
**DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA**

**FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN**

Informe de Práctica N°1

# LAZO DE CONTROL IMPLEMENTADO EN UN MICROCONTROLADOR

**Asignatura:** SISTEMAS PARA CONTROL

**Ingeniería Electrónica**

***Autor:***

*Avila, Juan Agustin – Registro 26076*

**2º Semestre**

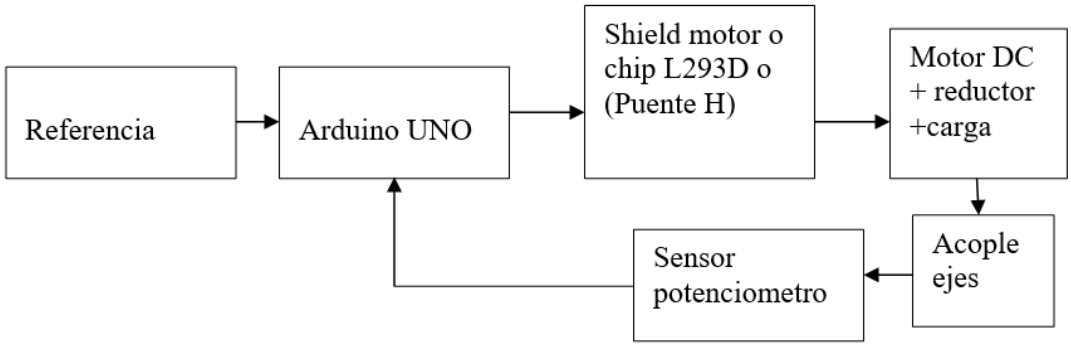
**Año 2020**

# 1 Introducción

En la siguiente práctica se realizará la aplicación de un sensor resistivo tipo potenciómetro en un lazo de control para el control de posición de un motor DC, tomando de referencia la posición en un eje de un smartphone. El control se realizará con la implementación de un controlador PID en una placa de desarrollo arduino UNO.

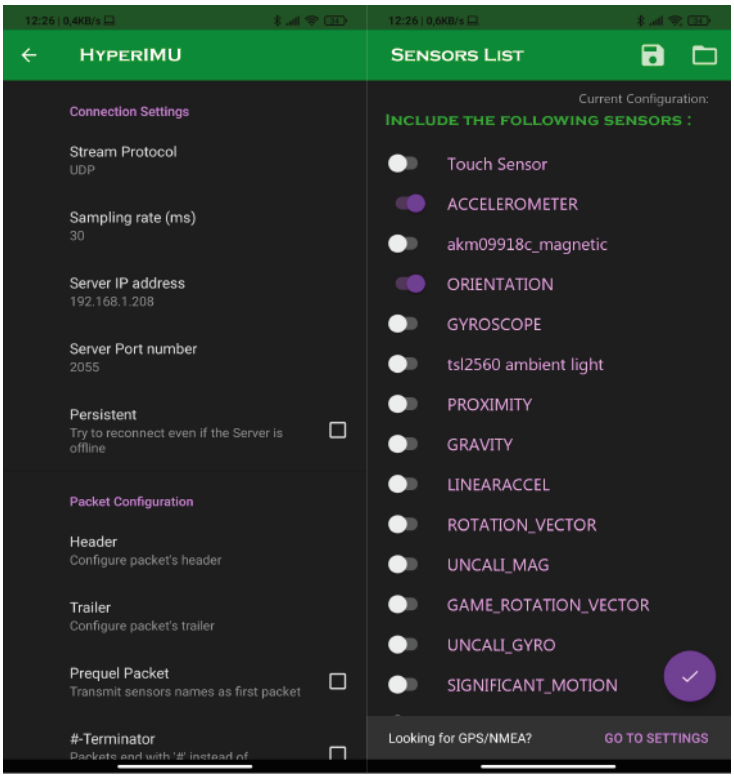
# 2 Desarrollo

El diagrama de bloques a implementar es el siguiente:




## 2.1 Referencia con HiperIMU

Para la entrada de referencia, se utilizó la aplicación HyperIMU, que puede utilizar los diversos sensores del smartphone utilizado y enviarlos mediante una red local a un servidor, que es el encargado de procesarlo y enviar esos datos procesados al arduino. Se eligió un tiempo de muestreo de 30ms, y el protocolo UDP para el envío de los paquetes. La configuración en el smartphone es la siguiente:



## 2.2 Motor utilizado

El motor utilizado es un motoreductor de dos ejes, cuyas especificaciones son las siguientes:

Tensión de funcionamiento: 3 V a 12 V DC, la tensión de funcionamiento recomendada es de aproximadamente 6 a 8 V

1, par máximo: 800gf, cm, min (3 V)
2, relación de reducción: 1:48
3, corriente de carga: 70mA (250mA, MAX) (3 V)

*Imagen 1 – motor utilizado*

Por disponibilidad, se alimentó el motor con una fuente de 5V. Uno de los ejes se conectó al sensor de posición, y el otro se conectó a una chapa metálica para determinar visualmente la posición.

