Plasma:
A new SMC Checker

Axel Legay

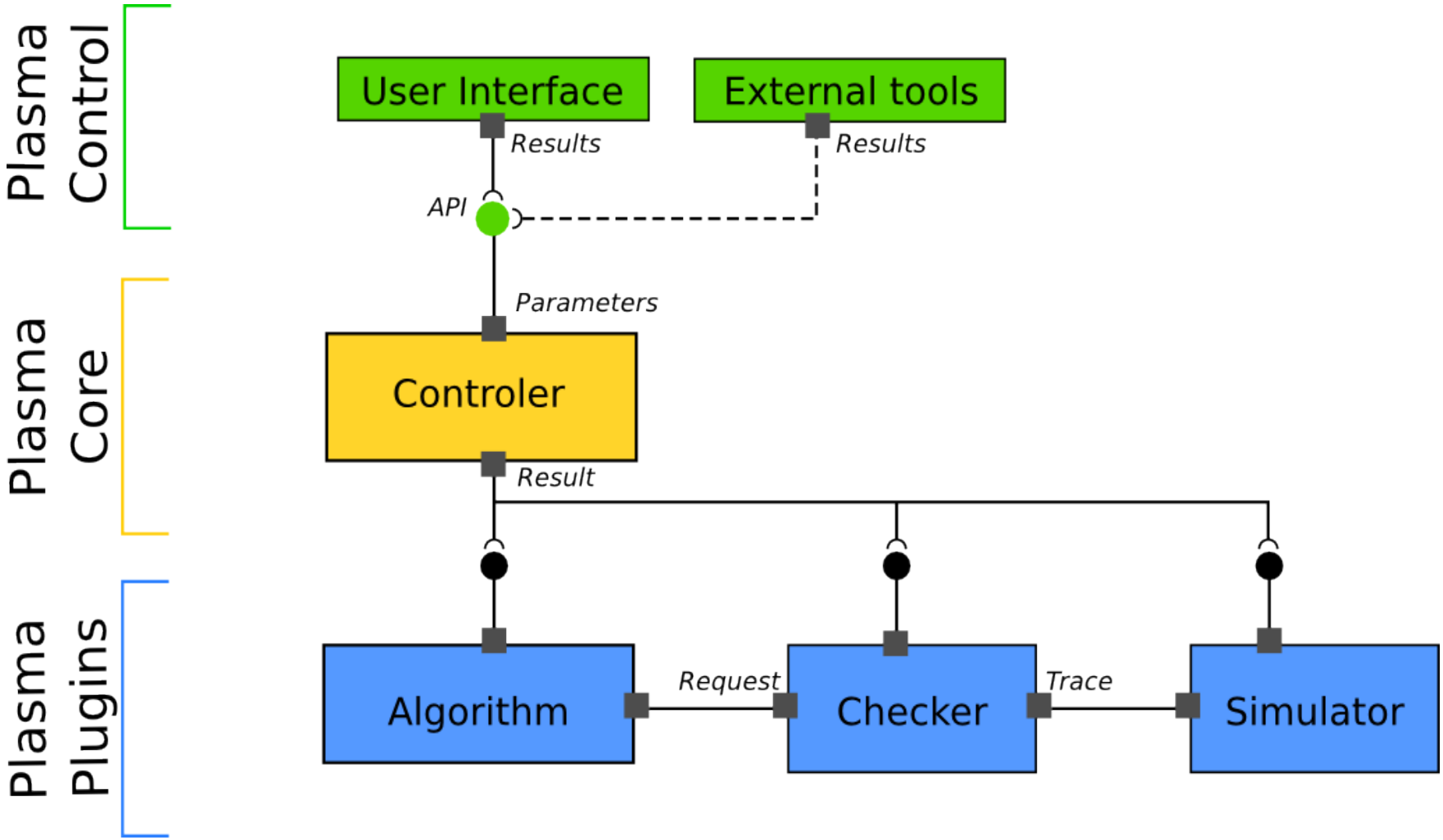
In collaboration with L. Traonouez and S. Sedwards.

Plasma Lab

A PLAtform for Statistical Model Analysis

- A library of statistical model-checking algorithms
  (Monte-Carlo, SPRT, rare events, CUSUM, nondeterminism,...)
- Generic analysis for any runnable language or model
- Easily distributed other computation grid
- An API that allows modularity:
  extendable with plugins to add new algorithms, new input languages
- Developed in Java 6

Plasma Lab Structure



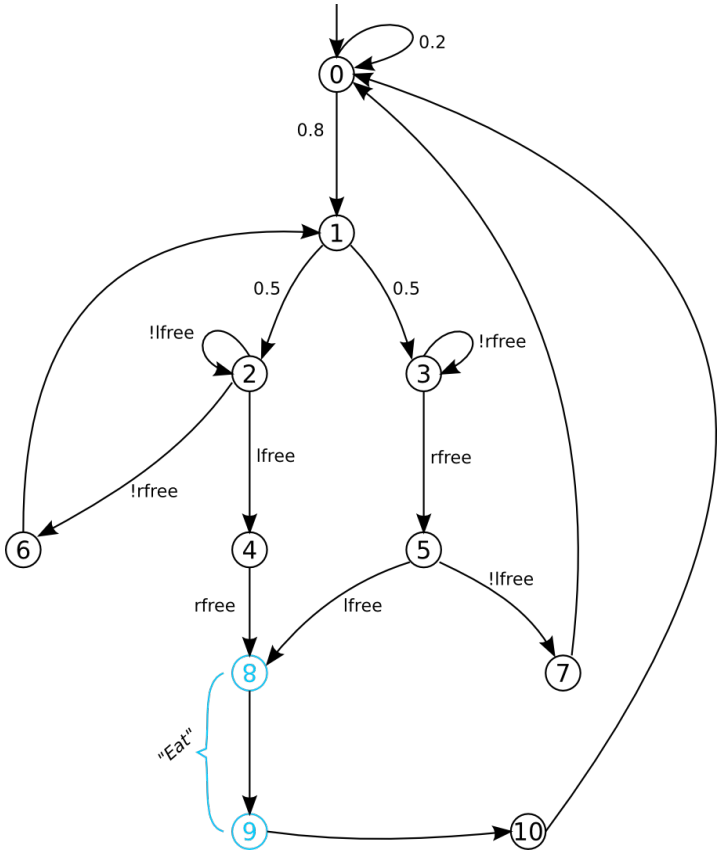
Model Language: PRISM

Input language of the model-checker PRISM:

- Textual language for modeling DTMC, CTMC, MDP, PTA
- Guarded commands transitions (systems biology):
  [synchro] guard → rate1:(action1) + rate2:(action2)
- System description via *modules renaming*
- Simulink, SystemC, ...
- And various specification languages
- Your language?

Model Language: PRISM:

Randomised dining philosophers




```
dtmc

formula lfree = p2>=0 & p2<=4 | p2=6 | p2=10;
formula rfree = p3>=0 & p3<=3 | p3=5 | p3=7;

module phil1
        p1: [0..10];
        [] p1=0 -> 0.2 : (p1'=0) + 0.8 : (p1'=1);
        [] p1=1 -> 0.5 : (p1'=2) + 0.5 : (p1'=3);
        [] p1=2 &  lfree -> (p1'=4);
        [] p1=2 & !lfree -> (p1'=2);
        [] p1=3 &  rfree -> (p1'=5);
        [] p1=3 & !rfree -> (p1'=3);
        [] p1=4 &  rfree -> (p1'=8);
        [] p1=4 & !rfree -> (p1'=6);
        [] p1=5 &  lfree -> (p1'=8);
        [] p1=5 & !lfree -> (p1'=7);
        [] p1=6  -> (p1'=1);
        [] p1=7  -> (p1'=1);
        [] p1=8  -> (p1'=9);
        [] p1=9  -> (p1'=10);
        [] p1=10 -> (p1'=0);
endmodule

module phil2  = phil1 [p1=p2,  p2=p3,  p3=p1]  endmodule
module phil3 = phil1 [p1=p3, p2=p1,  p3=p2] endmodule

// labels
label "hungry" = ((p1>0)&(p1<8))|((p2>0)&(p2<8))|((p3>0)&(p3<8));
label "eat" = ((p1>=8)&(p1<=9))|((p2>=8)&(p2<=9))|((p3>=8)&(p3<=9));
```

Model Language: Bio

Textual language for biological models:

- Write chemicals reactions with CTMC semantics:
  product1 + product2 rate-> product3 + product4

- Use Gillespie algorithm for simulating biological models:
  The time and probability of a reaction depends on its rate and the number of species.

```
species A=1000,B=1000,C,D,E

A + B -> C
C 10000 -> D
D -> E
```

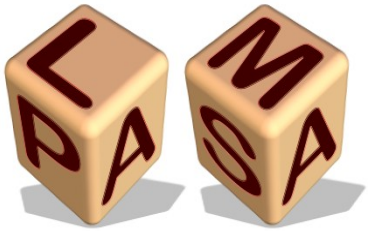
#		A	B	C	D	E
1	0.0	1000.0	1000.0	0.0	0.0	0.0
2	1.4033595075152232E-6	999.0	999.0	1.0	0.0	0.0
3	3.1610719924273105E-6	998.0	998.0	2.0	0.0	0.0
4	3.489499125856082E-6	997.0	997.0	3.0	0.0	0.0
5	4.133233001983935E-6	996.0	996.0	4.0	0.0	0.0
6	6.138193509039687E-6	996.0	996.0	3.0	1.0	0.0
7	7.56365258629189E-6	995.0	995.0	4.0	1.0	0.0
8	9.373545684697474E-6	994.0	994.0	5.0	1.0	0.0
9	1.0684090810801553E-5	993.0	993.0	6.0	1.0	0.0
10	1.0856701877068095E-5	992.0	992.0	7.0	1.0	0.0
11	1.2767547877520022E-5	991.0	991.0	8.0	1.0	0.0
12	1.2944710334570275E-5	990.0	990.0	9.0	1.0	0.0
13	1.4448346791495854E-5	989.0	989.0	10.0	1.0	0.0

Properties Language: BLTL

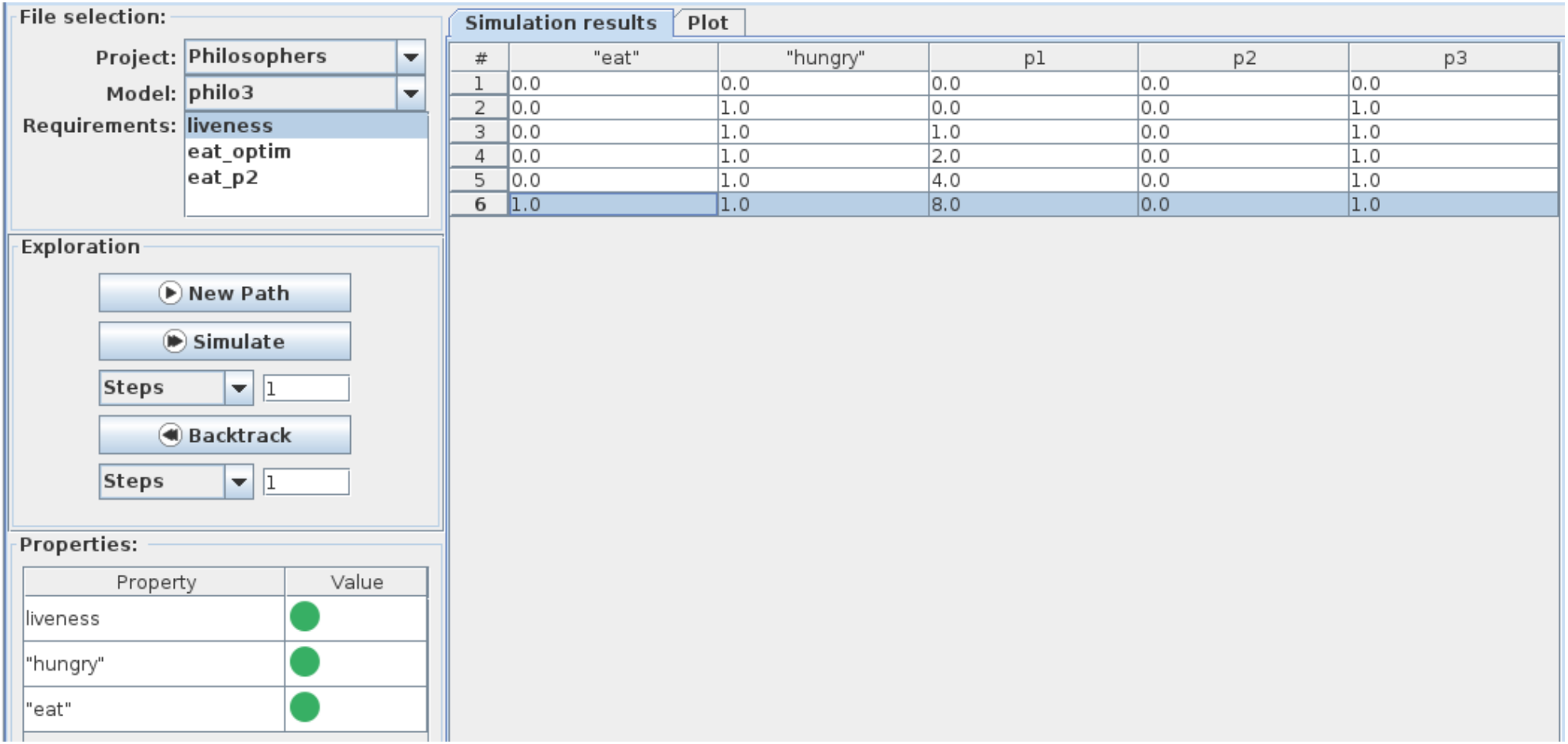
Bounded Linear Temporal Logic

- LTL with bounded temporal operators
  - F<=#50 "eat"
  - F<=#1000 ("hungry" & (X<=#1 F<=#1000 "eat"))

