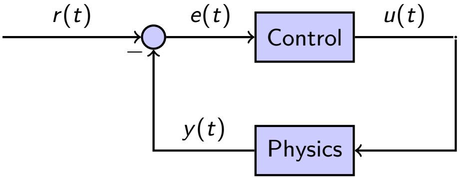
For one scenario, one behavior (with numerical criteria)

$\Rightarrow$ Some constraints on the robot's behaviors

Classical problems in robotics

Controller synthesis, Design, Path planning, Fault detection, Safety analysis, etc.

A small cyber-physical system: closed-loop control



- **Physics** is usually defined by non-linear differential equations (with parameters)

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), u(t), \mathbf{p}) \quad , \qquad\qquad \mathbf{y}(t) = g(\mathbf{x}(t))$$

- **Control** may be a continuous-time PI algorithm

$$e(t) = r(t) - y(t) \quad , \qquad\qquad u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

What is designing a controller?

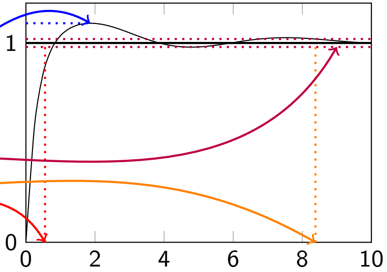
Find values for K_p and K_i such that a **given specification** is satisfied.

Specification of PID Controllers

PID controller: requirements based on closed-loop response

We observe the output of the plant

- Overshoot: Less than 10%
- Steady-state error: Less than 2%
- Settling time: Less than $10s$
- Rise time: Less than $2s$



Note: such properties come from the **asymptotic behavior** of the closed-loop system.

Classical method to study/verify closed-loop systems

Numerical simulations but

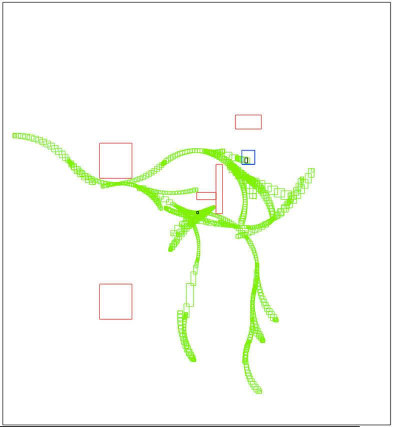
- do not take into account that models are only an approximation;
- produce approximate results.

and not adapted to deal with uncertainties

Synthesis and Verification methods for/of cyber-physical systems

Some requirements

- Shall deal with **discrete-time**, **continuous-time** parts and **their interactions**
- Shall **take into account uncertainties**: model, data, resolution methods
- Shall consider **temporal properties**



Example of properties (coming from box-RRT[1])

- system stays in safe zone ($\forall t$) or finishes in goal zone ($\exists t$)
- system avoids obstacle ($\exists t$)

for **different quantification's** of initial state-space ($\forall \mathbf{x}$ or $\exists \mathbf{x}$), parameters, etc.

[1] Pepy *et al.* Reliable robust path planning, Journal of AMCS, 2009

Our approach

Two antinomic facts

We want reliable results under uncertainties !

A known solution

Interval analysis works well for bounded uncertainties.

With dynamical systems ?

Validated simulation can help us.

Constraints on dynamical systems ?

A kind of temporal logic.

Set-based simulation

Definition

numerical simulation methods implemented with interval analysis methods

Goals

takes into account various uncertainties (bounded) or approximations to produce rigorous results

Example

A simple nonlinear dynamics of a car

$$\dot{v} = \frac{-50.0v - 0.4v^2}{m} \qquad \text{with} \quad m \in [990, 1010] \quad \text{and} \quad v(0) \in [10, 11]$$

One Implementation **DynIBEX**: a combination of **CSP solver** (IBEX[1]) with **validated numerical integration methods** based on **Runge-Kutta**

