Balancín basado en arduino

REGULACIÓN AUTOMÁTICA 2019

Diseño y control de un sistema de balancín basado en Arduino con un motor BLDC y un potenciómetro como sensor.

Antonio Arco Torres

Email:
antonio.arco.torres@alumnos.upm.es

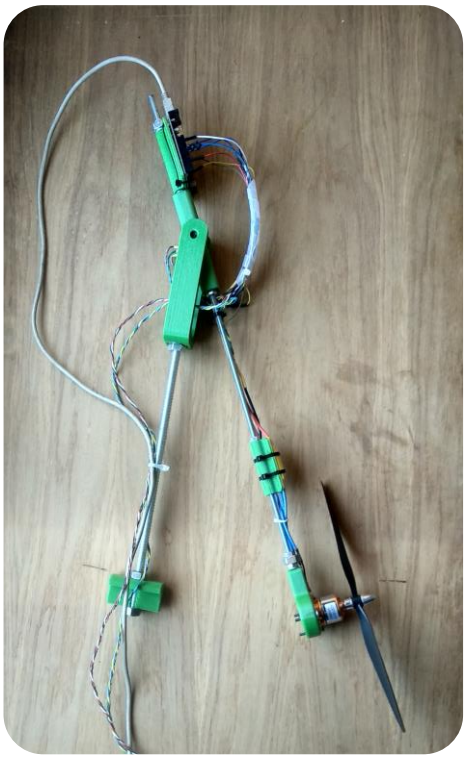
Contenido

ñ

Resumen y objetivos.

El objetivo de este proyecto es desarrollar y un sistema de balancín y posteriormente realizar el control de este para que mantenga una posición fija dada. El diseño de la estructura se realizará usando el programa Autodesk Inventor y el control se realizara con la ayuda de Matlab/Simulink y el IDE de Arduino.

Maqueta



Lista de materiales:

- Arduino Uno.
- Potenciómetro de 100K.
- Motor BLDC 1000KV.
- ESC 30A.
- Fuente de alimentación.
- Componentes Impresos en 3d.
- Tornillo de banco.
- Hélice.
- Diversa tornillería.
- 1M de varilla M6
- 1M de varilla M10
- Rodamiento 625ZZ.

Funcionamiento.

La estructura consta de un eje central en el que el potenciómetro actúa de eje. Sobre este eje se extienden dos brazos, uno que cuenta con el motor y la hélice y otro que actuará como soporte para el Arduino.

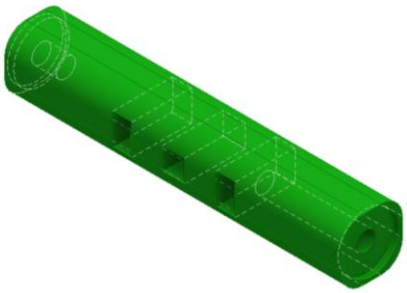
Al girar la hélice el brazo se eleva, haciendo girar el eje y por tanto el potenciómetro que registrará el nuevo ángulo al que se encuentra este.

La maqueta se sujetará a una mesa usando un tornillo de banco en el que esta diseñada para encajar.

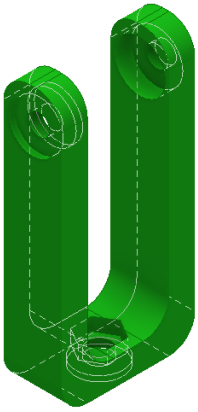
Desglose de las piezas impresas.

Los archivos de diseño de estas piezas se pueden encontrar en el GitHub del proyecto cuyo enlace se puede encontrar al final de esta memoria.

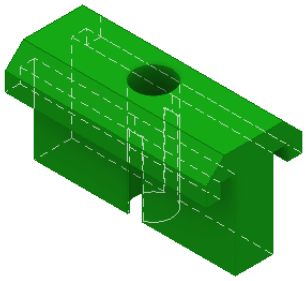
- **Brazo.ipt:** Esta es la pieza central del brazo de nuestra maqueta, se encarga de unir las dos varillas que roscan a cada uno de sus extremos, la que soporta el motor y la hélice y la que soporta la electrónica con el eje del potenciómetro, que pasa por su centro. Las dos varillas se roscan gracias a dos tuercas empotradas en la pieza.



