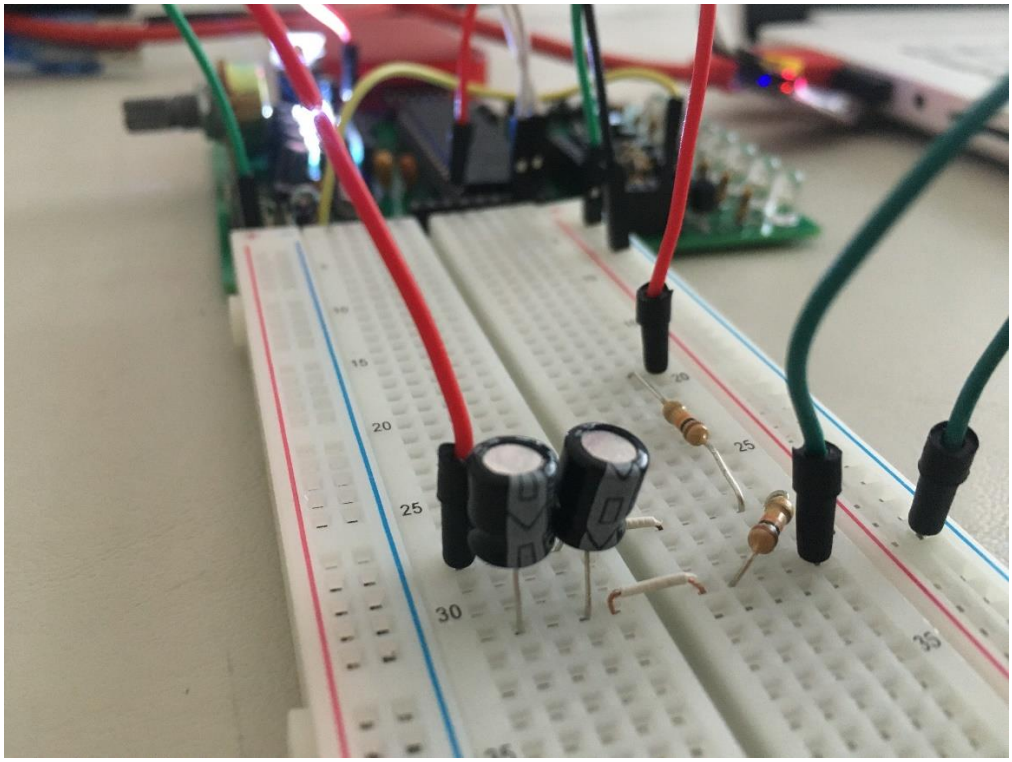
Universidad Católica Argentina

Facultad de Ingeniería y Ciencias Agrarias - Carrera de Ing. Electrónica

CONTROL AUTOMÁTICO



Registro	***Apellido y Nombre***
15-162138-5	Burs, Trinidad

CORRECCIÓN

Docente	***Fecha***	***Observaciones***
Vecchio, Ricardo		

APROBACIÓN:

Basándonos en la planta propuesta lo primero que se debe realizar es el cálculo de su transferencia:

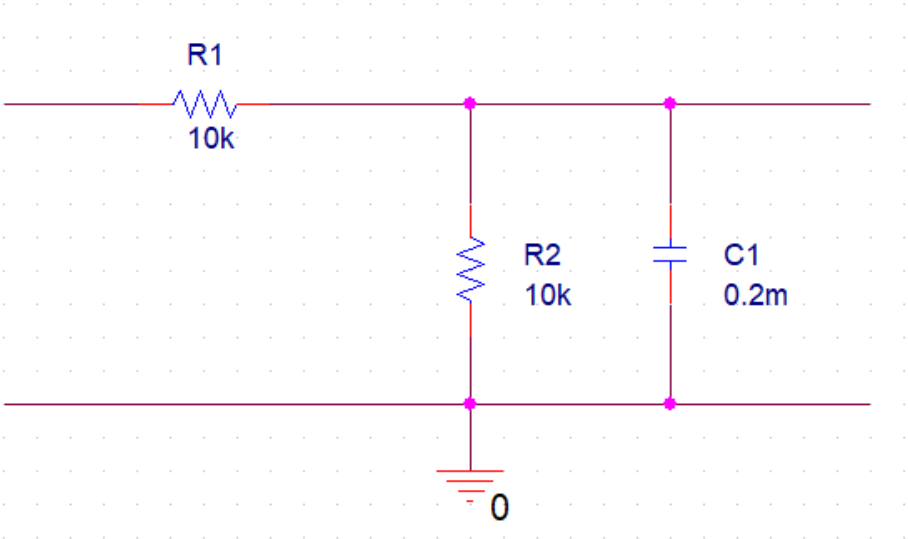


Imagen 1. Planta Propuesta

```
syms R1 R2 C s Vi
Zeq  = simplify(paralelo(R2,1/(s*C)));
Vo   = simplify(Zeq/(Zeq + R1) * Vi);
tf   = simplify(Vo/Vi)
aux1 = simplify(subs(tf,R1,10000));          %Reemplazo R1 por 10K
aux2 = simplify(subs(aux1,R2,10000));        %Reemplazo R2 por 10K
res  = simplify(subs(aux2,C,200*(10^(-6)))) %Reemplazo C por 200uF
```

```
tf =
 R2
-------------
R1 + R2 + C R1 R2 s

res =
   1
-------
2 (s + 1)
```

```
function [res]=paralelo(z1,z2)
    res=z1*z2/(z1+z2);
end
```

Imagen 2. Calculo de la transferencia de la planta utilizando MATLAB

Se obtuvo que su función transferencia es:

$$G(s) = \frac{1}{2\,(s+1)}$$

Observamos que tendrá un único polo. Si hubiésemos tenido un filtro de dos etapas, no solo tendríamos que considerar un segundo polo sino también como la segunda etapa carga a la primer. Para que la segunda etapa no cargue a la primera seria necesario conectar un OPAMP como buffer entre medio de ellas.

A su vez en este caso $H(s) = 1$ ya que no estamos trabajando con ningún sensor u otro dispositivo capaz de introducir un transitorio. Con estos datos podemos proceder a calcular el compensador. Para este cálculo utilizare una herramienta de MATLAB llamada “sisotool”.

Para diseñar el compensador se buscó que el settling time este lo más cerca posible a 2 seg (para que el esfuerzo de control no sea muy grande) y a su vez se intentó maximizar el valor de K para así disminuir el error a la rampa o a la parábola.