`http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

[1] Gilles Chabert (EMN) et al. `http://www.ibex-lib.org`

Validated numerical integration

Initial Value Problem of Ordinary Differential Equations

Consider an IVP for ODE, over the time interval $[0, T]$

$$\dot{\mathbf{y}} = f(\mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0$$

IVP has a unique solution $\mathbf{y}(t; \mathbf{y}_0)$ if $f : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz in $\mathbf{y}$
but for our purpose we suppose f smooth enough, *i.e.*, of class C^k

Goal of numerical integration

- Compute a sequence of time instants: $t_0 = 0 < t_1 < \cdots < t_n = T$
- Compute a sequence of values: $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_n$ such that

$$\forall i \in [0, n], \quad \mathbf{y}_i \approx \mathbf{y}(t_i; \mathbf{y}_0) \ .$$

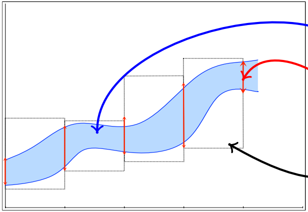
Validated solution of IVP for ODE

Goal of validated numerical integration

- Compute a sequence of time instants: $t_0 = 0 < t_1 < \cdots < t_n = T$
- Compute a sequence of values: $[\mathbf{y}_0], [\mathbf{y}_1], \ldots, [\mathbf{y}_n]$ such that

$$\forall i \in [0, n], \quad [\mathbf{y}_i] \ni \mathbf{y}(t_i; \mathbf{y}_0) \ .$$

A two-step approach



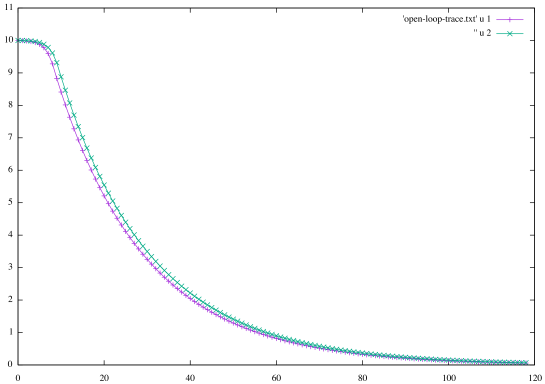
- Exact solution of $\dot{\mathbf{y}} = f(\mathbf{y}(t))$ with $\mathbf{y}(0) \in \mathcal{Y}_0$
- Safe approximation at discrete time instants
- Safe approximation between time instants

Simulation of an open loop system

A simple dynamics of a car

$$\dot{y} = \frac{-50.0y - 0.4y^2}{m} \qquad \text{with} \quad m \in [990, 1010]$$

Simulation for 100 seconds with $y(0) = 10$



The last step is $y(100) = [0.0591842, 0.0656237]$

Simulation of an open loop system

- ODE definition
- IVP definition
- Parametric simulation engine

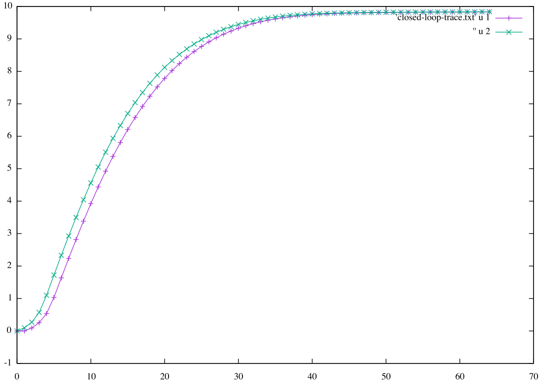
```
int main(){

  const int n = 1;
  Variable y(n);

  IntervalVector state(n);
  state[0] = 10.0;

  // Dynamique d'une voiture avec incertitude sur sa
masse
  Function ydot(y, ( -50.0 * y[0] - 0.4 * y[0] * y[0])
                / Interval (990, 1010));
  ivp_ode vdp = ivp_ode(ydot, 0.0, state);

  // Integration numerique ensembliste
  simulation simu = simulation(&vdp, 100, RK4, 1e-5);
  simu.run_simulation();

  //For an export in order to plot
  simu.export1d_yn("export-open-loop.txt", 0);

  return 0;
}
```

