Distributed Command, Monitoring and Control of a DC Motor with a PC and an ARDUINO

Prof. Alexandre Bernardino
Prof. José Gaspar
Prof. Rodrigo Ventura

Área Científica de Sistemas de Decisão e Controlo
Dept. de Eng. Electrotécnica e de Computadores
Instituto Superior Técnico
1 Introduction

The main objective of this project consists of developing a real-time distributed control architecture for commanding, monitoring and controlling a real physical system: a DC electrical motor. For this purpose we will use computational devices currently employed in modern control and automation systems, namely PC’s and microcontrollers. The information flowing in the control system (sensor, actuation or reference data) is transferred using the Robot Operating System (ROS) middleware, widely used in the Robotics academic community.

The dynamical system considered in this work consists in a DC motor instrumented with a potentiometer and a tachometer (see Figure 1) thus allowing the closed loop control of its position and velocity.

Figure 1 – The DC motor from Quanser™

The embedded system interfacing directly with the physical device will be an Arduino Duemilanove platform, a board with an ATMega 328 processor and interfacing electronics (Figure 2), equipped with an additional Ethernet communications card (Figure 3).

Figure 2 – The Arduino board with ATMega 328

Figure 3 – The Arduino Ethernet Shield

The project will be developed in the Linux operating system. Notwithstanding, it will make use of cross-platform tools and libraries that allow its usage, or facilitate its porting, to other operating systems.

2 Modelling and Controlling the Physical System

This project involves using the Arduino device, which is connect to the host PC by an USB connection, as well as ethernet provided by an EthernetShield daughter board for the Arduino (see ref, [ArduinoEthernetShield]). The physical device attached to one of the Arduino's IO channels is a DC-motor, which is driven by a power amplifier (see Figure 4). In this work we will focus on controlling the angular velocity of the DC motor. Coupled to the DC-motor is a speed sensor, tachometer, whose readings are performed by the one of the Arduino's analog-to-digital channels. The actuation of the DC-motor is done through one of the PWM channels of the Arduino.
The programming of the Arduino board is done through a USB connection (see ref. [ArduinoLib]). The Arduino language, called “Wiring”, is very similar to the C language. The application development environment (Arduino IDE) runs on the PC attached to the Arduino with the USB connection. It compiles the written programs using a cross-compiler (avr-gcc) and downloads them to the attached Arduino.

### 2.1 Dynamical Model of the Physical System

The next figure identifies schematically the DC motor and the most important electric and mechanical, constants and variables.

![Electromechanical model of the DC motor.](image)

Description of the variables:

- $V_a$ - Input voltage [V];
- $I_a$ - Armature current [A];
- $R_a$ - Armature resistance [$\Omega$];
- $L_a$ - Armature inductance [H];
- $\theta$ - Angle of the axis of the motor [rad];
- $J$ - Inertial moment of the rotor and the load of the motor [Kg.m$^2$];
$K_p \cdot \omega$  - Counter electromotive force [V]. $\omega = \frac{d}{dt} \theta$;
$T_m$  - Motor torque [N.m];
$\beta$  - Viscous damping coefficient at the rotor's support points [N.m.rad$^{-1}$.s];

The dynamics of the system can be represented by the following set of equations:

$$
\begin{align*}
V_a(t) &= R_a I_a(t) + L_a \dot{I}_a(t) + K_p \dot{\theta}(t) \\
T_m(t) &= K_m I_a(t) \\
J \ddot{\theta}(t) &= T_m(t) - \beta \dot{\theta}(t)
\end{align*}
$$

where the first equation is the mesh law of the inducted circuit, the second equation describes the proportional relation between the motor torque and the armature current and finally the third equation balances the angular moments (Newton's law for rotating systems).

### 2.2 Controller Specifications

The controller to design has the purpose of controlling the rotation speed of the motor. More precisely, the microcontroller generates Pulse Width Modulation (PWM) actuation signals given the rotation speed measurement acquired by the tachometer mounted on the motor.

The specifications desired for the controlled system are the following:

- Zero steady state error on the rotation speed for a step reference signal.
- Overshoot less than 20%.
- Settling time about 20ms.

It is suggested to study proportional and proportional-integral continuous controllers, verifying which controllers allow fulfilling, or not, the desired specifications. After selecting one continuous controller, it has to be converted to the discrete time domain by a Tustin (bilinear) transformation, and tuned experimentally for the real system.

