Pendulo Invertido Control PID Arbitrario

D. Carlos Sánchez López, *Member IEEE,* M.A. Villarreal-González,

***Abstract*—Para este proyecto se realizo mediante el uso de arduino el control PID arbitrario para un pendulo invertido usando 2 PIDs, uno para la posicion del carrito y otro para el balanceo de un pendulo invertido montado sobre un carro robotico de 4 ruedas.Utilizando como sensores de entrada el sensor ultrasonico hc-sr04 y para en angulo el acelerometro MPU6050.**

***Index Terms*—PID Discreto,PID arbitrario,Matlab,Arduino, Control,Carro Robotico,Ultrasonico,Sensor MPU6050**

I. Introduction

PARA la introduccion de este proyecto es nesecario conocer los principales temas de los cuales hablaremos durante este reporte, por lo cual explicaremos de manera mas detallada cada uno de sus componentes a continuacion.

- PID:Un controlador o regulador PID es un dispositivo que permite controlar un sistema en lazo cerrado para que alcance el estado de salida deseado. El controlador PID está compuesto de tres elementos que proporcionan una acción Proporcional, Integral y Derivativa. Estas tres acciones son las que dan nombre al controlador PID.
- Matlab: MATLAB es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio.
- Arduino:Arduino es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador re-programable y una serie de pines hembra. Estos permiten establecer conexiones entre el microcontrolador y los diferentes sensores y actuadores de una manera muy sencilla (principalmente con cables dupont).
- Método de Tustin: es usado habitualmente en el campo del procesamiento digital de señales y en la Teoría de control de señales discretas. Esta herramienta matemática suele usarse para transformar la representación en tiempo continuo de las señales en tiempo discreto y viceversa.

Con estos conceptos definidos de manera correcta daremos inicio a la implementacion del Controlador PID arbitrario para un pendulo invertido.

II. Simulacion

Este reporte se dividira en 2 partes, por una parte sera la simulacion utilizando matlab y simulink para realizar el controlador PID arbitrario tanto continuo para poder obtener los valores de las constantes Kp,Ki y Kd asi como discreto arbitrario mediante el criterio de Tustin y ver como este interactuara con el modelo fisico del cual hablaremos mas tarde.

Tecnológico de Monterrey Campus Puebla

Para comenzar nesecitamos conocer ciertas caracteristicas de nuestro modelo fisico para poder realizar una simulacion acertada de nuestro prototipo. Estas caracteristicas son las siguientes.

```
M=0.510;%kg
m=0.03; %kg
L=0.25;%m
g=9.81;%m/s^2
```

Fig. 1. Valores de caracteristicas del modelo fisico.

Una vez teniendo estos valores con sus respectivas unidades podremos proceder a obtener los valores continuos de nuestro PID.

A. PID continuo

Una vez teniendo las caracteristicas de nuestro sistema realizaremos el PID continuo utilizando Simulink de matlab. Gracias al siguiente esquematico podremos obtener los valores continuos de la constante Proporcional,Integral y Derivativa de nuestro PID haciendo uso de la funcion TUNE que nos ofrece el bloque PID de simulink.

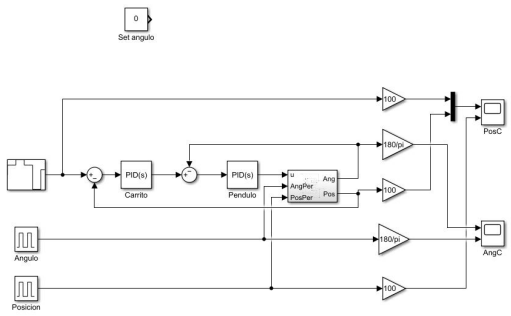

Fig. 2. Esquematico completo de Simulink para PID continuo

Como podemos observar tenemos algunos bloques los cuales fueron definidos por un usuario, por lo tanto revisaremos cada uno de estos bloques para poder realizar el proyecto.

El primer bloque que revisaremos sera el PID del carrito y el PID del pendulo,este bloque es un bloque que podemos obtener por medio de simulink que se llama PID 1dof, lo que nos permite realizar este bloque es sintonizar nuestro controlar PID con valores continuos usando la funcion de tune, lo cual nos regresara los valores correspondientes a cada una de nuestras constantes utilizadas en el PID, Proporcional,Integral y Derivativa asi como tambien el valor de nuestro Filtro

que tambien utilizaremos mas adelante para pasar los valores continuos a discretos. Este bloque nos permite hacer uso de 2 formas de programacion para hacer el tuning de nuestras constantes, una forma es ideal y otra es paralela, para esta ocasion haremos uso de la opcion de parallel para hacer el tuning de nuestro sistema PID.

