Modeling and Simulation of Attacks on Cyber-Physical Systems

Cinzia Bernardeschi, Andrea Domenici, Maurizio Palmieri

Department of Information Engineering
University of Pisa

Work presented at 3rd International Workshop on FORmal methods for Security Engineering - ForSE 2019

MeTRiD 2019 – April 6, 2019

Outline

1. Overview of the contribution

2. Background
    - Co-Simulation of Cyber-Physical Systems
    - Prototype Verification System (PVS)

3. Modeling Attacks in PVS
    - Formal modeling of attacks
    - Modification to existing model

4. Co-simulation scenario
    - Line Follower Robot

5. Conclusions

Contribution

UNIVERSITÀ DI PISA

The contribution consists in a case study that shows how a co-simulation environment can be used for modelling and simulate the effects of attacks in CPS

preliminary work:

manual control

attack to sensors and actuators

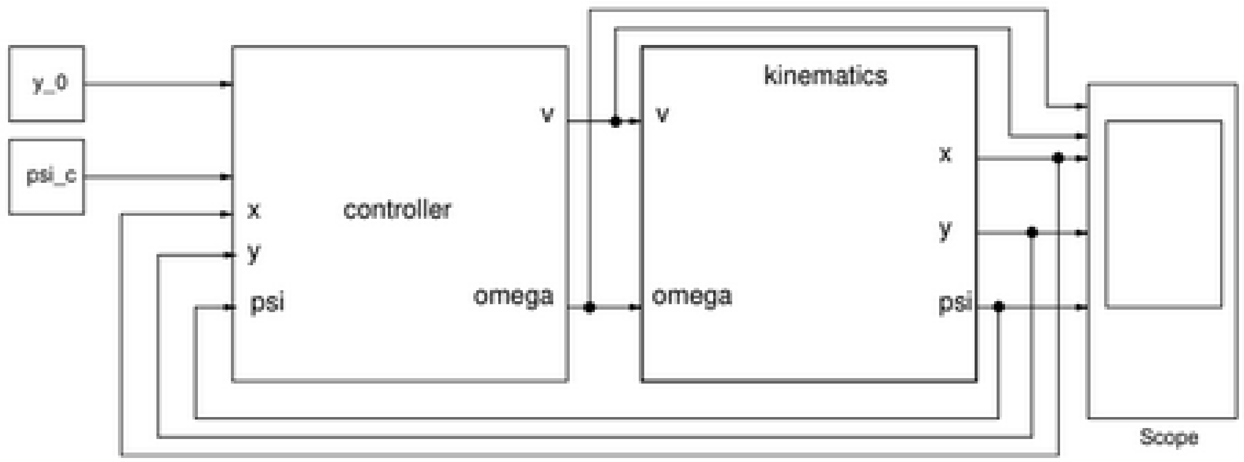
the co-simulation environment allows

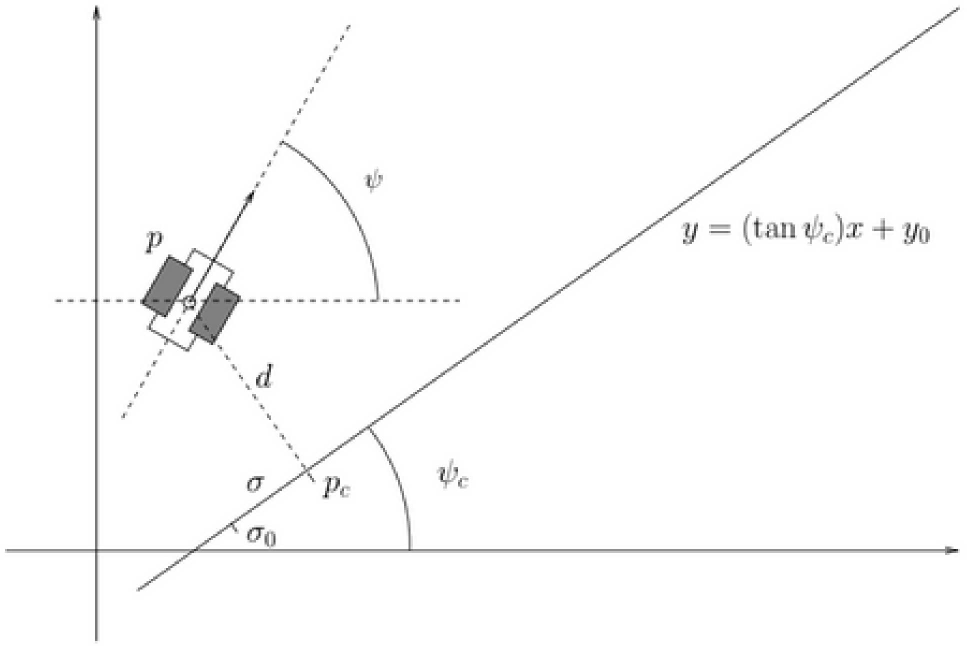
- the analysis of the behavior of the CPS under attack
- formal proofs for CPS

Cyber Physical Systems (CPS)

CPS systems are characterized by the coexistence of continuous and discrete behaviors, and of heterogeneous subsystems.

discrete systems: controller
continuous systems: plant (differential equations)