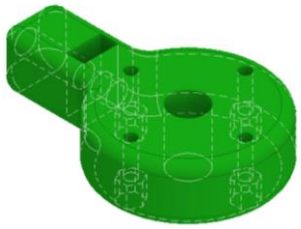
- **U.ipt:** Su función es la de sujetar el cuerpo del potenciómetro, que se atornilla en uno de sus extremos, en el otro se empotra un rodamiento 625ZZ que sujetara el eje del potenciómetro para darle robustez. En su base se empotrará una tuerca de M10 para posteriormente atornillar una varilla que será el principal soporte de la estructura.



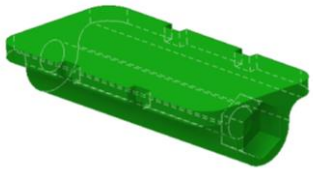
- **SoporteVarilla.ipt:** Esta pieza es la encargada de facilitar que la varilla de M10 que soporta el proyecto quede bien fijada en el gato, lo que nos permite sujetar este a casi cualquier mesa.



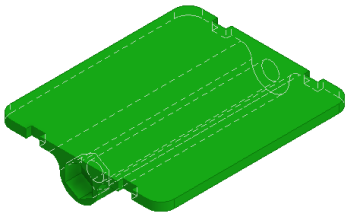
- **SoporteMotor.ipt:** Esta pieza es la encargada de unir el motor al brazo, para ello se empotrará una tuerca M6 en ella misma para poder atornillar la barilla. Para sujetar el motor se usan tornillos de M3 que cuentan con tuercas empotradas en la cara inferior de la pieza.



- **SoporteEsc.ipt:** Esta pieza actuará como soporte para poder fijar la ESC que controla el motor a la varilla de M6 gracias a unas tuercas empotradas.



- Soporte Arduino: Esta pieza actuara como soporte para poder fijar el Arduino a la varilla de M6 gracias a unas tuercas empotradas.



Modelado del sistema.

Respuesta al escalón.

Para obtener un modelo del sistema vamos a usar un osciloscopio Analog Discovery 2.

Vamos a estudiar cómo responde nuestra maqueta a una señal escalón que produciremos con el siguiente código:

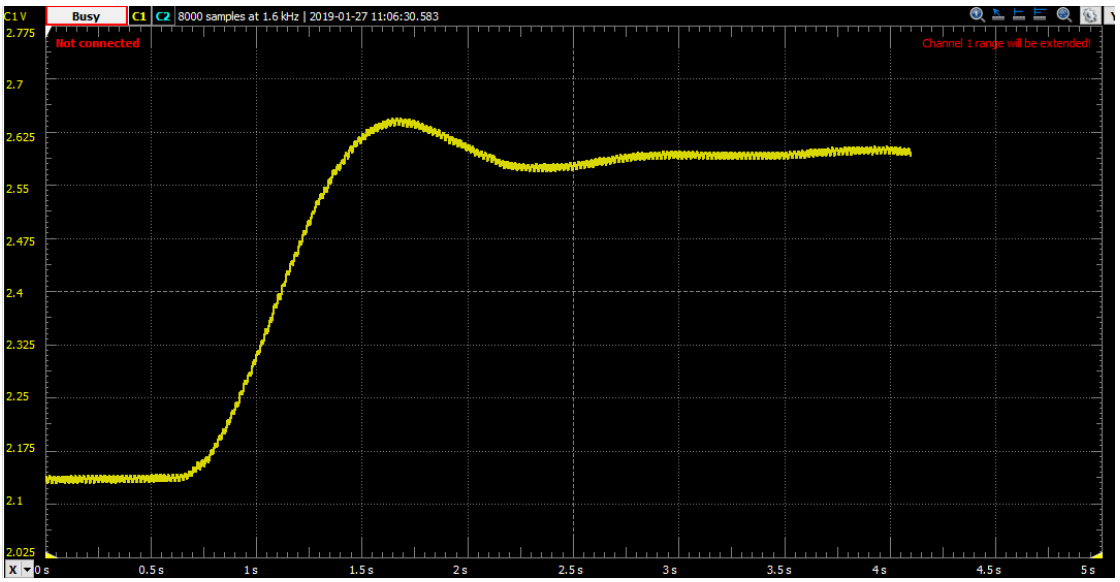
Se ha omitido el setup junto a otros fragmentos de código que se explicarán mas adelante con el motivo de obtener una memoria más limpia y comprensible.

