Reaction Wheel Application

Written by GT-DISC

Institut Supérieur de l'Aéronautique et de l'Espace SUPAERO

Table of Contents

Activity Summary

Real-time software :

- Methodology
- Specification
- Design
- Programming
- Tests and temporal analysis

Tool : Xenomai RTOS

Control theory :

- Modelling
- Identification
- Control laws synthesis
- Comparative study between model and experiment.

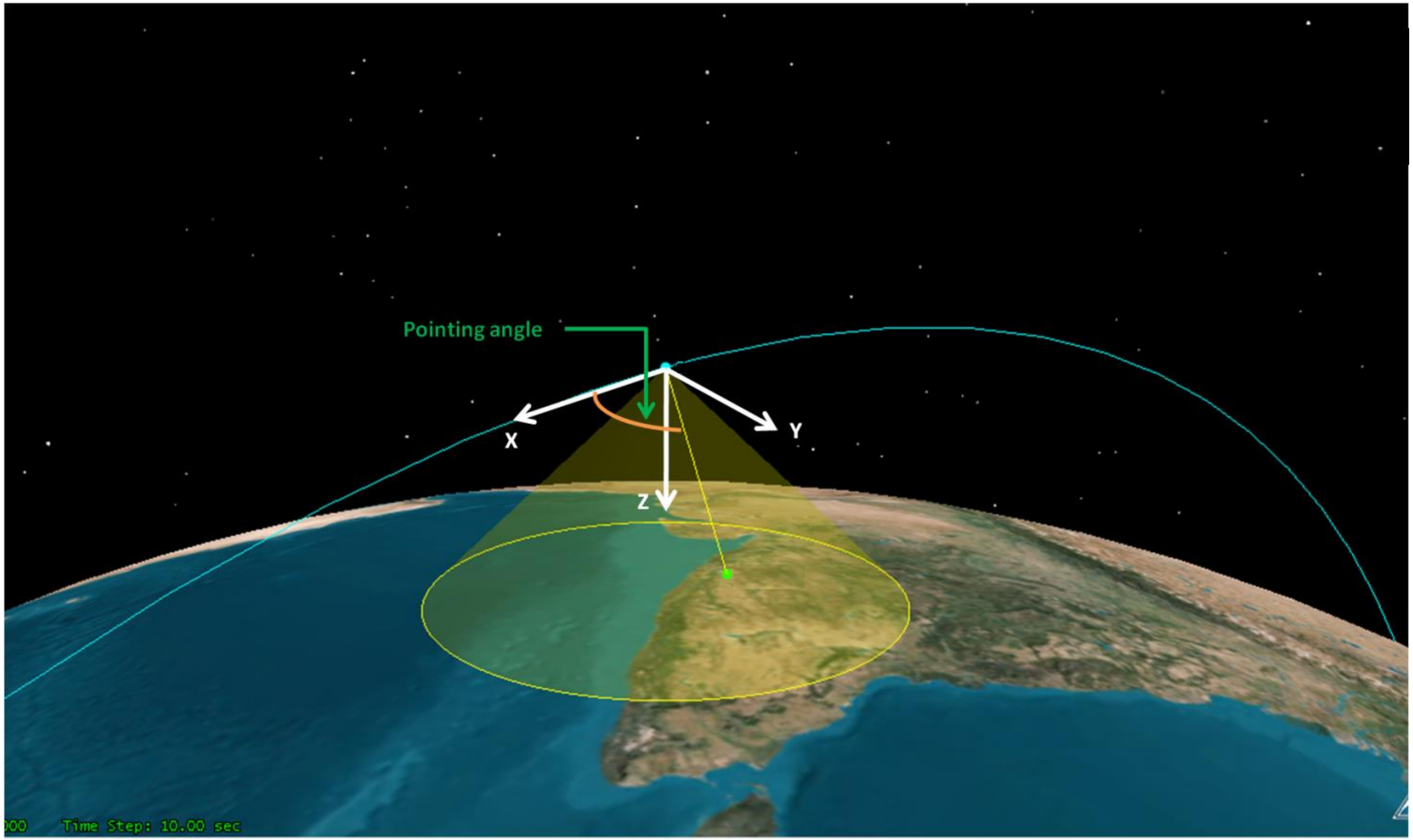
Tool : Matlab

Law implementation in the real time software

2. Attitude control overview

- *Definition*
- *Objectives of attitude control*
- *Attitude control actuators*
- *Example for 3 axis attitude control*
- *Attitude control subsystem*

Definition



Every spacecraft needs to carry an Attitude Control System in order to manage the orientation and position during its entire life.

It is a critical system, hence its design and implementation is one of the more importance of a space project.

Objectives of Attitude Control in Space

- Attitude modification :
  - During the life-cycle of the spacecraft, this will adopt different profiles depending on the mission commands, hence re-orienting for each purpose. (antenna directioning, solar pointing, etc)
- Attitude maintaining with a precision adapted to the mission :
  - In certain missions, the spacecraft needs high precision attitude in order to correctly perfomed its scientific endevour.
- Cancelling perturbation torques (detumbling, gravity-gradient, atmospheric drag, earth magnetic field, solar pressure, etc).

Attitude Control Actuators

» Passive :

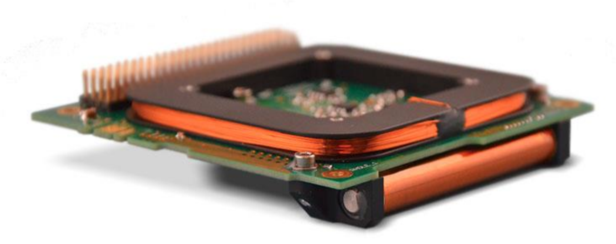
- Gravity-gradient Stabilization
- Embedded Magnet

» Active :

- Magneto torquers
- Reaction Control System (gas thrusters)
- Spin Stabilization
- Controm Moment Gyros
- **Reaction Wheel**

Reaction Control System (RCS)

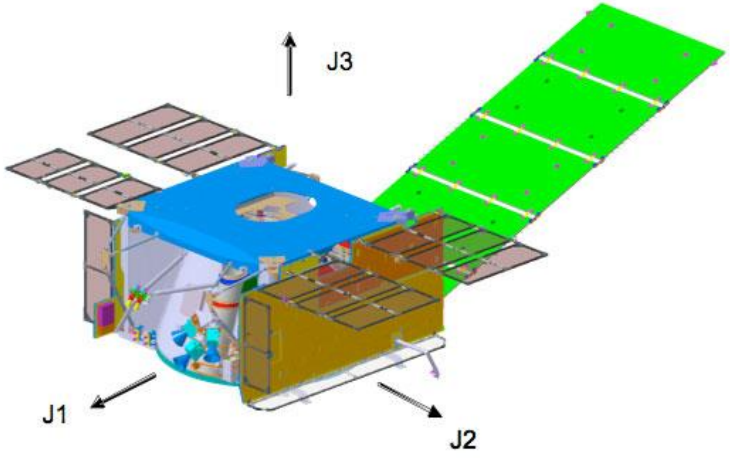
Cubesat Reaction Wheels (x4)



Cubesat Magnetotorques (3-axes)

Example of 3-axis Attitude Control

Star Tracker System



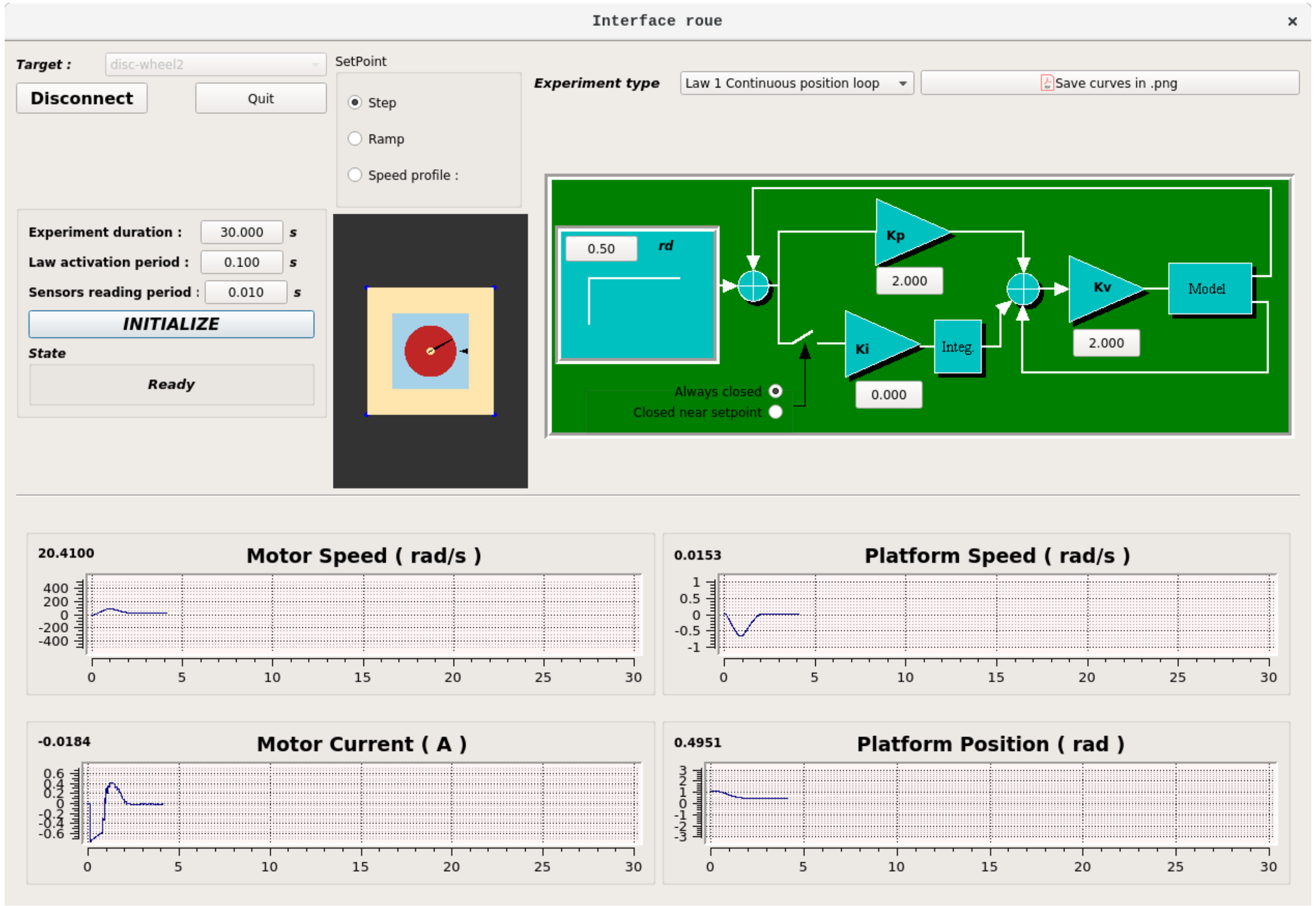
James Webb Spacecraft Bus (6x Gyros + Reaction Wheels + RCS)

- A Star tracker for the attitude measurement.
- 3 gyrometers for the angular speed measurement, and the estimated attitude (angular speed > 0.1°/s).
- 3 reaction wheels to apply torques to the platform.
- 3 magneto torquers (used to reduce wheels speed).

3. Human Machine Interface (HMI)

- *HMI Overview*
- *HMI Usage*
- *Communication Protocol*
- *Law Selection Overview*

HMI Overview (1)



HMI Overview (2)

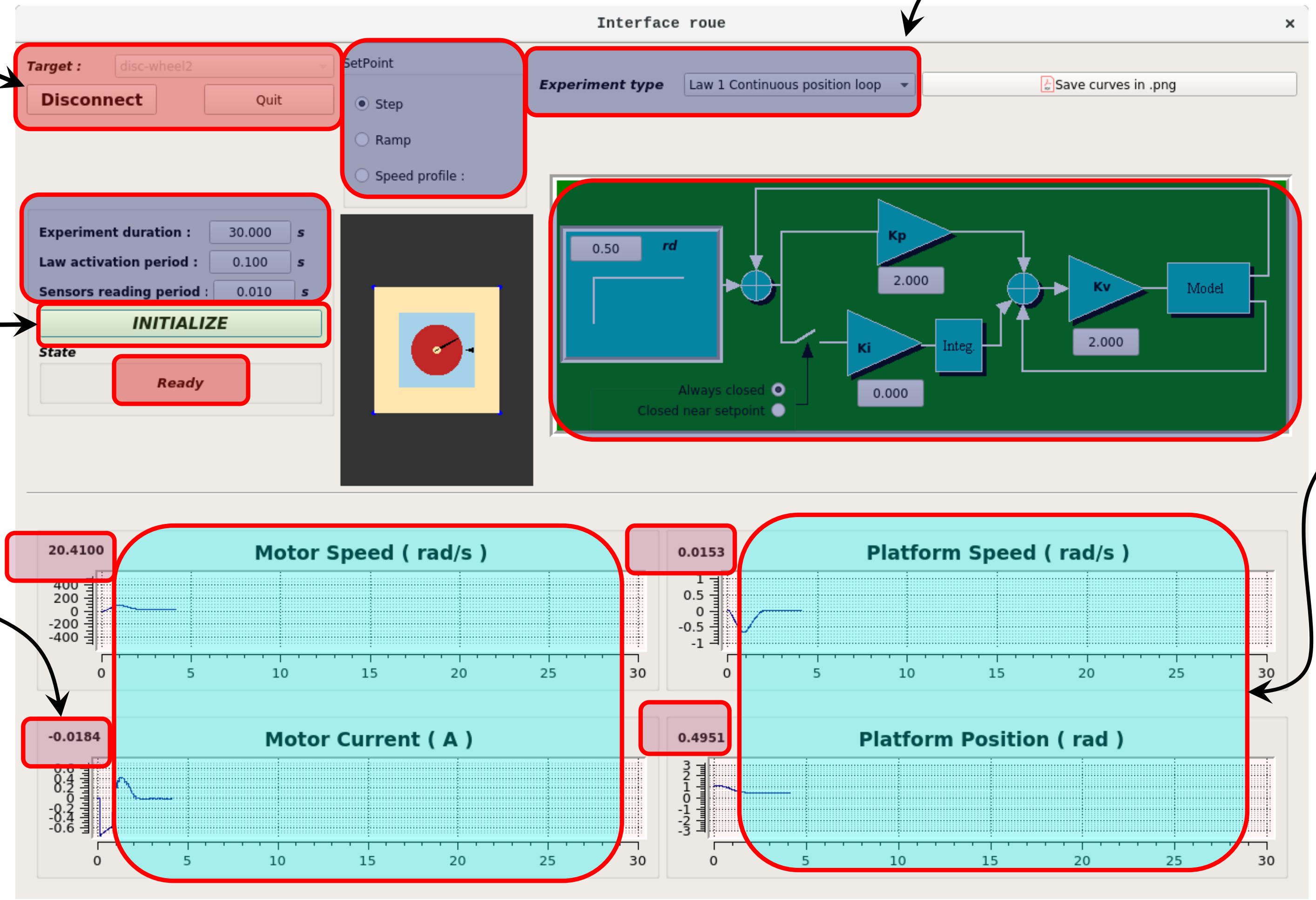
Experiment parameters setting

Target :
Selection
Connect/Disconnect

mode selector button

Sensors values display (Initialization)