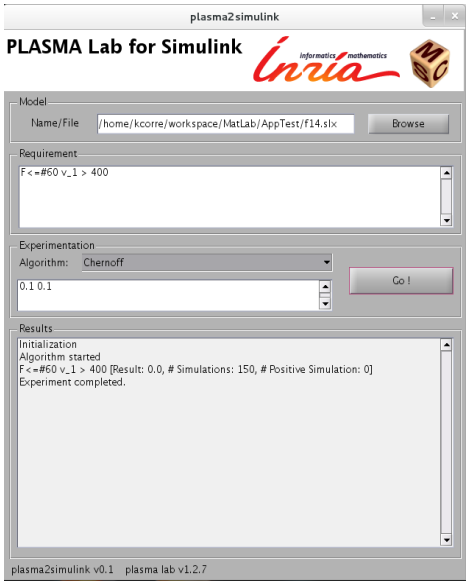
Using Plasma Lab GUI for Simulations

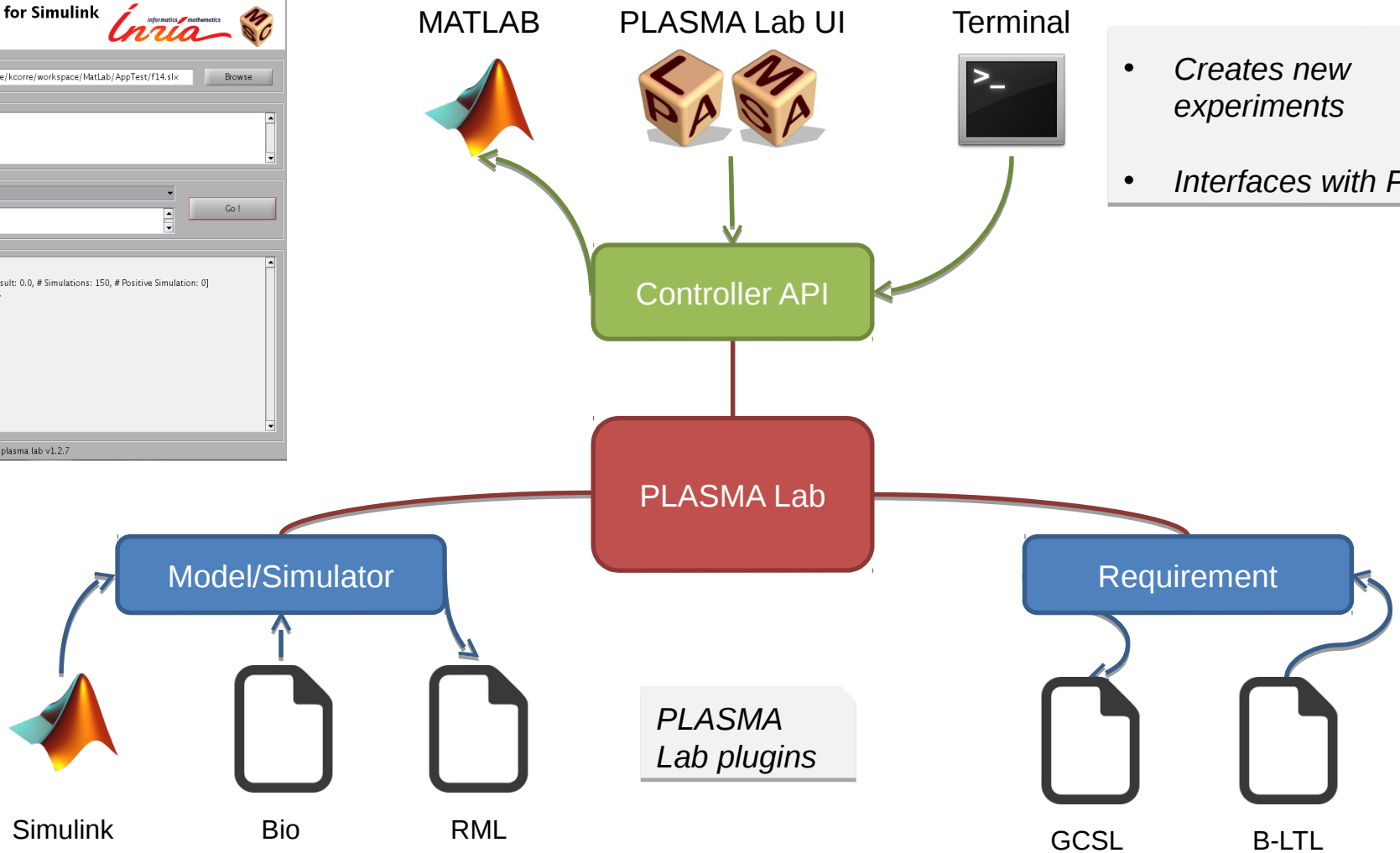


liveness = F<=#1000 ("hungry" & (X<=#1 F<=#1000 "eat"))



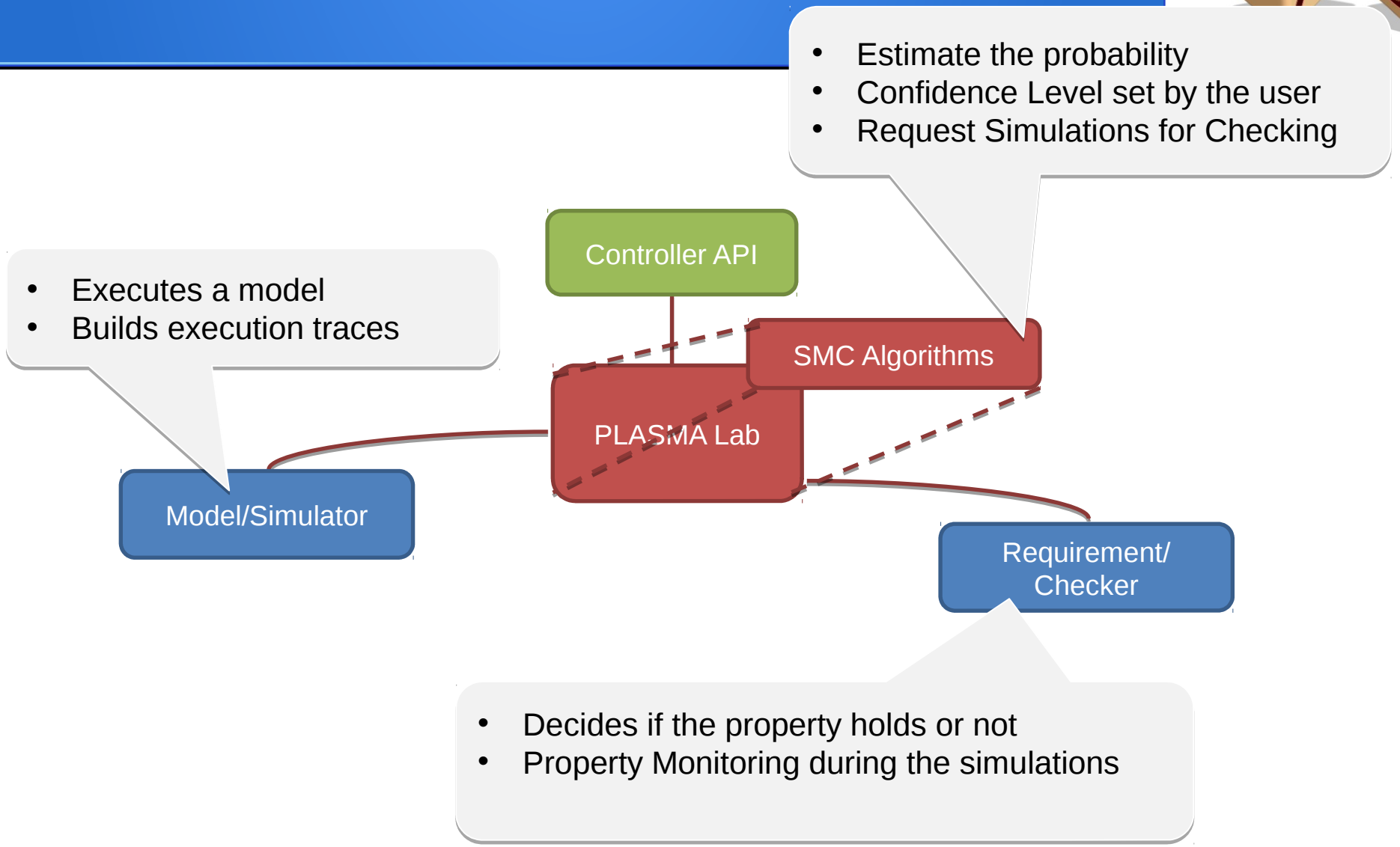
Plasma Lab API





- *Creates new experiments*
- *Interfaces with PLASMA*

Plasma Lab API



Developing New Plugins

Implement required interfaces from the API

New Simulator

- newPath()
  Start a simulation
- simulate()
  Simulate one step

New Checker

- check(path)
  - Check a trace until the property is decided
  - Return the result

New Algorithm

- run()
  - Run the algorithm
  - Send the results

Creating a New Algorithm

```
public class EMSIGSchool implements InterfaceAlgorithmScheduler {

    @Override
    public void run() {
        listener.notifyAlgorithmStarted("EMSIG");

        double res = 0.0;
        for(int i=0; i<nbSimu; i++) {
            InterfaceState path = model.newPath();
            res += requirement.check(path);
        }

        listener.notifyAlgorithmCompleted("EMSIG");
        listener.publishResults("EMSIG", new SMCResult(res/nbSimu));
    }
}
```

Start a new trace

Check the trace
and collect the result

Send the results
to the user interface

Application1: Dali

- European project
- Application of SMC beyond formal verification
  - A trolley to guide an old lady in a commercial center
  - Point of interest/repulsion
  - Embedded application, beyond software
  - Limited ressources
- Hot topic: national press, euronews, ...

Objectives

- Develop technologies to provide our system with a robust decision making mechanism that plans the motion of the AP from a source to a destination.
- The motion is in an environment populated by human agents and fixed obstacles.
- The objective is to keep in check the probability of accidents or hazards by offsetting possibly uncooperative behaviours of the AP.

Challenges

- To build a mathematical model to reason on the AP in its environment
- To design a planning algorithm; this algorithm will rely on the math model
- The resulting algorithm has to be embedded in the trolley
- This requires to model sensors and external environment as mathematical objects.

Approach

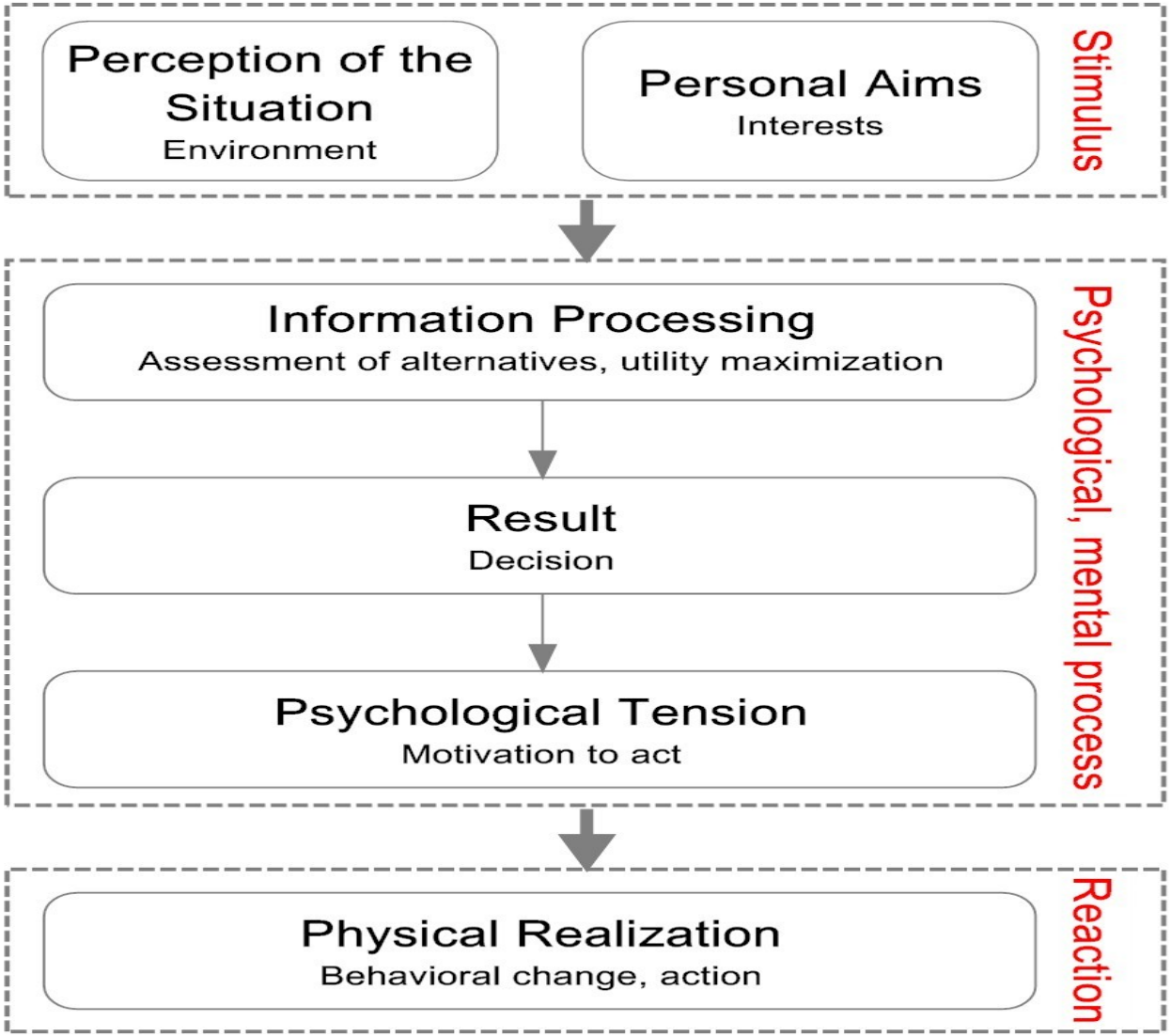
- A Markovian model that tracks the status of the AP and its environment
- The model is parametrised with variables representing the external environments
- 'Social force' model to reason on human motions in crowded areas
- Social force together with Statistical Model Checking helps the planing algorithm (motion planer) to predict safe moves for the AP

The social force model

- We need to model human behavior in a crowded environment
- We have adopted the 'social force model' (SFM)
- The SFM models groups of people having goals, using repulsive and attractive social forces
- E.g., as the AP gets closer to an object it reduces its speed and changes its trajectory; if the object is further than 5m, the AP makes no change to its trajectory.

