| **INTEGRANTES** | **CÓDIGO** |
| --- | --- |
| **Neil Sebastián Castro Caicedo** | **U00131867** |
| **Hubert Armando Delgado Maestre** | **U00091396** |
| **Paula Andrea Portilla Corredor** | **U00139413** |

| horizontal-naranja | **PROGRAMA DE INGENIERÌA MECATRÓNICA**  **UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA - UNAB** |
| --- | --- |
| **DISEÑO DE CONTROLADORES POR ESPACIO DE ESTADOS PARA UN MANIPULADOR DE UN GRADO DE LIBERTAD**  ***M.Sc. Hernando González Acevedo*** |

***MODELO MATEMÁTICO***

1. Determinar el modelo matemático de un manipulador de un grado de libertad acoplado a un motorreductor DC (figura 1). La variable de entrada al modelo es la señal de modulación de ancho de pulso (PWM) que modifica el voltaje de alimentación del motor (-100%≤PWM≤100%) y las variables de salida son la posición angular de la barra y la velocidad mecánica. Linealizar el modelo, asumiendo como punto de equilibrio , y representar la dinámica del sistema en espacio de estados.

|  |  |
| --- | --- |

*Fig. 1 Manipulador de un grado de libertad*

| Ecuaciones diferenciales (Modelo no lineal) | Ecuaciones diferenciales observadas:  Como tenemos cuatro derivadas, tenemos cuatro estados, de los cuales pueden ser 𝜃𝐿, 𝜃𝑚, 𝑊𝑚, 𝑊𝐿 y 𝑖𝑎(t) pero, dado que las W y la theta guardan proporcional entre sí, la cual está definida por la relación de engranajes n.    Entonces tomamos como estados 𝑖𝑎(𝑡), 𝜃𝐿 𝑦 𝑊𝐿, quedando:  𝒙𝟏 = 𝒊𝒂  𝒙𝟐 = 𝜽𝑳  𝒙𝟑 = 𝑾𝑳  La entrada del sistema será el voltaje 𝑉𝑖𝑛(𝑡), procedemos con la derivación de los estados, obteniendo: |
| --- | --- |
| Punto de equilibrio | **Asumiendo valor de x20:** |
| Modelo lineal en espacio de estados continuo | % Linealización numérica.  A\_par = subs(A,{R,L,Kv,Ki,n,Jm,Jl,Bm,Bl,m,d,g,x1,x2,x3,u1},{Ro,Lo,Kvo,Kio,no,Jmo,Jlo,Bmo,Blo,mo,do,go,X10,X20,X30,U10});  B\_par = subs(B,{R,L,Kv,Ki,n,Jm,Jl,Bm,Bl,m,g,x1,x2,x3,u1},{Ro,Lo,Kvo,Kio,no,Jmo,Jlo,Bmo,Blo,mo,go,X10,X20,X30,U10});  C\_par = subs(C,{R,L,Kv,Ki,n,Jm,Jl,Bm,Bl,m,g,x1,x2,x3,u1},{Ro,Lo,Kvo,Kio,no,Jmo,Jlo,Bmo,Blo,mo,go,X10,X20,X30,U10});  D\_par = subs(D,{R,L,Kv,Ki,n,Jm,Jl,Bm,Bl,m,g,x1,x2,x3,u1},{Ro,Lo,Kvo,Kio,no,Jmo,Jlo,Bmo,Blo,mo,go,X10,X20,X30,U10}); |
| Autovalores de la matriz de estados en continua |  |
| Periodo de muestreo |  |
| Modelo lineal en espacio de estados discreto |  |
| Autovalores de la matriz de estados en discreta |  |

***TÉCNICA DE CONTROL: SENSIBILIDAD MIXTA***

1. Diseñar un sistema de seguimiento en discreta utilizando la técnica de sensibilidad mixta. Evaluar la respuesta transitoria en lazo cerrado con el modelo no lineal y lineal, dado un setpoint variable Completar la siguiente tabla.

| Función de transferencia y |  |
| --- | --- |
| Función de transferencia y |  |
| Función de transferencia y |  |
| Controlador |  |
| Diagrama de Hankel |  |
| Controlador reducido |  |
| Matriz antiwindup |  |
| Código del controlador en el bloque de Matlab function | function [ut, Con\_out] = fcn(Sp,Pv,Con\_in)  xk = Con\_in(1:4); % Estados controlador  Du = Con\_in(5); % Actuadores  % Diseño mixsyn  Gc = [0.999707105424673,0.00151771236953691,0.000689904726261718,-0.000339220737457422;-0.00151772398850250,0.756753569411887,-0.321971210474965,0.0952642884793470;0.000689878641928251,0.321971196611839,0.627937488408591,0.244883619731361;-0.000339193385452838,-0.0952642611575348,0.244883581403790,0.768717841221103];  Hc = [13.3897425159846;34.4454472814104;-15.9073661414958;7.78030172576614];  Cc = [0.100423050184375,-0.258340858953882,-0.119305245563825,0.0583522705240672];  Dc = [34.7870470020576];  Ka = zeros(1,1);  Sp = [Sp];  Pv = [Pv];  error = Sp-Pv;  xk\_1 = Gc\*xk+Hc\*error - Ka\*Du;  u = Cc\*xk+Dc\*error;  u1s = u;  if u>=100  u1s = 100;  end  if u<=-100  u1s = -100;  end  Du = u - u1s;  ut = u1s;  Con\_out = [xk\_1; Du]; |
| Respuesta transitoria de la variable del modelo lineal y del modelo no lineal en lazo cerrado, dada una señal de referencia variable |  |
| Respuesta transitoria de la acción de control del modelo lineal y del modelo no lineal en lazo cerrado |  |

