UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD DE
INGENIERÍA

Control PID aplicado a modelo de Robot ABB IRB 7600

Proyecto final de la materia Control y sistemas

Olguin Nahuel (12297)

Junio 2023

Índice

1 Introducción **2**

2 Pruebas previas al proyecto **3**
2.1 Unity . . . 3
2.2 Simulink . . . 3
2.3 Comparación de resultados . . . 4

3 Control PID **6**
3.1 Método de sintonización . . . 6
3.2 Sintonización en la posición de menor inercia . . . 7
3.3 Sintonización en la posición de mayor inercia . . . 13

4 Procesamiento de la señal del sensor **15**
4.1 Ruido . . . 15
4.2 Filtrado de la señal . . . 16

5 Sintonización de PID en posiciones intermedias **17**
5.1 Utilizar solo las ganancias de la menor inercia . . . 17
5.2 Utilizar solo las ganancias de la mayor inercia . . . 17
5.3 Interpolación . . . 18
5.4 Conclusiones . . . 19

6 Bibliografia y Referencias **20**

7 Anexo **21**

1 Introducción

Este proyecto busca aplicar algunos de los conceptos estudiados en la asignatura control y sistemas. Los conceptos se aplican sobre un modelo del robot ABB IRB 7600 presente en el motor de desarrollo en tiempo real Unity, observado en la figura 1.

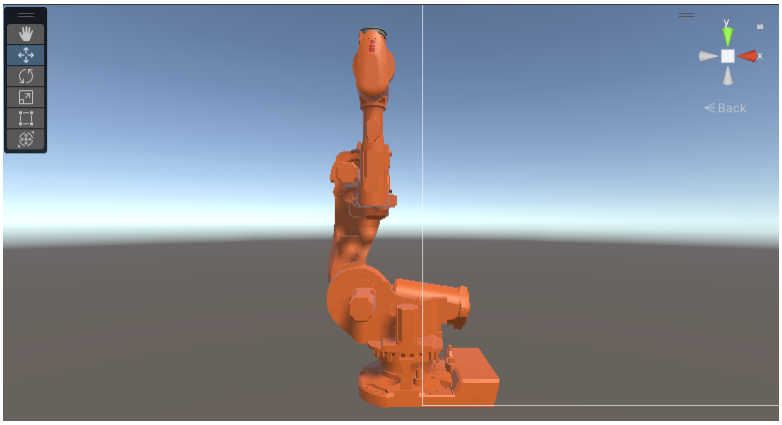
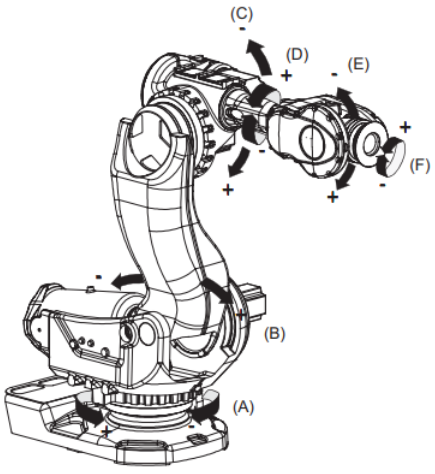


Figure 1: Modelo del robot en Unity

Haciendo uso de las herramientas de Unity el objetivo es controlar el motor asociado al eje 1, el cual es el encargado de mover la base del robot, como se observa en la figura 2, haciendo uso de un controlador tipo PID. Al momento de desarrollar un sistema de control se busca que el mismo sea preciso y robusto.



Pos	Descripción	Pos	Descripción
A	Eje 1	D	Eje 4
B	Eje 2	E	Eje 5
C	Eje 3	F	Eje 6

Figure 2: Movimientos de ejes

Para desarrollar el código de control se utilizó el Entorno de Desarrollo Integrado (IDE) Visual Studio 2019, el cual es utilizado para desarrollar aplicaciones y juegos en diversos lenguajes de programación. Visual Studio es muy utilizado para escribir scripts de C# para Unity, ya que Visual Studio Tools ofrece un amplio conjunto de características que facilitan el trabajo en proyectos de Unity

2 Pruebas previas al proyecto

2.1 Unity

Antes de comenzar con la realización del proyecto, se llevaron a cabo pruebas para verificar que tanto se aproximan las herramientas ofrecidas en Unity a la realidad. Para ello se realizó una prueba que consiste en generar un cuerpo que se sostiene por medio de una articulación, sobre la cual actúa un motor que es capaz de aplicar un torque en la misma. En la figura 3 se muestra el cuerpo con forma de cápsula utilizado para la prueba y sus propiedades (la flecha naranja indica el eje de giro)

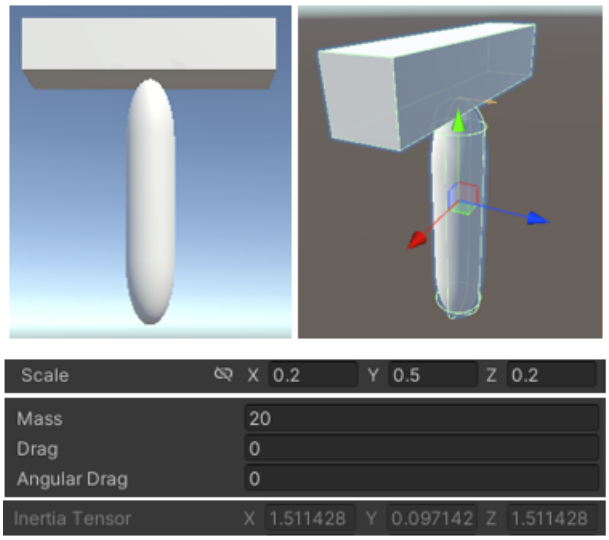


Figure 3: Sistema de pruebas en Unity y sus propiedades

2.2 Simulink

Para simular el sistema en Simulink primero se obtuvo el modelo matemático del sistema monoarticular observado en la figura 4. En la ecuación 1 se observa el modelo dinámico en cuestión, donde θ es la posición angular, M la masa, g la constante de gravedad, L la distancia del centro de masa al eje de giro, τ es el torque aplicado en la articulación e I es el momento de inercia del cuerpo con respecto al eje de giro (el cual se obtiene en la ecuación 2 con el teorema de Steiner, donde I_{CM} es el momento de inercia con respecto al centro de masa).

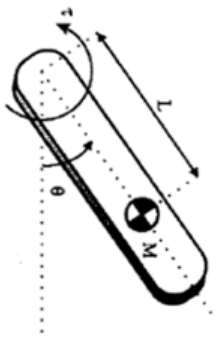


Figure 4: Sistema monoarticular

$$I\ddot{\theta} + MLg\,sin(\theta) = \tau \tag{1}$$

$$I = I_{CM} + ML^2 \tag{2}$$

En la figura 5 se muestran las variables en la base de datos de Simulink y el diagrama de bloques que representa el sistema en cuestión, en donde se incluyen la fuente que introduce la entrada escalón correspondiente al torque y el elemento que permite visualizar la señal de salida.

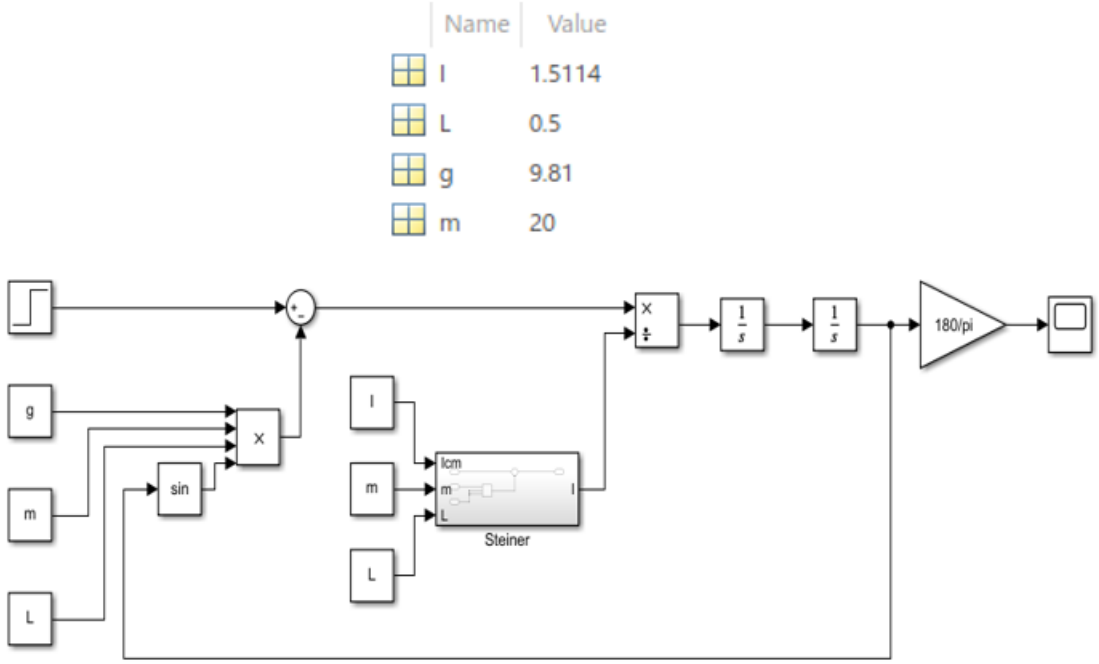


Figure 5: Sistema monoarticular representado en Simulink

2.3 Comparación de resultados

Una vez representado el sistema tanto en Unity como Simulink, se procede a aplicar una entrada de torque τ generada por el motor de 10 Nm y se comparan las respuestas de ambos modelos.
Se observa en la figura 6 que en Unity la propiedad *Motor* cuenta con una variable llamada *Force* que representa la magnitud del torque aplicado por el motor en la articulación en cuestión y otra variable llamada *Target Velocity* la cual es la velocidad objetivo/limite que podrá alcanzar el motor en esa articulación. Como no nos interesa limitar la velocidad le asignamos un valor muy grande y solo nos interesa el signo de esta variable ya que nos indica el sentido de giro en la articulación.

