ScicosLab and CANopen

Ing. Roberto Bucher

Scuola Universitaria Professionale della Svizzera Italiana (SUPSI)

Firenze - October 21. 2010

Objectives

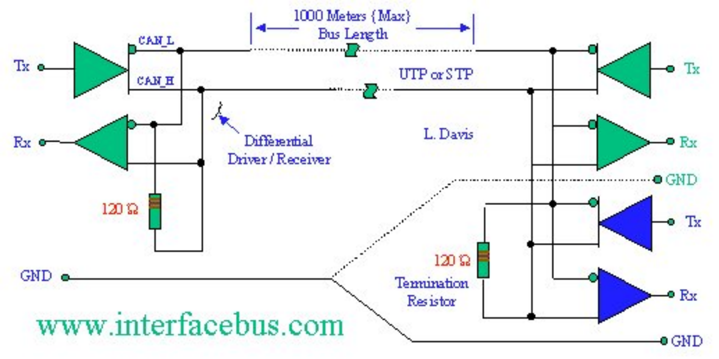
- See how CANopen devices can be integrated in a Scicos block diagram
- Learn how to implement new CANopen devices
- Introduce the CAN200 interface dongle

Outline

Controller Area Network

- Controller area network (CAN or CAN-bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer.
- It was designed specifically for automotive applications but is now also used in other areas.
- Development of the CAN-bus started originally in 1983 at Robert Bosch GmbH.

Interface


1000 Meters {Max}
Bus Length
CAN_L
CAN_H
Tx
Rx
UTP or STP
Differential
Driver / Receiver
L. Davis
120 Ω
GND
Termination
Resistor
www.interfacebus.com

CANopen

- CANopen is a communication protocol and device profile specification for embedded systems used in automation.
- In terms of the OSI model, CANopen implements the layers above and including the network layer.
- The lower level protocol implementing the data link and physical layers is usually Controller Area Network (CAN).

Device model

Every CANopen device must implement:

- A **Communication unit** (protocol for messaging)
- A **state machine** (Initialization, Pre-operational, Operational, Stopped)
- An **object dictionary** used for configuration and non-realtime communication with the device.
- The **application part**

Communication objects

	Function code	Node ID	RTR	Data length	Data
Length	4 bits	7 bits	1 bit	4 bits	0-8 bytes

SUPSI

Network management (NMT) protocol

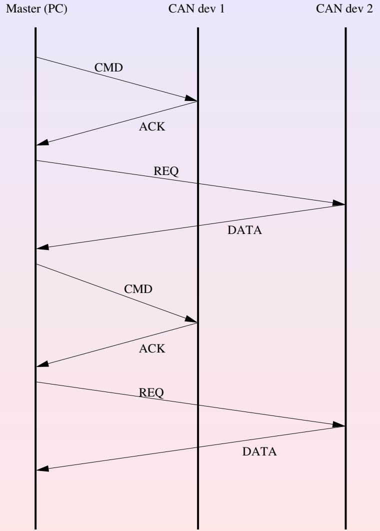
The NMT protocols are used to issue state machine change commands (ie. to start and stop the devices), detect remote device bootups and error conditions.

Service Data Object (SDO) protocol

The SDO protocol is used to set and read values from the object directory of a remote device.

3 bits	1 bit	2 bits	1 bit	1 bit	2 bytes	1 byte	4 bytes
ccs=1	reserved(=0)	n	e	s	index	subindex	data

SDO

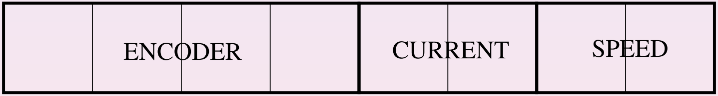


Process Data Object (PDO) protocol

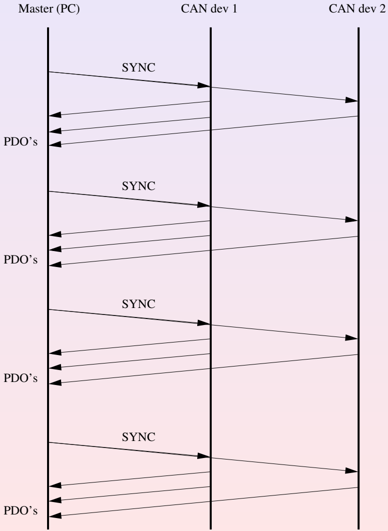
Process Data Object protocol is used to process real time data among various nodes.