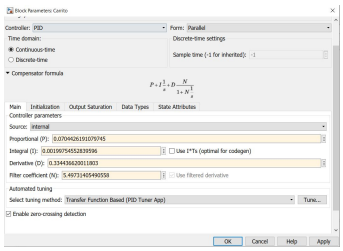

Fig. 3. Bloque PID para carrito

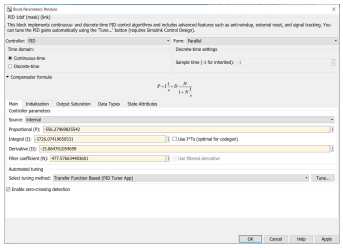

Fig. 4. Bloque PID para pendulo

Como se puede observar en las imagenes yo ya tengo valores para cada una de las variables debido a que ya se ha sintonizado este PID , pero la primera vez que entremos tendremos que realizar el siguiente procedimiento, lo primero a realizar es el PID del pendulo, para esto simularemos que el angulo del pendulo se encuentra en 0 gracias a nuestra variable SetAng que se observa en la figura [2]. realizando la conexion como se muestra en la figura 5, despues de esto procederemos a realizar el tuning de nuestro PID.

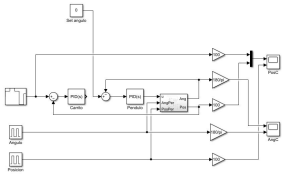

Fig. 5. Conexion en Simulink para realizar sintonizacion de PID continuo pendulo

Una vez realizada la conexion procederemos a dar click en el boton tune de nuestro bloque PID y nos debera aparecer la siguiente pantalla.

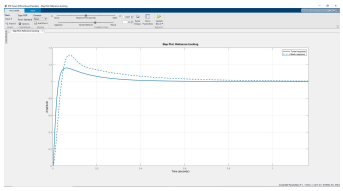

Fig. 6. Bloque tune dentro de PID

Dentro de esta pantalla procederemos a realizar el tuning de nuestro sistema a como sea nesecario dependiendo de nuestra aplicacion. Una vez realizada este tuning daremos ok y regreseramos a realizar el tuning de nuestro segundo PID para el carrito de la misma manera, solo que esta vez desconectaremos nuestro SetAng y volveremos a realizar la conexion como se muestra en la figura 2.

Ahora que tenemos nuestros valores de las constantes para nuestro PID continuo procederemos a revisar el subsistema que se ha realizado para el PID continuo.

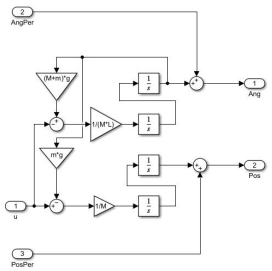

Fig. 7. Bloque subsistema PID continuo

Una vez tengamos estos valores podremos proceder a correr el programa en simulink y haciendo uso del scope podremos verificar nuestros calculos de manera simulada, como podemos observar tenemos 2 scopes uno para la posicion de nuestro carrito y otro para el angulo en el cual se encuentra nuestro pendulo. Con la simulacion bien realizada deberiamos poder obtener valores similares a los siguientes.

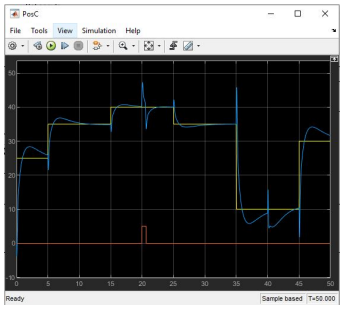

Fig. 8. Simulacion posicion

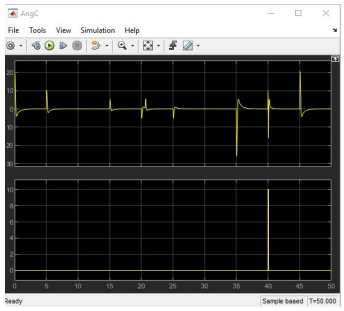

Fig. 9. Simulacion Angulo

Con esto podremos proceder a realizar la simulacion para nuestro PID discreto.

B. PID discreto

Una vez tengamos nuestros valores continuos obtenidos mediante simulink realizaremos la sintonizacion para valores discretos de nuestro PID, para este proyecto se utilizo la forma paralela, cabe recalcar que dependiendo el valor que seleccionemos en nuestro bloque PID continuo podremos elegir entre Ideal o Paralelo para realizar nuestro PID con diferencias en las ecuaciones a utilizar para este proyecto.

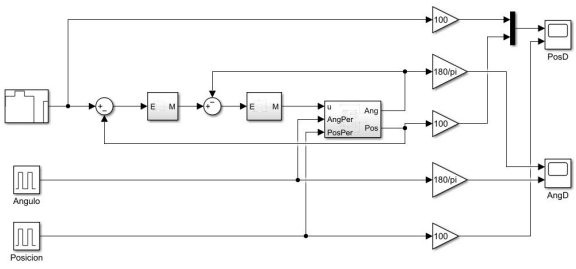

Fig. 10. Esquematico completo de Simulink para PID discreto

Se utilizara el siguiente esquematico para nuestro PID discreto en simulink, donde posteriormente hablaremos de cada subsistema con mas detalle.