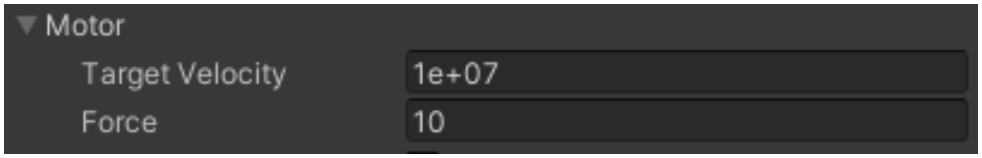


Figure 6: Valores de entrada en Unity

Por otro lado, en Simulink simplemente se aplicó una entrada escalón de 10 Nm en el tiempo $t = 0$s como se observa en la figura 7

Figure 7: Valores de entrada en Simulink

Ahora se inicia la simulación y se compara que tan similar es la respuesta que presenta el modelo de Unity con respecto al modelo de Simulink como se observa en la figura 8. Se observa que la respuesta de ambos sistemas es muy similar en cuanto a amplitud y frecuencia por lo que consideramos que las herramientas de Unity son útiles para representar al sistema real, y por lo tanto, el modelo de Unity puede ser utilizado como planta de prueba que permita obtener resultados aproximados a la realidad.

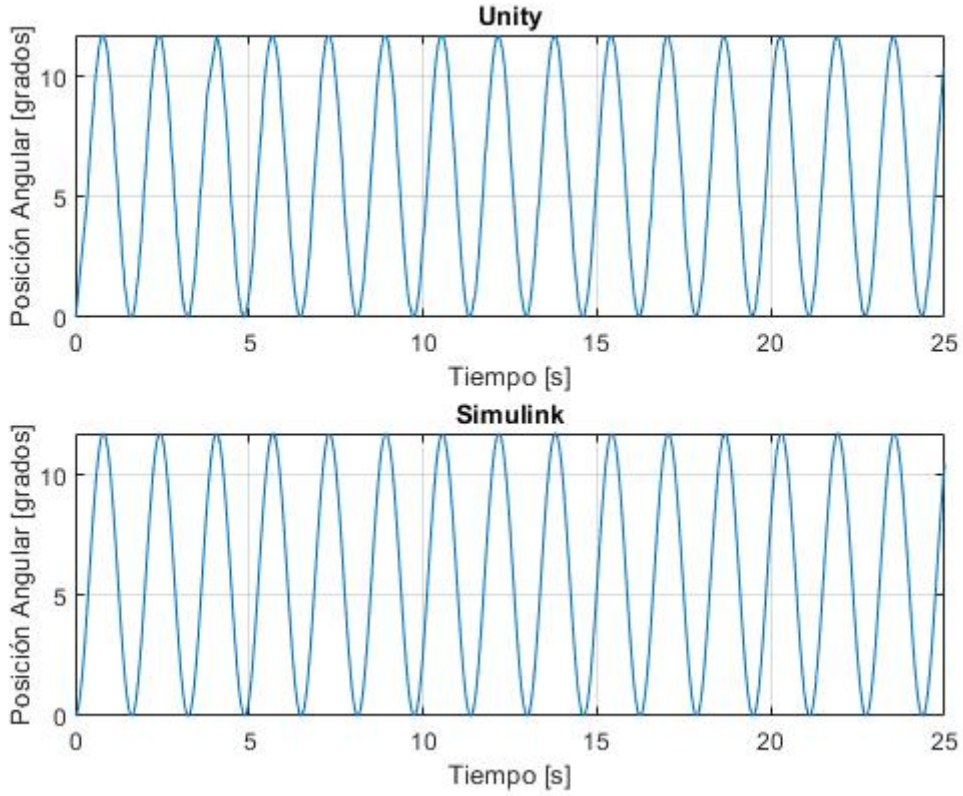


Figure 8: Respuesta del modelo de Unity VS Simulink

3 Control PID

Un controlador PID (controlador proporcional, integral y derivativo) es un mecanismo de control que a través de un lazo de retroalimentación permite regular variables de un proceso. El controlador calcula la diferencia o error $e(t)$, entre nuestra variable real y (en nuestro caso la posición angular actual) en comparación con la variable deseada r (posición angular deseada) y ajusta la salida del proceso u (torque entregado por el motor) para reducir la diferencia.
El controlador se compone de tres elementos que proporcionan una acción Proporcional, Integral y Derivativa, cuyas ganancias se conocen como K_p,K_i y K_d respectivamente como se observa en la ecuación 4.
En la Figura 9 se muestra el diagrama de bloques que representa el funcionamiento de un controlador PID

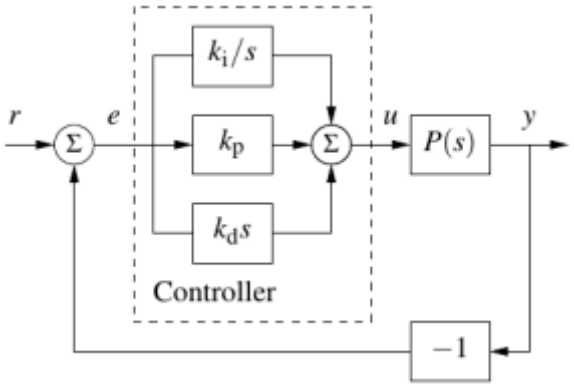


Figure 9: Diagrama de bloques de un controlador PID

$$e(t) = r(t) - y(t) \tag{3}$$

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{4}$$

$$K_i = K_p/T_i \qquad K_d = K_p.T_d \tag{5}$$

$$u(t) = K_p(e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt}) \tag{6}$$

3.1 Método de sintonización

La sintonización de un controlador PID es el proceso de encontrar los valores de las ganancias proporcional, integral y derivativa de un controlador PID para lograr el rendimiento deseado y cumplir con los requisitos de diseño. La sintonización de un controlador PID parece fácil, pero encontrar el conjunto de ganancias que garantice el mejor rendimiento del sistema de control es una tarea compleja.
Como no se cuenta con el modelo matemático de la planta en absoluto, entonces no es posible un enfoque analítico o computacional para el diseño del controlador PID. Entonces se debe recurrir a enfoques experimentales para la sintonización del controlador.
Ziegler y Nichols sugirieron reglas para ajustar los parámetros de controladores PID (valores de Kp, Ti y Td), basándose en el valor de K_p que da como resultado una estabilidad crítica cuando solo se controla proporcionalmente. A esta regla se la conoce como el *Segundo método de Ziegler y Nichols.*

Los pasos para aplicar este método son:

1. Se anulan las ganancias integral $K_i = 0$ y derivativa $K_d = 0$.
2. Se incrementa K_p desde cero hasta un valor critico K_{cr} para el cual la salida presenta oscilaciones sostenidas con amplitud constante y se obtiene el periodo crítico P_{cr} de dicha respuesta, como se muestra en la figura 10.

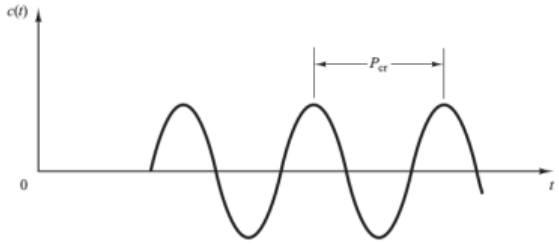


Figure 10: obtención del periodo critico

3. Con K_{cr} y P_{cr} conocidos, se los introduce en la tabla de la figura 11 de acuerdo al tipo de controlador en cuestión (en nuestro caso PID) y obtenemos los valores de K_p, K_i y K_d.

Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Figure 11: Tabla del segundo método Z-N

3.2 Sintonización en la posición de menor inercia

Primero se procede a aplicar este método para la posición del robot que presenta menor inercia, la cual corresponde a todos los eslabones posicionados verticalmente como se muestra en la figura 12. Para aplicar el método se va a introducir una entrada de referencia del 10% para cada valor de K_p y se analiza como varían las amplitudes a lo largo del tiempo.