Task 3.1: The social force model

EC Grant Agreement n. 288917



EC Grant Agreement n. 288917

Task 3.1: Mathematical elements of the social force model

objectives + hypothesised trajectory

social + physical forces

random movement

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \frac{\mathbf{v}_i^0 - \mathbf{v}_i}{\tau_i} + \frac{\mathbf{f}_i + \boldsymbol{\xi}_i}{m_i}$$

$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \mathbf{v}_i$$

actual position

actual velocity

reaction time

mass

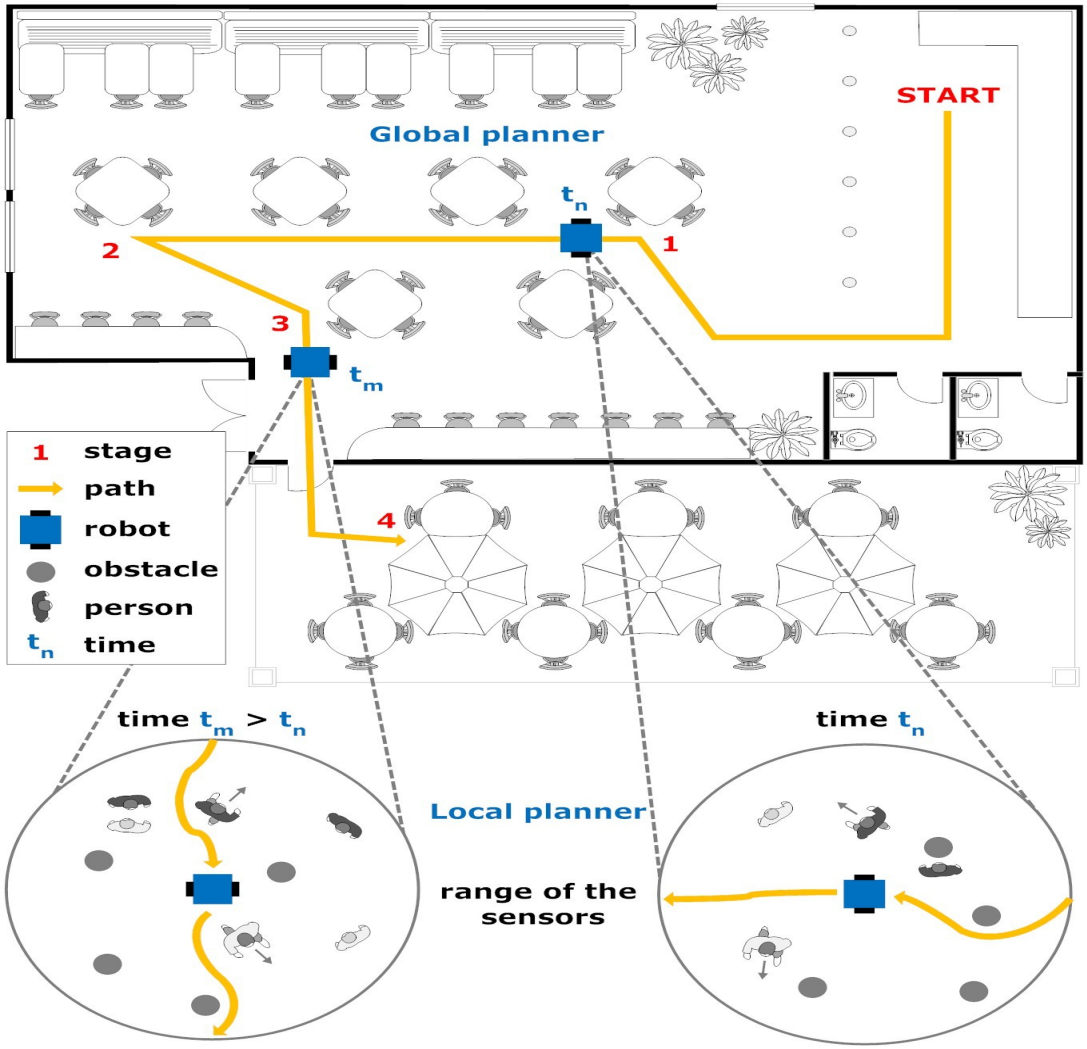
SEVENTH FRAMEWORK PROGRAMME

Two types of planing

Approach:

- We distinguish between local and global planning
- Statistical model checking helps to make the “best and most natural decision” in highly dynamical environment (local planning)
- Predictor uses social forces combined with statistical algorithms

Planning


START
Global planner
t_n
1
2
3
t_m
4
1 stage
path
robot
obstacle
person
t_n time
time t_m > t_n
time t_n
Local planner
range of the sensors

Two types of planning

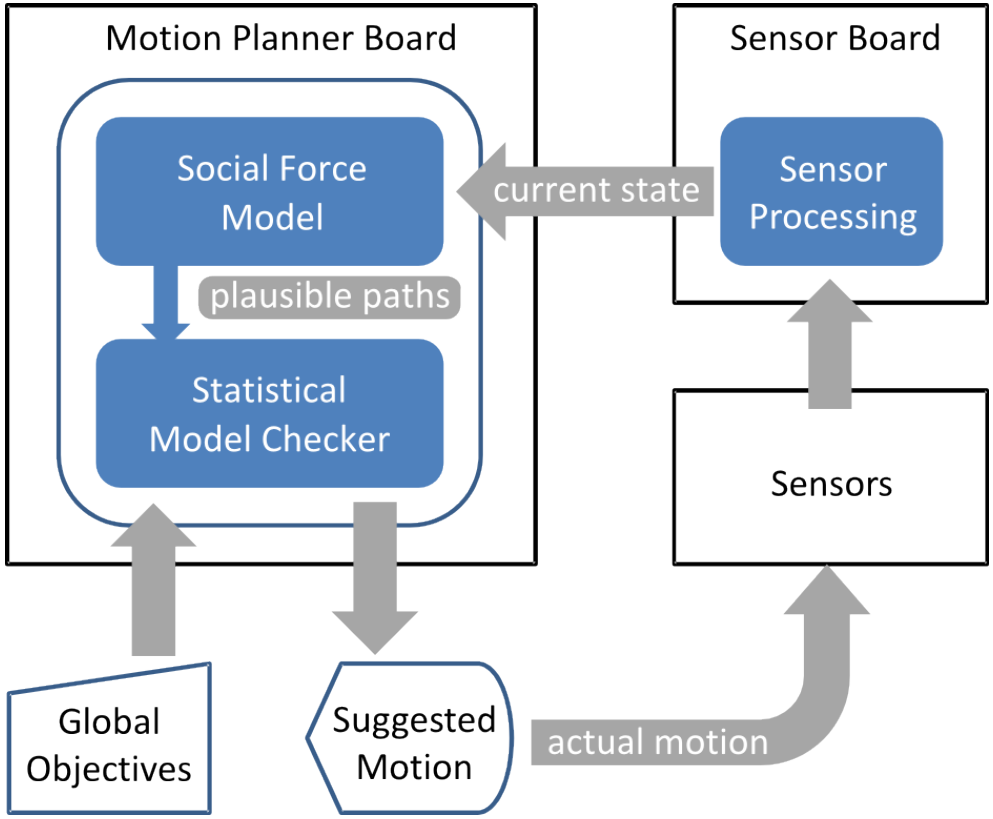
- The global planning uses 'static' information: it assumes the existence of a map and uses algorithms from GPS technology to derive the best path to reach a goal starting from an initial point

- The local planning uses dynamic information: the algorithm takes account of the global goal and sensor information
  - the objective is to follow the path suggested by the local planner and avoid collisions
  - The local planner is constantly active.
  - The global planner is re-activated when overall progress is too slow or when user objectives change (future work).

Local planning

- The challenge is to cope with unpredictable moving objects and environmental changes to avoid collisions
- First level: mathematical (social force) model
  - alone, not sufficiently reactive / predictive
- Second level: manage mathematical model with statistical model checking (SMC) using PLASMA
- PLASMA uses long term predictive simulations to derive the best path to be followed locally

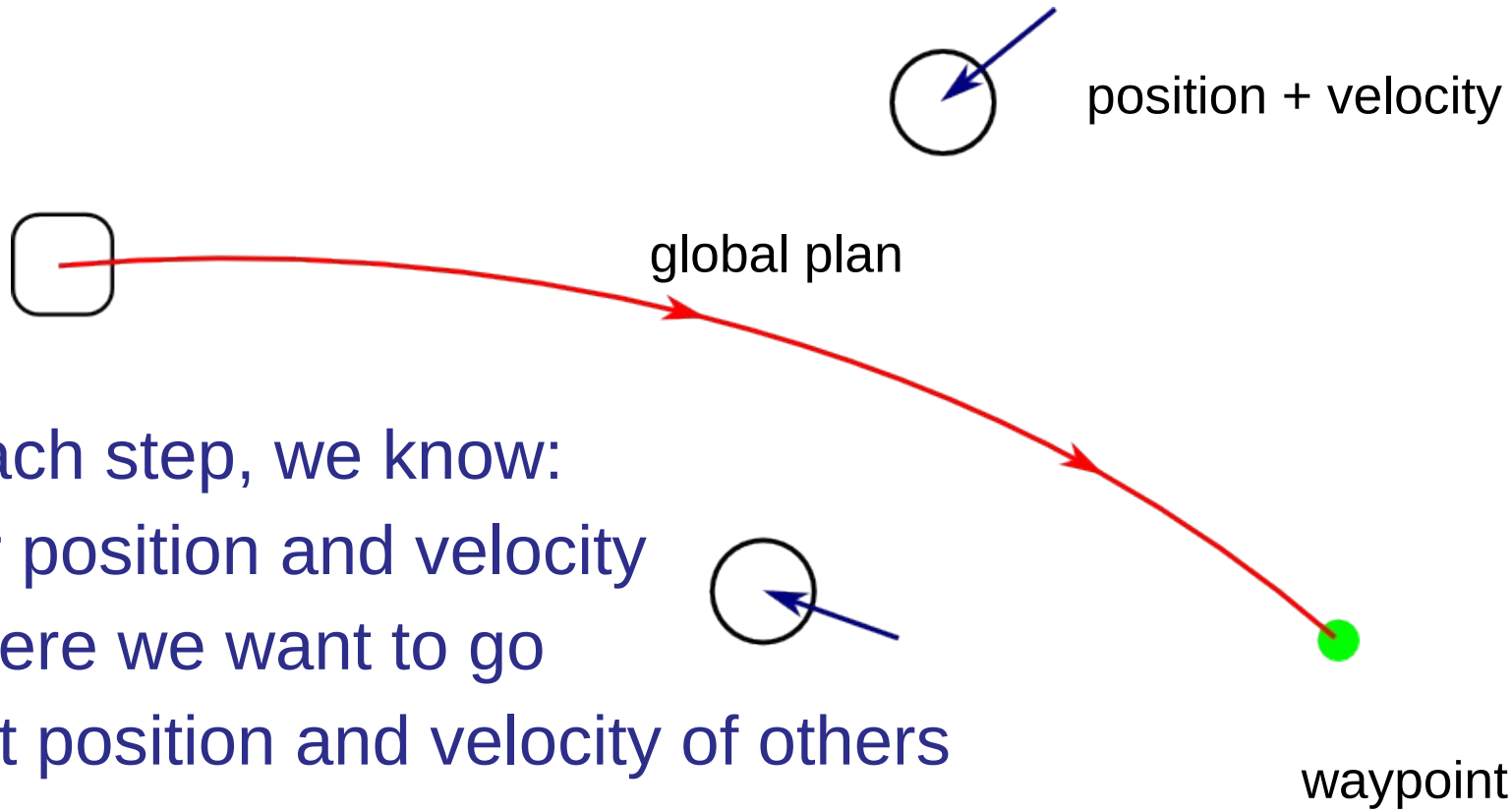
Motion planner architecture



Statistical Model Checking

- Verify temporal properties on paths
  - assumes a stochastic model
  - estimates probability of property with error bound
- Simulations provided by SFM
- Logic encodes high level objectives
  - minimum distance, maximum time, etc.
- e.g $\left(\mathrm{G}_{[0,4]} \bigwedge_{i \neq u} \underbrace{\|\mathbf{x}_u - \mathbf{x}_i\|}_{\text{dist. between user } \mathbf{x}_u \text{ and others } \mathbf{x}_i} > 0.5\right) \wedge \left(\mathrm{F}_{[0,4]} \underbrace{\|\mathbf{x}_u - \mathbf{w}\|}_{\text{dist. between user } \mathbf{x}_u \text{ and objective } \mathbf{w}} < 0.2\right)$
- Enhances predictive power of SFM