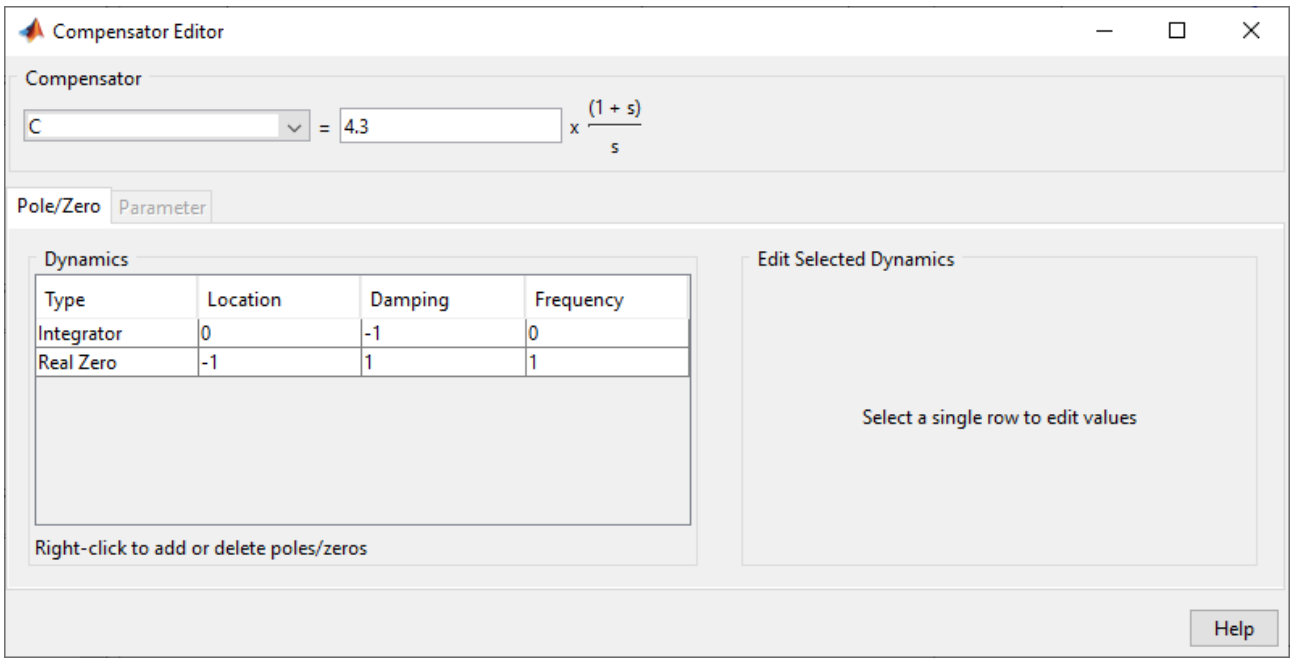


Imagen 3. Compensador

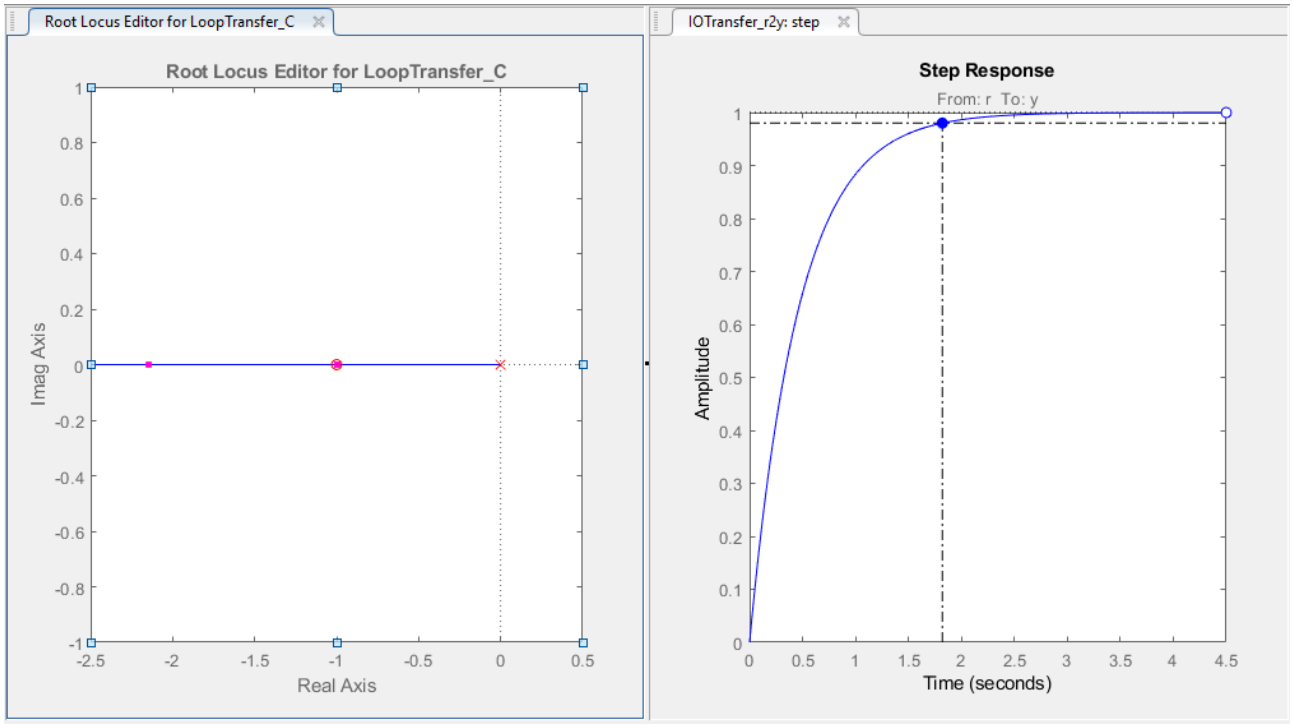


Imagen 4. Root Locus y Respuesta al escalón del sistema a lazo cerrado

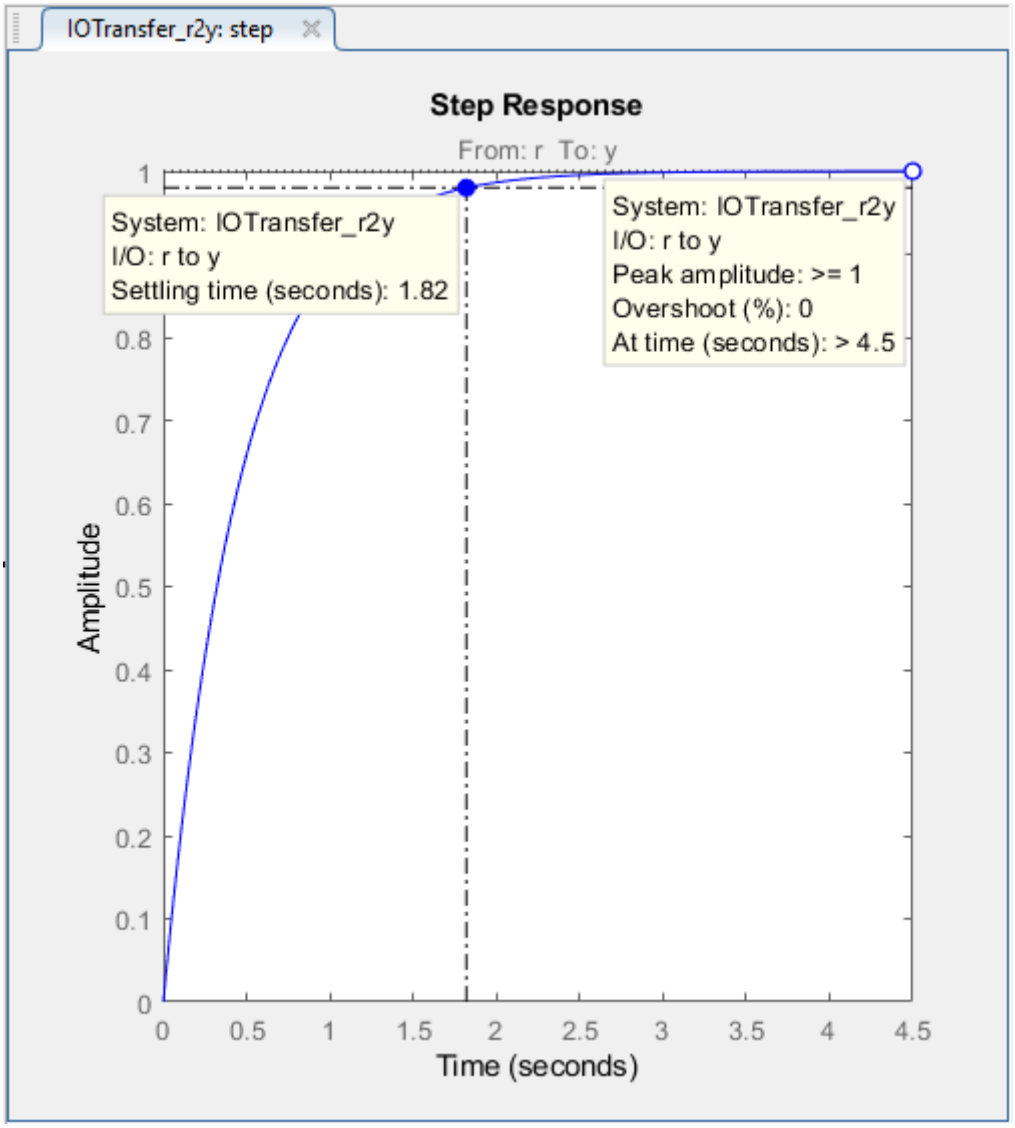


Imagen 5. Overshoot y Settling Time

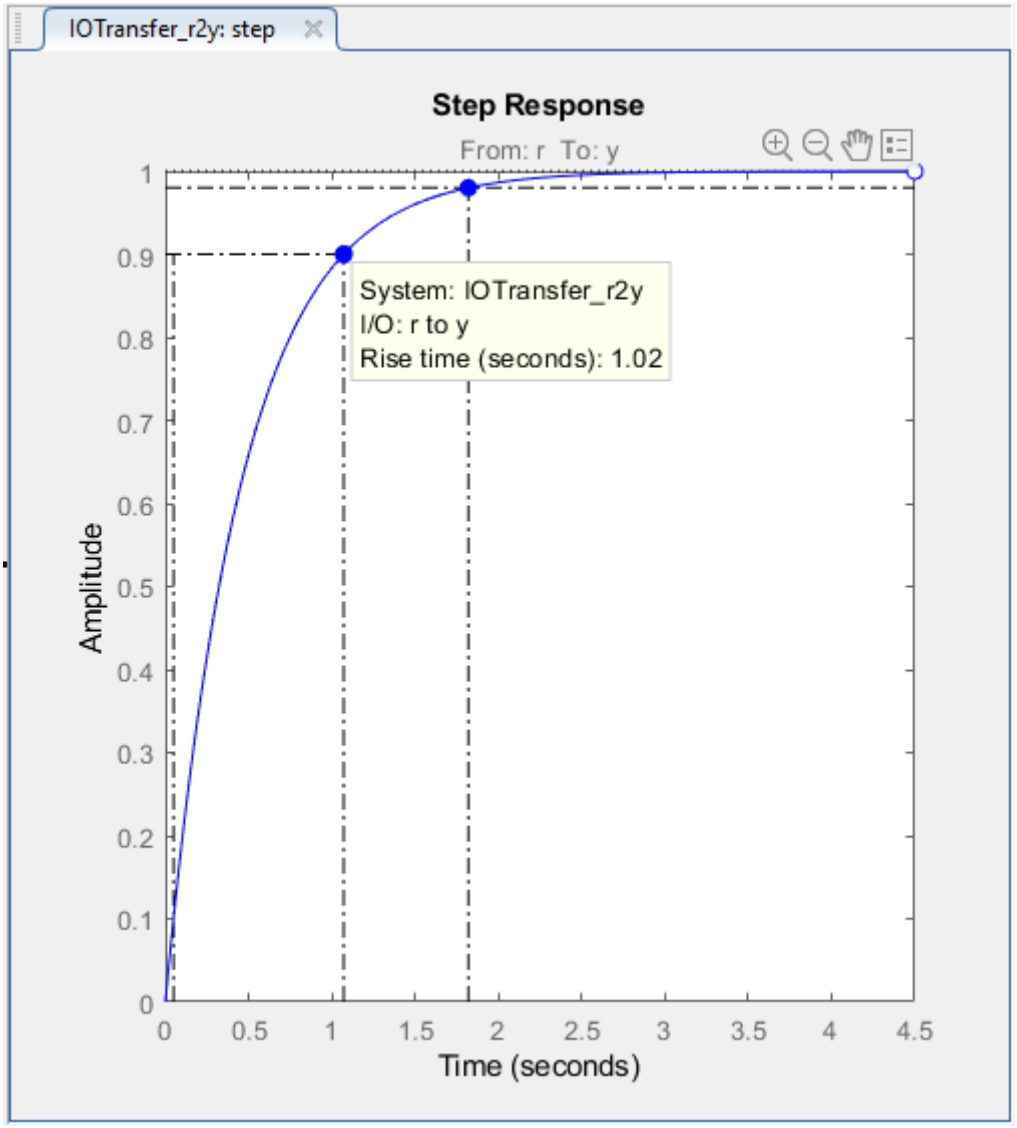


Imagen 6. Rise Time

Para implementar un compensador digital debo discretizarlo. Seleccionare un tiempo de muestreo de 0,017 segundos ya que será su $rise\_time/60$, de esta manera obtendré una buena aproximación.

La función transferencia de mi compensador será:

$$C(z) = \frac{4.3\,z - 4.2269}{z - 1}$$

El compensador lo implementare con el PIC18F4620 utilizando 8 bits para el conversor A/D (0V→0 cuentas, 5V→255 cuentas). Teniendo en cuenta esto hare la simulación:

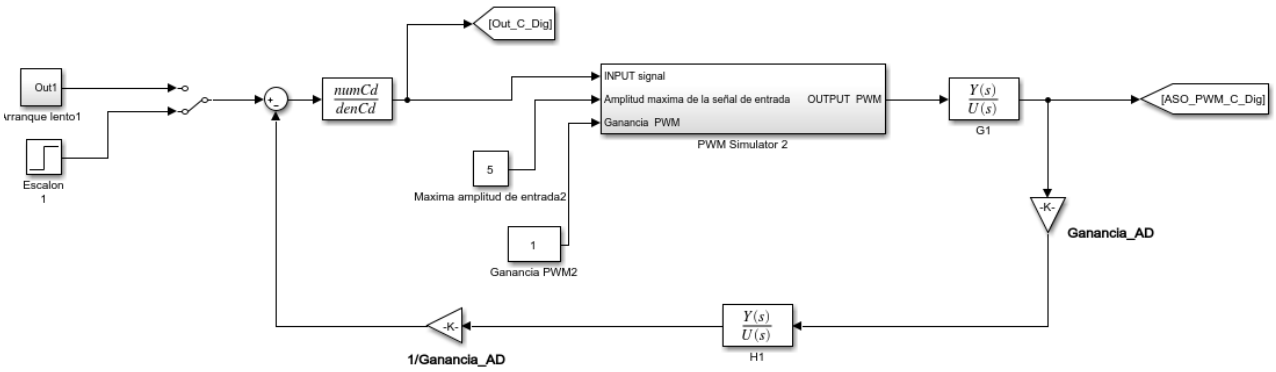


Imagen 7. Diagrama de mi lazo realimentado implementando compensador discreto

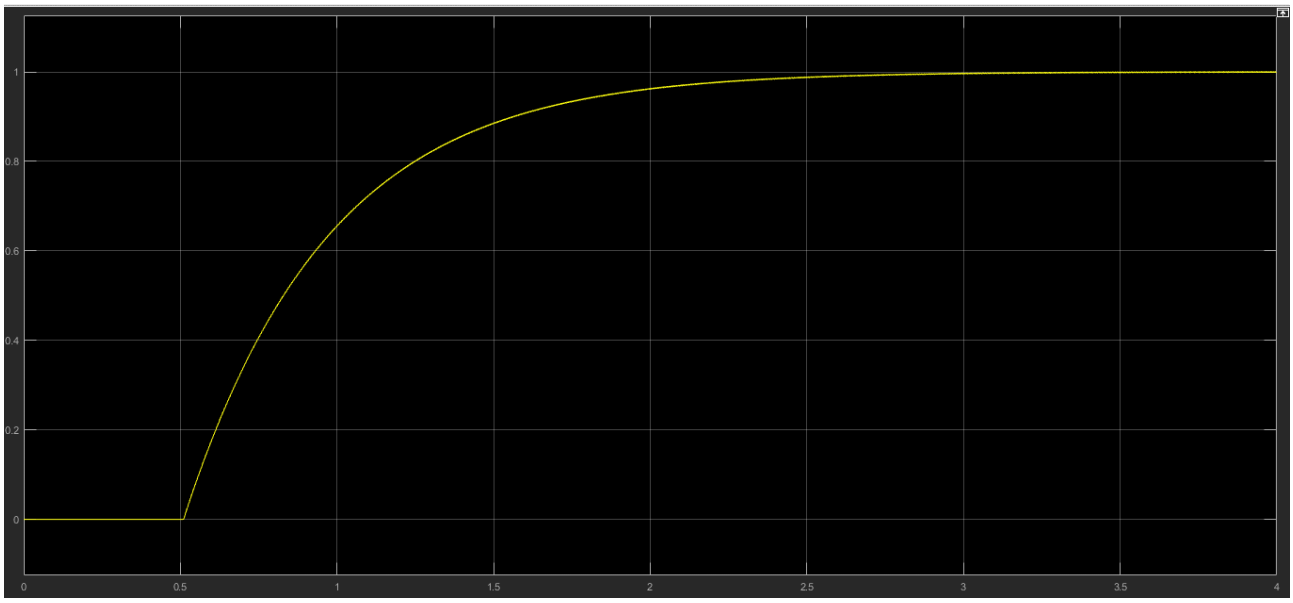


Imagen 8. Salida sistema Imagen 7 con escalón de entrada

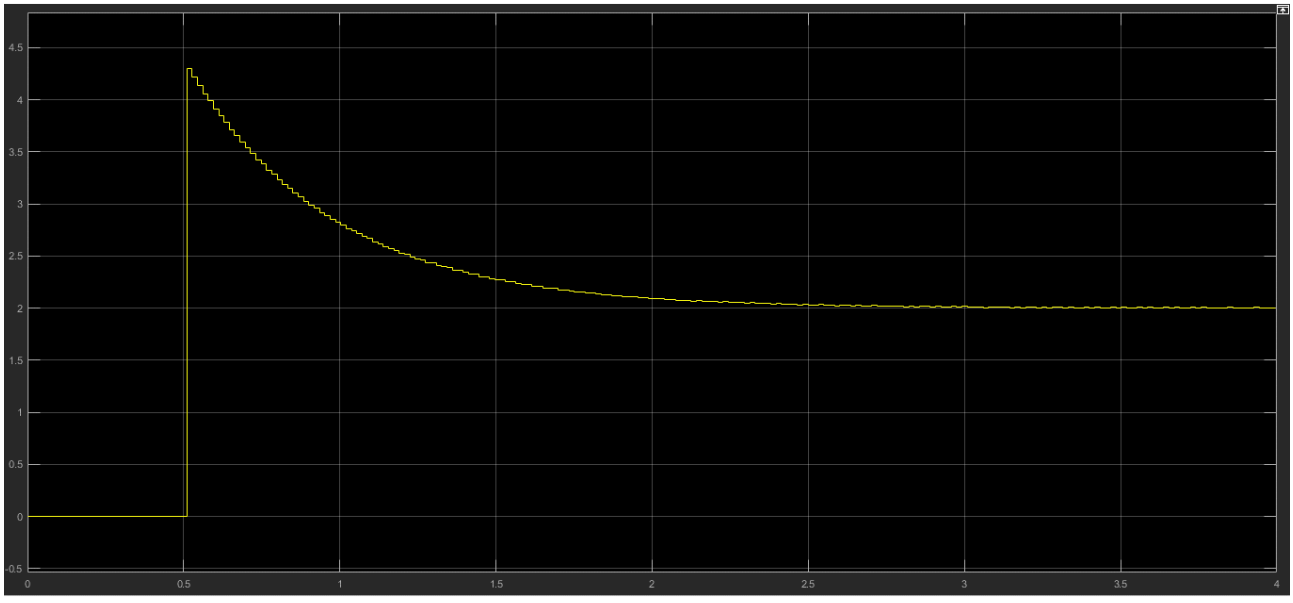


Imagen 9. Salida compensador Imagen 7 con escalón de entrada

Observamos que tanto la salida de la planta como el esfuerzo de control siempre estarán en el rango de 0V a 5V por lo que no será necesario adaptar la entrada ni la salida para evitar sobrecargar y quemar al PIC18F4620.

Para implementar el compensador debo escribirlo de la siguiente manera:

$$\frac{Y[n]}{X[n]} = \frac{4.3\,z - 4.2269}{z - 1}$$

$$\frac{Y[n]}{X[n]} = \frac{4.3\ - 4.2269\,z^{-1}}{1 - z^{-1}}$$

$$Y[n]\,.\,(\,1 - z^{-1}\,)\ =\ X[n].\,(\,4.3\ - 4.2269\,z^{-1}\,)$$

$$Y[n]\ -\ Y[n-1] =\ 4.3\,X[n] -\ 4.2269\,X[n-1]$$

$$Y[n]\ =\ 4.3\,X[n] -\ 4.2269\,X[n-1] +\ \ Y[n-1]$$

Al implementar el PWM es conveniente que la frecuencia de este sea un múltiplo entero de la frecuencia de muestreo, de lo contrario es como si muestreara el PWM todo el tiempo en lugares distintos. Como la planta es el sistema que filtra mi PWM, esto se manifiesta en su salida como un ripple que se desliza. Por esta razón elegí una frecuencia de 10KHz para el PWM (no debo pasarme de esta frecuencia ya que estoy trabajando con capacitores electrolíticos). A su vez debo asegurarme de que los capacitores que utilice trabajen como máximo al 60%-70% de su tensión indicada. Como voy a trabajar con 5V y la tensión máxima soportada por mis capacitores es 16V no tendré problemas.

$$f_{PWM} = 10\ KHz \qquad \therefore \qquad \frac{f_{PWM}}{1/T_s} = f_{PWM}.T_s = 170$$

Para que mi programa siga los lineamientos de la simulación debo asegurarme de tomar una muestra cada Ts. Para esto utilizare las interrupciones del TMR0. Como trabajo con una frecuencia de oscilación de 16MHz (es el cristal externo que tengo), mi Ts será mucho mayor que mi Tcy luego necesito utilizar el TMR0 en su modo de 16 bits y con un prescaler.

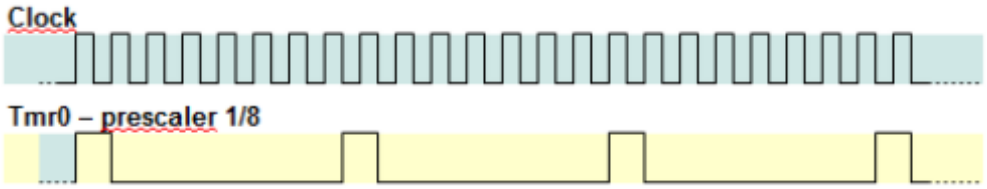


Imagen 10. Tcy y TMR0

Para imprimir la entrada de mi A/D por la UART armaré una función que convierte un numero de tres dígitos en un string. Esto lo hago para evitar utilizar "sprintf" y así ahorrar Tcys. Debo asegurarme de que el código que se ejecuta dentro de la rutina de interrupción no supere el valor de mi Ts.

```
char num_as_string[6] = { 0, 0, 0, '\r', '\n', '\0' };
void int_2_str(int num){
    /* Esta funcion convertira un numero de 3 digitos en un string para su impresion por la UART */
    /* La UART lo enviara y sera interpretado como codigo ASCII, luego por la posicion que ocupan
     * los numeros en la tabla ASCII debo sumarle 48 */
    *(num_as_string+2) = ((num%10)+48);
    *(num_as_string+1) = (((num%100)/10)+48);
    *(num_as_string+0) = ((num/100)+48);
}
```

Imagen 11. Función "int_2_str()"