En el código, mandamos la maqueta a una posición en la que se mantiene estable y pasados 20 segundos introducimos el escalón.

```
int escalon = 1240;
unsigned long tiempo;

void loop()
{
       tiempo = millis();
       esc.writeMicroseconds(escalon);

              if(tiempo >= 20000)
              {
              escalon = 1270;
              }
}
```

Resultados experimentales.

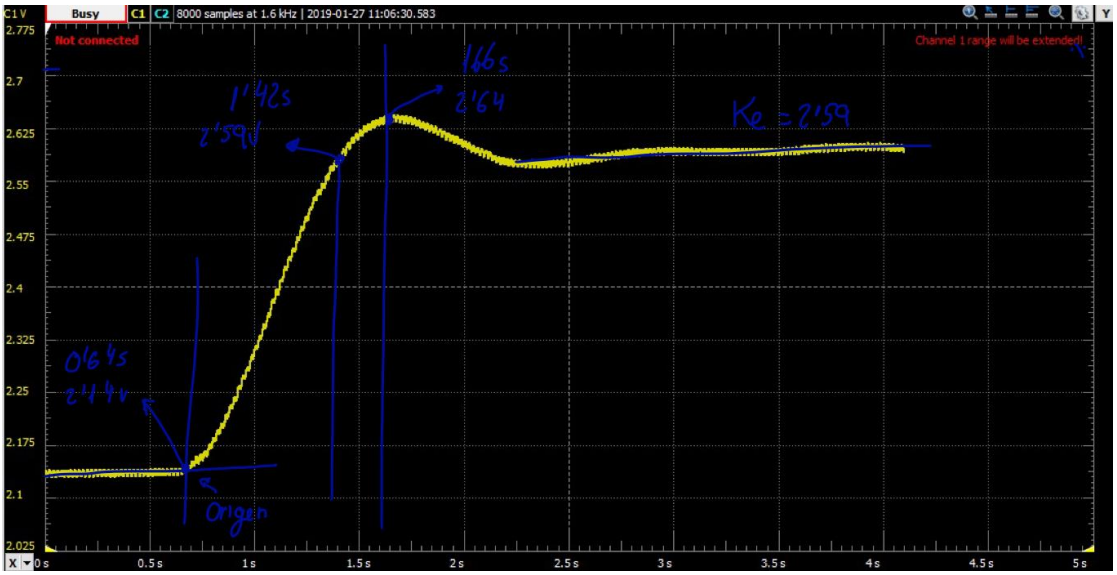
Se ha analizado la respuesta al escalón con un osciloscopio Analog Discovery 2. Tras cargar el código anterior podemos observar este comportamiento si conectamos el AnalogDiscovery al potenciómetro.



De esta forma de onda podemos observar que nuestro sistema será de segundo orden. Para obtener más datos necesitaremos realizar un estudio matemático.

Cálculos matemáticos.

Lo primero que haremos será marcar los puntos de interés en la gráfica, como el tiempo de respuesta, el de pico y el de estabilización, así como sus respectivos valores. Además, definiremos un origen de coordenadas. Nos quedara de la siguiente manera:



De aquí obtenemos:

Origen	Tr	Ts=Tp	Ke
0.64s → 0s	1.42s-0.64s = 0.78s	1.02s	Infinito
2.14V → 0v	2.59v-2.14v = 0.45v	0.5v	0.45v

Con estos datos podemos calcular todo lo que necesitamos.

$$Mp = \frac{yp - y\infty}{y\infty} * 100 = 11.11\%$$

$$Mp = 100 * e^{-\frac{\pi}{tg\theta}} \; ; \; \theta \; = 0.9605 \text{ rad}$$

Con este ángulo θ calculado ya podemos saber que nuestro sistema será sobre amortiguado, cosa que podemos corroborar en la práctica ya que observamos que es oscilatorio y estable.