Simulation of a closed-loop system

A simple dynamics of a car with a PI controller

$$\begin{pmatrix} \dot{y} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} \frac{k_p(10.0-y)+k_i w-50.0y-0.4y^2}{m} \\ 10.0-y \end{pmatrix} \quad \text{with} \quad m \in [990, 1010], k_p = 1440, k_i = 35$$

Simulation for 10 seconds with $y(0) = w(0) = 0$



The last step is $y(10) = [9.83413, 9.83715]$

Simulation of a closed-loop system

```
#include "ibex.h"

using namespace ibex;

int main(){

  const int n = 2;
  Variable y(n);

  IntervalVector state(n);
  state[0] = 0.0;
  state[1] = 0.0;

  // Dynamique d'une voiture avec incertitude sur sa masse + PI
  Function ydot(y, Return ((1440.0 * (10.0 - y[0]) + 35.0 * y[1]  - y[0] * (50.0  + 0.4 * y[0]))
                        / Interval (990, 1010),
                        10.0 - y[0]));
  ivp_ode vdp = ivp_ode(ydot, 0.0, state);

  // Integration numerique ensembliste
  simulation simu = simulation(&vdp, 10.0, RK4, 1e-7);
  simu.run_simulation();

  simu.export1d_yn("export-closed-loop.txt", 0);

  return 0;
}
```

Simulation of an hybrid closed-loop system

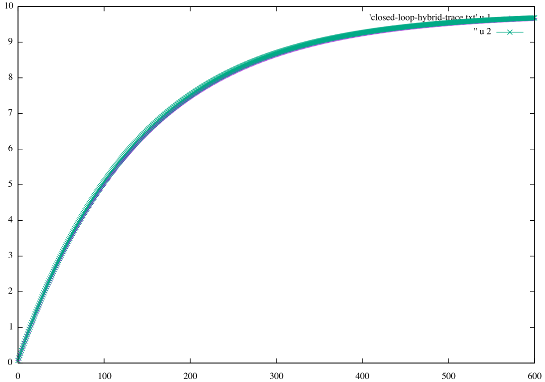
A simple dynamics of a car with a discrete PI controller

$$\dot{y} = \frac{u(k) - 50.0y - 0.4y^2}{m} \qquad \text{with} \quad m \in [990, 1010]$$

$$i(t_k) = i(t_{k-1}) + h(c - y(t_k)) \qquad \text{with} \quad h = 0.005$$

$$u(t_k) = k_p(c - y(t_k)) + k_i i(t_k) \qquad \text{with} \quad k_p = 1400, k_i = 35$$

Simulation for 3 seconds with $y(0) = 0$ and $c = 10$



Simulation of an hybrid closed-loop system

```
#include "ibex.h"

using namespace ibex;
using namespace std;

int main(){
  const int n = 2; Variable y(n);
  Affine2Vector state(n);
  state[0] = 0.0; state[1] = 0.0;

  double t = 0; const double sampling = 0.005;
  Affine2 integral(0.0);

  while (t < 3.0) {
    Affine2 goal(10.0);
    Affine2 error = goal - state[0];

    // Controleur PI discret
    integral = integral + sampling * error;
    Affine2 u = 1400.0 * error + 35.0 * integral;
    state[1] = u;

    // Dynamique d'une voiture avec incertitude sur sa masse
    Function ydot(y, Return((y[1] - 50.0 * y[0] - 0.4 * y[0] * y[0])
                        / Interval (990, 1010), Interval(0.0)));
    ivp_ode vdp = ivp_ode(ydot, 0.0, state);

    // Integration numerique ensembliste
    simulation simu = simulation(&vdp, sampling, RK4, 1e-6);
    simu.run_simulation();

    // Mise a jour du temps et des etats
    state = simu.get_last(); t += sampling;
  }
```