* Programar en el sistema embebido el controlador y evaluar el comportamiento dinámico del regulador para diferentes puntos de operación (incluir valores de referencia que saturen el actuador). Comparar la respuesta transitoria que se obtienen a partir del modelo dinámico con los datos experimentales, dado un setpoint variable.

| Periodo de muestreo (Histograma del delta del vector de tiempo) |  |
| --- | --- |
| Setpoint – Pv Experimental - Pv Simulada |  |
| Señal Control Experimental – Señal de Control Simulada |  |
| Análisis de los resultados | Como se observa en la gráfica de PV-EXP-SIM la simulación del modelo no lineal genera oscilaciones para SP grandes sin llegar a inestabilizar, en este caso la máxima variación de SP se realizó entre [70,-70]º. Se hicieron variaciones entre SP lejanos y cercanos, ya sean positivos o negativos. La acción de control se mantiene constante lo que demuestra que cuando la variable experimental se estabiliza dentro del rango de 6.5% de error la acción de control se mantiene, a excepción del segundo 60 donde se observa que el péndulo supera el SP y debe hacer las respectivas correcciones en la acción de control. |

**Nota:** Para compensar la zona muerta del motorreductor, incluir el siguiente condicional, donde la variable Emin puede ser una constante o un porcentaje con relación al setpoint.

| Error = Sp – Pv  If abs(Error) > Emin  Calcular la señal de control u(k)  Else  Mantener la señal de control u(k-1)  end |
| --- |

***TÉCNICA DE CONTROL: LOOPSHAPING***

1. Diseñar un sistema de seguimiento en discreta utilizando la técnica de loopshaping. Evaluar la respuesta transitoria en lazo cerrado con el modelo no lineal y lineal, dado un setpoint variable. Completar la siguiente tabla.

| Función de transferencia y |  |
| --- | --- |
| Controlador 1DOF |  |
| Diagrama de Hankel |  |
| Controlador reducido 1DOF |  |
| Matriz antiwindup |  |
| Código del controlador en el bloque de Matlab function | function [ut, Con\_out] = fcn(Sp,Pv,Con\_in)  xk = Con\_in(1:3); % Estados controlador  Du = Con\_in(4); % Actuadores  % Diseño loopsyn  Gc = [0.385387352026787 -0.351868912104334 0  0.351868912104333 0.104011862141422 0  0 0 1];  Hc = [73.4809041059298  -11.4856211615732  -293.409177798269];  Cc = [-0.551106780794473 -0.0861421587117992 -0.00390881653780578];  Dc = [64.540387713901720];  Ka = zeros(1,1);  Sp = [Sp];  Pv = [Pv];  error = Sp-Pv;  xk\_1 = Gc\*xk+Hc\*error - Ka\*Du;  u = Cc\*xk+Dc\*error;  u1s = u;  if u>=100  u1s = 100;  end  if u<=-100  u1s = -100;  end  Du = u - u1s;  ut = u1s;  Con\_out = [xk\_1; Du]; |
| Respuesta transitoria de la variable del modelo lineal y del modelo no lineal en lazo cerrado, dada una señal de referencia variable |  |
| Respuesta transitoria de la acción de control del modelo lineal y del modelo no lineal en lazo cerrado |  |
| Controlador 2DOF |  |
| Diagrama de Hankel |  |
| Controlador reducido 2DOF |  |
| Matriz antiwindup |  |
| Código del controlador en el bloque de Matlab function | function [ut, Con\_out] = fcn(Sp,Pv,Con\_in)  xk = Con\_in(1:4); % Estados controlador  Du = Con\_in(5); % Actuadores  % Diseño loopsyn 2DOF  Gc = [0.387740952147887 -0.330773217755832 0.0933023055793410 0  0.302858443809632 0.398806974639918 0.531579394339184 0  -0.122147507991330 0.367784789347473 0.241457501845062 0  0 0 0 1];  Hc = [ 7.49504652749455 -73.3356874631247  -9.61892940173992 7.89345204191609  -9.64896526640991 -6.79243127064359  -293.409177798267 293.409177798269];  Cc = [-0.553683399270205 -0.0742928123337326 0.0396733540563608 -0.00390881653780578];  Dc = [32.7641447476837 -64.5403877139017];  Ka = zeros(1,1);  xk\_1 = Gc\*xk + Hc\*[Sp; Pv] - Ka\*Du;  u = Cc\*xk + Dc\*[Sp; Pv];  u1s = u;  if u>=100  u1s = 100;  end  if u<=-100  u1s = -100;  end  Du = u - u1s;  ut = u1s;  Con\_out = [xk\_1; Du]; |
| Respuesta transitoria de la variable del modelo lineal y del modelo no lineal en lazo cerrado, dada una señal de referencia variable |  |
| Respuesta transitoria de la acción de control del modelo lineal y del modelo no lineal en lazo cerrado |  |

1. De acuerdo con los resultados de la simulación, justificar la selección de la estrategia de control, entre 1 DOF y 2DOF, y programarla en el sistema embebido. Evaluar el comportamiento dinámico del regulador para diferentes puntos de operación (incluir valores de referencia que saturen el actuador). Comparar la respuesta transitoria que se obtienen a partir del modelo dinámico con los datos experimentales, dado un setpoint variable.