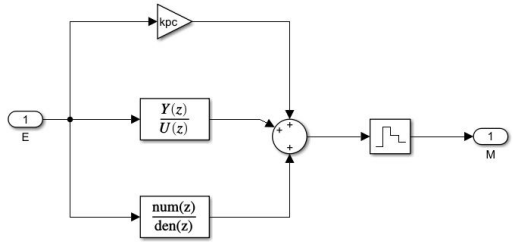

Fig. 11. Bloque de subsistema para cada uno de nuestros PID discretos

Para realizar estos subsistemas de cada uno de los PIDs se utilizaron las siguientes ecuaciones.

```
Proportional =kpc
Integral     =kic*Ts/2*(z+1)/(z-1)
Derivative   =2*Nc*kdc*(z-1)/((2+Nc*Ts)*z+(Nc*Ts-2))
```

Fig. 12. Ecuaciones a utilizar en cada uno de nuestros subsistemas para PID discreto

Una vez se haya configurado de manera correcta cada subsistema procederemos a correr el programa para verificar los resultados en nuestro scope, si todo ha salido bien podremos observar una salida similar a nuestras graficas de posicion y angulo continuo pero esta vez de manera discreta como observamos a continuacion.

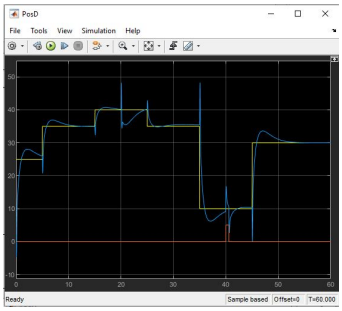

Fig. 13. Simulacion posicion Discreto

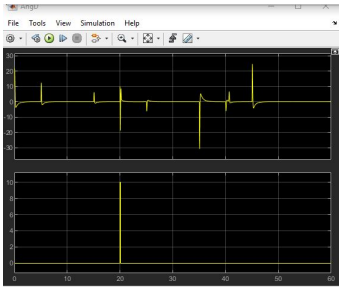

Fig. 14. Simulacion Angulo Discreto

Y con estos valores ya revisados y funcionando podremos proceder a la siguiente seccion, la cual sera realizar el programa y el simulink nesecarios para poder hacer nuestro controlodar PID arbitrario.

III. PID Arbitrario

Como hemos estado trabajando durante todo este semestre, normalmente cuando hablamos de controladores PID hablamos de controladores continuos y discretos de orden entero.Sin embargo existen otros tipos de controladores y estos se llaman Arbitrarios estos agregan dos parámetros más de sintonización: el operador derivativo fraccionario y el operador integral fraccionario, los cuales permiten un mayor rango al controlar una planta debido a que los polos de sistema pueden encontrarse en el lado derecho del semiplano imaginario, siempre y cuando cumplan ciertas condiciones.

Existen varias definiciones de la derivada en el cálculo fraccionario siendo tres de estas algunas de las más utilizadas Riemann-Liouville, Grünwald Letnikov y la de Caputo.

A continuacion se muestran las ecuaciones para cada una de las siguientes definiciones.