## 2.3 Actuador del motor

El arduino en sí mismo es incapaz de controlar el motor, por lo que se utilizó un shield controlador de motores “Motor shield V1.0”, que consiste de un puente H L293D y generadores de señal PWM, que tiene cuatro salidas a motores independientes, una frecuencia PWM de hasta 312 kHz y la posibilidad de utilizar una fuente externa.

En este caso se utilizó la salida de motor 4, con el PWM (encargado de regular la tension de salida) a 8KHz, y además se utilizó una fuente externa de 5V 1ª para evitar que el consumo del motor genere problemas en el arduino.

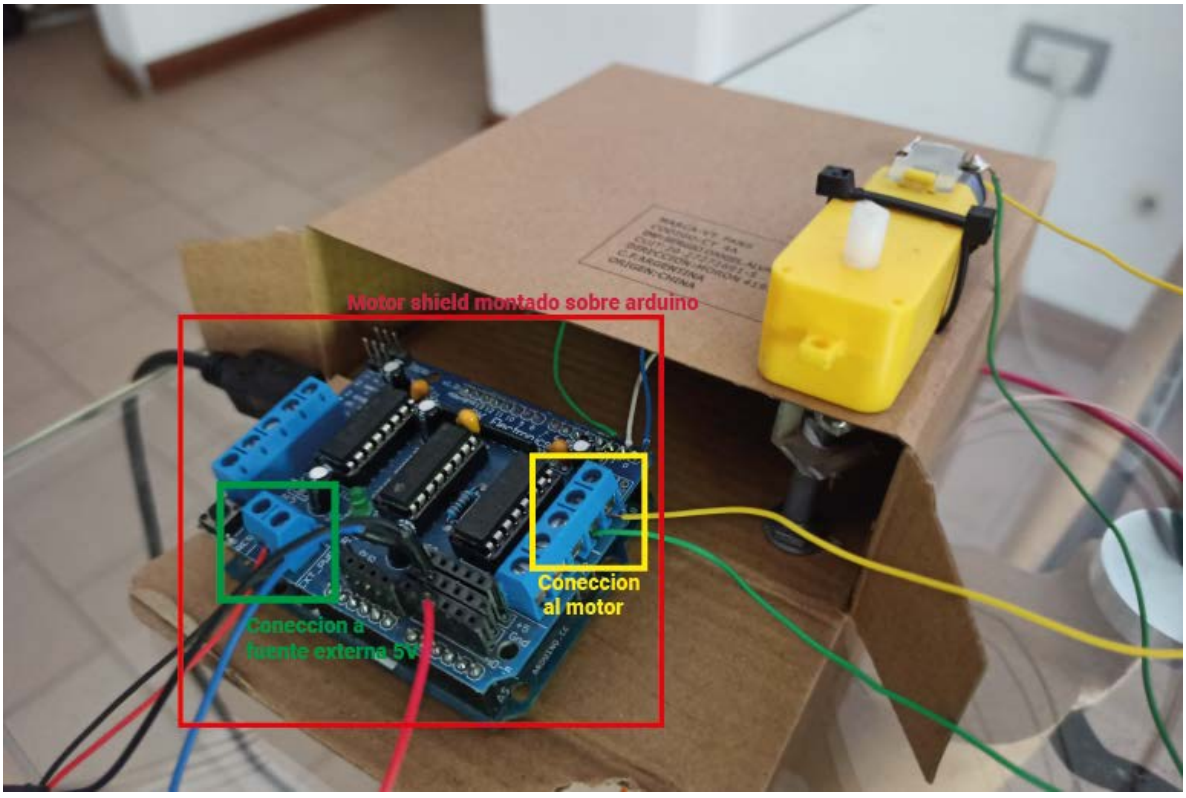



*Imagen 2 - Conexión del shield*

## 2.4 Sensado de posición

Para el sensado de la posición del motor DC, se utilizó un potenciómetro lineal de 10KΩ y 270° de recorrido, conectando sus pines externos a tierra y a vcc respectivamente, y el pin central se conectó al pin A1 del arduino uno. El eje móvil del potenciómetro se acopló a uno de los ejes del motor con un acople realizado con un perfil de aluminio, por lo tanto una variación en la posición del eje del motor se traduce en una variación de la posición relativa del potenciómetro, que a su vez se traduce en una variación de tensión entre 0V y 5V, leída por la entrada analógica del microprocesador. Es importante notar que la lectura introduce un error de cuantización.

