

TP3 - Arquitectura de computadores

Kevin Giribuela

Febrero 2022

Índice

1. Introducción	3
2. Objetivo	3
3. Desarrollo	4
3.1. Hardware	4
3.1.1. Encoder rotativo	5
3.1.2. Display LCD 16x2	6
3.1.3. ADC	6
3.1.4. PWM	6
3.2. Software	7
3.2.1. Encoder rotativo	7
3.2.2. Display LCD 16x2	8
3.2.3. Identificación de planta - Adquisidor	9
3.2.4. Controlador PID	12
3.3. Performance de control	13
3.4. Espacio en memoria	14
4. Conclusiones	15
5. Apéndice A	16
5.1. Código de MATLAB	16
5.2. Ruido	17
5.3. Coeficientes de los filtros	18
5.3.1. Filtro del PID	18
5.3.2. Filtro a la referencia	19

1. Introducción

En el presente informe se desarrolla y explica el funcionamiento de un controlador PID y también el proceso de adquisición de datos para la identificación de una planta. El controlador es implementado mediante el microcontrolador *ATMEGA328P* utilizando la plataforma *Arduino nano*.

El sistema embebido cuenta con un display LCD de 16x2 y un encoder rotativo que permite al usuario navegar en el menú principal. La placa cuenta con un LED piloto y otro LED para visualizar en tiempo real la salida de la planta.

Para la implementación del sistema embebido se ha desarrollado en el software libre *KiCAD* el esquemático y PCB del controlador y la planta, los archivos se encuentran anexados junto con el comprimido que contiene el presente informe. Se trata de una placa simple faz de $10x15cm^2$ donde se encuentra implementada la planta a controlar, las fuentes de ruido de entrada/salida de la misma, el display LCD y el encoder rotativo dispuesto de una manera comfortable para el usuario.

Debido a que el proyecto consiste en llevar a cabo dos tareas no simultáneas, se realiza un único código que permita realizar tales tareas utilizando la misma placa.

2. Objetivo

El principal objetivo es diseñar un controlador PID de sistemas con retardo para una planta de orden 6 la cual es modelada como una de primer orden más un tiempo muerto. De esta forma, el controlador es diseñado para controlar este último modelo.

Sin embargo, también se lleva a cabo la tarea de identificación de otra planta (la cual es una modificación de la planta a la que se le diseña el controlador), de esta forma el esquema de implementación es el de la Figura 1, donde P1, P2, y P3 son tres plantas con diferentes constantes de tiempo (cada una difiere en al menos 10 veces de la otra), y las plantas $\frac{1}{(s + a)^3}$ y $\frac{1}{(s + a)^2}$ tienen la misma dinámica debido al polo ubicado en $-a$.

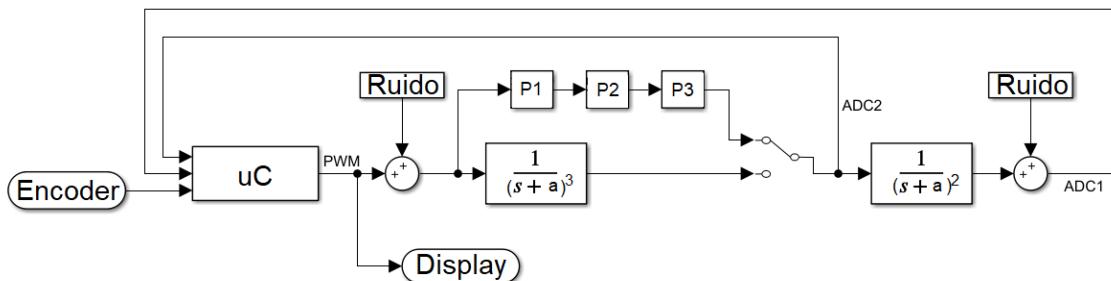


Figura 1: Esquema de lazo cerrado.

De la imagen anterior, queda en evidencia el uso de dos canales ADC del microcontrolador. Al mismo tiempo, una salida PWM haciendo las veces de acción de control de la planta. La idea de trasfondo es que a partir de los datos del ADC1, el microcontrolador implemente un algoritmo de control y actúe en la entrada de la planta variando el valor medio de la señal de PWM.

El usuario puede a través del encoder rotativo y el display desplazarse entre las diferentes opciones que presenta el menú principal.

Por otra parte, el canal 2 del ADC es utilizado para el proceso de adquisición de datos para la posterior identificación de la planta utilizando MATLAB.

3. Desarrollo

3.1. Hardware

Para llevar a cabo ambas tareas (controlador/adquisición de datos) se implementa un sistema embebido en una placa de $10 \times 15 \text{ cm}^2$ (los detalles de diseño no serán considerados en el presente informe). Del microcontrolador se hace uso del canal ADC1 para obtener los datos que utiliza el controlador y el ADC2 para la adquisición datos utilizados para la identificación de la planta. También se hace uso de los dos puertos de interrupciones externas INT0 e INT1 para la detección de giro y botón del encoder respectivamente. Finalmente, se utiliza parte del puerto C y parte del puerto B para el envío de datos al display.

A continuación se observa una imagen del sistema embebido.

