REAL TIME APPLICATION PROGRAMMING for

Digital control of a platform by a satellite's reaction wheel

Date: 15/01/2016

Developed by:

LéoSerre

Julien Jollés

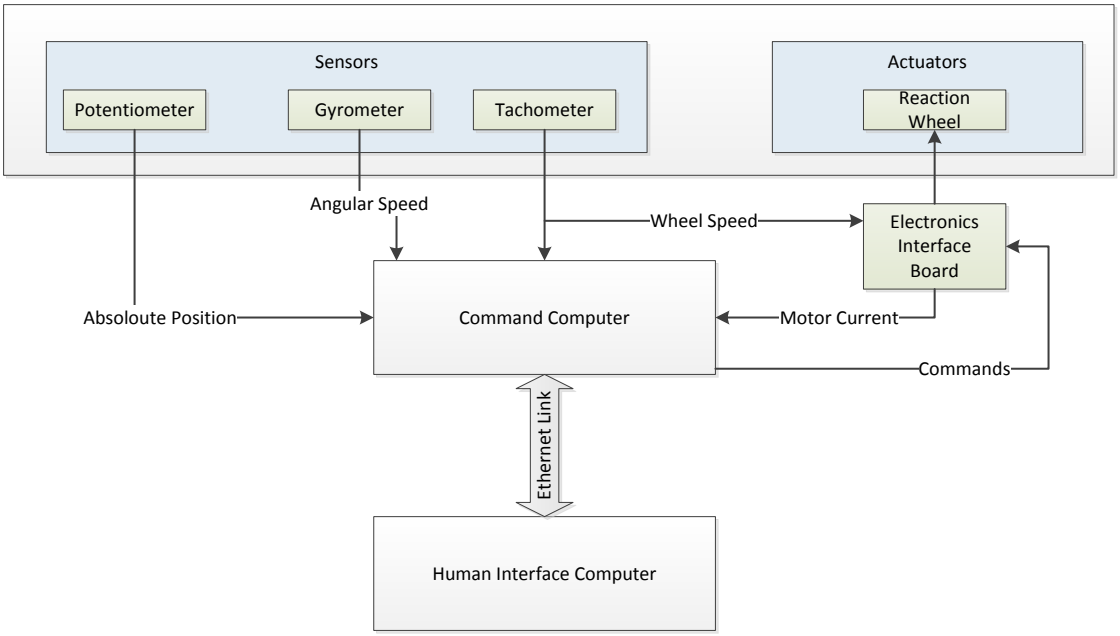
Sandeep Gopala Krishnan

PreranaShamsundar Punjabi

Table of Contents

EXECUTIVE SUMMARY

The Aim of the assignment is to develop the control system of a platform simulating one of the satellite's axes. The components of the platform include: a moving platform around the axes, a reaction wheel driven by a DC motor, sensors (1 platform angular speed sensor, 1 motor speed sensor, 1 absolute angular position sensor of the platform, 1 sensor measuring the motor current), an embedded computer to execute command laws and an interface human computer to set the control law types, parameters and display the curves. The platform is as shown in the figure below



The Human Interface computer executes the Wheel Interface Application, which provides an interface to the operator to control the reaction wheel. The Wheel Interface Application enables the operator to start and stop experiments, configure experiments, perform command law selection, Set point type selection, selection of control law and test parameter. The Wheel Interface Application also provides a graphical user interface to view the sensor parameters (Motor Speed, Motor Current, Platform Speed, Platform Position) in a graphical format. The Wheel Interface Application is already developed at ISAE.

The Wheel Interface Application interfaces with the Command Computer, via Ethernet. The Command Computer implements the software and hardware capabilities to process the commands received from the Wheel Interface Application read sensor data and execute the control laws to control the platform position. The Command Computer executes the real time software which would be developed as a part of this assignment.

The design process for the real time software development for the platform control is divided into 2 distinct phases. They are

1. Design and Development of Software on Xenomai RTOS.
2. Design of Control Algorithms in MATLAB.

Once the algorithms are developed, simulated in MATLAB, they need to be integrated into the real time software.

1 REQUIREMENTS

The software needs to perform the following activities:

1. To accurately control the platform movement over one axes
2. To be able to speedup and speed down the reaction wheel that is driven by a DC Motor
3. Read and provide response to commands/calibration parameters received on the Ethernet interface (sent by Wheel Interface)
4. Read Sensor Data from the hardware platform
5. Process the commands, sensor data and execute control laws
6. Ensure the timing requirements are strictly adhered to
7. Ensure that memory and hardware resources are used efficiently

2 SPECIFICATION

The following temporal constraints must be respected by the software:

1. The delay between reception of a request and response must be 100 ms
2. The activation law period is an integer number(in ms) between 10 ms and 500 ms
3. The sensor reading period is an integer number(in ms) between 2 ms and 200 ms
4. The experiment duration is an integer number (in ms). It must be respected with a precision equal to the law activation period
5. Meet the task worst case execution time criteria specified in the specifications.

3 DESIGN

3.1 TASK DEFNITION

To implement efficient software, we divide the software into 4 tasks. The primary consideration for this division is to ensure that related activities are grouped into the same task. Each task performs a distinct set of activities. This is more efficient as the tasks are simple and independent.

To manage the interface with the WheelInterface, the Dialog(tDialog) task is defined. The tDialog task is responsible for processing the commands received on the Ethernet, decoding the commands and generating the response for the commands. The tDialog task will send the command related information on the other tasks for processing.

The Schedular(tSchedular) task manages the scheduling of the tasks. The task is responsible for scheduling tAcquire and tActuator tasks. The tSchedular tasks also implements the logic to ensure that the timing for the tAcquire and tActuator task can be calibrated(as per the calibration parameter received from WheelInterface).

The Actuator(tActuator) task is responsible for implementing the control laws and sending commands to the platform hardware.

The Acquire(tAcquire) task is responsible for acquiring the sensor information periodically and providing the same to the tDialog task.

Following is a summary of the task structure of the Real Time application:

TABLE 1: TASK STRUCTURE

Task Name	Purpose	Timing
tScheduler	Schedular Task to manage duration and periods	1 ms
tAcquire	Reads samples from the sensors and convert the values to physical units using the calibration table. Outputs the values to a queue.	10 ms (Configurable using PA parameter)
tActuator	Makes the appropriate calculation to command the wheel using all given parameters from the HMI.	100 ms (Configurable using PL parameter)
tDialog	Makes the appropriate computation and giving the correct answer to the host.	Aperiodic task. Invoked each time a message is received on the bus.

3.1.1 SYNCHORNIZATION NEEDS:

Since there are multiple tasks and some shared resources, there is a need to communicate between the various tasks and also to protect the shared resources from multiple accesses.

To ensure communication and protection of resources between the various tasks the following operating system facilities are used by the software.

1. Binary Semaphores
2. Mutex
3. Queues

The resource sharing strategy is illustrated in the figure below:

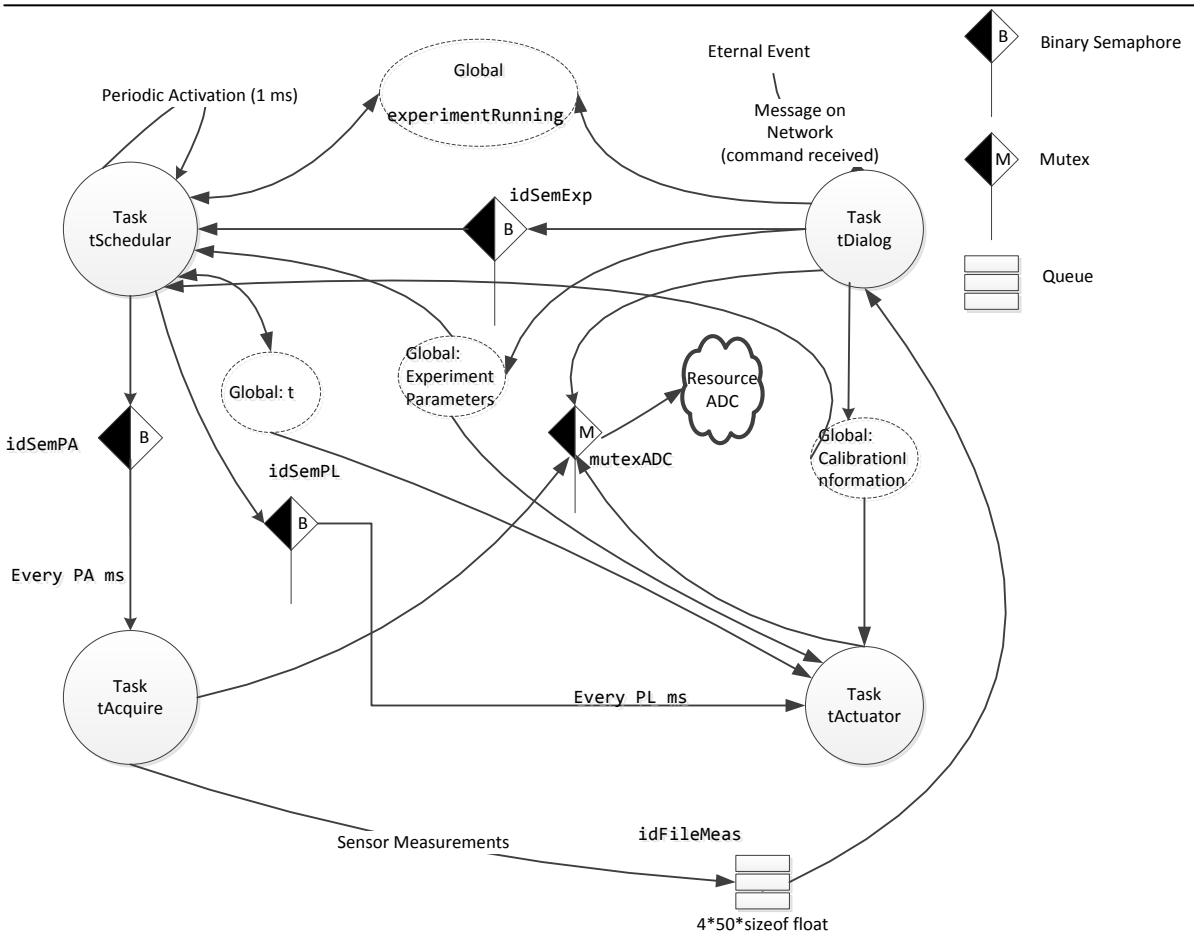


FIGURE 1: RESOURCE SHARING

TABLE 2: COMMUNICATION MECHANISIM BETWEEN TASKS

Data Flow	Synchronization need	Data queue	Asynchronous R/W	Communication tool
Experiment parameter	No	No	No	Global variable
Sensor measurements	No	Yes (50)	Yes	Message queue
t -(global variable to record experiment duration)	No	No	No	Global variable
Experiment execution status	No	No	No	Global variable
Calibration Data	No	No	No	Global variable

3.1.1.1 CRITICAL SHARED RESOURCES

The following resource needs to be considered for protection from multiple accesses:

1. ADC
2. Global variables

The ADC is a hardware resource and is accessed by thetAcquire, tActuator andtDialog tasks. The tasks run independently of each other and access the ADC to read sensor values. Hence it is required to synchronize the access to the ADC to ensure consistency of the values read and to ensure that there is no deadlock. We use a Mutex to protect the ADC. The Mutex is provided by the XenomaiRTOS. Each task which needs access to the ADC invokes the rt_mutex_acquire system call. Once the task has finished accessing the resource(ADC), it releases the mutex so that it is available to the other tasks.

