**ABSTRACCIÓN DE UN SISTEMA MOTOR DC – ENCODER EN UN MODELO DE MACHINE LEARNING PARA SINTONIZAR UN CONTROLADOR DE VELOCIDAD PID APLICANDO ALGORITMOS GENÉTICOS**

# **DEDICATORIA**

# **AGRADECIMIENTO**

# **RESUMEN**

# **ABSTRACT**

# **INTRODUCCIÓN**

## **Realidad problemática**

Es conocida la magnitud de la importancia del control de procesos en ambientes industriales, investigativos y didácticos. Tan es así que, en uno de los casos más difundidos de controladores, como es el controlador Proporcional Integral Derivativo debido a su alto desempeño y facilidad de implementación, se han desarrollado diversos métodos analíticos y experimentales para lograr hallar parámetros que garanticen un comportamiento adecuado del sistema de control. Sin embargo, con el desarrollo de nuevas tecnologías se han abierto múltiples posibilidades para expandir este campo de estudio y abordar la sintonización de controladores desde enfoques distintos a los tradicionales.

Respecto de los métodos modernos, se pueden mencionar el uso de redes neuronales, lógica difusa, algoritmos genéticos, algoritmos de colonia de hormigas, machine learning, etcétera; para funcionar tanto como elementos controladores, así como para optimizar controladores PID que necesitan del modelo matemático o de que el algoritmo funcione sobre la planta en tiempo real. Así también, ciertos métodos se vienen empleando como una alternativa a la identificación de sistemas desde sistemas simples de comportamiento lineal hasta sistemas complejos y caóticos de múltiples variables de comportamiento no lineal.

Entonces se presenta un escenario idóneo para dar soluciones nuevas y optimizadas a la sintonización de controladores, a través de la aplicación conjunta de algoritmos genéticos y machine learning.

## **Formulación del problema**

¿Cómo sintonizar un controlador PID a partir de un conjunto de datos obtenidos por muestreo digital de un sistema motor DC – encoder, aplicando algoritmos genéticos y un método de machine learning?

## **Justificación del estudio**

### **Relevancia Tecnológica**

Las capacidades computacionales de la tecnología actual y el avance de métodos de optimización e identificación de sistemas, presentan un escenario ideal para abordar problemáticas entorno al control de procesos, como es la sintonización de controladores PID, desde nuevos enfoques que integren técnicas que se aplicaban tradicionalmente de forma separada y en otros contextos.

### **Relevancia Institucional**

Resolver problemas de control e identificación de sistemas, son campos todavía en desarrollo, por lo que dar soluciones desde nuevos enfoques como los mencionados en este trabajo, permitirá impulsar la producción científica en la Escuela de Ingenierí Mecatrónica.

### **Relevancia Social**

La integración de técnicas modernas, como las mencionadas en este trabajo, pueden ser un aporte para dar inicio a la exploración de nuevas soluciones a los problemas que los estudiantes o cualquier persona interesada se enfrentan a menudo al tratar desde sistemas lineales de una variable de entrada y una de salida hasta complejos sistemas no lineales de múltiples variables de entrada y de salida.

### **Relevancia Económica**

### Todas las herramientas utilizadas en este trabajo son de uso y distribución libre, por lo que este trabajo permite la apertura a la colaboración y customización que las herramientas como el software privativo no admiten.

### **Relevancia Ambiental**

### El desarrollo de mejores controladores implica optimizar los procesos de control, lo cual, a su vez, está ligado a un ahorro de recursos en las acciones industriales, científicas o didácticas, contribuyendo a la preservación del ambiente.

## **Antecedentes**

## Carlos Pillajo, Paul Bonilla y Roberto Hincapié (2016) presentan la simulación de un algoritmo genético capaz de sintonizar controladores PID basado en el criterio de la integral del error absoluto. Así también, Mohd S. Saad, Hishamuddin Jamaluddin e Intan Z. M. Darus presentan la implementación de un sintonizador de controlador PID usando técnicas de algoritmos genéticos y evolución diferencial, cuyo desempeño es evaluado usando los criterios de la integral del error absoluto y el error cuadrático medio. Además, comparan el desempeño del controlador PID sintonizado, frente al obtenido mediante el método de Ziegler-Nichols, obteniendo resultados positivos.