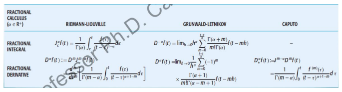

Fig. 15. Formulas para Definiciones de la derivada del calculo fraccionario

Ahora pasaremos a mostrar el codigo de matlab utilizado para este proyecto.

```
clear all;
close all;
clc;

M=0.510;%kg
m=0.03; %kg
L=0.25;%m
g=9.81;%m/s^2
ts=5;
%0.01<Ts0.15
Ts=0.01;%Sirve Fraccional


%%Valor pendulo continuo
kpp=-556.27969825542;%Ganancia Proporcional
kip=-3726.07419650531;%Ganancia Integral
kdp=-15.864761054699;%Ganancia Derivativa
Np=477.576634483601;%Filtro Pendulo
%%Valor carro continuo
kpc=0.0704426191079745;%Ganancia Proporcional
kic=0.00199754552839596;%Ganancia Integral
kdc=0.334436620011803;%Ganancia Derivativa
Nc=5.49731405490558;%Filtro Carrito

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Valores para PID arbitrario y fraccional
alphapendulo=.899;
alphacarrito=.9;
betapendulo=.999;
betacarrito=.6;
%Seleccionamos el valor de alpha para nuestro PID.
wl=0.01;
wh=100;
orden=3;%Seleccionamos el orden de PID
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Main Program
%llamamos a Derivative para encontrar el valor de Nd y Dd del pendulo y carrito
if (alphapendulo>=1)
    alphapendulo=alphapendulo-1;
    alphacarrito=alphacarrito-1;
    betapendulo=betapendulo-1;
    betacarrito=betacarrito-1;
    Ts=0.010;
end
[Ndp,Ddp,Hadp]=Derivative(betapendulo,wl,wh,orden,Np,Ts);
[Ndc,Ddc,Hadc]=Derivative(betacarrito,wl,wh,orden,Nc,Ts);
[Nip,Dip,Haip]=Integral(-alphapendulo,wl,wh,orden,Ts);
[Nic,Dic,Haic]=Integral(-alphacarrito,wl,wh,orden,Ts);
%Ndp,Ddp,Ndc,Ddc,Nip,Dip,Nic,Dic
%Hadp,Hadc,Haip,Haic
mdl = 'PIDFrac';
open_system(mdl)

blockpathCarrito = 'PIDFrac/Carrito';
blockpathPendulo = 'PIDFrac/Pendulo';

linsys1 = linearize(mdl,blockpathCarrito);
linsys2 = linearize(mdl,blockpathPendulo);
syms z;
LinearPIDCarrito=tf(minreal(linsys1));
[Num,Den] = tfdata(LinearPIDCarrito);
tfArduinotemp = taylor(poly2sym(cell2mat(Num),z)/poly2sym(cell2mat(Den),z),z,1/0,'Order',7);
tfArduinoCarrito=minreal(syms2tf(tfArduinotemp,Ts))

LinearPIDCarrito=tf(minreal(linsys2));
[Num,Den] = tfdata(LinearPIDCarrito);
tfArduinotemp = taylor(poly2sym(cell2mat(Num),z)/poly2sym(cell2mat(Den),z),z,1/0,'Order',7);
tfArduinoPendulo=minreal(syms2tf(tfArduinotemp,Ts))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Para la discretizacion de continuo a discreto utilizaremos el criterio de
%tustin (2/Ts)*((z-1)/(z+1)). Como matlab nos permite utilizar la funcio
%c2d con tustin como argumento se utilizara esta opcion para crear un
%codigo mas eficiente.

%Se crearan 2 funciones que llamaremos desde el main de matlab una sera
%para calcular la derivada de ambos PID y la otra para la integral de lo
%PID

%Valor Derivativo
function [Nd,Dd,Hadd]=Derivative(beta,wl,wh,orden,filtro,Ts)
[Ha,numd,dend]=CFI(beta,wl,wh,orden);
[numd,dend] = tfdata(Ha);
%Utilizamos los filtros
    if(filtro<200)
        dend{1}=numd{1}+dend{1}*filtro;
        numd{1}=numd{1}*filtro;
        Ha=tf(numd{1},dend{1});
    end
Hadd=c2d(Ha,Ts,'tustin'); %Hadd(Ha Discrete Derivative)
[numdmin,dendmin] = tfdata(minreal(Hadd));
Nd=numdmin{1};
Dd=dendmin{1};
end

function [Ni,Di,Hadi]=Integral(alpha,wl,wh,orden,Ts)
[Ha,numi,deni]=CFI(alpha,wl,wh,orden);
Hadi=c2d(Ha,Ts,'tustin'); %Hadd(Ha Discrete Integral)
[numimin,denimin] = tfdata(minreal(Hadi));
Ni=numimin{1};
Di=denimin{1};
end

function [Ha,num,den]= CFI(alpha,wl,wh,N)
w=logspace(log10(wl),log10(wh));
A=(j*w).^alpha;
Ha=fitfrd(frd(A,w),N);
[num,den]=ss2tf(Ha.A,Ha.B,Ha.C,Ha.D);
Ha=minreal(tf(num,den));
end

function[ans] = syms2tf(G,Ts)
[symNum,symDen] = numden(G); %Get num and den of Symbolic TF
TFnum = sym2poly(symNum);
%Convert Symbolic num to polynomial
TFden = sym2poly(symDen);
%Convert Symbolic den to polynomial
ans =tf(TFnum,TFden,Ts);
end
```

Ahora pasaremos a explicar lo que hace el codigo, la primera seccion declaramos los valores del prototipo que utilizaremos para esta prueba, en este caso M es el peso del carrito, m es el peso del contrapeso ubicado en la parte superior de nuestro carrito, L es la longitud de nuestra brazo del pendulo, g seria la gravedad de la tierra, ts es una variable que defini para que pudiera ser utilizada por un bloque para simular un desplazamiento de nuestro carrito y ver si nuestro PID lo lograria controlar de manera correcta y por ultimo tenemos Ts que sera el tiempo de muestreo que utilizaremos para este proyecto.

En la siguiente seccion tenemos los valores de nuestras ganancias proporcionales,integrales y derivativas de nuestros PID para cada uno de los PID que utilizaremos, en este caso tenemos 6 constantes debido a que se utilizaran 2 PID y 2 Filtros que tambien utilizaremos mas adelante. Todo esto es gracias a la sintonizacion realizada para los PID continuos que vimos anteriormente en la figura 2.

Ahora colocaremos los valores para cada uno de nuestros alphas y betas para cada uno de nuestros PID, cabe recalcar que para este proyecto debido a que es arbitrario nuestras alphas y betas tienen que ser menores a 1 , esto para que el sistema arbitrario pueda controlar de manera eficiente nuestro sistema y poder tener una fraccion de un PID de orden entero.

Tambien es importante mencionar que wl,wh son valores estaticos proporcionados por el profesor Carlos Sanchez Lopez y gracias a estos valores se pudo hacer el calculo para cada uno de nuestros parametros, tanto el parametro derivativo fraccional e integral fraccional los cuales veremos a continuacion.

Ahora pasaremos a explicar el main de nuestro programa, como podemos observar tenemos 2 tipos de funciones que nos entregaran cada uno de los parametros derivativos e integrales.

```
function [Ni,Di,Hadi]=Integral(alpha,wl,wh,orden,Ts)
[Ha,numi,deni]=CFI(alpha,wl,wh,orden);
Hadi=c2d(Ha,Ts,'tustin'); %Hadd(Ha Discrete Integral)
[numimin,denimin] = tfdata(minreal(Hadi));
Ni=numimin{1};
Di=denimin{1};
end
```

Para esta primera funcion de nuestro codigo de matlab se calcularan los parametros integrales fraccionales, lo primero que hacemos sera llamar a nuestra funcion de curvefitting que se encargara de entregarnos la funcion de transferencia para el parametro integral de cualquiera de nuestros PID, para esto tenemos que entregrarle a la funcion nuestra alpha, wl, wh, orden y Ts: donde alpha sera el valor fraccional al cual queremos aproximar nuestro control, wl y wh son parametros estaticos otorgados por el profesor, orden sera el orden de polinomio que queramos que nos entregue el programa, lo que esto quiere decir es que podremos aproximar nuestro valor integral a un orden mayor o menor dependiendo de nuestro requerimento y por ultimo pasamos nuestro Ts(Tiempo de muestreo). *Nota para las integrales tendremos que pasar el valor en negativo para nuestras alphas.

