RT control with microcontroller

Microprocessor-based systems - Carlos Xavier Rosero

Lab1 - July 2017

1 Objective

The objective of this lab is the implementation of a microprocessor based real-time control system. The platform will permit to control an electronic *Double Integrator (DI)* circuit as shown in Figure 1, by properly coding a task on top of a real-time implementation.

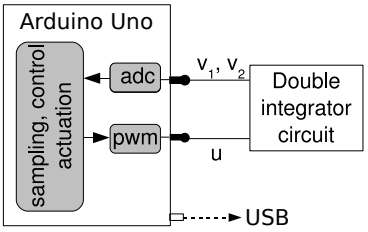

Figure 1: Microprocessor-based control.

The board is the Arduino UNO, equipped with a microcontroller ATMEGA328P (`https://www.arduino.cc/en/Main/ArduinoBoardUno`) from Arduino/Genuino. A daughter board with the DI electronic circuit, handmade by the student, has been plugged to the UNO board.

The interface between the microcontroller and the circuit to be controlled is the sensor and the actuator. The sensor is the analog/digital converter (ADC) to read the circuit output voltage V_1 and intermediate voltage V_2, and actuation is achieved through the pulse width modulator (PWM) by applying different voltage levels to the circuit input, u.

The control objective would be to have the circuit output voltage V_1 (controlled variable) to track a reference signal while meeting for example given transient response specifications, which mandates to use tracking structures. The control will be achieved by *sampling* V_1 and V_2, *executing* the control algorithm and *applying* the calculated control signal to the circuit via varying the circuit input voltage u (manipulated variable).

2 Control

A control system is a set of devices responsible for managing the behaviour of a dynamic system (plant), in order to reduce the probability of failure and obtain the desired results. A *feedback control system* is one that maintains a determined relationship between the output and the reference input, comparing them and using

the difference as a control means. An example would be the temperature control system of a room. By measuring the actual temperature and comparing it with the reference temperature (desired temperature), the thermostat activates or deactivates the heating or cooling equipment to ensure that the temperature of the room is kept at a comfortable level regardless of external conditions.

A PID controller is a feedback control mechanism widely used in industrial control systems. Like any controller, it calculates the deviation or error between a measured value and a desired value. The PID control algorithm consists of three different parameters: the *proportional*, the *integral*, and the *derivative*. The proportional value depends on the current error. Integral depends on past errors and the derivative is a prediction of future errors. The sum of these three actions is used to adjust the process by means of a control element.

When not aware of the process (plant) historically has been considered that the PID controller is the most appropriate. The controller performance can be described in terms of the control response to an error, the degree to which the controller exceeds the setpoint, and the degree of oscillation of the system. Note that the use of PID does not guarantee optimal control nor stability.

Even though there are two types of implementation of PID widely spread (standard and parallel), we will concentrate on using the parallel PID (see figure 2) of the form

$$\mathrm{u(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t) \tag{1}$$

where
K_p: Proportional gain, a tuning parameter
K_i: Integral gain, a tuning parameter
K_d: Derivative gain, a tuning parameter
e: Error $= r - y$
r: Setpoint or reference (what the user wants)
y: Output (current state)
t: Time or instantaneous time (the present)
τ: Variable of integration; takes on values from time 0 to the present t.

Equivalently, the transfer function in the Laplace domain of the PID controller is

$$U(s) = K_p + K_i/s + K_d s, \tag{2}$$

where s is the complex number frequency

2.1 Hints on discrete implementation of PID

A digital implementation of the parallel PID into a microcontroller requires the continuous form to be discretized using the Z-transform of the parallel form in (1). To simplify the discussion, because this subject does not concert digital control systems, we omit the effect of sampling period on the PID parameters. Also, this particular form is called *backguard euler*. Calculating the Z-transform we have