Initial information

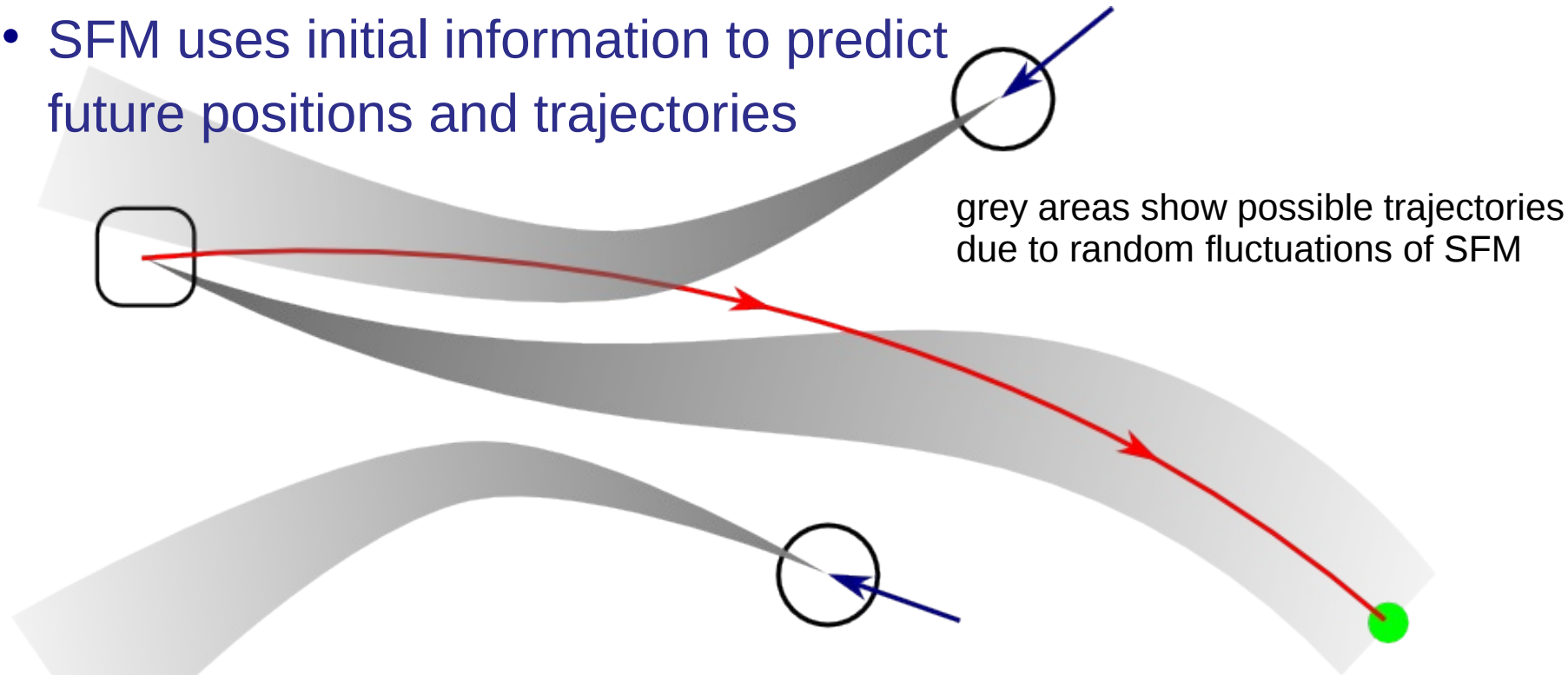


At each step, we know:

- our position and velocity
- where we want to go
- last position and velocity of others

SFM without correction

- SFM uses initial information to predict future positions and trajectories



- Without correction, agents may get arbitrarily close
  - potential for collision

SFM + SMC using hypothesised alternative directions

- We cannot directly change the trajectories of others
- We *can* change our own trajectory
- At each step we
  - hypothesise alternative initial directions
    - initial angular impulses, e.g., {-60, -30, 0, 30, 60} deg.
  - perform SMC using hypothesised direction
    - check property against multiple simulated paths
  - select direction that maximises success
    - fewest problems (collisions, stress)
    - least perturbation to current trajectory

Demos

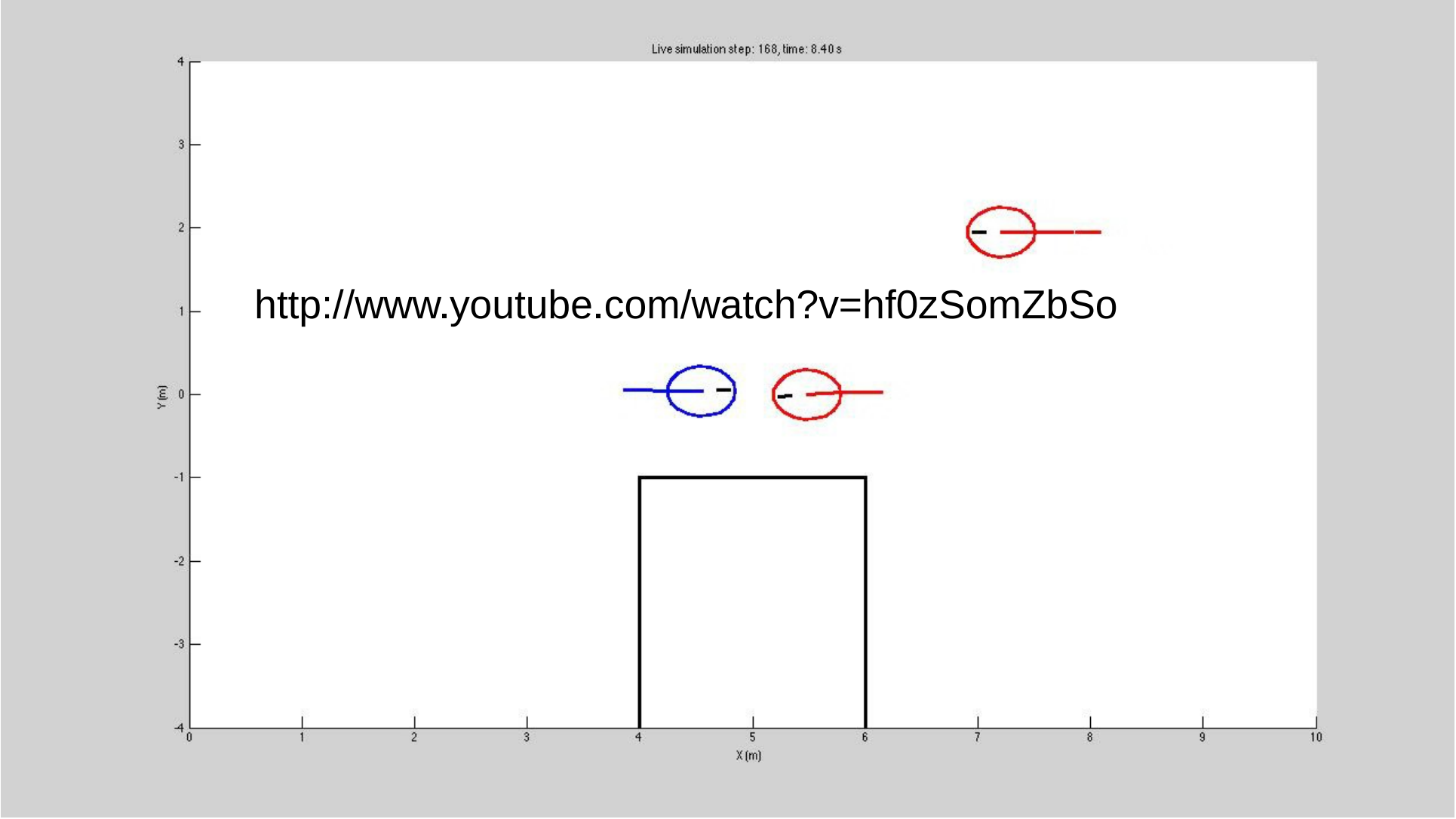
- Simulations of agents and c-Walker moving in various environments
- Each agent respects the SFM with randomness
- Every agent has a goal to reach in the environment
- The c-Walker has given a path in the environment: **global path**
- Objective for the c-Walker: follow the global path with **no collision**

Demo – Without SMC

- The evolution of the system follows just the SFM
- The c-Walker moves according to the forces suggested by its objective and the SFM alone

Motion planning with SFM alone

EC Grant Agreement n. 288917

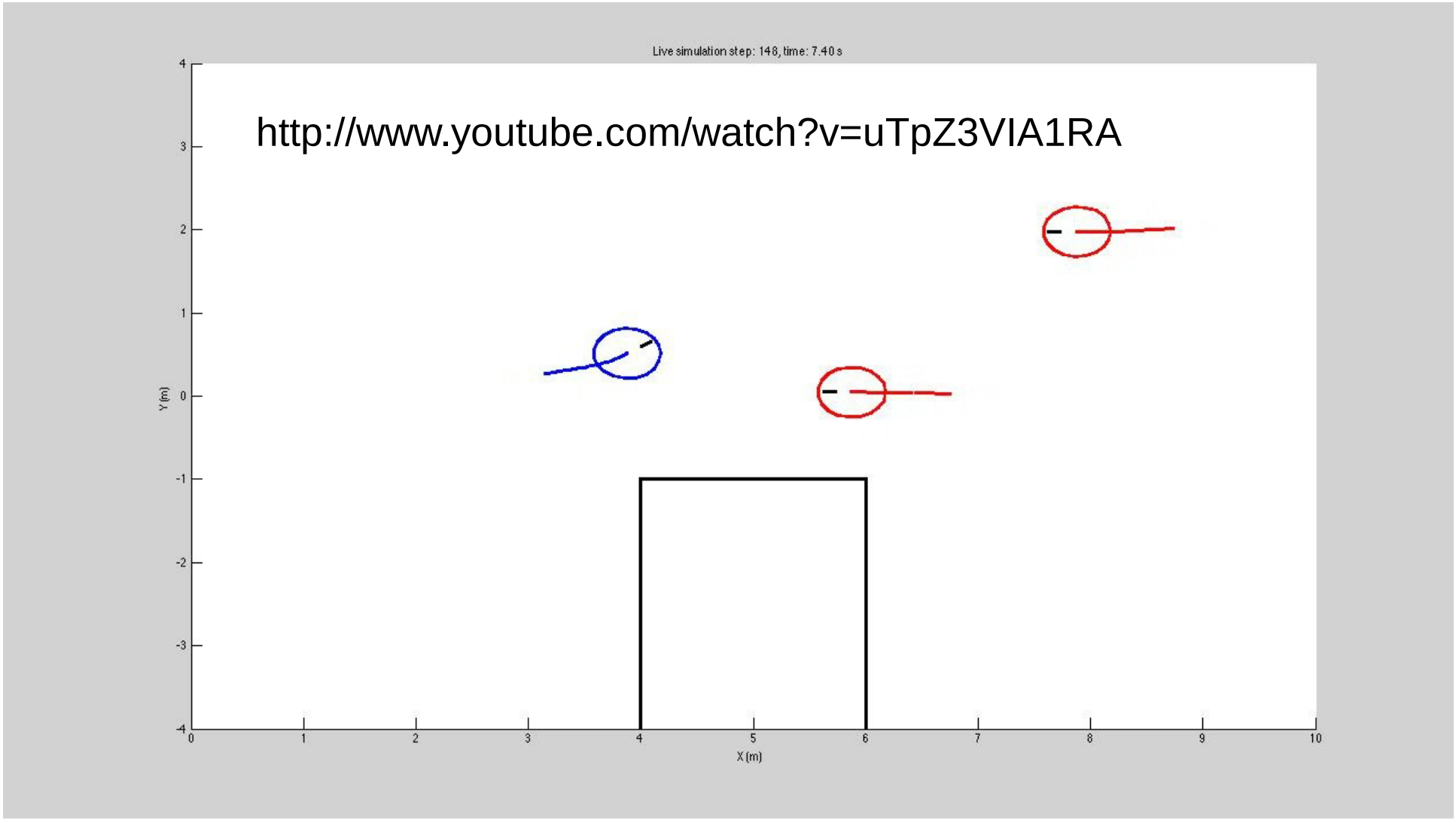


Demo – With SMC

- The c-Walker uses SFM to predict multiple probable future paths of the agents within the sensor range
- The c-Walker moves according to the forces suggested by SFM and the movement direction suggested by PLASMA