Despues de esto al recibir dicha funcion haremos uso del metodo de tustin para poder obtener nuestra funcion de transferencia de manera discreta y con esto poder obtener los valores de numerador y denominador que utilizaremos en el siguiente paso para poder crear una simulacion en simulink que nos permita controlar nuestro sistema con un PID arbitrario.

```
%Valor Derivativo
function [Nd,Dd,Hadd]=Derivative(beta,wl,wh,orden,filtro,Ts)
[Ha,numd,dend]=CFI(beta,wl,wh,orden);
[numd,dend] = tfdata(Ha);
%Utilizamos los filtros
    if(filtro<200)
        dend{1}=numd{1}+dend{1}*filtro;
        numd{1}=numd{1}*filtro;
        Ha=tf(numd{1},dend{1});
    end
Hadd=c2d(Ha,Ts,'tustin'); %Hadd(Ha Discrete Derivative)
[numdmin,dendmin] = tfdata(minreal(Hadd));
Nd=numdmin{1};
Dd=dendmin{1};
end
```

Ahora bien, para esta funcion el comportamiento es muy similar que el parametro integral de nuestro PID arbitrario, la unica diferencia que tenemos en nuestra funcion es el uso de filtros, lo que estos filtros hacen es corregir los errores que podamos obtener de las funciones de transferencia por parte del curve fitting approximation. De igual manera hacemos uso de la funcion c2d con metodo de tustin para pasar del dominion continuo al dominio discreto.

A. Metodo de Tustin

Transformación de Tustin

$$s = 2(z-1)/T(z-1) \tag{1}$$

B. Curve Fitting Approximation

Gracias a este metodo que nos proporciono el profesor,teniendo como base de referencia el archivo en [1] obtendremos los valores de funcion de transferencia para cada uno de nuestros parametros derivativos e integrales que calcularemos para obtener el comportamiento de un PID arbitrario.

a continuacion se muestra el codigo de matlab utilizado para realizar dicha accion.

```
function [Ha,num,den]= CFI(alpha,wl,wh,N)
w=logspace(log10(wl),log10(wh));
A=(j*w).^alpha;
Ha=fitfrd(frd(A,w),N);
[num,den]=ss2tf(Ha.A,Ha.B,Ha.C,Ha.D);
Ha=minreal(tf(num,den));
end
```

Una vez con todas estas variables y funciones de transferencia para cada uno de nuestros PID, se procede a realizar las simulacion de simulink para esto se utilizara el siguiente modelo de simulink.

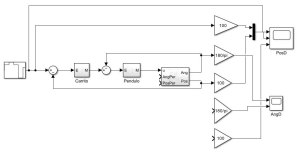

Fig. 16. PID arbitrario Simulink

Para el PID arbitrario se utilizo como base nuestro PID discreto mostrado en la figura 10 teniendo ciertas modificaciones, la primera y la mas presente es que dentro de cada

uno de nuestros subsistemas de controladores PID tendremos las funciones de transferencia obtenidas por nuestras funciones Derivative e Integral. Las cuales obtenemos apartir del metodo de curve fitting y habiendo hecho el cambio del dominio continuo a discreto.

```
Hadp =

  198.8 z^3 - 547 z^2 + 498.3 z - 150.1
  -------------------------------------
  z^3 - 0.7552 z^2 - 0.9903 z + 0.7531

Sample time: 0.01 seconds
Discrete-time transfer function.


Hadc =

  4.311 z^3 - 11.88 z^2 + 10.84 z - 3.274
  ---------------------------------------
    z^3 - 2.584 z^2 + 2.188 z - 0.6038

Sample time: 0.01 seconds
Discrete-time transfer function.


Haip =

  0.02069 z^3 - 0.05072 z^2 + 0.03944 z - 0.009411
  ------------------------------------------------
         z^3 - 2.991 z^2 + 2.983 z - 0.9914

Sample time: 0.01 seconds
Discrete-time transfer function.


Haic =

  0.02052 z^3 - 0.05023 z^2 + 0.03896 z - 0.009256
  ------------------------------------------------
         z^3 - 2.991 z^2 + 2.983 z - 0.9914
```

Fig. 17. Funciones de transferencia Integrador y Derivativo fraccional

Como podemos observar obtenemos 4 funciones de transferencia , estas funciones de transferencia pertenecen a cada uno de nuestros PID con sus respectivos parametros derivativos e integrales.

- Hadp:Funcion de transferencia derivativa pendulo
- Haip:Funcion de transferencia integral pendulo
- Hadc:Funcion de transferencia derivativa carrito
- Haic:Funcion de transferencia integral carrito

Donde cada uno de sus respectivos numerador y denominador se crearan bloques de funciones discretas los cuales tendremos anotaremos sus respectivos datos como se muestra a continuacion.