- Synchronous
- On change
- TX and RX PDO's
- More values in the same message

PDO



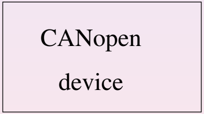
PDO


Master (PC)
CAN dev 1
CAN dev 2
SYNC
PDO's
SYNC
PDO's
SYNC
PDO's
SYNC
PDO's

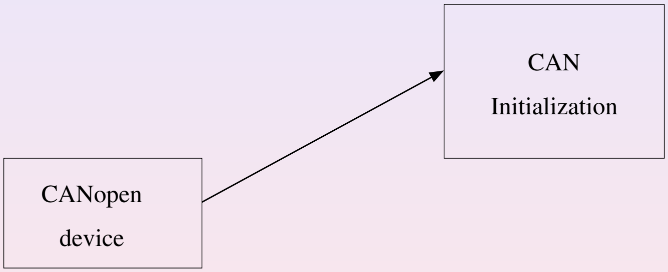
Others

- Synchronization Object (SYNC) protocol
- Time Stamp Object (TIME) protocol
- Emergency Object (EMCY) protocol

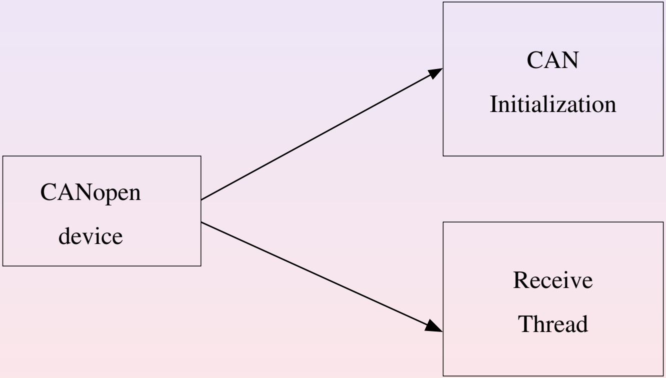
Integration in Scicos



Integration in Scicos



Integration in Scicos



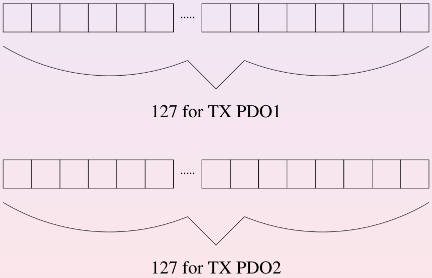
canopen.c

- Initializes the RTDM driver
- Contains a function to register the TX PDO
- Starts a high priority thread for receiving the CAN messages
- Handles TX PDO1 and TX PDO2, but can be easy extended to handle more PDO's

