On Chip Rapid Control Prototyping for DC motor

Romain Delpoux, Lubin Kerhuel, Vincent Léchappé

On Chip Rapid Control Prototyping for DC motor

Romain Delpoux[1], Lubin Kerhuel[2] and Vincent Léchappé[1]

[1]R. Delpoux and V. Léchappé are with Univ Lyon, INSA Lyon, ECL, Université Claude Bernard Lyon 1, CNRS, Ampère, F-69621, Villeurbanne, France. surname.name@insa-lyon.fr
[2]L. Kerhuel is with Microchip Technology Inc, F-64100, Bayonne, France. lubin.kerhuel@microchip.com

Abstract

This paper proposes a method for Rapid Control Prototyping (RCP) targeting microcontrollers. The methodology relies on a Matlab/Simulink interface which makes the target configuration and coding easier. Developing low level embedded code is bypassed by a high-level implementation which is straightforward for control system engineers. This article is intended for students, engineers or researchers looking to validate the effectiveness of their control algorithms on industrial targets. The design procedure is illustrated by testing various speed and current feedback loop on a DC motor.

keywords: DC motor, Rapid Control Prototyping (RCP), Matalb/Simulink, Mplab device block for simulink, Embedded system, Microcontroller, PI controller, Sliding Mode Controller.

1 INTRODUCTION

Digital controllers are increasingly present in industrial applications (automotive, aeronautics, space, industry ...). However, manufacturers seek solutions to reduce global cost while maintaining good performances. For example, one can replace a high precision mechanics for a pointing antenna

Article publié par EDP Sciences et disponible sur le site https://www.j3ea.org ou https://doi.org/10.1051/j3ea/20210001

with a simpler mechanical design but with an advanced control algorithm. Control engineers are dedicated to design these advanced controllers in order to guarantee that specifications are verified.

Developing embedded code for microcontrollers requires skills that are out of scope of the curriculum of control system engineers which is typically limited to high level tools like Matlab®/Simulink® and RCP tools like dSPACE® or Speedgoat. This task is often left to embedded system engineers who are specialists of hardware architecture but have little knowledge in control theory. According to some recent discussions with automotive and aerospace specialists, there is a lack of engineers able to deal with the full process from control design to hardware implementation. This observation is also confirmed by the increasing desire of students in control system courses to learn how to implement algorithms directly on dedicated hardware. Based on this assessment, it has been decided to propose a simple and fast methodology for RCP based on the interface of a microcontroller directly with Matlab/Simulink by taking advantage of the graphical interface provided by the Microchip Technology MPLAB® device blocks for Simulink. This is important to note that unlike some existing setup [1], the objective is not to 'hide' and make totally transparent the code generation but on the contrary to make it accessible in order to tackle the problems that arise from this implementation. In addition to the teaching benefit, the second objective is to fill the gap between convenient but expensive RCP platforms and complex but cheaper microcontroller tools. The developed methodology will speed up the implementation of advanced control algorithms on real hardware by removing the phase of coding on the target (see Figure 1). A dsPIC® Digital Signal Controllers (DSCs) and its dedicated MPLAB toolbox for Simulink has been used for this article but other manufacturer DSCs or Microcontroller Unit (MCU) may benefit this methodology whenever a dedicated toolbox exists. The goal of the article is to illustrate a methodology for real-time control regardless of the targeted architecture. For sake of simplicity, the example is based on a familiar DC motor, however the methods and tools used (software and hardware) can be adapted to other types of hardware. The proposed control algorithms are available online [2].

Many works in the literature present RCP solutions, however, they are usually aimed at control teaching on a dedicated board or platform, and do not propose a flexible solution that allows a deep analysis of the implementation problems: [3] (with Lego Mindstorms NXT), [4], [5] (with Arduino), [1] (with a Digital Signal Processing (DSP) system), [6] (16 bit microcontroller).

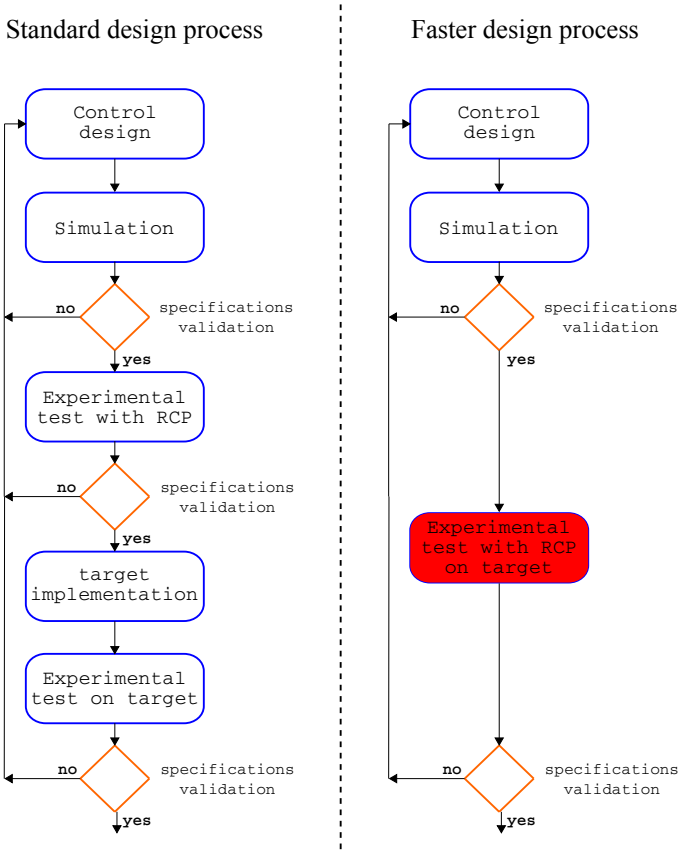

Figure 1: Control design steps

Some other works on RCP are dedicated to research purposes but they are usually focused on illustrating the efficiency of the control algorithm than providing a handy interface to analyse in detail the technical issues related to hardware coding : [7] (dsPIC DSCs), [8] (RTAI-Lab).