## Olalekan Ogunmolu, Xuejun Gu, Steve Jiang y and Nicholas Gans demuestran que las redes neuronales pueden proporcionar modelos estimadores efectivos a partir de datos de entrada y salida de sistemas dinámicos no lineales. Así mismo, Yu Wang realiza un estudio comparativo entre dos tipos de redes neuronales, evaluando su desempeño en la identificación de sistemas dinámicos y en el diseño de controladores.

## Por otro lado, Arthur Gretton, Amaud Douce, RalfHerbrich, Peter J. W Rayner y Bernhard Scholkop estudian técnicas de regresión de soporte vectorial para identificación de sistemas de caja negra, aplicadas a un brazo robótico hidráulico, encontrando resultados positivos respecto de publicaciones anteriores. Así también, Robert Salat, Michał Awtoniuk, Krzysztof Korpysz proponen el uso de técnicas de regresión de soporte vectorial para identificación de sistemas de caja negra, como una alternativa útil y eficiente para modelos basados en estructuras ARX y NARX para sistemas lineales y no lineales, respectivamente.

## **Objetivos**

### **General**

Abstraer un sistema motor DC – encoder en un modelo de machine learning para sintonizar un controlador de velocidad PID mediante algoritmos genéticos.

### **Específicos**

* Elección de una señal de entrada para el sistema motor DC – encoder y obtención de un conjunto adecuado de datos experimentales en tiempo discreto para la identificación del sistema.
* Elección de un método de regresión para abstraer el sistema en un modelo de machine learning y definición de sus restricciones.
* Implementación de un controlador PID discreto sobre el modelo de machine learning y definición de su evaluación de desempeño.
* Sintonización del controlador PID aplicado al modelo de machine learning por medio de algoritmos genéticos.

## **Marco teórico**

## **Sistema de control**

## **Controladores**

## **Desempeño de los sistemas de control**

## **Algoritmos genéticos**

## **Identificación de sistemas**

## **Arduino**

## **Jupyter Notebook**

## **Marco conceptual**

## **Variable controlada**

## **Variable manipulada**

## **Planta**

## **Proceso**

## **Perturbación**

## **Señal de referencia**

## **Sensor**

## **Actuador**

## **Modelo matemático**

## **Modelo de caja negra**

## **Modelo de caja blanca**

## 

# **MARCO METODOLÓGICO**

## **Hipótesis**

Es posible aplicar algoritmos genéticos para iterar sobre múltiples valores para parámetros PID que permitan sintonizar un controlador discreto, cuyo desempeño será evaluado en cada iteración sobre la abstracción de un sistema motor DC - encoder, a través de un modelo de machine learning entrenado con datos obtenidos por muestreo digital.

## **Variables**

### **Variables independientes**

* Señal de entrada.
* Frecuencia de muestreo.
* Cantidad de datos.
* Cifras de la ganancia proporcional.
* Cifras de la ganancia integral.
* Cifras de la ganancia derivativa.
* Resolución digital de la ganancia proporcional.
* Resolución digital de la ganancia integral.
* Resolución digital de la ganancia derivativa.
* Tamaño de la población.
* Probabilidad de recombinación.
* Probabilidad de mutación.
* Número de genes.