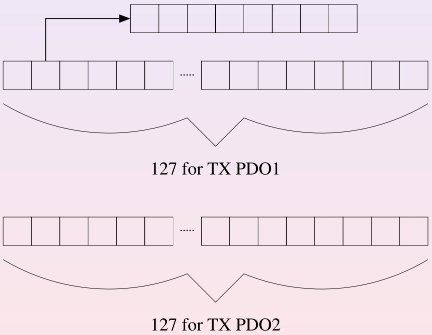
canopen.c

- The canopen.c file

PDO handling



PDO handling



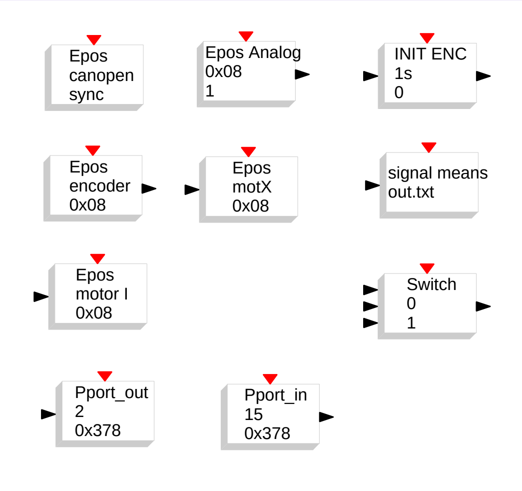
The “canopen_sync.c” file

- Simply sends a SYNC (0x80) message to all CANopen devices
- All the devices handle the SYNC message as follow:
  - Devices with configured TX PDO’s send the requested values to the master
  - Devices with configured RX PDO’s set the last received value

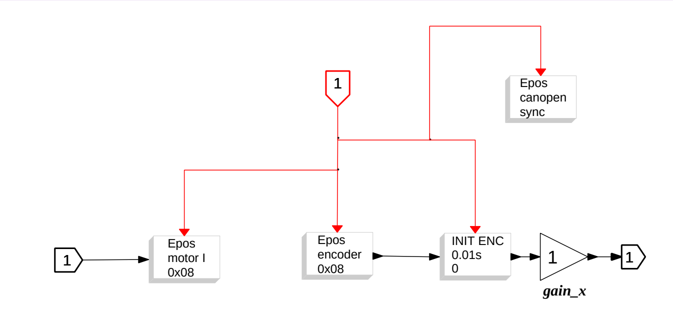
Reading from the CAN Bus

- A real-time thread scheduled with SCHED_FIFO and priority "1" is responsible to get the messages from the CAN bus.
- The receiving procedure runs using the interrupt of the CAN bus RTDM driver.
- The registered PDO's are saved into the two PDO matrices.
- 8 procedures allow to extract the values from the PDO messages (1,2,4 bytes, signed, unsigned)