The contributions of the proposed method is twofold i) it speeds up the control design process by using a graphical interface that makes the configuration as easy as a dSPACE or Speedgoat RCP system, but with the major difference that it targets microcontrollers which can be embedded in custom boards ii) it smooths the learning curve for students in control system course by providing a handy interface that offers a way to gradually introduce some implementation issues (data acquisition, discretization, fixed-point conversion, Pulse Width Modulation (PWM) generation,...).

The paper is organized as follows. The DC motor model and its experimental validation are presented in Sections 2 and 3 respectively. Section 4 explains the design of the control algorithms and shows the experimental results. Some hardware related considerations are exposed in Section 5 and finally conclusions are drawn in Section 6.

List of acronyms

ADC Analog Digital Converter

DSP Digital Signal Processing

EMF Electro-Motive Force

LSB Least Significant Bit

MCU Microcontroller Unit

DSCs Digital Signal Controllers

PIM Plug In Module

PWM Pulse Width Modulation

QEI Quadrature Encoder Interface

RCP Rapid Control Prototyping

SMC Sliding Mode Controller

STA Super Twisting Algorithm

2 Study case introduction: DC motor

The methodology is illustrated by using a DC motor. It represents a complete example where control algorithm are easy to understand. The motor used is a Pravalux Brushed DC Motor (90 W, 24 V dc, 3000 rpm). The motor parameters and variables are given in Table 1.

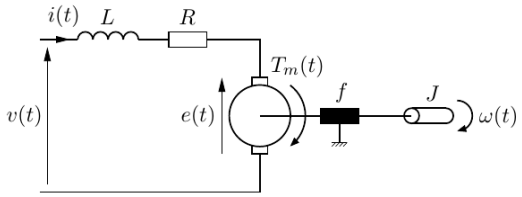

Figure 2: DC motor schematic

Table 1: List of variables and parameters.

Variables	**Values**	**Units**
$i(t)$: current	$[-4.4\ 4.4]$	A
$v(t)$: voltage	$[-12\ 12]$	V
$e(t)$: back EMF		V
R : Resistance	1.22	Ω
L : Inductance	2.7×10^{-3}	H
K_ϕ : back EMF Constant	0.061	V.s/rad
$T_m(t)$: Motor Torque		N.m
$\omega(t)$: Angular velocity	$[-300\ 300]$	rad.s^{-1}
f : friction torque	1.1×10^{-4}	N.m.s.rad^{-1}
J : Moment of inertia	2.2×10^{-4}	kg.m^2

2.1 DC motor description

From Kirchhoff's voltage law, one has

$$L\frac{di(t)}{dt} = v(t) - Ri(t) - e(t). \tag{1}$$

By Newton's law, one gets

$$J\frac{d\omega(t)}{dt} = \sum T(t) = T_m(t) - f\omega(t) \tag{2}$$

where $T(t)$ is the total torque applied on the rotor. The back Electro-Motive Force (EMF) is proportional to the speed

$$e(t) = K_e\omega(t)$$

with K_e the electromotive constant ($V.rad^{-1}.s$). The motor torque is proportional to the current

$$T_m(t) = K_T i(t)$$

with K_T, the torque constant ($N.m.A^{-1}$). The mechanical power produced by the DC motor is $T_m\omega = K_T i\omega$. The electric power $P_e = vi$ delivered by the source is transformed into heat loss in the resistance R, into stored magnetic energy in the inductance L and the remaining quantity $iK_e\omega$ is converted to mechanical energy $T_m\omega$. It leads to $T_m\omega = K_T i\omega = K_e i\omega$ where $K_T = K_e = K_\phi$ [9].

The DC motor has two dynamics, a fast electrical dynamics (1) and a slow mechanical one (2). This will be treated using a multi-rate controller. High computation constraints are required for the fast dynamic control and lower constraints for the other one.

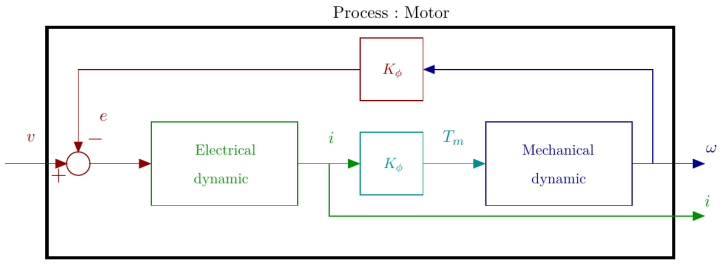

Figure 3: DC Motor Bloc Diagram

3 Model validation

In the previous section, a theoretical model for the DC motor was presented, however differences between theory and practice always exist. That is why, the first step is to compare the theoretical model with real data from the experiment to ensure that the model is accurate. This phase makes it possible to refine the simulation model and consider the maximum of uncertainties in the simulation model. It is important to differentiate between simulation and control model. The **simulation model** must take into account most of the physical phenomena in order to be the closest to the experimental results. The **control model** on the other side must encompass the dominant dynamics in order to be able to propose a constructive control design. The application of the control law on the simulation model shows the robustness of the control and highlights some potential problems that could arise on the real system. Possible issues can be saturation of the command due to high frequency noise or dynamic not taken into account by the control model.

Overflow and lacks of resolution can also be detected if fixed-point calculation are performed but not scaled appropriately. Fixed-point calculation can be used in critical steps where high speed execution is required. Section 5 provides further insight on fixed point scaling tools.

Real-time experiments data can be sent to MATLAB/Simulink using the UART. The received logs can be used in simulation to compare an input sent both to the model and the real system. Figure 4 represents such a scenario. The experimental data are played in simulation. For this experience, open loop voltage steps are applied to the motor as well as to the model proposed in Section 2. The comparison of current and velocity measurements are plotted on Figure 5. The figure shows that the velocity and current dominant dynamics are well represented in our model. It means that the low frequency behaviour is well considered in the model so it validates the control model. However the experimental current exhibits high frequency oscillations. To observe the behaviour, a zoom of the figure is given on Figure 5. The oscillation has a frequency 12 times higher than the motor speed. It corresponds to the number of motor poles pairs and represents the current peaks caused by the passage of coals from one pole to the next. This was not considered in our modeling and should be added in the simulation model by adding an oscillation on the motor back EMF.