When the task1 requests for a Mutex to access a resource (ADC) and the resource is being used by other task then the RTOS preempts the tasks1 till the time the resource is available(or till timeout specified with the Mutex call is reached). The task1 is scheduled for execution once the resource is available.

In our case the global variables are used to communicate values between tasks.

1. Experiment Running: Boolean value shared between tScheduler and tDialog. Used to indicate if an experiment has been commanded on or not. The value is set and reset by the tDialog task on receipt of a command from WheelInterface. The tScheduler uses the value for operations and resets the value
2. t: used to maintain the timing information. The value is updated by the tScheduler. The tActuatortask only reads the value. Hence no protection is required
3. ExperimentParameters: The parameters are received by the tDialog task. This data is used by the tSchedular and the tActuator task. The parameter is updated prior to the start of the experiment. Hence the resource need not be protected against shared access.
4. CalibrationParameters: The parameters are received by the tDialog task. This data is used by the tSchedular and the tAcquire task. Since the calibration data is updated only during the initialization the data need not be protected against shared access.

3.1.1.2 SYNCHRONIZATION AND COMMUNICATION BETWEEN TASKS

The tasks are required to communicate with each other for synchronization and to notify the occurrence of certain events. The tScheduler indicates to the tAcquire and tActuator task when they are ready to be scheduled. Similarly the tDialog task invokes the tSchedular task when an experiment is started so that it can start the scheduling of other tasks.

Binary semaphores are used for the following inter-task communication/sysncronization:

1. tScheduler indicates to the tAcquire that PA time has elapsed and it is ready to be scheduled for execution.
2. tScheduler indicates to the tActuator that PL time has elapsed and it is ready to be scheduled for execution.
3. tDialog indicates to the tScheduler that the experiment is commanded on and it can start the scheduling of other tasks.

Semaphores are used for inter-task communication and synchronization. A task(say tAcquire) will wait on a semaphore for the occurrence of an event. When the task (tAcquire) is ready to be scheduled, the task tSchedular will post the semaphore. Since the task tAcquire is waiting on the semaphore the RTOS will ready the task for execution.

The tAcquire task is responsible to collect the sensor data information periodically. The tAcquire send the information to the tDialog tasks which then passes over the information to the WheelInterface via the Ethernet. Since the data is accumulated at a rate higher than the rate at which the tDialog processes the data, the tAcquiretasks write the data into the Queue. The Queue, provided by Xenomai, stores the data is the First In First Out format. The tDialogtasks read the data from the Queue, processes the information and sends it over the network.

The tasks sequence execution is as described in the figure below:

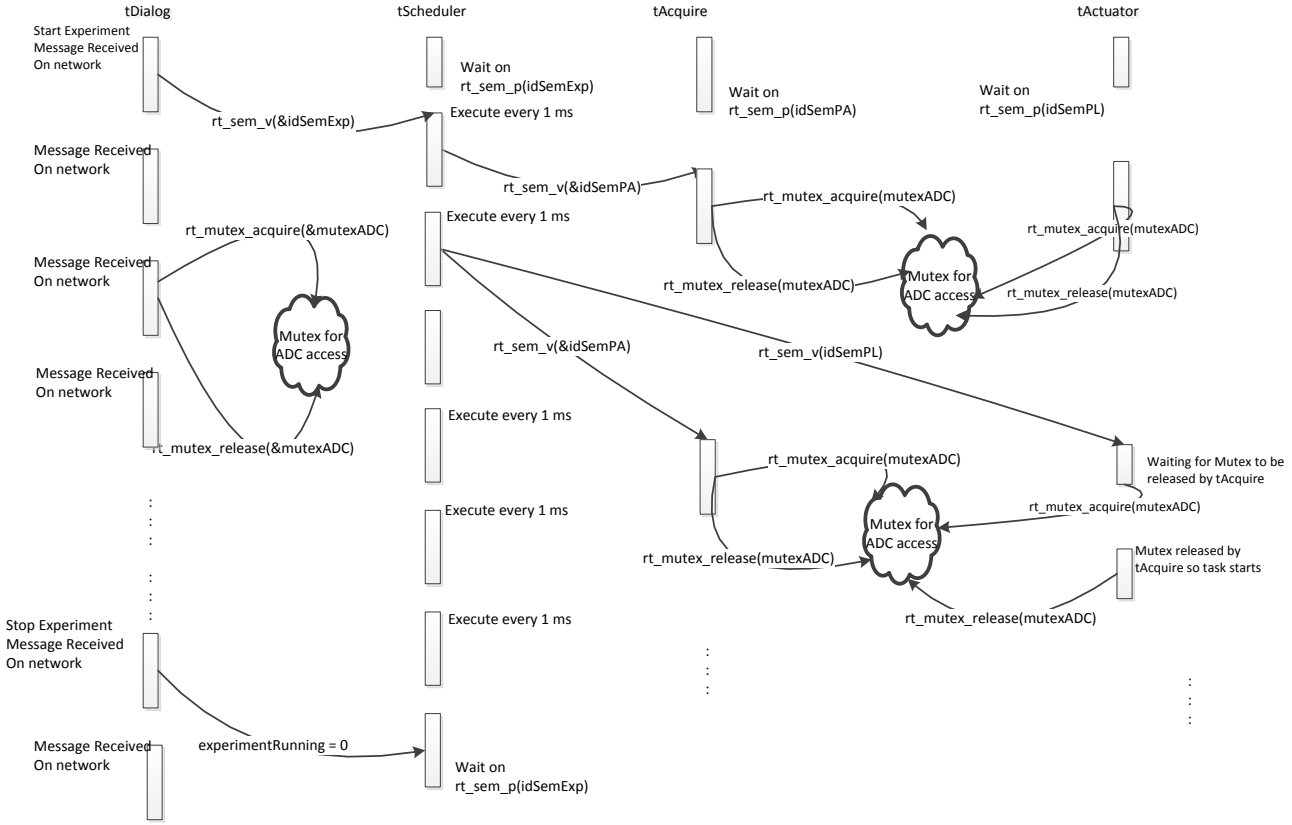


FIGURE2: TASK EXECUTION SEQUENCE

3.1.2 ARCHITECTURE SOFTWARE DIAGRAM

The software architecture diagram is as shown below:

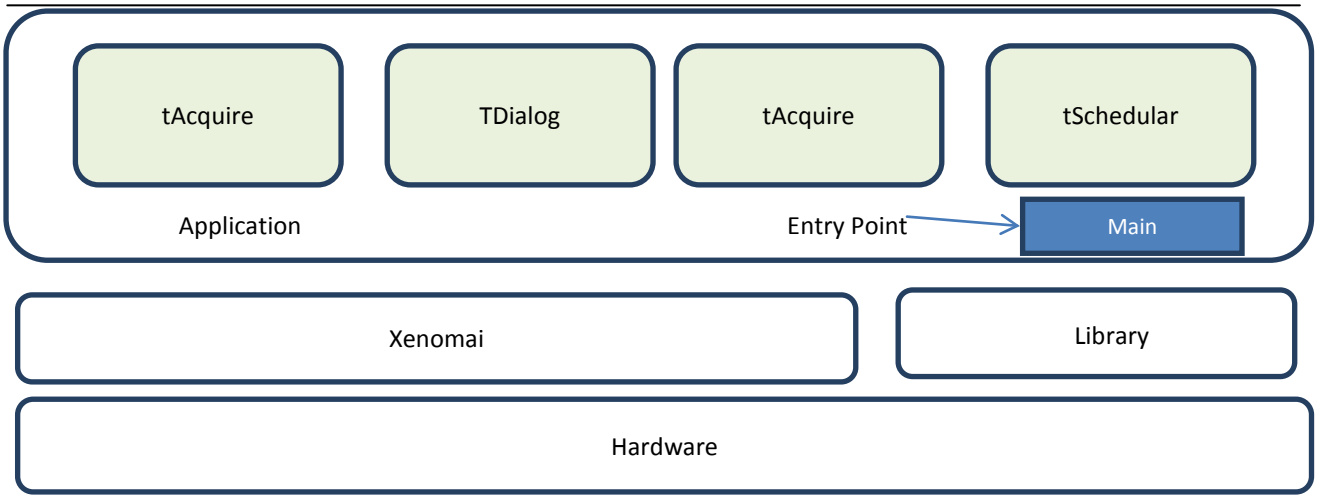


FIGURE3: ARCHITECTURE DIAGRAM

The System uses Xenomai as the RTOS. Xenomai provides the scheduler, task manager, mutex, semaphores and queues to support the execution of the application.

The libraries provide the low level interface to the application layer to access the hardware resources.

The **Main()**function is the entry point for the application execution.

The **Main()**function performs the following activities:

1. Initializes signals to stop program with CTL+C
2. Lock memory pagination for the application
3. Initialize the hardware and drivers
4. Create Queue (idFileMeas)
5. Create Semaphores (idSemPA, idSemPL, idSemExp)
6. CreateMutex (mutexADC)
7. Invoke the pause system call to suspend execution
8. Delete the tasks, Mutexs, Queues and Semaphores

The tasks are detailed in the following sections.