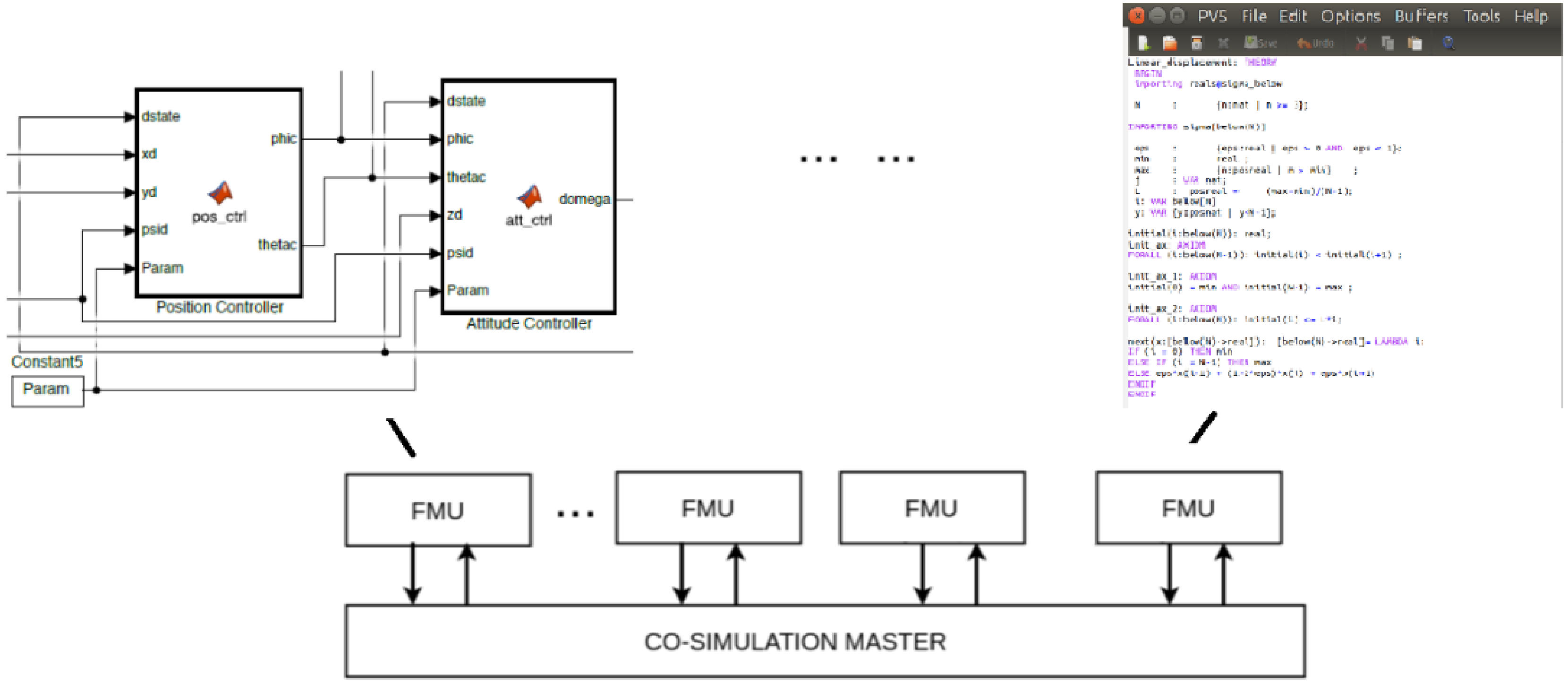


Model-based development of CPSs can benefit from the availability of different modeling languages and tools, each tailored to different components or aspects.

Co-simulation of Cyber-Physical Systems

UNIVERSITÀ DI PISA

FMI: emerging standard for co-simulation of cyber-physical systems.