| Periodo de muestreo (Histograma del delta del vector de tiempo) |  |
| --- | --- |
| Justificación de la selección de la estrategia de control de 1DOF o 2 DOF | Se eligió la estrategia de 1dof por dos motivos principales, primero la respuesta transitoria de 2DOF durante su periodo de levantamiento genera un leve sobrepeso el cual puede ser crítico teniendo en cuenta que el motor cuenta con backlash y zona muerta. Segundo, la acción de control de 1DOF empieza con acciones de control fuertes lo cual puede minimizar los efectos de la zona muerta pero no lo garantiza. |
| Setpoint – Pv Experimental - Pv Simulada |  |
| Señal Control Experimental – Señal de Control Simulada |  |
| Análisis de los resultados | Gracias al compensador se observa una respuesta transitoria parecida a la técnica de MIXSING (se puede verificar la veracidad de los datos en el EXCEL de datos adjunto), la zona crítica sigue estando en el segundo 60 cuando intenta pasar de -70º a -30º. El compensador genera esta similitud en los datos experimentales dado que este actúa rápidamente cuando percibe que el motor se ha detenido. Se intentaron SP mayores a +/-70º pero no se logró una estabilidad con repetibilidad. El controlador garantiza estabilidad de PV y U para valores de SP lejanos y cercanos entre sí. De las tres técnicas esta genera mejores resultados simulados, permitiendo llegar un poco más alto de forma simulada pero no se garantiza experimental. |

**Nota:** Para compensar la zona muerta del motorreductor, incluir el siguiente condicional, donde la variable Emin puede ser una constante o un porcentaje con relación al setpoint.

| Error = Sp – Pv  If abs(Error) > Emin  Calcular la señal de control u(k)  Else  Mantener la señal de control u(k-1)  end |
| --- |

***TÉCNICA DE CONTROL: METODOLOGÍA GLOVER-MCFARLANE***

1. Diseñar un sistema de seguimiento en discreta utilizando la metodología de Glover-Mcfarlane. Evaluar la respuesta transitoria en lazo cerrado con el modelo no lineal y lineal, dado un setpoint variable Completar la siguiente tabla.

| Función de transferencia y |  |
| --- | --- |
| Controlador |  |
| Diagrama de Hankel |  |
| Controlador reducido |  |
| Matriz antiwindup |  |
| Código del controlador en el bloque de Matlab function | function [ut, Con\_out] = fcn(Sp,Pv,Con\_in)  xk = Con\_in(1:3); % Estados controlador  Du = Con\_in(4); % Actuadores  % Diseño nfcsyn  Gc = [0.853862925588747 0.187830959530800 0  -0.187830959530800 0.954205726727783 0  0 0 1];  Hc = [-8.29512388959399  -1.99815784321956  -101.934103302858];  Cc = [-0.0622134291719549 0.0149861838241466 -0.00900000000000000];  Dc = [0];  Ka = zeros(1,1);  Sp = [Sp];  Pv = [Pv];  error = Sp-Pv;  xk\_1 = Gc\*xk+Hc\*error - Ka\*Du;  u = Cc\*xk+Dc\*error;  u1s = u;  if u>=100  u1s = 100;  end  if u<=-100  u1s = -100;  end  Du = u - u1s;  ut = u1s;  Con\_out = [xk\_1; Du]; |
| Respuesta transitoria de la variable del modelo lineal y del modelo no lineal en lazo cerrado, dada una señal de referencia variable |  |
| Respuesta transitoria de la acción de control del modelo lineal y del modelo no lineal en lazo cerrado |  |

1. Programar en el sistema embebido el controlador y evaluar el comportamiento dinámico del regulador para diferentes puntos de operación (incluir valores de referencia que saturen el actuador). Comparar la respuesta transitoria que se obtiene a partir del modelo dinámico con los datos experimentales, dado un setpoint variable.

| Periodo de muestreo (Histograma del delta del vector de tiempo) |  |
| --- | --- |
| Setpoint – Pv Experimental - Pv Simulada |  |
| Señal Control Experimental – Señal de Control Simulada |  |
| Análisis de los resultados | Para el caso de la técnica NFCSYN se observan dos puntos donde la señal no logra estabilizarse, cuando intenta pasar de -60º a 40º y nuevamente de -70º a -30º. Se hicieron varias tomas de datos y en la mayoría surgía el mismo problema para las mismas zonas. En conclusiones ahondaremos más sobre las posibles causas de esto. Nuevamente, no se logró garantizar estabilidad para SP mayores a +/- 70º, el modelo lineal genera pequeñas oscilaciones para valores de SP grandes pero sin llegar a desestabilizar. |

**Nota:** Para compensar la zona muerta del motorreductor, incluir el siguiente condicional, donde la variable Emin puede ser una constante o un porcentaje con relación al setpoint.

| Error = Sp – Pv  If abs(Error) > Emin  Calcular la señal de control u(k)  Else  Mantener la señal de control u(k-1)  end |
| --- |

***CONCLUSIONES.***

1. Establecer las conclusiones más relevantes del desarrollo del laboratorio.

| * El compensador ayuda a generar resultados parecidos entre sí en las distintas técnicas, esto surge por la forma en que está planteado, se diseñó como un compensador correctivo, que entra a funcionar cuando entra en zona muerta o backlash, esto se realiza haciendo una medición de la velocidad para saber en qué momento se ha detenido el péndulo por estos fenómenos. La medición de velocidad no se hace directamente para evitar falsas banderas, entonces se diseñó de manera indirecta observando los valores del error comparándolos con 3 datos anteriores, esto quiere decir que el compensador entra a funcionar hasta que no hayan pasado 3 periodos de muestreo, aprox. 90ms.   Ahora bien, el compensador lo que ataca es la acción de control para acelerarla, de esta manera ayuda al controlador a calcularla más rápido para salir de la zona muerta lo más rápido posible. Estos se realiza sumando el 19% de la acción de control al valor parcial.    Como se observa en los resultados experimentales y como se mencionó al principio, el compensador es correctivo, solo actúa cuando la zona muerta o backlash aparece, pero ayuda a que el motor calcule mas rapido la acción de control y salga lo más pronto posible de la zona muerta.    ROJO: PV VERDE: U AZUL: SP  Por esta razón se genera similitud en los resultados experimentales de las tres distintas técnicas, dado que el compensador va corrigiendo a medida que aparecen estos fenómenos. Se logra observar el momento en que el fenómeno afecta y lo rápido que el compensador saca al motor de este mismo.  El compensador no entra en acción la mayoría de veces cuando el cambio de SP está alejado el uno del otro, en este caso las acciones de control son más fuertes y esto permite llegar al punto de consigna sin verse afectado por estos fenómenos. Por esto no se observan las correcciones leves en SP y U.  ROJO: PV VERDE: U AZUL: SP  Ahora bien, el compensador garantiza muy buenos resultados para cambios de SP cercanos entre sí, en estos casos los fenómenos se agudizan y el compensador al ser correctivo logra sacar al péndulo adelante.  AZUL: SP VERDE: EXPERIMENTAL CELESTE: SIMULADO  Por último, ¿por qué se generó el mismo efecto en este punto para las tres técnicas?.:   AZUL: SP VERDE: EXPERIMENTAL CELESTE: SIMULADO  Se hicieron distintas tomas de datos con las tres técnicas y simpre se generaba se error en ese punto, después de analizar los resultados se llegó a la conclusión que el motor tiene algunas zonas donde el backslash es más fuerte que otras, por entre hay algunas combinaciones de SP en las cuales se nota más o menos este efecto. Por esta razón para todas las tomas de datos que se realizaron se genero este error en el mismo punto. |
| --- |