- Manual handling of discrete-time evolution

Differential constraint satisfaction problems

Basics of interval analysis

- **Interval arithmetic** (defined also for: sin, cos, etc.):

$$
\begin{aligned}
[\underline{x},\overline{x}] + [\underline{y},\overline{y}] &= [\underline{x}+\underline{y},\overline{x}+\overline{y}] \\
[\underline{x},\overline{x}] * [\underline{y},\overline{y}] &= [\min\{\underline{x}*\underline{y},\underline{x}*\overline{y},\overline{x}*\underline{y},\overline{x}*\overline{y}\}, \\
&\qquad \max\{\underline{x}*\underline{y},\underline{x}*\overline{y},\overline{x}*\underline{y},\overline{x}*\overline{y}\}]
\end{aligned}
$$

- Let an **inclusion function** $[f] : \mathbb{IR} \to \mathbb{IR}$ for $f : \mathbb{R} \to \mathbb{R}$ is defined as:

$$\{f(a) \mid \exists a \in [I]\} \subseteq [f]([I])$$

  with $a \in \mathbb{R}$ and $I \in \mathbb{IR}$.

Example of inclusion function: Natural inclusion

$[x] = [1,2]$, $\quad [y] = [-1,3]$, and $\quad f(x,y) = xy + x$

$$[f]\big([x],[y]\big) := [x] * [y] + [x]$$

$$= [1,2] * [-1,3] + [1,2] = [-2,6] + [1,2] = [-1,8]$$

Numerical Constraint Satisfaction Problems

NCSP

A NCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is defined as follows:

- $\mathcal{V} := \{v_1, \ldots, v_n\}$ is a finite set of variables which can also be represented by the vector $\mathbf{v}$;
- $\mathcal{D} := \{[v_1], \ldots, [v_n]\}$ is a set of intervals such that $[v_i]$ contains all possible values of v_i. It can be represented by a box $[\mathbf{v}]$ gathering all $[v_i]$;
- $\mathcal{C} := \{c_1, \ldots, c_m\}$ is a set of constraints of the form $c_i(\mathbf{v}) \equiv f_i(\mathbf{v}) = 0$ or $c_i(\mathbf{v}) \equiv g_i(\mathbf{v}) \leqslant 0$, with $f_i : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$ for $1 \leqslant i \leqslant m$.
  Note: Constraints $\mathcal{C}$ are interpreted as a conjunction of equalities and inequalities.

Remark: The solution of a NCSP is a valuation of $\mathbf{v}$ ranging in $[\mathbf{v}]$ and satisfying the constraints $\mathcal{C}$.

Example

- $\mathcal{V} = \{x\}$
- $\mathcal{D}_x = \big\{[1, 10]\big\}$ $\quad\Longrightarrow\quad$ $x \in [1, 1.09861]$
- $\mathcal{C} = \big\{x \exp(x) \leqslant 3\big\}$

Remark: if $[\mathbf{v}] = \emptyset$ then the problem is not satistafiable

IBEX in one slide

- Easy definition of functions
- Numerical constraints
- Pruning methods
- Interval evaluation of functions

```
#include "ibex.h"

using namespace std;
using namespace ibex;

int main() {

  Variable x;
  Function f (x, x*exp(x));

  NumConstraint c1(x, f(x) <= 3.0);

  CtcFwdBwd contractor(c1);

  IntervalVector box(1);
  box[0]=Interval(1,10);

  cout << "f" << box << " = " << f.eval(box) << endl;
  contractor.contract(box);
  cout << "after contraction box = " << box << endl;
}
```

IBEX is also a parametric solver of constraints, an optimizer, etc.