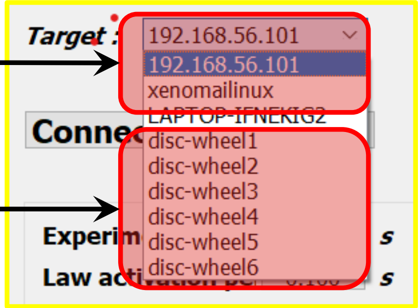
Sensors values display (Experiment)

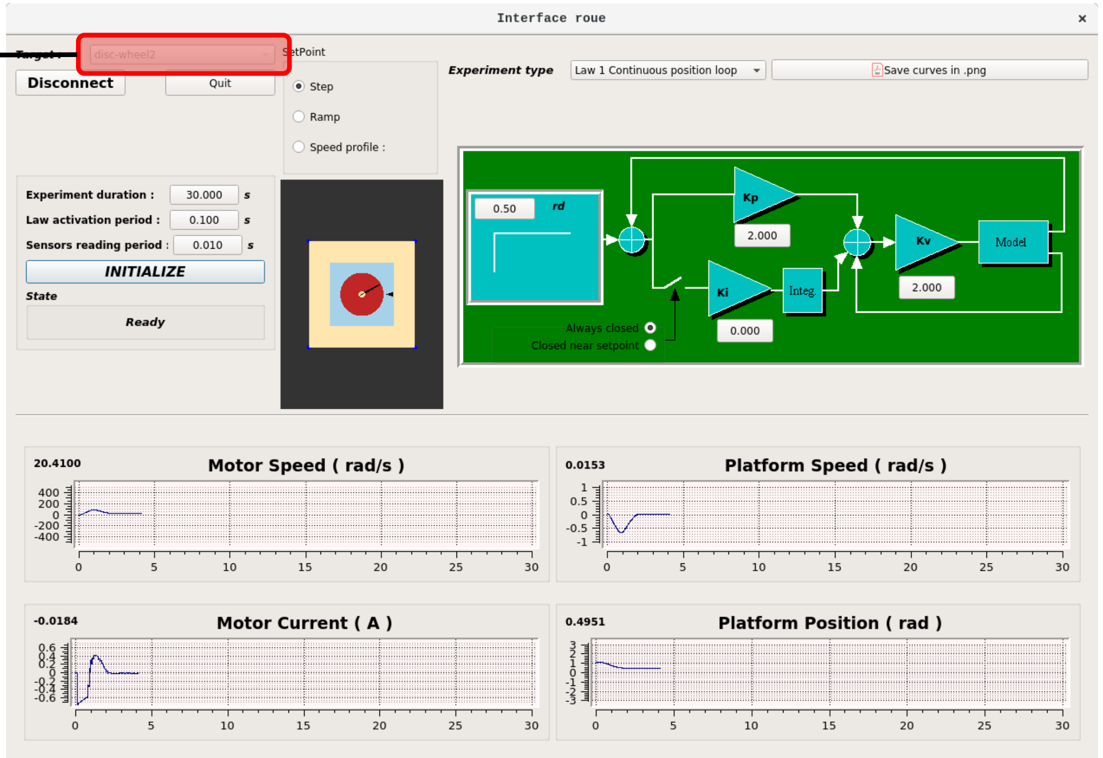


» Target Selection

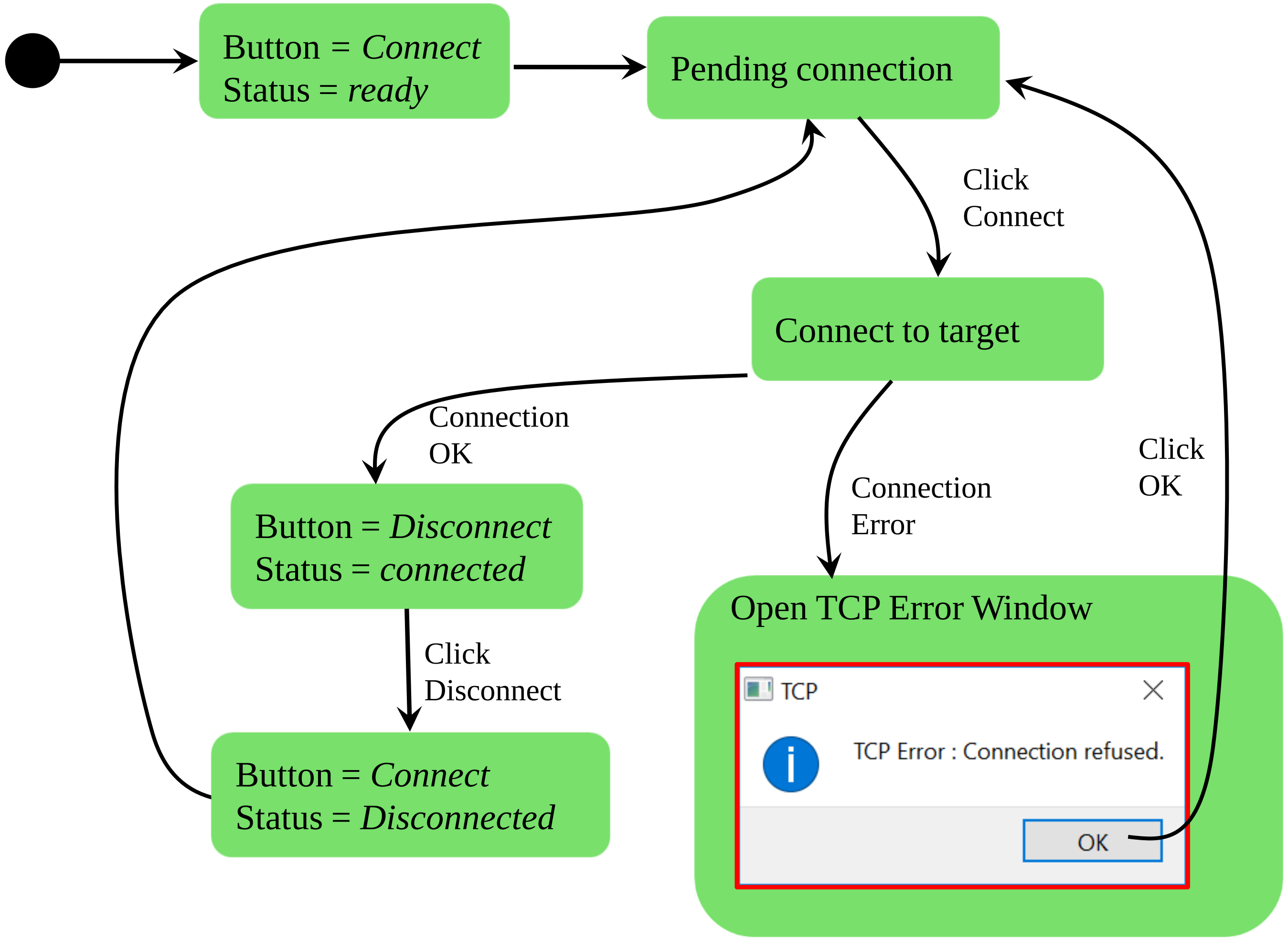
Virtual machine target

Real targets

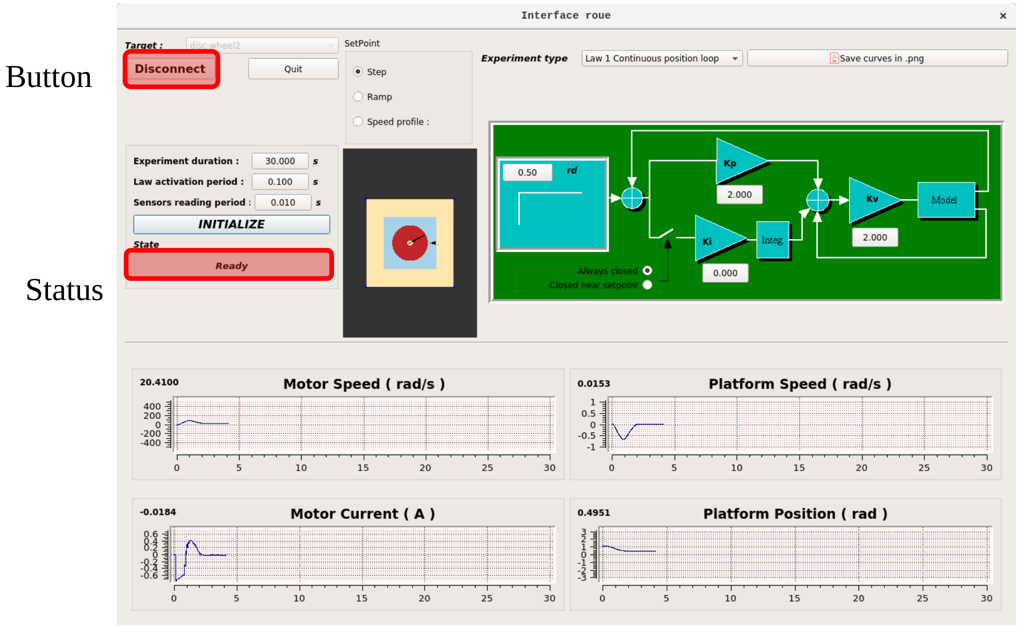




» Target Connection

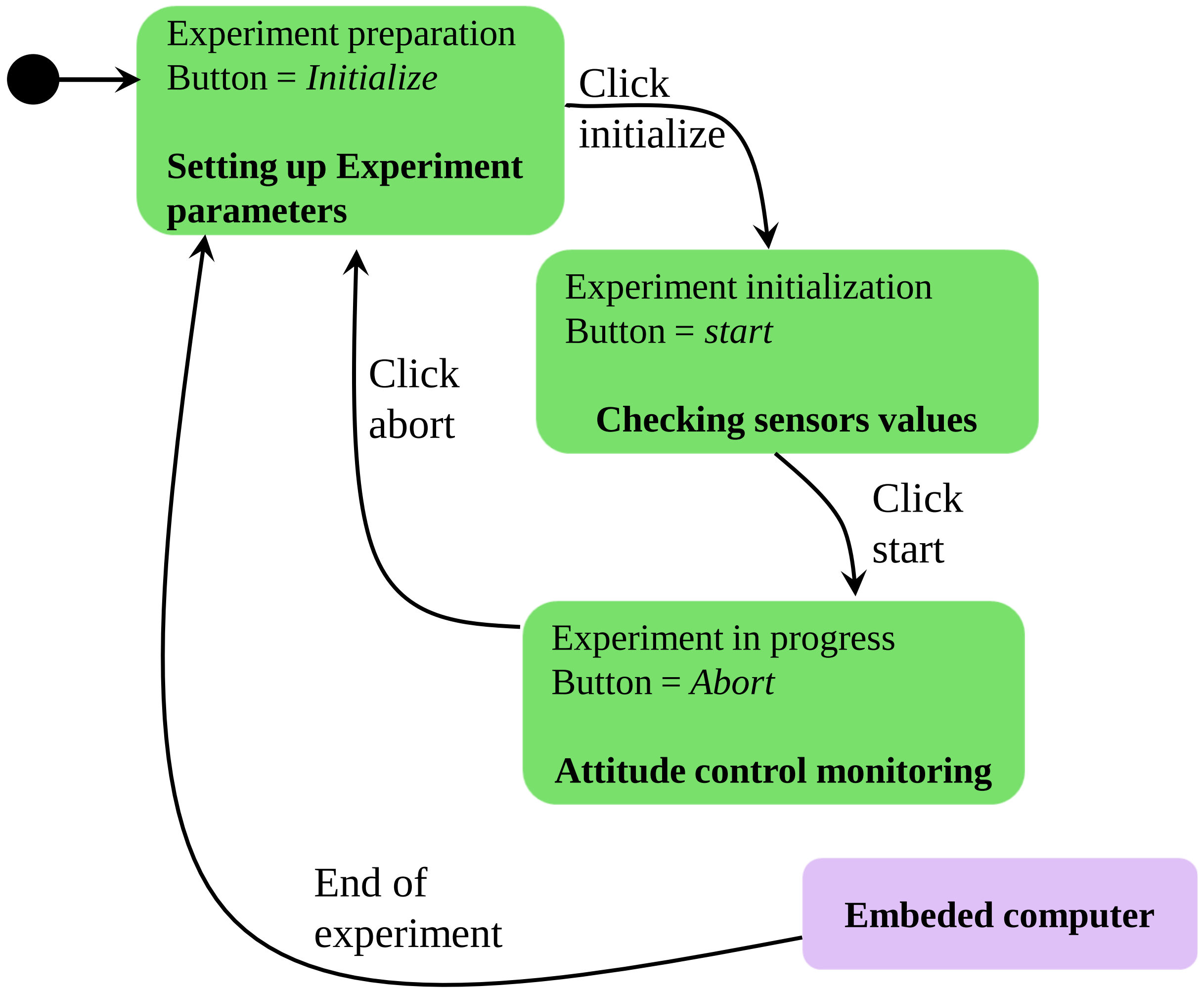


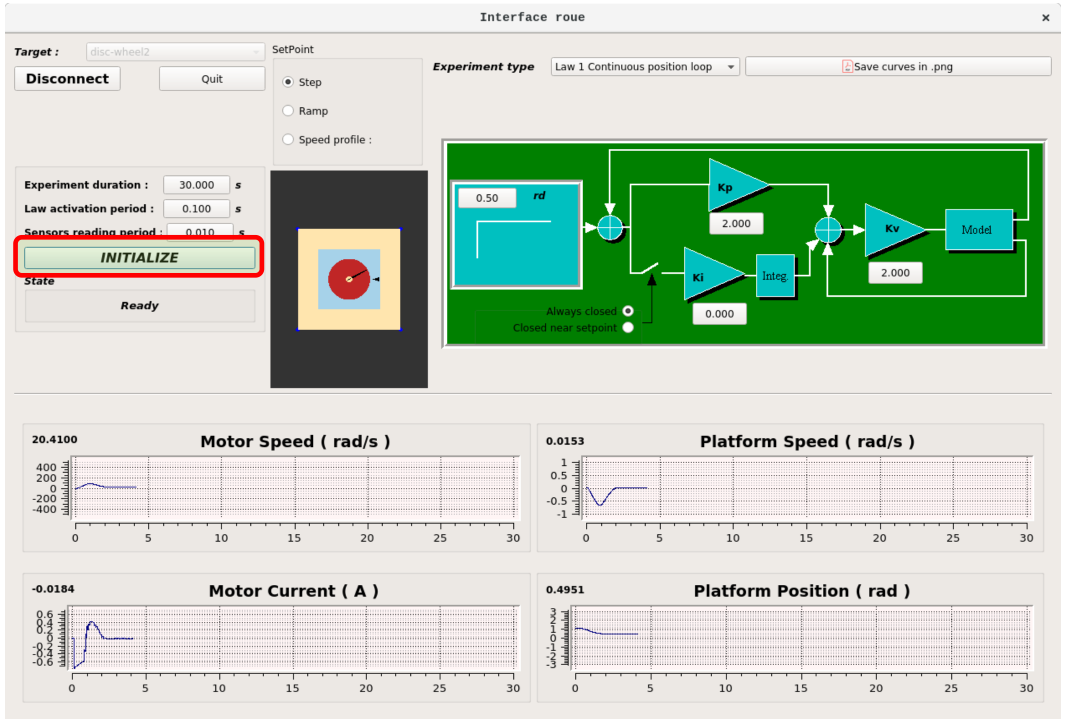
To avoid a connection error. The target must be launched before connection