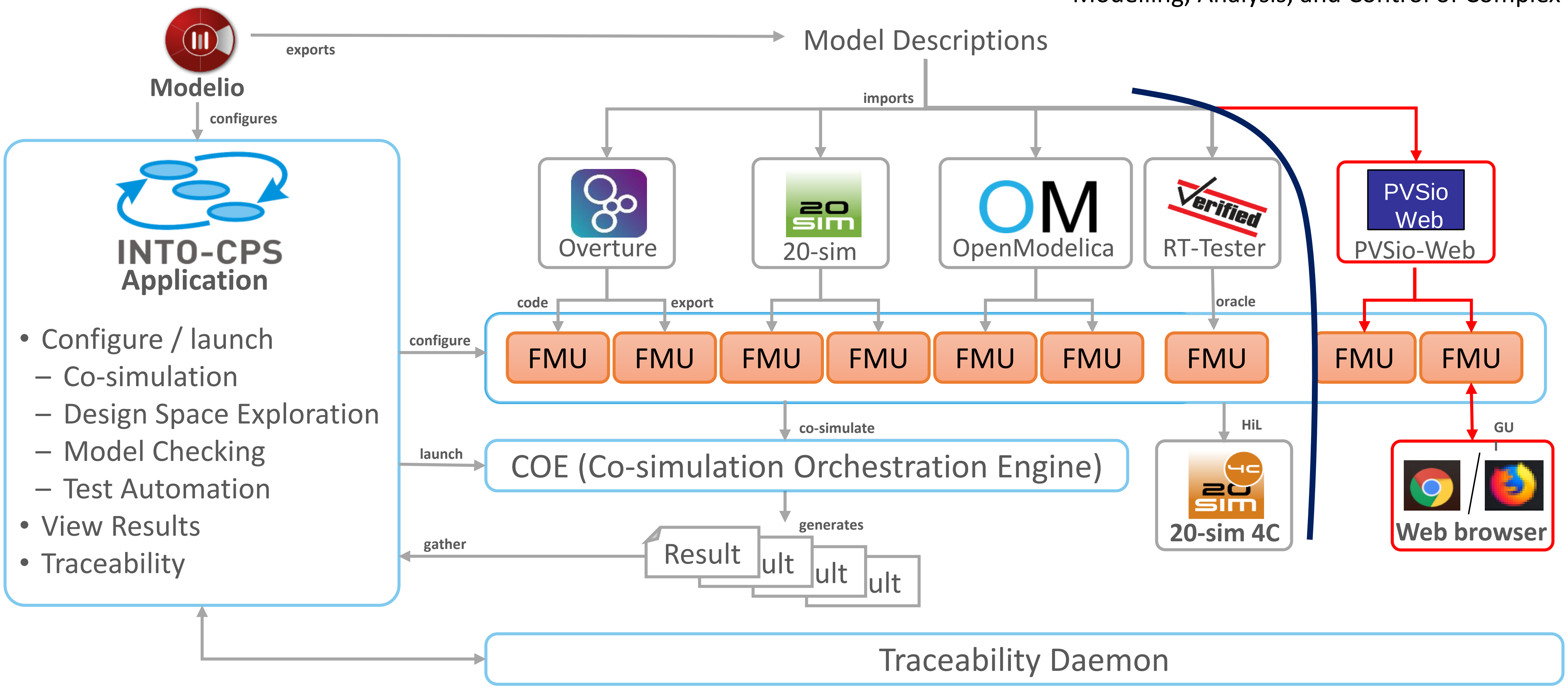
Standard interface for coupling of tools.
Master-Slave approach.
Functional Mock-up Unit (FMU)
Data exchange is restricted to discrete points.

T. Blochwitz, M. Otter, et al. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *Proc. of the 8th Intl. Modelica Conference*, pages 105-114. Linköping University Electronic Press, 2011.

FMI co-simulation with the INTO-CPS tool chain

INTO-CPS: Integrated Tool Chain for Model-based Design of Cyber-Physical Systems. Horizon H2020 project .

Larsen, P. G., Fitzgerald, J., Woodcock, et al. (2016). Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data).



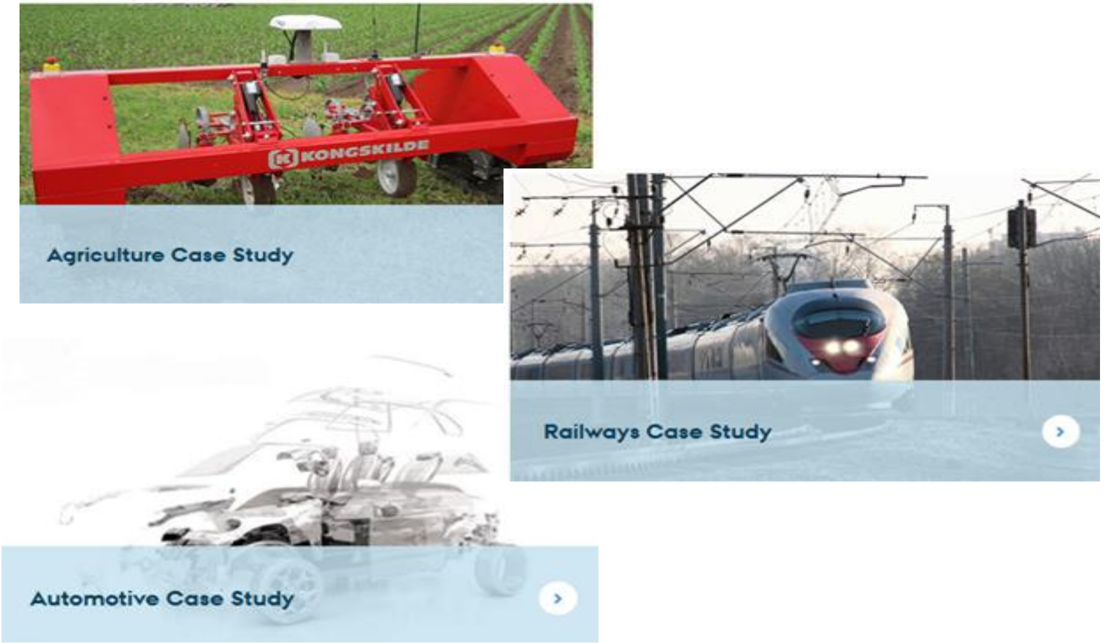
INTO-CPS project case study: LFR

INTO-CPS

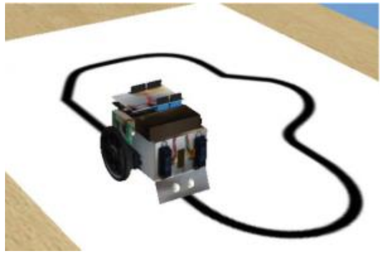
http://into-cps.org/

Integrated Tool Chain for Model-based Design of Cyber-Physical Systems, Horizon 2020, Jan. 2015 – Dec. 2017