Motion planning with SFM + SMC

EC Grant Agreement n. 288917

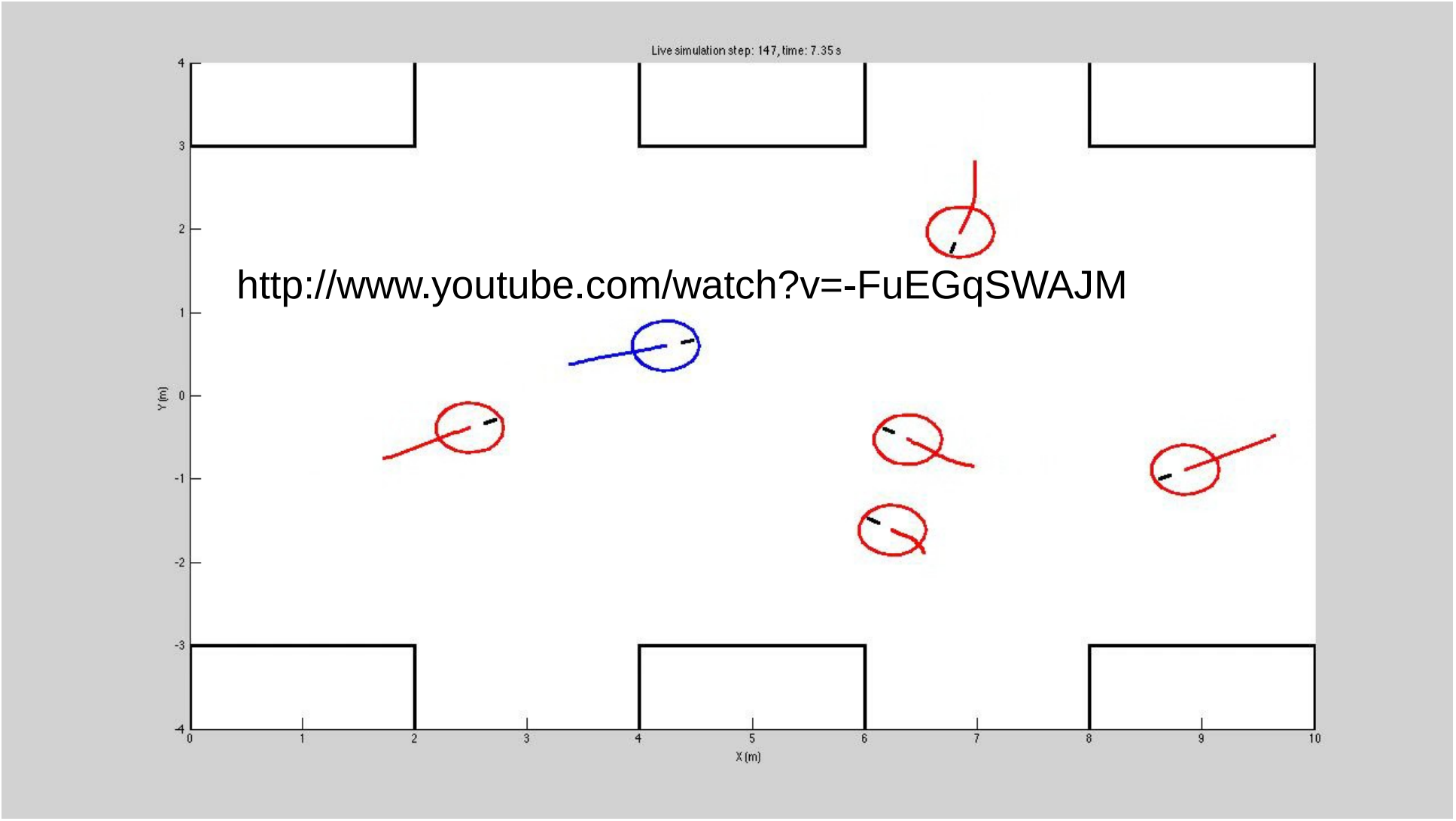


SEVENTH FRAMEWORK PROGRAMME

Demo – With SMC + SFM in complex environment

- A more realistic demonstration of the motion planner
- A complex environment of
  - fixed obstacles
  - moving agents
- Algorithm must track and simulate multiple trajectories in real time

Motion planning in a complex environment



http://www.youtube.com/watch?v=-FuEGqSWAJM

Performance on typical hardware

- Probability of success (avoiding collisions and reaching goal)
  - ~ 0.14 with SFM alone
  - ~ 0.70 with SFM + SMC
- Implemented on embedded computing device (Beagleboard)
  - no optimisations yet, but ...
- Ability to re-plan every 500ms
  - ~ 92% of the time for simple scenario
  - ~ 55% of the time for complex scenario

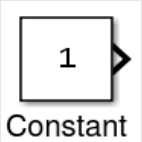
Application two: MATLAB/Simulink

Graphical modeling language for dynamic systems

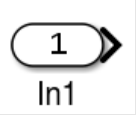
- Block library for continuous and discrete signals

Sources

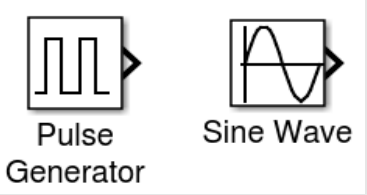
- Constant value:



- Inputs:

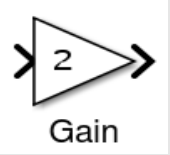


- Periodic signals:

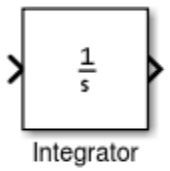


Signal transformation

- Gain:



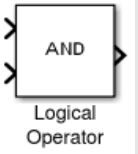
- Integrator:



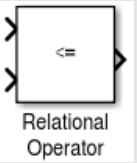
Signals composition

- Product, sum:

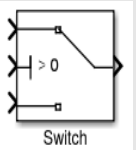
- Logical operation (AND, OR, …)



- Comparison: (<, <=, >=, >, …)

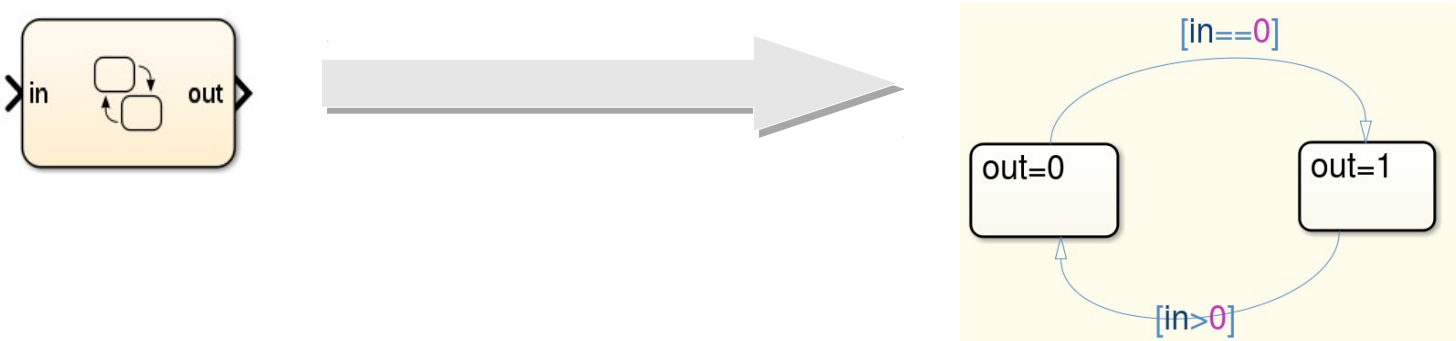


- Signals routing:

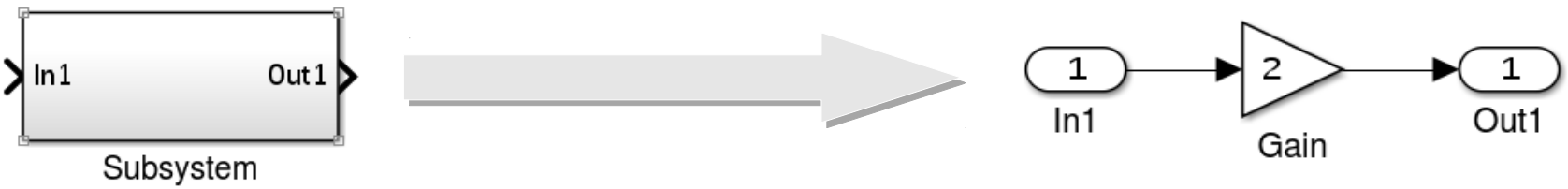


MATLAB/Simulink

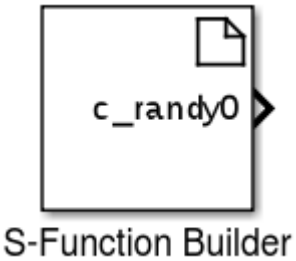
- Stateflow charts for discrete state automata:



- Hierarchical description with subsystem:



- Custom S-function blocks to include C-code:



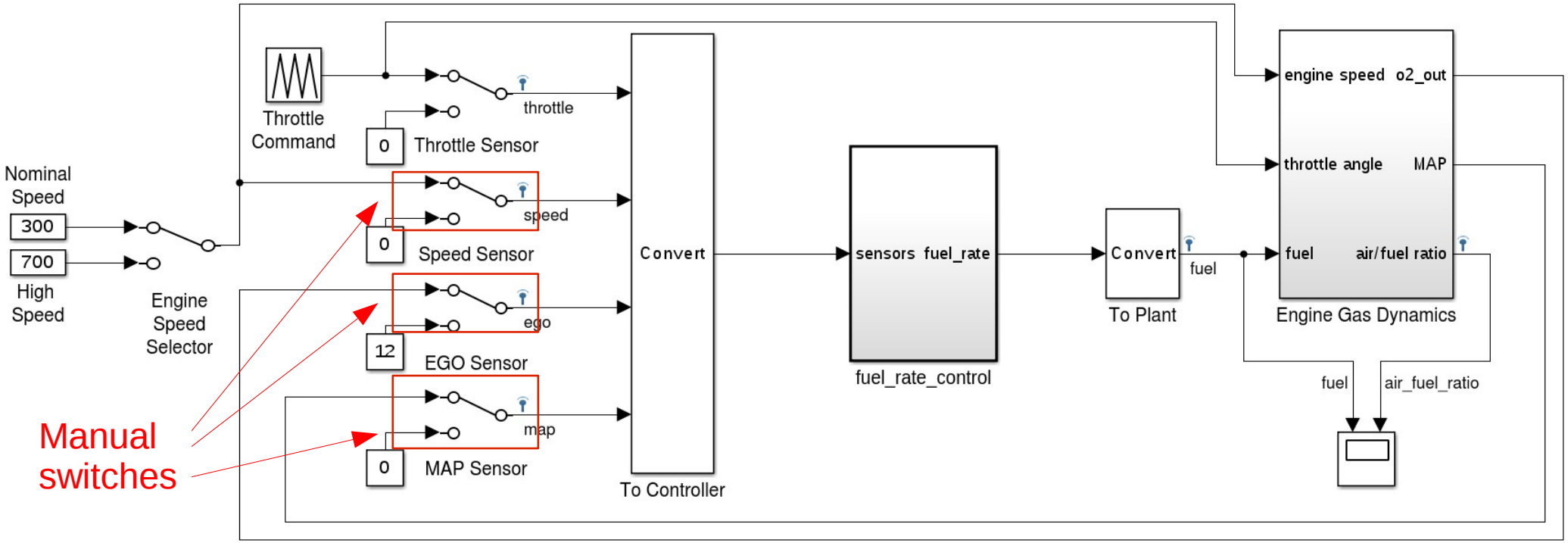
Fault-Tolerant Fuel Control System

A hybrid system with continuous and discrete dynamics:

- Robust control of the fuel distribution of a gasoline engine
- 4 sensors: throttle, speed, exhaust gas (EGO), and air pressure (MAP)
- If a sensor fails, the control system is dynamically reconfigured for uninterrupted operation

Fault-Tolerant Fuel Control System

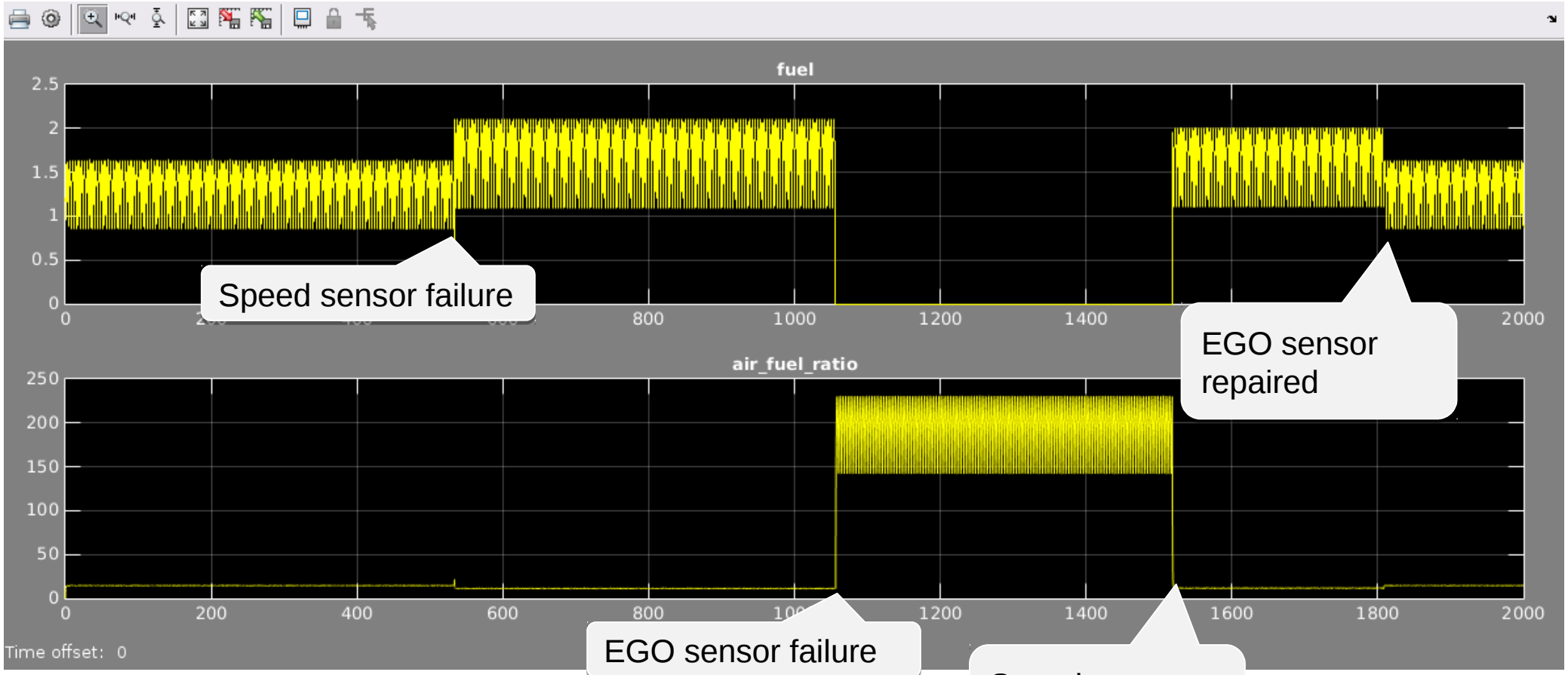
Original model



The sensor switches simulate any combination of sensor failures.
The Engine Speed Selector switch simulates different engine speeds (rad/sec).