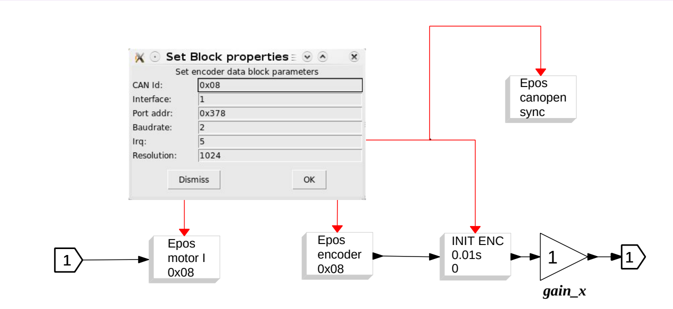
Additional palettes



The interface



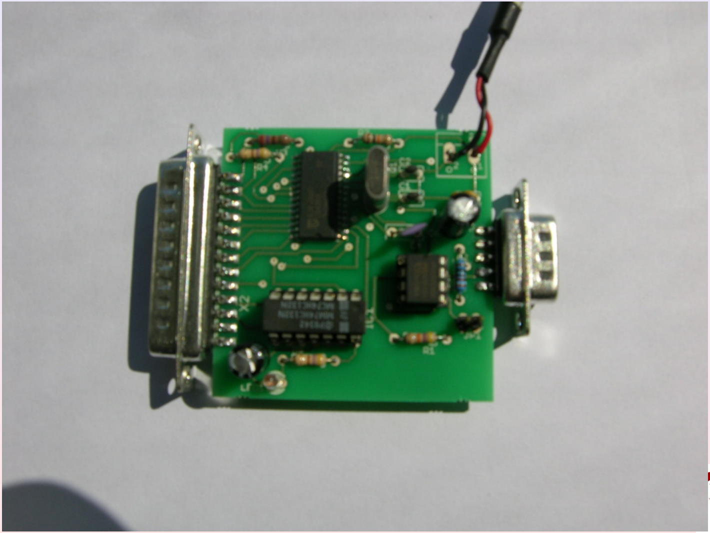
The interface



The implementation

- The Implementation file

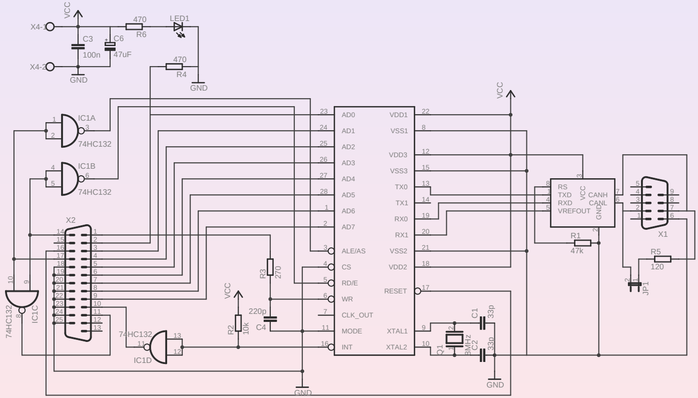
The CAN200 dongle



Details

- based on the SJA1000 chip
- Little modifications on the original CAN200 board
  - 16 Mhz quarz
  - Schmitt trigger chip instead of a nand
- Quite simple
- PCB and scheme are provided

The CAN200 dongle - Scheme

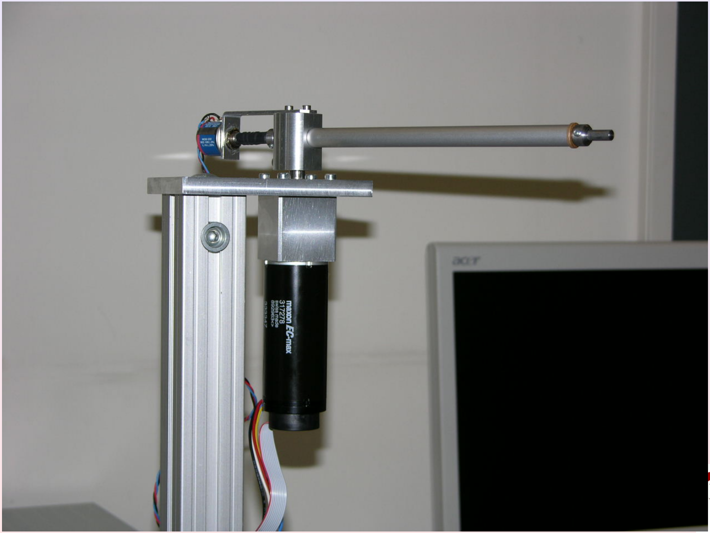


6/23/2009 15:45:44 /home/bucher/eagle/can200/can200.sch (Sheet: 1/1)

The Driver

- RTDM driver
- Modified peaks dongle driver
- Few lines changed

The Servo motor



The Servo motor

Features

- Motor Output
- Hall sensor+Incremental encoder
- 2 Analog inputs $-10V \div +10V$
- 1 Analog output $0V \div 10V$
- 10 Digital inputs
- 5 Digital outputs

Motor equations

Differential equation

$$J_m \cdot \ddot{\varphi}_m = -D_m \cdot \dot{\varphi}_m + K_t \cdot I$$

In Laplace

$$\Phi(s) = \frac{\frac{K_t}{J_m}}{s^2 + \frac{D_m}{J_m} \cdot s} \cdot I(s)$$

Motor equations

$$\Phi(s) = \frac{\frac{K_t}{J_m}}{s^2 + \frac{D_m}{J_m} \cdot s} \cdot I(s)$$

can be rewritten as

$$\Phi(s) = \frac{K}{s^2 + \alpha \cdot s} \cdot I(s)$$

where

$$K = \frac{K_t}{J_m}, \alpha = \frac{D_m}{J_m}$$

Motor equations

With $I(s)$ as step signal we obtain:

$$\Phi(s) = \frac{K}{s^2 + \alpha \cdot s} \cdot \frac{I_0}{s}$$

that in time domain becomes

$$\varphi(t) = -\frac{K \cdot I_0}{\alpha^2} + \frac{K \cdot I_0}{\alpha} \cdot t + \frac{K \cdot I_0}{\alpha^2} \cdot e^{\alpha \cdot t}$$