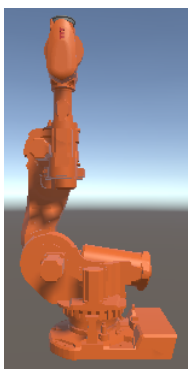


Figure 12: Posición de menor inercia

Se comienza a aumentar la ganancia proporcional hasta alcanzar la ganancia crítica. Se observa en la figura 13 que para $K_p = 20$ y $K_p = 40$ la amplitud de las oscilaciones disminuye considerablemente, para $K_p = 60$ la amplitud aumenta por lo que el sistema se considera inestable, y para $K_p = 50$ la disminución de la amplitud es muy leve. Finalmente adoptamos como ganancia critica a este último valor $K_{cr} = 50$, cuyo periodo crítico es $P_{cr} = 1.9$ s"

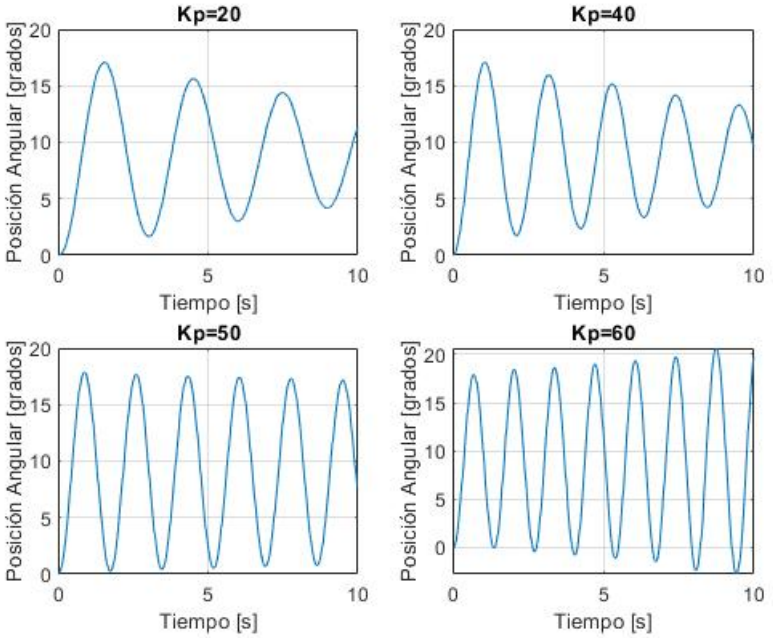


Figure 13: Pruebas para hallar el K_{cr} en la posición de menor inercia

Ahora que se tiene la ganancia crítica y el periodo crítico, se puede calcular el valor de las ganancias del controlador PID haciendo uso de la tabla de la figura 11 y las ecuaciones 5. Se obtienen las siguientes ganancias: $K_p = 30$, $K_i = 31,57$ y $K_d = 7,12$. Como medida de desempeño del controlador se tendrán en cuenta el sobrepaso, o sobreimpulso, y el tiempo de asentamiento, o tiempo de establecimiento, los cuales se muestran a través de un ejemplo en la figura 14

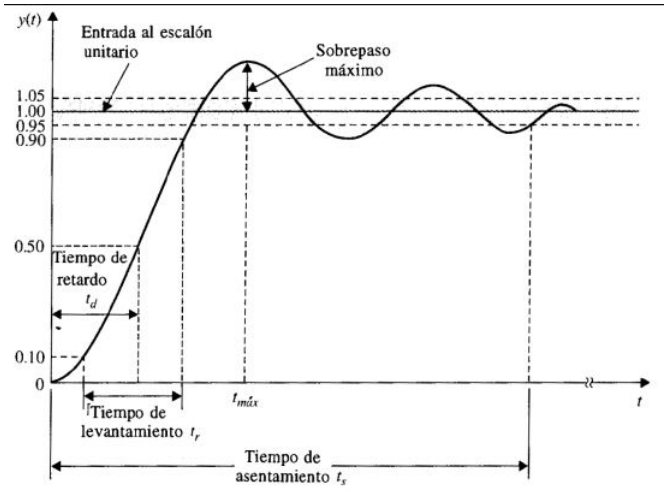


Figure 14: Parámetros para evaluar la calidad del controlador

Con estas nuevas ganancias, la respuesta del sistema se muestra en la figura 15. Se observa que la respuesta es estable, pero presenta un gran sobreimpulso y tiempo de establecimiento debido a que presenta muchas oscilaciones.

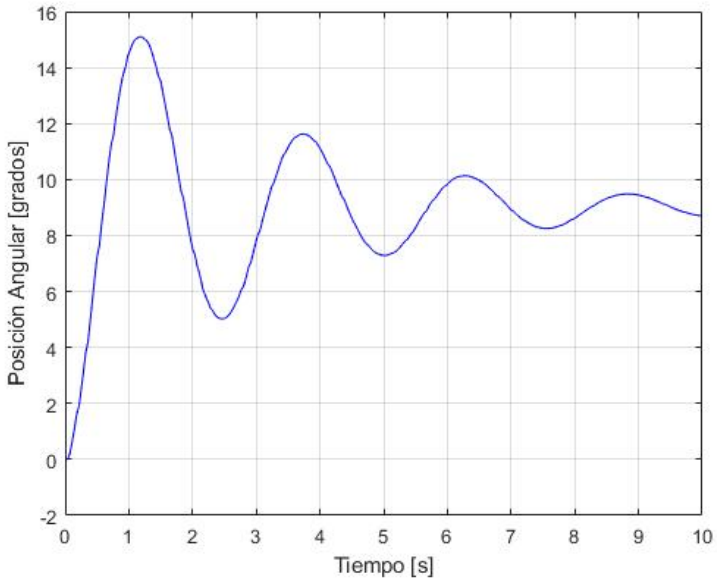


Figure 15: Respuesta en la posición de menor inercia y ganancias obtenidas por Z-N

Para atenuar las oscilaciones se van a modificar los parámetros del PID, buscando alcanzar un sobreimpulso menor al 20% y un tiempo de establecimiento menor a los 10 segundos.
Primero se observan los efectos al modificar el parámetro K_p en la figura 16. Se aprecia que el aumento de K_p tiene gran influencia en la amortiguación de las oscilaciones, pero puede llegar a provocar sobreimpulsos muy grandes si se aumenta demasiado.

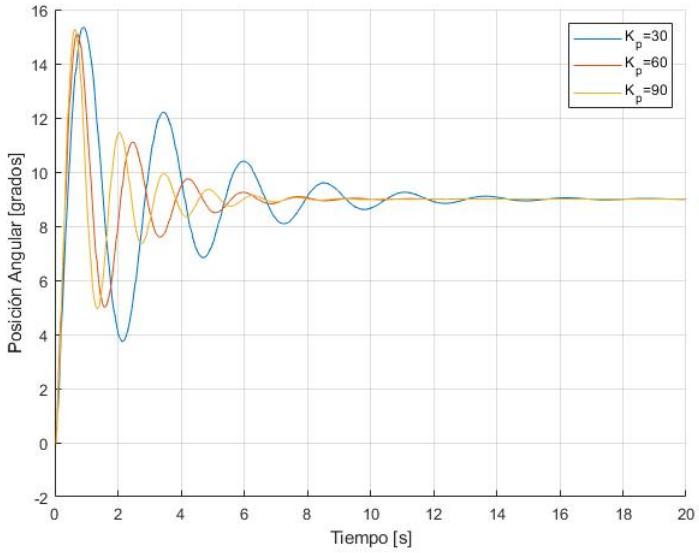


Figure 16: Efectos de K_p en la respuesta

En la figura 17 se observa que aumentar demasiado la ganancia integral aumenta las posibilidades de introducir inestabilidad al sistema, por lo que su modificación debe realizarse con mucha precaución. Ajustar correctamente esta ganancia permite reducir los tiempos de establecimiento, además de eliminar el error en estado estacionario.

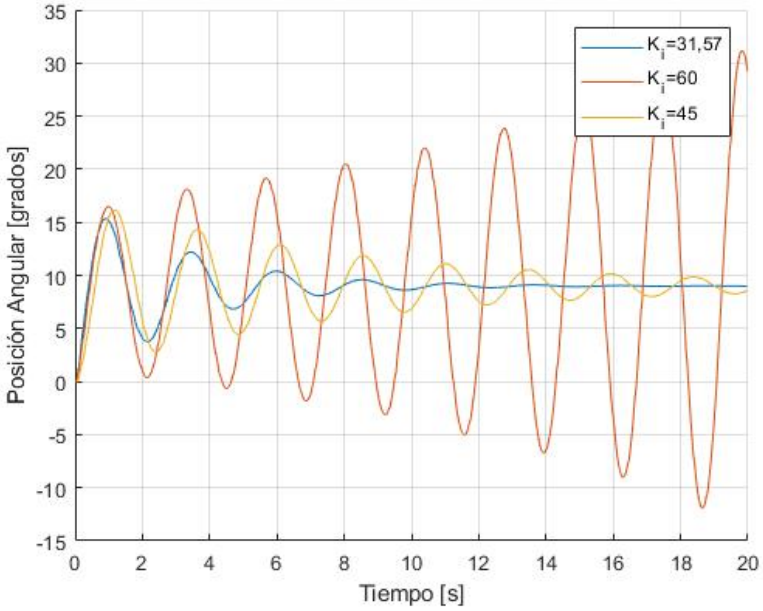


Figure 17: Efectos de K_i en la respuesta

Por ultimo, en la figura 18 se observa que aumentar la ganancia derivativa disminuye considerablemente el sobreimpulso y el tiempo de establecimiento. Sin embargo, un aumento excesivo genera sobreimpulsos mas grandes y tiempos de establecimientos mas altos.