Fault-Tolerant Fuel Control System

Simulation with manually triggered failures



Speed sensor failure

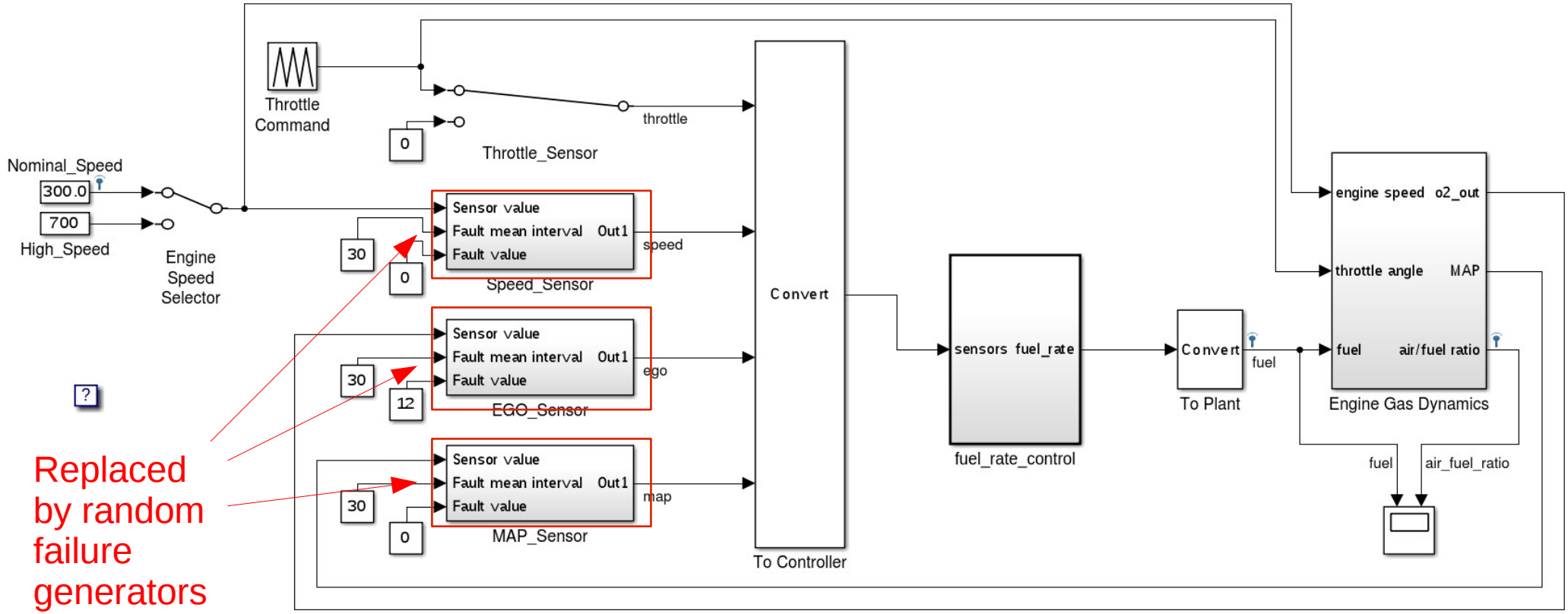
EGO sensor repaired

EGO sensor failure

Speed sensor repaired

Fault-Tolerant Fuel Control System

Introducing random failures

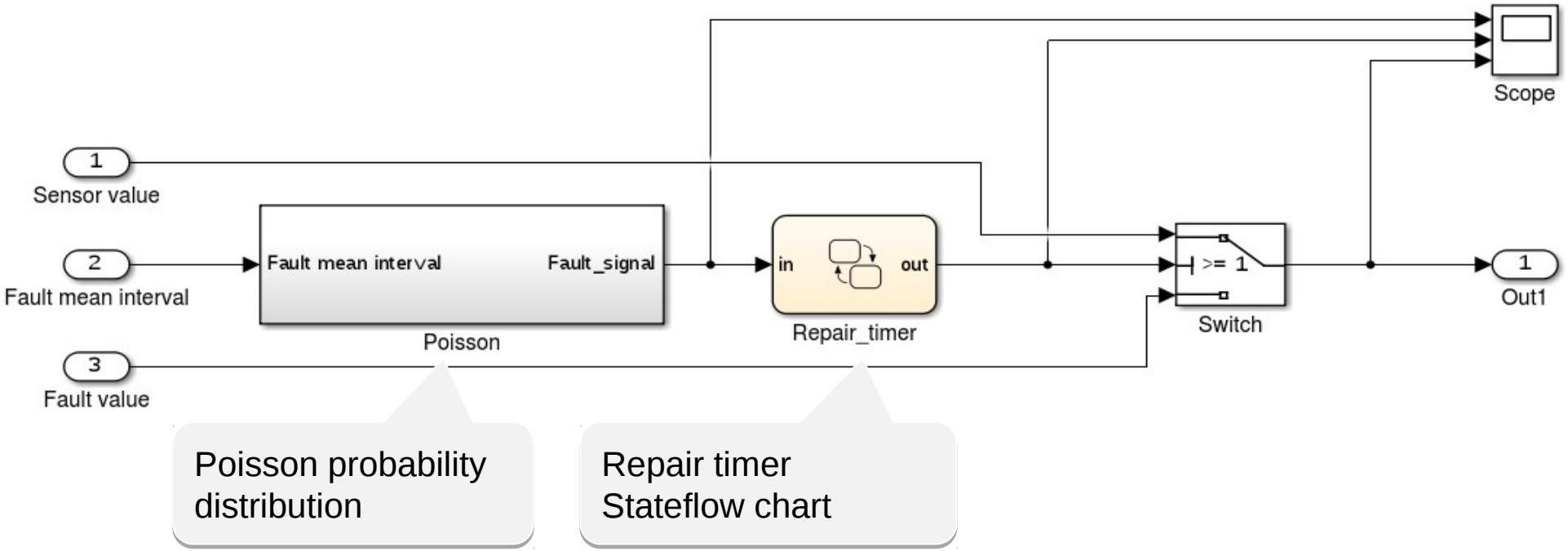


Replaced by random failure generators

The sensor switches simulate any combination of sensor failures.
The Engine Speed Selector switch simulates different engine speeds (rad/sec).

Random failures generator

- Generates random failures according to a Poisson probability distribution.
- Failures last 1 t.u. and then are automatically repaired.

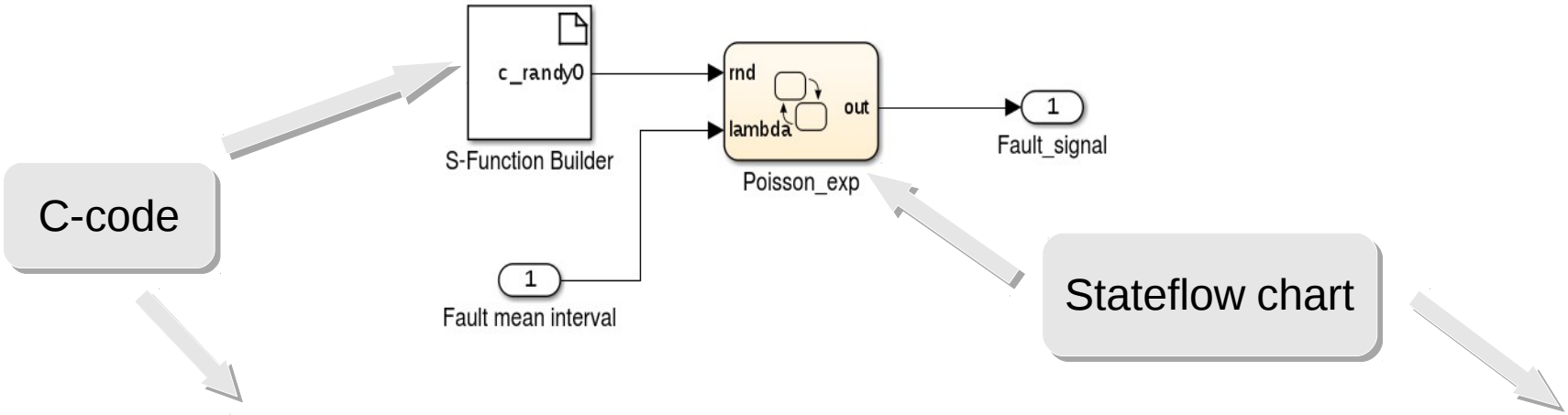


Poisson distribution subsystem

Generate failures at regular time intervals:

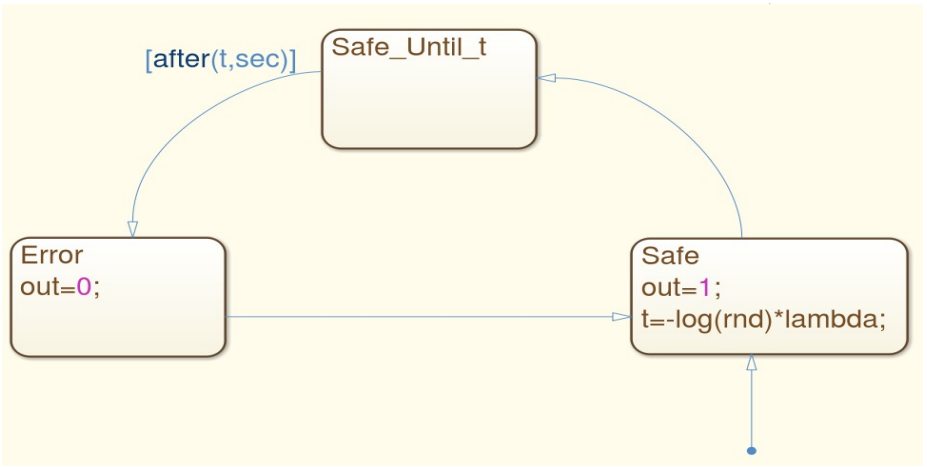
- Parameterized by the fault mean interval (lambda).
- Select a random number rnd using a C-code random generator.
- The time of the next failure is: t=-log(rnd)*lambda
- Use a Stateflow chart to update the sensor status.

Poisson distribution subsystem

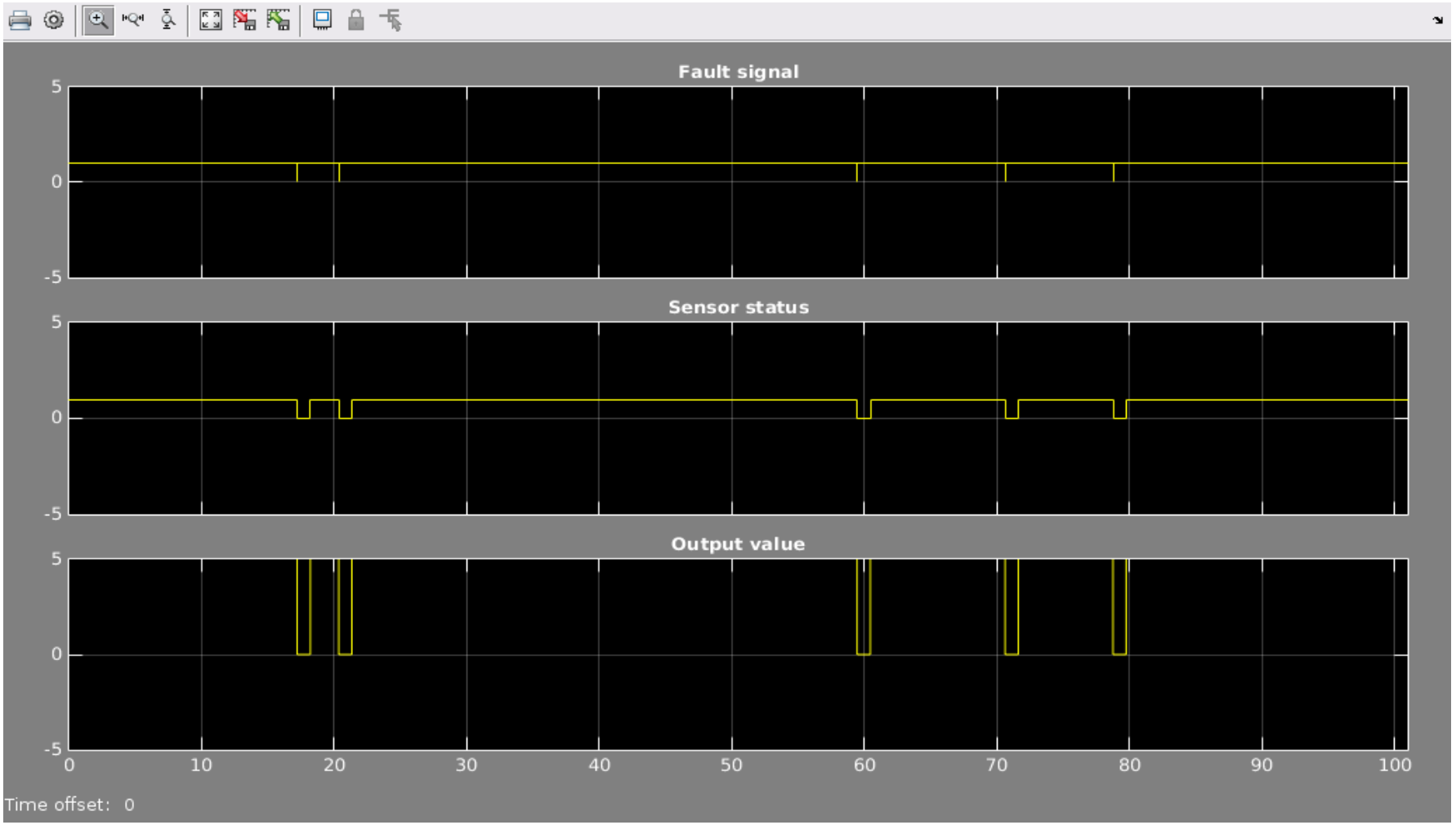



```
static bool init = true;
if(init)
{
    init = false;
    int t = time();
    srand(t);

}
double rnd;
rnd=((double) rand() / (RAND_MAX)) ;
y0[0] = rnd;
```