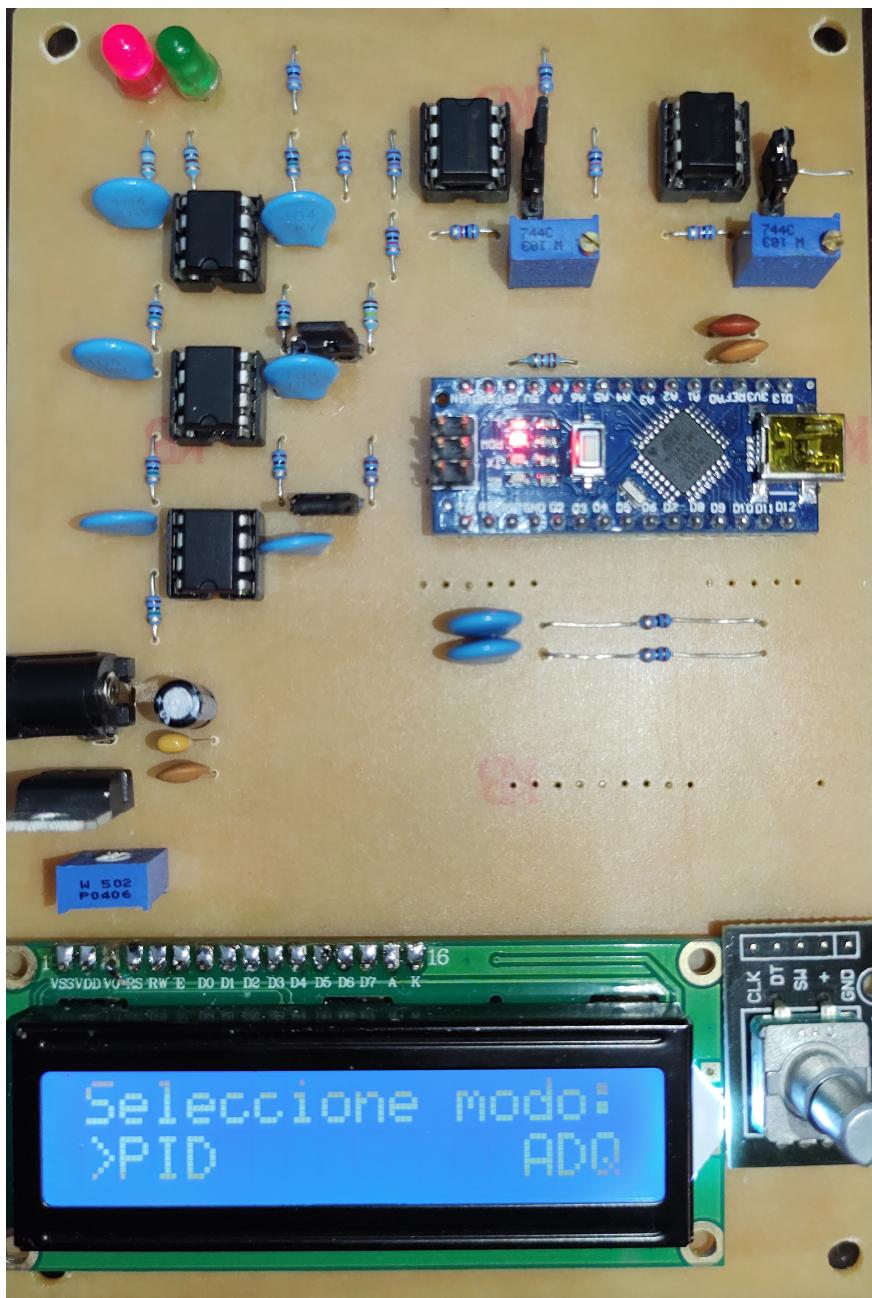


Figura 2: Sistema embebido.

3.1.1. Encoder rotativo

Debido a que el usuario debe interactuar con el sistema embebido, se hace uso del encoder rotativo para navegar en el menú del programa. El mismo permite modificar los parámetros críticos del controlador y el modo de funcionamiento en el que se encuentre el microcontrolador (adquisición de datos o controlador PID). Para el conexionado del encoder se implementa el circuito de la Figura 3.

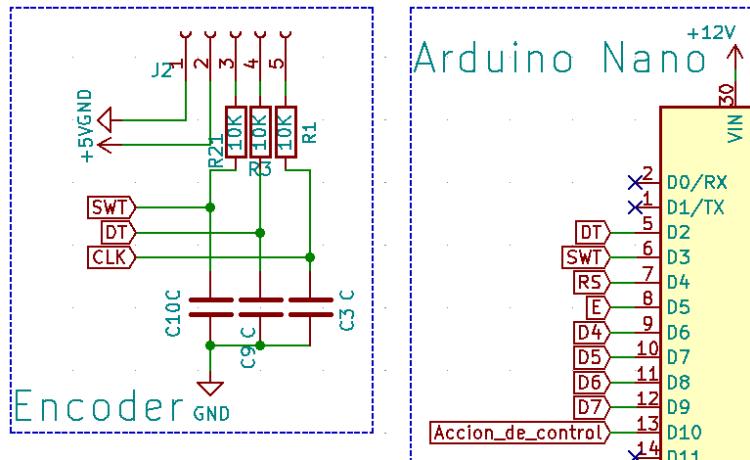


Figura 3: Conexionado del encoder.

Si bien se utiliza el módulo KY-040 (módulo que cuenta con las resistencias correspondientes de los pines DT y CLK), se agrega un filtro pasabajos a la entrada de cada pin del microcontrolador para evitar posibles rebotes presentes a la hora de girar el potenciómetro o presionar el pulsador. Se obtiene así un sistema de anti-rebote implementado por hardware, liberando al microcontrolador de aplicar una rutina de *debounce* por software.

Por otra parte, se utilizan como interrupciones externas los pines DT y SWT del encoder mientras que el pin CLK es utilizado como entrada digital, de esta forma es posible detectar el sentido de giro y la acción del pulsador de manera independiente, es decir, sin estar continuamente revisando si los pines han cambiado de estado lógico. Si bien no es recomendable el uso de pulsadores en interrupciones externas debido a su comportamiento errático con los rebotes, la implementación del filtro pasa bajos soluciona este inconveniente permitiendo de esta forma que el microcontrolador atienda las interrupciones externas cuando sea necesario y no estar continuamente revisando entradas digitales.

La lógica implementada para obtener el sentido de giro del encoder es sencilla: cuando el pin DT genere una interrupción, se activa una bandera que compruebe el estado lógico del pin A0 (PORTC0) y a partir de la siguiente tabla de verdad se determina el sentido de rotación.

DT	CLK	GIRO
Falling Edge	"1"	Horario
Falling Edge	"0"	Anti Horario

Figura 4: Tabla de verdad

Finalmente, el pin SWT se conecta al pin D3 (PORTD3) para ser utilizado como interrupción externa también.

Tal botón, es utilizado para implementar la acción de *enter* en el menú.

3.1.2. Display LCD 16x2

Como se ha mencionado, el usuario puede interactuar con el programa a través del display. Tal dispositivo tiene 2 modos de funcionamiento, 8 y 4 bits respectivamente. Con el fin de ahorrar pines de los puertos del microcontrolador se implementa en su versión de 4bits. Para ello se conectan los pines RS, E, y D4,...,D7 a los puertos GPIO D4,...,D9 del kit, mientras que el pin R/W del display va a masa (*write only*) y los restantes quedan flotando. Por otra parte, se coloca una resistencia de 100Ω en el ánodo del backligh para limitar la corriente y así no dañar al display.

Para finalizar el conenxiado del display se debe conectar al pin 3 del mismo un preset el cuál hará las veces de regulador para ajustar el contraste del mismo.