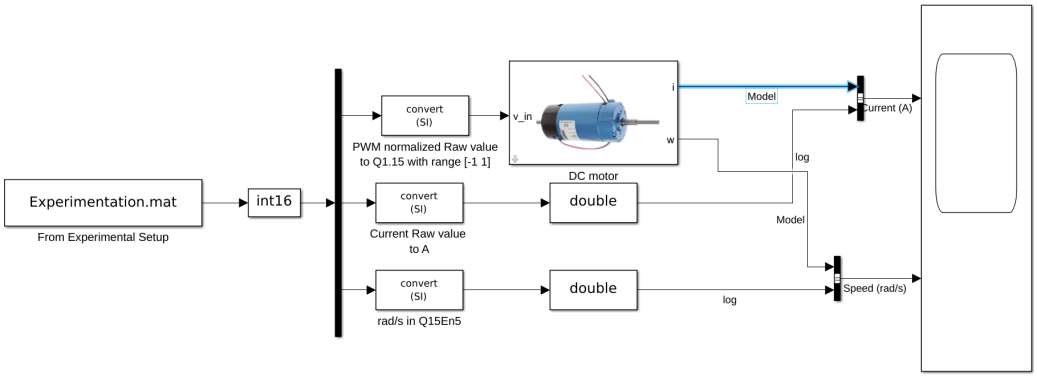

Figure 4: Use of log as simulation input

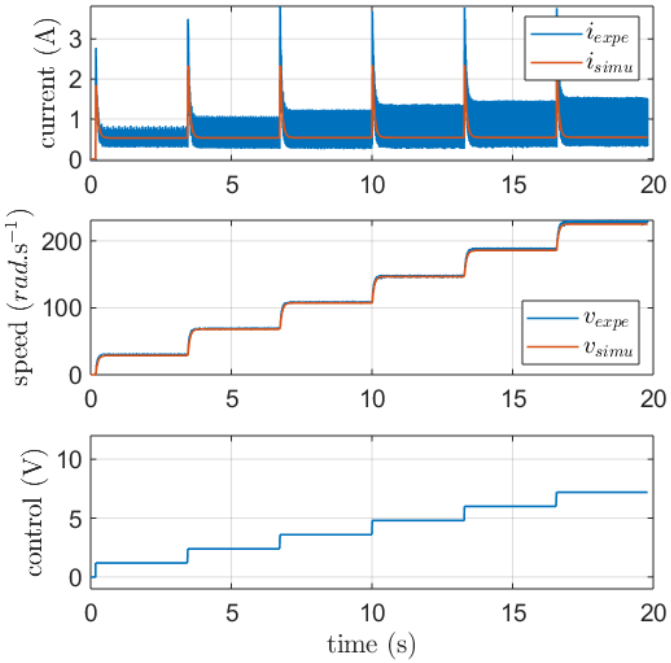

Figure 5: Experimental results vs. simulation

4 Control

The control design is based on a cascaded control method. Indeed, the frequency separation allows to split the controller into two control loops (Figure 7). The inner loop controls the fast electrical dynamics while the outer loop handles the slow mechanical dynamics. For each loop, different types of controllers can be designed. Here, a PI controller, a conventional Sliding Mode Controller (SMC) and Super Twisting Algorithm (STA) will be tested for the electrical loop. As far as the mechanical control loop, a PI controller will be implemented. The design of these controllers is given in the next subsections.

Remark 1 *Generally, for small DC motor, the dynamics of the electrical part is neglected and only the mechanical dynamics is considered. However, whenever the motor resistance is low, this strategy may damage the motor because of current peaks during transients.*

Remark 2 *The fast inner loop is sampled at* 20 kHz *and the output loop at*

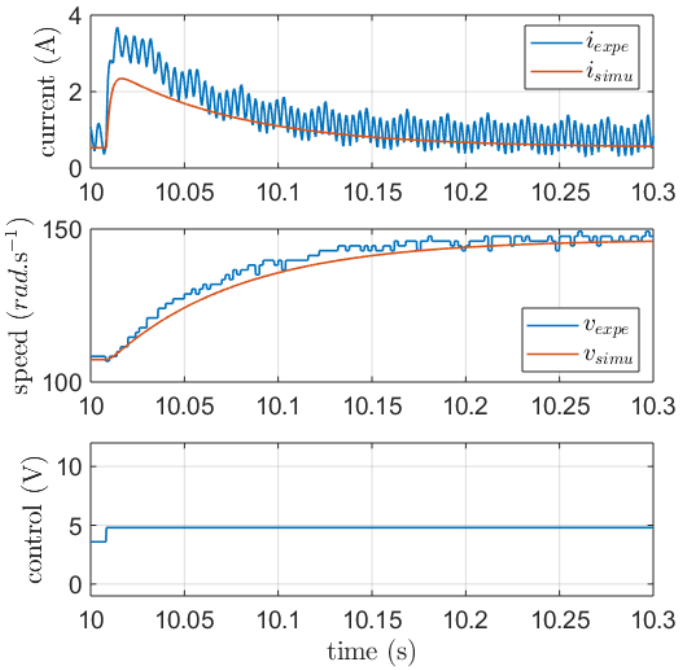

Figure 6: Zoom of the experimental results vs. simulation

1 kHz. *Note that this cascaded control method is a good example to illustrate a multi-rate controller implementation.*

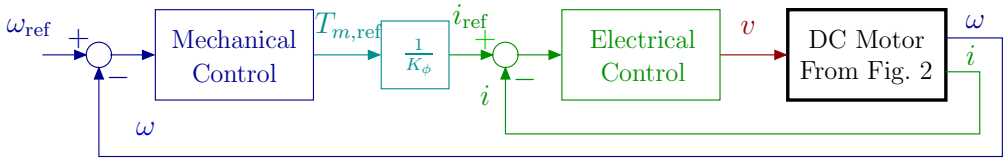

Figure 7: Closed-loop cascaded system

4.1 Electrical control loop

The current dynamics in the motor is governed by (1) and the control objective is to ensure that $i \to i_{\text{ref}}$ where i_{ref} is the current reference provided

by the output of the mechanical controller (Figure 7) designed in the next subsection. Note that controlling the current will impose the torque value since $T_m(t) = K_\phi i(t)$.