Industry follower groups



Case study: Line follower robot

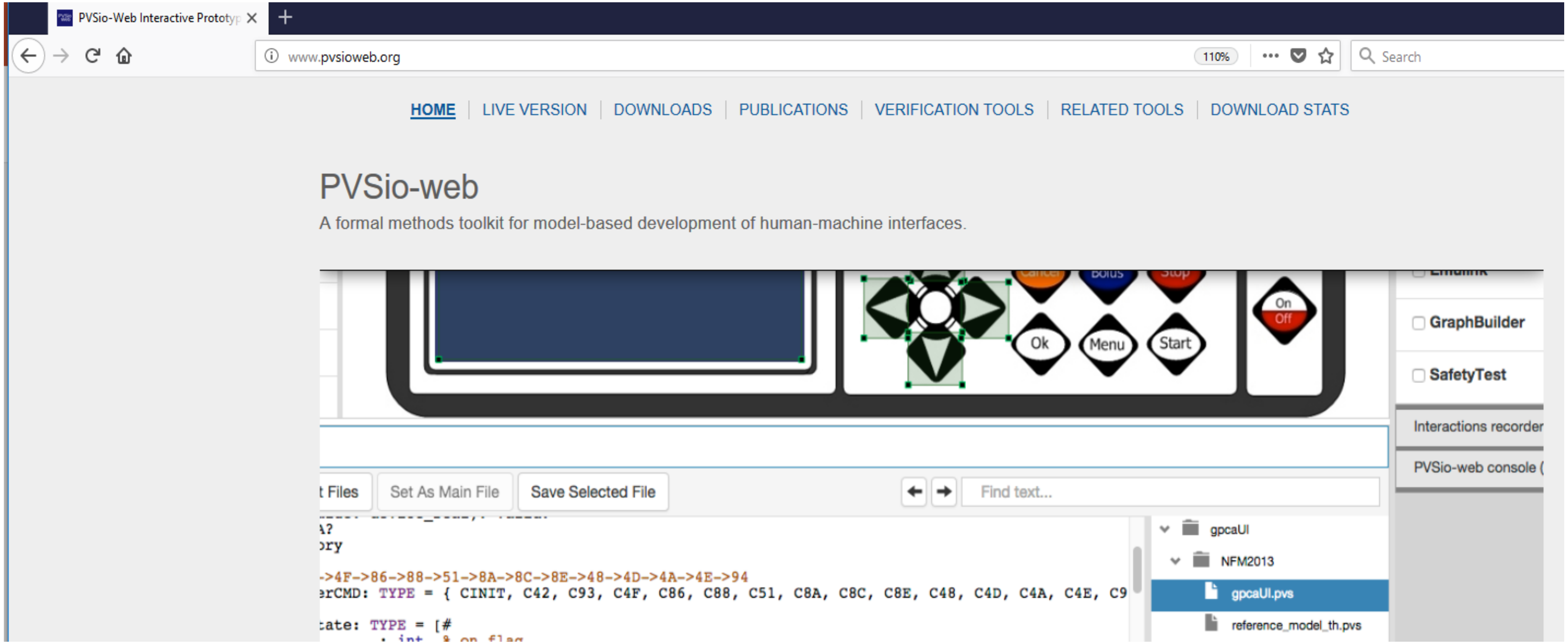


co-simulation of a robot that can follow a line painted on the ground

the line contrasts from the background and the robot uses a number of sensors to detect light and dark areas on the ground.

PVSio-web

A formal methods toolkit for model-based development of human-machine interfaces



Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In: FMIS2013, 5th International Workshop on Formal Methods for Interactive Systems (2013)

PVSio-web

Technologies involved

Ubuntu-Linux OS

- Prototype Verification System
  https://github.com/SRI-CSL/PVS
- PVSio-web
  https://github.com/pvsio-web
- Libwebsocket library
  https://libwebsockets.org/

PVS Theorem prover with an extensive number of inference rules
Providing an assisted formal verification process based on the Sequent Calculus.