HMI Usage (3)

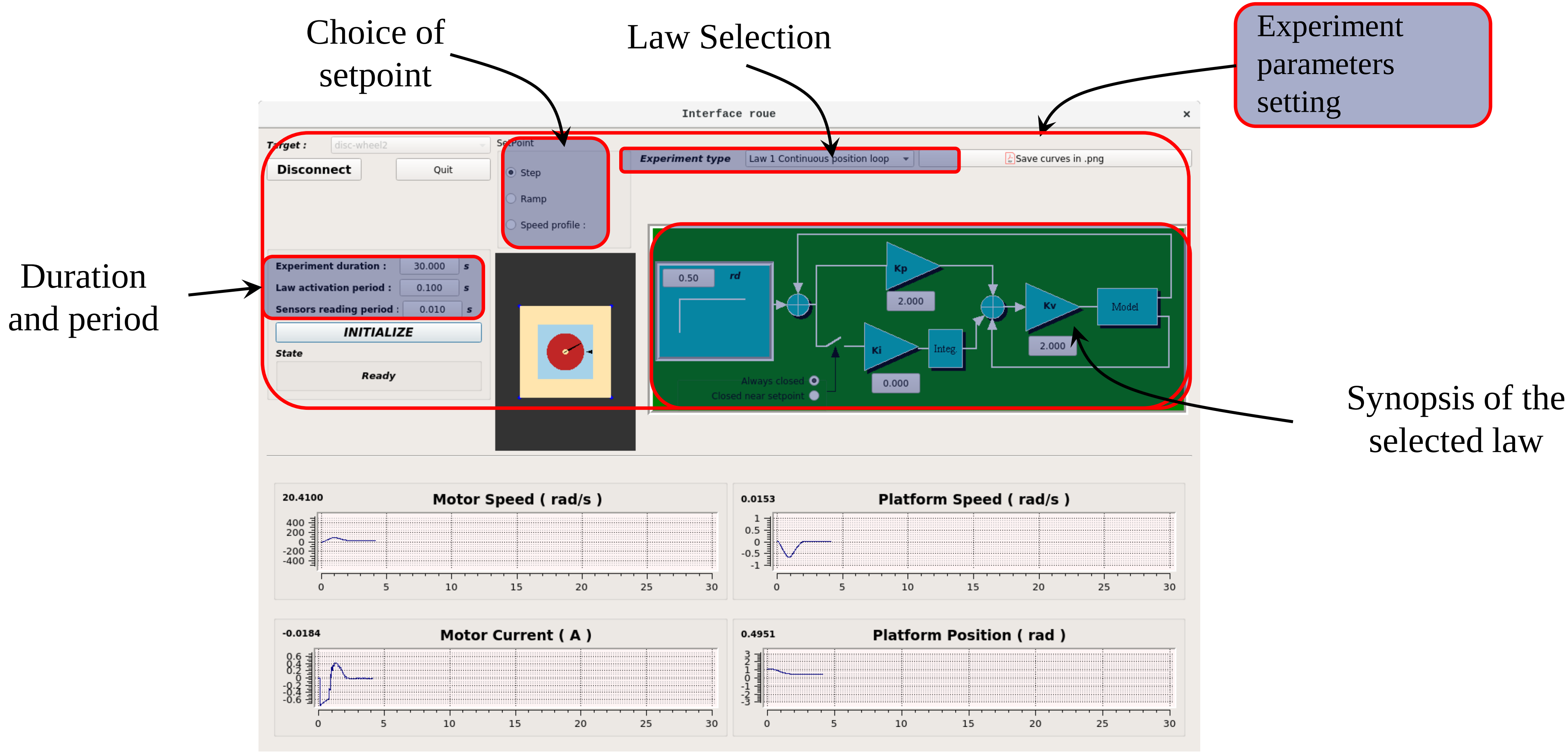
» Mode selector button





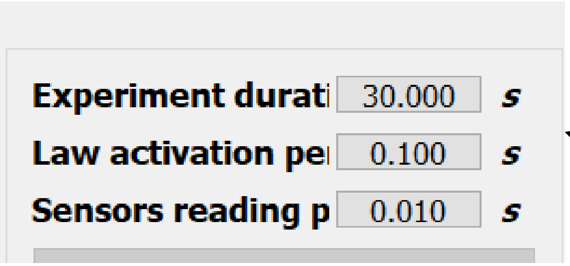
HMI Usage (4)

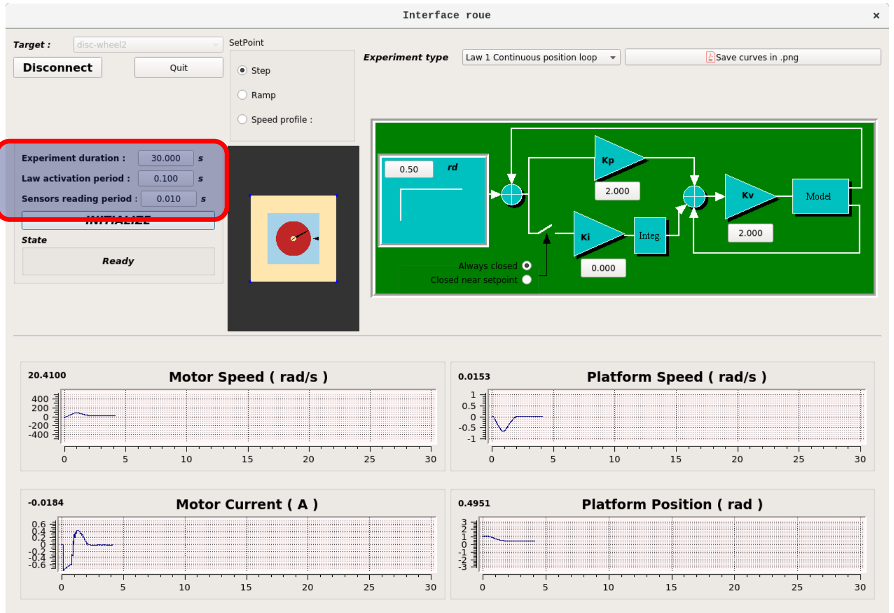
» Experiment Parameters setting



HMI Usage (5)

» Duration and period





The user can adjust by clicking on the values :

- **The experiment duration** (0.01 to 30 seconds)
- **The law activation period** (0.01 to 0.5 second)
- **The reading sensors period** (0.002 to 0.2 second)

HMI Usage (6)

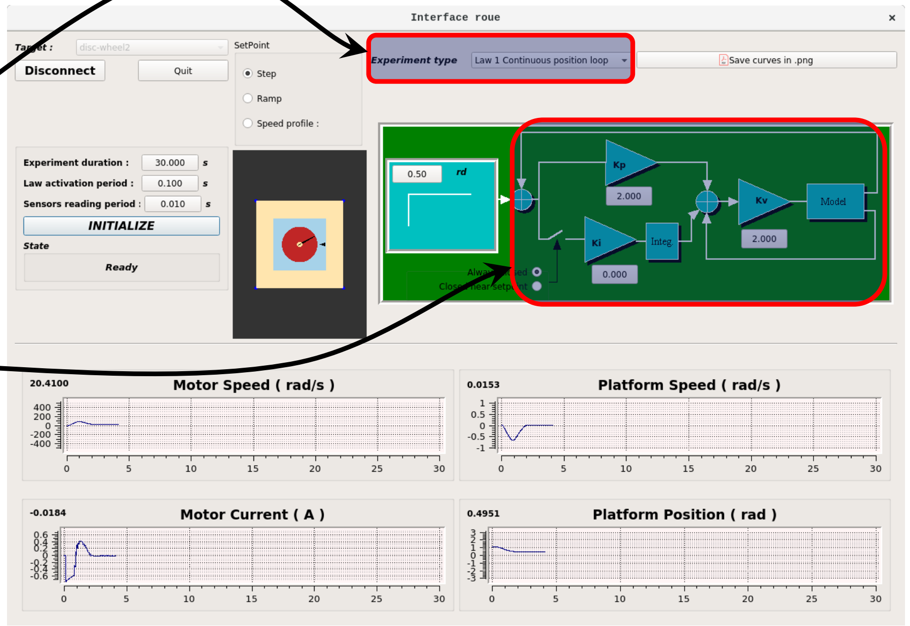
» Law Selection

One of the 8 following laws may be selected:

- **Open loop**
  - Open loop
- **Static gains controled laws**
  - Speed loop
  - Continuous position loop
  - State feedback
  - State feedback with integrator
- **Dynamic laws**
  - Discrete position loop
  - Lead compensator loop
  - Lead lag compensator loop

Area to click for law selection

synopsis automatically updated after a new law selection



Communication Protocol (1)

Attitude control operator (HMI)

Requests:

Responses

Embedded computer

Fields :

Messages composition

#1	#2	#3	#4
Message ID	Data Type	Data Length	Datas

- Datas: Array of data itself
- Data Length: Number of elements in data field
- Data Type: Type of elements in data field (string, integer, array of reals ....)
- Message ID: A single character identifying request/response message

Communication Protocol (2)

» Type of Messages Sent/Received by the HMI

Attitude control operator (HMI)

Exchanged messages

Embedded computer

Message	Request (from Human Interface)	Reply (from embedded application)
Set experiment parameters	Message ID = ‘L’ Data = array of 23 reals	Message ID = ‘A’ Data = Empty
Start experiment	Message ID = ‘D’ Data = Empty	Message ID = ‘A’ Data = Empty
Abort experiment	Message ID = ‘A’ Data = Empty	Message ID = ‘A’ Data = Empty
Get current sensors values (Initialize)	Message ID = ‘C’ Data = Empty	Message ID = ‘T’ Data = array of 4 reals
Get last N sensors values (Running)	Message ID = ‘B’ Data = Empty	Message ID = ‘S’ or ‘F’ Data = n * array of 4 reals

Communication Protocol (3)

» Experiment parameters message structure

Message ID = ‘L’

Array of 20 reals	Set experiment message : data description	Unit
Data[0]	Type of law	NA
Data[1]	Law option	NA
Data[2]	Experiment duration	Millisecond
Data[3]	Law activation period	Millisecond
Data[4]	Reading sensors period	Millisecond
Data[5]	Setpoint type (step / ramp / speed)	NA
Data[6]	Final setpoint value	Radian
Data[7]	Setpoint rise duration	Millisecond
Data[8]	Maximum setpoint speed	Radian/s
Data[9]	Setpoint acceleration	Radian/s^2
Data[10 .. 20]	Law coefficients	NA

» Sensors values array structure (current / last)

	Request (HMI)	*Response (embedded computer)*
Init :	**Message ID = ‘C’**	**Message ID = ‘T’**
Run :	**Message ID = ‘B’**	**Message ID = ‘S’ or ‘F’**

Array of 4 reals	Sensors values message : data description	Unit
Data[0]	Motor speed	Radian/s
Data[1]	Platform speed	Radian/s
Data[2]	Platform position	Radian
Data[3]	Motor current	Ampere

Law Selection Overview

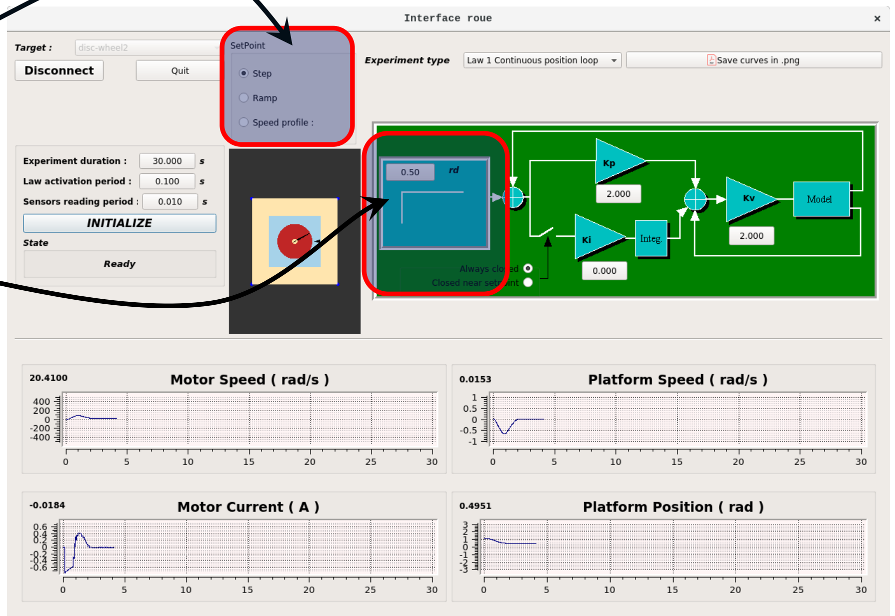
» List of available control laws :

- **Open loop**
  - Open loop
- **Static gains controled laws**
  - Speed loop
  - Continuous position loop
  - State feedback
  - State feedback with integrator
- **Dynamic laws**
  - Discrete position loop
  - Lead compensator loop
  - Lead lag compensator loop

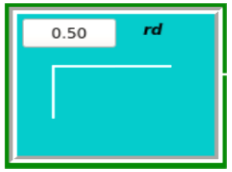
» Setpoint overview :

Area to click for law selection

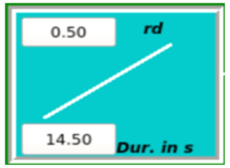
Synopsis automatically updated after a new setpoint selection



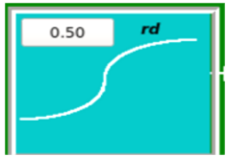
Step



Ramp

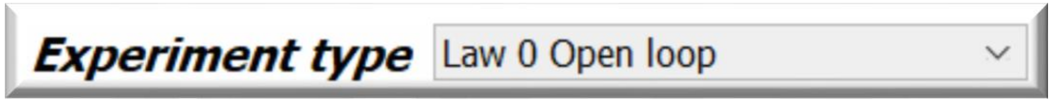


Speed Profile

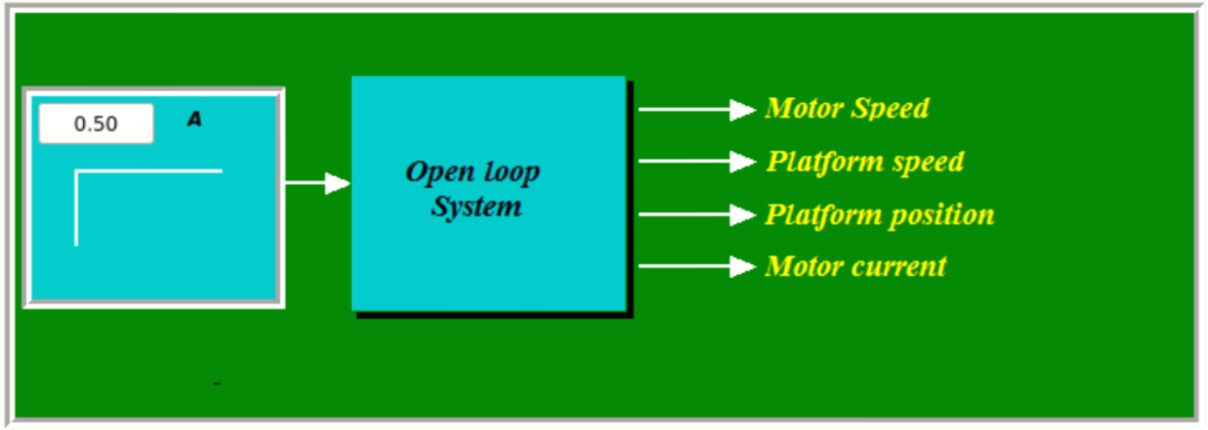


Law Selection (1)