Random failures generator



SMC Analysis

Compute the probability that fuel distribution is stopped for at least 1 t.u. :

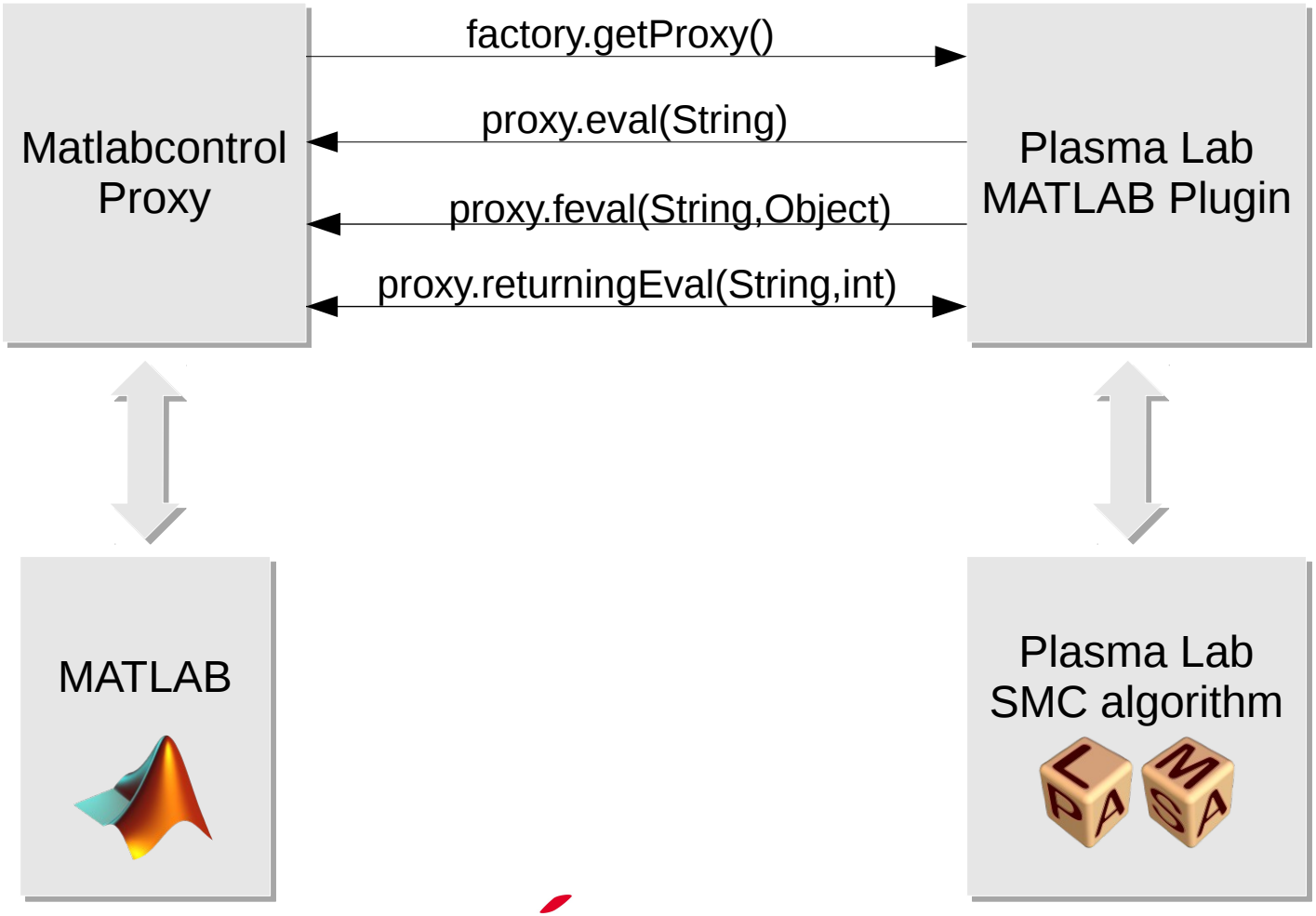
$$\Phi = F_{\leq 100}(G_{\leq 0.999}\mathsf{Fuel} = 0)$$

Plasma/Simulink Interface

A MATLAB plugin for Plasma Lab:

- Implements the simulator interfaces of Plasma API:
  - `public class MatLabIdentifier implements InterfaceIdentifier (to show the results)`
  - `public class MatLabSessionFactory implements AbstractModelFactory (to build the model)`
  - `public class MatLabSessionModel extends AbstractModel (newpath, simulate)`
  - `public class MatLabState implements InterfaceState (Type of state)`
- Use the **matlabcontrol** Java API to control Simulink simulations:
  - Create a MatlabProxy object
  - Call proxy.eval(String), proxy.feval(Sring, Object), or proxy.returningEval(String, int) to launch MATLAB commands

Plasma/Simulink Interface



Plasma/Simulink Simulation Trace

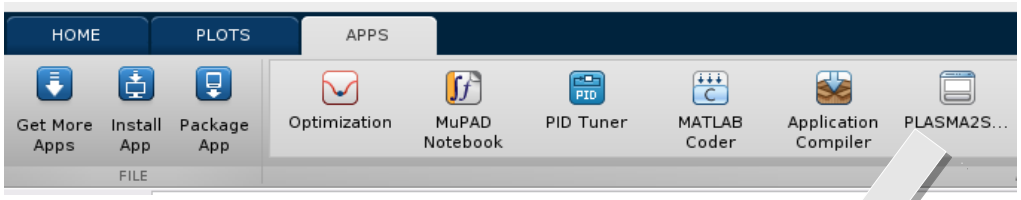
- Plasma receives the values of the signals that are logged in Simulink
- Time is discretized:
  - According to the sample time of Simulink blocks
  - At each occurrence of a discrete event in Stateflow charts

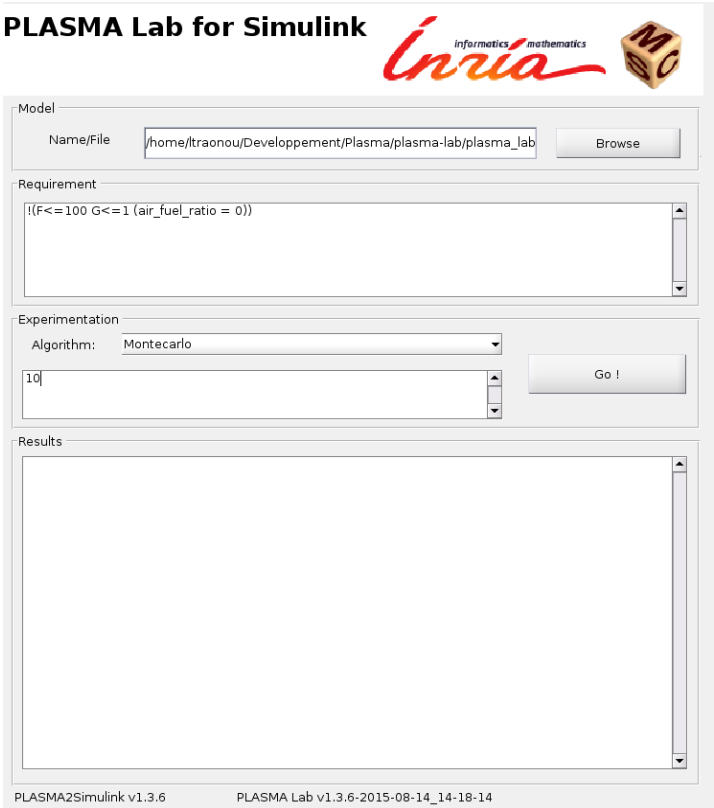
#	Time	air_fuel_ratio	fuel
1	0.0	0.0684931506849315	1.2090280055999756
2	5.619886205879556E-4	0.09245151889045834	1.2090280055999756
3	0.0033719317235277333	0.2110067826259353	1.2090280055999756
4	0.01	0.48272660930430195	1.1729249954223633
5	0.02	0.8852234347776664	1.140881061553955
6	0.03	1.2762905238743816	1.1123958826065063
7	0.04	1.6562756236081757	1.0872750282287598
8	0.05	2.0254656415942236	1.0651990175247192
9	0.05532799663845003	2.2188831929979123	1.0651990175247192
10	0.05532799663845048	2.2188831929979282	1.0651990175247192

Plasma/Simulink GUIs

Plasma MATLAB plugin can used:

- From Plasma Lab main GUI
- From Plasma2Simulink GUI, a small App that can be installed in MATLAB





Launch Plasma2Simulink from MATLAB

Interface

MATLAB R2014b

HOME | PLOTS | APPS

Get More Apps | Install App | Package App | Optimization | MuPAD Notebook | PID Tuner | MATLAB Coder | Application Compiler | PLASMA2S... | Distribution Fitting | PDE

FILE | APPS

/ ▸ home ▸ ltraonou ▸ Developpement ▸ Plasma ▸ plasma-lab ▸ plasma_lab_examples ▸ Simulink ▸ fuel_control

sldemo_fuelsys_estasys *

File Edit View Display Diagram Simulation Analysis Code Tools Help

101

sldemo_fuelsys_estasys | To Controller

sldemo_fuelsys_estasys ▸

Fault-Tolerant Fuel Control System

Throttle Command
Throttle_Sensor
Nominal_Speed
High_Speed
Engine Speed Selector
Speed_Sensor
EGO_Sensor
MAP_Sensor
Convert
To Controller
fuel_rate_control
To Plant
Engine Gas Dynamics

The sensor switches simulate any combination of sensor failures.
The Engine Speed Selector switch simulates different engine speeds (rad/sec).

Ready | 80% | ode45

plasma2simulink

PLASMA Lab for Simulink

informatics mathematics

Model

Name/File: /home/ltraonou/Developpement/Plasma/plasma-lab/plasma_lab | Browse

Requirement

!(F<=100 G<=1 (air_fuel_ratio = 0))

Experimentation

Algorithm: Montecarlo

10

Go !

Results

Initialization
Algorithm started
!(F<=100 G<=1 (air_fuel_ratio = 0)) [# Simulations: 10, # Positive Simulation: 10, Result: 1.0]
Experiment completed.

PLASMA2Simulink | PLASMA Lab v1.3.7-SNAPSHOT

SMC Analysis

Compute the probability that fuel distribution is stopped for at least 1 t.u. :