Quantified Constraint Satisfaction Differential Problems

$$S \equiv \dot{\mathbf{y}} = f(\mathbf{y}(t), u(t), \mathbf{p})$$

QCSDP

Let S be a differential system and $t_{\text{end}} \in \mathbb{R}_+$ the time limit. A QCSDP is a NCSP defined by

- a set of variables $\mathcal{V}$ including *at least* t, a vector $\mathbf{y}_0$, $\mathbf{p}$, $\mathbf{u}$
  We represent these variables by the vector $\mathbf{v}$;
- an initial domain $\mathcal{D}$ containing *at least* $[0, t_{\text{end}}]$, $\mathcal{Y}_0$, $\mathcal{U}$, and $\mathcal{P}$;
- a set of constraints $\mathcal{C} = \{c_1, \ldots, c_e\}$ composed of predicates over sets, that is, constraints of the form

$$c_i \equiv Q\mathbf{v} \in \mathcal{D}_i.f_i(\mathbf{v}) \diamond \mathcal{A}, \qquad \forall 1 \leqslant i \leqslant e$$

  with $Q \in \{\exists, \forall\}$, $f_i : \wp(\mathbb{R}^{|\mathcal{V}|}) \to \wp(\mathbb{R}^q)$ stands for non-linear arithmetic expressions defined over variables $\mathbf{v}$ and solution of differential system S, $\mathbf{y}(t; \mathbf{y}_0, \mathbf{p}, \mathbf{u}) \equiv \mathbf{y}(\mathbf{v})$, $\diamond \in \{\subseteq, \cap_\emptyset\}$ and $\mathcal{A} \subseteq \mathbb{R}^q$ where $q > 0$.

Note: we follow the same approach that Goldsztejn et al.[2]

[2] Including ODE Based Constraints in the Standard CP Framework, CP10

DynIBEX: a Box-QCSDP solver with restrictions

Solving arbitrary quantified constraints is hard!

We focus on particular problems of robotics involving quantifiers

- Robust controller synthesis: $\exists \mathbf{u}$, $\forall \mathbf{p}$, $\forall \mathbf{y}_0$ + temporal constraints
- Parameter synthesis: $\exists \mathbf{p}$, $\forall \mathbf{u}$, $\forall \mathbf{y}_0$ + temporal constraints
- etc.

We also defined a set of temporal constraints useful to analyze/design robotic application.

Verbal property	QCSDP translation
Stay in $\mathcal{A}$	$\forall t \in [0, t_{\text{end}}],\ [\mathbf{y}](t, \mathbf{v}') \subseteq \text{Int}(\mathcal{A})$
In $\mathcal{A}$ at τ	$\exists t \in [0, t_{\text{end}}],\ [\mathbf{y}](t, \mathbf{v}') \subseteq \text{Int}(\mathcal{A})$
Has crossed $\mathcal{A}$*	$\exists t \in [0, t_{\text{end}}],\ [\mathbf{y}](t, \mathbf{v}') \cap \text{Hull}(\mathcal{A}) \neq \emptyset$
Go out $\mathcal{A}$	$\exists t \in [0, t_{\text{end}}],\ [\mathbf{y}](t, \mathbf{v}') \cap \text{Hull}(\mathcal{A}) = \emptyset$
Has reached $\mathcal{A}$*	$[\mathbf{y}](t_{\text{end}}, \mathbf{v}') \cap \text{Hull}(\mathcal{A}) \neq \emptyset$
Finished in $\mathcal{A}$	$[\mathbf{y}](t_{\text{end}}, \mathbf{v}') \subseteq \text{Int}(\mathcal{A})$

*: shall be used in negative form

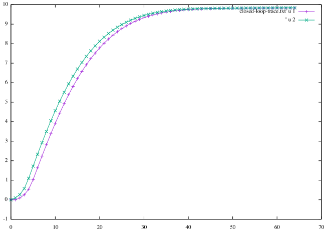
Simulation of a closed-loop system with safety