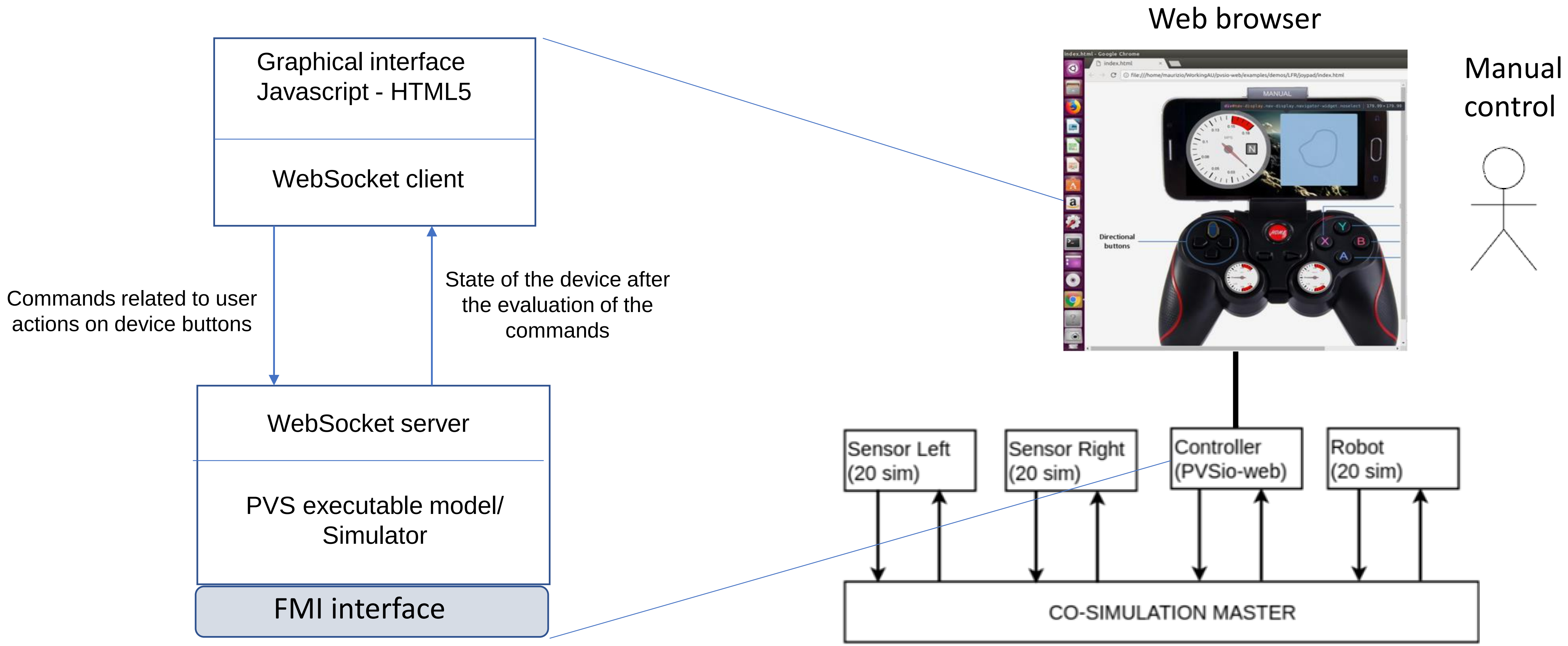
The *Prototype Verification System* (PVS) is an interactive theorem prover developed at SRI International by S. Owre, N. Shankar, J. Rushby and others.

The PVS language builds theories based on classical typed higher-order logic (a pure declarative language)

PVSio (Ground evaluator) For each function a Lisp procedure to compute its value is generated.

PVSio-web is an open-source environment enabling developers and users of interactive devices to assess and validate them with respect to human-machine interaction. Implemented in JavaScript by a software platform composed of several scripts, invoked through a web interface

Semi-autonomous Line Follower Robot



Semi-autonomous Line Follower Robot

LFR: robotic model

original robot model is extended to support two operating modes

Manual operating mode

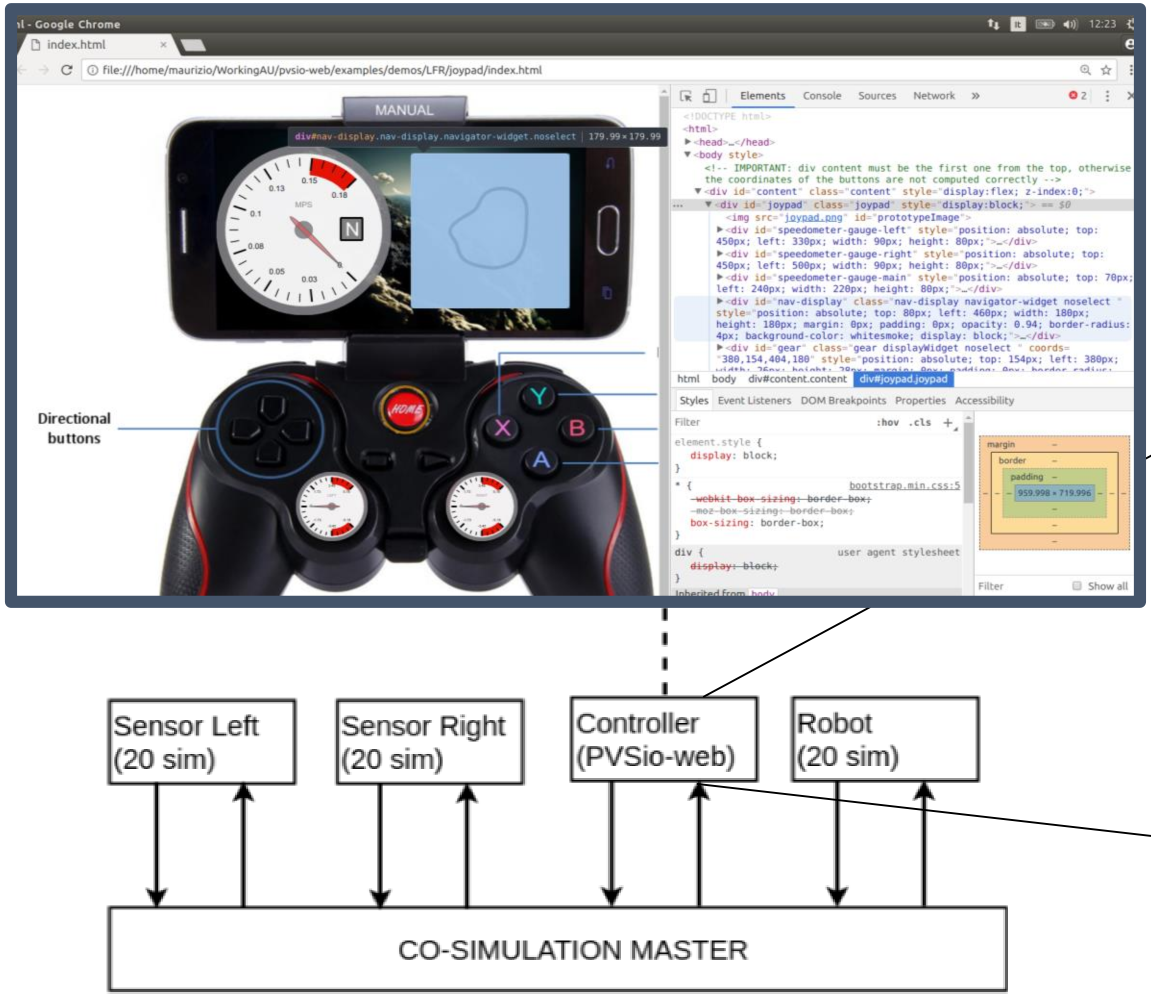
the operator can send commands to the robot using a gamepad controller.