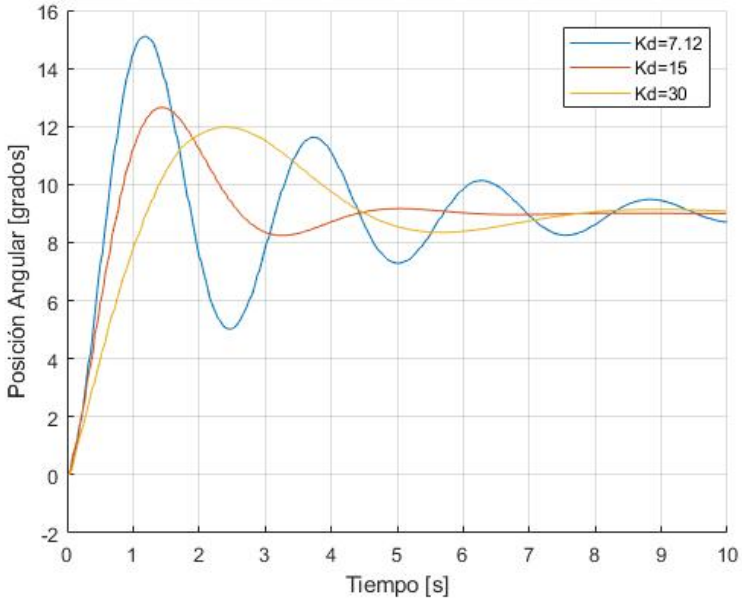


Figure 18: Efectos de K_d en la respuesta

Mediante la realización de pruebas, se observa en la figura 19 la mejor respuesta obtenida que presento un sobreimpulso de 11.28% y un tiempo de establecimiento de 3.17 segundos. Las ganancias del controlador que entregan esta respuesta son: $K_p = 60$, $K_i = 31,57$ y $K_d = 30$.

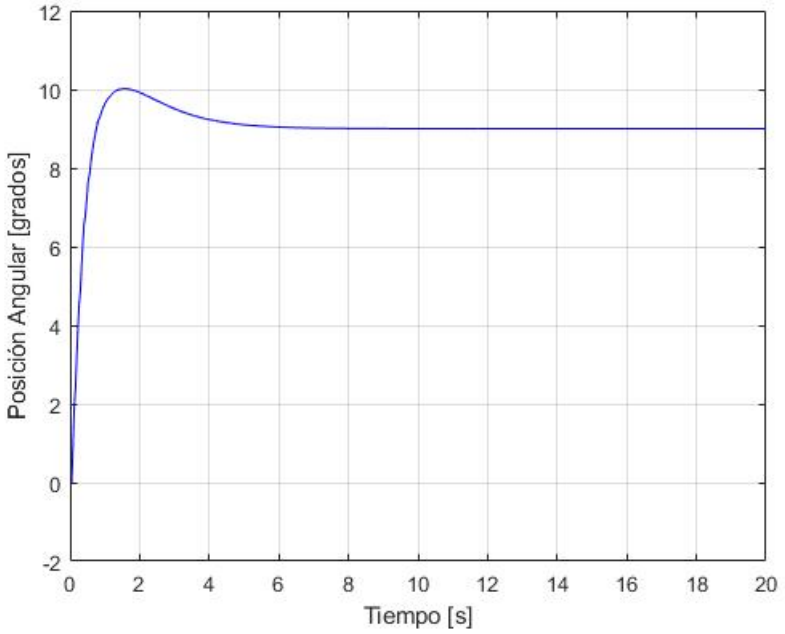


Figure 19: Mejor respuesta alcanzada para la posición de menor inercia

Cabe resaltar que todas las simulaciones se realizan respetando los valores máximos de torque y velocidad que se presentan en el eje 1 de acuerdo al datasheet del robot ABB IRB 7600-500 observado en la figura 20.

Specification

Robot versions IRB	Reach	Handling capacity	Center of gravity	Max. wrist torque
IRB 7600-500	2.55 m	500 kg	360 mm	3010 Nm
IRB 7600-400	2.55 m	400 kg	512 mm	3010 Nm
IRB 7600-340	2.8 m	340 kg	360 mm	2750 Nm
IRB 7600-325	3.1 m	325 kg	360 mm	2680 Nm
IRB 7600-150	3.5 m	150 kg	360 mm	1880 Nm
(IRB 7600-150 loaded with 100 kg			1660 mm)	

Extra loads can be mounted on all variants

50 kg on upper arm and 550 kg on frame of axis 1.

Axis max speed

	325/500kg	400 kg	340 kg	150 kg
Axis 1	75°/s	75°/s	75°/s	100°/s
Axis 2	50°/s	60°/s	60°/s	60°/s
Axis 3	55°/s	60°/s	60°/s	60°/s
Axis 4	100°/s	100°/s	100°/s	100°/s
Axis 5	100°/s	100°/s	100°/s	100°/s
Axis 6	160°/s	160°/s	160°/s	190°/s

A supervision function prevents overheating in applications with intense and frequent

Figure 20: Datasheet del robot ABB IRB 7600

La velocidad máxima es limitada por Unity como se explico en la sección de "Pruebas previas al proyecto". Mientras que el torque es limitado por saturación mediante el Script en #C como se observa en la figura 21.

```
//Aplicamos el Torque
Torque= kp * e + kd * de + ki * ie;

//limitamos el torque maximo por saturacion (Consideramos torque max 3010Nm)
if (Mathf.Abs(Torque) > 3010) {
    if (Torque >= 0)
    {
        Torque = 3010;
    }
    else {
        Torque = -3010;
    }
}

//Aplicamos el torque en unity considerando su magnitud y sentido
if (Torque >= 0)
{
    hingeMotor.targetVelocity = 75;
    hingeMotor.force = Mathf.Abs(Torque);
}
else {
    hingeMotor.targetVelocity = -75;
    hingeMotor.force = Mathf.Abs(Torque);
}
hinge2.motor = hingeMotor;
```

Figure 21: Aplicación de torque por Script

En la figura 22 se muestra que no se superan los valores limites de velocidad y torque en la ultima respuesta obtenida correspondiente a la figura 19.

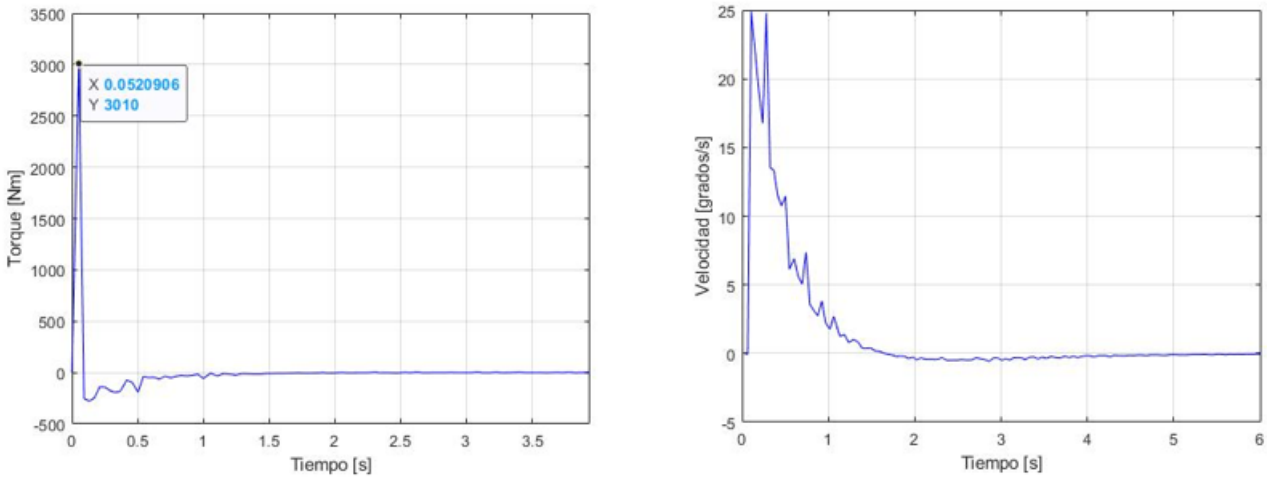


Figure 22: Torque y velocidad en la ultima respuesta alcanzada

3.3 Sintonización en la posición de mayor inercia

Ahora se repite el mismo procedimiento considerando la posición del robot que presenta mayor inercia, en la cual todos los eslabones están posicionados horizontalmente como se muestra en la figura 23. Nuevamente se introduce una referencia del 10% como en el caso anterior.