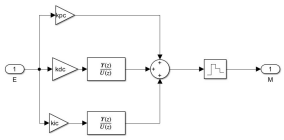

Fig. 18. PID carrito Arbitrario

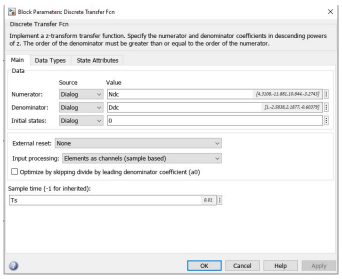

Fig. 19. Funcion Discreta Derivativa Carrito

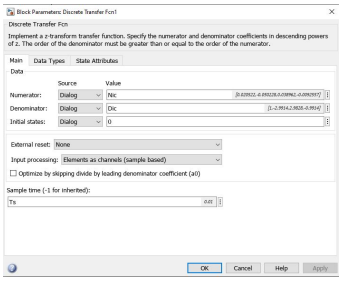

Fig. 20. Funcion Discreta Integral Carrito

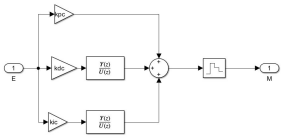

Fig. 21. PID pendulo Arbitrario

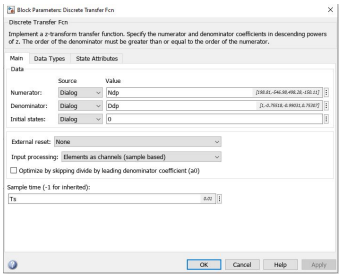

Fig. 22. Funcion Discreta Derivativa Pendulo

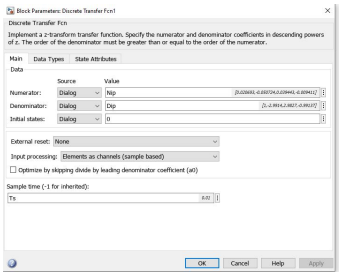

Fig. 23. Funcion Discreta Integral Pendulo

Una vez tengamos todo esto podremos proceder a realizar la simulacion de simulink para comprobar que nuestro sistema en realidad se controle como tenemos planeado.

C. Pruebas

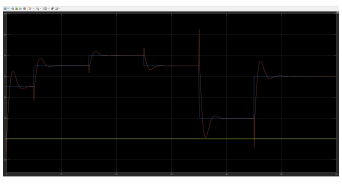

Fig. 24. Prueba de posicion con alphapendulo=.899 alphacarrito=.9 betapendulo=.999 betacarrito=.6 y orden de polinomio 3

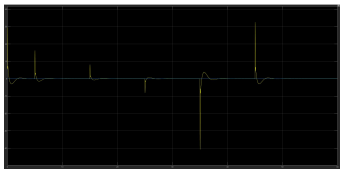

Fig. 25. Prueba de angulo con alphapendulo=.899 alphacarrito=.9 betapendulo=.999 betacarrito=.6 y orden de polinomio 3

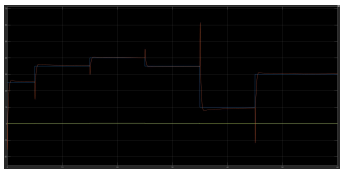

Fig. 26. Prueba de posicion con alphapendulo=.999 alphacarrito=.999 betapendulo=.999 betacarrito=.999 y orden de polinomio 6

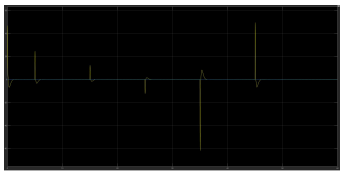

Fig. 27. Prueba de angulo con alphapendulo=.999 alphacarrito=.999 betapendulo=.999 betacarrito=.999 y orden de polinomio 6

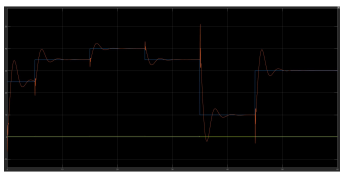

Fig. 28. Prueba de posicion con alphapendulo=.753 alphacarrito=.82 betapendulo=.9 betacarrito=.5 y orden de polinomio 6

Fig. 29. Prueba de angulo con alphapendulo=.753 alphacarrito=.82 betapendulo=.9 betacarrito=.5 y orden de polinomio 6

Como podemos observar cada una de nuestras pruebas tienen comportamientos diferentes pero a pesar de esto podemos ver que cada uno lo controla de forma distinta en esta ocasion nos quedaremos con la primera prueba ya que los valores de nuestro angulo se mantienen en un margen de +- 30 grados lo cual permite a nuestro carrito poder hacer la correcion de nuestro angulo sin llegar a un punto donde el contrapeso le pueda llegar a ganar.