Example

Controller

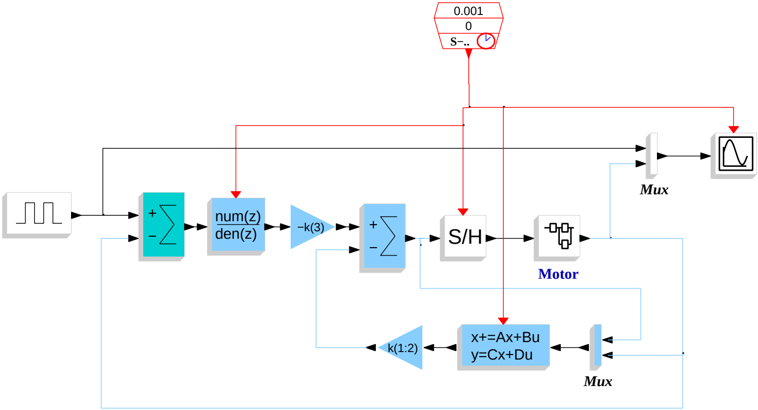
ScicosLab

State feedback controller

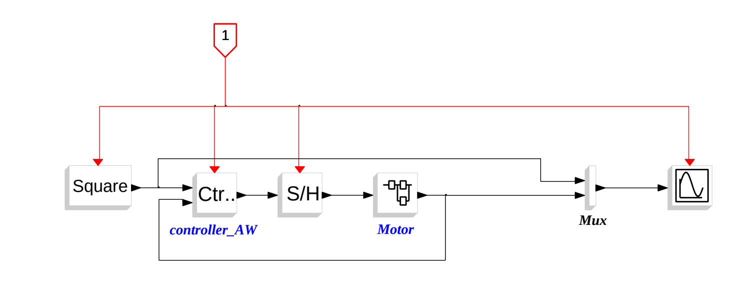
The controller has been implemented as

- State feedback controller
- Reduced order observer
- Integral part for eliminating the steady state error
- Feedback gains+Observer+discrete integrator realized in a unique state space block
- The controller has been then divided into a direct state space block and a feedback state space block for implementing a anti-windup mechanism