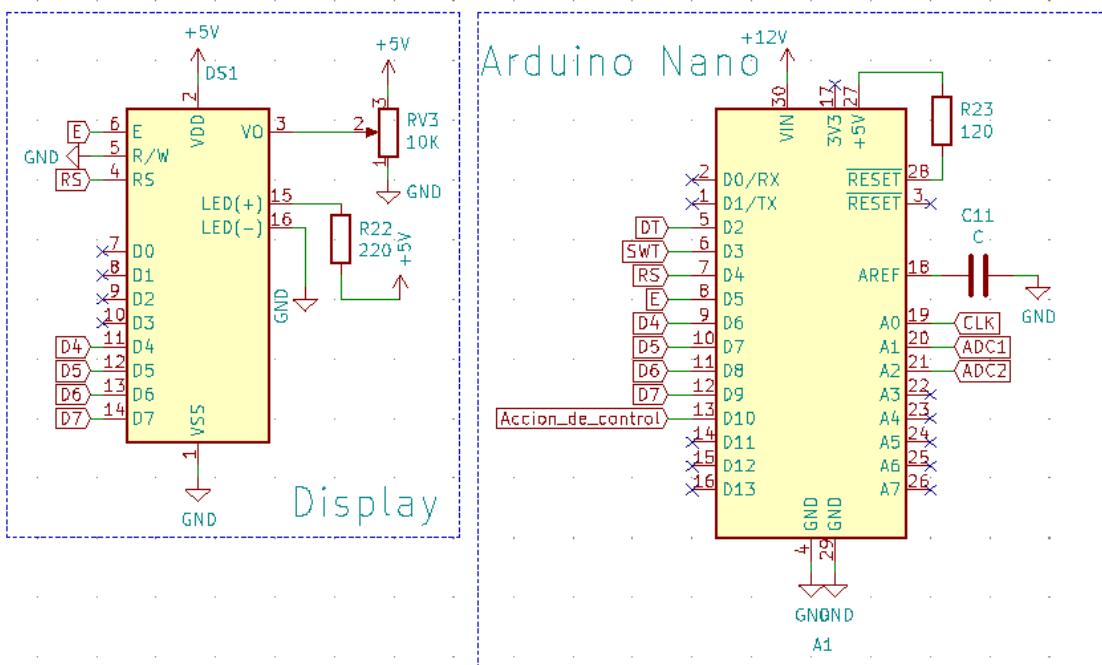


Figura 5: Conexi o del display.

3.1.3. ADC

Para leer los valores de la salida de las plantas a tratar, se hace uso de dos canales analógicos, AD1 y AD2. El primero es utilizado para muestrear la salida de la planta de orden seis, mientras que el segundo se utiliza para muestrear la salida de la planta de orden tres. Si bien no se requiere mucho circuito extra, en la hoja de datos del microcontrolador se sugiere colocar un capacitor entre tierra y el pin AREF para evitar posibles perturbaciones en la lectura del ADC. En la Figura 5 puede observarse tal conexión en el pin 18.

3.1.4. PWM

Por último, se observa en la Figura 5 que el pin utilizado para actuar como acción de control mediante una señal PWM es el pin D10 del Arduino Nano o, equivalentemente, el puerto PORTB2 del ATMEGA328P.

3.2. Software

3.2.1. Encoder rotativo

Para obtener una lectura correcta del sentido de giro del encoder rotativo, se conecta el pin DT al pin de interrupción externa INT0 y se configura el puerto digital PC0 (A0 en el kit) como entrada digital para la señal CLK. De esta forma el sentido de giro se detecta leyendo el valor del puerto PC0 en el momento en que la interrupción externa debido a DT ocurre.

En resumen, se tiene lo siguiente:

- Flanco descendiente de DT y CLK==0, entonces gira a la derecha.
- Flanco descendiente de DT y CLK==1, entonces gira a la izquierda.

La Figura 6 muestra el razonamiento previo.

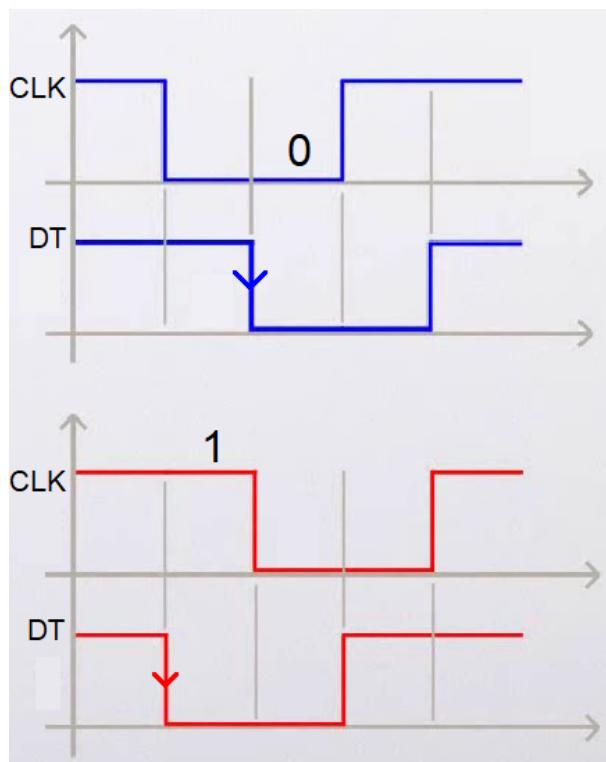


Figura 6: Detección de giro

Sin embargo, dado que el encoder excita dos veces el estado lógico del pin CLK, hay un giro que no es detectado, esto genera que se deba correr dos veces el vástagos del encoder para detectar un giro. Este problema es solucionado cambiando el tipo de evento que genera la interrupción externa al modo *Any logical change*, de esta forma, cualquier cambio lógico en el pin DT generará una interrupción y se ejecutará la rutina de detección de giro correspondiente. La tabla de la Figura 4 queda ahora como la tabla de la Figura 7.

En cuanto al pulsador, el mismo es conectado al pin PD3 del puerto D (INT1). Esto permite detectar al pulsador siempre que se lo presione sin necesidad de estar corroborando continuamente su estado.

Para generar el efecto de desplazamiento dentro del menú de selección, cada vez que se produzca una interrupción por el giro del encoder, se activa un flag (evitando permanecer

DT	CLK	GIRO
"0"	"0"	Horario
"0"	"1"	Anti-horario
"1"	"0"	Anti-horario
"1"	"1"	Horario

Figura 7: Tabla de verdad actualizada

mucho tiempo dentro de la rutina de interrupción). Tal flag provoca que el *despachador* llame a la función `char rotation (void)` para que se encargue de la detección del sentido de giro, y muestre en display el desplazamiento generado. La función `rotation()` devuelve '+' o '-' indicando el sentido giro *horario* u *antihorario* respectivamente.

Finalmente, la interrupción externa INT1 provocada por el pulsador, siguiendo el mismo razonamiento anterior, hace las veces de *enter* en el menú.