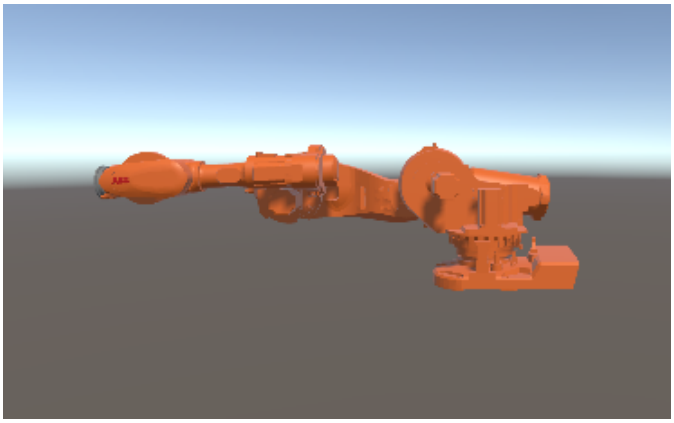


Figure 23: Posición de mayor inercia

Se comienza a aumentar la ganancia proporcional hasta alcanzar la ganancia crítica. Vemos que para $K_p = 30$ y $K_p = 80$ la amplitud de las oscilaciones disminuye considerablemente y para $K_p = 95$ la variación de la amplitud es muy leve. Finalmente adoptamos como ganancia critica a este último valor $K_{cr} = 95$ cuyo periodo crítico es $P_{cr} = 3,77s$

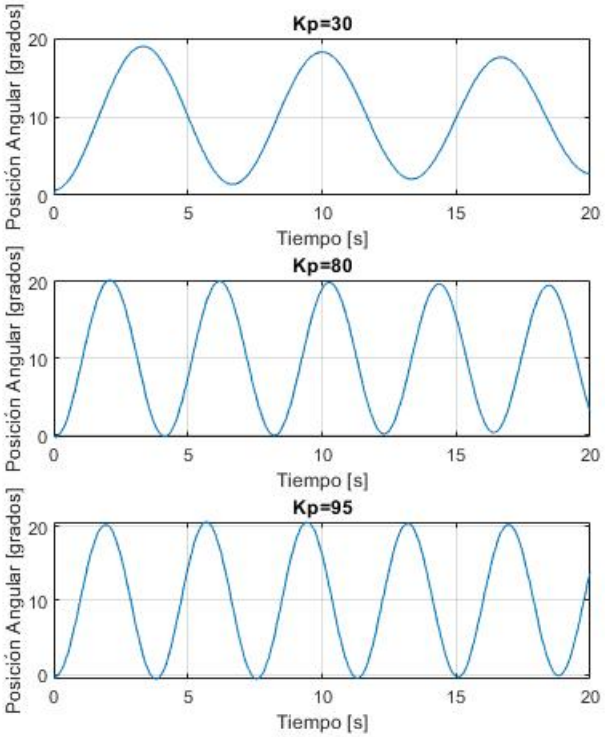


Figure 24: Pruebas para hallar el K_{cr} en la posición de mayor inercia

Ahora que tenemos la ganancia crítica y el periodo crítico podemos obtener el valor de las ganancias del controlador PID haciendo uso de la tabla de la 11. Se obtienen las siguientes ganancias: $K_p = 57$, $K_i = 30,24$ y $K_d = 26,86$.
Con estas nuevas ganancias la respuesta del sistema se muestra en la figura 25. Donde se observa que la respuesta es estable pero presenta un gran sobreimpulso y tiempo de establecimiento debido a que presenta muchas oscilaciones.

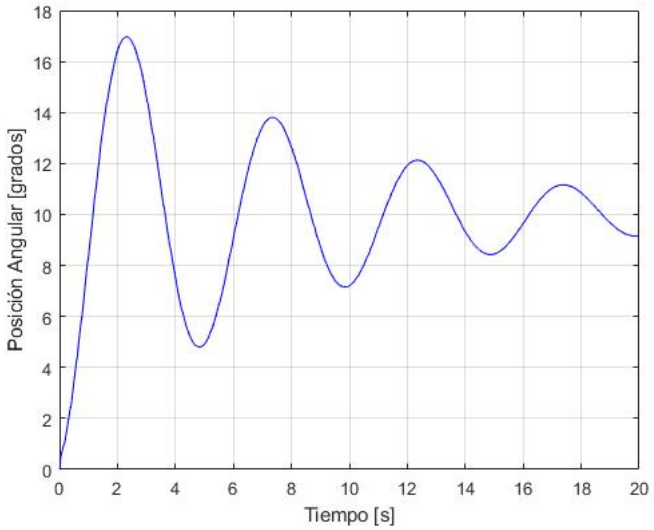


Figure 25: Respuesta en la posición de mayor inercia y ganancias obtenidas por Z-N

Los efectos al cambiar las ganancias son similares a los vistos anteriormente para la posición de menor inercia, por lo que se realizaron ajustes siguiendo la metodología anterior para obtener una respuesta con menor sobreimpulso y tiempo de establecimiento.
En la figura 26 se observa que con los valores de $K_p = 150$,$K_i = 30,24$ y $K_d = 100$, se logro una respuesta con un sobreimpulso del 16% y un tiempo de establecimiento de 5.62 segundos

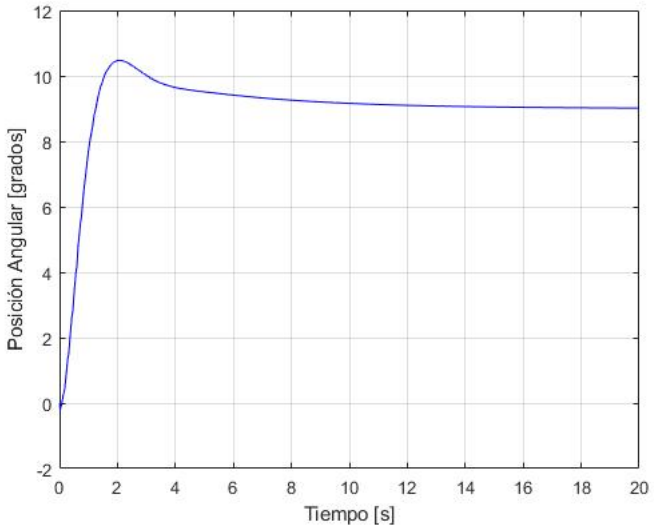


Figure 26: Respuesta en la posición de mayor inercia y ganancias corregidas

4 Procesamiento de la señal del sensor

4.1 Ruido

Los sensores angulares son dispositivos que miden la posición angular de un objeto. El ruido en los sensores angulares puede ser causado por una variedad de factores, como la interferencia electromagnética.
Unity nos permite acceder directamente a la posición angular del eslabón que estamos controlando, por lo que se trataría de un sensor ideal. Sin embargo, con el objetivo de generar una situación más realista consideraremos un sensor serie HA-D200 obtenido del catálogo 6 el cual tiene una precisión de ±2".
Se va a considerar que debido a la interferencia electromagnética de otros equipos en el ambiente de trabajo, se induce ruido que provoca una variación ±1° a la señal entregada por el sensor. Esto se simula a través del Script, como se observa en la figura 27.

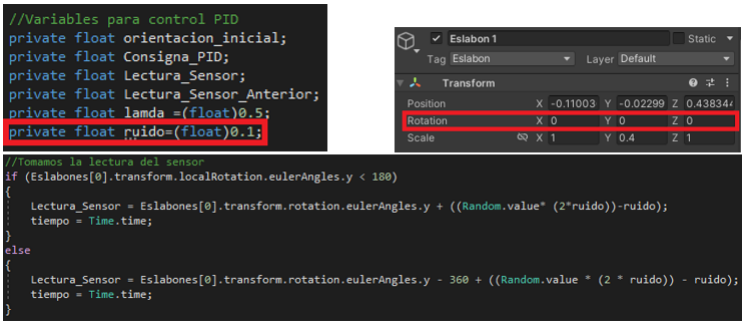


Figure 27: Introducción de ruido a la señal proveniente del sensor

La respuesta del sistema con menor inercia (brazo totalmente vertical) en presencia del ruido se observa en la figura 28. Se pierde mucha precisión en la respuesta debido a que se presentan oscilaciones considerables alrededor de la posición de referencia.