4.1.1 PI controller

Assuming that the mechanical dynamic is slower than the electrical, *i.e*

$$\tau_{\text{elec}} = \frac{L}{R} \ll \tau_{\text{meca}} = \frac{J}{f},$$

then the velocity ω can then be considered as constant with respect to the dynamics of the current i. In this case, one can consider the term $-\frac{K_\phi}{L}\omega$ as a constant perturbation which will be rejected by adding an integral action in the controller in order to obtain a zero steady state error [10]. The PI controller has the following expression

$$v^{PI} \quad = \quad -K_{P_i} i + K_{I_i} \varepsilon_i \tag{3}$$

with $\dot{\varepsilon}_i = i_{\text{ref}} - i$. The resulting closed-loop dynamics with $v = v^{PI}$ is

$$\begin{bmatrix} \frac{di}{dt} \\ \frac{d\varepsilon_i}{dt} \end{bmatrix} = \underbrace{\begin{bmatrix} -\frac{R+K_{P_i}}{L} & \frac{K_{I_i}}{L} \\ -1 & 0 \end{bmatrix}}_{\bar{A}_i} \begin{bmatrix} i \\ \varepsilon_i \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} i_{\text{ref}} + \begin{bmatrix} -\frac{K_\phi}{L}\omega \\ 0 \end{bmatrix}.$$

The closed-loop dynamics depends on the eigenvalues of the matrix $\bar{A}_i$ which can be fixed in order to verify predefined specifications like overshoot and convergence speed. By identification with the canonical form of a second order system one gets

$$K_{P_i} = 2L\xi\omega_n - R \qquad \text{and} \qquad K_{I_i} = L\omega_n^2$$

where ξ and ω_n are the desired damping ratio and natural frequency respectively.

Remark 3 *Assuming that K_ϕ is perfectly known and since the speed ω is measured, it would have been possible to cancel the term $-\frac{K_\phi}{L}\omega$ in (1) by defining $v^{PI} = -K_{P_i} i + K_{I_i}\varepsilon_i + K_\phi\omega$. Here it is has been decided to treat the whole term $-\frac{K_\phi}{L}\omega$ as a perturbation. This choice is made from a pedagogical point of view because it allows studying the robustness of the controller.*

4.1.2 Conventional Sliding Mode Controller

Sliding mode control is known to be robust to uncertainties and external disturbances thanks to the presence of a discontinuous term in the controller. In our case the term $-\frac{K_\phi}{L}\omega$ is considered as perturbation that is why it is relevant to test sliding mode controllers. First, the conventional sliding mode controller of order 1 is defined by

$$v^{\text{SMC}} = Ri + K_{\text{SMC}} \cdot \text{sign}(\sigma_i) \tag{4}$$

where $\sigma_i = i_{ref} - i$ is the sliding surface. Taking $v = v^{\text{SMC}}$, the dynamics (1) becomes

$$\frac{di}{dt} = -\frac{K_{\text{SMC}}}{L}\text{sign}(\sigma_i) - \frac{K_\phi}{L}\omega.$$

By a Lyapunov analysis, the following stability condition

$$K_{\text{SMC}} > L\left|\frac{di_{ref}}{dt}\right| + K_\phi|\omega| \tag{5}$$

is obtained [11]. Assuming that i_{ref} is slow, condition (5) is approximated by $v_{max} > K_\phi\omega_{max}$ where ω_{max} is the maximum speed of the motor. Note that taking K_{SMC} too large will increase undesired fast oscillations known as chattering phenomenon [12]. As a consequence, there is a compromise between response time[1] and chattering.

Remark 4 *Note that the term Ri in (4) is not necessary but will allow to reduce the chattering effect.*

4.1.3 Super Twisting Algorithm

Conventional sliding mode controllers suffer from chattering, which produce a very unpleasant noise and can damage the motor in the long run. To overcome this problem while preserving the robustness property of the SMC, a STA is tested. This algorithm has the advantage to have a continuous control v^{STA}. The STA is defined by

$$\begin{cases} v^{\text{STA}} &= Ri + c_i|\sigma_i|^{0.5}\text{sign}(\sigma_i) + w_i \\ \dot{w}_i &= b_i \cdot \text{sign}(\sigma_i) \end{cases} \tag{6}$$

with $\sigma_i = i_{ref} - i$ and b_i, c_i positive constants. A careful choice of b_i and c_i guarantees the stability of the closed-loop system with $v = v^{\text{STA}}$ [12].

[1] It can be shown that the larger K_{SMC}, the faster the response time.