*Imagen 3 - Acople del potenciómetro al eje del motor*

### 2.4.1 Error de cuantizacion del sensor

Para determinar el error introducido por el ADC, se toma el caso más desfavorable que es cuando la muestra se encuentra entre medio de nos niveles de cuantificación:

$$e_{max} = \frac{q}{2} = \frac{Rango}{2 * Cant\ niveles} = 2.441mV \quad (1)$$

Teniendo en cuenta que el recorrido del sensor es entre -90° y 90°, la diferencia de niveles es:

Nivel minimo =170

Nivel máximo= 853

Diferencia de niveles= 683

Por lo tanto, por regla de 3 se tiene:

$$683\ niveles \rightarrow 180°$$
$$0.5\ niveles \rightarrow 0.13177° \quad (2)$$

Por lo tanto el error de cuantización del ADC será de 0.1317° aproximadamente

## 2.5 Configuración del microprocesador

En esta configuración, el arduino es el encargado de recibir la referencia por el puerto serie, leer y filtrar la posicion actual, calcular el error del lazo de control, calcular la accion de control con un

controlador PID y enviar la salida al motor shield para que controle el motor. Además enviará por el puerto serie las variables utilizadas

### 2.5.1 Transmision y recepcion de datos

Para esto, se utilizó el puerto serie integrado en el arduino UNO, configurado a una velocidad de transferencia de 115200 baudios, recibiendo la referencia cada 30ms (mismo valor utilizado en HyperIMU), y enviando las variables del sistema cada 10ms. Las variables enviadas son la referencia, la posicion actual, la accion de control y el error calculado. Es importante notar que estas referencias son enviadas como numeros enteros, por lo tanto se multiplican por 100 para no perder precision.

### 2.5.2 Filtrado de la señal de posicion

Se realizo un filtro promediador de tres muestras con el siguiente codigo:

```
float filtroFIR()
{
    int N = 3; //cantidad de puntos
    int k;
    static float ent[N] = {analogRead(A1)}; //inicializa arreglo en el
 primer valor del pote
    float out = 0;
    for (k = 1; k < N; k++)
    {
        ent[k - 1] = ent[k]; //desplaza los valores
    }
    ent[N - 1] = analogRead(A1); //lee la nueva entrada
    for (k = 0; k < N; k++)
    {
        out = out + ent[k]; //Sumatoria de las ultimas entradas
    }
    return (float)out / N; //devuelve el promedio de las ultimas N mue
stras
}
```

Luego de varias pruebas, se comprobó que el retraso que introducia este filtro volvia a la planta marginalmente estable, generando oscilaciones periodicas. Para solucionarlo, se introdujo el mismo retraso de 3 muestras a la referencia.

### 2.5.3 Controlador PID

Para cumplir con las especificaciones de velocidad y error en estado estacionario, se decidió utilizar un controlador PID ya que es una solucion estandarizada en el mercado y de facil implementacion en arduino. Para discretizar la señal, se utilizo un tiempo de muestreo de 1ms, que es un periodo bastante utilizado para el control de motores.

Para esto, se utilizó la librería “PID_V1” (https://github.com/br3ttb/Arduino-PID-Library), creada por Brett Beauregard. Esta librería tiene la opcion de ser utilizada como un PID modificado, con el siguiente diagrama de bloques:

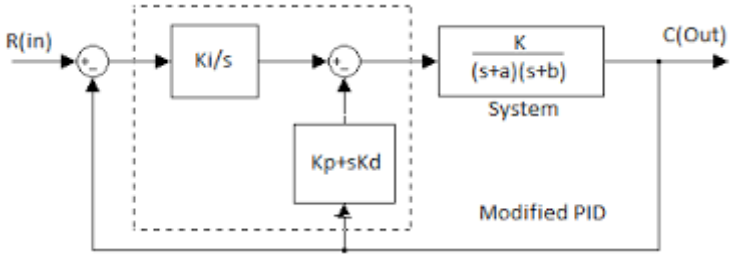



*Ilustración 1 – esquema del controlador PID modificado*

Ademas, la librería tiene otras ventajas como control anti-windup, limitando la accion de control a los valores maximos que soportan los actuadores, y ademas limitando independientemente la accion integral para evita el lag.