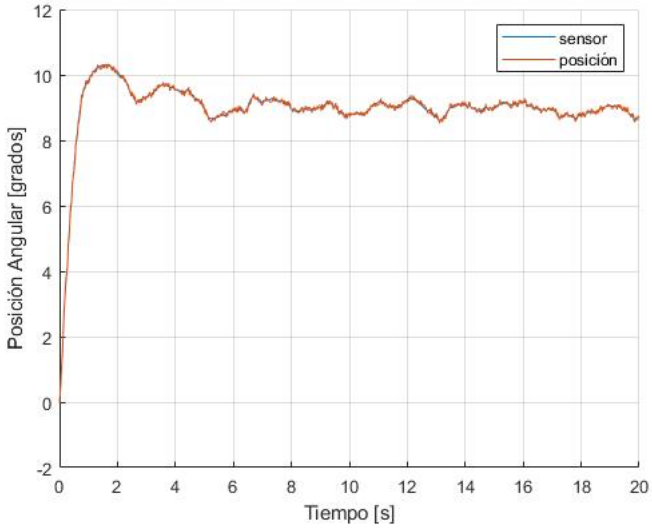


Figure 28: Efectos del ruido en la respuesta

4.2 Filtrado de la señal

Los filtros se utilizan para reducir el ruido en las señales de los sensores atenuando las frecuencias no deseadas en una señal. Hay varios tipos de filtros que se pueden utilizar para reducir el ruido en las señales de los sensores angulares, como los filtros pasa-bajos y los filtros pasa-altos
Para atenuar las continuas desviaciones alrededor de la entrada de referencia vamos a aplicar un filtro IIR en el dominio del tiempo llamado Leaky integrator Un filtro IIR, además de utilizar retrasos para los valores a la entrada del filtro (como en los filtros FIR), toma también los valores de la salida, les aplica una nueva cadena de retrasos y retroalimenta esta señal a la entrada del filtro. Esta retroalimentación es la que permite que una respuesta impulso se extienda por un tiempo infinito.
La ecuación 5 corresponde al filtro Leaky integrator, donde $x[n]$ es la señal de entrada al filtro o la señal medida por el encoder angular, $y[n]$ es la señal filtrada o salida del filtro, $y[n-1]$ es la señal filtrada en el instante anterior y λ es un factor que determina el grado de suavizado de la señal.

$$y[n] = \lambda\, y[n-1] + (1-\lambda)\, x[n] \tag{7}$$

En la figura 29 se observan las respuestas para distintos valores de λ. Para $\lambda = 0,5$ las oscilaciones se reducen considerablemente y prácticamente no presenta retardo con respecto a la señal real, por otro lado para $\lambda = 0,75$ y $\lambda = 0,9$ el suavizado de la señal es mayor, pero también se presentan mayores retardos con respecto a la señal real lo que induce oscilaciones no deseadas por parte de la acción de control. En conclusión, debido al análisis realizado, vamos a adoptar $\lambda = 0,5$ como factor de filtrado en las siguientes pruebas.

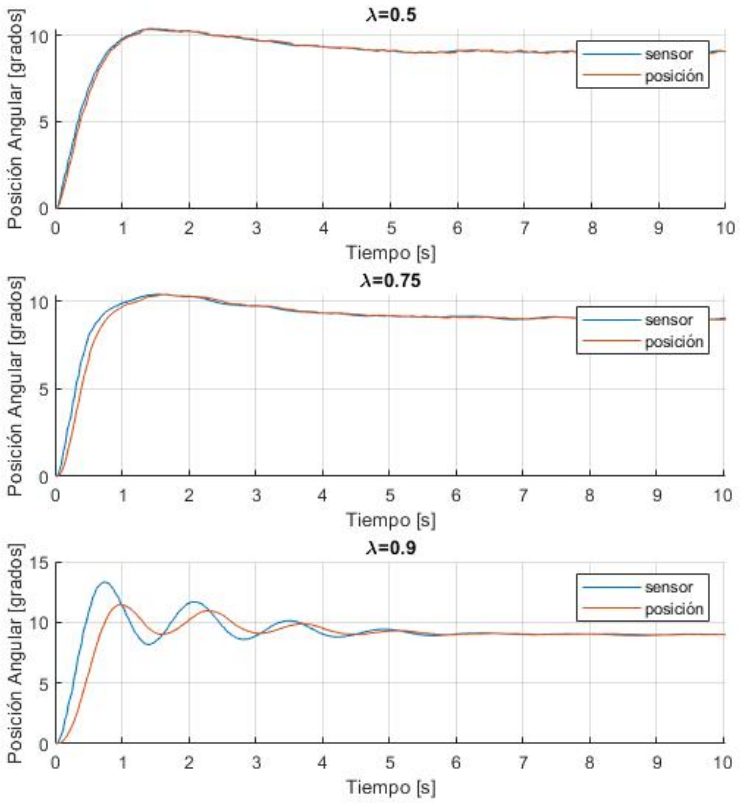


Figure 29: Respuestas frente a distintos factores de suavizado

5 Sintonización de PID en posiciones intermedias

Hasta ahora solo se ha realizado la sintonización del PID solo para dos posiciones, la de menor inercia y la de mayor inercia. En esta sección propondremos alguno de los métodos para asignar valores a las ganancias del PID en las posiciones intermedias y verificar experimentalmente si dan resultados.

5.1 Utilizar solo las ganancias de la menor inercia

Inicialmente consideramos utilizar las ganancias obtenidas en la sintonización del PID para la posición de menor inercia, las cuales eran: $K_p = 60$, $K_i = 31,57$ y $K_d = 30$. A estas ganancias las vamos a utilizar para aplicar el control en el resto de posiciones que presenta el robot.
En la tabla de la figura 30 se observan los sobreimpulsos y tiempos de establecimientos para cada posición.

$\theta_1 \backslash \theta_2$	0°	15°	30°	45°	60°	75°	90°
0°	11.72% 3.76	14.72% 3.56	17.6% 3.39	20.77% 3.35	23.51% 2.58	30.65% 2.97	27.40% 2.59
15°	22.70% 2.13	29.63% 2.39	34.21% 2.63	39.98% 3.46	47.91% 3.98	41.71% 3.94	51.80% 4.19
30°	41.16% 4.00	43.44% 4.21	58.95% 6.41	51.08% 6.48	57.55% 6.65	59.40% 6.71	55.85% 6.47
45°	56.85% 8.59	63.45% 9.07	76.46% 11.31	72.96% 11.51	61.56% 11.28	60.67% 10.82	66.70% 8.99
60°	75.92% 13.83	75.96% 16.57	70.85% 16.52	68.97% 16.49	71.82% 14.25	79.87% 13.74	63.79% 10.84
75°	97.75% 19.99	74.22% 19.83	80.49% 19.97	84.09% 19.48	73.45% 16.42	79.18% 13.83	72.35% 11.33
90°	87.01% >20	88.24% >20	73.73% 19.49	63.31% 16.68	87.45% 14.40	82.49% 13.37	61.42% 8.83

Figure 30: Resultados utilizando las ganancias de menor inercia

Se logra apreciar que a medida que nos alejamos hacia las posiciones de mayor inercia los sobreimpulsos y tiempos de establecimientos se hacen cada vez mas grande, llegando a alcanzar valores de 97% y mayores a 20 segundos, respectivamente. Debido a estos grandes valores se descartaría este método por el momento.

5.2 Utilizar solo las ganancias de la mayor inercia

De forma similar al método anterior, ahora consideramos en utilizar las ganancias obtenidas en la sintonización del PID para la posición de mayor inercia, las cuales eran: $K_p = 150$,$K_i = 30,24$ y $K_d = 100$; y las mantenemos para el resto de posiciones.
En la tabla de la figura 31 se observan los sobreimpulsos y tiempos de establecimientos para cada posición. Se observa que presenta bajos valores sobreimpulsos incluso estando en las posiciones de menor inercia, lo cual es muy bueno. Sin embargo, si observamos los tiempos de establecimiento en las zonas de menor inercia pasan a ser aproximadamente el doble de los vistos en el método anterior. A pesar de esto ultimo, parece ser un método valido para controlar el robot en todas las posiciones.

$\theta_1 \backslash \theta_2$	0°	15°	30°	45°	60°	75°	90°
0°	9.24% 6.34	9.54% 7.08	9.19% 8.03	8.82% 7.39	9.54% 6.00	10.27% 7.46	9.27% 7.42
15°	10.88% 6.50	9.28% 5.50	10.04% 6.17	8.50% 7.71	10.4% 6.15	12.13% 6.33	9.41% 7.41
30°	11.12% 6.34	9.42% 6.32	10.36% 7.26	10.32% 6.47	11.35% 5.89	11.84% 6.02	10.01% 17.92
45°	10.94% 8.05	11.61% 6.65	14.23% 6.40	15.28% 6.20	13.03% 6.14	12.32% 7.01	11.35% 6.77
60°	13.53% 6.23	14.38% 6.52	14.56% 5.43	16.34% 6.28	13.78% 4.79	13.50% 5.31	10.99% 5.60
75°	16.27% 6.48	16.39% 6.62	16.36% 7.03	16.29% 12.15	16.43% 7.30	12.59% 5.98	11.99% 5.99
90°	15.93% 6.02	17.81% 6.47	16.46% 6.04	16.92% 6.24	16.13% 6.90	12.80% 7.12	11.67% 7.70

Figure 31: Resultados utilizando las ganancias de mayor inercia

5.3 Interpolación

Implementaremos una interpolación para las tres ganancias K_p, K_i y K_d; considerando como valores límites a las encontradas anteriormente para el robot con la menor inercia, que pasaran a llamarse K_{min} y las de mayor inercia como K_{max}.
Los valores intermedios K serán asignados de acuerdo a los ángulos absolutos de los ejes más significativos del robot θ_1 y θ_2 como se muestra en la ecuación 8, siendo θ_{21} el ángulo del eslabón dos referido al eslabón uno, como se muestra en la ecuacion 9. Los eslabones más significativos se observan en la Figura 32.