3.1.3 TASKPRIORITY

TABLE 3: TASK PRIORITY

Task Name	Priority
tScheduler	4
tAcquire	3*
tActuator	3*
tDialog	5

The tDialog task is aperiodic and has hence being given the highest of 5. The tDialog task waits for a message (recv command) on the Ethernet bus. When a message is received it will process the message with the highest priority.

The tSchedular has a priority of 4. The task is responsible to schedule the processing of the other tasks. It has a short activation period and hence it has being assigned higher priority (than tAcquire and tActuator) as per the principles of rate montonic scheduling (short period->higher priority).

The tAcquire and tActuator have been assigned the priority of 3. Hence these tasks have the same priority. These tasks perform data/algorithm processing which takes significant amount of processing time. There periods are longer compared to the tSchedular. Hence these tasks are assigned lower priority. The tasks with the same priority are executed in a round robin manner by Xenomai. Hence, in case of simultaneous activation, thetAcquire task will given higher priority over the tActuator task (as tAcquire is created earlier to tActuator).

3.1.4 HIGH LEVEL ALGORITHM

The algorithm for implementing tScheduler is as follows:

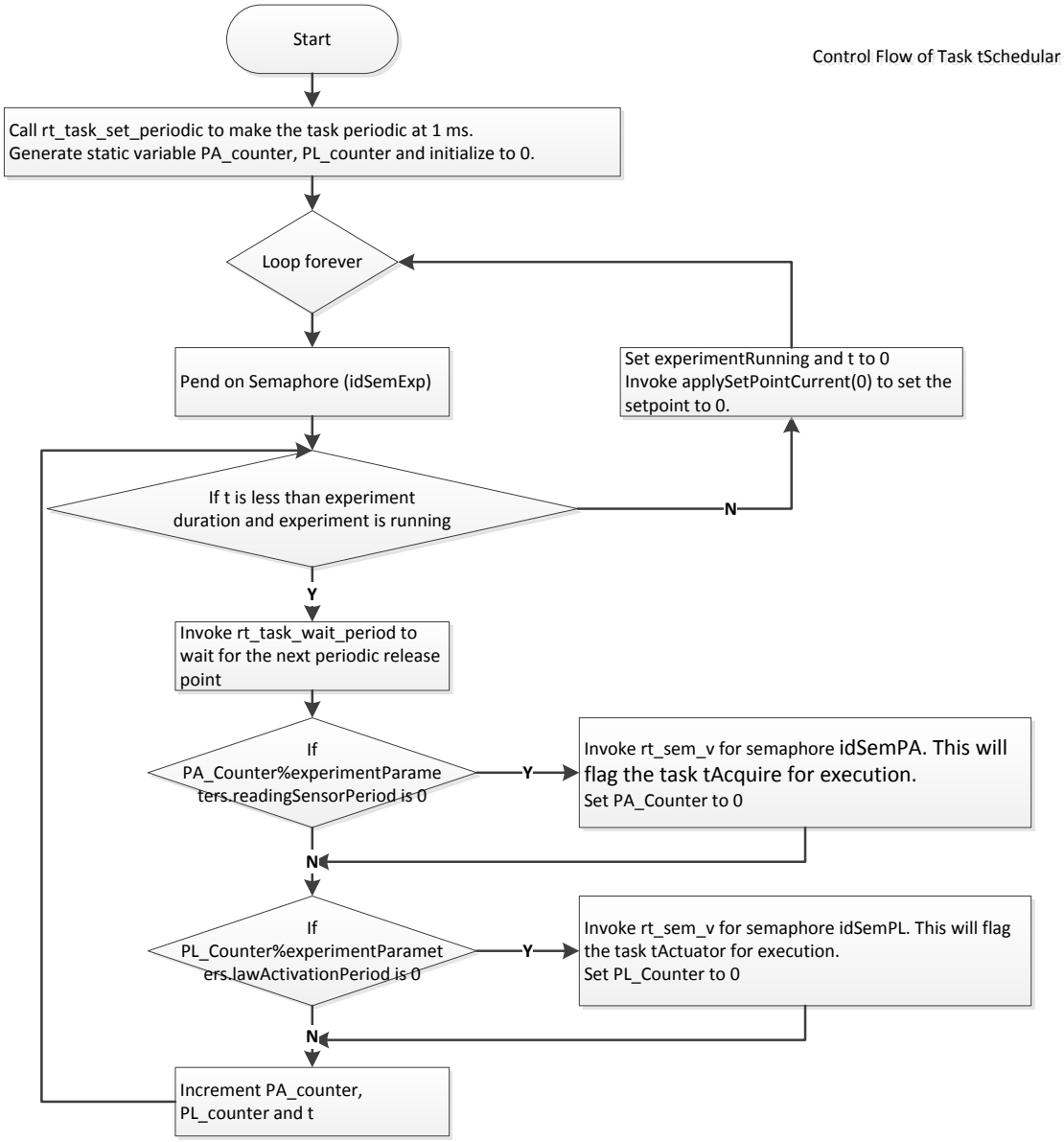


FIGURE4: FLOW CHART FOR - TASK 'tScheduler'

The algorithm for implementing tAcquire is as follows:

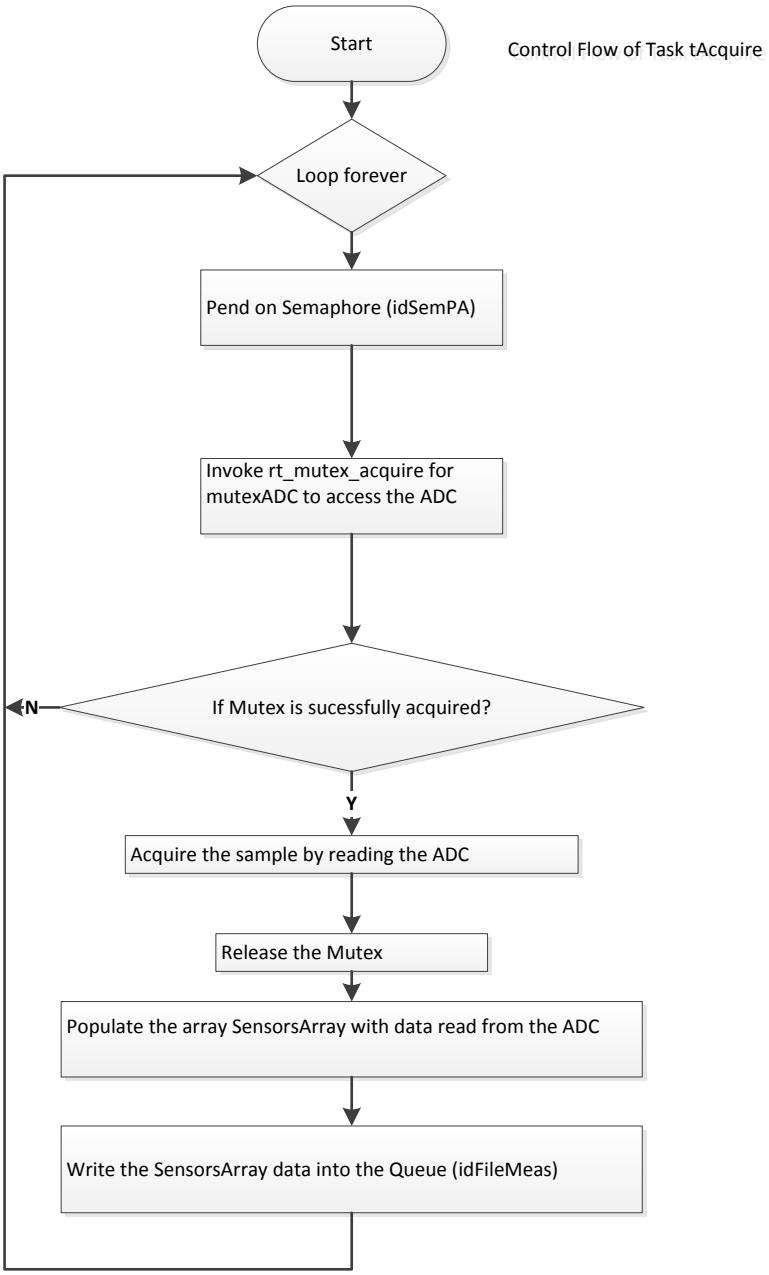


FIGURE5: FLOW CHART FOR - TASK 'tAcquire'

The algorithm for implementing tActuator is as follows:

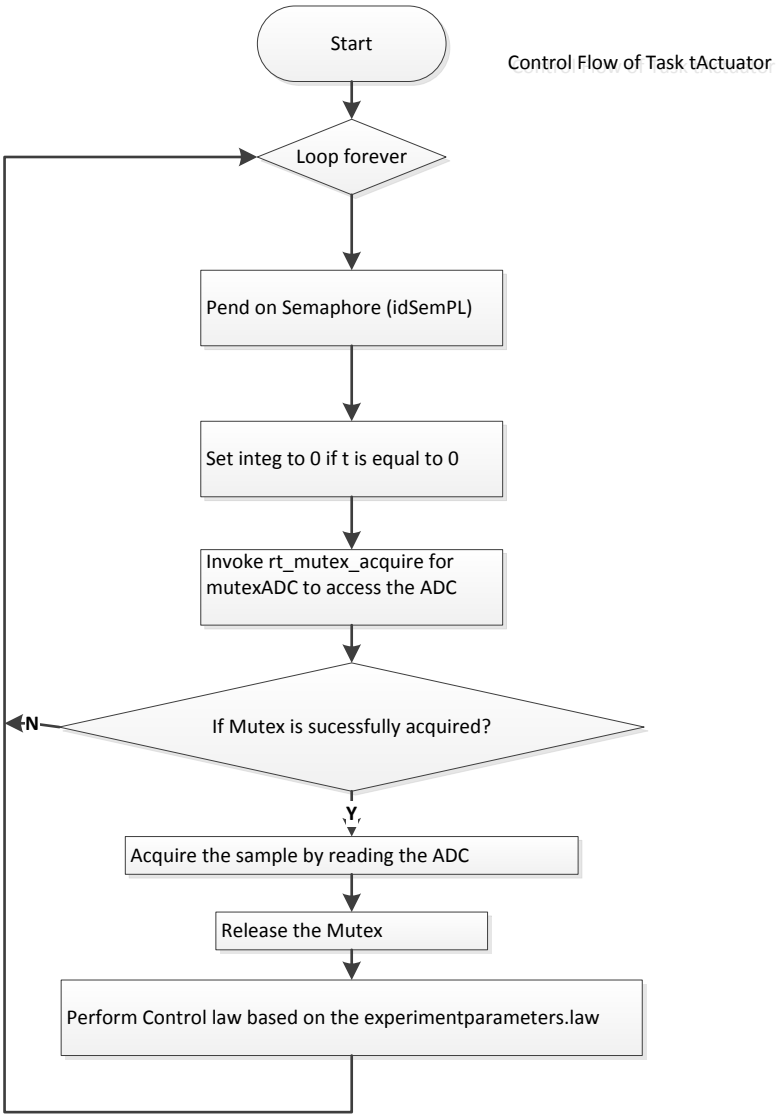


FIGURE6: FLOW CHART FOR - TASK 'tActuator'