Es importante notar las limitaciones que posee la librería en sus aproximaciones de las acciones integrales y derivativas, una aproximacion escalon para la accion integrativa y una aproximacion lineal para la derivativa.

La aproximacion lineal para la accion D puede generar grandes cambios en la accion de control, ya que se calcula multiplicando la constante kd por la diferencia entre el error actual y el error en el instante de muestreo previo. Por lo tanto, es importante que la ganancia kd se mantenga pequeña.

Respecto a la accion integral, se incorporo una mejor pasando de la aproximacion escalonada a una aproximacion trapezoidal, con el siguiente codigo:

```
      outputSum+= (ki * (error+lastError))/2;   //Saca el promedio ent
re el error actual y el anterior
```

Con lo cual multiplica la constante ki por el promedio entre el error actual y el previo, y lo suma al error por accion integral acumulado.

Si bien la librería posee una librería complementaria para hacer autotuning, e incluso la posibilidad de modificar las tres ganancias mientras el sistema está andando, se optó por el metodo de prueba y error para el ajuste de los parametros, utilizando el siguiente procedimiento:

1. Se anularon los términos integrativo y derivativo.
2. Se aumentó el término proporcional hasta obtener una respuesta lo más rápida posible sin que presente oscilaciones.
3. Se incorporó el efecto derivativo para disminuir el sobreimpulso y aumentar el tiempo de respuesta.
4. Como consecuencia del agregado de D, se aumentó nuevamente P para mejorar la respuesta.
5. Se repitieron los pasos 3 y 4 hasta obtener una respuesta adecuada, teniendo cuidado de que la accion derivativa sea pequeña.
6. Se aumentó el parametro I para que el error en estado estacionario fuese cero.

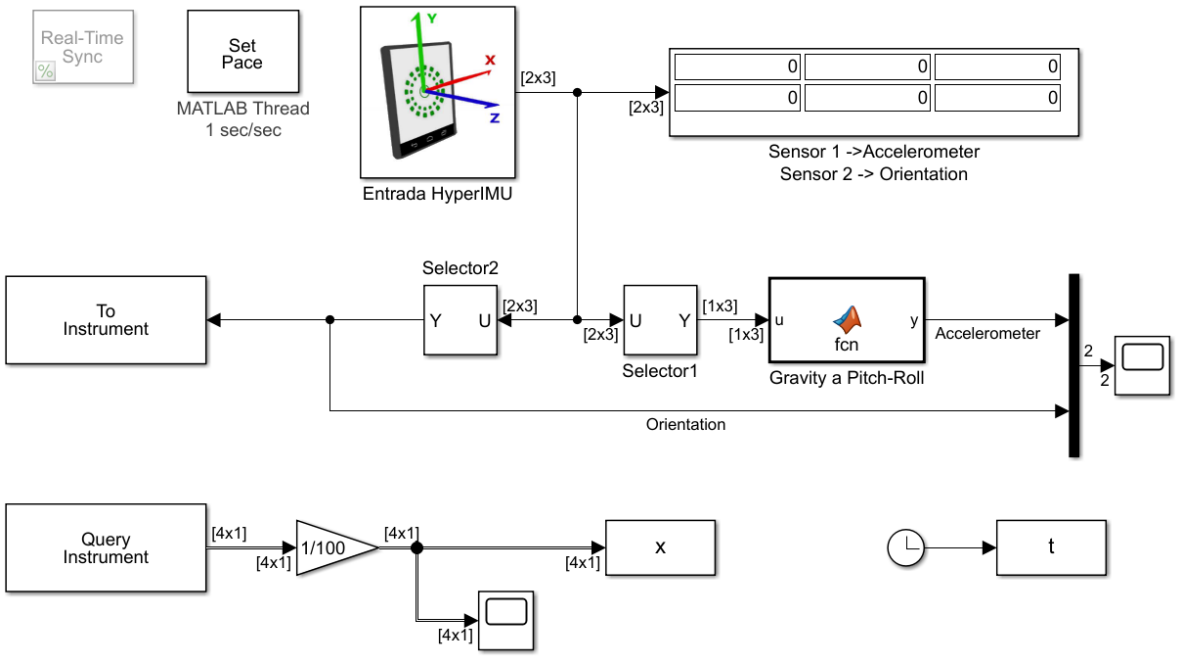
Con esto, se llego a los siguientes parámetros:

```
//Parametros PID
float Kp = 0.7, Ki = 0.1, Kd = 0.005;
```