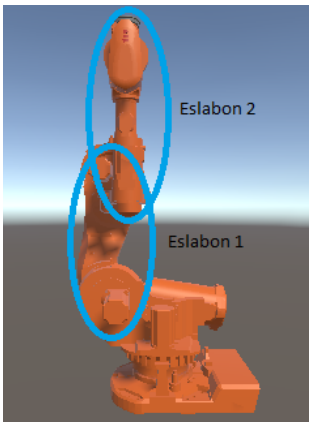


Figure 32: Eslabones más significativos en la inercia del sistema

$$K = \frac{|sin(\theta_1) + sin(\theta_2)|}{2}(K_{max} - K_{min}) + K_{min} \tag{8}$$

$$\theta_2 = \theta_1 + \theta_{21} \tag{9}$$

En la tabla de la figura 33 se muestran los resultados. Los resultados nos muestran que presenta una leve desventaja con respecto al método anterior en cuanto a los sobreimpulsos, principalmente en las zonas de mayor inercia. Por otro lado, presenta grandes mejoras con respecto al tiempo de establecimiento siendo mas rápido en la mayoría de posiciones.

$\theta_1 \backslash \theta_2$	0°	15°	30°	45°	60°	75°	90°
0°	10.61% 3.54	14.63% 4.78	12.64% 4.61	15.27% 5.69	11.32% 4.99	13.13% 5.96	11.64% 5.62
15°	9.96% 3.82	11.41% 5.11	9.96% 4.81	13.74% 6.05	8.40% 3.99	12.64% 6.24	11.13% 6.02
30°	12.20% 5.29	14.97% 6.60	11.05% 5.18	12.02% 5.59	13.62% 6.36	13.82% 6.68	9.17% 5.26
45°	12.29% 5.26	14.99% 6.82	13.81% 5.72	13.36% 5.48	16.20% 6.50	15.96% 5.99	16.48% 5.80
60°	13.78% 5.97	11.57% 5.22	9.89% 4.52	9.16% 4.05	16.36% 5.62	15.43% 5.24	19.66% 6.09
75°	17.09% 5.73	13.97% 5.14	17.60% 5.92	17.25% 5.56	19.72% 5.99	15.48% 4.69	17.21% 4.38
90°	16.18% 6.31	17.82% 6.79	9.74% 3.47	23.24% 6.57	16.11% 4.60	24.63% 5.72	23.48% 4.78

Figure 33: Resultados utilizando interpolación de ganancias

Es importante tener en cuenta que este método puede ser mejorado aumentando la cantidad de semillas en la interpolación, por ejemplo aplicando Ziegler-Nichols y ajustes en la posición de 45°. Esto nos permitiría trabajar con ganancias mas eficientes en las posiciones intermedias.
Otra forma de mejora es ajustando la influencia que presenta la posición de cada eslabón en la ecuación 8. Por ejemplo el eslabón 1 al tener mas masa podría influir en un 60% en la determinación de las ganancias K, mientras que el eslabón 2 solo influye en un 40%.

5.4 Conclusiones

La selección del mejor de los tres métodos presentados va a depender del tipo de aplicación a la que será destinado el robot.
El método que utiliza las ganancias de menor inercia presenta como ventaja su sencillez y que en las zonas de menor inercia presenta buenos tiempos de establecimiento. Podría utilizarse en aplicaciones donde el robot no se aleje demasiado de las posiciones de menor inercia, donde se necesite una rápida velocidad de respuesta y no se requiera mucha precisión debido a que los sobreimpulsos son demasiado altos
El segundo método, el de las ganancias de mayor inercia, es igual de sencillo que el anterior con la ventaja de que podría utilizarse para todas las posiciones del robot. Es el mejor de los tres métodos en aplicaciones donde se requiera mucha precisión debido a que presenta muy bajos valores de sobreimpulso. Su aspecto mas débil se observa en la velocidad de respuesta debido a que aumentan los tiempos de asentamiento en las zonas de menor inercia con respecto al método anterior.
El tercer método es un muy buen equilibrio entre rapidez y precisión, ya que presenta buenos resultados en ambos aspectos.

6 Bibliografia y Referencias

- Datasheet Robot ABB 7600:
  https://www.irsrobotics.com/wp-content/uploads/2017/11/Data-sheet-IRB-7600-IRC5.pdf
- Manual de Robot ABB 7600:
  https://search.abb.com/library/Download.aspx?DocumentID=3HAC023934-005&LanguageCode=es&DocumentPartId=M2004&Action=Launch/
- Información sobre métodos de control:
  https://controlautomaticoeducacion.com/control-realimentado/ziegler-nichols-sintonia-de-conti
- Sensores en robótica:
  http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/robotica/sistema/sensores.htm#posicion
- Manual de Unity:
  https://docs.unity3d.com/es/530/Manual/UnityManual.html
- Catalogo de Encoders:
  https://www.interempresas.net/FeriaVirtual/Catalogos_y_documentos/2526/Cat_Encoders_ES.pdf
- Material de clase de la cátedra “Control y Sistemas”

7 Anexo

A continuación, se expone el código fuente en C# utilizado para aplicar el control PID en Unity.

Listing 1: Codigo C♯

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using System.IO;

public class PID : MonoBehaviour
{
    public TMP_Dropdown Selector_eslabones;
    public TMP_Text Selector;
    public Slider Deslizador;
    //public GameObject BaseAR;

    private GameObject Robot_prefab;
    private GameObject Robot;
    private GameObject Base;
    private GameObject[] Eslabones;

    private float[] posiciones_deslizador;
    private int N_eslabon_anterior;
    private float posicion;

    //Variables para control PID
    private float orientacion_inicial;
    private float Consigna_PID;
    private float Lectura_Sensor;
    private float Lectura_Sensor_Anterior;
    private float lamda =(float)0.5;
    private float ruido=(float)0.1;

    private float Angulo1;
    private float Angulo2;

    private float e;
    private float e_ant;
    private float de;
    private float ie;

    private float kp;
    private float ki;
    private float kd;

    private float kpmin= (float)60;
    private float kpmax = (float)150;
    private float kimin=(float)31.57;
    private float kimax=(float) 30.24;
    private float kdmin=(float)30;
```

```
private float kdmax= (float)100;

private float tiempo;
private float tiempoAnt;
private float deltaTime;

private float Torque;
private float Tmax;  //Para visualizar el torque maximo ->
    descomentar su seccion del codigo

private float Spring1 = 0;
private float Spring2 = 0;
private bool inicio = true;
private bool pruebas = false; //Activa el modo pruebas
private bool secuencia = false; //Activa el modo secuencia de
    pruebas
private float T_inicial;
private float T_prueba;

string path;
string path2;
string path3;     //Para almacenar en un .txt la posicion angular
    real -> descomentar su seccion del codigo
string path4;
public List<string> myList = new List<string>();
public List<string> myList2 = new List<string>();
public List<string> myList3 = new List<string>();
public List<string> myList4 = new List<string>();

// Start is called before the first frame update
void Start()
{
    //Desactivamos el robot prefabricado
    //Robot_prefab = GameObject.Find("Robot");
    //Robot_prefab.SetActive(false);

    //Instanciamos el una copia del robot que sirve para simular
    //Robot = Instantiate(Robot_prefab);
    //Destroy(Robot.GetComponent<No_Destruir>());
    //Robot.transform.SetParent(BaseAR.transform);
    //Robot.SetActive(true);

    //Auxiliar
    Robot = GameObject.Find("Robot");

    //Obtenemos el gameobject de la base
    Base = GameObject.FindGameObjectWithTag("Base");
    //Armamos una lista con los eslabones (No olvidar poner el tag
        al eslabon)
    Eslabones = GameObject.FindGameObjectsWithTag("Eslabon");

    //Anulamos el congelamiento de las posiciones
    //foreach (GameObject eslabon in Eslabones)
    //{
```

```
        //      Rigidbody rb = eslabon.GetComponent<Rigidbody>();
        //      rb.constraints = RigidbodyConstraints.None;
        //}

        //Actualizamos la lista de seleccion de eslabones
        Selector_eslabones.ClearOptions();
        //Create la lista con las nuevas opciones
        List<string> Options = new();
        foreach (GameObject eslabon in Eslabones)
        {
            Options.Add(eslabon.name);
        }
        //Aderimos las nuevas opciones a la lista
        Selector_eslabones.AddOptions(Options);

        //Armamos lista q almacena los valores de posicion de cada
            eslabon

        posiciones_deslizador = new float[Eslabones.Length+1];
        for (int i = 1; i < posiciones_deslizador.Length; i++)
        {
            posiciones_deslizador[i] = (float)0.5;
        }

        //Almacenamos el valor de la posicion inicial del eslabon con
            PID
        orientacion_inicial = Eslabones[0].transform.rotation.
            eulerAngles.y;

        //Comenzamos seleccionando el eslabon 1
        N_eslabon_anterior = 1;
    }

    // Update is called once per frame
    void Update()
    {
        //Vuforia activa los renderers de todos los objetos
        //Desactivamos el cilindro de la base durante la ejecucion
        //Base.GetComponent<Renderer>().enabled = false;

        foreach (GameObject eslabon in Eslabones)
        {
            //Desactivamos las capsulas de los eslabones
            //eslabon.GetComponent<Renderer>().enabled = false;

            //Obtenemos el eslabon correspondiente
            if (Selector.text == eslabon.name)
            {
                int N_eslabon_actual = (int)char.GetNumericValue(
                    eslabon.name[8]);

                //Actualizamos el valor de la barra deslizadora si
                    cambiamos de eslabon
                if (N_eslabon_actual != N_eslabon_anterior)
```