Para la configuración de las interrupciones externas se hace uso de los registros EICRA y EIMSK, los cuales se modificaron con el objetivo de setear el tipo de detección en la interrupción, y habilitar las mismas, a saber:

```

1 // Any logical change on INT0 generates an interrupt request.
2 EICRA&=(1<<ISC01) ;
3 EICRA|= (1<<ISC00) ;
4
5 // The falling edge of INT1 generates an interrupt request.
6 EICRA|= (1<<ISC11) ;
7 EICRA&=(1<<ISC10) ;
8
9 // Interrupt INT1/0 enable
10 EIMSK|= (1<<INT1) ;
11 EIMSK|= (1<<INT0) ;

```

3.2.2. Display LCD 16x2

Dado que el display es configurado en el modo de 4bits, es necesario realizar varias configuraciones previas para inicializar correctamente al mismo en el modo deseado.

En primer lugar se debe reasignar los pines de datos y control de display para que coincidan con el diseño del PCB, esto es:

- RS: PORTD4
- E: PORTD5
- D4: PORTD6
- D5: PORTD7
- D6: PORTB0
- D7: PORTB1

Como los datos que se deben enviar son de 8 bits pero solo se utilizan 4, los mismos serán enviados en 2 etapas, comenzando por el nibble alto y luego por el bajo. Para realizar tal tarea, a los puertos D y B se le deben asignar correctamente los valores correspondientes

a los pines de entrada del display. Para ello, se utiliza la siguiente lógica

- Nibble alto:

```
1 PORTB = (PORTB&0xFC) | ((cmd>>6)&(0x03)) ; // PORTB = 0bxxxx xxD7D6
2 PORTD = (PORTD&0x3F) | ((cmd<<2)&(0xC0)) ; // PORTD = 0bD5D4xx xxxx
```

- Nibble bajo:

```
1 PORTB=(PORTB&0xFC) | ((cmd>>2)&(0x03)) ; // PORTB = 0bxxxx xxD3D2
2 PORTD=(PORTD&0x3F) | ((cmd<<6)&(0xC0)) ; // PORTD = 0bD1D0xx xxxx ...
```

donde `cmd` es el comando que se desea enviar al display. Es importante resaltar que el hecho de haber reasignado los pines de esta forma, evita cualquier conflicto posible en la escritura de datos al display ya que los demás bits de cada puerto quedan inalterados.

De la hoja de datos del HD44780U se obtienen pasos de inicialización necesarios (ver Figura 8).

De esta forma, ya queda definida la secuencia en la cual debe inicializarse el display y los tiempos requeridos por el mismo.

Debido que el programa realizará dos tareas bien distinguidas (adquisición de datos y controlador PID) el display ofrece a su inicio dos opciones, las cuales el usuario debe seleccionar por medio del encoder, a saber:

- Adquisidor de datos (ADQ).
- Controlador PID (PID).

Una vez seleccionada la opción, el microcontrolador ejecuta el código correspondiente a la tarea seleccionada.

3.2.3. Identificación de planta - Adquisidor

Este es el más sencillo de los dos modos de funcionamiento disponibles, en este modo el microcontrolador se encarga de muestrear y transmitir por el puerto serie los valores de tensión leídos a la salida de la planta N°2. Para ello se hace uso del ADC2 y el puerto serie.

En primer lugar, se inicializa el puerto serie y el ADC en el canal 2. Luego, el programa queda a la espera de que MATLAB le envíe por el puerto serie el número 1 indicando el inicio de la adquisición de datos, por otra parte, el usuario cuenta con la opción de volver para *atrás* si así lo desea y volver al menú de selección de modo antes de que MATLAB envíe la señal. Una vez que se reciba el número 1 por el puerto serie, se pone en alto el pin PORTB2 del puerto B simulando de esta forma un salto unitario a la entrada de la planta, luego se entra en un bucle en el que cada vez que el ADC cuente con una nueva muestra, se ingrese a la función correspondiente de envío de datos. El proceso de toma de datos continuará de manera indefinida hasta que MATLAB envíe nuevamente el número