Ahora que conocemos el ángulo podemos calcular:

$$\xi = \cos\theta = 0.5 < 731$$

$$Ts = \frac{\pi}{\sigma} \rightarrow \frac{\pi}{1.02} = \; \sigma = 3.08$$

$$Tp = \frac{\pi}{\varpi d} \rightarrow \frac{\pi}{1.02} = \; \varpi d = 3.08$$

Obteniendo ϖd ya témenos todo lo que necesitábamos para calcular ϖn y obtener la función de transferencia.

$$\varpi d = \; \varpi n * \text{sen}\,\theta \rightarrow \; \varpi n = 3.7584$$

La función de transferencia será de la forma:

$$FDT = \frac{Ke * \varpi n^2}{s^2 + 2\xi\varpi n * s + \varpi n^2}$$

Sustituyendo y realizando los cálculos nos queda:

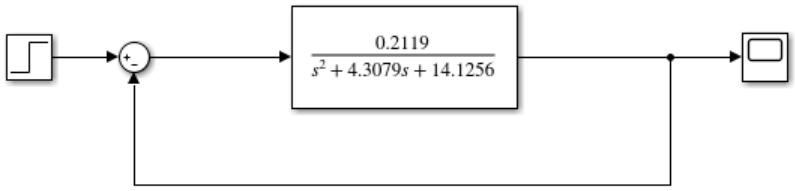
$$FDT = \frac{6.3565}{s^2 + 4.3079s + 14.1256}$$

Sin embargo aun no habríamos terminado ya que si nos fijamos en el código de Arduino con el que hemos producido el escalón observamos que no es un escalón unitario, sino que pasamos del valor “1240” al “1270” por lo que nuestro escalón será de 30 unidades. Para conseguir la respuesta al escalón unitario deberemos dividir nuestra FDT entre 30, por lo que el resultado final será:

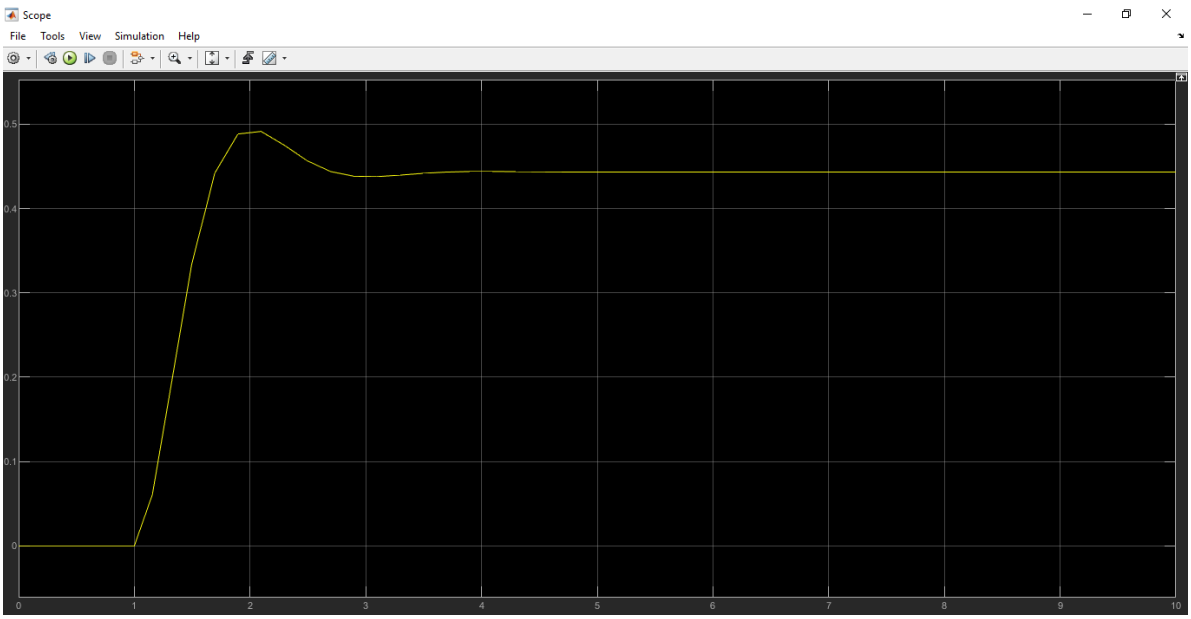
$$FDT = \frac{0.2119}{s^2 + 4.3079s + 14.1256}$$