$$\Phi = F_{\leq 100}(G_{\leq 0.999}\mathsf{Fuel} = 0)$$

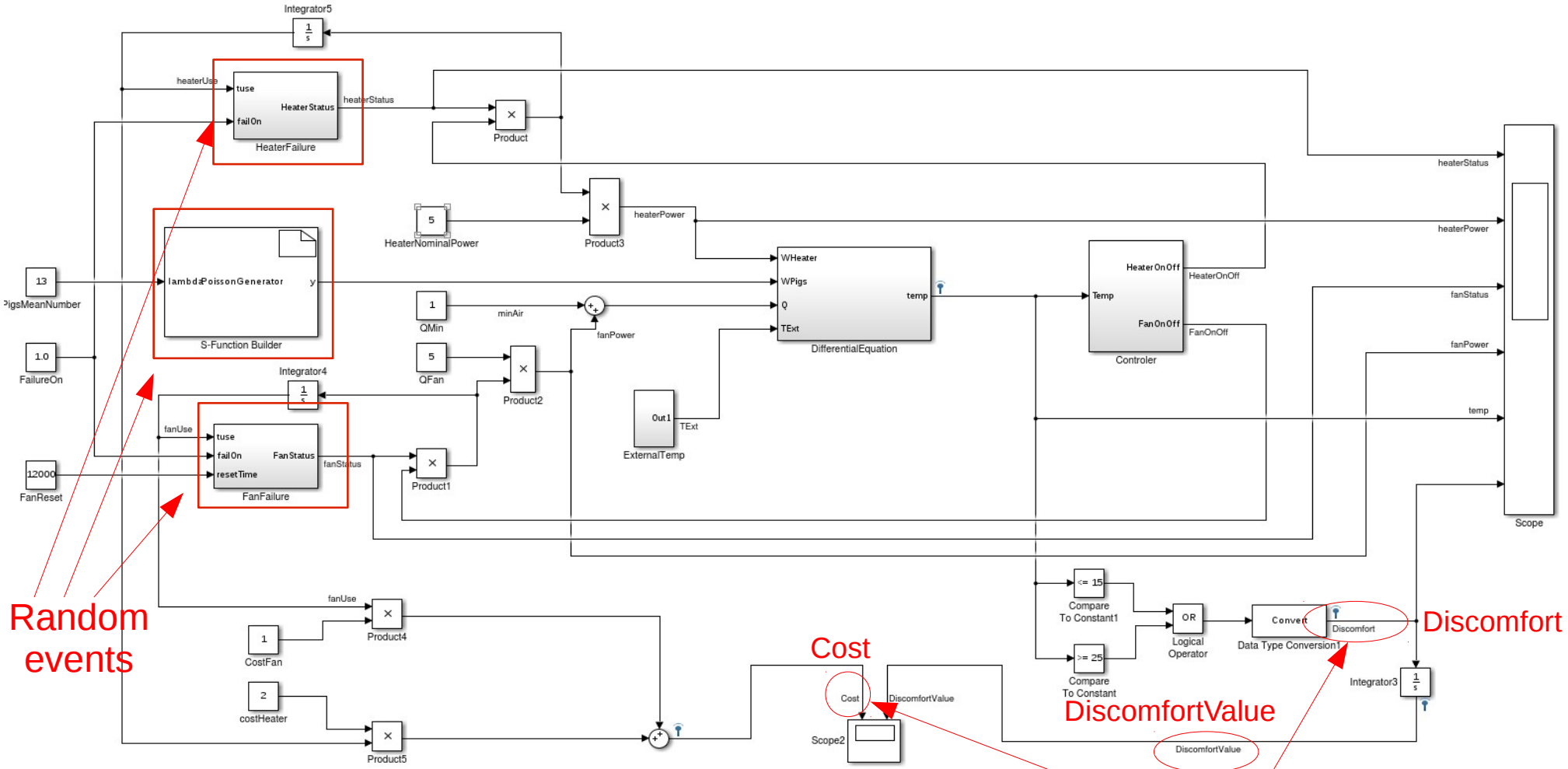
Sensor fault rates (Speed, EGO, MAP)	Failure probability
(3, 7, 8)	0.604
(10, 8, 9)	0.252
(20, 10, 20)	0.07
(30, 30, 30)	0.015

A Pig Shed Case Study

Temperature controller of a pig shed

- Internal temperature is subjected to random variations.
  (number of pigs, external temperature)
- Regulate the temperature by activating fan and heater.
- Fan and heater are subjected to random failures.
- **Objectives:** check and optimize the controller:
  - Internal temperature must remain comfortable.
  - Minimize heating and cooling costs.

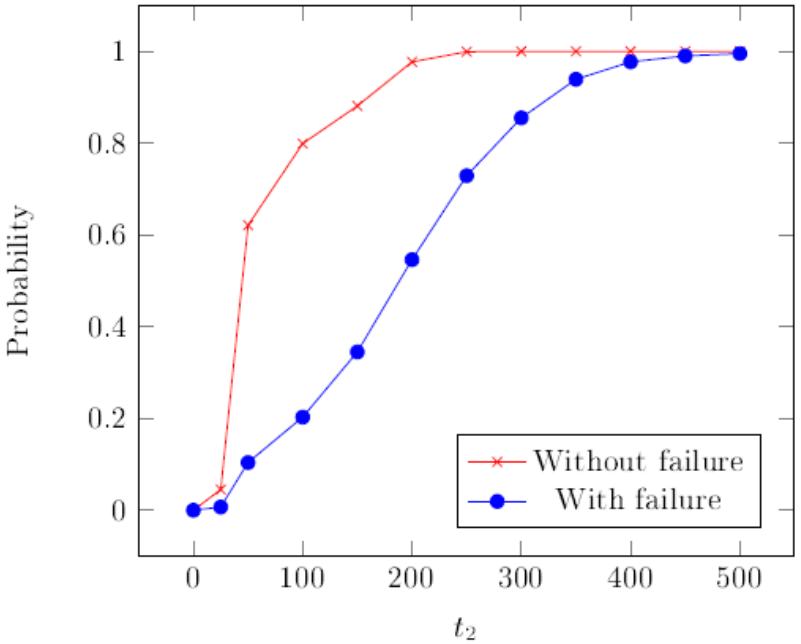
A Pig Shed Case Study



A Pig Shed Case Study Quantitative Verification

Monte-Carlo analysis with
- t1 = 12000 t.u.
- Precision 0,01
- Confidence 0,01

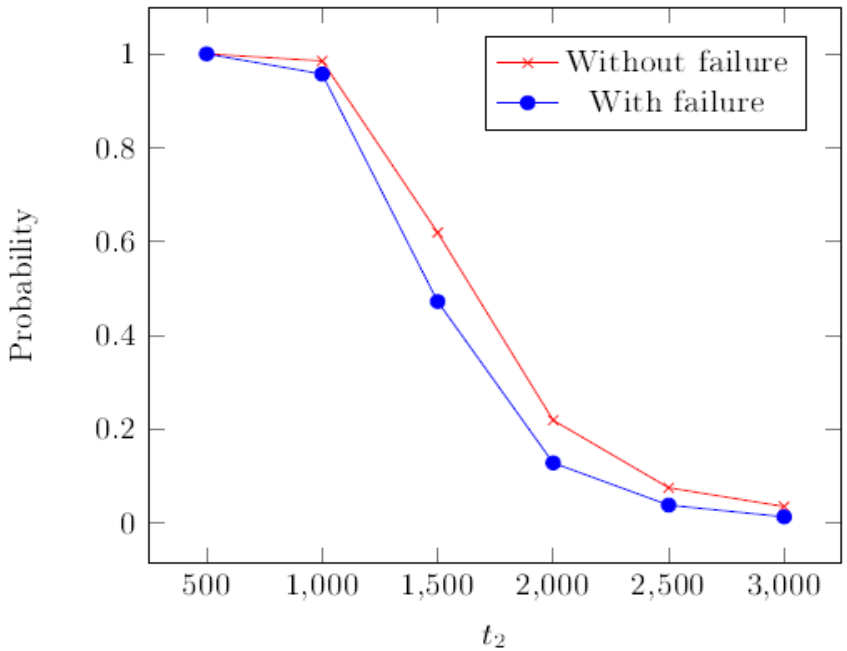
$$\Phi_1 = G_{\leq t_1} F_{\leq t_2} \neg \text{Discomfort}$$



A Pig Shed Case Study Quantitative Verification

Monte-Carlo analysis with
- t1 = 12000 t.u.
- Precision 0,01
- Confidence 0,01

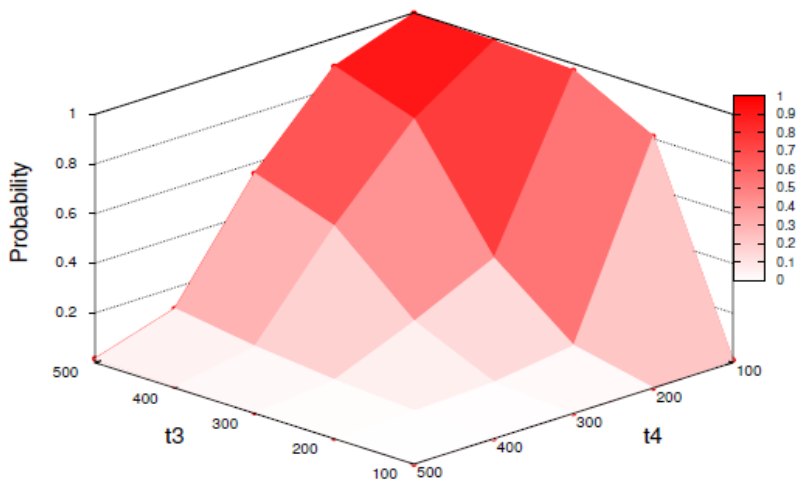
$$\Phi_2 = F_{\leq t_1} G_{\leq t_2} \neg \text{Discomfort}$$



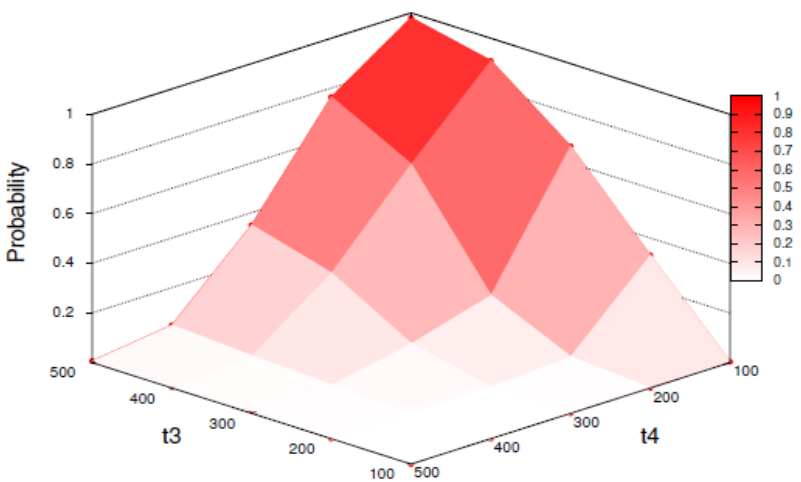
A Pig Shed Case Study Quantitative Verification

Monte-Carlo analysis with
- t1 = 12000 t.u.
- t2 = 25 t.u.
- Precision 0,01
- Confidence 0,01

$$\Phi_3 = G_{\leq t_1}\left(G_{\leq t_2}\text{Discomfort} \Rightarrow F_{\leq t_3}\left(G_{\leq t_4}\neg\text{Discomfort}\right)\right)$$



Without failures



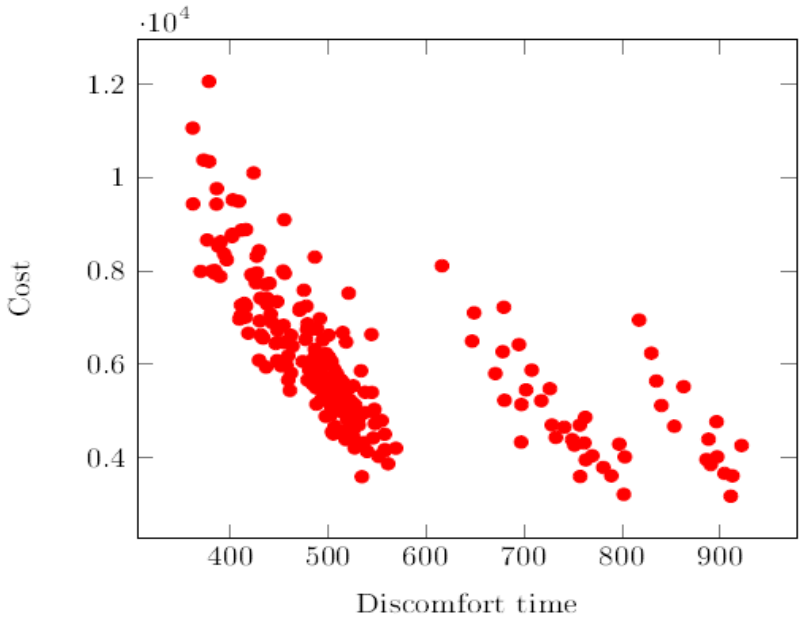
With failures

A Pig Shed Case Study Optimisation

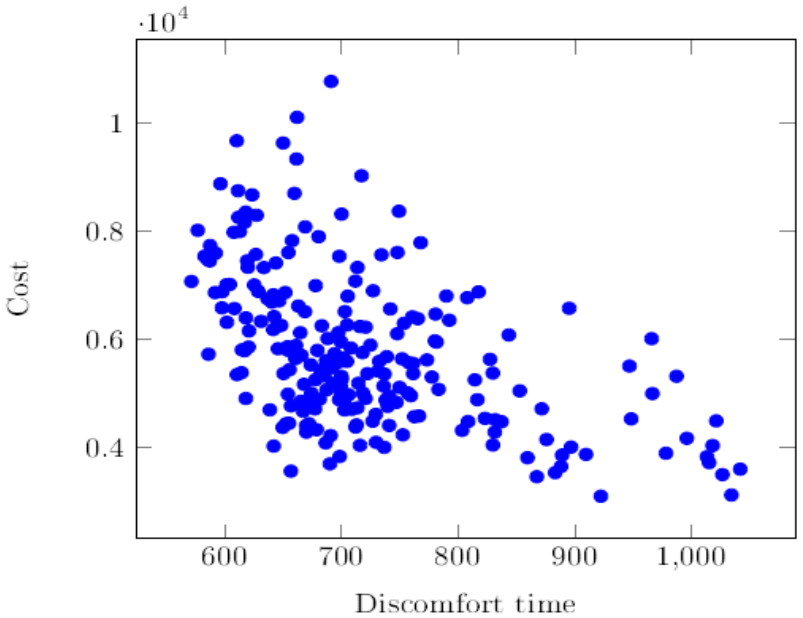
- Use Monte-Carlo to estimate the expected values of two variables:
  - Cost
  - DiscomfortValue
- Compute the results for several values of initial parameters:
  - THeaterOn between [15,20]
  - THeaterOff between [15,20]
  - TFanOn between [20,25]
  - TfanOff between [20,25]
- With additional constraints:
  - TFanOff < TFanOn
  - THeaterOn < THeaterOff
  - THeaterOn < TFanOn

A Pig Shed Case Study Optimisation

Select the best values over the the 225 configurations



Without failures



With failures

Conclusion

- Plasma, a new flexible SMC checker
- New applications
- Integrated inside industry tools