Para contabilizar cuanto tarda en ejecutarse la rutina dentro de la interrupción voy a habilitar el TMR1. Es un timer de 16 bits y le asignare 2 de prescaler. Para esto voy a modificar la rutina de interrupción de la siguiente manera:

```
186  void __interrupt() ISR(){
187      start = TMR1;
188      INTCONbits.GIE=0;
189      initialize_TMR0();
190      ADCON0bits.GO_nDONE=1;  //GO/DONE: A/D
191                              //  1 = A/D co
192                              //  This bit i
193      while(ADCON0bits.GO_nDONE==1);
194      compensador(ADRESH);
195      PWM_CCP1_set_duty(Y[0]);
196      int_2_str(ADRESH);
197      UART_Write_Text(num_as_string);
198      INTCONbits.TMR0IF = 0;
199      INTCONbits.GIE = 1;
200      stop = TMR1;
201      INTCONbits.GIE = 0;
202      count = stop - start;
203      sprintf(string,"Count:%d\n\r",count);
204      UART_Write_Text(string);
205      INTCONbits.GIE = 1;
206  }
```

Imagen 12. Rutina de interrupción modificada

Voy a agregar la línea 187 y aquellas comprendidas entre la 200 y la 205. La línea 201 se debe agregar porque al utilizar “sprintf” el código dentro de la rutina tarde más en ejecutarse que el tiempo que tardan en dispararse las interrupciones. Luego en la línea 205 vuelvo a habilitar las interrupciones. Se podría decir que esto es equivalente a omitir las líneas 199 y 201, pero aunque a fines prácticos parece lo mismo, si omito la línea 199 estaría subestimando el tiempo que tarda el código en ejecutarse. Este código solo se ejecutó una vez para obtener este valor, luego se volvió al código original.

Obtendré los siguientes valores:

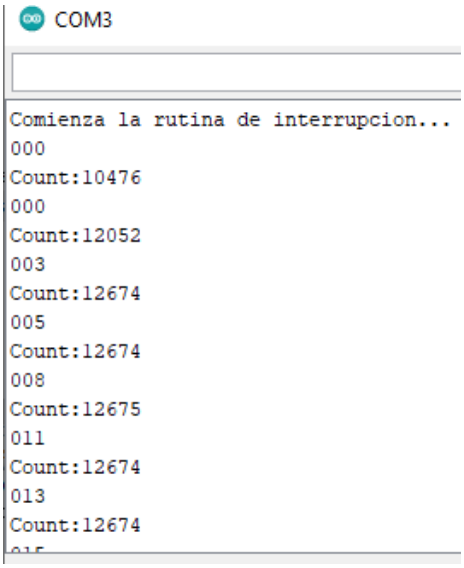


Imagen 13. Algunos valores obtenidos con el TMR1

La diferencia de valores se va a deber a que en las operaciones se manejan diferentes valores, es decir que un float largo puede demorarse más en una multiplicación que un valor nulo. Considerare el peor caso (aquel en el cual la rutina de interrupción tarda más en ejecutarse).

$$Cuenta_{TMR1} = 12675$$

Como estoy usando un 2 de prescaler:

$$T_{rutina} = \frac{4}{F_{osc}}.PRESACLER_{TMR1}.Cuentas_{TMR1} = 6{,}3375\ mseg$$

De esta manera compruebo que efectivamente la rutina se ejecuta antes de que se desencadene una nueva interrupción.

Aunque mido este tiempo debo considerar que tardara mas de los estimado ya que no estoy teniendo en cuenta el tiempo que tarda en activarse la interrupción y en entrar a la rutina. Esto se podría calcular manualmente con un osciloscopio pero es muy engorroso hacerlo por software. A fines prácticos asumiré que estos tiempos son despreciables.

Una vez que compruebo que el programa se ejecuta correctamente iré realizando pequeñas modificaciones para obtener distintos valores.

A) Programare para que salga por la uart el valor del “ADRESH” (salida de la planta)
B) Programare para que salga por la uart el valor “Y[0]” (salida del compensador)
C) Desactivare el TMR0 (las muestras no se tomaron cada Ts), programare que el modulo A/D desencadene las interrupciones y sacare por la uart el valor del “ADRESH”

Estos valores los mostrare por pantalla utilizando la interfaz de Arduino y luego los copiare a un archivo de Excel. Luego abriré desde MATLAB para efectuar los gráficos correspondientes.