» Open Loop



Synopsis :



Law Type : 0

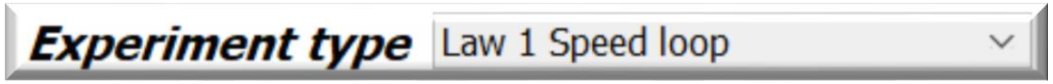
Setpoint unit:
- Amps

Setpoint type :
- Step
- Ramp

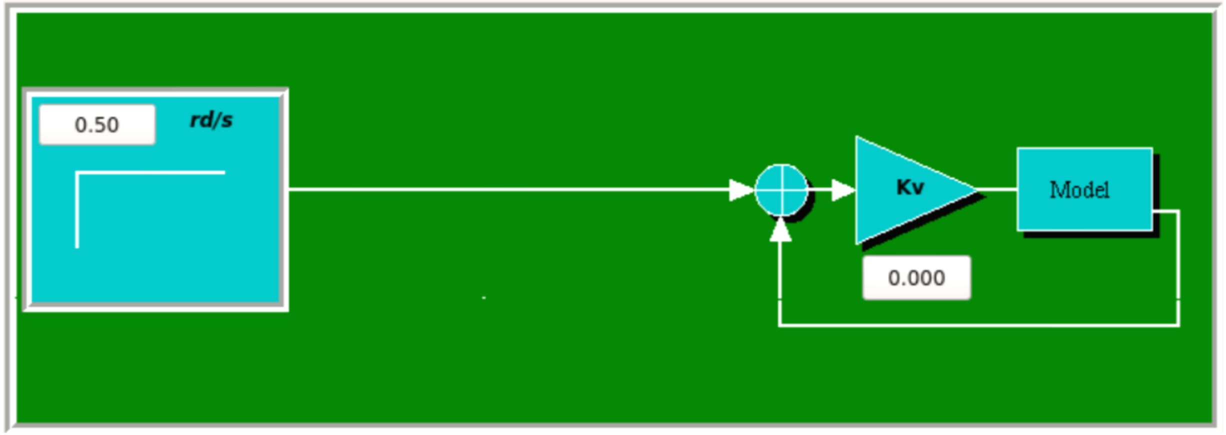
Law parameters
- None

Law Selection (2)

» Speed Loop



Synopsis :



Law Type : 10

Setpoint unit:
- rd/s

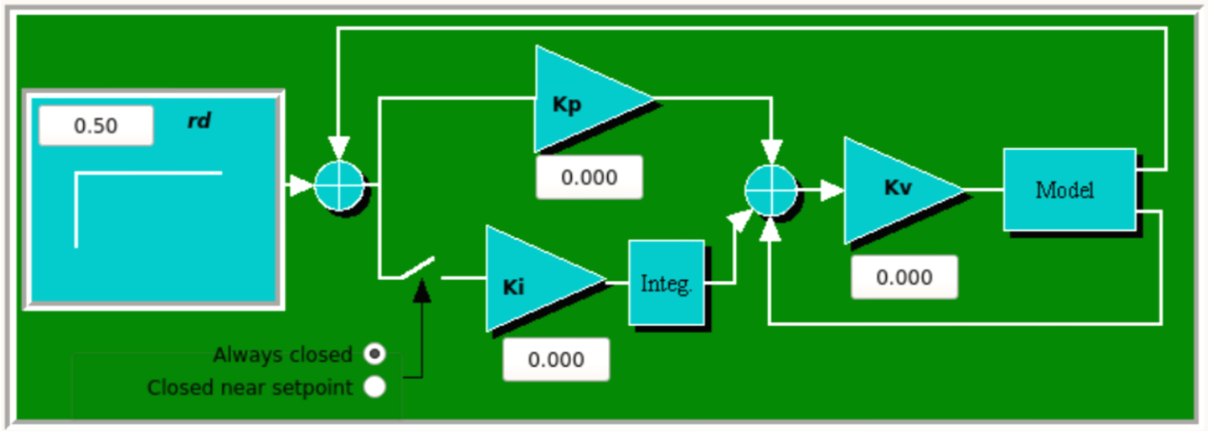
Setpoint type :
- Step
- Ramp

Law parameters
- Kv : LawCoeff[0]

Law Selection (3)

» Continuous position Loop

Synopsis :



Law Type : 11

Setpoint unit:
- rd

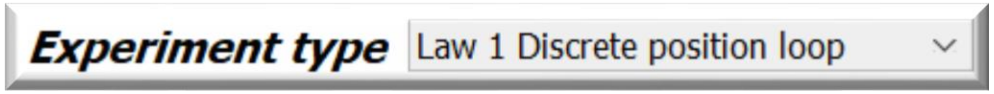
Setpoint type :
- Step
- Ramp
- Profile

Law parameters :
Ki : LawCoeff[0]
Kp : LawCoeff[1]
Kv : LawCoeff[2]

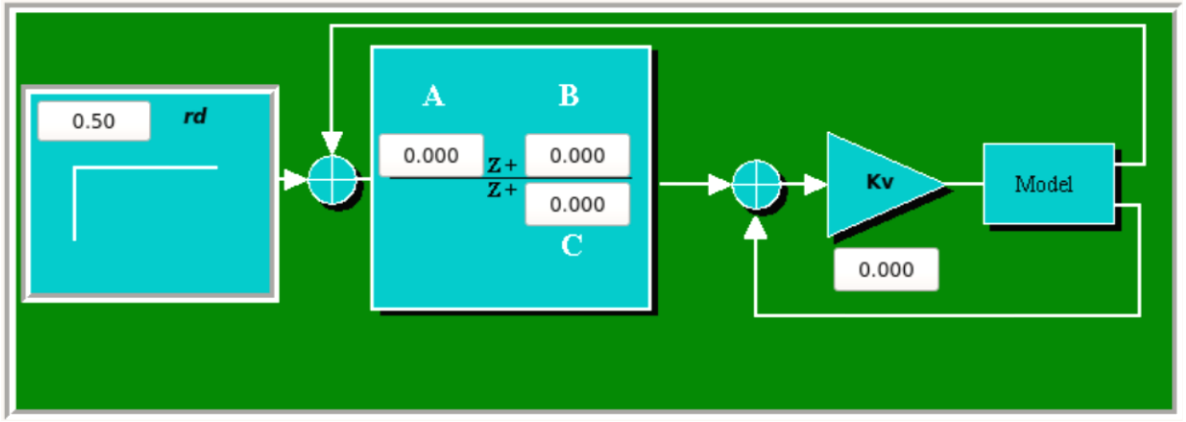
Law option :
Integ command
- Near setpoint
- Always closed

Law Selection (4)

» Discrete position Loop



Synopsis :



Law Type : 12

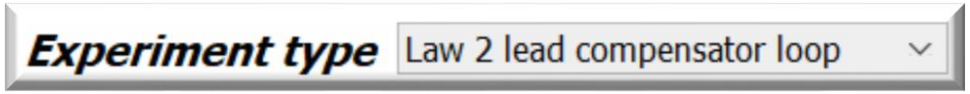
Setpoint unit:
- rd

Setpoint type :
- Step
- Ramp
- Profile

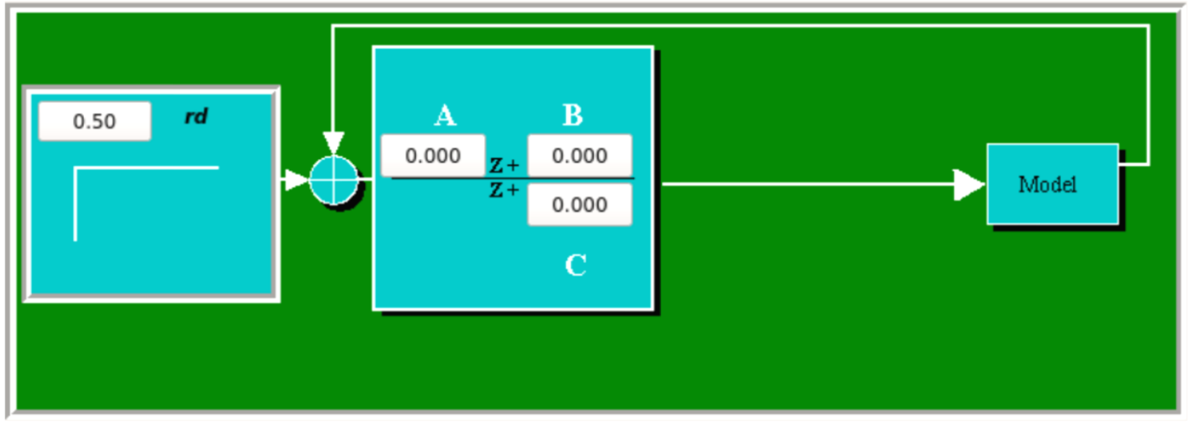
Law parameters
A : LawCoeff[0]
B : LawCoeff[1]
C : LawCoeff[2]
Kv :LawCoeff[3]

Law Selection (5)

» Lead Compenstor Loop



Synopsis :



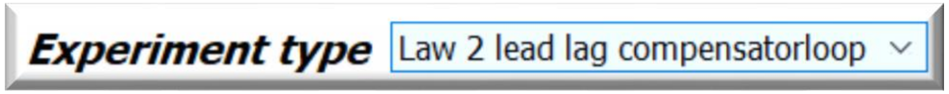
Law Type : 20

Setpoint unit:
- rd

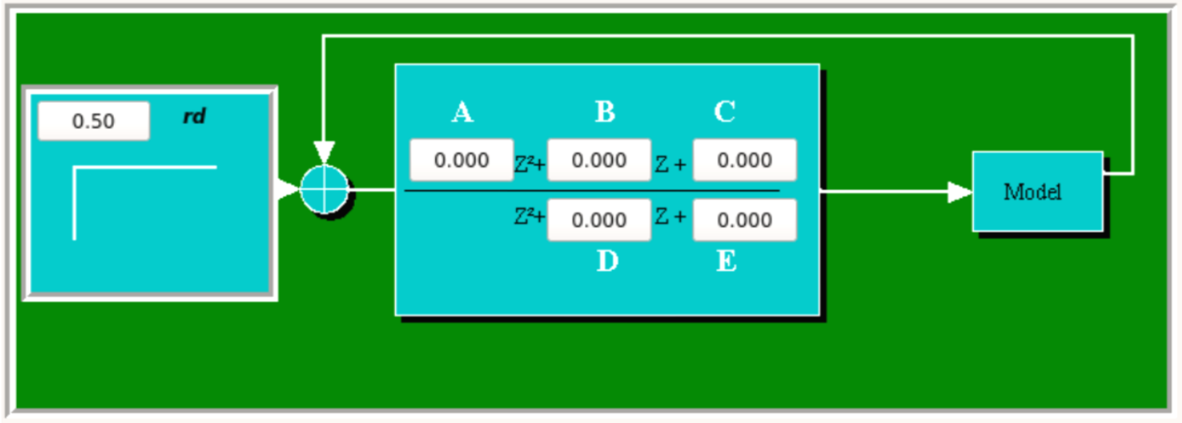
Setpoint type :
- Step
- Ramp
- Profile

Law parameters :
A : LawCoeff[0]
B : LawCoeff[1]
C : LawCoeff[2]

» Lead-Lag Compensator



Synopsis :



Law Type : 21

Setpoint unit:
- rd

Setpoint type :
- Step
- Ramp
- Profile

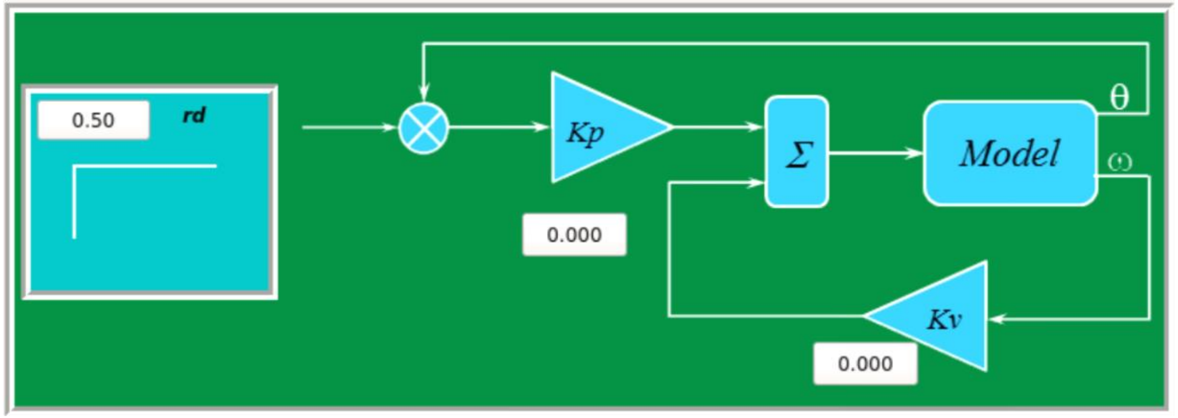
Law parameters :
A:LawCoeff[0]
B: LawCoeff[1]
C: LawCoeff[2]
D: LawCoeff[3]
E : LawCoeff[4]

Law Selection (7)

» State Feedback

Experiment type Law 3 State Feedback

Synopsis :



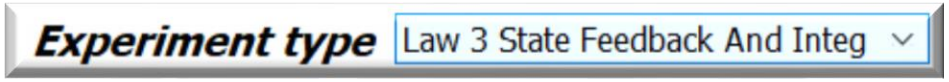
Law Type : 50

Setpoint unit:
- rd

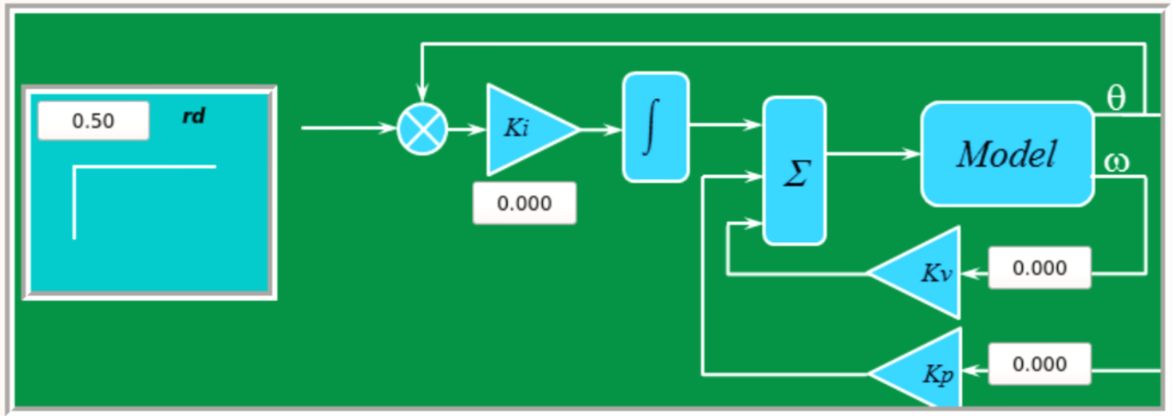
Setpoint type :
- Step
- Ramp
- Profile

Law parameters :
Kp : LawCoeff[0]
Kv : LawCoeff[1]