### 3 The Distributed Architecture

For controlling, commanding and monitoring our physical system, a modular distributed approach will be taken into account. We will consider three main components (see Figure 5):

- A user interface application to run on a general purpose PC, consisting in one console for manually commanding (turn on, turn off, setting operating mode and parameters, etc.) and monitoring the setup (check state variables, system statistics). This process is not required to have high reliability or real-time functionality, but should provide easy interface for commanding and monitoring the system by a human operator. We will denote this application the Commanding and Monitoring Console (CMC).
- A high-level process-controller responsible for generation of trajectory references (position and/or velocity) to be executed by the DC motor low-level controller. This process will communicate upstream with the CMC using Inter-Process Communication (IPC), and with the low-level process controller in the embedded system using the USB link. This application must be reliable because any malfunction may generate irregular control trajectories in the physical system. We will denote this system the Communications and Control Server (CCS).
• A low level process controller running in the embedded system. This process must implement the low-level dynamical controller of DC motor using a high sampling rate to ensure a stable control and good reference tracking. Reference values coming from the CCS at a lower frame rate will be used to compute the control values at each sampling time. This must the module of highest reliability because system stability depends critically on its appropriate operation.

Each of the three main components has specific functionalities to be detailed in the following sections.

3.1 Commanding and Monitoring Console (CMC)

The commanding and monitoring console architecture component is an autonomous application running on a general purpose desktop PC, which in essence provides a user interface to the setup. A ROS node should be developed to communicate directly to the CCS, using ROS services over the network. This application should encompass two main tasks:

**Task 1** – to monitors the physical process state (e.g. speed of the motor and the actuation signal) and show them to the user on a periodic basis. The display prints in the screen the state of the system every half second (2Hz).

**Task 2** – to read the commands set by the user and send them to the CCS. It is responsible for reading the keyboard and writing the commands to a communications port using ROS. The user input is based on the keyboard. Options must be given to set constant reference values, ways to increment / decrement the current reference value, and ways to specify the parameters sinusoidal / square / saw-tooth or triangle waves, to be generated in the CCS.
3.2 **Communications- and Control- Server (CCS)**

This is an application running on the dedicated real-time PC that manages all the communications. This application links to the CMC using ROS, and links to the embedded system with real-time communications over the USB link. Thus, in normal operation, this ROS node has the following tasks:

**Task 1** – An event-driven process receiving asynchronous commands from the commanding and monitoring console using ROS, and storing these commands internally.

**Task 2** – A periodic process sending status-data to the commanding and monitoring console from internal memory. In normal operation the node should send status-data twice per second (2Hz).

**Task 3** – A periodic process, for interfacing with the embedded system. It will also generate the references to the low-level control system, at a sampling rate about 10x lower than the embedded controller sampling rate. In each cycle, this process should perform the following functions:

- Check the most recent commands from the CMC and update its state accordingly (internal reference generator).
- Send the control references and configuration commands to the embedded system using the USB link.
- Get the state of the low-level controller (position, velocity, reference, control value) and store it internally.

The CCS is responsible for generating reference signals and sending configuration commands to the Arduino. Reference signals can be constant values, square waves or sine waves. The sensor (tachometer) readings associated to the DC-motor and obtained by the Arduino, are available at the CCS, eventually sub-sampled, and can be forwarded to the CMC or any other GUI. Due to efficiency reasons, the CCS should not have heavy graphical components.

3.3 **Low-level Dynamical Controller**

The low-level dynamical controller is responsible for stabilizing and tracking according to the specifications the references provided by the CCS. It will implement a linear feedback controller as illustrated in the following figure.

![Feedback Controller Diagram](image)

The controller consists in an application running on the Arduino platform which has direct access to the physical system (sensor and motor actuator). It uses a USB connection for both development and communication with the controller PC, and it also has an additional Ethernet card for networking. This application synchronizes (defines an upper bound for) the sampling rate of the control loop. It manages AD/DA conversions (sensor readings), closed control loop implementation
and PWM generation (motor actuation). Communications with the CCS are made using ROS over the USB link. The Arduino should additionally run a web server for diagnostic purposes, accessible through the Ethernet port. It should serve a simple web page containing detailed information about the controller.

The main tasks of the application to run on the Arduino are:

**Task 1** – Implement the control-loop: at each sampling time, read data from the sensors, compare with the current reference, compute the control value and actuate the motor.