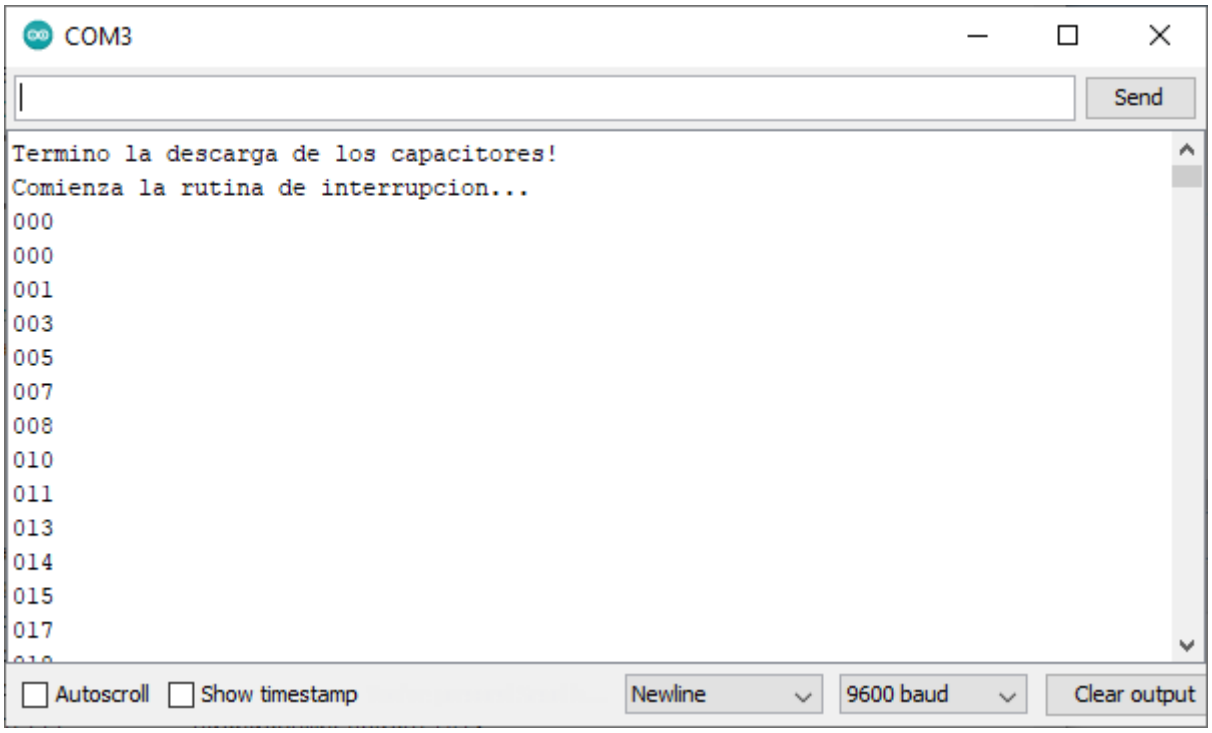


Imagen 14. Valores obtenidos en el caso A

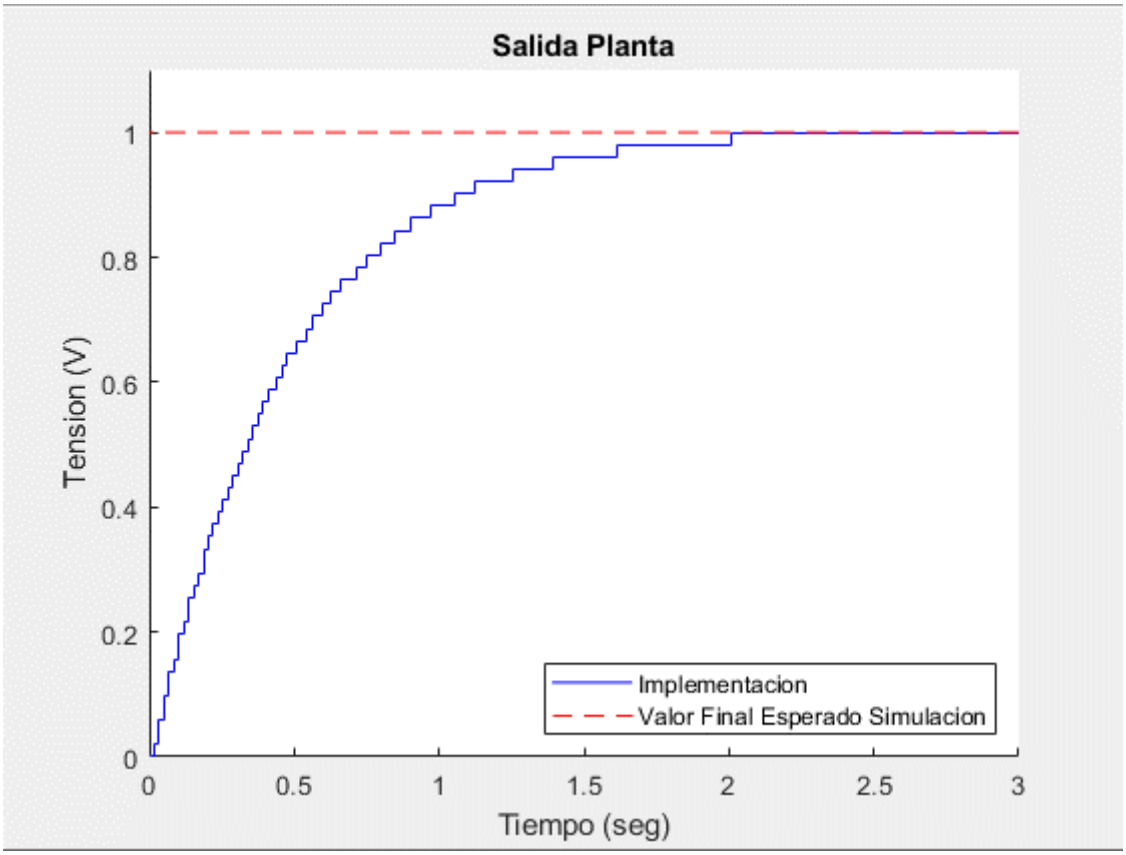


Imagen 15. Gráfico basado en los valores obtenidos en el caso A

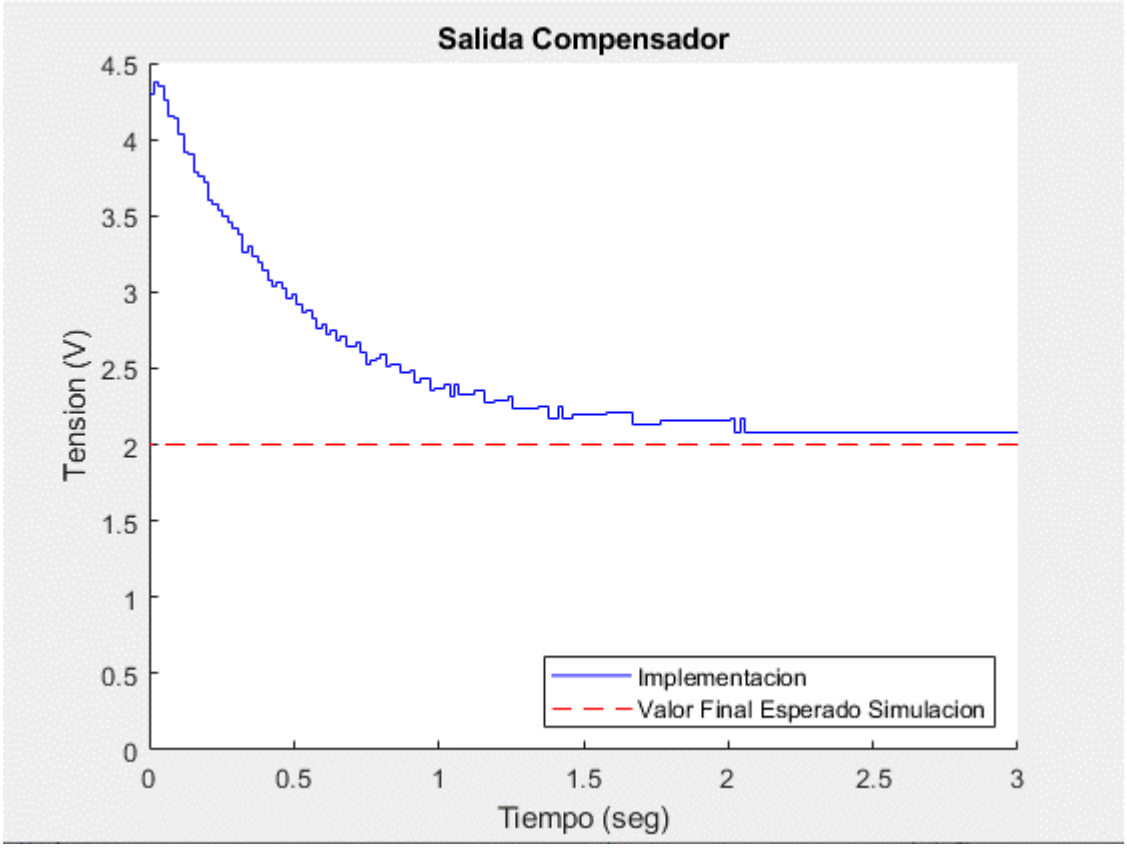


Imagen 16. Gráfico basado en los valores obtenidos en el caso B

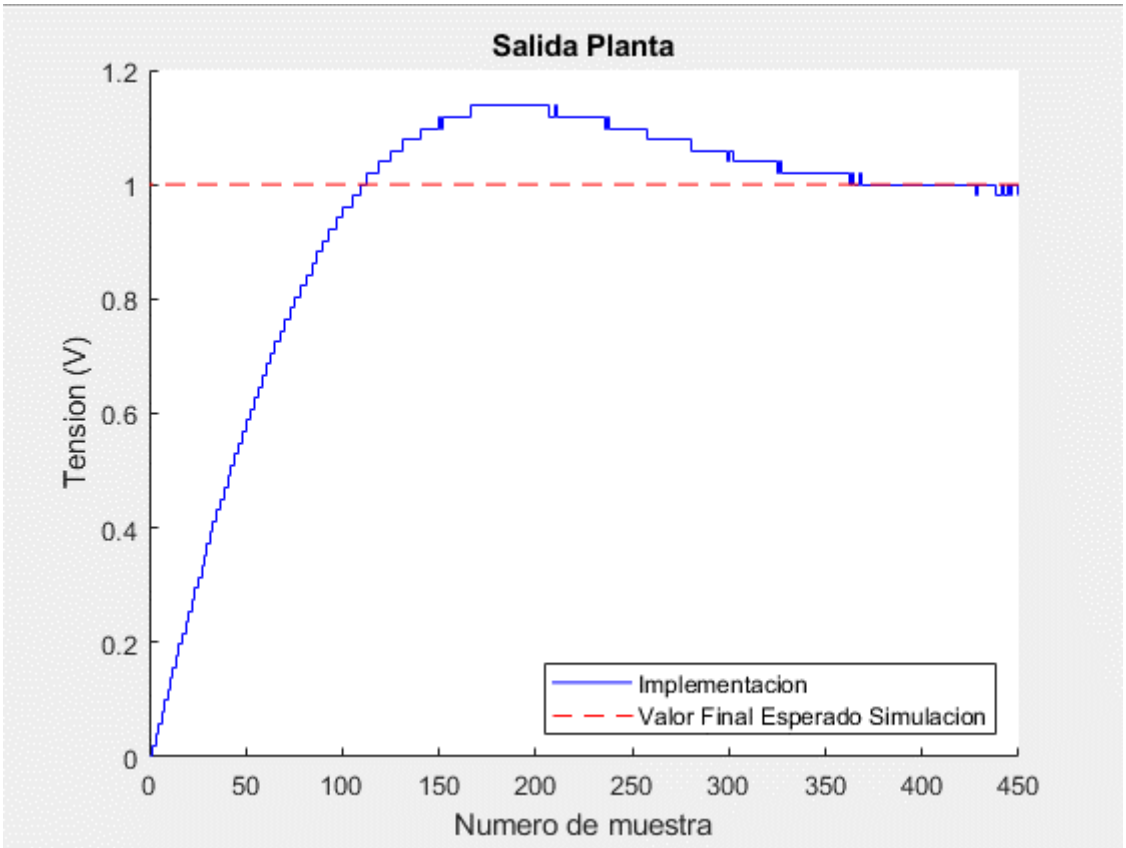


Imagen 17. Gráfico basado en los valores obtenidos en el caso C

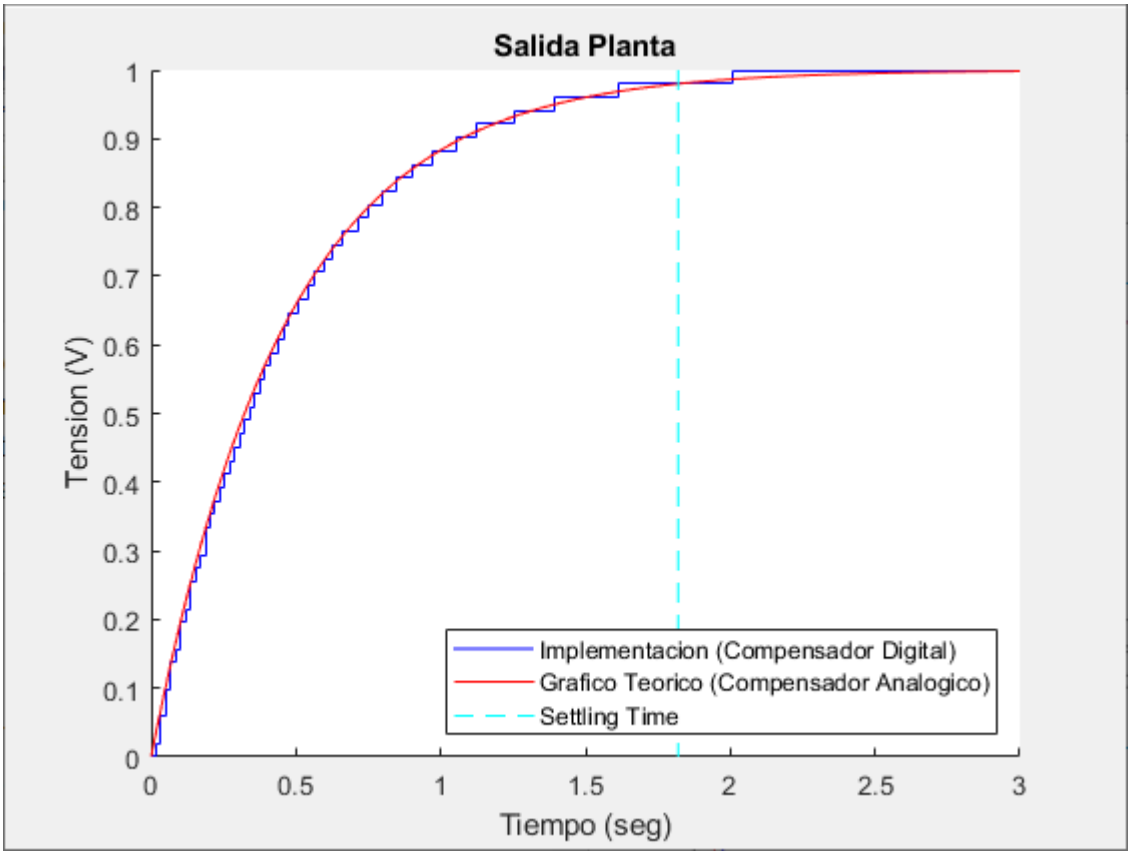


Imagen 18. Salida Real VS Salida Simulación

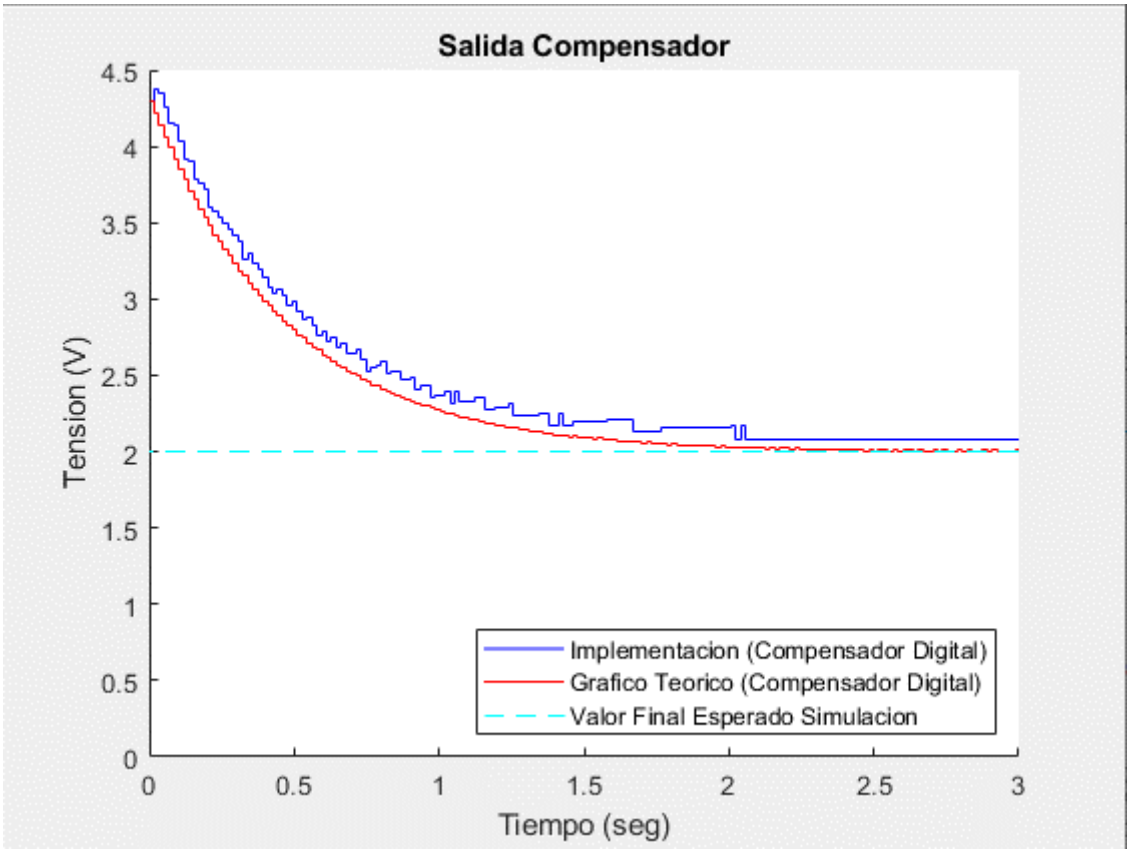


Imagen 18. Salida Compensador Real VS Salida Compensador Simulación

Observamos que la salida estabiliza perfectamente en el valor esperado, por otro lado notamos que el esfuerzo que debe hacer el compensador es apenas mayor. Esto tiene logica ya que se trata de un sistema real. Tambien notamos que si no se muestrea la señal cada Ts la salida tendra sobrepico (lo que no deberia suceder ya que es un sistema de primer orden). Esto se debe a que si no utilizo el Ts el cero que yo puse en "-1" para compensar no cae exactamente ahí, cae en cualquier lado, lo que hace que tenga dos polos.