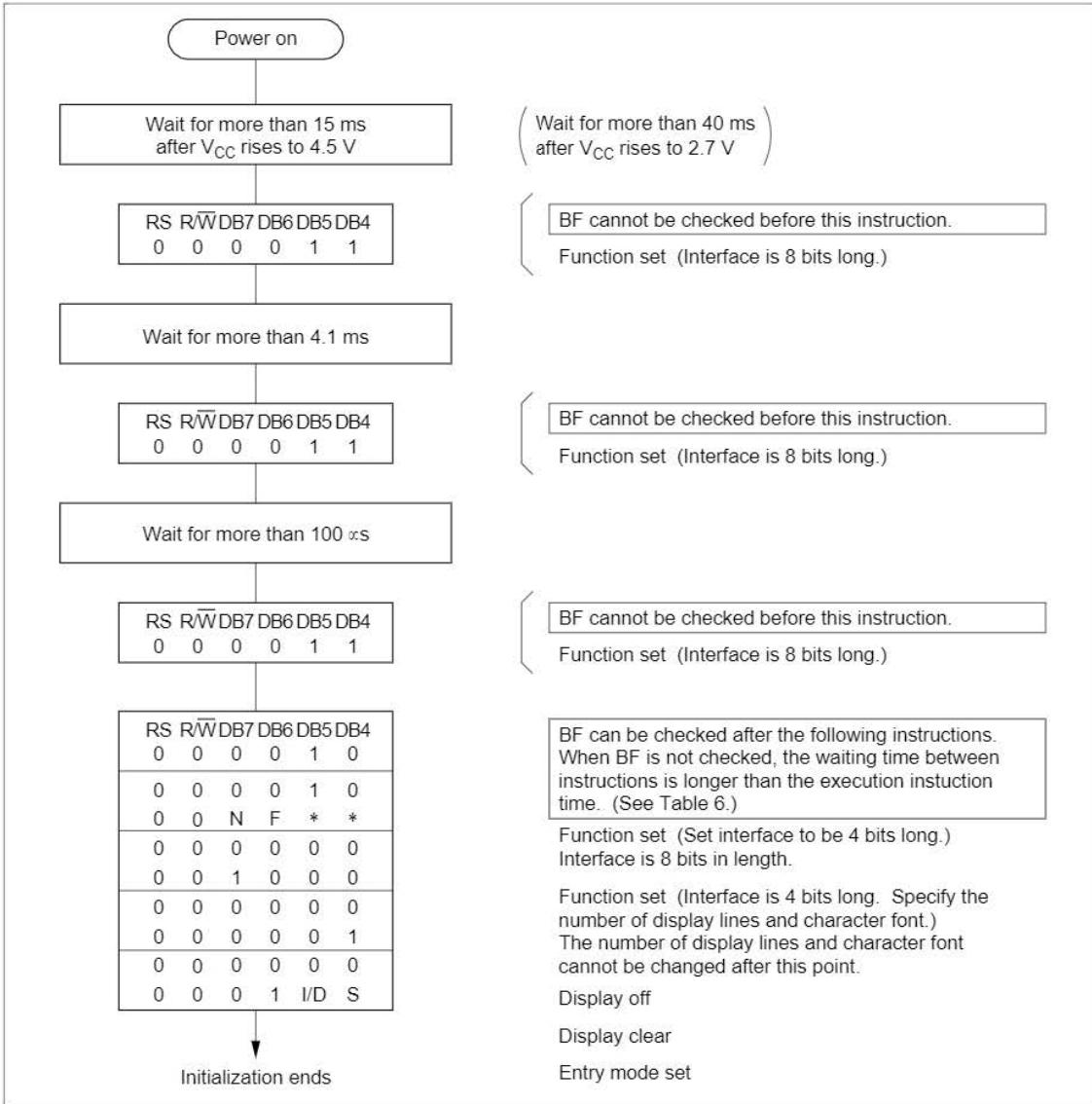


Figura 8: Secuencia de inicialización

- Al recibir tal valor, el microcontrolador finaliza la tarea, apaga los periféricos, y regresa al menú de selección de modo.

La frecuencia de muestreo del ADC es de 325 *samp/sec*. La tarea de temporización es llevada a cabo por el Timer0. Se adopta el modo *Auto trigger* para el ADC, donde una conversión se iniciará cada vez que el Timer0 active la interrupción de *Output on compare match*. De esta forma se obtiene un muestreo uniforme de la salida de la planta.

Si bien el numero de muestras por segundos resulta relativamente elevada para la planta si el tiempo de adquisición es alto, la restricción de la frecuencia de muestreo es debido a la dinámica del polo más rápido de una de las plantas, a saber 1000 *rad/seg* o equivalentemente $f \approx 159,15\text{Hz}$. De allí, resulta que la frecuencia de muestreo debe ser $f_s \geq 2 \cdot f_N = 2 \cdot 159,15\text{Hz} = 318,3\text{Hz}$, donde f_N es la *frecuencia de Nyquist*, entonces $f_s = 320\text{Hz}$.

Para configurar f_s se presetea el valor del registro OCR0A del Timer0 utilizando la siguiente expresión

$$f_s = \frac{f_{osc}}{\text{prescaler} \cdot (1 + OCR0A)} \rightarrow OCR0A = \frac{16\text{MHz}}{1024 \cdot 320\text{Hz}} - 1 \approx 48,8$$

De esta manera, fijando el registro OCR0A en 47, se obtiene $f_s \approx 325Hz$. Esto último implica que cada vez que el Timer0 llegue al valor de 47, se resetee el timer, se active una interrupción por comparación y el ADC muestre la salida de la planta.

El tiempo durante el cual el microcontrolador se encuentre adquiriendo de muestras queda definido por el tiempo que se haya definido en MATLAB. Para más información acerca del script utilizado en MATLAB ver el apéndice A

Para la transmisión de datos se configura al puerto serie modificando los registros correspondientes al mismo. La configuración adoptada es la siguiente:

- Modo: Asincrónico.
- Paridad: No.
- Bit de parada: 1 bit.
- Tamaño de datos: 8 bits.
- Baud rate: 9600 baudios.

Debido a que las plataformas de Arduino fueron diseñadas de manera que al conectarse al puerto serie el kit se autoreinicie, es necesario colocar una resistencia entre Vcc y RST para mantener la pin de reset a pull-up en todo momento. Esto provoca que para programar el kit, el programador deba resetear a mano el kit. En caso de no agregar tal resistencia, el kit se reiniciará cada vez que MATLAB se conecte por el puerto serie al mismo.

La Figura 9 es un resumen del funcionamiento descripto previamente.

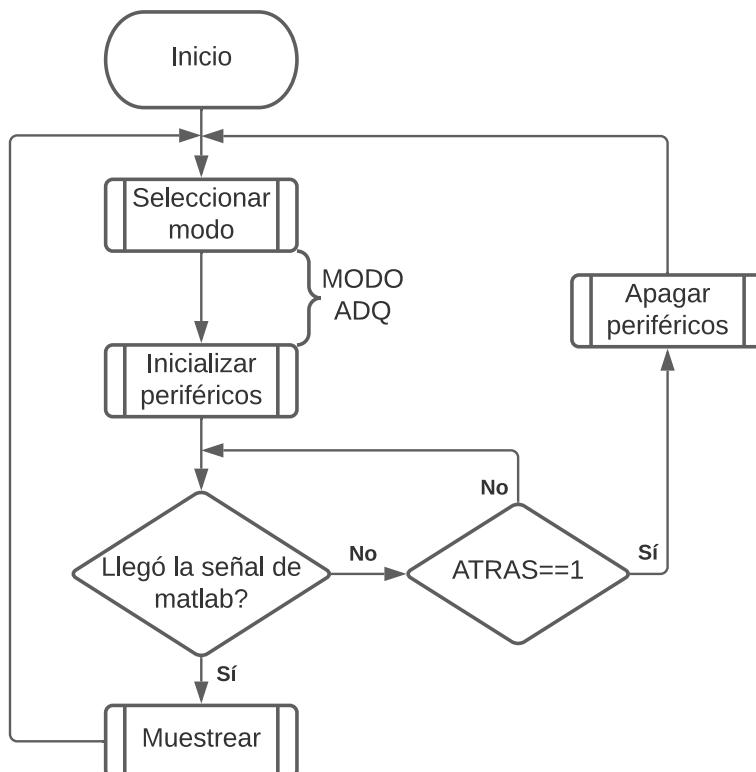


Figura 9: Diagrama de flujo. Modo ADQ

3.2.4. Controlador PID

Debido a que el microcontrolador no cuenta con una unidad DAC, la acción de control se implementa a través de un PWM, para ello se hace uso del Timer1. Dicho temporizador cuenta con diferentes modos de funcionamiento, entre ellos, el de *fast PWM*. El modo *fast PWM* permite obtener a la salida de un determinado pin una señal PWM de manera sencilla.

La idea es setear la frecuencia del PWM fijando el valor del registro ICR1, y luego variando el valor del registro OCR1A cambiar el ciclo de trabajo del mismo. La frecuencia del PWM se calcula como

$$ICR1 = \frac{F_CPU}{N * PWM_freq} - 1 \quad (1)$$

de esta forma, con una frecuencia de $1kHz$ se obtiene un valor de $ICR1 = 15,999$.