IV. Prototipo Fisico

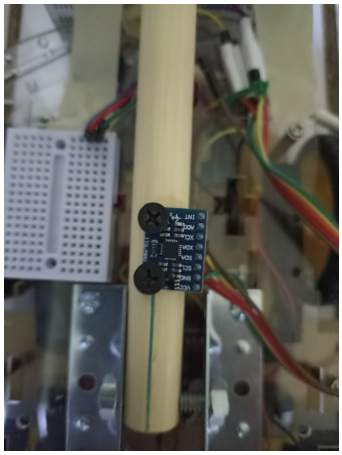

Fig. 30. Acelerometro MPU6050

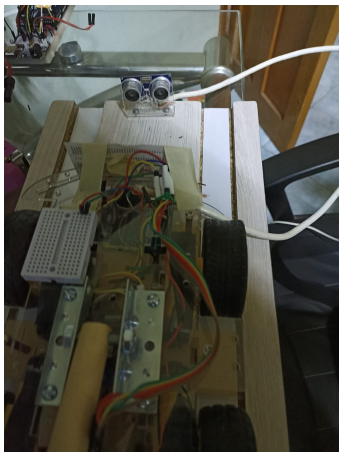

Fig. 31. Ultrasonico Hc-Sr04

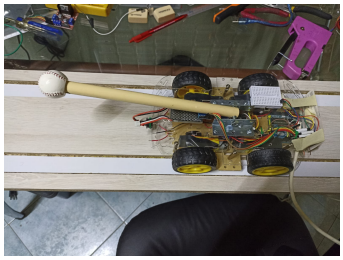

Fig. 32. Carrito con pendulo Invertido

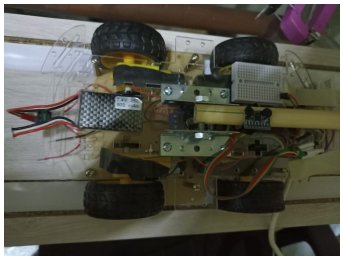

Fig. 33. Carrito con pendulo invertido

Durante la realizacion de este proyecto se tuvieron que realizar ciertos cambios debido a la complejidad de este proyecto, por lo tanto se trato de reducir los problemas lo mas posible , es por esta razon que el sensor ultrasonico fue conectado de manera alambrica en lugar de mandar los datos por Radiofrecuencia entre otros problemas encontrados, pero se encontro que este diseño funcionaba de manera adecuada por lo cual se mantuvo como prototipo final.

A. *Diagrama de Conexiones*

Debido a este siendo un prototipo fisico se implemento un diagrama de conexiones para cada uno de los sensores asi como el uso de un puente H L298N, el cual de igual manera se explicara su funcionamiento.

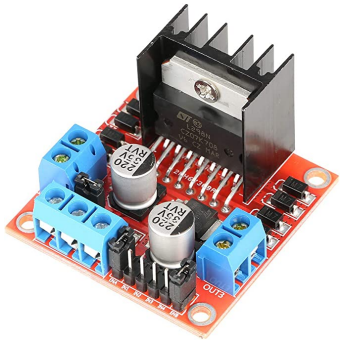

Fig. 34. Puente H L298N

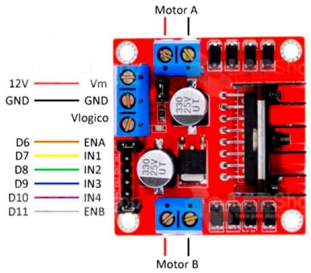

Fig. 35. Diagrama de Pines Puente H L298N

El driver puente H L298N es el modulo más utilizado para manejar motores DC de hasta 2 amperios. El chip L298N internamente posee dos puentes H completos que permiten controlar 2 motores DC o 4 en nuestro caso conectando 2 motores a la misma salida de motor A o B.

El módulo permite controlar el sentido y velocidad de giro de motores mediante señales TTL que se pueden obtener de microcontroladores y tarjetas de desarrollo como Arduino, Raspberry Pi o Launchpads de Texas Instruments. El control del sentido de giro se realiza mediante dos pines para cada motor, la velocidad de giro se puede regular haciendo uso de modulación por ancho de pulso (PWM por sus siglas en inglés).

Gracias a este modulo se pudo mover los 4 motores de nuestro carrito sin problema alguno teniendo como fuente para los motores una bateria LIPO de 7.4v a 500mAh como se muestra en la Imagen 18.

Ahora pasaremos al Diagrama electrico completo de nuestro sistema.

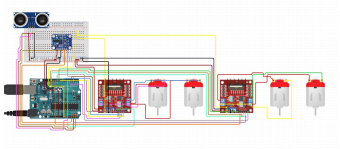

Fig. 36. Diagrama electrico de nuestro sistema

B. *Codigos*

Para revisar el codigo completo favor de seguir la siguiente liga de github donde estara el repositorio completo para este proyecto:

https://github.com/ManuelAmadeoVillarrealGonzalez /ProyectosUniversidad/tree/main/PIDArbitrario

En este repositorio se encontraran los codigos y simulaciones realizadas en Matlab documentadas lo mejor posible.