Automatic operating mode

the robot behaves like in the original INTO-CPS example.

the console display reports robot status information, such as robot position, and speed

Semi-autonomous Line Follower Robot




```
State: TYPE =
    [#lightSensors: LightSensors,
     motorSpeed: MotorSpeed,
     gear: Gear,
     cc: CruiseControl#]

init_state: State =
    (#lightSensors := (#left := 0, right := 0#),
      motorSpeed := (#left := IDLE, right := IDLE#),
      gear := DRIVE,
      cc := AUTO#)
```

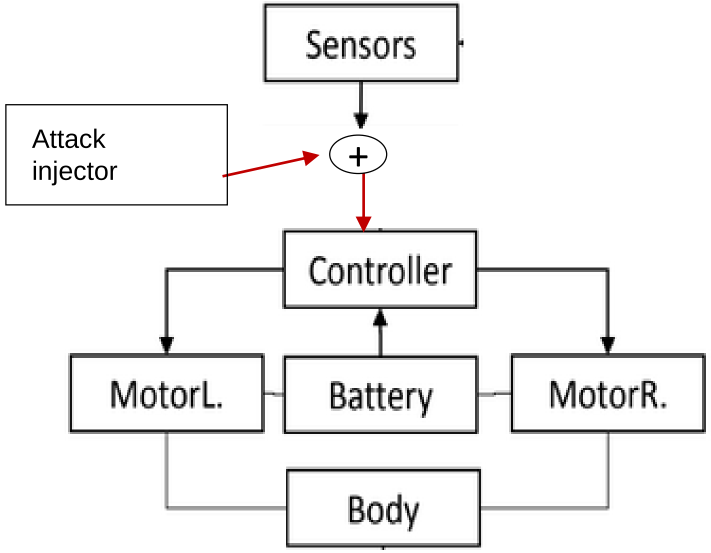
```
BRAKE_STEP: Speed = 1/10

brake(st: State): State =
    st
      WITH [cc := MANUAL,
            motorSpeed
              := (#left := dec_speed(motorSpeed(st)`left, BRAKE_STEP),
                  right
                    := dec_speed(motorSpeed(st)`right,
                                 BRAKE_STEP)#)]
```

Maurizio Palmieri, Cinzia Bernardeschi, Paolo Masci: Co-simulation of Semi-autonomous Systems: The Line Follower Robot Case Study. SEFM Workshops 2017: 423-437

Considerations on the attacks

- Attack to sensors
  - Can be masked (if fault tolerance techniques are applied) attack --- malicious fault
  - Can alter the value of local/hidden variables of the control
  - Can affect logging/tracing mechanism of the system



- Attack to actuators
  - Can have more impact on the behaviour of the system

A combination of them.

Considerations on the attacks

The attacker can acquire control of the robot from the joystick, manually control the robot with buttons, and change its speed or direction.

Considerations on the attacks

Attacks generated internally by the simulation algorithm

Attacks to sensors

- The effect is the corruption of data received from sensors
- The output written at the end is based on corrupted inputs

Attacks to actuators

- The effect is the corruption of data sent to actuators
- The output written at the end ignores the computed one

```
Control loop(){
 readInput();
 SensorsAttack
 computeOutput();
 writeOutput();
}
```

```
Control loop(){
 readInput();
 computeOutput();
 ActuatorsAttack
 writeOutput();
}
```

Implementation: attacks as functions that alter the state of the controller.