Usando los registros TCCR1B/A se configura al Timer1 para que funcione en el modo *fast PWM*, se le da el valor de ICR1 al TOP del Timer1, y se pone a 0 el pin OC1B cada vez que el registro TCNT1 llegue al valor del registro OCR1A (*Clear output on compare match*).

Con el objetivo de generar un muestreo uniforme, se utiliza la misma configuración del ADC adoptada para el modo de adquisición de datos utilizando el Timer0 como disparador de conversión (con la diferencia de que el canal utilizado es el canal 1).

Una vez seleccionado el modo PID, el usuario debe elegir qué controlador utilizar para llevar acabo el control de la planta. Para ello se utiliza el encoder para desplazarse entre las diferentes opciones. Elegido el controlador, el menú de selección de los valores de los parámetros aparece en pantalla. En él se elige el valor de los parámetros claves del controlador seleccionado. Al igual que para el modo adquisidor de datos, el usuario cuenta con la opción de regresar para atrás las veces que lo deseé.

Una vez seleccionado el controlador y sus parámetros, se debe presionar la opción *OK*. Acto seguido, se calculan los coeficientes de los respectivos filtros en función de los valores del PID y se entra en un bucle que queda en continua observación del encoder y actualización de ciclo de trabajo de la acción de control.

Para actualizar el ciclo de trabajo del PWM, se utiliza la función `void PWM_duty(float)` la cuál se le pasa como parámetro el valor que retorna otra función llamada `float accion_de_control(void)`. La primer función simplemente multiplica a la Ecuación 1 por el valor que retorna la segunda función dividido 5 y se lo asigna al registro OCR1B, es decir:

$$OCR1B = ICR1 * \frac{\text{retorno de accion de control}}{5}$$

asignar el valor anterior al registro OCR1B provoca que el Timer1 ponga a 0 el pin OC1B del microcontrolador en tiempos diferentes cambiando así el *duty cycle* de la señal.

En cuanto a la segunda función, en ella se calcula el valor de la acción de control según el valor de los parámetros del controlador elegido en función de el error actual y el error anterior.

Debido a que se requiere precisión a la hora de definir los parámetros, es necesario el uso de variables de tipo `float`. Este tipo de variables otorgan una mayor precisión en la asignación de polos y coeficientes que una variable de tipo `int`, sin embargo, no hay que perder de vista que no se cuenta con una unidad de punto flotante y que los costos de realizar operaciones en este tipo de variables demandan mucho tiempo. Por otra parte, no hay perder de vista el espacio en memoria disponible, esta puede verse comprometida en caso de que el código del programa comience a elevar su complejidad y las variables de

tipo flotante comiencen a ser un factor clave debido la cantidad de tamaño que ocupan en memoria.

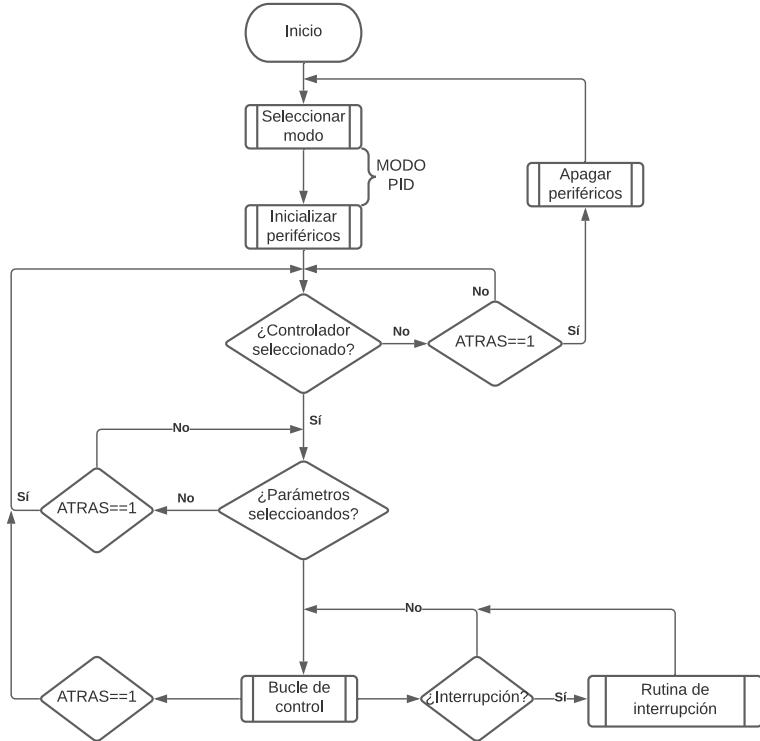


Figura 10: Diagrama de flujo del controlador PID

3.3. Performance de control

Como se ha mencionado, el algoritmo de control implementado debe utilizar variables de tipo flotante. La tarea que lleva a cabo el microcontrolador para calcular el valor de la acción de control cada vez que se obtenga una muestra es demandante para la arquitectura con la que fue desarrollado el mismo.

Con el fin de no demandarle demasiado computo en punto flotante al microcontrolador, una vez que el parámetro α es calculado, se asigna a una nueva variable de tipo flotante el valor del producto y divisiones de otras variables de tipo flotantes cuyo valor no varía si no α no cambia. De esta forma, el microcontrolador deberá realizar un solo producto en tipo flotante cada vez que se calcula la acción de control. Un razonamiento similar es aplicado para re-calcular el valor de los coeficientes de los filtros que se aplican.

Como es sabido, la ecuación en diferencias de un filtro con 4 coeficientes tiene la siguiente forma:

$$y[n] = \frac{1}{a_0} (b_0 x[n] + b_1 x[n - 1] - a_1 y[n - 1])$$

para reducir el costo computacional, se reasigna el valor de los coeficientes b_0 , b_1 , y a_1 de la siguiente forma

$$\begin{aligned} b_0 &= b_0/a_0 \\ b_1 &= b_1/a_0 \\ a_1 &= a_1/a_0 \end{aligned}$$

Esto último reduce la cantidad de productos en punto flotante de 6 a 3. Más aún, dado que $b_0 = b_1$ para el filtro del PID, la cantidad de multiplicaciones se puede reducir de 3 a 2.

3.4. Espacio en memoria

Para concluir con los detalles técnicos, se detalla la cantidad de memoria que consume el código. Si bien no se ha hecho tal análisis en TPs anteriores, en esta oportunidad el código utilizado consume mucho de los recursos del microcontrolador, al mismo tiempo que utiliza variables que ocupan mucho espacio en memoria.

La Tabla 1 muestra la información que arroja *Microchip Studio* una vez que compila y graba el programa.