The algorithm for implementing tDialog is as follows:

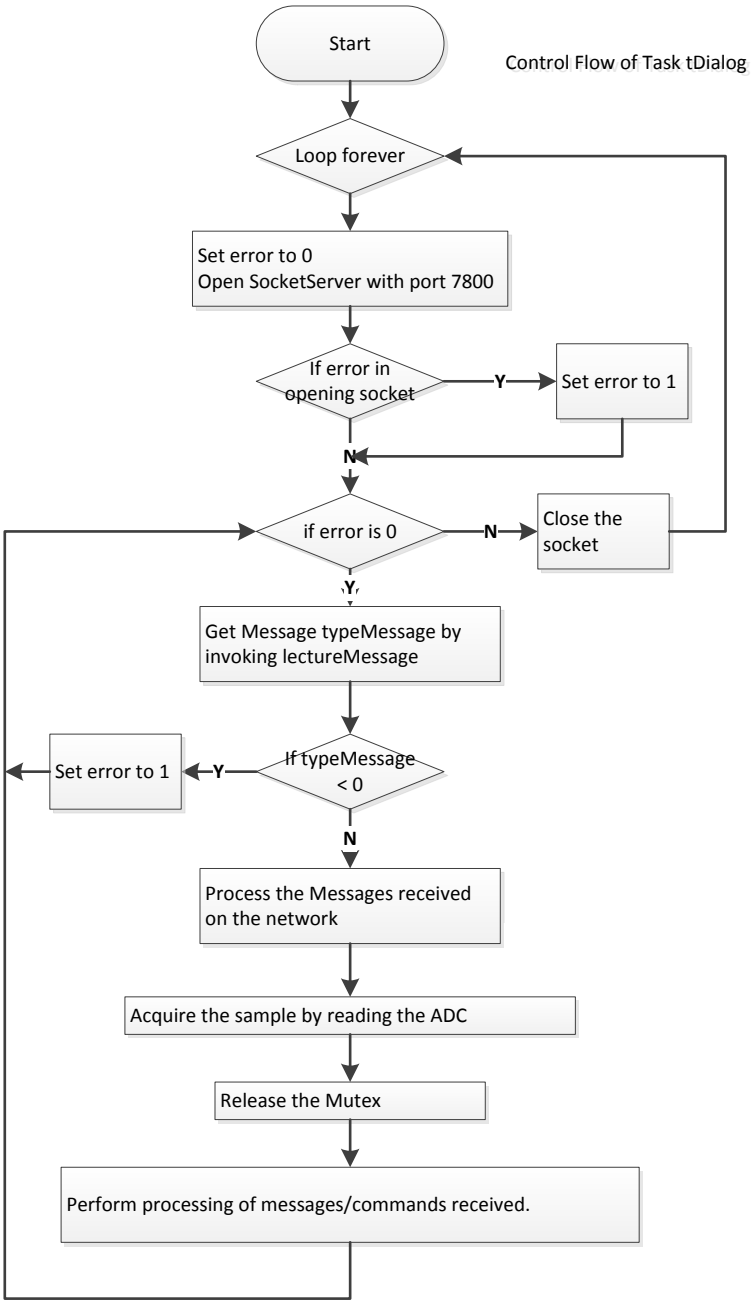


FIGURE7: FLOW CHART FOR - TASK 'tDialog'

3.2 APPLICATION DEVELOPMENT – ENCODING

The following files are used in the development of the software.

File Name	Comment
libmm32.c/.h	Provides library routine to manage low level hardware access
socketsTCP.c/.h	Provides services to accesses the network using TCP/IP
wheelLibUtil.c/.h	Provides function to access the sensor data from ADC
wheelTasks.c	Provides the task definition and functional logic for the software. This is the only file updated as a part of the assignment.

3.2.1 LISTING

The following functions are used for the development of the software (in file wheelTasks.c).

TABLE 4: FUNCTIONs IMPLEMENTED

Function Name	Description
`tScheduler`	Scheduler Task to manage duration and periods
`tAcquire`	Reads samples from the sensors and convert the values to physical units using the calibration table. Outputs the values to a queue.
`tActuator`	Makes the appropriate calculation to command the wheel using all given parameters from the HMI.
`tDialog`	Makes the appropriate computation and giving the correct answer to the host.
`main`	Used for initialization.
`catch_signal`	Catches the CTL+C signal.
`sendPresentSensorsValues`	Sends the current sensor values on the network
`sendLastSamplesBlock`	Sends the last collected sensor values on the network.

3.2.2 TEMPORAL ANALYSIS

The worst case computation times for the tasks are presented in the Table below. The task times have been computed by using the system call at the start of the task, system call at the end of the task and computing the difference between the start and end times. From the data we observe that the real time software application meets the timing constraints that are imposed by the system requirements.

TABLE 5: TEMPORAL CONSTRAINTS

Task Name	Expected Execution Time (as expected by requirements) in ms	Actual Execution Time in ms
tScheduler	0.2ms	0.0086 ms
tAcquire	0.3ms	0.1937 ms
tActuator	1ms	0.1975 ms
tDialog	2ms	0. 350 ms

3.3 CONTROL SYSTEMS

In order to accurately control the position, speed and angular movement of the platform we model our system, structure the command laws, implement feedback analysis, simulate the laws, choose the gains and other parameters and analyze the results using feedback control. To do so Models are developed in Matlab and Simulink and a detail analysis is performed on the system.

3.3.1 SYSTEM MODELLING

Our system is composed of 4 different sensors:

- Current of the motor
- Angular speed of the wheel
- Angular speed of the platform
- Angular position of the platform

Final goal is to control the angular position of the platform.

3.3.2 COMPLETE SYSTEM MODEL ANALYSIS

Using Laplace transformations the below equations are obtained for the system:

$$u(t) = Ri(t) + L\frac{di(t)}{dt} + k_e w_w(t) \rightarrow u = Ri + Lpi + k_e\omega_w$$

$$T = k_m i(t) \rightarrow T = k_m i$$

$$T = J_w\frac{d\omega_w(t)}{dt} \rightarrow T = J_w\omega_w p$$

$$T = J_{pl}\frac{d\omega_{pl}(t)}{dt} \rightarrow T = J_{pl}\omega_{pl}p$$

$$\omega_{pl} = \frac{d\theta_{pl}(t)}{dt} \rightarrow \omega_{pl} = \theta_{pl}p$$

The first open loop model is developed using Simulink.

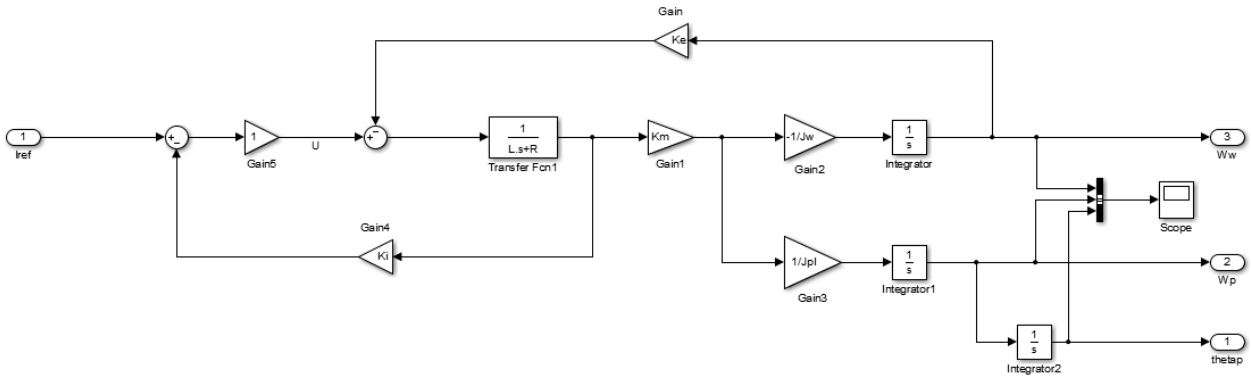


FIGURE8: SIMULINK MODEL – COMPLETE SYSTEM ANALYSIS

TheMatlab code to run the above open loop model is as follows.

```
%% 2.OL system

openloop_linmod = linmod('simu_OL');
openloop_ss                                                                    =
ss(openloop_linmod.a,openloop_linmod.b,openloop_linmod.c,openloop_linmod.d);
[OLnum,                                    OLden]                              =
ss2tf(openloop_ss.a,openloop_ss.b,openloop_ss.c,openloop_ss.d,1);
rlocus(OLnum(1,:),OLden)
```

We can see fromthe root locus graph below that the system mightbe stable in close loop because all the poles are negative and on zero axis. Also we can notice the fast dynamic due to p = 200

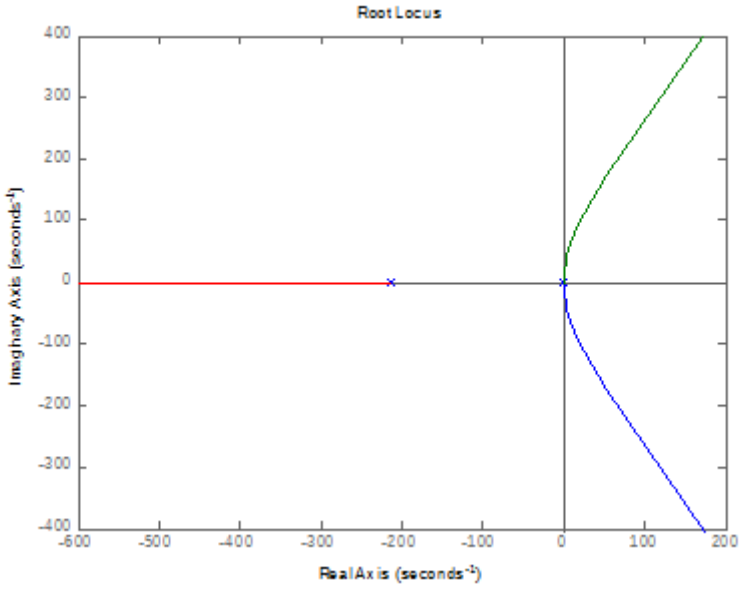


FIGURE9: ROOT LOCUS – COMPLETE SYSTEM ANALYSIS

3.3.3 OPEN LOOP SIMPLIFIED

Further, the above open loop model can be simplified because we do not consider the wheel and the electromechanical part, as it is considered as a disturbance. Additionally as we want to control the angular position of the platform we need not consider the wheel. The new model obtained is as shown below:

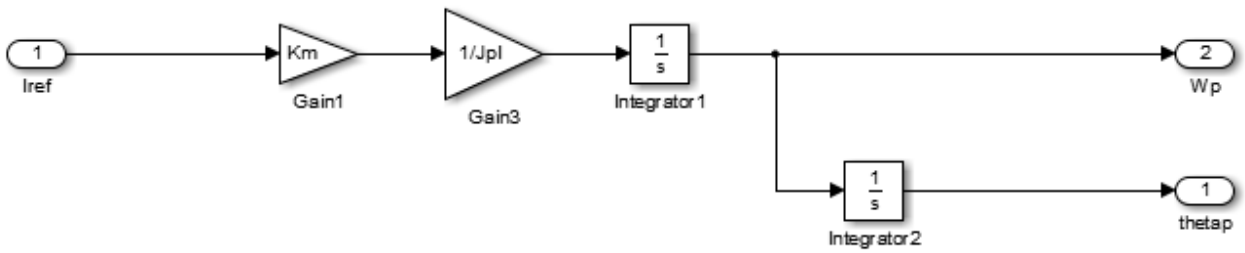


FIGURE10: SIMULINK MODEL – SIMPLIFIED SYSTEM

```
%% 3.OL simplified

openloop_linmod_sOL = linmod('simu_simplified_OL');
openloop_ss_sOL                                                    =
ss(openloop_linmod_sOL.a,openloop_linmod_sOL.b,openloop_linmod_sOL.c,openloop_lin
mod_sOL.d);
[OLnum_sOL,                                OLden_sOL]              =
ss2tf(openloop_ss_sOL.a,openloop_ss_sOL.b,openloop_ss_sOL.c,openloop_ss_sOL.d,1);
rlocus(OLnum_sOL(1,:),OLden_sOL)
```

The root locus plot for our simplified model shows the pole at zero axis.

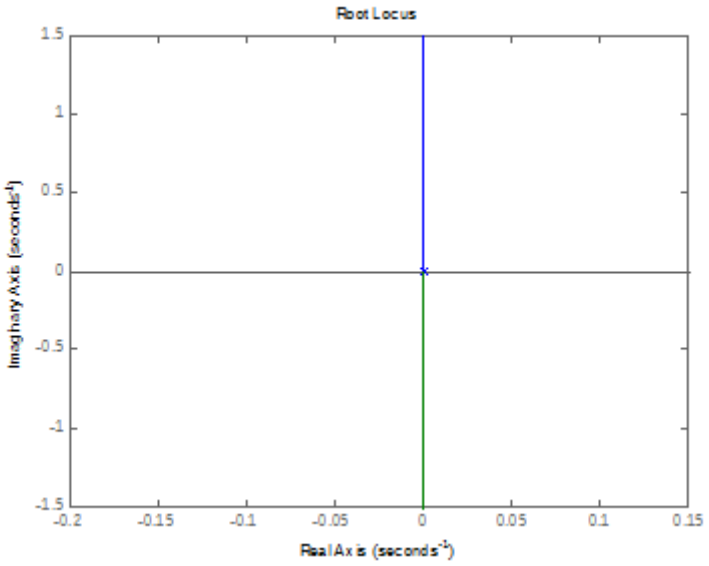


FIGURE11: ROOT LOCUS – SIMPLIFIED SYSTEM

3.3.4 MODAL APPROCH

Now we will try to determine the matrices An, Bn, Cn and Dn that will be used for the gains computation. (computation of the state space model with position, speed and current as state variable is done as shown below).

```
Cn = eye(2); % Identity matrix
M = inv(sys.c)*Cn;
An = inv(M)*sys.a*M;
Bn = inv(M)*sys.b;
Dn = sys.d;
```

3.3.5 STATE FEEDBACK

To describe the state feedback, we will use the following model in closed loop. We added a gain to correct the values of speed and position.

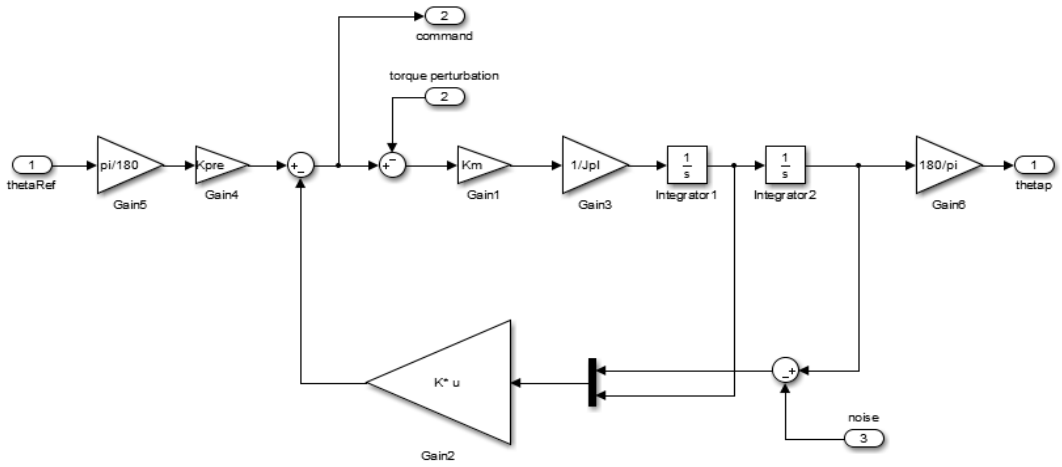


FIGURE12: SIMULINK MODEL – STATE FEEDBACK MODEL

The following code describes the way to compute the K gain using the previous matrices from modal approach. We set delta to 0.7 to be at the limit of overshoot. It also generates the input reference vector.

```
% Computation of the controller gain K
wn = 2.5;
delta = 0.7;
lambda = [roots([1/(wn*wn) 2*delta/wn 1])];
K  = place(An,Bn,lambda);
K = K*inv(Cn);
Kpre=K(1);

% Time
t = 0:0.1:30;
t1 = 0:0.1:14;
t2 = 14.1:0.1:21;
t3 = 21.1:0.1:30;
% Input reference
E = [10*ones(size(t'))
[zeros(size(t1'));0.02*ones(size(t2'));0.02*ones(size(t3'))]
[zeros(size(t1'));zeros(size(t2'));0.005*sin(3*t3')]];
```

The values obtained for K and Kpre (with omega 2.5) are as follows:

```
K =

    3.2979    1.8468

Kpre =

    3.2979
```

We calculate the K gain for 3 different values of omega and we draw the angular position and the command.

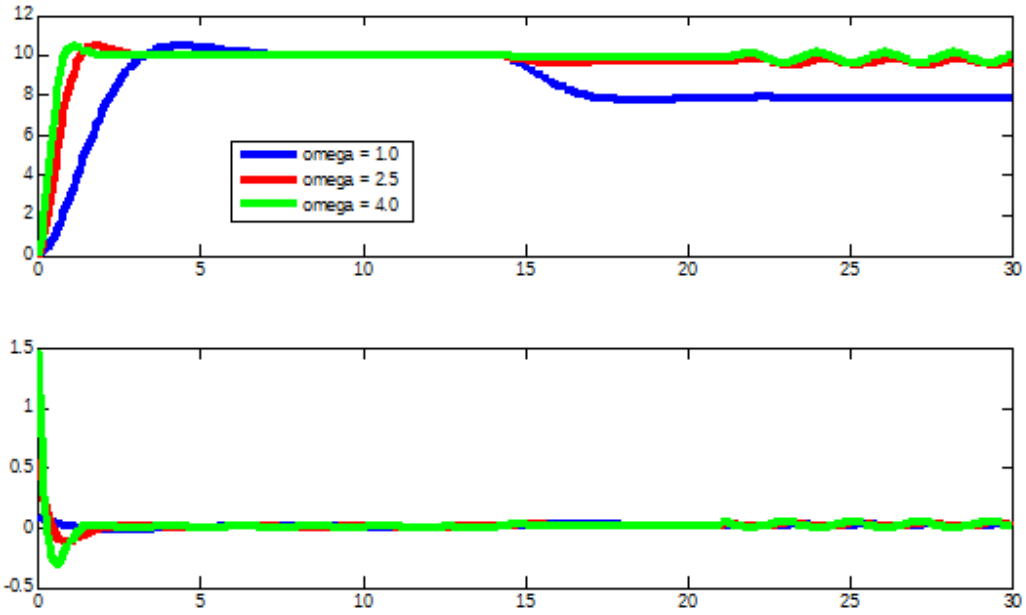


FIGURE13: PLOT – STATE FEEDBACK SYSTEM

We can clearly notice that the lower the omega the higher noise amplification. But it's inverted for the torque perturbation and we need a low omega. To conclude, we need a kind of compromise between those parameters.