### **Variables dependientes**

* Modelo de machine learning
* Ganancia proporcional.
* Ganancia integral.
* Ganancia derivativa.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Definición conceptual** | **Definición operacional** | **Indicadores** | **Escala de medición** |
| Señal de entrada | Señal de excitación sobre la planta de control. | Señal de excitación escogida para estudiar la planta de control | PWM | 0 - 255 |
| Frecuencia de muestreo | Número de muestras por unidad de tiempo que se toman de una [señal continua](https://es.wikipedia.org/wiki/Se%C3%B1al_anal%C3%B3gica) para producir una señal discreta. | Frecuencia de muestreo de las señales de entrada y salida de la planta de control. | Hz | 0 - 50000 |
| Número de muestras | Cantidad de datos obtenidas en un proceso de recolección. | Cantidad de datos obtenidos por el proceso de muestreo digital. | - | 0 – 100 K |
| Cifras de la ganancia proporcional | Cantidad de símbolos o caracteres gráficos que sirven para representar la ganancia proporcional. | Cantidad de dígitos del valor máximo de la ganancia proporcional. | Nº | 0 – 1000 |
| Cifras de la ganancia integral | Cantidad de símbolos o caracteres gráficos que sirven para representar la ganancia integral. | Cantidad de dígitos del valor máximo de la ganancia integral. | Nº | 0 – 1000 |
| Cifras de la ganancia derivativa | Cantidad de símbolos o caracteres gráficos que sirven para representar la ganancia derivativa. | Cantidad de dígitos del valor máximo de la ganancia derivativa. | Nº | 0 – 1000 |
| Resolución de la ganancia proporcional | Longitud de la palabra digital que conforma la ganancia proporcional. | Cantidad de bits de la ganancia proporcional. | Nº | 0 – 100 |
| Resolución de la ganancia integral | Longitud de la palabra digital que conforma la ganancia integral. | Cantidad de bits de la ganancia integral. | Nº | 0 – 100 |
| Resolución de la ganancia derivativa | Longitud de la palabra digital que conforma la ganancia derivativa. | Cantidad de bits de la ganancia derivativa. | Nº | 0 – 100 |
| Población | Conjunto de seres vivos de la misma especie que habitan en un lugar determinado. | Tamaño o número de cromosomas de la población inicial o posteriores. | Nº | 0 – 100 |
| Recombinación | La recombinación de la [reproducción sexual](https://es.wikipedia.org/wiki/Reproducci%C3%B3n_sexual) biológica. | [Operador genético](https://es.wikipedia.org/wiki/Operador_gen%C3%A9tico_(computaci%C3%B3n_evolutiva)) que genera la variación de un [cromosoma](https://es.wikipedia.org/wiki/Cromosoma_(computaci%C3%B3n_evolutiva)) o cromosomas de una generación a la siguiente. | % | 0 – 1 |
| Mutación | Alteración permanente de la [secuencia](https://en.wikipedia.org/wiki/Base_sequence) de [nucleótidos](https://en.wikipedia.org/wiki/Base_sequence) del [genoma](https://en.wikipedia.org/wiki/Genome) de un elemento genético. | [Operador genético](https://en.wikipedia.org/wiki/Genetic_operator) que mantiene [la diversidad genética](https://en.wikipedia.org/wiki/Genetic_diversity) de una generación a la siguiente. | % | 0 – 1 |
| Generaciones | Conjunto de individuos creados en un mismo periodo de tiempo. | Número de generaciones a analizar por el algoritmo genético. | Nº | 0 – 100 |
| Ganancia proporcional | Valor del controlador que puede reducir, pero no eliminar, el error en estado estacionario. | Salida del controlador es proporcional a la señal de error. | Nº | 0 – 1000 |
| Ganancia integral | Valor del controlador que elimina errores estacionarios. | Salida del controlador que es la integral de la señal de error. | Nº | 0 – 1000 |
| Ganancia derivativa | Valor del controlador que anticipa el efecto de la acción proporcional para estabilizar la variable controlada después de cualquier perturbación. | Salida del controlador que es la derivada de la señal de error. | Nº | 0 – 1000 |

## **Metodología**

### **Tipo de estudio**

# Exploratorio

### **Diseño**

# Experimental.

## **Población y muestra**

### **Población**

Sistemas lineales SISO.

### **Muestra**

Sistema motor DC – encoder.