A continuación incluiré imágenes de cómo fue la implementado del lazo:

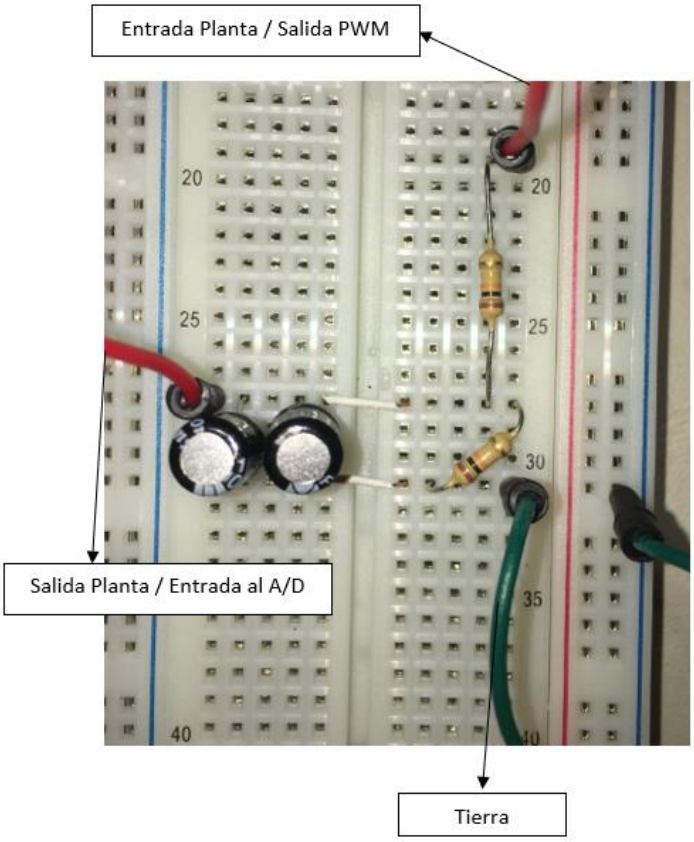


Imagen 19. Circuito Real

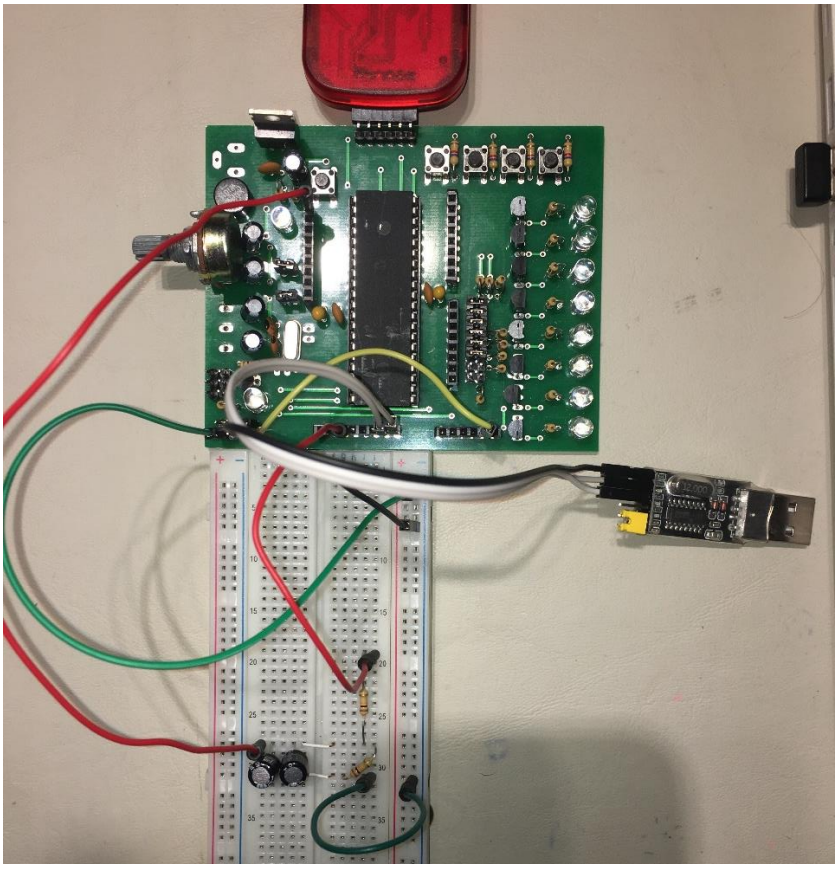


Imagen 20. Circuito, PIC 18F4620, UART y Pickit3 (Vista Superior)

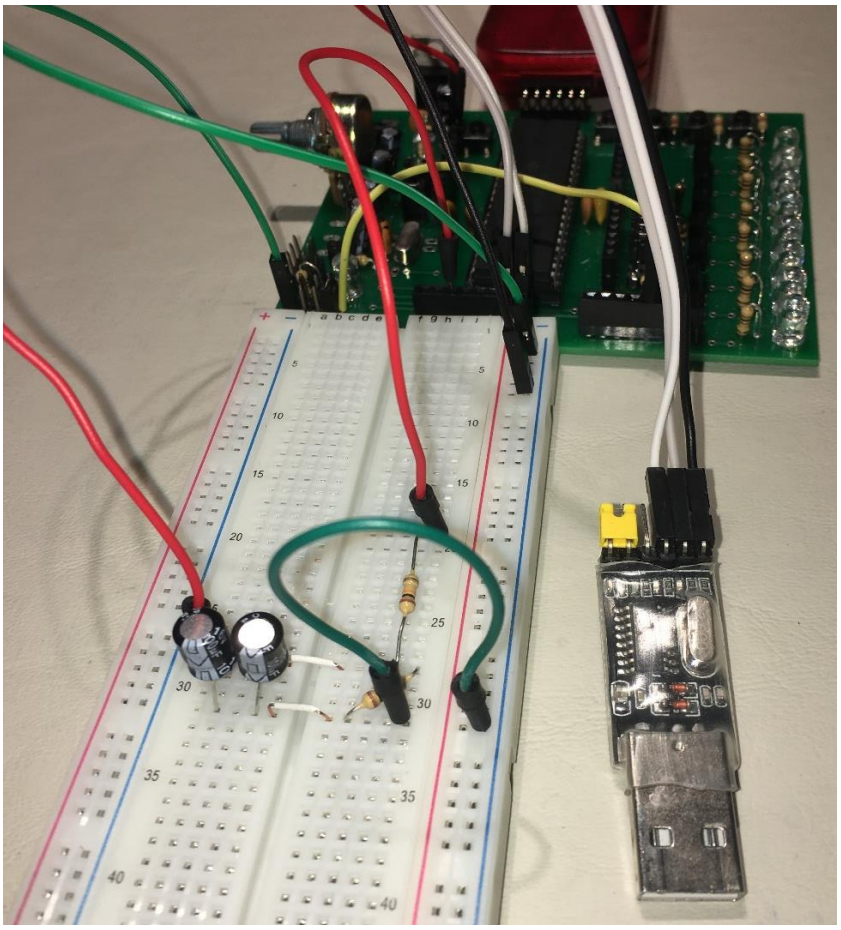


Imagen 21. Circuito, PIC 18F4620, UART y Pickit3 (Vista Frontal)