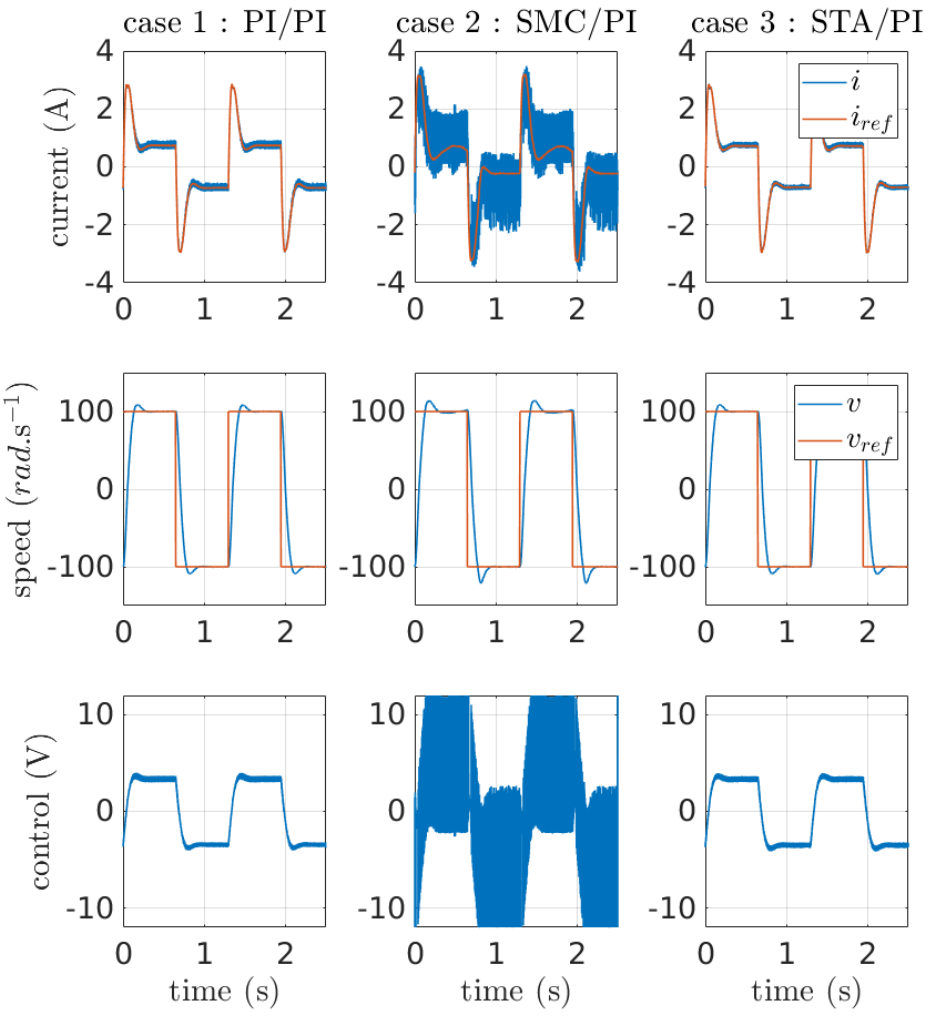

Figure 8: Experimental results comparison between electrical loop controllers (see Table 2 for the explanation of the different cases)

4.2 Mechanical control loop

Assuming that the electrical loop is much faster than the desired mechanical dynamics, then the mechanical control loop can be designed considering that

$i = i_{ref}$. From (2), the mechanical dynamics becomes

$$\frac{d\omega}{dt} = \frac{K_\phi}{J} i_{\text{ref}} - \frac{f}{J}\omega. \tag{7}$$

the control objective is to get $\omega \to \omega_{\text{ref}}$ where ω_{ref} is the desired speed. Unlike the electrical control loop, it is not possible to design a conventional sliding mode controller because a smooth reference i_{ref} is required for the electrical loop [11]. The PI controller and the STA are both continuous and could be implemented. However, for clarity and because of space limitation, only the PI controller is not presented here. The control design is similar to the one proposed for the electrical dynamics. The expression of the controller is

$$i_{\text{ref}}^{PI} = -K_{P_\omega}\omega + K_{I_\omega}\varepsilon_\omega \tag{8}$$

with $\dot{\varepsilon}_\omega = \omega_{\text{ref}} - \omega$. The resulting closed-loop dynamics with $i_{\text{ref}} = i_{\text{ref}}^{PI}$ is

$$\begin{bmatrix} \dot{\omega} \\ \dot{\varepsilon}_\omega \end{bmatrix} = \underbrace{\begin{bmatrix} -\frac{K_\phi K_{P_\omega} + f}{J} & \frac{K_\phi K_{I_\omega}}{J} \\ -1 & 0 \end{bmatrix}}_{\bar{A}_\omega} \begin{bmatrix} \omega \\ \varepsilon_\omega \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \omega_{\text{ref}}.$$

Similarly, to the electrical control loop design one can choose

$$K_{P_\omega} = \frac{2J\xi\omega_n - f}{K_\phi} \quad \text{and} \quad K_{I_\omega} = \frac{J\omega_n^2}{K_\phi}$$

where ξ and ω_n are selected according to the speed dynamics specifications.

4.3 Experimental results

In this section, a comparison of the different current controllers is drawn. The different scenario are shown in Table 2. The electrical and mechanical PI controllers are tuned to get a 5% overshoot and a time response three times faster than the open-loop one. The experimental results are displayed on Figure 8. For the three cases, one can see that the speed tracking is fast and exhibits an overshoot and a time response that are coherent with the specifications. It can be seen that the SMC causes chattering on the current. In addition, the SMC produces an unpleasant noise during the experiment. The PI and STA lead to similar performances with slightly better noise attenuation for the STA. For future works, it would be interesting to add an external torque perturbation in order to study the robustness of the different algorithms and see if the STA is more robust than PI controller.

	Electrical control loop	Mechanical control loop
case 1	PI (3)	PI (8)
case 2	SMC (4)	PI (8)
case 3	STA (6)	PI (8)