```
            {
                Deslizador.value = posiciones_deslizador[
                    N_eslabon_actual];
            }

            //Almacenamos la posicion del deslizador para la
                articulacion en cuestion
            posiciones_deslizador[N_eslabon_actual] = Deslizador.
                value;

            //Calculamos la posicion objetivo
            posicion = (eslabon.GetComponent<HingeJoint>().limits.
                max - eslabon.GetComponent<HingeJoint>().limits.min
                ) * Deslizador.value + eslabon.GetComponent<
                HingeJoint>().limits.min;
            //Posicion objetivo aplicada al eslabon
            HingeJoint hinge = eslabon.GetComponent<HingeJoint>();
            JointSpring hingeSpring = hinge.spring;
            hingeSpring.targetPosition = posicion;
            hinge.spring = hingeSpring;

            //Actualizamos el setpoint
            if (N_eslabon_actual == 1)
            {
                Consigna_PID = orientacion_inicial + (eslabon.
                    GetComponent<HingeJoint>().limits.max - eslabon
                    .GetComponent<HingeJoint>().limits.min) *
                    Deslizador.value + eslabon.GetComponent<
                    HingeJoint>().limits.min;
            }

            //Almacenamos el eslabon actual para la siguiente
                iteracion
            N_eslabon_anterior = N_eslabon_actual;
        }
    }

    //Fijar consigna PID para analizar entrada escalon (si el modo
        pruebas=true -> el deslizador de unity no funcionara)
    if (T_prueba<10 && pruebas==true)
    {
        Consigna_PID = 0;
        //Forzar pos 0 en eje 1
        Eslabones[0].transform.rotation = Quaternion.identity;
    }
    else if(T_prueba >= 10 && pruebas == true)
    {
        Consigna_PID = 9;
    }

    //Aplicamos el control PID al eslabon 1
    //Tomamos la lectura del sensor
    if (Eslabones[0].transform.localRotation.eulerAngles.y < 180)
    {
```

```
        Lectura_Sensor = Eslabones[0].transform.rotation.
            eulerAngles.y + ((Random.value* (2*ruido))-ruido);
        tiempo = Time.time;
    }
    else
    {
        Lectura_Sensor = Eslabones[0].transform.rotation.
            eulerAngles.y - 360 + ((Random.value * (2 * ruido)) -
            ruido);
        tiempo = Time.time;
    }

    //Aplicamos el filtro leaky integrator
    Lectura_Sensor = lamda * Lectura_Sensor_Anterior + (1 - lamda)
        * Lectura_Sensor;

    //Obtenemos la posicion angular de los 2 eslabones mas
        relevantes
    if (Eslabones[1].transform.localRotation.eulerAngles.z < 180)
    {
        Angulo1 = Eslabones[1].transform.rotation.eulerAngles.z;
    }
    else
    {
        Angulo1 = Eslabones[1].transform.rotation.eulerAngles.z -
            360;
    }

    if (Eslabones[2].transform.localRotation.eulerAngles.z < 180)
    {
        Angulo2 = Eslabones[2].transform.rotation.eulerAngles.z;
    }
    else
    {
        Angulo2 = (Eslabones[2].transform.rotation.eulerAngles.z -
            360);
    }

    //Ajustamos las ganancias por metodo interpolacion
    kp = Mathf.Abs(Mathf.Sin(Angulo1 * Mathf.PI / 180) + Mathf.Sin(
        Angulo2 * Mathf.PI / 180)) / 2 * (kpmax - kpmin) + kpmin;
    ki = Mathf.Abs(Mathf.Sin(Angulo1 * Mathf.PI / 180) + Mathf.Sin(
        Angulo2 * Mathf.PI / 180)) / 2 * (kimax - kimin) + kimin;
    kd = Mathf.Abs(Mathf.Sin(Angulo1 * Mathf.PI / 180) + Mathf.Sin(
        Angulo2 * Mathf.PI / 180)) / 2 * (kdmax - kdmin) + kdmin;

    /*
    //Ajustamos las ganancias por metodo de inercia minima
    kp = (float)kpmin;
    ki = (float)kimin;
    kd = (float)kdmin;
    */

    /*
```

```
//Ajustamos las ganancias por metodo de inercia maxima
kp = (float)kpmax;
ki = (float)kimax;
kd = (float)kdmax;
*/

//Calculamos el paso
deltaTime = tiempo - tiempoAnt;

//Aplicamos el control PID
if (deltaTime > 0) {
    //Calculamos el error
    e = Consigna_PID - Lectura_Sensor;
    //Calculamos la integral del error
    ie = (e + e_ant) * deltaTime / 2 + ie;
    //Calculamos la derivada del error
    de = (e - e_ant) / deltaTime;

    //Aplicamos el control en unity
    HingeJoint hinge2 = Eslabones[0].GetComponent<HingeJoint>()
        ;
    JointMotor hingeMotor = hinge2.motor;

    //Aplicamos el Torque
    Torque= kp * e + kd * de + ki * ie;

    //limitamos el torque maximo por saturacion (Consideramos
        torque max 3010Nm)
    if (Mathf.Abs(Torque) > 3010) {
        if (Torque >= 0)
        {
            Torque = 3010;
        }
        else {
            Torque = -3010;
        }
    }

    //Aplicamos el torque en unity considerando su magnitud y
        sentido
    if (Torque >= 0)
    {
        hingeMotor.targetVelocity = 75;
        hingeMotor.force = Mathf.Abs(Torque);
    }
    else {
        hingeMotor.targetVelocity = -75;
        hingeMotor.force = Mathf.Abs(Torque);
    }
    hinge2.motor = hingeMotor;

    //Mostrar Tmax
    /*
    if (Mathf.Abs(hingeMotor.force) > Mathf.Abs(Tmax)) {
```

```
    Tmax = hingeMotor.force;
    Debug.Log(Tmax);
}*/

//Graficas
//Acomodamos los eslabones
HingeJoint hinge = Eslabones[1].GetComponent<HingeJoint>();
JointSpring hingeSpring = hinge.spring;
hingeSpring.targetPosition = Spring1;
hinge.spring = hingeSpring;

hinge = Eslabones[2].GetComponent<HingeJoint>();
hingeSpring = hinge.spring;
hingeSpring.targetPosition = Spring2;
hinge.spring = hingeSpring;

//Generamos el path
float numero = Spring1;
string cadena = numero.ToString();

numero = Spring2;
cadena=cadena+"-"+numero.ToString();

path = Application.dataPath + "/Resources/" + cadena + "
    _tiempo.txt";
path2 = Application.dataPath + "/Resources/" + cadena + "
    _sensor.txt";

//Damos tiempo suficiente para que se acomoden los
    eslabones
if (inicio == true && Spring2 <= 90) {
    T_inicial = tiempo;
    inicio = false;
}

T_prueba = tiempo - T_inicial;

if (T_prueba >= 10 && T_prueba<=30 && pruebas==true) {
    //Almacenamos el tiempo
    Datos(T_prueba-10, path, myList);

    //Almacenamos la posicion medida por el sensor
    numero = Lectura_Sensor;
    if (numero > 180)
    {
        numero = numero - 360;
    }
    Datos(numero, path2, myList2);

    //Almacenar posicion real
    /*
    path3 = Application.dataPath + "/Resources/rotacion.txt
        ";
    numero = Eslabones[0].transform.localEulerAngles.y;
```

```
                if (numero > 180) {
                    numero = numero - 360;
                }
                Datos(numero, path3, myList3);
                */

                /*
                //Almacenar torque
                path3 = Application.dataPath + "/Resources/" + cadena +
                    "_torque.txt";
                numero = Torque;
                Datos(numero, path3, myList3);
                */

                /*
                //Almacenar velocidad
                path4 = Application.dataPath + "/Resources/" + cadena +
                    "_velocidad.txt";
                numero = (Lectura_Sensor-Lectura_Sensor_Anterior)/
                    deltaTime;
                Datos(numero, path4, myList4);
                */
            }
            else if (T_prueba >= 20 && Spring2 <= 90) {
                Debug.Log(cadena + "Grafica finalizada");
                if (secuencia == true) {
                    myList.Clear();
                    myList2.Clear();
                    Spring2 = Spring2 + 15;
                    inicio = true;
                    if (Spring2>90 && Spring1 < 90) {
                        Spring1 += 15;
                        Spring2 = 0;
                    }
                }
            }
            //Almacenamos el error anterior y el tiempo
            e_ant = e;
            tiempoAnt = tiempo;
            Lectura_Sensor_Anterior = Lectura_Sensor;
        }
    }

    public void Cerrar_simulacion()
    {
        //Al salir de la simulacion volvemos a activar el prefab
        Robot_prefab.SetActive(true);
    }

    public void Datos(float dato, string path, List<string> Lista)
    {
        float numero = dato;
        string cadena = numero.ToString();
        Lista.Add(cadena);
```

```
        File.WriteAllText(path, "");
        //Write some text to the test.txt file
        StreamWriter writer = new StreamWriter(path, true);
        foreach (string s in Lista)
        {
            writer.WriteLine(s);
        }
        writer.Close();
    }
}
```