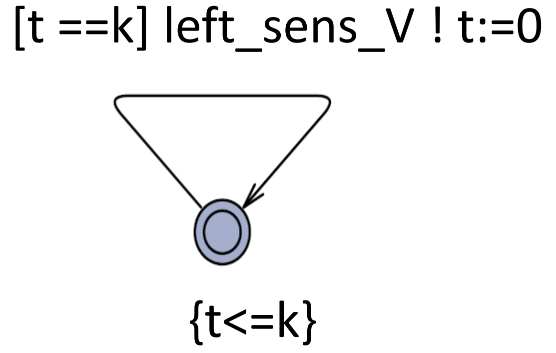
Introducing Attacks in control components

The attack is triggered only if certain condition is met

- Always from the beginning
- Always after a specific timestep
- Always after a random timestep
- During random intervals
- During specific intervals
- Only on specific timesteps
- Only on random timesteps
- Only once

Different models of the attack injector

Injector specified as a timed automaton (TA)



Rajeev Alur and David L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2), 1994.

Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on UPPAAL 4.0, 2006.

Formal modeling of attacks on CPS

$$A = \langle Var_A, Clk_A \cup \{stepCounter\}, Com_A \rangle$$

- Var_A is the set of variables altered by the attack A
- Two types of clocks are used:
  - Clk_A represents the set of attacker clocks
  - $stepCounter$ represents the global time
- Com_A is a set of guarded statements in the form:
  - $Condition \rightarrow x_1 := v_1; \ldots; x_n := v_n$
- $Condition$ is a guard on clocks
- $x \in Var_A \cup Clk_A$

PVS skeleton of attack function

```
attack(st: ext_State): ext_State =
  IF condition
    THEN st
         WITH [
                x1 := v1,
         ...,
                xn := vn
                ]
         ELSE st
   ENDIF
```

ext_state :
    the state of the controller + Clk_A + $stepCounter$

The IF-THEN-ELSE statement represents Com_A

$\mathrm{x1}, \dots, \mathrm{xn} \in Var_A \cup Clk_A$

If no condition is met the state of the system is not modified

Sensor/Actuator attacks in PVS

```
model_under_attack(st: ext_State) : ext_State =
      LET st1 = Sensor_attack(st),
       IN tick(st1)
```

Attack on sensors

```
Control loop(){
 readInput();
 model_under_attack();
 writeOutput();
}
```

tick(st: ext_State): ext_State
computes the output and also increments *stepcounter*.

```
model_under_attack(st: ext_State) : ext_State =
      LET st1 = tick(st),
       IN Actuator_attack(st1)
```

Attack on actuators

Example: Line Following Robot (LFR)

UNIVERSITÀ DI PISA

```
State: TYPE =  [#
 lightSensors: LightSensors,
 motorSpeed: MotorSpeed,
 time: real,
 cc: CruiseControl #]
```

```
ext_State: TYPE =  [#
 state: State,
 stepcounter: int,
 clk1: int,
 clk2: int,
 clk3: int  #]
```

```
tick(st: State): State =
 IF cc(st) = AUTO
   THEN st WITH [
       motorSpeed :=  (#
              left := update_left_speed(st),
       right := update_right_speed(st)
              #),
     time := time(st)+0.01   ]
    ELSE st WITH [time := time(st)+0.01]    ENDIF
```

```
tick(st:ext_State): ext_State = st WITH[
       state := tick(st`state),
       stepcounter := stepcounter +1]
```

Attack to Wheel Actuator to LFR

The attack sporadically switches off each wheel for the duration of a single step

```
Actuator_attack(st: ext_State):  ext_State =
  IF clk2(st) = clk3(st) THEN
          st WITH [
                    state`motorSpeed := (#
                       left := 0,
                       right := 0 #),
             clk2 := NRANDOM(20),
             clk3 := 0 ]
   ELSE  st WITH [ clk3 := clk3 + 1 ]

 ENDIF
```

$$Var_{Actuator\_attack} = \{motorSpeed\}$$
$$Clk_{Actuator\_attack} = \{clk2, clk3\}$$
$$Com_{Actuator\_attack} = \{Actuator\_attack\}$$

Clk2 is initialized with a random value

Clk3 is initialized with zero

Attack to LightSensors of LFR

UNIVERSITÀ DI PISA

The attack forces the value of the left sensor starting from a random timestep

```
Sensor_attack(st: ext_State): ext_State =
    IF stepCounter(st) >= clk1(st)
       THEN st  WITH [
         state`lightSensors`left := 140]
    ELSE st
    ENDIF
```

$$Var_{Sensor\_attack} = \{lightSensor'left\}$$
$$Clk_{Sensor\_attack} = \{clk1\}$$
$$Com_{Sensor\_attack} = \{Sensor\_attack\}$$

Clk1 is initialized with a random value

Value below the threshold of 150 means “black”

Co-simulation: no attack

UNIVERSITÀ DI PISA

- Using the INTO-CPS Application, we have run different co-simulation scenarios
  - fixed stepsize of 0.01 seconds
  - duration of 20 seconds.