Table 2: Different control scenario

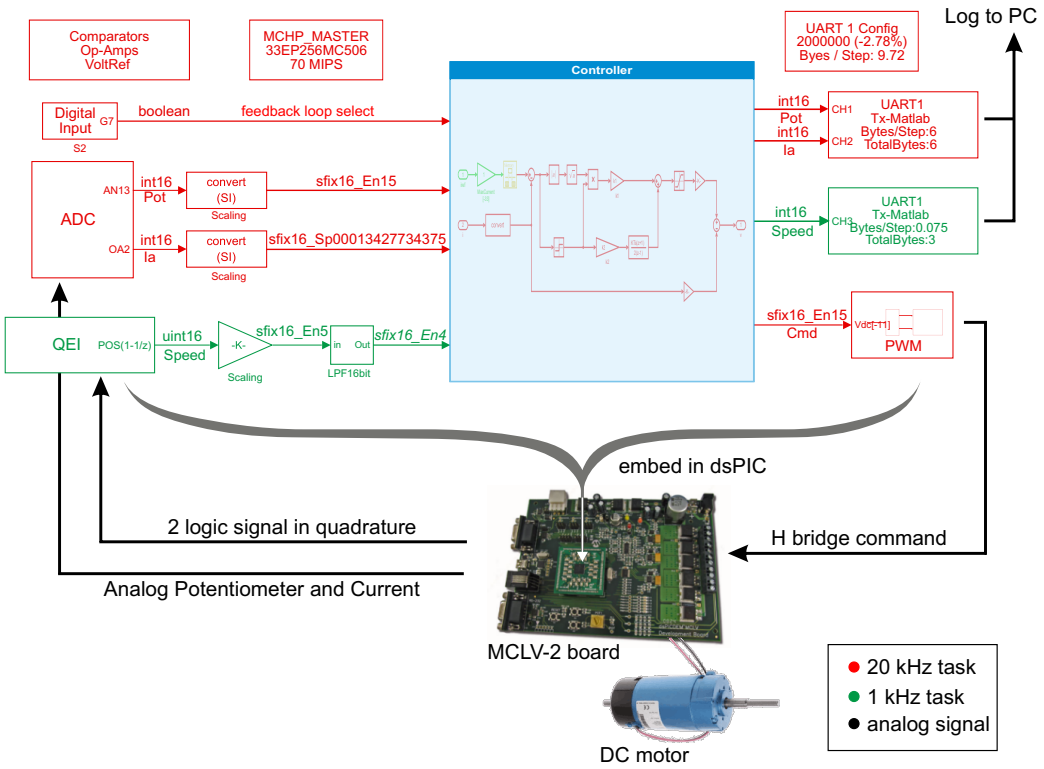

Figure 9: DC control motor global scheme. The model is compiled and run as a standalone program on the targeted board

5 Technical considerations

This section details the important steps for the real-time implementation of a multi-rate control law. The Simulink interface with the motor using the MPLAB device blocks is represented on Figure 9. The control voltage is applied to a PWM signal driving a H-bridge converter. The inputs of the

control algorithm are the current and velocity measurements. The velocity is obtained from an incremental encoder through the use of a Quadrature Encoder Interface (QEI) peripheral. The current is measured using shunt resistor, conditioned with the dsPIC DSCs internal op-amps and converted with the ADC peripheral. The Analog Digital Converter (ADC) conversion is synchronized with the PWM period. It ensures that the ADC sampling time is taken when the low side of the H-bridge is on. Speed and current control loops are sampled at different rates. An option in Simulink highlights with color the different rates in a model as presented on Figure 9. The red color represents the fastest sampling rate. Here the sampling frequency for the control of the fast current dynamics is $F_{\text{elec}} = 20$ kHz. The green color represents the sampling rate for the mechanical dynamics, here $F_{\text{meca}} = 1$ kHz.

5.1 Requirements

Hardware The Microchip MCLV-2 board is targeted. It is equipped with the 16-bit dsPIC DSCs 33EP256MC506 Plug In Module (PIM) version using the dsPIC internal op-amps for current signal conditioning. The board drives the DC motor thanks to a H-bridge.

Software Matlab/Simulink is used with the Embedded Coder and the fixed-point toolbox from MathWorks. The MPLAB device blocks for Simulink adds to Simulink the capability to target up to 270 Microchip microcontrollers (dsPIC, PIC32). Theses blocks rely on XC16 compiler and the MPLAB X IDE [2] to compile the generated code and program the board from the Simulink interface with a single 'Build' push button on top right of the simulink interface (see Figure 10). The compiled model has a discrete time solver and few dsPIC peripheral blocks.

5.2 Rapid Control Prototyping (RCP)

Standard Simulink blocks generate code, which compiles on a dsPIC DSCs target. Any model can thus be embedded with respect to the following constraints:

[2] blockset, Compiler and IDE are tools available from www.microchip.com.

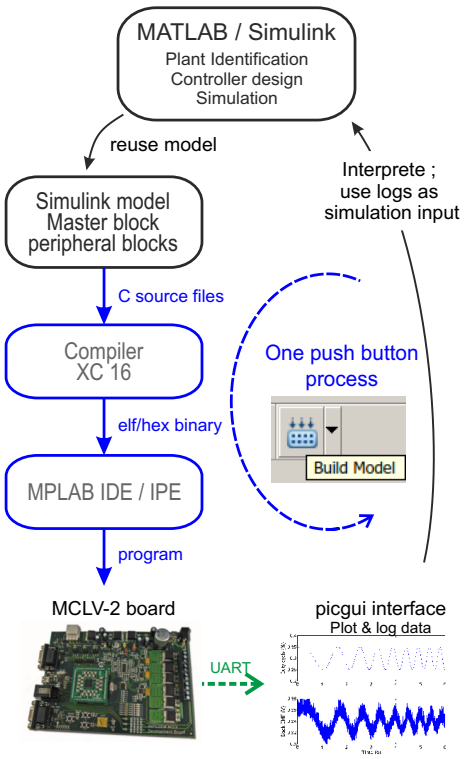

Figure 10: Rapid Prototyping Scheme

Discrete time Simulation are typically performed in the continuous time domain. A differential equation solver is used to solve simulation outputs. The real-time embedded software does not implement such solver so a discrete-time equivalent controller must be used. Multi-rate model comprises blocks running at various rates. For each block, it is possible to set its rate and its offset. Blocks rate offset are multiple of the base model time step defined in the solver panel. A multi-rate model can be implemented using a multi-tasking scheduler (default settings) or a single tasking program.

In a **Single tasking program**, all tasks started at a given base model time step must be completed within the end of that time step slot to respect real-time constraints.

In a **Multi-tasking scheduler**, a monotonic-rate scheduler is implemented where higher rate tasks have higher priority and interrupt lower task rate which have lower priority whenever required. This multi-tasking scheduler has a simple priority rule which is well suited for automatic. Its limited

implementation penalty in execution time worth the gain in flexibility.

Figure 11 presents a timing analysis of the cascaded PI algorithms. One pins driven by the 'CPU load' block and two pins driven by the 'task state' block shows respectively the CPU state (black curve where 1 is busy and 0 is idle state), the fast task D1 (red) and slow task D2 (green) respective start and stop on rising and falling edges. The lower graph is a magnification of the higher priority task D1 preemption of the lower priority task D2. Note that, the slow D2 task pin state is not cleared when beeing preempted but exclusively when task is completed. The shaded region in the tasks D1 and D2 shows the CPU execution on each task. Here the control algorithm is modified to show an example of preemption: The mechanical control loop is implemented using floating point instead of fixed point so as to increasing the related task 'D2' execution time to 55μs which is above the 50μs period of the 20kHz D1 task. This slow down of the D1 task allows illustrating the benefit of a Multi-tasking scheduler as presented on Figure 11.