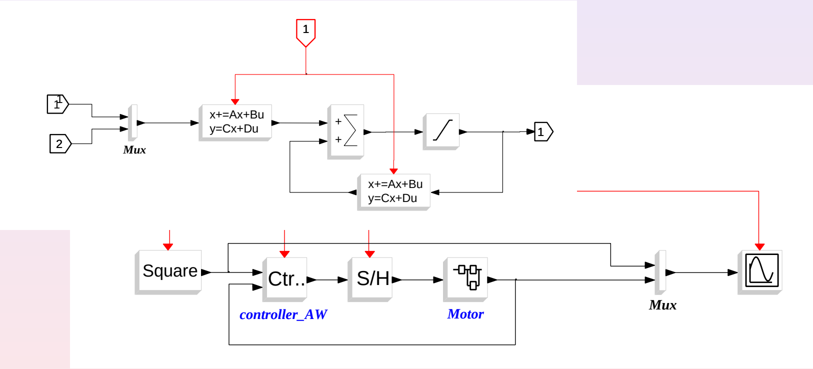
Controller and Observer



Simulation

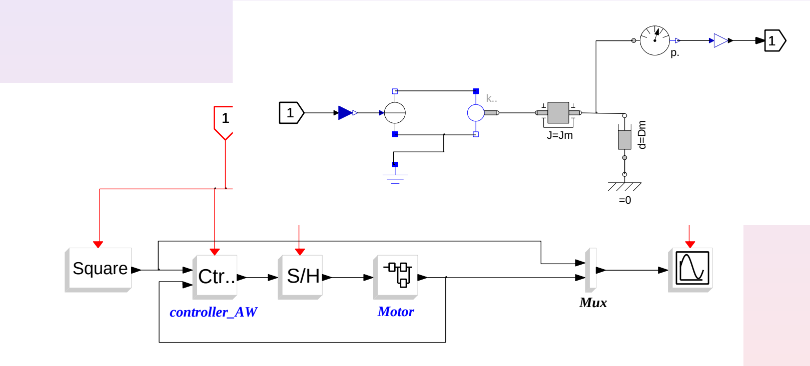


Simulation

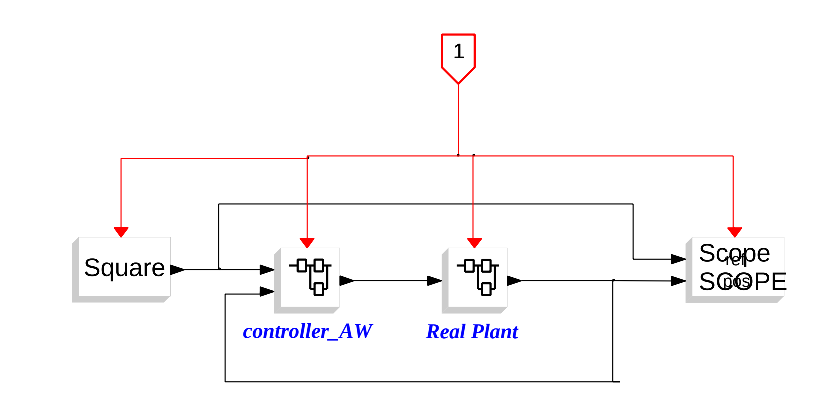

1
1
2
Mux
x+=Ax+Bu
y=Cx+Du
+
+
1
x+=Ax+Bu
y=Cx+Du
Square
Ctr..
S/H
controller_AW
Motor
Mux