3.3.6 STATE FEEDBACK WITH INTEGRATOR

We now add an integrator in order to correct the noise and keep the amplification. In addition, we decided to work only with omega = 2.5 (because of the compromise). The new close-loop schematic is as follows:

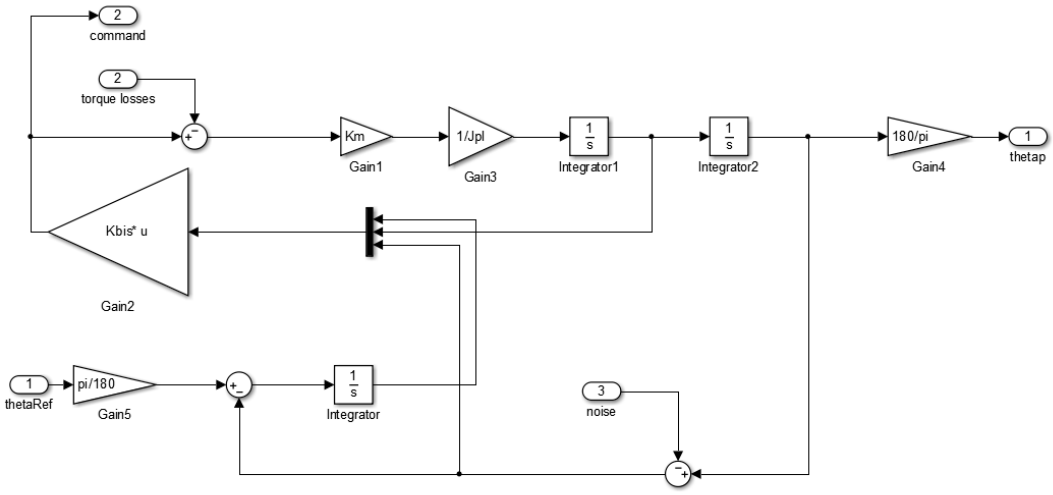


FIGURE14: SIMULINK MODEL – STATE FEEDBACK WITH INTEGRATOR

Now we have 3 output values instead of 2. Thus, we need to re-compute the following code with a new gain Kbis:

```
lambda = [roots([1/(wn*wn) 2*delta/wn 1]); -3];
Kbis  = place(An,Bn,lambda);
Kbis = Kbis*inv(Cn);
```

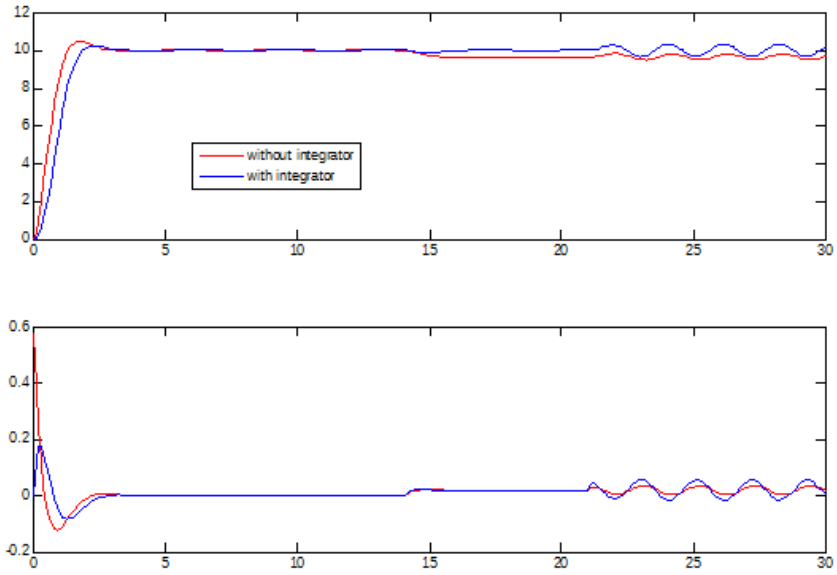


FIGURE15: PLOT – STATE FEEDBACK WITH INTEGRATOR

We can notice that adding the integrator corrected the noise, but it also amplifies the torque perturbation.

This simulation gives us: **Kbis = { 9,89 ; -3,43 ; -8,84 }**

3.3.7 LEAD COMPENSATOR

To build the lead compensator, we need to impose a phase margin of 45 degrees. Then we can calculate the *α* value. We can now use the ltiview tool on the open loop system to determine the cut-off pulsation. And it is possible to compute the Tau value.

The according Simulink model and Matlab code are as follows:

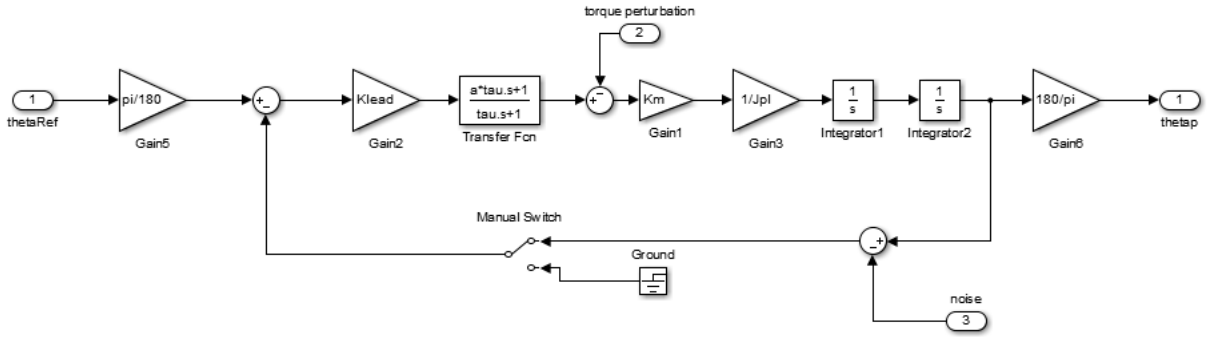


FIGURE16: SIMULINK MODEL – LEAD COMPENSATOR

```
PhaseMargin = pi*45/180;
a = (1+sin(PhaseMargin))/(1-sin(PhaseMargin));
% We search for K=1 on ltiview
Klead = 10^(-7.58/20);
% ltiview gives us wc = 1.38
wc = 1.38;
tau = 1 / (wc * sqrt(a));
```

Note: 0.66 is the input current constraint that is to be taken into account.

```
% 10 is used as it is number of steps.
% 10 * (pi/180) * k* a<=0.66
% k = 0.66/ (10 * (pi/180)*a)
% k = 0.6488
```

while paying attention on the constraints

$$\begin{cases} -10V < \text{Voltage} < 10V \\ -0.66A < \text{Current} < 0.66A \\ \text{reference step of 10 degrees} \end{cases} \rightarrow \textit{Saturation}$$

The Discretization for the Lead Compensator is done as below:

```
lead_compensator_discrete =

  2.147 z - 2.028
  ---------------
    z - 0.7144

Sample time: 0.1 seconds
Discrete-time transfer function.
```

The following figure is a comparison between the KpKv method and the lead compensator.

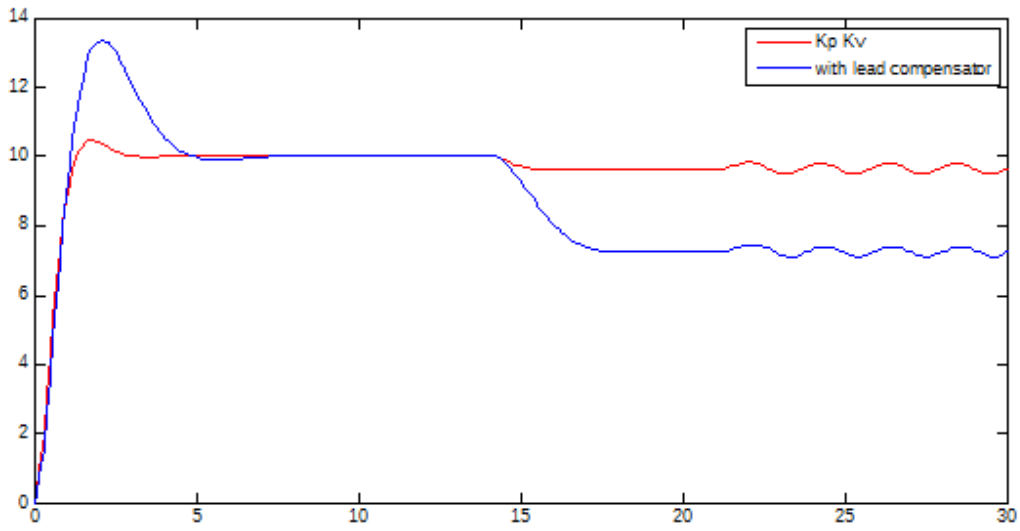


FIGURE17: PLOT – LEAD COMPENSATOR

We see that the system with lead compensator is not robust at all to disturbances.

3.3.8 LAG AND LEAD COMPENSATOR

In order to create our lag compensator, we need to increase the DC gain without changing the phase at the crossover frequency.

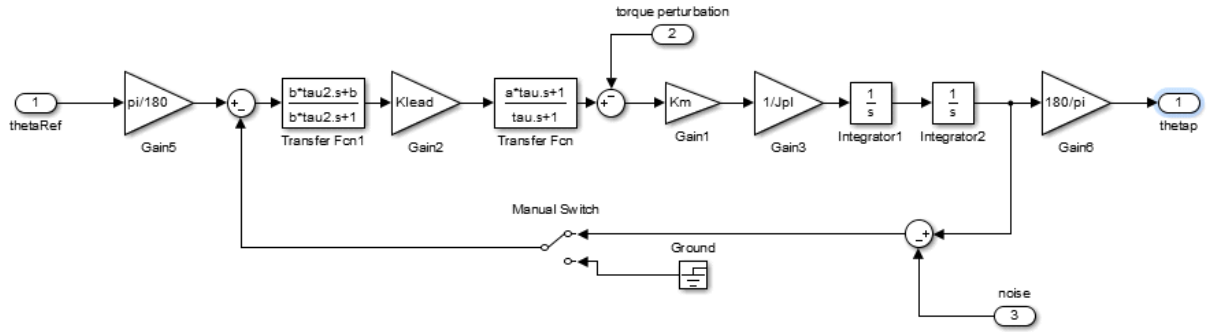


FIGURE18: SIMULINK MODEL – LAG AND LEAD COMPENSATOR

We set b to 2 in order to divide the error by two. Also we choose the value of Tau not to change the phase in the crossover frequency.

```
b=2;
tau2 = 10*(1/wc);
```

The Discretization for the Lead Lag Compensator is done as shown below:

```
lag_compensator_discrete =

  2.155 z^2 - 4.16 z + 2.007
  --------------------------
    z^2 - 1.708 z + 0.7095

Sample time: 0.1 seconds
Discrete-time transfer function.
```