Otra limitacion importante para notar en el controlador, es que el motor DC no tiene un comportamiento lineal, si no que tiene una zona muerta. Esto se intentó subsanar teniendo en cuenta la zona muerta y restándosela a la acción de control. Por lo tanto, se configuró una zona muerta de 1V y se restó de la tensión máxima de la fuente (5V), limitando así la acción de control entre -4V y 4V, y a eso sumándole o restándole la zona muerta, siempre y cuando el error fuera mayor a 1 grado (especificación de error pedida)

## 2.6 Conexión entre HyperIMU y arduino

Para la conexión entre arduino e hyperIMU, como tambien para el registro de datos, se utilizó matlab y simulink, con el siguiente esquema:




En el cual se observa el primer bloque que abre la conexión UDP en el puerto especificado, y adquiere los datos para los dos sensores (tres datos por sensor, uno para cada eje espacial respectivamente).

Esa salida es luego separada, por una lado se envian los datos del acelerometro a la funcion “gravity a pitch-roll”, que con los datos de aceleracion calcula la posicion del smartphone. Tambien se envia el sensor de posicion al mismo scope que la salida de la funcion “gravity a pitch-roll” para verificar la similitud de los datos.

Por otro lado, se envia solo un eje de los datos de posicion a la salida del puerto serie, que será utilizado como referencia por el microcontrolador. Estos datos se envian cada 30ms, que es igual al tiempo de muestreo de hyperIMU y de arduino.

En la parte inferior, el bloque “Query instrument” recibe los datos enviados por arduino, y luego se pasan por una ganancia de 1/100 (ya que previamente se habian multiplicado por 100). Esos datos se envian a otro scope y a la vez a la variable x.

Ademas, el sistema de simulink tiene el bloque “Set pace” que logra que la simulacion corra en tiempo real, y un clock que se envia a una variable “t”.