**Task 2** – Interface with the CCS. Upon a CCS message, reply to the message: update reference value or respond with the current state of the controller.

**Task 3** – Simple web server displaying diagnostic information about the controller.

Since the Arduino’s operating system is single threaded, these two tasks must be manually interleaved. Care must be taken in the selection of the sampling rate to make sure all computations can be performed within a single sampling time.

4 **Main stages of the work**

All architectural components have largely overlapping underlying technologies (e.g. all modules have some kind of multi-processing or communications). Hence we re-organize the project into stages focusing on separate aspects. At the end of semester, the students will have covered the main techniques for developing efficient and versatile real-time distributed control systems.

The project will be developed in two main stages. The first one will be dedicated to setting up the software modules and communications between the CMC and the CCS. The focus will be on the development of applications and communications involving ROS. In the second stage, we will focus on modelling the physical system, designing a discrete time control and implement the ROS-based communications between the CCS and the embedded system.

4.1 **Stage 1 – Multi-tasking and IPC**

In this stage we consider multi-process programming and IP communications under ROS. We will develop three ROS nodes: the Commanding and Monitoring Console (CMC), the Communications and Control Server (CCS), and a Simulator of the Closed-loop Control System (SIM), to be implemented as ROS nodes and communicating through ROS. All applications will run on PC’s. The first application will perform the interface between the user and the control system, both for sending high level commands (starting and stopping the controller, defining set points and references, etc), and to visualize output data (controller state, system output, etc). The second application will implement the generation of control references interfacing with the embedded system. The process and its interface (Arduino microcontroller) will not be considered at this stage. Instead a third application will simulate these components by implementing a discrete recursive difference equation to emulate the behaviour of the closed loop controller and communicating with reliable communications. This simulator will be used just for testing the distributed software architecture and not the controller design. Thus, a simple first order approximation to the DC Motor can be implemented using a transfer function of the form:
\[ G(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \]

with parameters suitably chosen to mimic the required specifications for the control system:

- Zero steady state error on the rotation speed for a step reference signal.
- Overshoot less than 20%
- Settling time about 20ms.

Figure 7 – Architecture to implement in the first stage of the work
4.2 Stage 2 - Integration with the Physical System

In the second stage we will replace the dynamical system simulator module by the actual physical system. This will involve the development of the embedded system real-time discrete control law and communications to and from the CCS. The communications interface between the Arduino and the CCS will be performed using ROS over the USB link.

4.2.1 Guidelines for Controller Design

The embedded controller’s purpose is to stabilize DC motor and track the reference commands generated by the CCS with a performance as close as possible to specified requirements. This involves a careful modelling of the system dynamics (see section 2.1), the design of the digital control law and its implementation in the Arduino platform.

After modelling the physical system equations, we will have to identify some of its parameters. A simulator of the system can be designed in Simulink to facilitate the test of the developed controllers before their utilization in the real system.

4.2.2 System Identification and Development Methodology

In order to fulfill the controller specifications and the required programming, it is necessary to identify the motor parameters and characterize the PWM commanding. Given that the specifications focus on speed control, the DC-motor model can be simplified to include just a mechanical pole. Starting from the electromechanical model introduced earlier, if one considers negligible the armature inductance, La, and denotes by W(s) the Laplace transform of the rotation speed \( \omega(t) \), then the DC-motor can be modeled by the first order model:

\[
\frac{W(s)}{I_a(s)} = \frac{K_0}{1 + s \tau},
\]

where \( K_0 \) stands for the static gain of the motor, including the power amplifier, and \( \tau \) the time constant. Note that these constants can be estimated from a simple step response. Generating the step involves just changing the level of a digital output pin from Low to High.

Assuming that the PWM works at a frequency much higher than the frequency associated to mechanical pole of the DC-motor, than one can model as a gain the DC-motor commanding by the PWM:

\[
\frac{W(s)}{X_{PWM}(s)} = K_1 \frac{K_0}{1 + s \tau},
\]

where \( X_{PWM} \) denotes the PWM input. The PWM in the Arduino is set by the selection of an unsigned char value (0-255), where the lowest and maximum values imply the minimum and the maximum duty-cycles, respectively. Setting a constant input, \( x_{PWM}(t)=A \), i.e. a constant duty cycle square wave input, than the output \( \omega(t) \) gets constant at steady state, \( \omega(t \to \infty) = AK_1K_0 \). Constant \( K_1K_0 \) is the one to use for designing the controller.
In order to achieve a good controller design, we suggest the following sequence of steps:

- Obtain the full system’s model from physical laws.
- Implement a simulator in Simulink.
- Design and tune a controller to obtain the desired control specs in the simulation.
- Obtain the time and frequency responses that confirm the performance of the control system.
- Select and adequate sampling time and discretize the controller.
- Implement and test the digital controller’s performance with the simulator.