Comprobación Teórica.

Una vez que tenemos la FDT, podemos modelarla en Simulink y observar su respuesta al escalón para comprobar si se comporta tal y como hemos calculado.



Al simular la respuesta nos queda exactamente una función como la que habíamos modelado así que podemos asumir que nuestros cálculos son correctos.

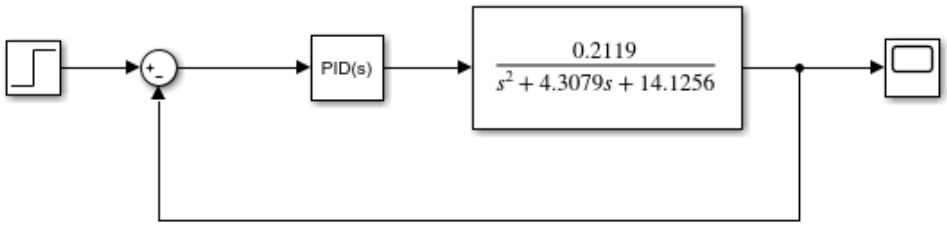


Control del sistema.

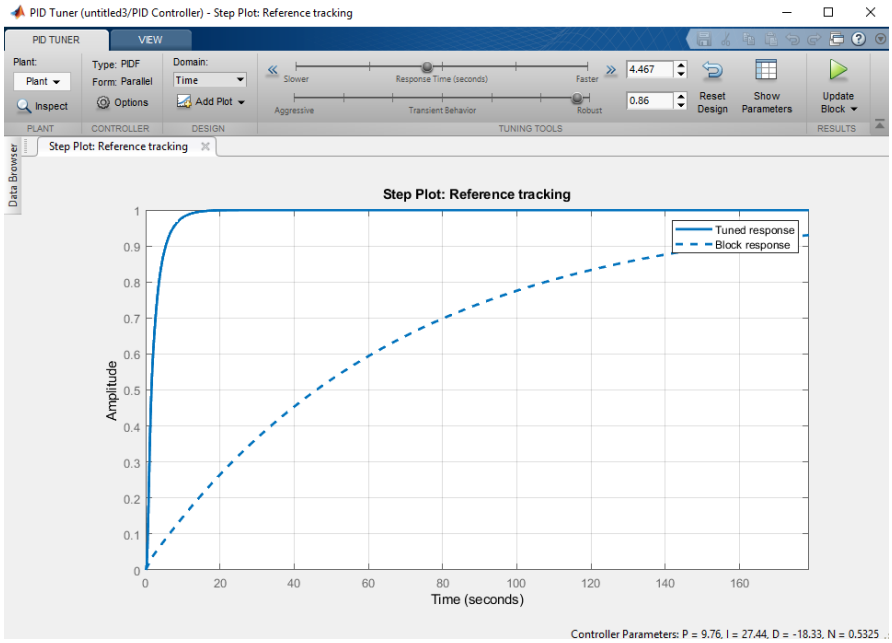
Ahora que tenemos nuestro modelo de la planta a controlar podemos añadir en el propio archivo de Simulink, creado anteriormente, un regulador PID y gracias a las herramientas de Matlab ajustarlo para conseguir los valores óptimos de KP, KI y KD, que luego llevaremos a nuestro regulador real.

Modelado en Simulink.