Tabla 1: Resultados obtenidos.

Program Memory Usage	Data Memory Usage
11178 bytes \equiv %34,1	530 bytes \equiv %25,9

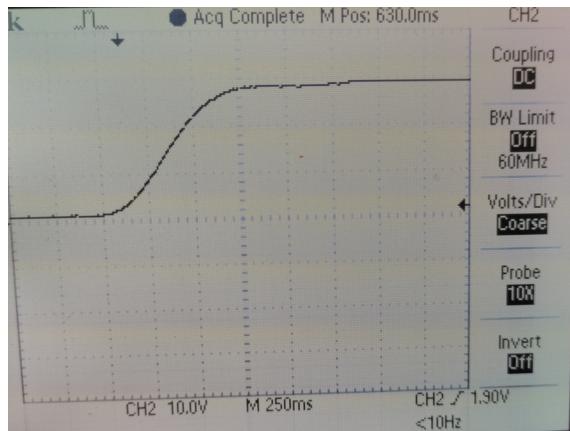
4. Conclusiones

Se ha atravesado a través del presente informe diferentes aspectos prácticos y teóricos que fueron necesarios enfrentar para poder obtener el sistema embebido final. Si bien no se ha mencionado, el diseño del esquemático y PCB han jugado un papel importante para el correcto funcionamiento de los periféricos del microcontrolador, entre ellos, sistemas anti-rebote, capacitores de filtrado de ruido, pull-ups externos a los pines de entrada, etc.

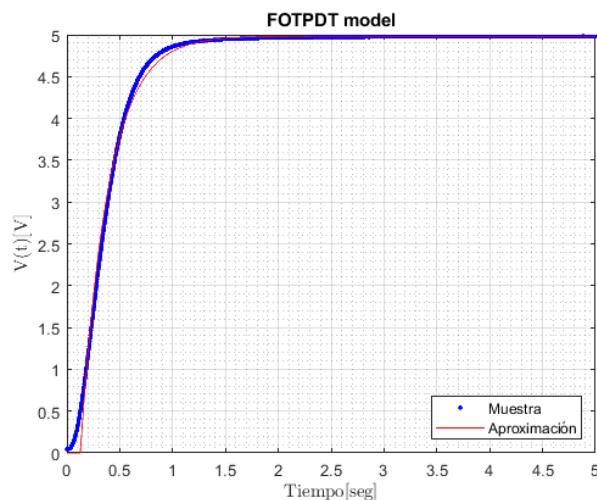
El resultado obtenido cumple con las especificaciones planteadas en un primer momento (controlador PID robusto, e identificador de planta). El código escrito para la implementación del controlador e identificador, se optimizó en medida de lo posible, aunque no se descarten posibles mejoras a futuro.

se ha visto que el espacio en memoria consumido por las variables utilizadas es casi del 26 %, un número que quizás no parece muy elevado, sin embargo, a medida que se desea actualizar el código, implementar nuevos controladores, cambiar un modo, etc, la memoria será un factor clave al momento de programar.

Para finalizar, se adjuntan dos imágenes del resultado final. En ellas se muestra por un lado la salida de la planta utilizando el controlador PID, y por otro los datos adquiridos del ADC junto con el modelo de primer orden utilizado para la identificación de la planta.



(a) Salida de la planta



(b) Muestras del ADC

Figura 11: Resultados obtenidos

5. Apéndice A

5.1. Código de MATLAB

Para el modo de adquisición de datos se ha utilizado el prograrma MATLAB. Se ha seleccionado dicho programa debido a las prestaciones que otorga, otra opción de software libre pudo haber sido Python.

El código utilizado para la toma de datos es el siguiente:

```
1 clear variables; clc; close all;
2
3 s = serialport("COM10", 9600);
4
5 tension=zeros;
6 i=1;
7
8 disp('Presione enter para comenzar la adquisicion de datos... ');
9 pause();
10 flush(s,"output");
11 write(s,1,"CHAR"); % Inicia adquisicion de datos
12 disp('Muestreando... ');
13 tic;
14 tf=0;
15 while tf<5
16     tension(1,i)=read(s,1,"UINT16"); % Leo el puerto serie.
17     i=i+1;
18     tf=toc;
19 end
20 flush(s,"output");
21 write(s,1,"CHAR"); % Finaliza adquisicion de datos
22 clear s;
23 disp('Adquisicion de datos finalizada.');
24 % Procesamiento de señal
25 tension=tension*5/1023; % Convierro a tensión.
26 t=linspace(0,5,length(tension));
27
28 % Grafico de respuesta al escalon
29 figure('Name','Datos adquiridos');
30 plot(t,tension,'r','LineWidth',2); grid on; grid minor;
31 title('Rta al escalón.');
32 xlabel('Tiempo[seg]', 'Interpreter', 'Latex');
33 ylabel('V(t) [V]', 'Interpreter', 'Latex');
34 legend('Muestras')
35
36 % Procesamiento FOPDT
37 figure('Name','FOPDT');
38 FT=fittype("Kp*(1-exp(-(x-L)/T)).*(x>L)");
39 FOPDT=fit(t',tension',FT, 'StartPoint',[0.01 0.01 0.01]);
40 plot(FOPDT,t',tension');grid on; grid minor;
41 title('FOTPDT model');
42 xlabel('Tiempo[seg]', 'Interpreter', 'Latex');
43 ylabel('V(t) [V]', 'Interpreter', 'Latex');
44 legend('Muestra', 'Aproximación', 'Location', 'SouthEast');
45 parametros=coeffvalues(FOPDT);
46 Kp1=parametros(1);
47 L1=parametros(2);
48 T1=parametros(3);
```

```

49
50 % Procesamiento SOPDT
51 figure('Name','SOPDT');
52 FT2=fittype("Kp*(1-exp(-(x-L)/T).*(1+(x-L)/T)).*(x>L)");
53 SOPDT=fit(t',tension',FT2,'StartPoint',[0.01 0.01 0.01]);
54 plot(SOPDT,t',tension');grid on; grid minor;
55 title('SOPDT model');
56 xlabel('Tiempo[seg]', 'Interpreter', 'Latex');
57 ylabel('V(t) [V]', 'Interpreter', 'Latex');
58 legend('Muestra', 'Aproximacion', 'Location', 'SouthEast');
59 parametros=coeffvalues(SOPDT);
60 Kp2=parametros(1);
61 L2=parametros(2);
62 T2=parametros(3);

```

El código anterior realiza la tarea de identificación del sistema y realiza el modelado de manera automática para sistemas FOPDT (First Order Plus Dead Time) y SOPDT (Second Order Plus Dead Time)