RTAI
SUPSI

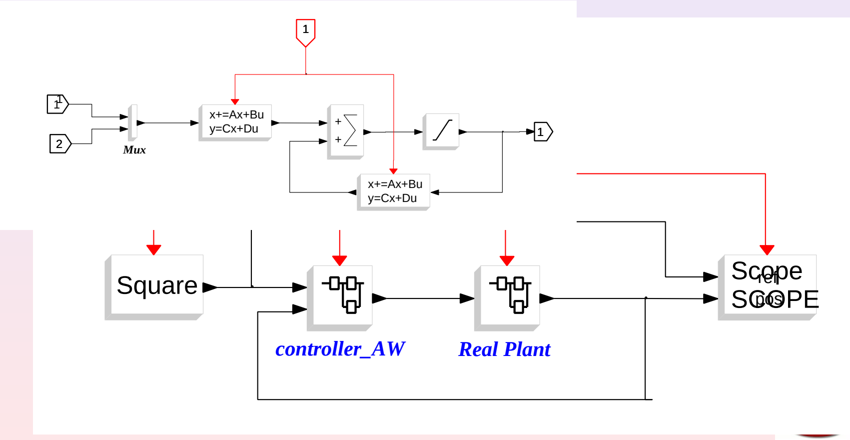
Simulation



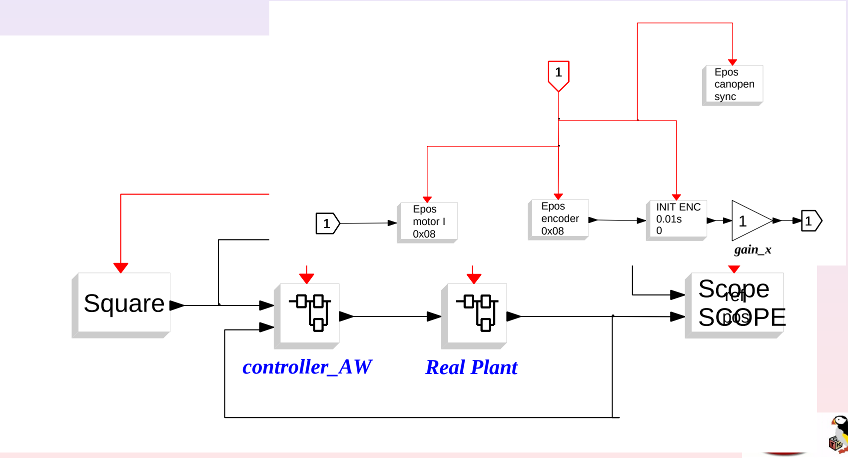
Code generation



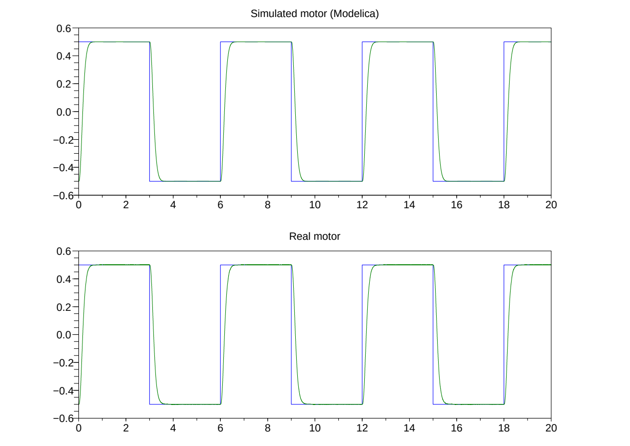
Code generation



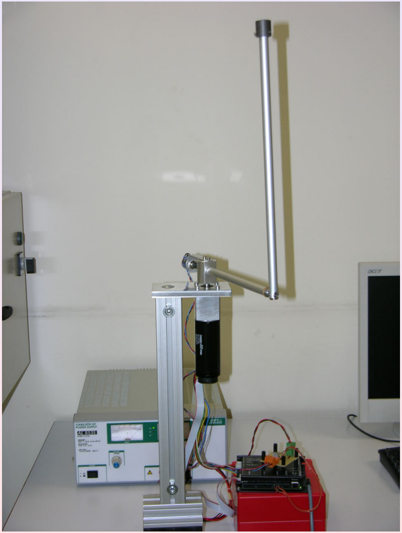
Code generation



Compare results



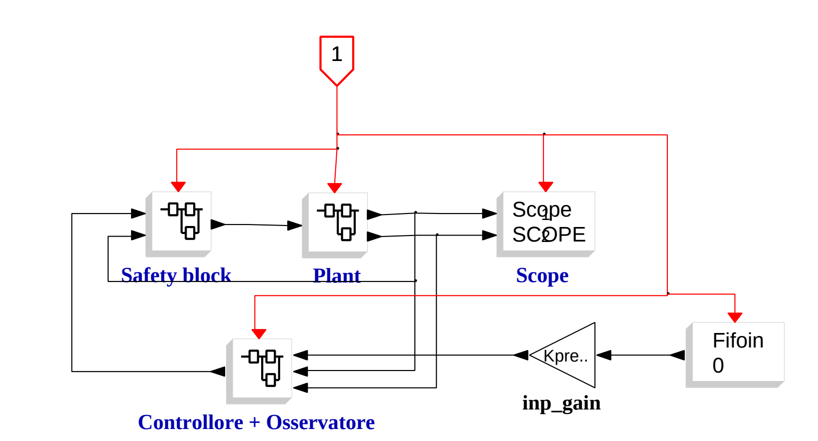
The Furuta pendulum



Goals

- Cheap system (compared to commercial systems)
- Reusable HW
- Possibility to be reproduced
- COTS hardware

Code generation



Code generation