***VIDEO CON LOS RESULTADOS DE LA VALIDACIÓN EXPERIMENTAL***

1. Realizar un video en el cual se valide el funcionamiento del sistema de control. En el video se debe observar de forma simultánea el movimiento de la barra, para diferentes valores de referencia, como la representación gráfica de la respuesta transitoria del SP, PV y MV, tal como se visualiza en la interfaz del sistema embebido.

| Sensibilidad mixta | <https://youtu.be/iU4VlG4Emw4> |
| --- | --- |
| Loopshaping | <https://youtu.be/kYS8HcZI-18> |
| Metodología Glover-Mcfarlane | <https://youtu.be/vjkzbvqT3oY> |

***ANEXOS.***

| Código fuente del sistema embebido – Sensibilidad mixta | **#include "MeanFilterLib.h"**  **#include <BasicLinearAlgebra.h>**  **MeanFilter<float> pwmFil(3);**  **MeanFilter<float> pFil(3);**  **MeanFilter<float> rpmFil(3);**  **const byte Encoder\_C1 = 2; // Cable amarillo pin 3 digital**  **const byte Encoder\_C2 = 3; // Cable verde al pin 4 digital**  **byte Encoder\_C1Last;**  **int paso = 0;**  **boolean direccion, flagPWM, autoincrement;**  **int entrada3 = 13;**  **int entrada4 = 12;**  **int enableB = 11;**  **int maxPWM = 130; // VCC 7V**  **int userE, Sp;**  **const int tMuestro = 30;**  **volatile int n = 0;**  **volatile byte ant = 0;**  **volatile byte act = 0;**  **double P = 0;**  **double R = 3840;**  **float arduinoPWM;**  **float rpm = 0;**  **unsigned long timeold = 0;**  **unsigned long startTime = 0;**  **unsigned long elapsedTime = 0;**  **float ut, us = 0;**  **float u1max = 100;**  **float u1min = -100;**  **float en\_1, en\_2, en\_3 = 0;**  **// MIXSYN**  **BLA::Matrix<4, 4> Gc = {0.999707105424673,0.00151771236953691,0.000689904726261718,-0.000339220737457422,**  **-0.00151772398850250,0.756753569411887,-0.321971210474965,0.0952642884793470,**  **0.000689878641928251,0.321971196611839,0.627937488408591,0.244883619731361,**  **-0.000339193385452838,-0.0952642611575348,0.244883581403790,0.768717841221103};**  **BLA::Matrix<4, 1> Hc = {13.3897425159846,**  **34.4454472814104,**  **-15.9073661414958,**  **7.78030172576614};**  **BLA::Matrix<1, 4> Cc = {0.100423050184375,-0.258340858953882,-0.119305245563825,0.0583522705240672};**  **BLA::Matrix<1, 1> Dc = {34.7870470020576};**  **BLA::Matrix<1, 1> Ka = {0};**  **BLA::Matrix<1, 1> Du = {0};**  **BLA::Matrix<1, 1> u = {0};**  **BLA::Matrix<4, 1> xk = {0, 0, 0, 0};**  **BLA::Matrix<4, 1> xk\_1 = {0, 0, 0, 0};**  **void reiniciarVAR() {**  **autoincrement = false;**  **startTime = 0; // Reiniciar el tiempo de inicio si autoincrement es false**  **elapsedTime = 0;**  **timeold = 0;**  **rpm = 0;**  **P = 0;**  **Sp = 0;**  **flagPWM = false;**  **analogWrite(enableB, 0); // Equivalente a 50% del máximo (128 de 255)**  **}**  **void calculapulso() {**  **int Lstate = digitalRead(Encoder\_C1);**  **if ((Encoder\_C1Last == LOW) && Lstate == HIGH)**  **{**  **int val = digitalRead(Encoder\_C2);**  **if (val == LOW && direccion)**  **{**  **direccion = false; //Reverse**  **}**  **else if (val == HIGH && !direccion)**  **{**  **direccion = true; //Forward**  **}**  **}**  **Encoder\_C1Last = Lstate;**  **if (!direccion) paso++;**  **else paso--;**  **ant = act;**  **if (digitalRead(Encoder\_C1)) bitSet(act, 1); else bitClear(act, 1);**  **if (digitalRead(Encoder\_C2)) bitSet(act, 0); else bitClear(act, 0);**  **if (ant == 2 && act == 0) n++;**  **if (ant == 0 && act == 1) n++;**  **if (ant == 3 && act == 2) n++;**  **if (ant == 1 && act == 3) n++;**  **if (ant == 1 && act == 0) n--;**  **if (ant == 3 && act == 1) n--;**  **if (ant == 0 && act == 2) n--;**  **if (ant == 2 && act == 3) n--;**  **}**  **void direMotor() {**  **// Cambia la dirección del motor y establece la velocidad**  **if (flagPWM) {**  **digitalWrite(entrada3, LOW);**  **digitalWrite(entrada4, HIGH);**  **} else {**  **// IZQUIERDA POSITIVO**  **digitalWrite(entrada3, HIGH);**  **digitalWrite(entrada4, LOW);**  **}**  **analogWrite(enableB, map(abs(ut), 0, 100, 0, maxPWM));**  **}**  **void setup()**  **{**  **Serial.begin(9600);**  **pinMode(entrada3, OUTPUT);**  **pinMode(entrada4, OUTPUT);**  **pinMode(enableB, OUTPUT);**  **pinMode(Encoder\_C2, INPUT);**  **flagPWM = false;**  **autoincrement = false;**  **attachInterrupt(digitalPinToInterrupt(Encoder\_C1), calculapulso, CHANGE);**  **attachInterrupt(digitalPinToInterrupt(Encoder\_C2), calculapulso, CHANGE);**  **Serial.println("LISTO");**  **}**  **void loop()**  **{**  **if (Serial.available()) { // Verificar si hay datos disponibles en el puerto serial**  **userE = Serial.parseInt(); // Leer el valor entero enviado por el puerto serial**  **// Verificar si el valor recibido está dentro del rango válido (0-100)**  **if (userE != 0 && userE != 404 ) {**  **flagPWM = (userE < 0) ? true : false;**  **autoincrement = true;**  **//Sp = userE; // Para enviar por teclado**  **}**  **if (userE == 404) {**  **reiniciarVAR();**  **}**  **}**  **if (autoincrement == true) {**  **if (startTime == 0) {**  **startTime = millis(); // Registrar el tiempo de inicio**  **}**  **elapsedTime = millis() - startTime; // Calcular el tiempo transcurrido**  **if (elapsedTime > 3000) {**  **if (millis() - timeold >= tMuestro) {**  **escalonPWM();**  **//rpm = (paso \* (0.100 / (millis() - timeold)) \* (60000 / 360)) \* -1 ;**  **//rpm = rpmFil.AddValue(rpm);**  **P = (n \* 360.0) / R;**  **P = pFil.AddValue(P);**  **controlador();**  **direMotor();**  **impresionSerial();**  **timeold = millis();**  **paso = 0;**  **}**  **if (elapsedTime >= 113000 ) {**  **reiniciarVAR();**  **}**  **}**  **}**  **}**  **void escalonPWM() {**  **if (elapsedTime > 3001) Sp = 50; // PWMX60x80**  **if (elapsedTime > 13000) Sp = -60; // PWMX60x80**  **if (elapsedTime > 23000) Sp = 40; // PWMX60x80**  **if (elapsedTime > 33000) Sp = 50; // PWMX60x80**  **if (elapsedTime > 43000) Sp = 70; // PWMX60x80**  **if (elapsedTime > 53000) Sp = -70; // PWMX60x80**  **if (elapsedTime > 63000) Sp = -30; // PWMX60x80**  **if (elapsedTime > 73000) Sp = 30; // PWMX60x80**  **if (elapsedTime > 83000) Sp = -40; // PWMX60x80**  **if (elapsedTime > 93000) Sp = -50; // PWMX60x80**  **if (elapsedTime > 103000) Sp = -60; // PWMX60x80**  **}**  **void impresionSerial() {**  **arduinoPWM = pwmFil.AddValue(arduinoPWM);**  **//Serial.print(Sp ); Serial.print(", "); Serial.print(P ); Serial.print(", "); Serial.print(arduinoPWM); Serial.print("\n");**  **Serial.print(elapsedTime ); Serial.print(", "); Serial.print(Sp ); Serial.print(", "); Serial.print(P ); Serial.print(", "); Serial.print(arduinoPWM); Serial.print("\n");**  **;**  **}**  **// CONTROLADORES**  **void controlador() {**  **BLA::Matrix<1, 1> en = {(Sp \* 0.0174533) - (P \* 0.0174533)};**  **double enn = (Sp \* 0.0174533) - (P \* 0.0174533);**  **if (abs(en(0, 0)) > abs(0.072 \* Sp \* 0.0174533)) {**  **xk\_1 = Gc\*xk+Hc\*en;**  **u = Cc\*xk+Dc\*en;**  **us = u(0, 0);**  **if (u(0, 0) >= u1max) {**  **us = u1max;**  **}**  **if (u(0, 0) <= u1min) {**  **us = u1min;**  **}**  **ut = us;**  **//a**  **Du = u(0, 0) - ut;**  **xk = xk\_1;**  **// mayor**  **if (en\_3 == enn) ut = ut + (ut \* 0.19);**  **en\_1 = enn;**  **float en\_2 = en\_1;**  **en\_3 = en\_2;**    **arduinoPWM = ut;**  **/\***  **if (abs(rpm) >= 0.7 && abs(rpm) < 3 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.6;**  **if (abs(rpm) >= 3 && abs(rpm) < 7 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.4;**  **if (abs(rpm) >= 7 && abs(rpm) < 10 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.2;**  **if (abs(rpm) >= 10 && abs(rpm) < 15 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2;**  **\*/**  **flagPWM = (ut < 0) ? true : false;**  **}**  **}** |
| --- | --- |
| Código fuente del sistema embebido - Loopshaping | **#include "MeanFilterLib.h"**  **#include <BasicLinearAlgebra.h>**  **MeanFilter<float> pwmFil(3);**  **MeanFilter<float> pFil(3);**  **MeanFilter<float> rpmFil(3);**  **const byte Encoder\_C1 = 2; // Cable amarillo pin 3 digital**  **const byte Encoder\_C2 = 3; // Cable verde al pin 4 digital**  **byte Encoder\_C1Last;**  **int paso = 0;**  **boolean direccion, flagPWM, autoincrement;**  **int entrada3 = 13;**  **int entrada4 = 12;**  **int enableB = 11;**  **int maxPWM = 130; // VCC 7V**  **int userE, Sp;**  **const int tMuestro = 30;**  **volatile int n = 0;**  **volatile byte ant = 0;**  **volatile byte act = 0;**  **double P = 0;**  **double R = 3840;**  **float arduinoPWM;**  **float rpm = 0;**  **unsigned