### 4.2.3 Controller Implementation

The implementation of the controller in a distributed environment has, as described before, an embedded programming component based on the Arduino USB device, and a host-PC based component encompassing the Commanding and Monitoring Console (CMC) and the Communications and Control Server (CCS).

The embedded programming on the Arduino actuates the DC-motor, by implementing a digital controller, \( C(z) \) given the sensor readings, \( y(n) \) and the reference signals sent by the CCS, \( r(n) \). The sensor readings made by the Arduino should be sent back to the CCS. In order to account for various controller forms, it is suggested to implement a two zeros and two poles digital controller:

\[
C(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}.
\]

The CCS should be allowed to configure the controller parameters \([a_0 \ a_1 \ a_2 \ b_1 \ b_2]\) used by the Arduino, and selecting whether or not closing the local loop. In other words, the CCS can select having the Arduino working in open loop, meaning that the reference signals \( r(n) \) should be sent directly to the DC-motor, and the sensor readings \( y(n) \) returned to the CCS.

The embedded program encompasses therefore two main working modes: (i) configuration mode, (ii) controller timed-loop. The configuration mode allows configuring the controller parameters, configuring the controller sampling period and starting the controller timed-loop mode. The configuration mode should also allow sending actuation values to PWM channels and receiving sensor readings from the analog-to-digital converters.

The controller timed-loop works in closed-loop, receives reference values from the CCS, sends actuation values to the DC-motor and sends sensor readings to the CCS. The timed-loop contains the computation of the difference equation associated with \( C(z) \). The timing of the loop is based on the Arduino's clock. In order to reduce the timing constraints associated to the communications, the reference readings and sensor reporting (from/to the CCS) should be allowed to run at a frequency lower than the controller loop by sub-sampling the loop runs that effectively communicate with the CCS. The timed-loop mode can be stopped with a CCS command. Stopping the loop returns the device to the configuration mode.

Associated to the implementation of the difference equation associated with \( C(z) \), it is important to include a saturation of the resulting actuation values. These values must be in the range accepted by the Arduino's PWM generator. Note that depending on the hardware and on the compiler, the default conversion of negative double values to unsigned integers can result in unexpected high values due to lacking saturation and just implementing the truncation of bits.
5 Concluding Remarks

Despite this manual suggests a certain functionality and operation of the overall control system, we welcome any ideas that may improve its performance and/or facilitate its implementation and testing. The students are strongly encouraged to propose new solutions that can prove better, in some aspect, to the ones proposed here.

5.1 Guidelines for Development

Test and debug incrementally - It is essential, for a steady and efficient progress, to sketch a plan for subdividing the work into elemental steps that can be tested and validated incrementally. For instance, it is not required to have the complete system coded before testing the individual parts. Incremental testing is a recommended technique in systems development. The students may feel more comfortable with other techniques, but it is essential to debug the code frequently, to discard nasty bugs early in the development, otherwise it will become more and more time consuming.

Visualize data – Either on the console terminal, or on a graphical user interface, visualize all information flowing through the system. Moreover, collecting data can be useful later for analysis. This will help both detecting bugs (even hidden ones that do not manifest on the overall system behaviour), as well as improving and tuning the code.

5.2 What to submit and reports

After the conclusion of stage 1 each group should demonstrate the operation of the developed work, according to the requirements. After stage 2, a report with less than 20 pages should be delivered.

Software delivery - At both stages, the developed software must be also delivered by email.

Final report notes - The final report describes the complete project in a succinct manner (max 20 pages). The report should include an introduction explaining the problem to address and the objectives of the project. The body sections should detail the dynamical system’s model, the computational architecture and the controller design (choose the order that best fits your style). A section with the results should both address the experiments performed to identify the system and evaluate the closed loop controller. Finally, a section with the conclusions should identify the main contributions and lessons learned from this work. If required, some extra sections may be added as appendices to describe more technical aspects of the work, e.g. the developed code, important classes, implementation details, etc.

6 References