» State Feedback with Integrator



Synopsis :



Law Type : 51

Setpoint unit:
- rd

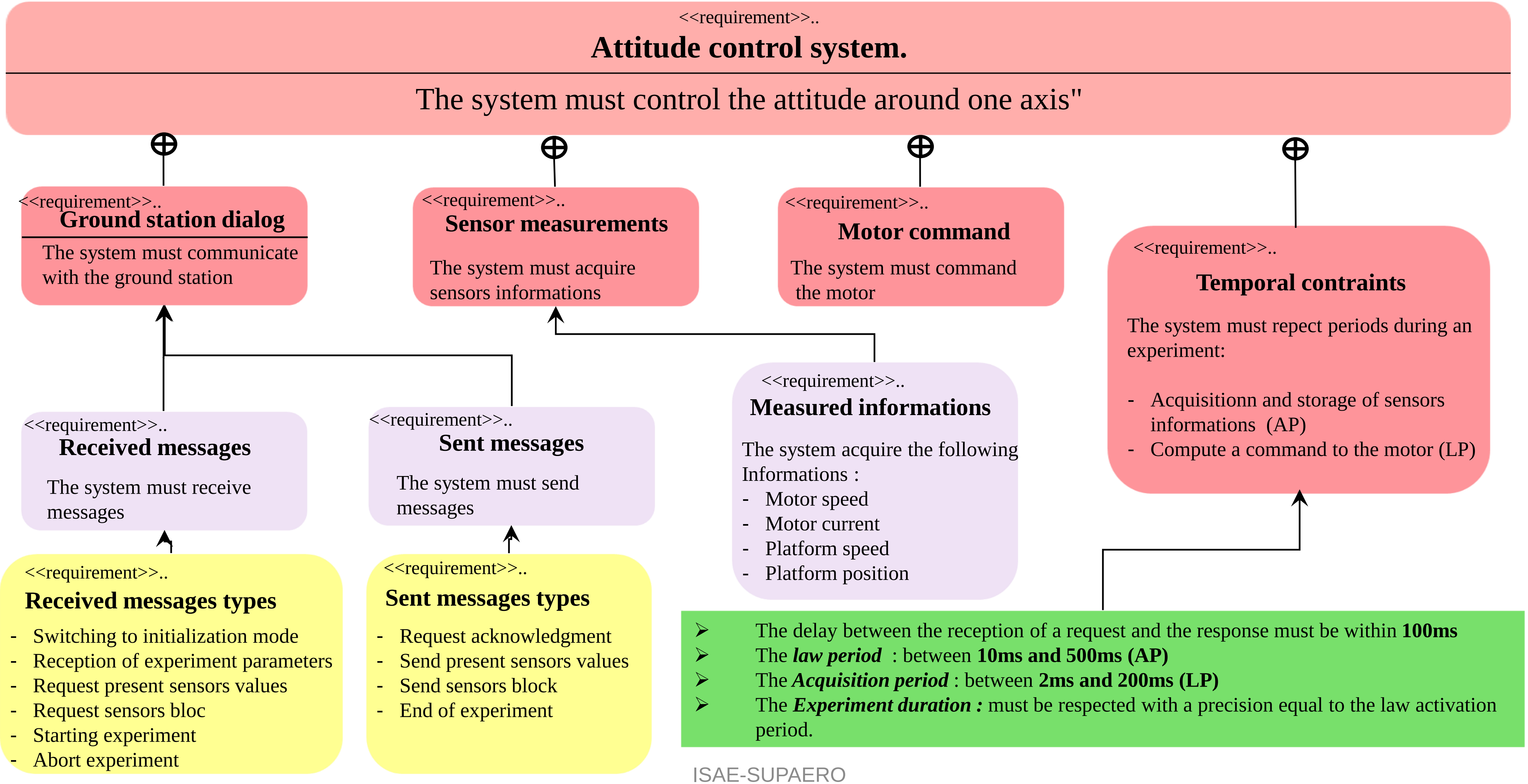
Setpoint type :
- Step
- Ramp
- Profile

Law parameters :
Ki : CoeffLaw[0]
Kp : CoeffLax[1]
Kv : CoeffLaw[2]

4. Application Presentation

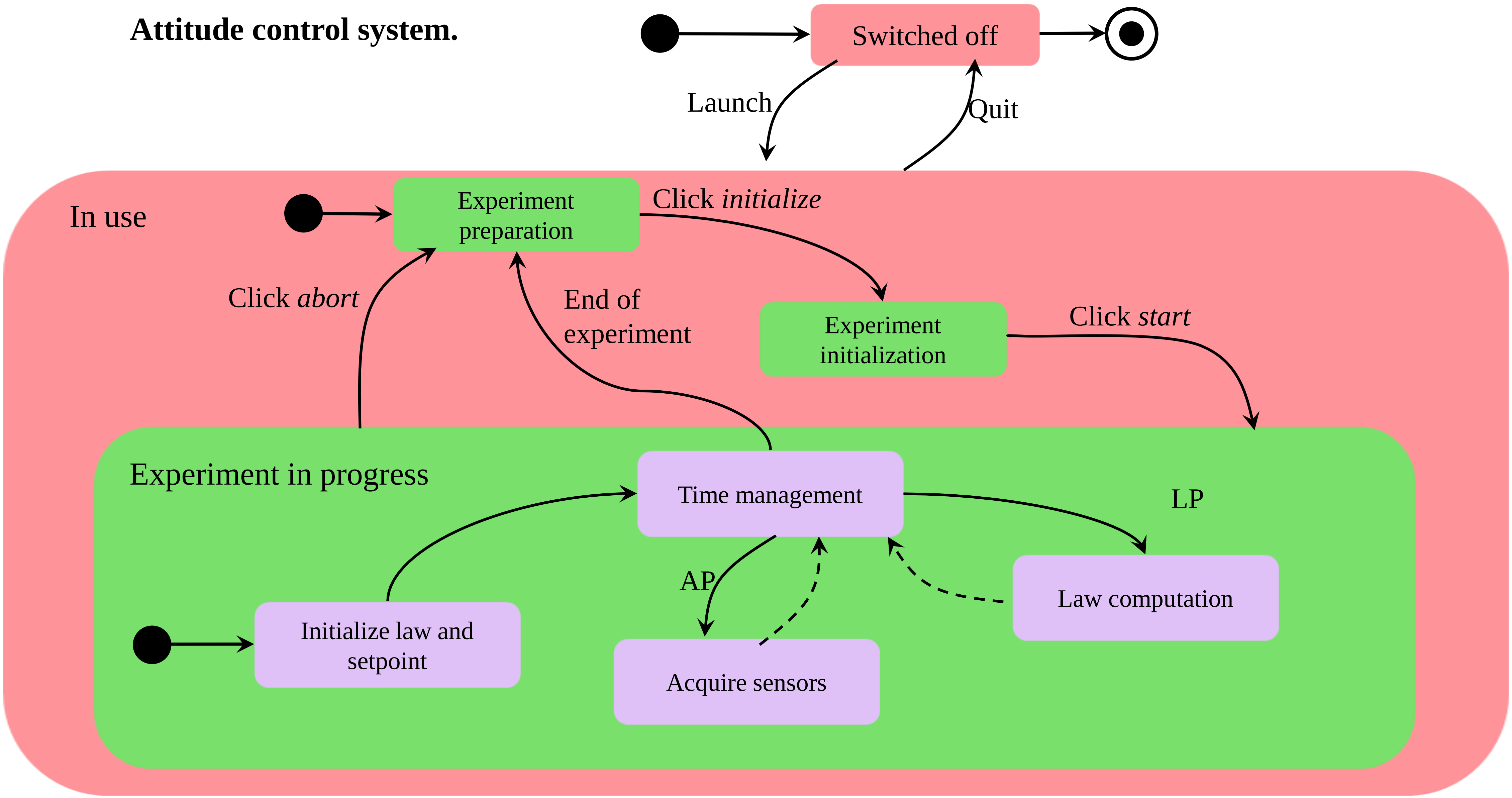
- *SysML Diagrams Overview*
- *Dialog Description*
- *Temporal Constraints*
- *Synthesis Diagram*

Requirements Diagram

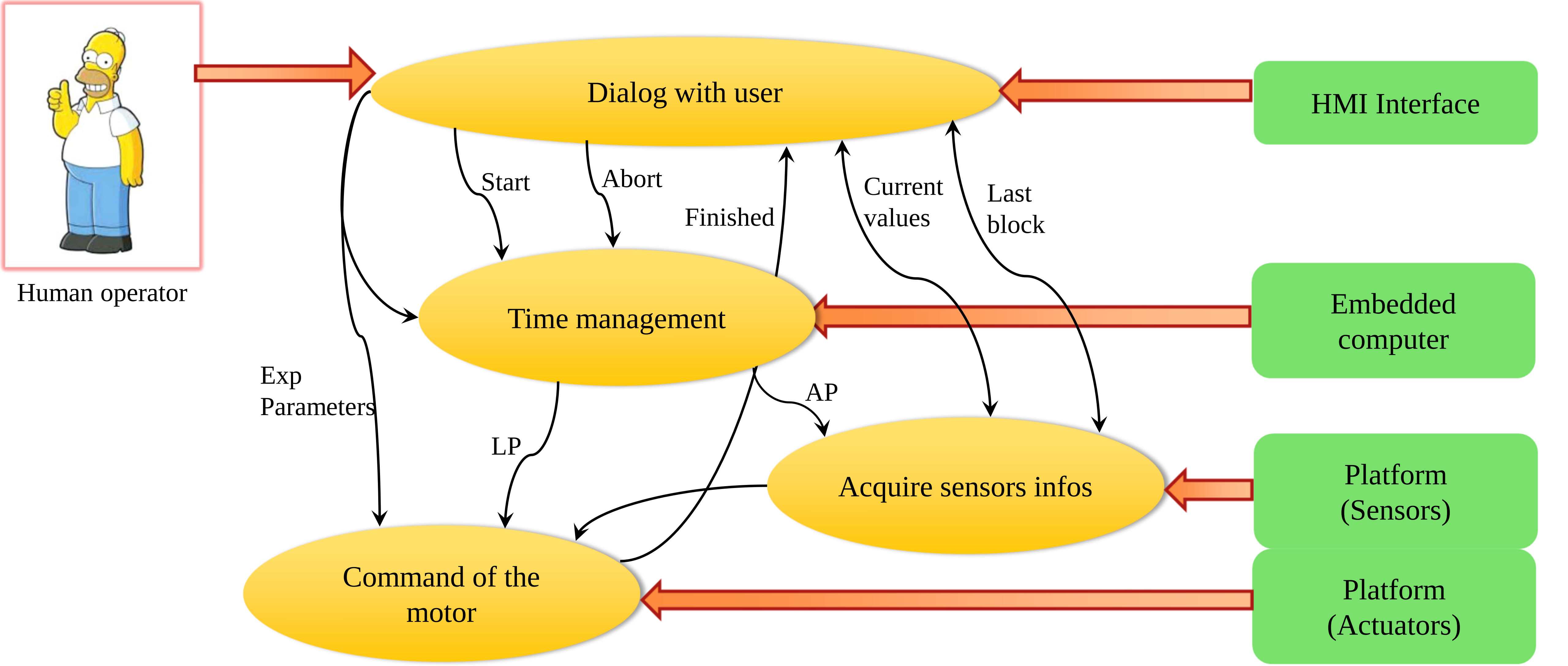


State Machine Diagram

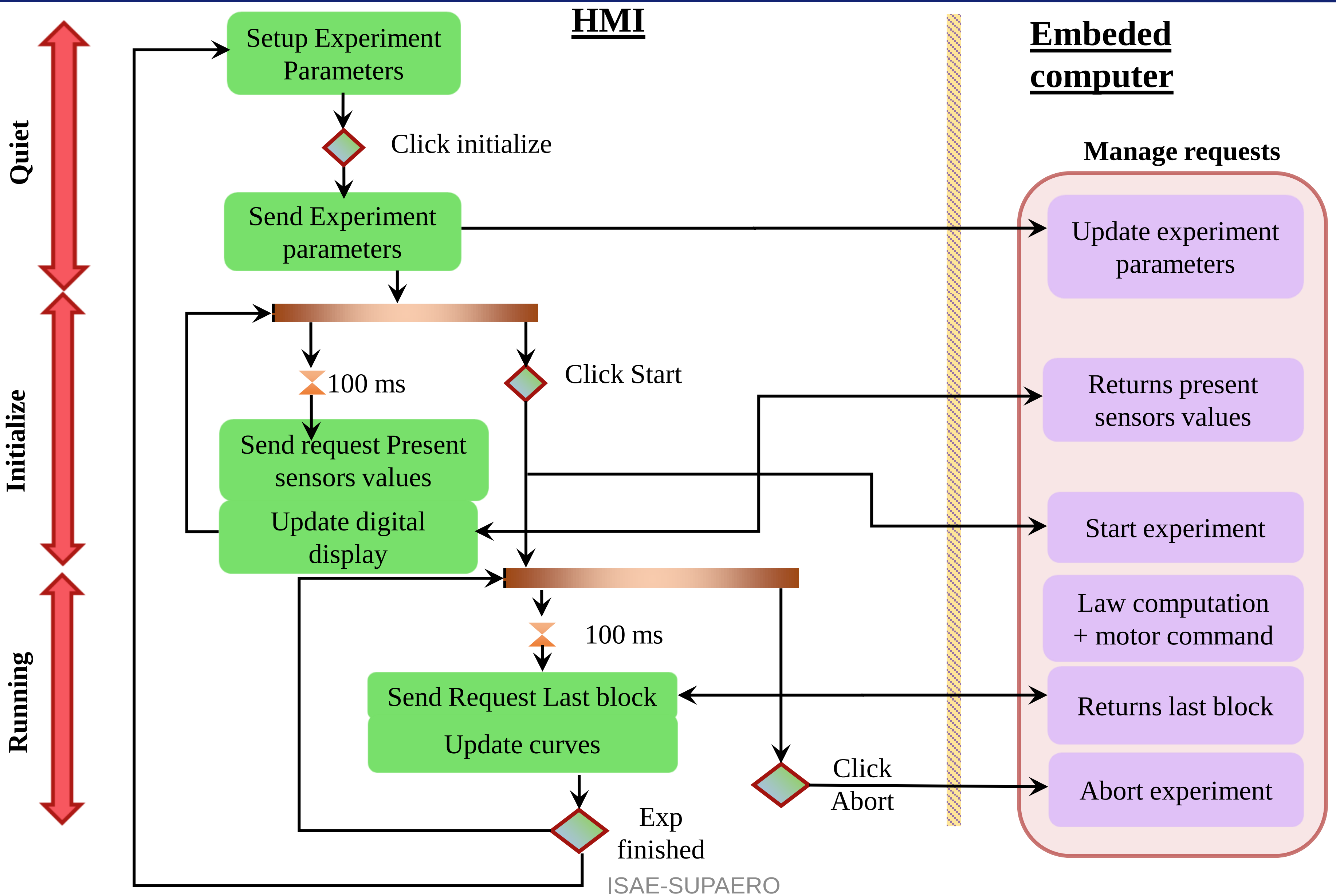
Attitude control system.