- The first scenario is the LFR without any attack

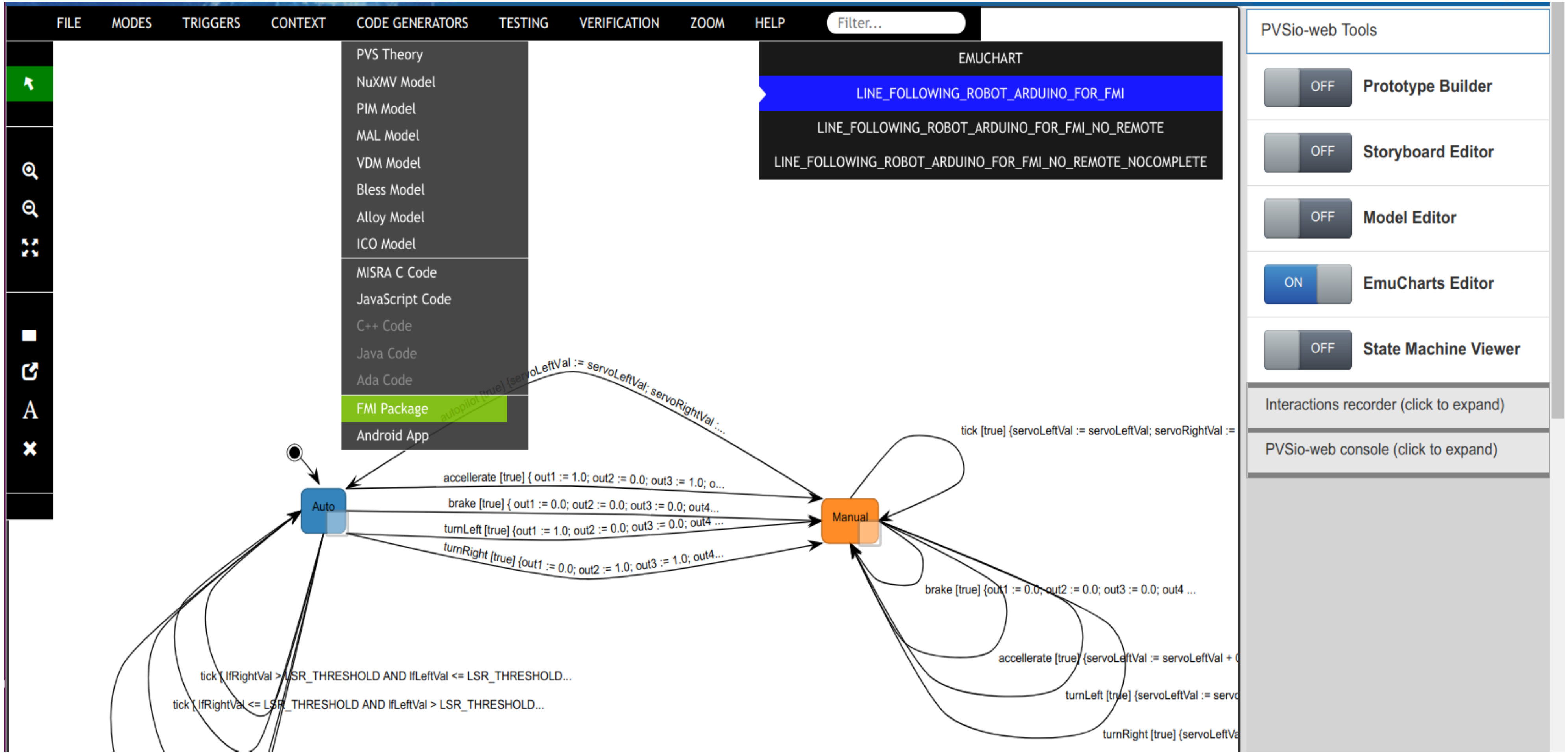
Co-simulation: attack on actuators

- The second scenario is the one where we applied the attack on the actuator
  - The robot behaves in a consistent way, but it has encountered a reduction of performance

Co-simulation: attack on sensors

- The third scenario is the one where we applied the attack on the sensor
  - The robot is forever stuck in a circle

Automatic FMU generation from a graphic PVSio-web editor



Formal verification

Formal verification is an important complement to co-simulation

For example, we can prove that the LFR system satisfies this property:

It is always the case that if the value of the right sensor is set to white, the power of the left motor is lower than or equal to the power of the right motor.

```
never_turn_right: THEOREM
   kth_step(NRANDOM(500)+1)`lightSensors`right>150 IMPLIES
         FORALL    ((K:above(NRANDOM(500)+1)) :
                              motorSpeed(kth_step(K))`left <= -motorSpeed(kth_step(K))`right
```

Summary

- We allowed the study of attacks on CPS with formal methods and co-simulation
- We have provided an easy way to include attacks in PVS model
  - Encouraging the use of formal methods in model-based design
- The extended models can be simulated together with models from different tools
  - Enhancing the validation of the model
  - Reducing the effort required to formally specify a Cyber-Physical System
  - Exploiting the advantages offered by well known tool-chains

Conclusions

UNIVERSITÀ DI PISA

- To assess safety properties of the system under attack
  - By exploiting the PVS theorem prover
- To allow end-user training in recognizing attack scenarios
  - By implementing interactive attacks with PVSio-web
- To find critical elements of the system and provide mitigation mechanism
  - By exploiting the results of the previous points