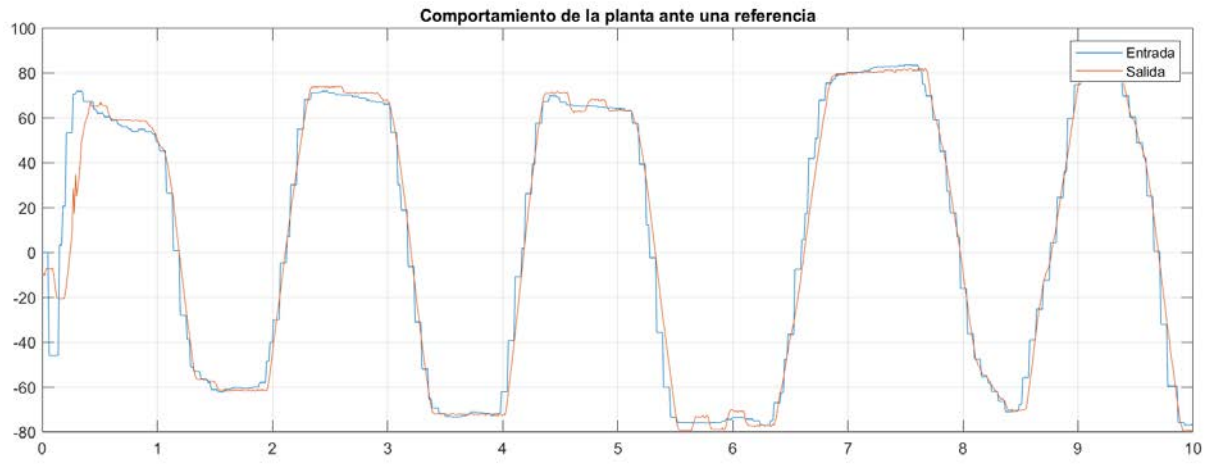
## 2.7 Resultado de las mediciones

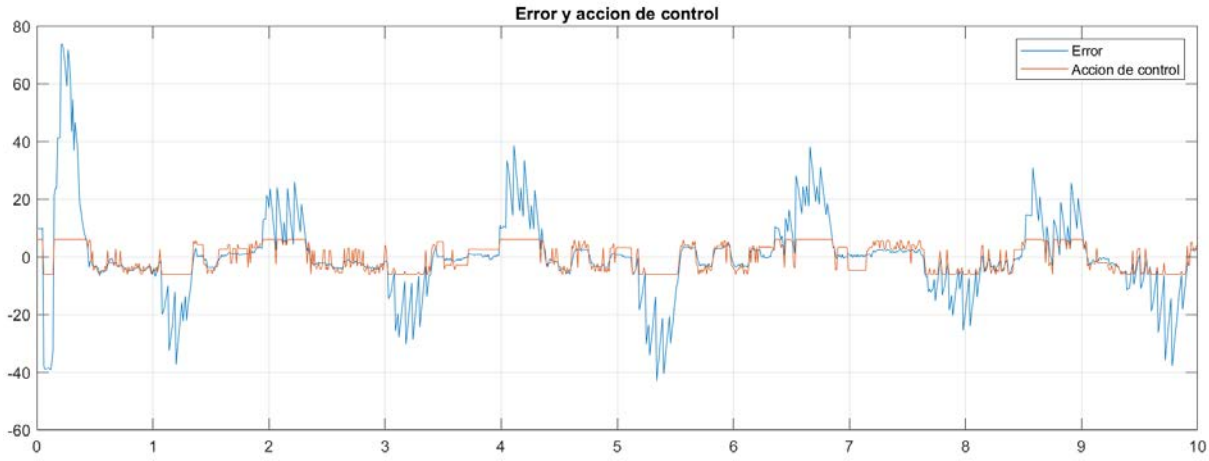
Se corrió la simulacion en simulink, y los datos generados (x matriz con los 4 valores devueltos por arduino, y t la base temporal) se guardaron en el archivo “medidas hiperimu.mat”.

Luego, con el siguiente script, se graficaron las entradas y salidas en un grafico, y el error y accion de control en otro:

```
clc;clear all;
load('medidas hiperimu.mat');    %se carga el archivo con los valores de la simulacion
% para generar cada variable utilice el comando "squeeze(x(var,1,:))" (para reducir las dimensiones)
entrada=squeeze(x(1,1,:));
salida=squeeze(x(2,1,:));
salida=salida(4:end);   %Para comparar, se eliminan las primeras 3 muestras que es el retraso del arduino
```

```
salida=[salida; 0;0;0;];% y para que tengan la misma dimension los vector
es, se agregan tres valores nulos al final
accioncontrol=squeeze(x(3,1,:)/10); %Se divide en 10 para que quede en vo
lts
for i=1:length(accioncontrol)
    if accioncontrol(i)>0
        accioncontrol(i)=accioncontrol(i)+2; %le suma la zona muerta
    elseif accioncontrol(i)<0
        accioncontrol(i)=accioncontrol(i)-
2; %si es negativa le resta la zona muerta
    end
end
error=squeeze(x(4,1,:));
tmin=0;
tmax=10;
%% Graficacion
figure(1);
plot(t,entrada,t,salida);
grid on;
xlim([tmin tmax]); %Se selecciona un rango con mucho movimiento de la ref
erencia
legend("Entrada","Salida");
title("Comportamiento de la planta ante una referencia");
saveas(1,"entradaysalida.png")
%% grafica con error y accion de control
figure(2);
plot(t,error,t,accioncontrol);
grid on;
xlim([tmin tmax]); %Se selecciona un rango con mucho movimiento de la ref
erencia
legend("Error","Accion de control");
title("Error y ");
saveas(2,"error y accioncontrol.png")
```






# 3 Conclusion

Durante la práctica se obtuvo experiencia en cuanto a la planificación y desarrollo de este proyecto que integró conocimientos adquiridos durante de la carrera, principalmente en el area de control. La principal dificultad fue que no se contaba con experiencias previas en el armado de plantas que involucraran elementos mecánicos, y a la vez tampoco se habian realizado practicas fuera de la teoria, por lo tanto fue la primera vez que se lidió con ruidos y errores introducidos por caracteristicas propias de la planta, como el ruido que incorpora el sensado del potenciometro.
Se puede encontrar el codigo utilizado en el siguiente link:

https://github.com/agustinavila/8semestreELO/tree/master/Sistemas%20para%20control/Practica%201