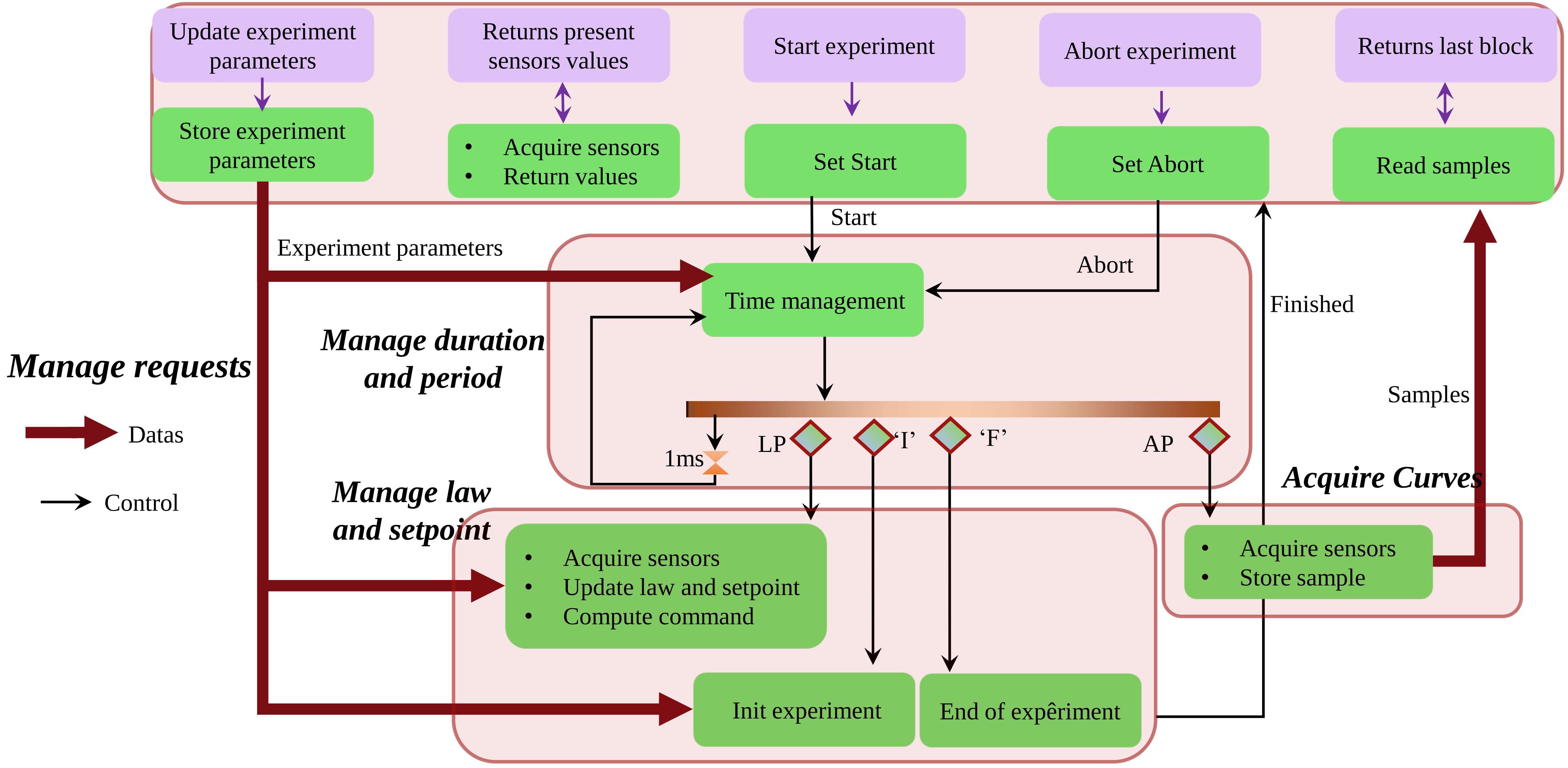
User Case Diagram



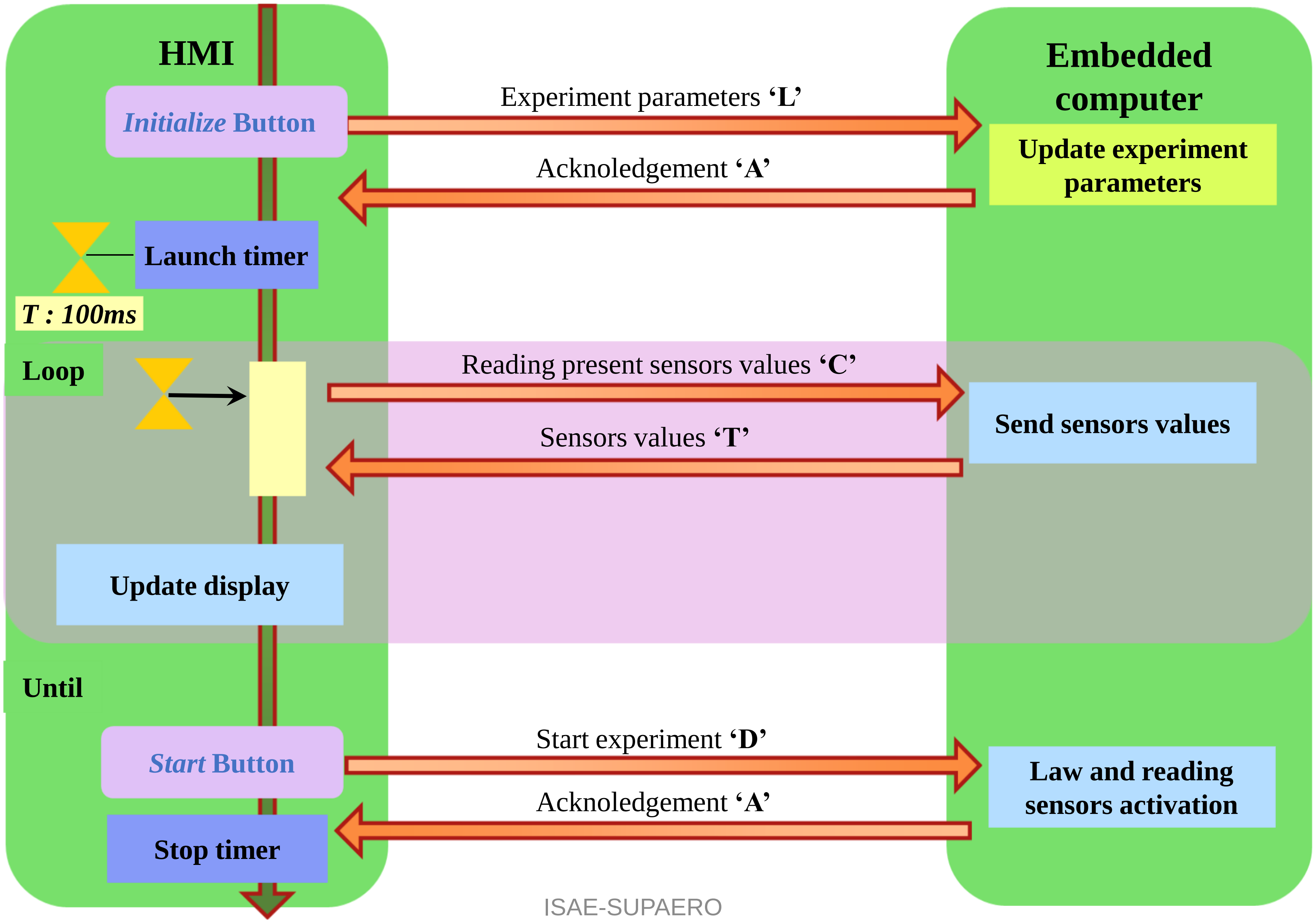
Activity Diagram (1)



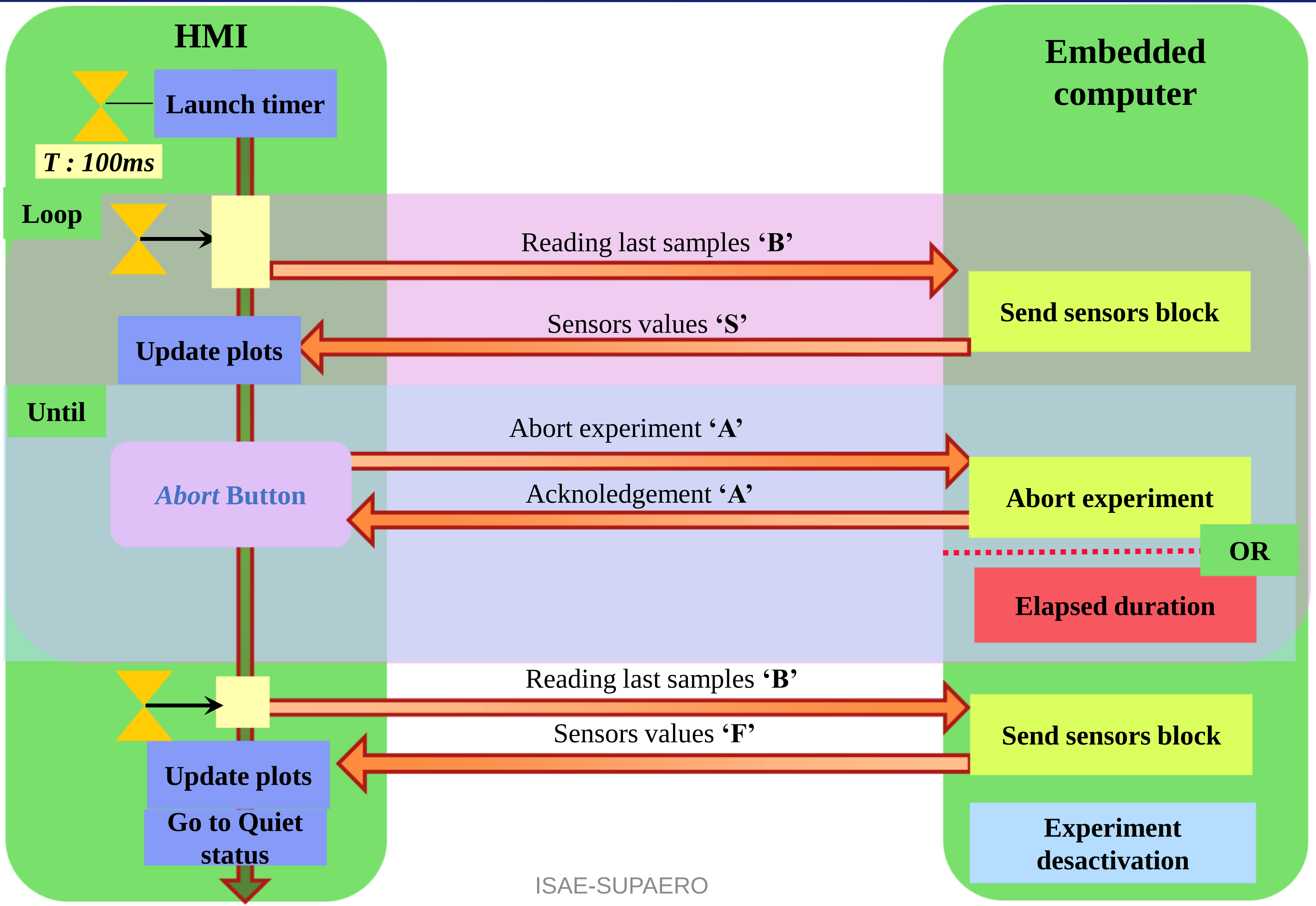
Activity Diagram (2)



Dialog Descript (1)



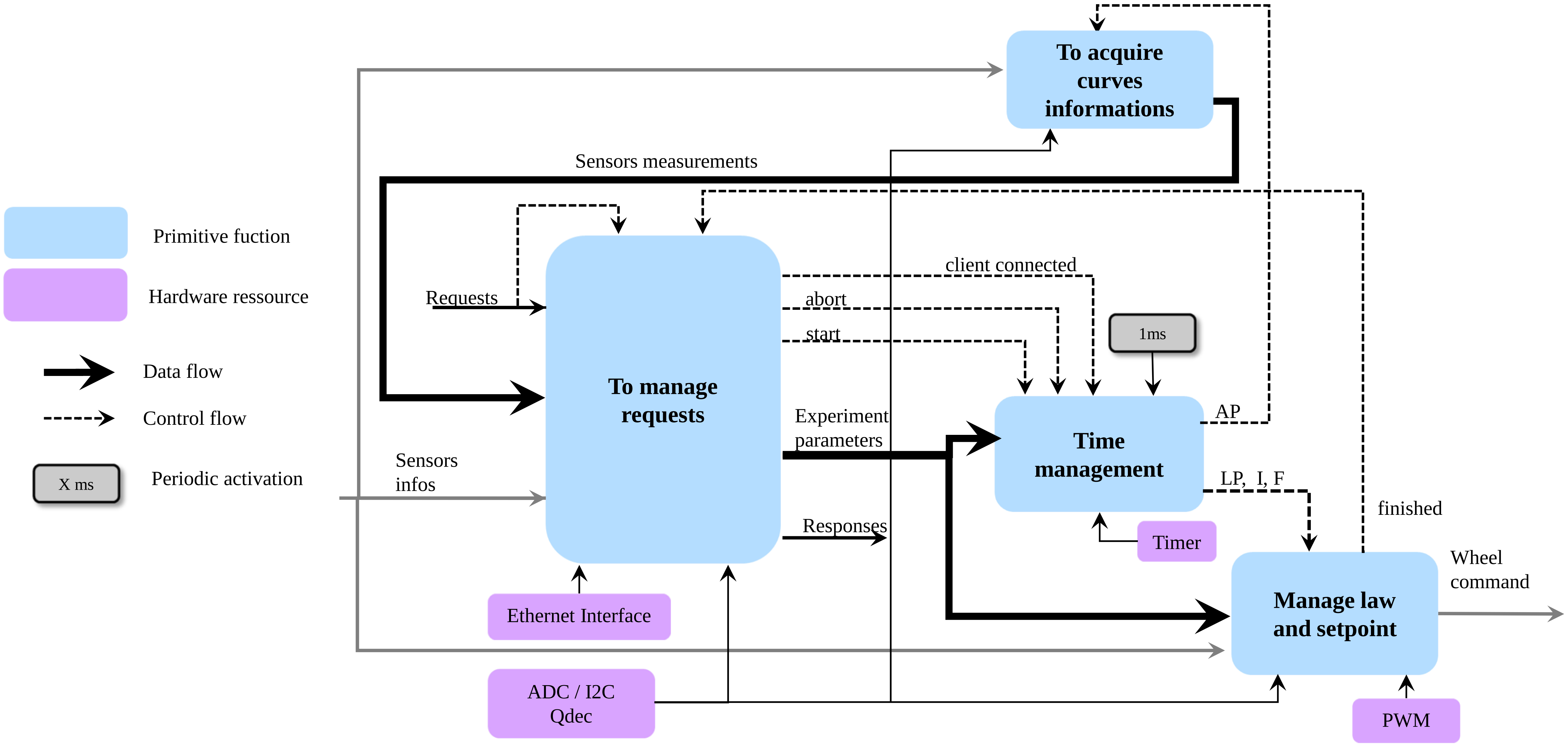
Dialog Descript (2)