Fixed point Another constraint is the limited power from low cost micro-controllers in comparison to dSPACE of Speed-Goat. Discrete step of 25μs or (40kHz control loop frequency) can be reached but floating-point calculation should be avoided in high rate task. Floating-point calculation can be used in lower rate tasks as illustrated by the D2 task in Figure 11.

Matlab/Simulink provides a Fixed-point toolbox to help with the conversion of the floating-point model into a fixed-point equivalent. One method consists in logging values taken by each variable during a simulation using double datatype. Then for each variable, it is possible from the distribution histogram to choose the variable data length, which typically would be either 16 or 32 bits, and to select its scaling.

Scaling: Simulink is not limited to floating point value, integers and the classical fixed point (noted Qx.xx) value within [−1 1] or [0 1] range. The Simulink fixed-point datatype is very flexible as the Least Significant Bit (LSB) can represent any quantity. For example if we use the common 16 bits signed fixed-point (Q1.16), the LSB would represent the quantity 2^{-15}. Simulink sine and cosine blocks output provide a 16 bit output value with a LSB representing 2^{-14}, allowing to represent value in the range $[2-2^{-14}\ -2]$, including thus the value 1.

Notation: Math blocks can handle inputs having different datatype. Block output datatype could be either determined from the blocks input

datatype, or forced to a given datatype. A Simulink option shows on the model the datatype for each block connection. Simulink notation example with their meaning is given below:

- sfix16_En14 is a signed 16 bit variable with a negative power of 2 exponent: LSB represents 2^{-14}.
- ufix16_E2 is an unsigned 16 bit variable with a power of 2 exponent: LSB represents 2^2.
- sfix16_Sp05 is a signed 16 bit variable where the LSB represents 0.05.

Blocks, where internal or output datatype can be set, provide a user interface with options like 'inherited from internal rules' or 'inherited from back propagation'. The datatype can also be forced using either the 'Binary point', the 'Slope and bias' or the 'Best precision' option. The 'Binary point' and 'Slope and Bias' options set the variable LSB representation value. The 'Best precision' option compute the lowest LSB power of 2 scaling so as to be able to represent the 'min' and 'max' values set in the block GUI. This will provide the best resolution for this variable.

5.3 Timing analysis and efficiency

Some blocks are available for timing analysis.

MCU overload The block detects overload. A physical pin can be used to detect an overload through a LED or a scope. A block output can provide overload information to the embedded program. The pin output works asynchronously: the pin is set as soon as an overload occurs and is never reset by the DSCs load block. It is however possible to reset the pin using a 'digital output write' block which can be set again by the MCU overload block. The block output works synchronously. For each MCU overload block present on the model, any overload occurrence between successive evaluation of the block are reported in a 16 bit-field integer where the bit 0 code for an overload of the task D0 (fastest rate), bit 1 for the task D1 until D14. The bit D15 represents overload on task D15 and higher if the model use more than 16 different tasks.

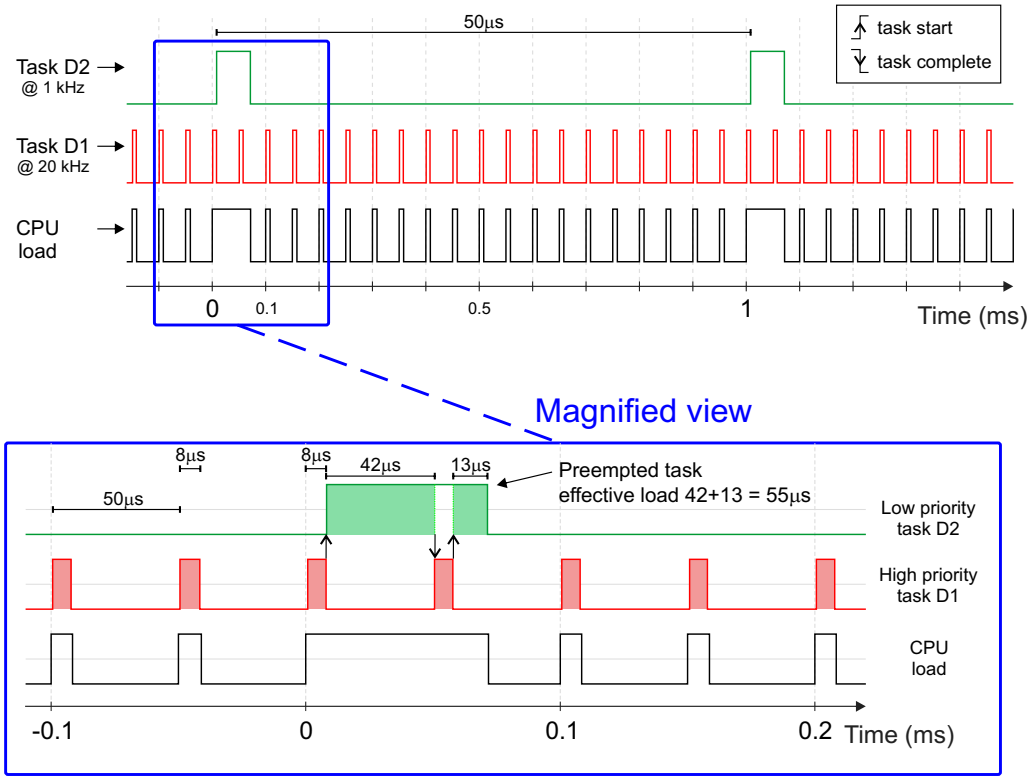

Figure 11: Scope measurement of task execution for a Multi-rate model with a multi-tasking rate monotonic scheduler

MCU Load The block measures the overall load of the DSCs. It can output the load by toggling a pin. The high state shows the load. The block can also measure the DSCs load using internal timers providing the value to the embedded program as a block output. A timer is incremented when the DSCs is either running a task, or running a peripheral interrupt (*i.e* not idle). The block report the timer increment between two evaluations of this block. The timer resolution should be selected to be able to measure a period corresponding to the block sample time. The measured time should be equal or lower than the block sample time. It's worth noticing that a 100% load on one time step (for example in Figure 11 from time 0 to 50μs) does not mean that an overload takes place. Multiple MCU load blocks can be placed within one model with different sample times, allowing to average the load over different time period corresponding to the respective block sample time.