long timeold = 0;**  **unsigned long startTime = 0;**  **unsigned long elapsedTime = 0;**  **float ut, us = 0;**  **float u1max = 100;**  **float u1min = -100;**  **float en\_1, en\_2, en\_3 = 0;**  **// LOOPSING 1DOF**  **BLA::Matrix<3, 3> Gc = {0.3963, -0.3751, 0,**  **0.3751, 0.0019, 0,**  **0, 0, 1.0000};**  **BLA::Matrix<3, 1> Hc = {73.1508,**  **-13.2603,**  **-293.4822};**  **BLA::Matrix<1, 3> Cc = {-0.5486, -0.0995, -0.0039};**  **BLA::Matrix<1, 1> Dc = {64.6527};**  **BLA::Matrix<1, 1> Ka = {0};**  **BLA::Matrix<1, 1> Du = {0};**  **BLA::Matrix<1, 1> u = {0};**  **BLA::Matrix<3, 1> xk = {0, 0, 0};**  **BLA::Matrix<3, 1> xk\_1 = {0, 0, 0};**  **void reiniciarVAR() {**  **autoincrement = false;**  **startTime = 0; // Reiniciar el tiempo de inicio si autoincrement es false**  **elapsedTime = 0;**  **timeold = 0;**  **rpm = 0;**  **P = 0;**  **Sp = 0;**  **flagPWM = false;**  **analogWrite(enableB, 0); // Equivalente a 50% del máximo (128 de 255)**  **}**  **void calculapulso() {**  **int Lstate = digitalRead(Encoder\_C1);**  **if ((Encoder\_C1Last == LOW) && Lstate == HIGH)**  **{**  **int val = digitalRead(Encoder\_C2);**  **if (val == LOW && direccion)**  **{**  **direccion = false; //Reverse**  **}**  **else if (val == HIGH && !direccion)**  **{**  **direccion = true; //Forward**  **}**  **}**  **Encoder\_C1Last = Lstate;**  **if (!direccion) paso++;**  **else paso--;**  **ant = act;**  **if (digitalRead(Encoder\_C1)) bitSet(act, 1); else bitClear(act, 1);**  **if (digitalRead(Encoder\_C2)) bitSet(act, 0); else bitClear(act, 0);**  **if (ant == 2 && act == 0) n++;**  **if (ant == 0 && act == 1) n++;**  **if (ant == 3 && act == 2) n++;**  **if (ant == 1 && act == 3) n++;**  **if (ant == 1 && act == 0) n--;**  **if (ant == 3 && act == 1) n--;**  **if (ant == 0 && act == 2) n--;**  **if (ant == 2 && act == 3) n--;**  **}**  **void direMotor() {**  **// Cambia la dirección del motor y establece la velocidad**  **if (flagPWM) {**  **digitalWrite(entrada3, LOW);**  **digitalWrite(entrada4, HIGH);**  **} else {**  **// IZQUIERDA POSITIVO**  **digitalWrite(entrada3, HIGH);**  **digitalWrite(entrada4, LOW);**  **}**  **analogWrite(enableB, map(abs(ut), 0, 100, 0, maxPWM));**  **}**  **void setup()**  **{**  **Serial.begin(9600);**  **pinMode(entrada3, OUTPUT);**  **pinMode(entrada4, OUTPUT);**  **pinMode(enableB, OUTPUT);**  **pinMode(Encoder\_C2, INPUT);**  **flagPWM = false;**  **autoincrement = false;**  **attachInterrupt(digitalPinToInterrupt(Encoder\_C1), calculapulso, CHANGE);**  **attachInterrupt(digitalPinToInterrupt(Encoder\_C2), calculapulso, CHANGE);**  **Serial.println("LISTO");**  **}**  **void loop()**  **{**  **if (Serial.available()) { // Verificar si hay datos disponibles en el puerto serial**  **userE = Serial.parseInt(); // Leer el valor entero enviado por el puerto serial**  **// Verificar si el valor recibido está dentro del rango válido (0-100)**  **if (userE != 0 && userE != 404 ) {**  **flagPWM = (userE < 0) ? true : false;**  **autoincrement = true;**  **//Sp = userE; // Para enviar por teclado**  **}**  **if (userE == 404) {**  **reiniciarVAR();**  **}**  **}**  **if (autoincrement == true) {**  **if (startTime == 0) {**  **startTime = millis(); // Registrar el tiempo de inicio**  **}**  **elapsedTime = millis() - startTime; // Calcular el tiempo transcurrido**  **if (elapsedTime > 3000) {**  **if (millis() - timeold >= tMuestro) {**  **escalonPWM();**  **//rpm = (paso \* (0.100 / (millis() - timeold)) \* (60000 / 360)) \* -1 ;**  **//rpm = rpmFil.AddValue(rpm);**  **P = (n \* 360.0) / R;**  **P = pFil.AddValue(P);**  **controlador();**  **direMotor();**  **impresionSerial();**  **timeold = millis();**  **paso = 0;**  **}**  **if (elapsedTime >= 113000 ) {**  **reiniciarVAR();**  **}**  **}**  **}**  **}**  **void escalonPWM() {**  **if (elapsedTime > 3001) Sp = 50; // PWMX60x80**  **if (elapsedTime > 13000) Sp = -60; // PWMX60x80**  **if (elapsedTime > 23000) Sp = 40; // PWMX60x80**  **if (elapsedTime > 33000) Sp = 50; // PWMX60x80**  **if (elapsedTime > 43000) Sp = 70; // PWMX60x80**  **if (elapsedTime > 53000) Sp = -70; // PWMX60x80**  **if (elapsedTime > 63000) Sp = -30; // PWMX60x80**  **if (elapsedTime > 73000) Sp = 30; // PWMX60x80**  **if (elapsedTime > 83000) Sp = -40; // PWMX60x80**  **if (elapsedTime > 93000) Sp = -50; // PWMX60x80**  **if (elapsedTime > 103000) Sp = -60; // PWMX60x80**  **}**  **void impresionSerial() {**  **arduinoPWM = pwmFil.AddValue(arduinoPWM);**  **//Serial.print(Sp ); Serial.print(", "); Serial.print(P ); Serial.print(", "); Serial.print(arduinoPWM); Serial.print("\n");**  **Serial.print(elapsedTime ); Serial.print(", "); Serial.print(Sp ); Serial.print(", "); Serial.print(P ); Serial.print(", "); Serial.print(arduinoPWM); Serial.print("\n");**  **;**  **}**  **// CONTROLADORES**  **void controlador() {**  **BLA::Matrix<1, 1> en = {(Sp \* 0.0174533) - (P \* 0.0174533)};**  **double enn = (Sp \* 0.0174533) - (P \* 0.0174533);**  **if (abs(en(0, 0)) > abs(0.072 \* Sp \* 0.0174533)) {**  **xk\_1 = Gc\*xk+Hc\*en;**  **u = Cc\*xk+Dc\*en;**  **us = u(0, 0);**  **if (u(0, 0) >= u1max) {**  **us = u1max;**  **}**  **if (u(0, 0) <= u1min) {**  **us = u1min;**  **}**  **ut = us;**  **//a**  **Du = u(0, 0) - ut;**  **xk = xk\_1;**  **// mayor**  **if (en\_3 == enn) ut = ut + (ut \* 0.19);**  **en\_1 = enn;**  **float en\_2 = en\_1;**  **en\_3 = en\_2;**    **arduinoPWM = ut;**  **/\***  **if (abs(rpm) >= 0.7 && abs(rpm) < 3 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.6;**  **if (abs(rpm) >= 3 && abs(rpm) < 7 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.4;**  **if (abs(rpm) >= 7 && abs(rpm) < 10 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.2;**  **if (abs(rpm) >= 10 && abs(rpm) < 15 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2;**  **\*/**  **flagPWM = (ut < 0) ? true : false;**  **}**  **}** |
| Código en el sistema embebido - Metodología Glover-Mcfarlane | **#include "MeanFilterLib.h"**  **#include <BasicLinearAlgebra.h>**  **MeanFilter<float> pwmFil(3);**  **MeanFilter<float> pFil(3);**  **MeanFilter<float> rpmFil(3);**  **const byte Encoder\_C1 = 2; // Cable amarillo pin 3 digital**  **const byte Encoder\_C2 = 3; // Cable verde al pin 4 digital**  **byte Encoder\_C1Last;**  **int paso = 0;**  **boolean direccion, flagPWM, autoincrement;**  **int entrada3 = 13;**  **int entrada4 = 12;**  **int enableB = 11;**  **int maxPWM = 130; // VCC 7V**  **int userE, Sp;**  **const int tMuestro = 30;**  **volatile int n = 0;**  **volatile byte ant = 0;**  **volatile byte act = 0;**  **double P = 0;**  **double R = 3840;**  **float arduinoPWM;**  **float rpm = 0;**  **unsigned long timeold = 0;**  **unsigned long startTime = 0;**  **unsigned long elapsedTime = 0;**  **float ut, us = 0;**  **float u1max = 100;**  **float u1min = -100;**  **float en\_1, en\_2, en\_3 = 0;**  **// NFCSYN**  **BLA::Matrix<3, 3> Gc = {0.853862925588747, 0.187830959530800, 0,**  **-0.187830959530800, 0.954205726727783, 0,**  **0, 0, 1};**  **BLA::Matrix<3, 1> Hc = {-8.29512388959399,**  **-1.99815784321956,**  **-101.934103302858};**  **BLA::Matrix<1, 3> Cc = {-0.0622134291719549, 0.0149861838241466, -0.00900000000000000};**  **BLA::Matrix<1, 1> Dc = {0};**  **BLA::Matrix<1, 1> Ka = {0};**  **BLA::Matrix<1, 1> Du = {0};**  **BLA::Matrix<1, 1> u = {0};**  **BLA::Matrix<3, 1> xk = {0, 0, 0};**  **BLA::Matrix<3, 1> xk\_1 = {0, 0, 0};**  **void reiniciarVAR() {**  **autoincrement = false;**  **startTime = 0; // Reiniciar el tiempo de inicio si autoincrement es false**  **elapsedTime = 0;**  **timeold = 0;**  **rpm = 0;**  **P = 0;**  **Sp = 0;**  **flagPWM = false;**  **analogWrite(enableB, 0); // Equivalente a 50% del máximo (128 de 255)**  **}**  **void calculapulso() {**  **int Lstate = digitalRead(Encoder\_C1);**  **if ((Encoder\_C1Last == LOW) && Lstate == HIGH)**  **{**  **int val = digitalRead(Encoder\_C2);**  **if (val == LOW && direccion)**  **{**  **direccion = false; //Reverse**  **}**  **else if (val == HIGH && !direccion)**  **{**  **direccion = true; //Forward**  **}**  **}**  **Encoder\_C1Last = Lstate;**  **if (!direccion) paso++;**  **else paso--;**  **ant = act;**  **if (digitalRead(Encoder\_C1)) bitSet(act, 1); else bitClear(act, 1);**  **if (digitalRead(Encoder\_C2)) bitSet(act, 0); else bitClear(act, 0);**  **if (ant == 2 && act == 0) n++;**  **if (ant == 0 && act == 1) n++;**  **if (ant == 3 && act == 2) n++;**  **if (ant == 1 && act == 3) n++;**  **if (ant == 1 && act == 0) n--;**  **if (ant == 3 && act == 1) n--;**  **if (ant == 0 && act == 2) n--;**  **if (ant == 2 && act == 3) n--;**  **}**  **void direMotor() {**  **// Cambia la dirección del motor y establece la velocidad**  **if (flagPWM) {**  **digitalWrite(entrada3, LOW);**  **digitalWrite(entrada4, HIGH);**  **} else {**  **// IZQUIERDA POSITIVO**  **digitalWrite(entrada3, HIGH);**  **digitalWrite(entrada4, LOW);**  **}**  **analogWrite(enableB, map(abs(ut), 0, 100, 0, maxPWM));**  **}**  **void setup()**  **{**  **Serial.begin(9600);**  **pinMode(entrada3, OUTPUT);**  **pinMode(entrada4, OUTPUT);**  **pinMode(enableB, OUTPUT);**  **pinMode(Encoder\_C2, INPUT);**  **flagPWM = false;**  **autoincrement = false;**  **attachInterrupt(digitalPinToInterrupt(Encoder\_C1), calculapulso, CHANGE);**  **attachInterrupt(digitalPinToInterrupt(Encoder\_C2), calculapulso, CHANGE);**  **Serial.println("LISTO");**  **}**  **void loop()**  **{**  **if (Serial.available()) { // Verificar si hay datos disponibles en el puerto serial**  **userE = Serial.parseInt(); // Leer el valor entero enviado por el puerto serial**  **// Verificar si el valor recibido está dentro del rango válido (0-100)**  **if (userE != 0 && userE != 404 ) {**  **flagPWM = (userE < 0) ? true : false;**  **autoincrement = true;**  **//Sp = userE; // Para enviar por teclado**  **}**  **if (userE == 404) {**  **reiniciarVAR();**  **}**  **}**  **if (autoincrement == true) {**  **if (startTime == 0) {**  **startTime = millis(); // Registrar el tiempo de inicio**  **}**  **elapsedTime = millis() - startTime; // Calcular el tiempo transcurrido**  **if (elapsedTime > 3000) {**  **if (millis() - timeold >= tMuestro) {**  **escalonPWM();**  **//rpm = (paso \* (0.100 / (millis() - timeold)) \* (60000 / 360)) \* -1 ;**  **//rpm = rpmFil.AddValue(rpm);**  **P = (n \* 360.0) / R;**  **P = pFil.AddValue(P);**  **controlador();**  **direMotor();**  **impresionSerial();**  **timeold = millis();**  **paso = 0;**  **}**  **if (elapsedTime >= 113000 ) {**  **reiniciarVAR();**  **}**  **}**  **}**  **}**  **void escalonPWM() {**  **if (elapsedTime > 3001) Sp = 50; // PWMX60x80**  **if (elapsedTime > 13000) Sp = -60; // PWMX60x80**  **if (elapsedTime > 23000) Sp = 40; // PWMX60x80**  **if (elapsedTime > 33000) Sp = 50; // PWMX60x80**  **if (elapsedTime > 43000) Sp = 70; // PWMX60x80**  **if (elapsedTime > 53000) Sp = -70; // PWMX60x80**  **if (elapsedTime > 63000) Sp = -30; // PWMX60x80**  **if (elapsedTime > 73000) Sp = 30; // PWMX60x80**  **if (elapsedTime > 83000) Sp = -40; // PWMX60x80**  **if (elapsedTime > 93000) Sp = -50; // PWMX60x80**  **if (elapsedTime > 103000) Sp = -60; // PWMX60x80**  **}**  **void impresionSerial() {**  **arduinoPWM = pwmFil.AddValue(arduinoPWM);**  **//Serial.print(Sp ); Serial.print(", "); Serial.print(P ); Serial.print(", "); Serial.print(arduinoPWM); Serial.print("\n");**  **Serial.print(elapsedTime ); Serial.print(", "); Serial.print(Sp ); Serial.print(", "); Serial.print(P ); Serial.print(", "); Serial.print(arduinoPWM); Serial.print("\n");**  **;**  **}**  **// CONTROLADORES**  **void controlador() {**  **BLA::Matrix<1, 1> en = {(Sp \* 0.0174533) - (P \* 0.0174533)};**  **double enn = (Sp \* 0.0174533) - (P \* 0.0174533);**  **if (abs(en(0, 0)) > abs(0.072 \* Sp \* 0.0174533)) {**  **xk\_1 = Gc\*xk+Hc\*en;**  **u = Cc\*xk+Dc\*en;**  **us = u(0, 0);**  **if (u(0, 0) >= u1max) {**  **us = u1max;**  **}**  **if (u(0, 0) <= u1min) {**  **us = u1min;**  **}**  **ut = us;**  **//a**  **Du = u(0, 0) - ut;**  **xk = xk\_1;**  **// mayor**  **if (en\_3 == enn) ut = ut + (ut \* 0.19);**  **en\_1 = enn;**  **float en\_2 = en\_1;**  **en\_3 = en\_2;**    **arduinoPWM = ut;**  **/\***  **if (abs(rpm) >= 0.7 && abs(rpm) < 3 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.6;**  **if (abs(rpm) >= 3 && abs(rpm) < 7 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.4;**  **if (abs(rpm) >= 7 && abs(rpm) < 10 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2.2;**  **if (abs(rpm) >= 10 && abs(rpm) < 15 && (abs(Sp) > abs(P)) && abs(P) < 100) ut = ut \* 2;**  **\*/**  **flagPWM = (ut < 0) ? true : false;**  **}**  **}** |