A simple dynamics of a car with a PI controller

$$\begin{pmatrix} \dot{y} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} \frac{k_p(10.0-y)+k_i w-50.0y-0.4y^2}{m} \\ 10.0-y \end{pmatrix} \quad \text{with} \quad m \in [990, 1010], k_p = 1440, k_i = 35$$

and a safety propriety

$$\forall t, y(t) \in [0, 11]$$



Failure

$$y([0, 0.0066443]) \in [-0.00143723, 0.0966555]$$

Simulation of a closed-loop system with safety property

```
#include "ibex.h"

using namespace ibex;

int main(){
  const int n = 2;
  Variable y(n);

  IntervalVector state(n);
  state[0] = 0.0; state[1] = 0.0;

  // Dynamique d'une voiture avec incertitude sur sa masse + PI
  Function ydot(y, Return ((1440.0 * (10.0 - y[0]) + 35.0 * y[1]  - y[0] * (50.0  + 0.4 * y[0]))
                          / Interval (990, 1010),
                          10.0 - y[0]));
  ivp_ode vdp = ivp_ode(ydot, 0.0, state);

  simulation simu = simulation(&vdp, 10.0, RK4, 1e-6);
  simu.run_simulation();

  // verification de surete
  IntervalVector safe(n);
  safe[0] = Interval(0.0, 11.0);
  bool flag = simu.stayed_in (safe);
  if (!flag) {
    std::cerr << "error safety violation" << std::endl;
  }

  return 0;
}
```

Case study – tuning PI controller [SYNCOP'15]

A cruise control system two formulations:

- uncertain linear dynamics;

$$\dot{v} = \frac{u - bv}{m}$$

- uncertain non-linear dynamics

$$\dot{v} = \frac{u - bv - 0.5\rho CdAv^2}{m}$$

with

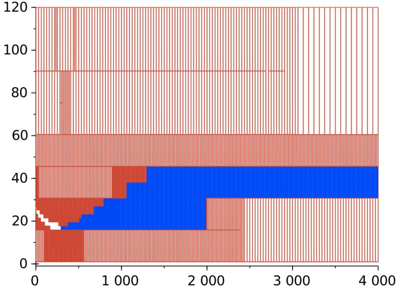
- m the mass of the vehicle
- u the control force defined by a PI controller
- bv is the rolling resistance
- $F_{\text{drag}} = 0.5\rho CdAv^2$ is the aerodynamic drag (ρ the air density, CdA the drag coefficient depending of the vehicle area)

Case study – paving results

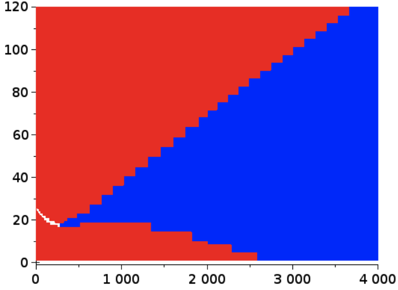
Result of paving for both cases with

- $K_p \in [1, 4000]$ and $K_i \in [1, 120]$
- $v_{\text{set}} = 10$, $t_{\text{end}} = 15$, $\alpha = 2\%$ and $\epsilon = 0.2$ and minimal size=1
- constraints: $y(t_{end}) \in [r - \alpha\%, r + \alpha\%]$ and $\dot{y}(t_{end}) \in [-\epsilon, \epsilon]$

Linear case (CPU $\approx$ 10 minutes)



Non-linear case (CPU $\approx$ 80 minutes)

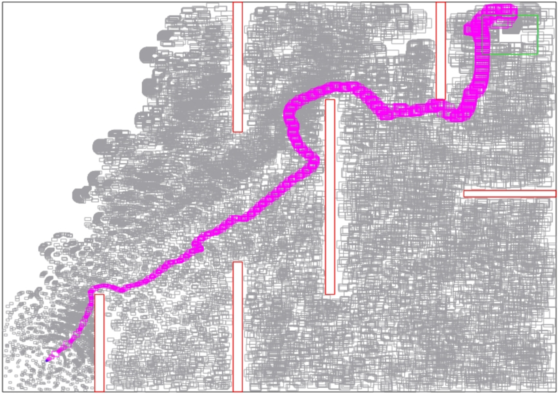


Robust path planer – 1

Enhancement of Box-RRT (Pepy *et al.*) with

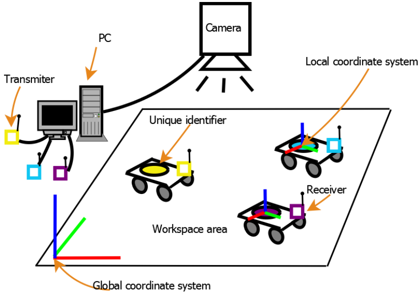
- dedicated control law
- cost function to minimize distance (Box-RRT*)