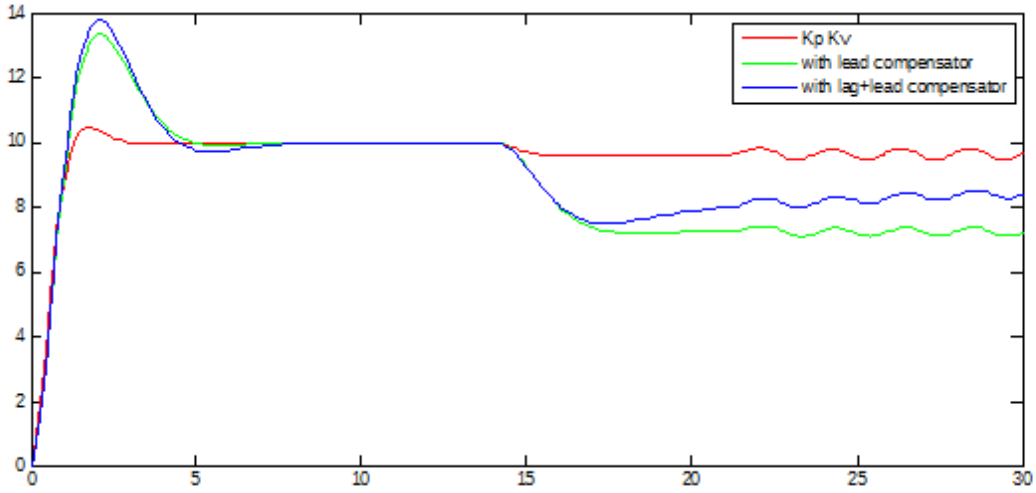


FIGURE19: PLOT – LAG AND LEAD COMPENSATOR

We can see better results once the lag compensator has been added. However the compensation delay is long and we will never remove completely the error.

3.3.9 DELAY MARGIN

We will try to see the robustness while setting a delay as we will have in the real system. In the real time C code, the control law is computed every period. The robustness will give us the maximum period we can set to keep having a stable system.

We need to modify all Simulink models to implement a delay as done in this example:

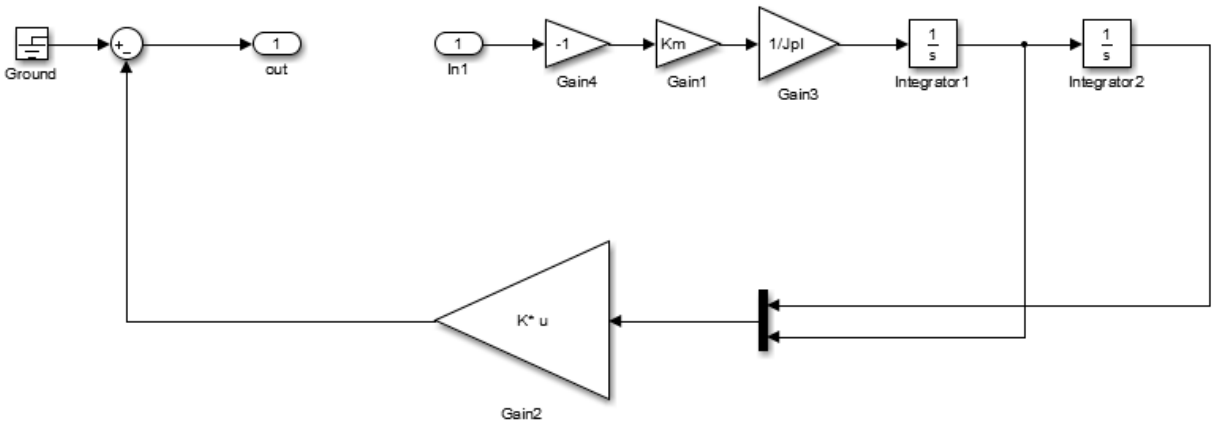


FIGURE20: SIMULINK MODEL – DELAY MARGIN

Then, for all four control laws, we calculate the robustness:

- t ≤ 0.295s for state feedback
- t ≤ 0.178s for state feedback with integrator
- t ≤ 0.567s for lead compensator
- t ≤ 0.530s for lead and lag compensator

We can see that the robustness is better without the compensators.

The software also saturates the current (+/-0.66 A) values to the limits specified in the specification.

The Source code developed and the Matlab files are provided as attachments within the report.

3.3.10 EXPERIMENTAL RESULTS AND ANALYSIS

The results for the actual execution using the Modal approach (with Kp and Kv) is as shown below:

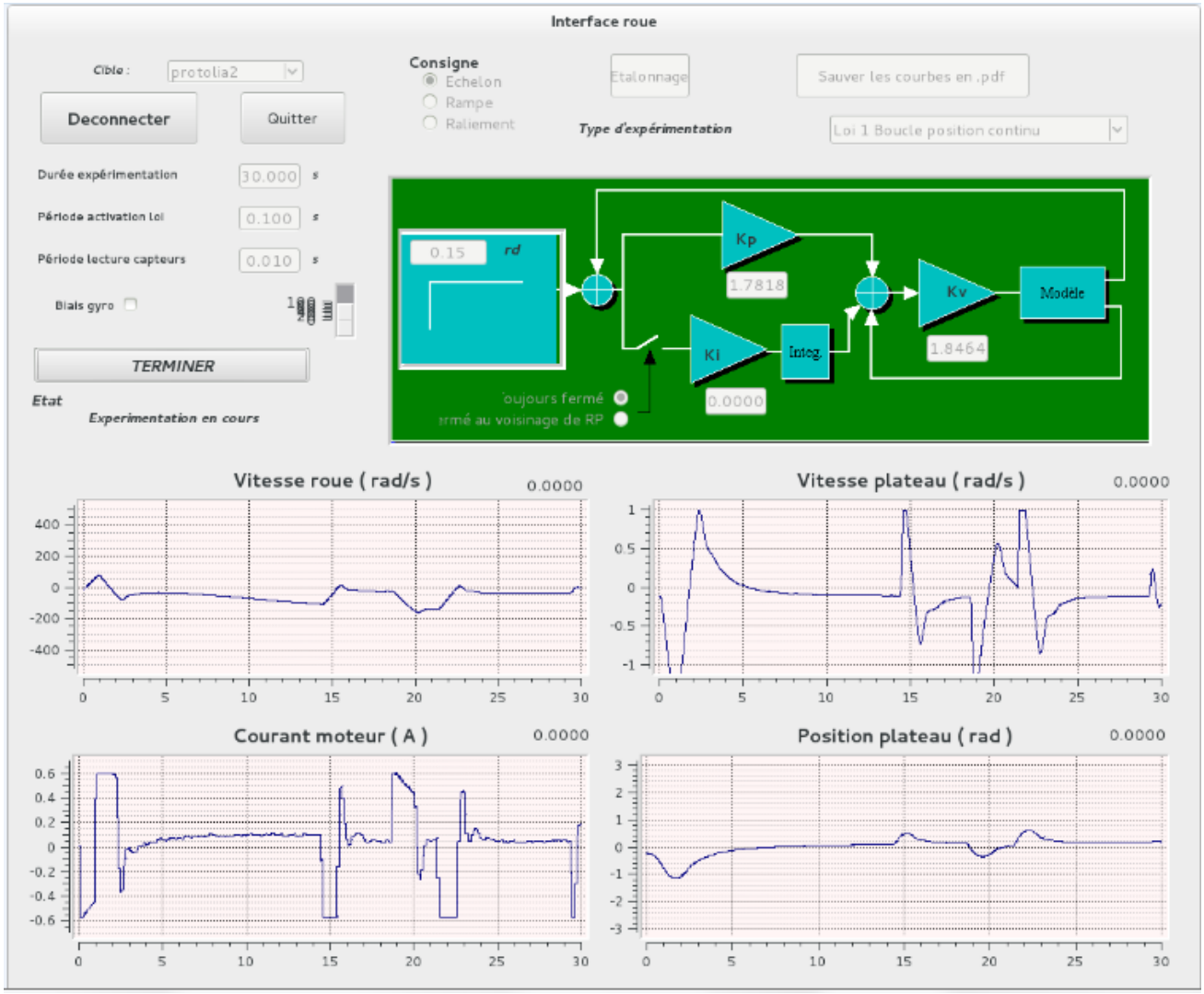


FIGURE21: Results of angular position control with Kp and Kv

The results for the actual execution using the Modal approach (with Kp, Kv and Ki) is as shown below:

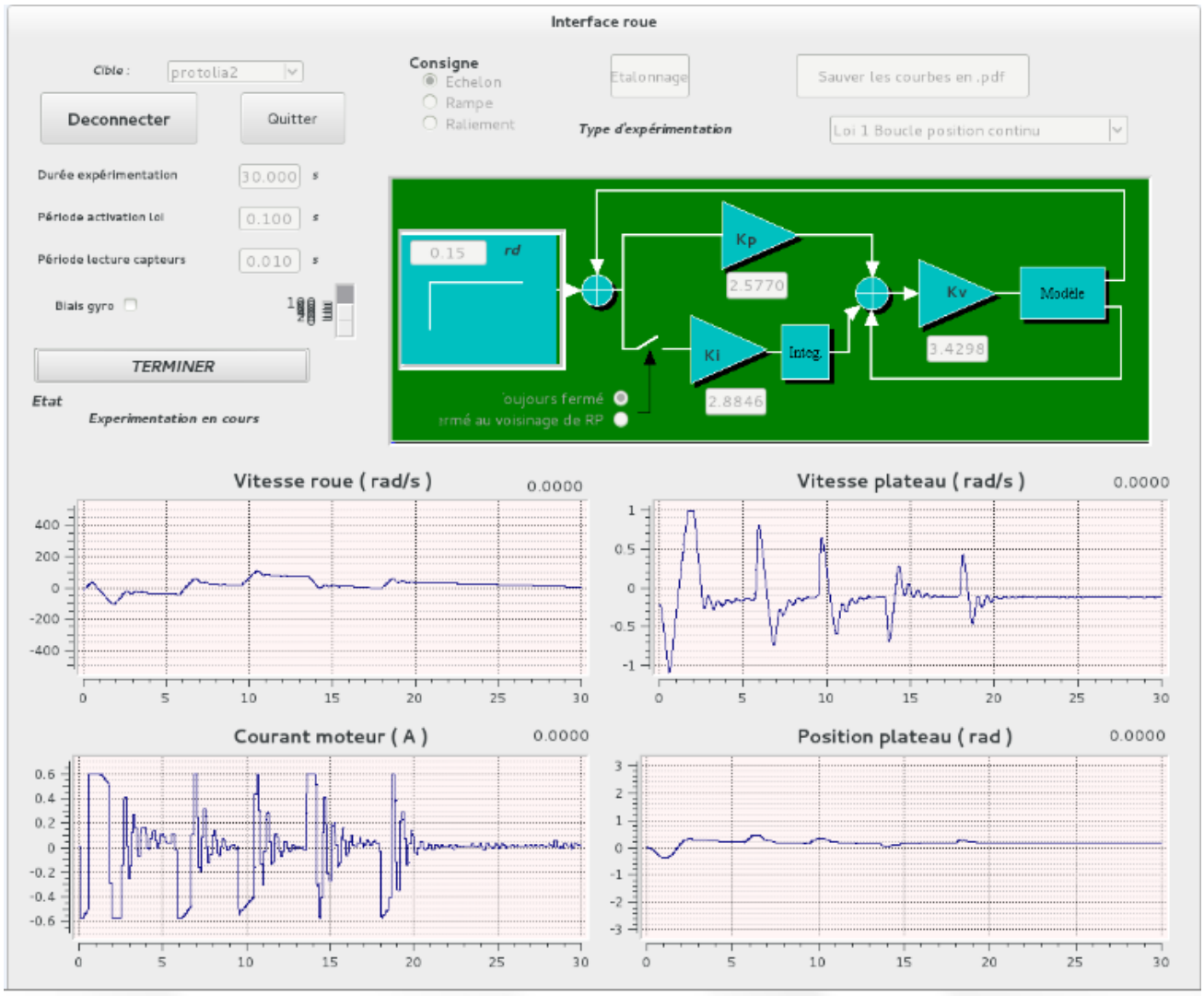


FIGURE22: Results of Position Control with Kp, Kv and Ki

The results for the actual execution using the Lead Compensator is as shown below:

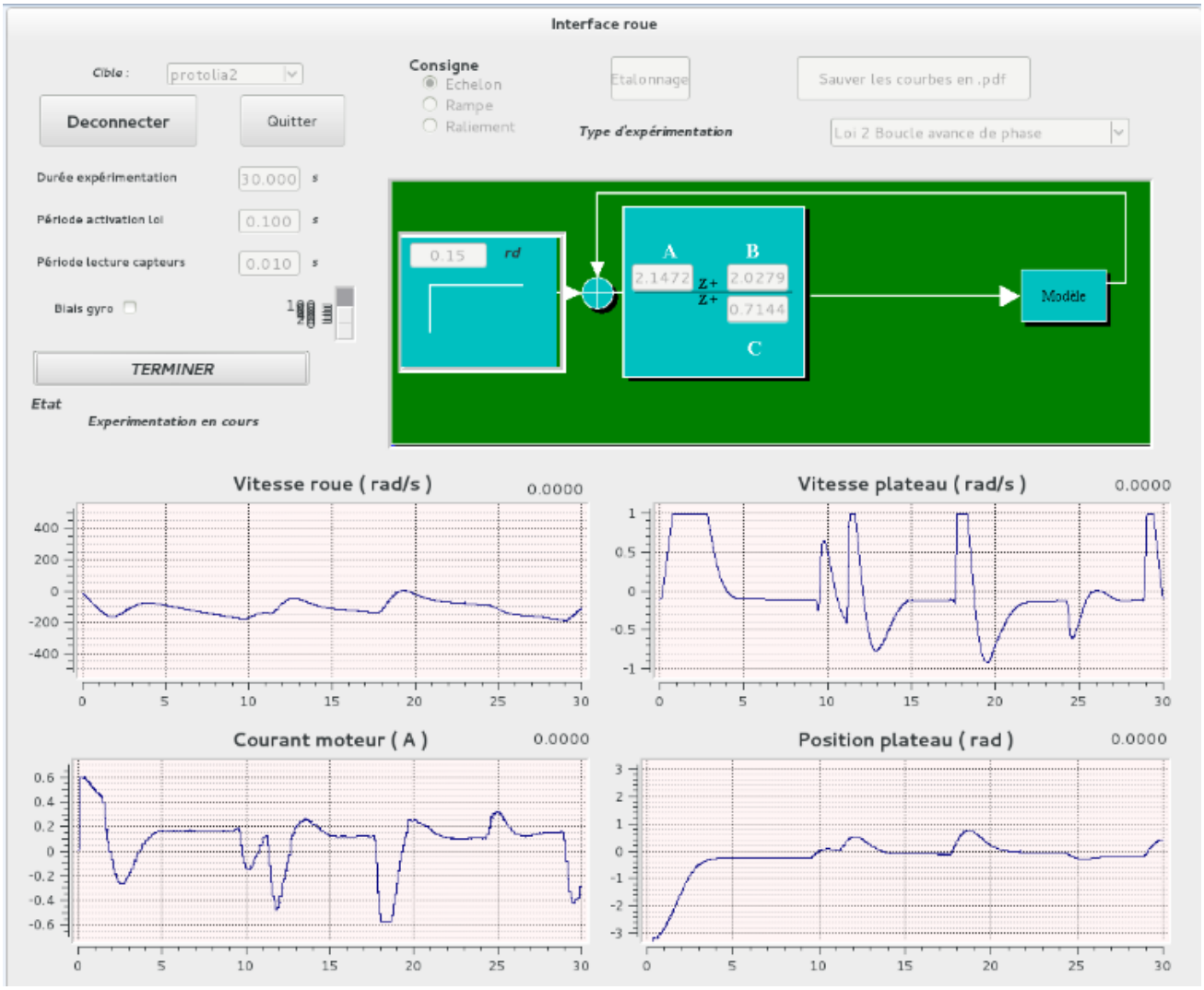


FIGURE23: Results of Control with Lead Compensator

The actual results of implementation using Lead and Lag Compensator is as shown below:

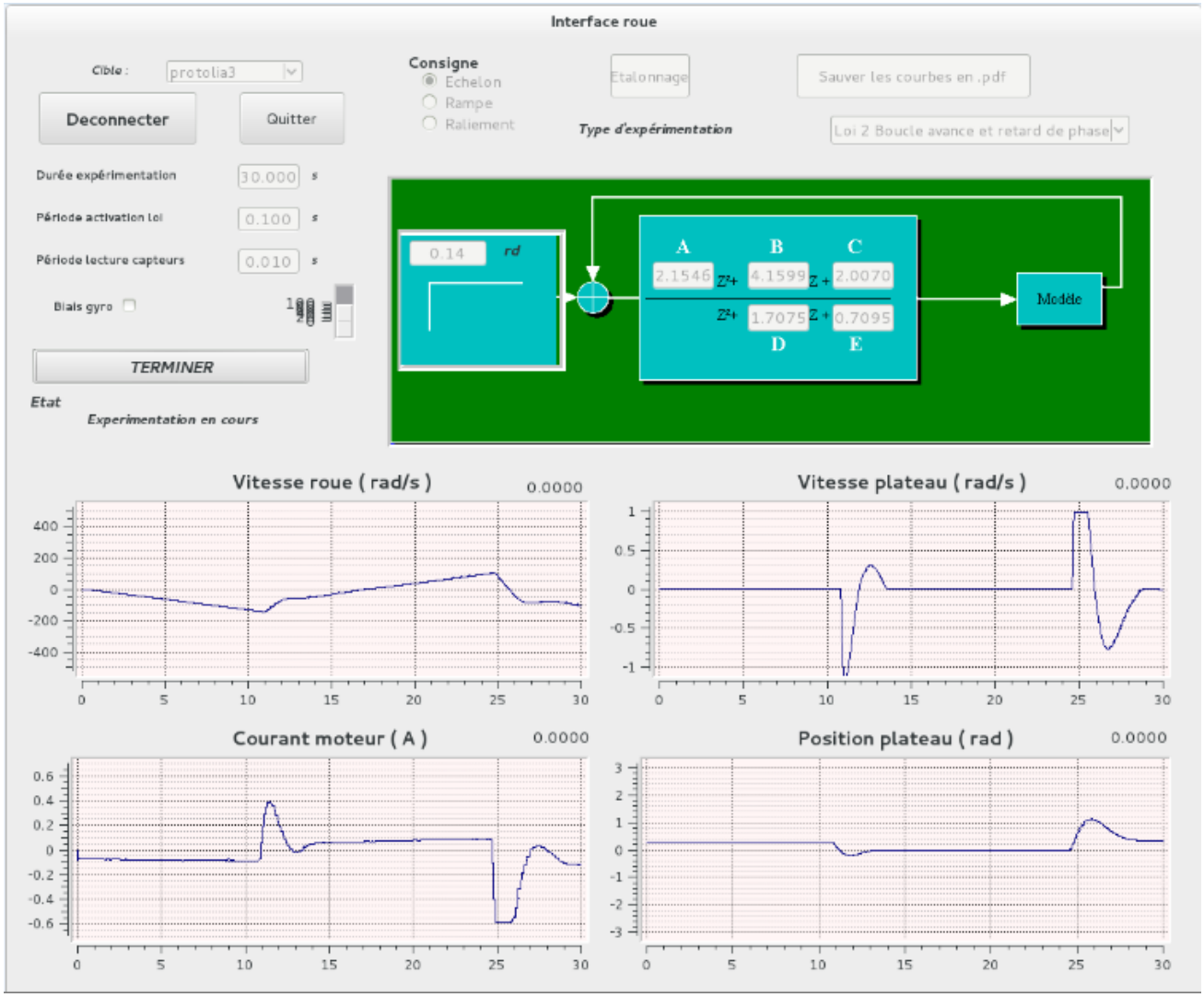


FIGURE24: Results of Control with Lead and Lag Compensator

4 CONCLUSION

We were able to successfully implement he software and the control loop using each of the 4 control modes.

The lab sessions have been very helpful in improving our understanding of Matlab and working of the real time software. It provided us the insight on how to approach a controls problem and work with matlab to arrive at a solution. The assignment taught us on how to solve a problem using several different techniques using matlab and the advantages and disadvantages of each approach. The assignment also provided insight on how to develop, derive and digitize the control loops/control parameters from matlab for software implementation.

From the lab sessions we have observed that the state feedback using modal approach method is more flexible and provide better control as we are able to control the output by monitoring multiple feedback parameters. With the lead/lead lag compensator we are able to control only one feedback parameter. This leads to loss of accurate control in the system. However, this method is useful when only one parameter is available for control.