La planta con el regulador nos quedará:



Usando la herramienta de Matlab se nos muestra la siguiente ventana:



Ajustando con las deslizaderas la parte superior podremos regular la respuesta que queremos en nuestro sistema, lo que nos deja unos valores de:

Kp	Ki	Kd
9.75976669268827	27.438613663467	-18.3285976721823

Programa de control.

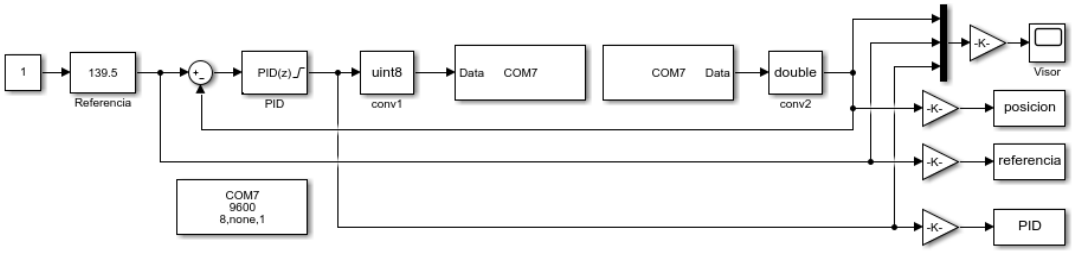
- **Simulink**

Ahora que ya tenemos los valores necesarios para nuestro regulador pasamos a implementar el sistema que controlará nuestra maqueta, para ello realizaremos los cálculos en Simulink, usando el bloque de PID que incorpora.

El funcionamiento será el siguiente:

1. El Arduino se comunicará por el puerto serie con Simulink y le pasará un valor tipo byte, para poder simplificar las comunicaciones, con el estado del potenciómetro, sobre el que Simulink actuará, comparando este valor con la referencia que le hemos dado. Esta referencia se corresponderá en la vida real con el ángulo al que estará el brazo de la maqueta.
2. Con el valor del potenciómetro Simulink calculará la respuesta que tiene que ejecutar el motor y se la transmitirá también por puerto serie al Arduino que enviará esta a la ESC para mover el motor. De esta forma el Arduino actuara como un mero interprete ente Simulink y las entradas y actuadores de la maqueta.

El diagrama de la planta nos quedará:



- **Arduino**

Como hemos mencionado antes el Arduino no será nada mas que un mero interprete, sin embargo, tenemos que cargar en él un programa que le permita saber que datos debe enviar a Simulink y por el otro lado saber que hacer con los datos que recibe.

Para ello hemos programado lo siguiente:

Estamos usando la librería Servo.h para controlar nuestra ESC ya que estas utilizan el mismo sistema de comunicación que los servos.

Las líneas de código donde enviamos valores a la ESC en el setup, son para poder arrancar el motor correctamente, ya que por seguridad la ESC antes de arrancar el motor espera que en un mando de radiocontrol, que es para donde está diseñado su uso, el joystick se lleve a su valor más bajo y se mantenga en esa posición unos segundos (de esta manera por ejemplo al arrancar un quadcoptero no sale volando inmediatamente o directamente se descontrola al encender el mando), esto podemos emularlo por software con esas líneas de código.

A parte de esto las demás partes del código se encuentran comentadas y son autoexplicatorias.

```
#include <Servo.h>
Servo esc;
int Input = 0;

void setup()
{

  Serial.begin(9600);

  esc.attach(8);
  esc.writeMicroseconds(1000);
  delay(3000);
  esc.writeMicroseconds(1300);
  delay(500);
  esc.writeMicroseconds(1000);


  //El potenciometro esta conectado a A1.
  Input = analogRead(A1);
}
```