Temporal Constraints

Function	Activation	Dead line	WCET (worst case execution time)
Manage request	Request occurence	100 ms	2ms
Manage duration and period	Period 1ms	1ms	0.2ms
Manage law and setpoint	Period LP (10 to 500 ms)	LP	1ms
Acquire Curves	Period AP (2 to 200 ms)	AP	0.3ms

Synthesis Diagram

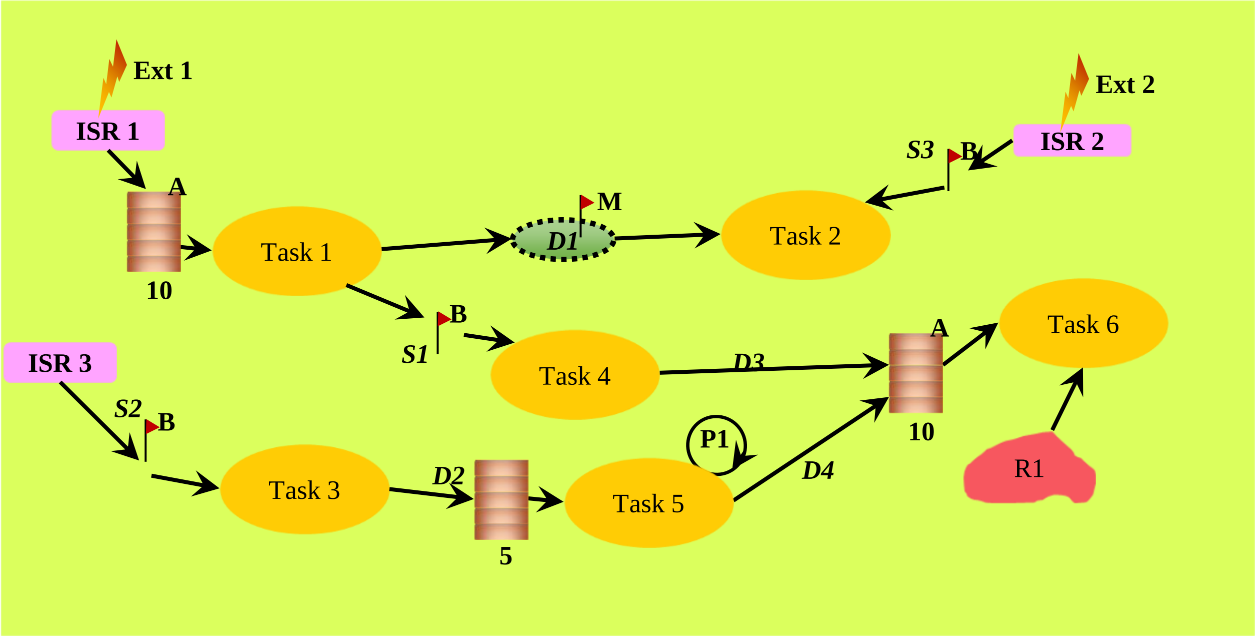


Architecture Definition

Methodology

Application presentation

⇒



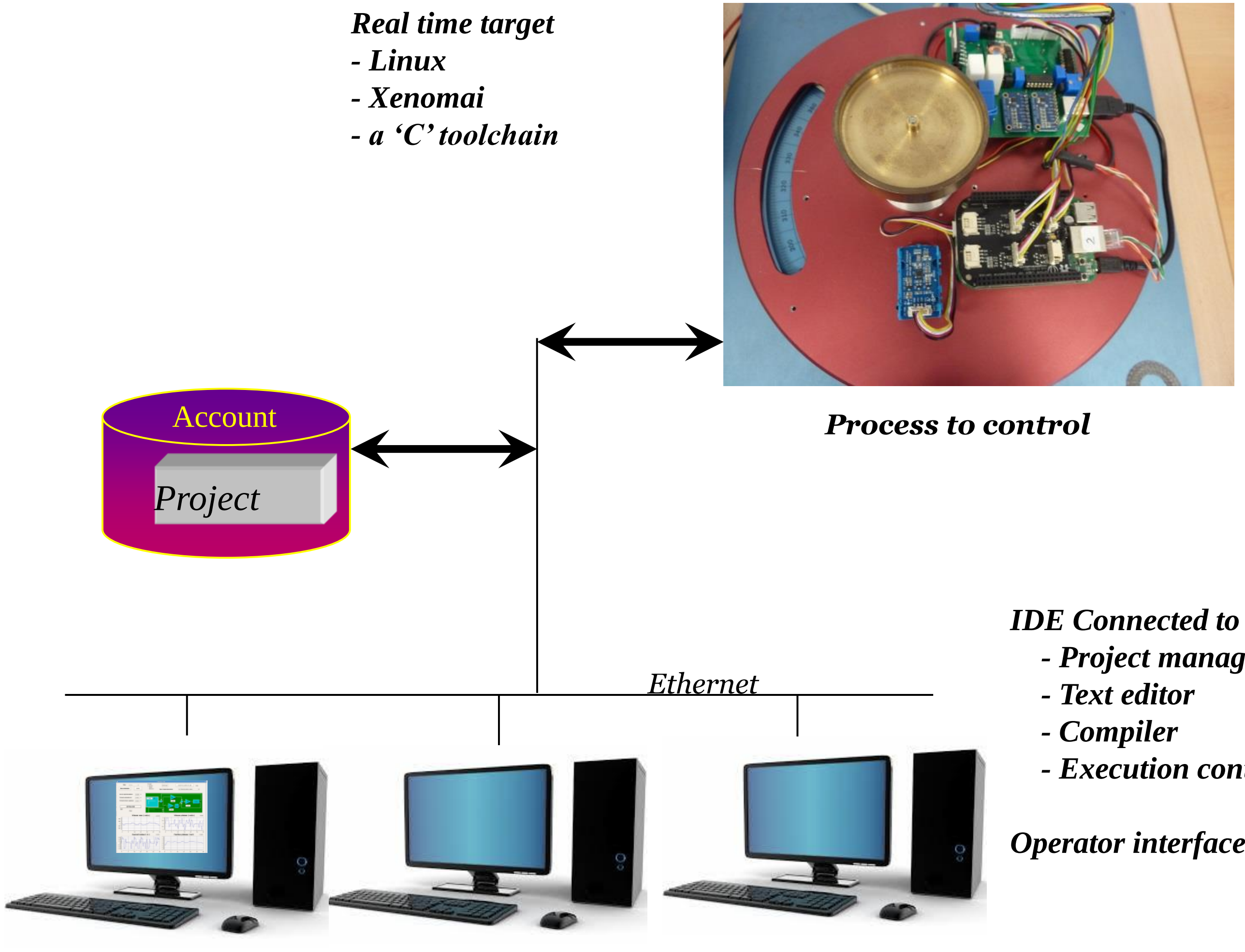
⇓

Advice for newbies:

For those who have little to none experience in coding, it is recommended to write down the pseudocode before coding your application.

Annex. Encoding Information

Hardware Architecture

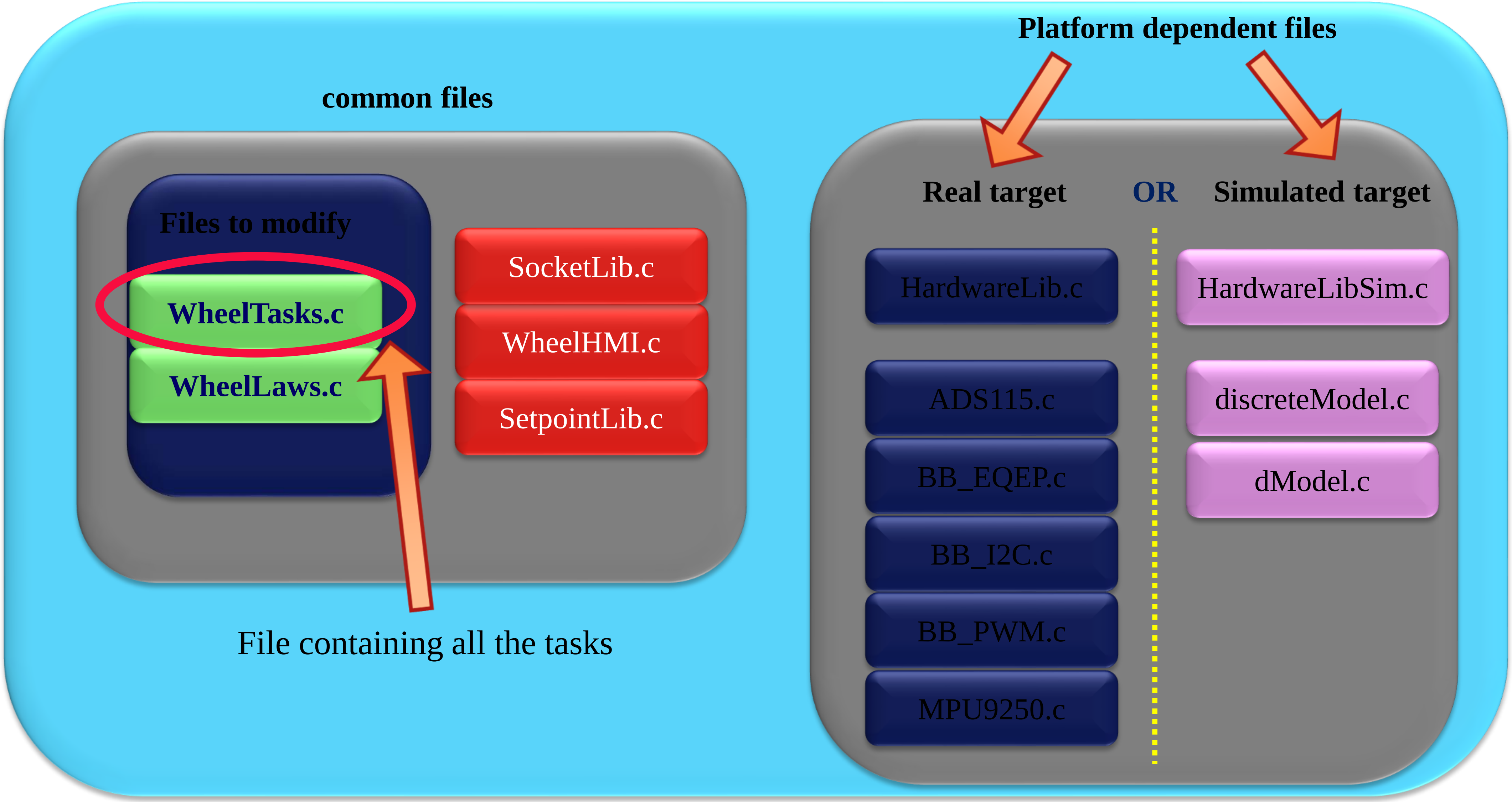


IDE Connected to the target

- ***Project manager***
- ***Text editor***
- ***Compiler***
- ***Execution control***

Operator interface (HMI)

Software application description



Application Software Organization

Advices :

- 1 file containing all the application tasks.

- Backgound task : ***main*** (to complete) containing :

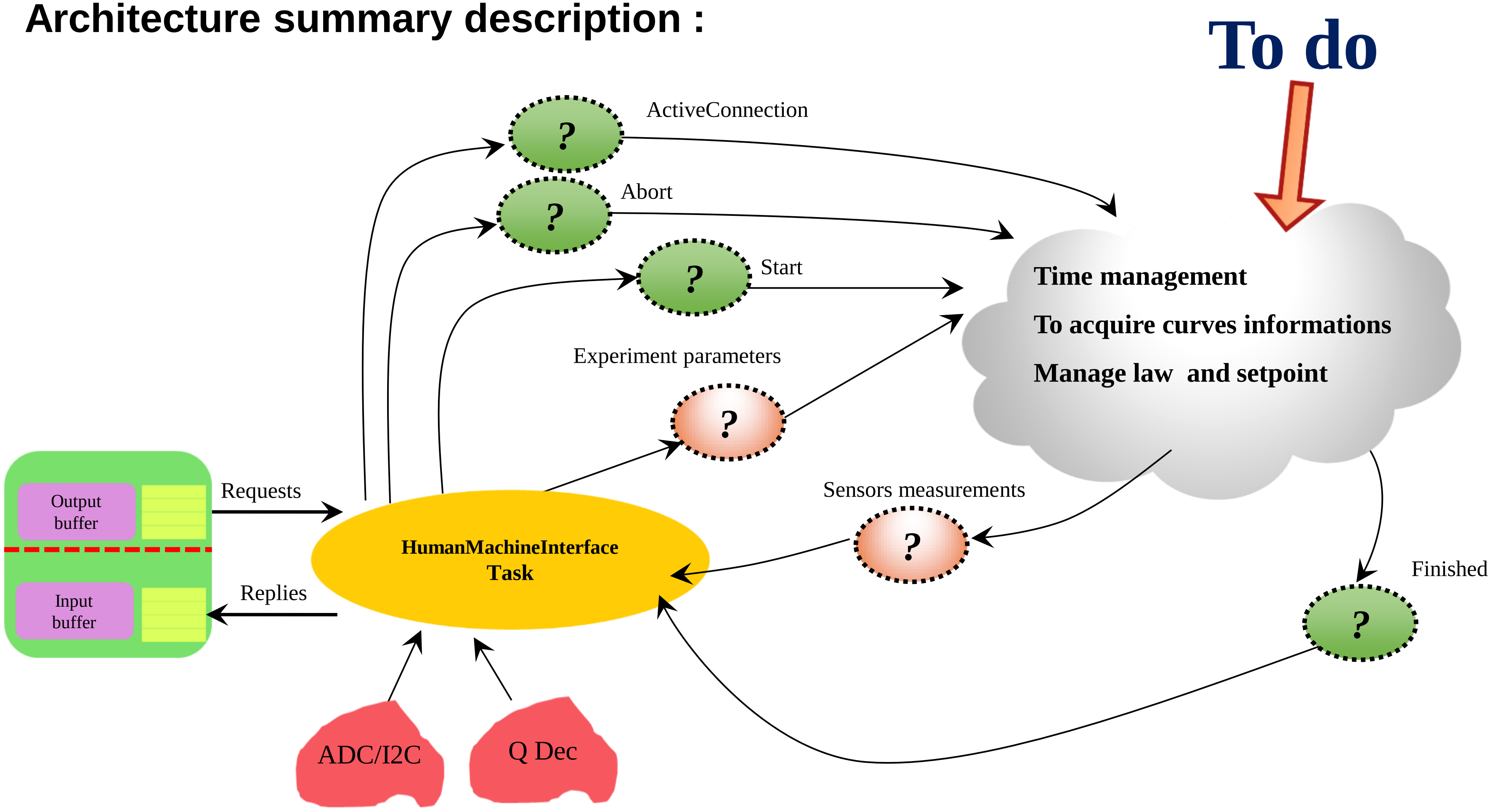
- The declaration and the creation of synchronization elements and tasks communication.
- Initialization of variables.
- Creating and starting the tasks of the application.