Task State The block allows to represent task execution state through output pin. Each pin representing one selected task. Each task switches its pin high when it starts, and switches back its dedicated pin low when completed. Figure 11 shows such analysis for the inner and outer control loop tasks of the cascaded PI controller detailed in the above section.

Figure 11 presenting DSCs load measurement in a multi-tasking scheduler use theses blocks.

Code efficiency The code performance are surprisingly good. Peripheral are handled as in background through interrupts. Peripheral blocks for input peripherals are thus not blocking as they just have to read the last result obtained from that peripheral; keeping the CPU time dedicated to the control algorithm. For example, the base time step is typically triggered by the ADC block end of conversion interrupt thus when the ADC block is evaluated within the algorithm, the conversion results are already available. The ADC block do not have to wait for a sample and hold sequence nor by the following conversion sequence. The same remark holds for others peripheral blocks like the UART transmission/reception, the SPI or I2C blocks...

Code replacement This MathWorks functionality allows replacing code of standard Simulink blocks by an optimized code for the target so as to benefit from the optimized target hardware architecture like its DSP unit or specific instruction set. Code replacement is implemented for common operations like rounding, saturation and few operations on matrix. It is also implemented for functions like square root, sine and cosine functions when used with fixed point datatype input.

5.4 Human Machine Interface (HMI)

No specific HMI was developed for this lab. The MCLV-2 board possesses two switch buttons and a potentiometer which are used to set the modes and the reference. To plot and log data, a built-in graphical user interface (picgui) provides an easy way to visualize and log data. Data to be plotted are sent using UART. The picgui tool is represented Fig. 12

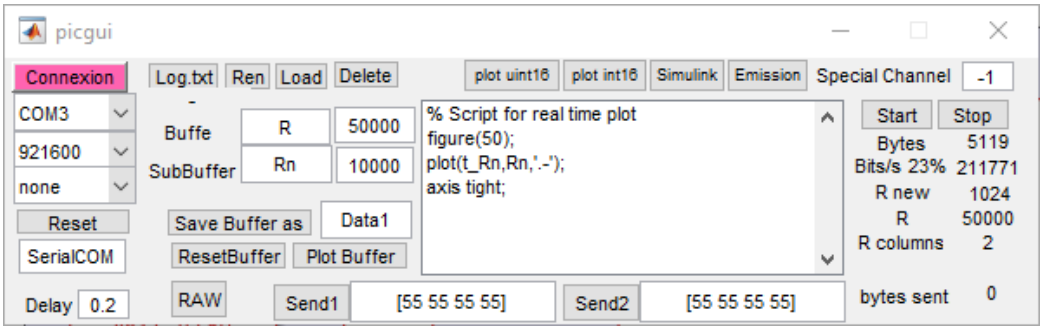

Figure 12: picgui

6 Conclusion

In this paper, we have proposed a method for RCP that targets microcontrollers which can be embedded in custom boards. The different steps for a real-time embedded multi-rate control where presented in detail. The proposed method was illustrated with experimental results from a DC motor control. This method was taught to a group of around 15 students in last year of engineering school (Master 2 equivalent) at INSA Lyon. In 8 hours, starting from scratch, students were able to control the DC motor using the microcontroller. Students feedbacks were excellent. In the future, the method will be extended to control a brushless DC motor with field oriented control.

References

[1] D. Hercog, M. urkovi, and K. Jezernik, "DSP Based Rapid Control Prototyping Systems for Engineering Education and Research," in *Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design*, pp. 2292–2297.

[2] "www.ctrl-elec.fr."

[3] C. Rodrguez, J. L. Guzman, M. Berenguel, and S. Dormido, "Teaching real-time programming using mobile robots," in *11th IFAC Symposium on Advances in Control Education ACE 2016*, vol. 49, pp. 10–15.

[4] H. M. Omar, "Enhancing automatic control learning through Arduino-based projects," vol. 43, no. 5, pp. 652–663.

[5] A. Soriano, L. Marn, M. Valls, A. Valera, and P. Albertos, "Low Cost Platform for Automatic Control Education Based on Open Hardware." in *Proceedings of the 19th IFAC World Congress*, vol. 47, pp. 9044–9050.

[6] R. Chakirov and Y. Vagapov, "Rapid Control Prototyping Platform for the Design of Control Systems for Automotive Electromechanical Actuators," in *5th IFAC Symposium on Mechatronic Systems*, ser. 5th IFAC Symposium on Mechatronic Systems, vol. 43, pp. 646–651.

[7] M. Lizarraga, G. H. Elkaim, and R. Curry, "SLUGS UAV: A flexible and versatile hardware/software platform for guidance navigation and control research," in *Proceedings of the 2013 American Control Conference*, pp. 674–679.

[8] R. Bucher and S. Balemi, "Rapid controller prototyping with Matlab/Simulink and Linux," vol. 14, no. 2, pp. 185–192.

[9] J.-N. Chiasson, *Modeling and High-Performance Control of Electric Machines*, ieee press ed., 2005.

[10] K. Ogata, *Modern Control Engineering*, ser. Instrumentation and controls series. Prentice Hall, 2010.

[11] V. Utkin, J. Guldner, and J. Shi, *Sliding Mode Control in Electro-Mechanical Systems*, ser. Automation and Control Engineering. CRC Press, vol. 31.

[12] Y. Shtessel, C. Edwards, L. Fridman, and A. Levant, *Sliding Mode Control and Observation*, ser. Control Engineering. Springer.