$$U(z) = [K_p + \frac{K_i}{1 - z^{-1}} + K_d(1 - z^{-1})]E(z) \tag{3}$$

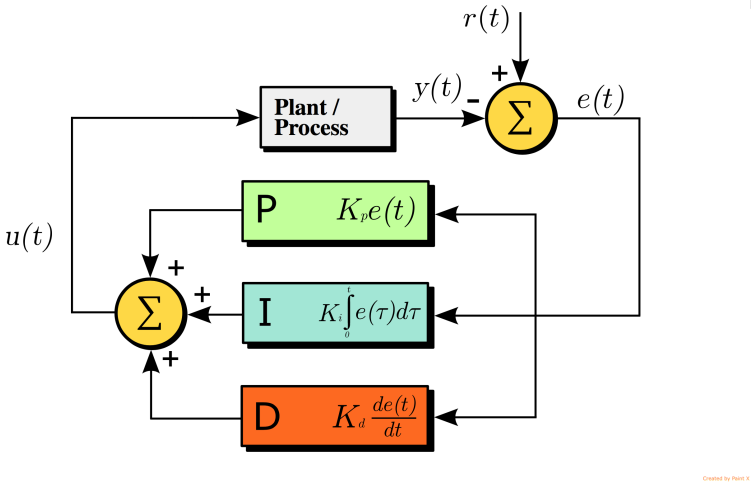

Figure 2: Parallel PID implementation

Rearranging the equation gives

$$U(z) = \frac{(K_p + K_i + K_d) + (-K_p - 2K_d)z^{-1} + K_d Z^{-2}}{1 - z^{-1}} E(z) \tag{4}$$

Defining

$$K_1 = K_p + K_i + K_d \tag{5}$$
$$K_2 = -K_p - 2K_d \tag{6}$$
$$K_3 = K_d \tag{7}$$

then $U(z)$ can be written as

$$U(z) - z^{-1}U(z) = [K_1 + K_2 z^{-1} + K_3 z^{-2}]E(z), \tag{8}$$

and finally, the resulting difference equation is given as

$$u(k) = u(k-1) + K_1 e(k) + K_2 e(k-1) + K_3 e(k-2) \tag{9}$$

which is a suitable form for implementation on a microcontroller, considering k as the number of sampling instant.

3 On the plant implementation

Before any controller implementation, the circuit of the plant must be realized following the diagram of figure 3. By knowing that the control input is u and the circuit output is $y = V_1$, pay attention to the two voltage dividers formed by the four resistors $R_3 = 1K\Omega$, and consider $R_1 = R_2 = 100K\Omega$ and $C_1 = C_2 = 420nF$. Both operational amplifiers are incorporated in one LM358 integrated circuit (keep in mind the power supply of this element).

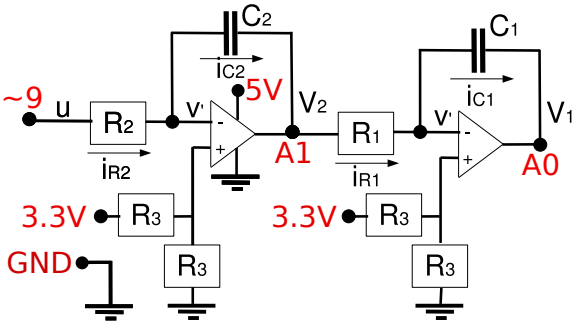

Figure 3: Double integrator implementation (in red are the Arduino UNO connections)

4 Testing the platform: orange led blinking

- Plug the UNO board to the computer using the USB cable.
- Open the Arduino IDE and once you are in the application, select $Tools -> Board$, and then $Tools -> Port$, in order to set both the Arduino UNO board as the certain serial port, respectively.
- Open the **.ino* script attached to this document and press *Compile* and then *Upload* buttons.
- The led should start blinking.

5 Open loop control

To start working with the *DI* circuit, a first approach is to observe how the system reacts to a given input, approach that we call **open loop control**. Using the *microprocessor-based control* architecture shown in Figure 1, we will apply a reference signal in the form of a square wave of amplitude $1V$ (from -0.5V to 0.5V) and frequency $2Hz$ to the plant input and will observe the plant output. Use the definitions named *topCountRef*, *refPos* and *refNeg*, to set the desired values in the Arduino program.

The observation will be in Matlab, executing the *interface.m* file that displays the data received through USB. This file will get the data sent from the board through the USB port, as it works as an "oscilloscope".

Below, a brief description of the various subroutines implemented in the microcontroller is presented:

- ISR(TIMER2_OVF_vect): it is a interrupt service routine induced each sampling time by a timer overflow, contains

```
ISR(TIMER2_OVF_vect) // interrupt service routine
{
  TCNT2 = 0x63; // preload timer (change this value
```

```
    // to change sampling time)
  changeReference();
  sysTime = millis(); //count system time in milliseconds
  readStates();
  calculateControl();
  writeControl();
  sendSerial();
}
```