- Pending on exitApplication semaphore.

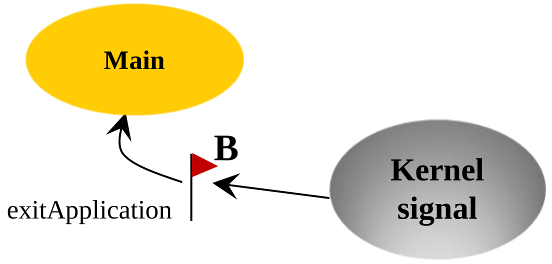
- Stop and destroy the applications tasks.
- The destructions of synchronization elements and tasks communication.

Architecture

Architecture summary description :



Main program :



HumanMachineInterface_Task implementation

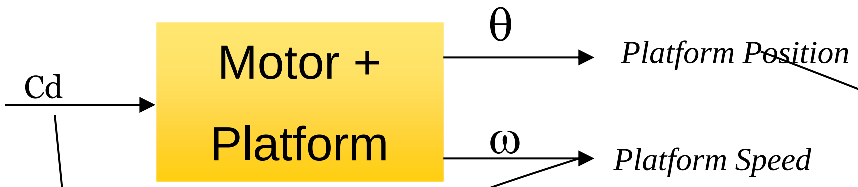
In file *wheelTasks.c*

```
void HumanMachineInterface_Task()
{
   rt_printf("Starting Human/Machine
                  Interface task\r\n");
   ManageRequest();
}

void StartExperiment()
{
 /* Set the Start event here */
}

void AbortExperiment()
{
 /* Set the Abort event here */
}

void ReturnSensorsMeasurement()
{
 (**modify the given answer **)
}
```

In file *wheeHMI.c*

```
void ManageRequest(void) {
        while(ActiveConnection_Flag == true) {
        WaitForMessage(); // Blocking call
        request  = ReturnRequest();
        datatype = ReturnDataType();
        switch(request) {
                case 'L': // Set command laws parameters
                if(datatype == REAL_ARRAY_TYPE) {
                        UpdateExperimentParameters(realsArray);
                        WriteResponse('A');}          break;
                case 'D' :
                        StartExperiment();            // Start experiment
                WriteResponse('A');           break;
                case 'A':
                        AbortExperiment();            // Abort experiment
                        WriteResponse('A') ;          break;
                case 'C':
                        SendPresentSensorsValues();     // Read present
sensors values
        break;
                case 'B':
                        ReturnSensorsMeasurement();     // Read last samples
block
        break;
    }
  }
}
```

Sensors/ Actuator Implementation

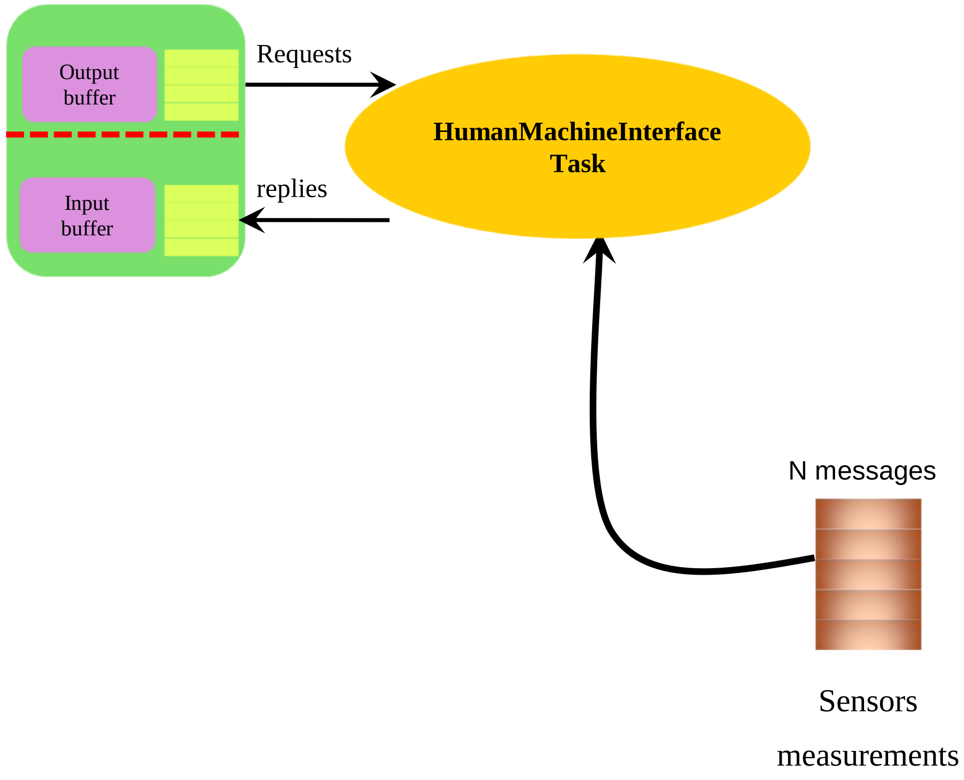


HardwareLib.h

```
void sampleAcquisition(SampleType *sample);
typedef struct {
    float motorSpeed;        // Unit : radian/s
    float platformSpeed;     // Unit : radian/s
    float platformPosition;  // Unit : radian
    float motorCurrent;      // Unit : ampere
   } SampleType;

void applySetpointCurrent(float current)
```

Functions to modify

```
void ReturnSensorsMeasurement) {
            float samplesBlock[50 * 4];
            char terminatorChar;
            int i;

            //   by default, returns 10 constant samples
            //  **********  Must be replaced   ************
            for (i = 0; i < 10; i++)  {
                samplesBlock[(i * 4)] = 0.1;
                samplesBlock[(i * 4) + 1] = 0.2;
                samplesBlock[(i * 4) + 2] = 0.3;
                samplesBlock[(i * 4) + 3] = 0.4;
    }
            /* 'S' or 'F'  */
    terminatorChar = 'S'
    // terminatorChar = 'F'  if the experiment is finished
    writeRealArray(terminatorChar,samplesBlock, 10 * 4);
}
```



InitializeExperiment function

InitializeExperiment :

Declared in file wheelLaws :

```
void initializeExperiment(SampleType sample);
```

The function **sampleAcquisition** must be called before **initializeExperiment**

Example :

```
SampleType sample;      // Declaration

SampleAcquisition   (&sample);    // gets a sample before calling the function
InitializeExperiment(sample);
```

ComputeLaw function

InitializeExperiment : Must be called to compute a new command (lawPeriod)
Returns the command in Ampere

Declared in file wheelLaws :

```
float ComputeLaw(SampleType sample);
```

The function **SampleAcquisition** must be called befrore **ComputeLaw**

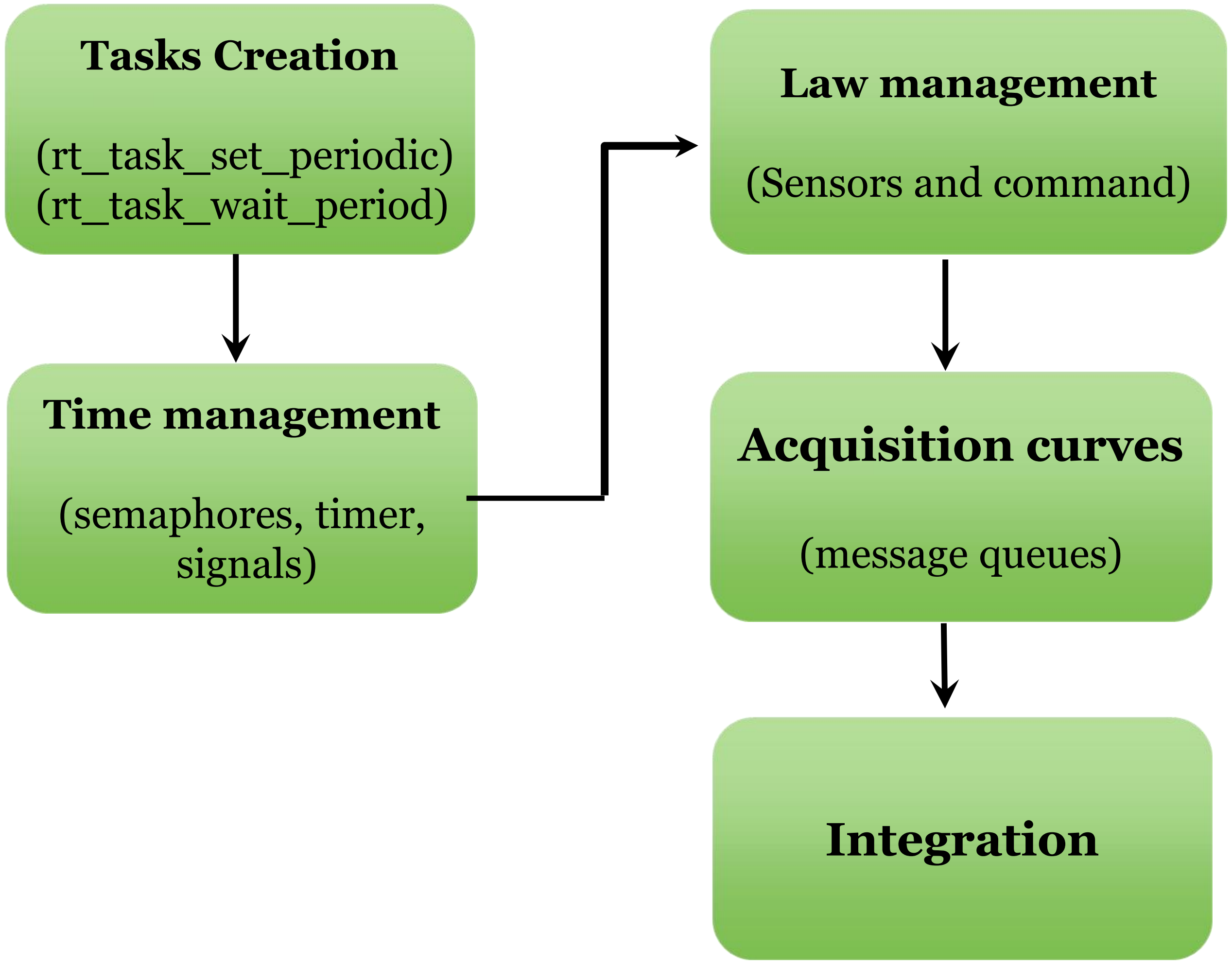
To send a command to the motor, the function **ApplySetpointCurrent** must be called after **ComputeLaw**

Example :

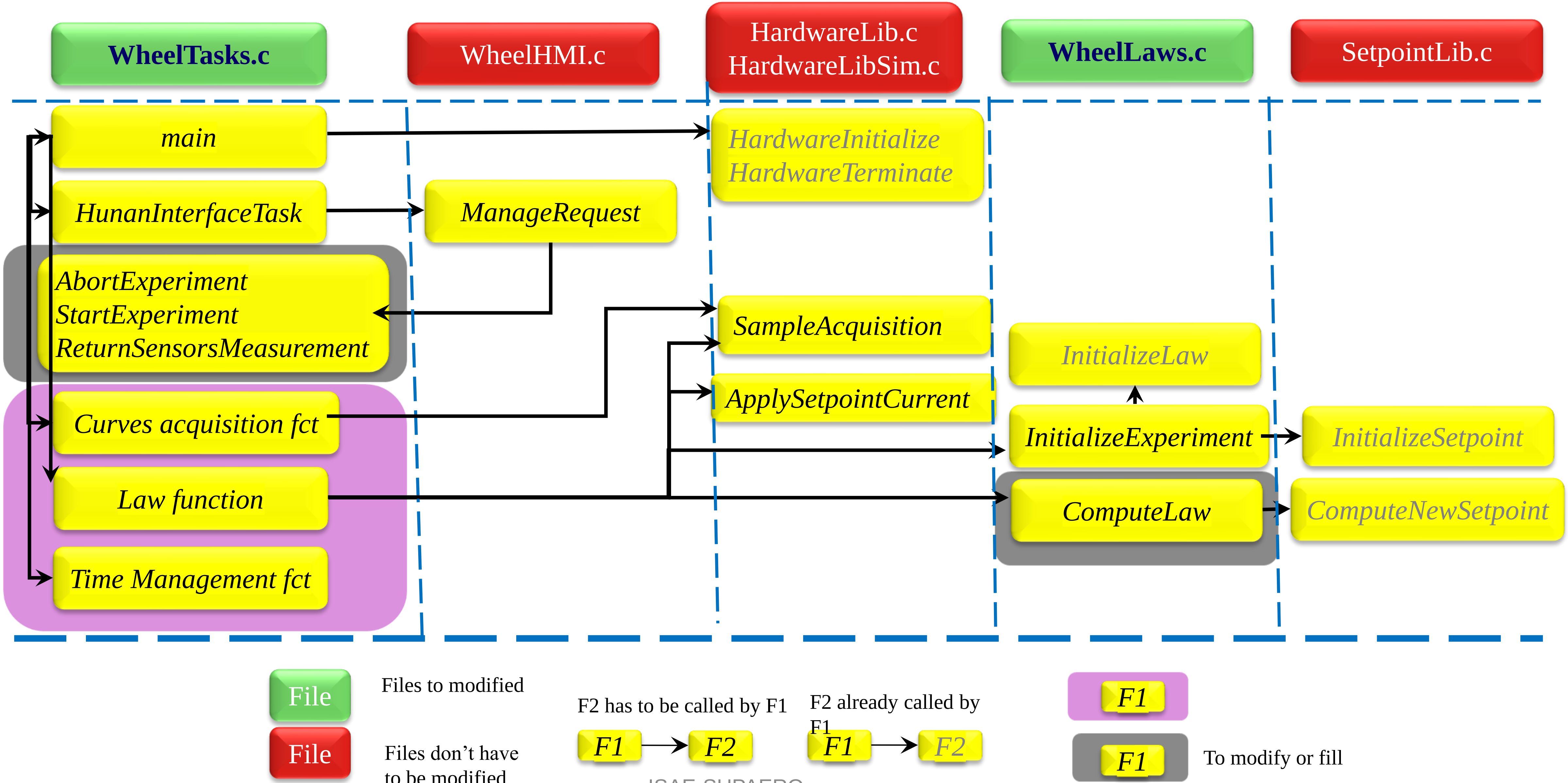
```
SampleType sample;      // Declaration
float command;

SampleAcquisition   (&sample);    // gets a sample before calling the function
Command = ComputedLaw(sample);
ApplySetpointCurrent(Command);
```


50

Steps to implement the application



Software application description



Institut Supérieur de l’Aéronautique et de l’Espace

10, avenue Edouard Belin – BP 54032
31055 Toulouse Cedex 4 – France
T +33 5 61 33 80 80

www.isae-supaero.fr

Institut Supérieur de l’Aéronautique et de l’Espace
SUPAERO