Dado que se obtuvieron las funciones de transferencia para cada uno de nuestras variables integradoras y derivativas podriamos obtener la funcion de transferencia completa de cada uno de nuestros PID y con esto podemos hacer los calculos nesecarios para obtener las variables nesecarios para la programacion de arduino como se obtuvo en el 2do parcial de la materia de control computarizado. Asi que como primera opcion te recomiendo revisar el siguiente repositorio en donde estaran los codigos de matlab y arduino para un PID discreto:

https://github.com/ManuelAmadeoVillarrealGonzalez /ProyectosUniversidad/tree/main/ControlPIDPenduloInvertido

Como conocemos cada PID nesecita tener una respuesta de salida basada en el siguiente principio donde:

$$Funciondetransferenciade PID = \frac{M(k)}{E(k)}$$

Por lo tanto lo que nesecitariamos realizar seria lo siguiente, una vez obtenidas las funciones de transferencia de cada uno de nuestros parametros que podemos observar en la figura 17, Ahora bien para poder hacer uso de estas funciones de transferencia y que los valores nos puedan acomodar a nuestras nesecidades haremos uso de la funcion linearize de matlab de Model Linearizer

```
mdl = 'PIDFrac';
open_system(mdl)

blockpathCarrito = 'PIDFrac/Carrito';
blockpathPendulo = 'PIDFrac/Pendulo';

linsys1 = linearize(mdl,blockpathCarrito);
linsys2 = linearize(mdl,blockpathPendulo);
syms z;
LinearPIDCarrito=tf(minreal(linsys1));
[Num,Den] = tfdata(LinearPIDCarrito);
tfArduinotemp = taylor(poly2sym(cell2mat(Num),z)/poly2sym(cell2mat(Den),z),z,1/0,'Order',7);
tfArduinoCarrito=minreal(syms2tf(tfArduinotemp,Ts))

LinearPIDCarrito=tf(minreal(linsys2));
[Num,Den] = tfdata(LinearPIDCarrito);
tfArduinotemp = taylor(poly2sym(cell2mat(Num),z)/poly2sym(cell2mat(Den),z),z,1/0,'Order',7);
tfArduinoPendulo=minreal(syms2tf(tfArduinotemp,Ts))
```

usando esta porcion de codigo de matlab, podremos abrir nuestro archivo de simulink y conseguir la funcion de transferencia que nos entrega cada uno de los PID, y mediante series de taylor poder aproximar los valores a polinomios de orden 7 para que de esta forma el denominador de cada una de

nuestras funciones de transferencia. Obteniendo los siguientes resultados.

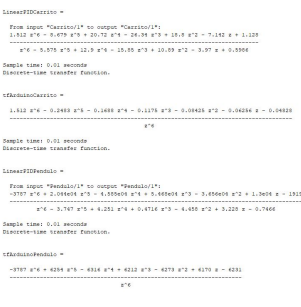

Fig. 37. Funcion de transferencia para cada PID

Ya con esto nos es mas sencillo poder obtener los valores para la programacion de arduino que van desde a0-a6 y desde b0-b6. Como lo siguiente:

Para el carrito utilizamos LinearPIDCarrito:

- a0=1.512
- a1=-8.679
- a2=20.72
- a3=-26.34
- a4=18.8
- a5=-7.142
- a6=1.128
- b0=1
- b1=-5.575
- b2=12.9
- b3=-15.85
- b4=10.89
- b5=-3.97
- b6=0.5986

Para el pendulo utilizamos LinearPIDPendulo:

- a0=-3787
- a1=2.044e4
- a2=-4.585e4
- a3=5.46e4
- a4=-3.65e4
- a5=1.3e4
- a6=-1919
- b0=1
- b1=-3.747
- b2=4.251
- b3=0.4716
- b4=-4.458
- b5=3.228
- b6=-0.7466

Con estos valores nuestros PID en la forma de M(k) quedarian de la siguiente manera:

$$a0E(k-6)-a1E(k-5)+a2E(k-4)-a3E(k-3)+a4E(k-2)-a5E(k-1)+a6E(k)=M(k-6)-b1M(k-5)+b2M(k-4)-b3M(k-3)+b4M(k-2)-b5M(k-1)+b6M(k)$$

Y si lo queremos en terminos de M(K) seria algo de esta forma:

$$M(k)=a0E(k-6)-a1E(k-5)+a2E(k-4)-a3E(k-3)+a4E(k-2)-a5E(k-1)+a6E(k)-b0M(k-6)-b1M(k-5)+b2M(k-4)-b3M(k-3)+b4M(k-2)-b5M(k-1)/b6$$

Una vez obtenidas estas funciones con M(k), podremos proceder a realizar lo que es la configuracion de arduino para calcular cada uno de los valores como se realizo para el segundo parcial de control computarizado.

V. Conclusion

Como conclusion este proyecto fue muy retador debido a su alta complejidad, pero a pesar de esto fue muy interesante de poder realizar ya que reafirmo temas como PIDs tanto discretos como continuos y gracias a esto se pudo realizar el proyecto de manera adecuada.Si bien lamentablemente no se pudo realizar el pendulo de manera fisica se obtuvieron resultados muy positivos durante la simulacion.

Gracias a este proyecto me fui de mucha utilidad ya que para su servidor la materia de control en general ha sido muy retador para mi, debido a su complejidad pero con esta materia me ayudo a comprender como trabaja el mundo y como articulos que ya tienen tiempo de haber sido expuestos podemos realizar objetivos muy interesantes, si bien es cierto que este tipo de controles arbitrarios ofrecen muchas ventajas sobre PIDs de orden entero podemos decir que la tecnologia avanza a pasos agigantados y no podemos darnos el lujo de dejar de aprender incluso de teorias y metodologias propuestas en tiempos anteriores que no fueron posibles crear debido a su complejidad en su momento.