- void changeReference(void): it generates the square wave reference signal and blinks the orange led.
- void readStates(void): function that is used every time the plant output/s must be read (it reads the ADC).
- void calculateControl(void): function that codes the control algorithm; in open loop, it has the code

```
void calculateControl(void)
{
  u = r; //reference on the input
         //comment it once the control has been developed
  if (u > (v_max/2)) //output limiting, to adjust to the hardware used
  {
    u = v_max/2;
  }
  if (u < -(v_max/2))
  {
    u = -v_max/2;
  }
}
```

- void writeControl(void): it generates the certain PWM signal used as actuator.
- void sendSerial(void): function that sends information to the Matlab interface using indirect addressing via pointers.
- Meanwhile, open Matlab, and open the *interface.m* file in which care must be taken with the serial port. It has to be specified correctly, eg. *s=serial('/dev/ttyUSB0')* or *s=serial('/dev/ttyS0')* and so on (read the hints suggested at the file header).

The subroutine *calculateControl* will be the part requiring more work in future sections. In figure 4 the reference signal (r, green) and the control signal (u, red) are the same because of the code that specifies $u = r$. The DI response, in blue, is unstable as expected. Note that it saturates.

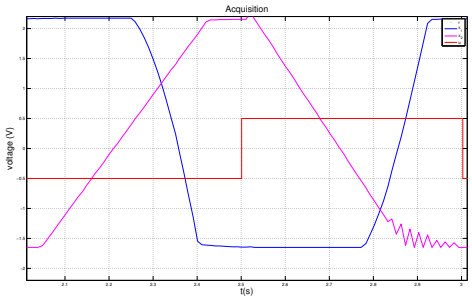

Figure 4: Open-loop response

6 Exercises on tuning PID

Before initializing the tests, implement the relevant code using the equations 5, 6, 7 and 9, in the subroutine named *calculateControl*. Do not forget to comment or erase the part that says $u = r$. Ensure a sampling time of 50ms by setting the preloaded value in register *TCNT2*.

In the following steps, the control signal u must be properly calculated. In particular, in the code shown next, question marks suggest places where the control algorithm must be coded.

```
void calculateControl(void)
{
  u=????; //Control law
  {
    u = v_max/2;
  }
  if (u < -(v_max/2))
  {
    u = -v_max/2;
  }
}
```

- **Proportional controller.** K_p reduces the rise time, increases the overshoot, and reduces the steady-state error. Plot the closed-loop step response using a proportional controller and setting K_p such that the state approaches the reference. Capture the graph of the experiment results. Indicate the values set in the constants.

- **Proportional-derivative controller.** K_d reduces both the overshoot and the settling time. Use the same K_p obtained from the last statement and set K_d such that the oscillations disappear and the overshoot is minimized. Capture the graph of the experiment results. Indicate the values set in the constants.

- **Proportional-integral controller** An integral controller K_i decreases the rise time, increases both the overshoot and the settling time, and eliminates

the steady-state error. Use the same K_p from proportional controller and set K_i such that the steady-state error is minimized and the settling time is admissible. Capture the graph of the experiment results. Indicate the values set in the constants.

- **Proportional-integral-derivative control.** Finally, after several trial and error runs, the gains K_p, K_i and K_d are already found. To confirm, put all gains together into a new controller and get the step response. In figure 6 is shown an example of the required final response.
- Test the proportional-integral-derivative control already obtained, but now using a sampling time of 100ms.

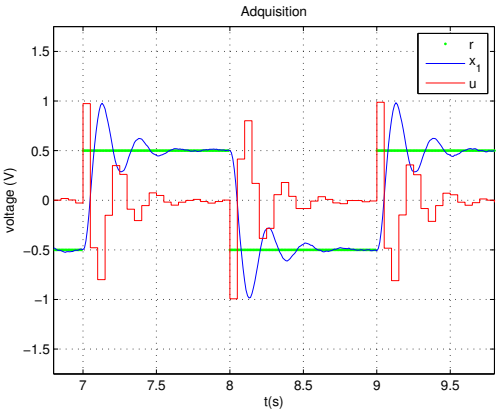

Figure 5: Example of final response

7 Work to be done

1. Get familiar with the setup: **orange led blinking** (see section 4)
2. **Open loop control**: In order to implement any microprocessor-based controllers, the setup shown in Figure 1 must be ready (see section 5)
3. **Exercises on tuning PID**: Implement the discrete-time controllers assuming that the error is $e = r - V_1$. (see section 6)

8 Deadline

- Write a short report giving answer to 1, 2 and 3. Do not forget to attach the arduino files and everything that shows the correct accomplishment of the tasks.
- The work can be done in couples (only 2 people, no more!).
- Deadline: July 27, 21h00, to `cxrosero@utn.edu.ec`.