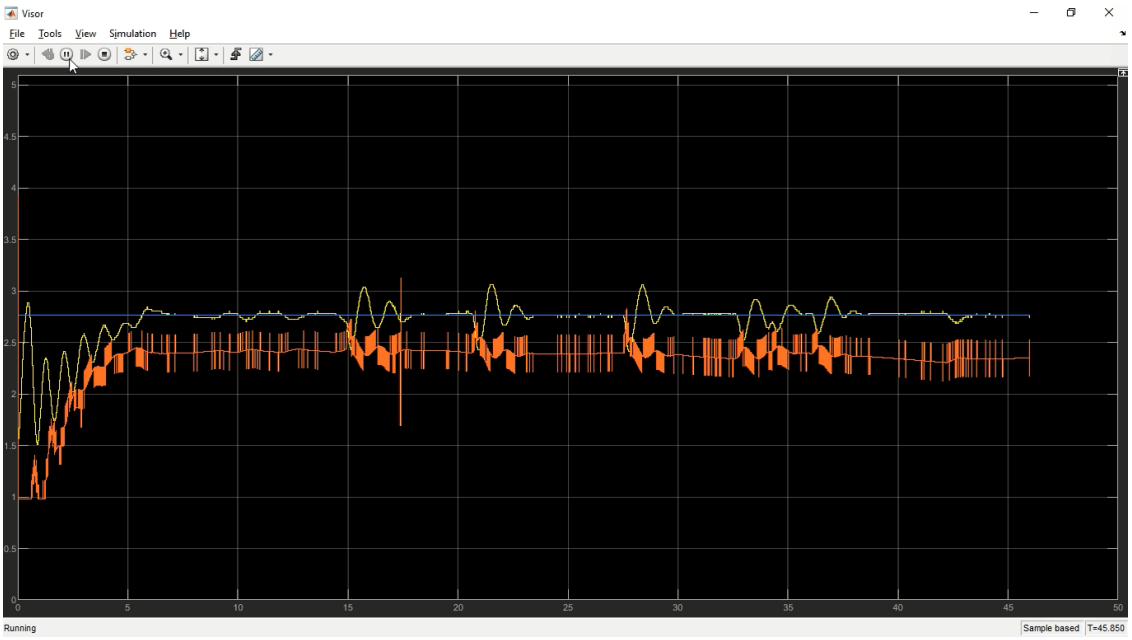
```
void loop()
{
  byte Outesc;
  int OutescM;
  byte InputM;


  Input = analogRead(A1);

  //Con map ponemos Input dentro del rango de una variable byte.
  InputM=map(Input, 0, 1023, 0, 255);
  //Enviamos el dato a Simulink.
  Serial.write(InputM);

  //Leemos el puerto para obtener los datos que nos envia Simulink

   if(Serial.available()){
    Outesc = Serial.read();
  }


  //Simulink tambien nos envia un byte, asi que tenemos que volver a hacer
un map
  //para adaptarlo al rango que entiende nuestro motor.
  OutescM = map(Outesc, 0, 255, 1200, 1400);

  esc.writeMicroseconds(OutescM);

  //este delay es para estabilizar el conversor.
  delay(20);
}
```

Resultados experimentales.

El diseño que hemos realizado en Simulink nos muestra por el Scope el estado del sistema:

- En azul nos muestra donde está la referencia.
- En amarillo donde se encuentra el sistema.
- En rojo la respuesta que el PID esta mandando.

Por lo que si arrancamos nuestro sistema y lo perturbamos un poco con la mano observamos lo siguiente:



De esta gráfica podemos ver que el sistema es lo suficientemente estable y preciso ya que cuando alcanza la referencia se mantienen bastante bien en esta, podemos observar que su velocidad es correcta ya que tras dos picos se ha estabilizado.

Por otro lado, podríamos considerar que la respuesta de Kd, conclusión que observamos al ver la zona del arranque. Reducir Kd ayudaría a suavizarla. Reducir Kp también ayudaría aunque haría que el sistema fuera más lento, habría que equilibrar ambas.

Conclusiones.

A lo largo de esta memoria se ha detallado como pasar de la maqueta al mundo de la simulación y de vuelta a la maqueta para poder regular esta.

Toda la fase teórica y práctica ha ido como se esperaba, sin embargo los valores de PID calculados por Matlab eran demasiado exagerados y al final se ha terminado usando otros, se desconoce la razón por la que no han funcionado ya que según creo he planteado todo de una manera correcta.

Enlaces de interés:

Github:

https://github.com/antoniobeta/Balancin-Arduino

Youtube:

https://youtu.be/0H3d7ssZbtQ