$$\exists K > 0 \text{ and } \mathbf{u} \in \mathbb{U} \text{ such that}$$
$$\forall \mathbf{s}_0 \in \mathbb{S}_{\text{init}},\ \forall\ \mathbf{s}(K\Delta t; \mathbf{s}_0) \in \mathbb{S}_{\text{goal}} \text{ and } \forall t \in [0, K\Delta t],\ \mathbf{s}(t; \mathbf{s}_0) \in \mathbb{S}_{\text{free}},$$



Robust path planer – 2

Experimental table with a fully mastered environment using ROS



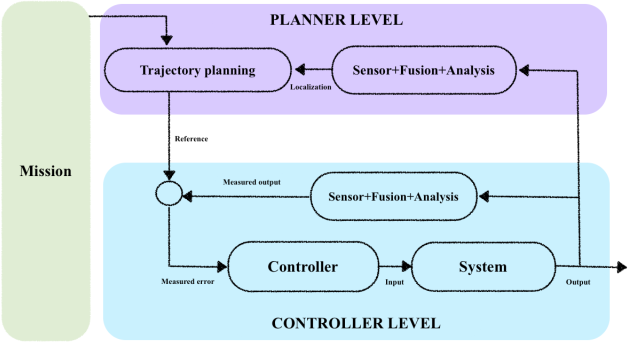
Implementation of Box-RRT* on embedded systems to

- understand interaction between planner and controller
- understand interaction between sensor and Box-RRT*

Ongoing project: safety for mobile robots

DGA MRIS project with École polytechnique and ENSTA Bretagne

Autonomous vehicles

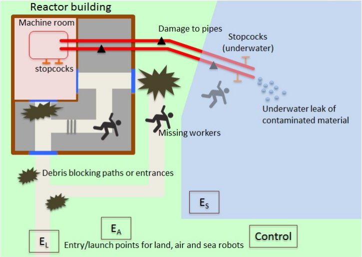


Main goals of the project:

- understand main pieces of the system and validate their behaviors
- validate the behaviors of the overall system.

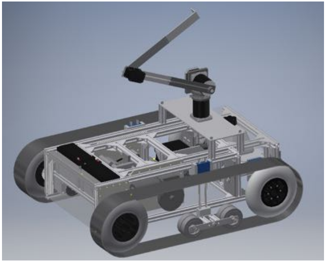
and also student project in Robotics

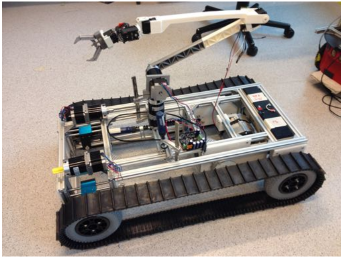
ERL Emergency Robots, http://www.eurathlon.eu



Team between both ENSTA

- ENSTA Bretagne: underwater and aerial robots
- ENSTA ParisTech: ground robot





Conclusion

DynIBEX is one **ingredient** of verification tools for cyber-physical systems.
It can **handle uncertainties**, can **reason on sets of trajectories**.

Also applied on

- Computation of viability kernel [SWIM'15]
- Controller synthesis of sampled switched systems [SNR'16]
- Parameter tuning in the design of mobile robots [MORSE'16]
- Motion planning of UAV [under submission]
- box-RRT* motion planning algorithm [under submission]

and enhanced with

- methods to solve algebraic-differential equations [Reliable Computing'16]
- a Box-QCSDP framework [IRC'17] and a contractor approach [SWIM'16]

Future work (a piece of)

- model checking techniques: SAT modulo ODE