5.2. Ruido

Debido a que fue necesario simular ruido de a la entrada y a la salida de la planta, se ha implementado mediante unos amplificadores operacionales TL082 la resta de una tensión continua a la entrada y salida de la planta de manera independiente. El circuito implementado es el de la Figura 12

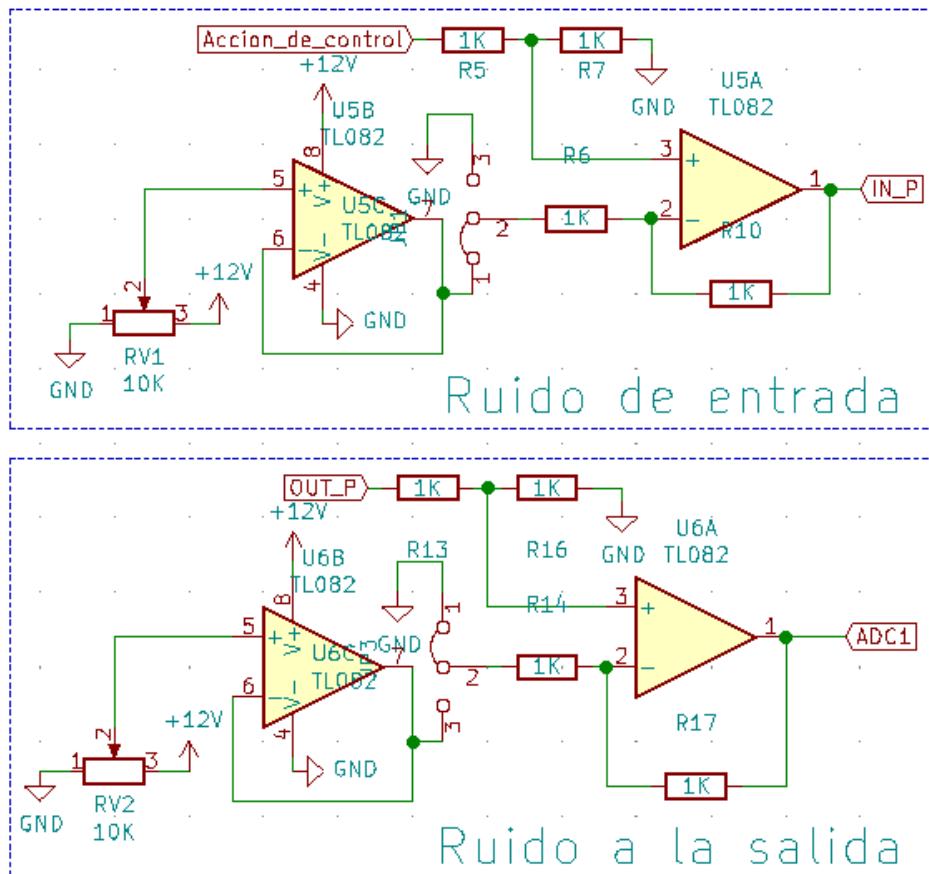


Figura 12: Circuito de ruido. En colaboración con Santiago Garaventta.

A pesar de que los operacionales son rail-to-rail, al aplicar una tensión de 5V a la entrada de un circuito seguidor, la tensión no llega jamas a 5V por el efecto del offset. Si se desea una respuesta aún mejor, debe pensarse en el uso de un operacional más costoso (por ejemplo el OP295).

5.3. Coeficientes de los filtros

Para poder implementar la acción de control de una manera correcta, fue necesario filtrar la referencia y la salida del PID paralelo. Tales filtros fueron obtenidos teóricamente y luego implementados en el bucle de control.

Debido a que se buscó una variación en tiempo real de los parámetros del controlador, los coeficientes de los filtros utilizados debían cambiar en función del valor de α seteado. Esto último provoca que tales coeficientes no sean variables constantes las cuales pueden ser definidas una sola vez durante el *setup*.

Como se dijo, contamos con dos filtros, uno para la salida del PID paralelo y otro para la referencia, se analizará primero el caso del filtro a la salida del PID y luego el filtro a la referencia.

5.3.1. Filtro del PID

La salida de un PID clásico viene dada por la siguiente Ecuación:

$$PID = K_C \left(error + \frac{1}{T_I} * \int error + T_D \frac{derror(t)}{dt} \right)$$

La Ecuación anterior no es propia, es por eso que se coloca un filtro pasa bajos a la salida para hacer propia la transferencia. El filtro a utilizar es

$$F_{PID}(s) = \frac{1}{1 + T_D * \alpha s}$$

como se lo quiere implementar en tiempo discreto, se aplica la transformación bilineal a la transferencia anterior utilizando la frecuencia de muestreo de 325Hz descripta en la Sección 3.2.3. El resultado obtenido es el siguiente:

$$F_{PID}(z) = K_F \frac{1 + z^{-1}}{1 - az^{-1}}$$

luego, anti-transformando...

$$y[n] = K_F (x[n] + x[n - 1]) + ay[n - 1]$$

y de esta forma se obtienen los coeficientes del filtro a la salida del PID donde

$$a = -\frac{T_s - 2T_D\alpha}{T_s + 2T_D\alpha}$$

$$K_F = \frac{T_s}{2T_D\alpha + T_s}$$

donde el tiempo derivativo es el tiempo en el controlador PID paralelo. La implementación del filtro se le deja como tarea al lector analizarla en la rutina `float accion_de_control(void)`.

5.3.2. Filtro a la referencia

Según los requisitos del controlador, fue necesario aplicarle un filtro a la referencia utilizada (la cual es constante e igual a 2). El filtro en el dominio de Laplace tiene la siguiente expresión:

$$F_{ref}(s) = \frac{1 + bT_I s + cT_I T_D s^2}{1 + T_I s + T_I T_D s^2}$$

esta vez la ecuación es propia, por lo que sólo se debe aplicar la transformación bilineal y antitransformar. Aplicando los pasos descriptos se obtiene la siguiente expresión para el filtro:

$$y[n] = \frac{1}{A^*} (Ax[n] + Bx[n - 1]Cx[n - 2] - B^*y[n - 1] - C^*y[n - 2])$$

donde

$$\begin{aligned} A &= T_s^2 + b2T_I T_s + c4T_I T_D \\ A^* &= T_s^2 + 2T_I T_s + 4T_I T_D \\ B &= 2T_s^2 - c8T_I T_D \\ B^* &= 2T_s^2 - 8T_I T_D \\ C &= T^2 - b2T_I T_s + c4T_I T_D \\ C^* &= T^2 - 2T_I T_s + 4T_I T_D \end{aligned}$$

Nuevamente, la implementación del filtro se puede encontrar en la función `float accion_de_control(void)`.