## **Método de investigación**

## Cuantitativa

## **Técnicas e instrumentos de recolección de datos**

### **Técnicas**

# Revisión de bibliografía especializada para explorar señales de excitación sobre diversos sistemas de control.

# Estudio del comportamiento del sistema de control frente a diversos estímulos.

### **Instrumentos**

# Jupyter Notebook

# Plataforma Arduino

# Actuador

# Sensor

### **Métodos de análisis de datos**

# Pre procesamiento de la señal de salida de la planta de control: aplicación de filtrado digital sobre los datos provenientes del sensor para su posterior procesamiento.

# **DESARROLLO Y RESULTADOS**

## **Elección de una señal de entrada para el sistema motor DC – encoder y obtención de un conjunto adecuado de datos experimentales en tiempo discreto para la identificación del sistema.**

Después de realizar pruebas experimentales con señales de entrada de tipo escalón y pulso, con distintos valores de amplitud y duración, fue posible determinar que un cambio en la excitación del sistema motor DC - encoder requiere un tiempo no mayor a 5 segundos para estabilizarse, sin importar la amplitud de la señal. Por lo que se eligió como señal de entrada para el sistema, a una señal tipo escalón con amplitud variable entre 0 y 255, y una duración de 5 segundos.

Además, para el proceso de muestreo de las señales de entrada y salida del sistema, se determinó un tiempo de muestreo de ms.

A continuación, se muestra la implementación del sistema:

A continuación, se muestra el código de programación de la placa arduino para el muestreo digital del sistema:

volatile unsigned int cuenta = 0;

int potAnaPin = 0;

int PwmPin = 9;

int encoderDIGPin = 2;

unsigned long t1, t2;

float t, f, f2;

int Sp, n=0;

void setup() {

Serial.begin(115200);

pinMode(PwmPin, OUTPUT); // sets the pin as output

pinMode(encoderDIGPin, INPUT);

SREG = (SREG & 0b01111111); //Desabilitar interrupciones

TIMSK2 = TIMSK2|0b00000001; //Habilita la interrupcion por desbordamiento

TCCR2B = 0b00000111; //Configura preescala para que FT2 sea de 7812.5Hz

SREG = (SREG & 0b01111111) | 0b10000000; //Habilitar interrupciones //Desabilitar interrupciones

}

void loop() {

t1 = pulseIn(encoderDIGPin, HIGH);

t2 = pulseIn(encoderDIGPin, LOW);

t = t1 + t2;

if (t == 0) {

f = 500000;

} else {

f = 500000/t;

}

if (f > 1000) {

f = 0;

}

f = f \* 17 / 8;

f = 0.0609 \* f + f2 \* 0.9391;

f2 = f;

if (t == 0) {

f = 0;

}

analogWrite(PwmPin, Sp);

}

ISR(TIMER2\_OVF\_vect){ // función que se llama cada 32.64ms

cuenta++;

if(cuenta > 150) { // CAMBIANDO EL SETPOINT CADA 150 cuentas = 150 \* 32.64ms

Sp = random(0, 255);

cuenta=0;

}

if(cuenta % 7 == 0) { // MUESTREO CADA 7 cuentas = 7 \* 32.64m

analogWrite(PwmPin, Sp);

Serial.print(Sp);

Serial.print(" ");

Serial.print(f);

Serial.println(" ");

}

}

**import** **serial**

**from** **time** **import** time

ser = serial.Serial('/dev/ttyUSB1', 115200)

start\_time = time()

timepoints = []

ydata = []

yydata = []

yrange = [-0.1,5.1]

view\_time = 4 *#seconds of data to view at once*

duration = 4000 *#total seconds to collect data*

*# flush any junk left in the serial buffer*

ser.flushInput()

*# ser.reset\_input\_buffer() # for pyserial 3.0+*

run = **True**

*# collect the data and plot a moving frame*

**while** run:

ser.reset\_input\_buffer()

data = ser.readline().decode("utf-8").split(' ')

*# sometimes the incoming data is garbage, so just 'try' to do this*

**try**:

*# store the entire dataset for later*

ydata.append(float(data[0]))

yydata.append(float(data[1]))

timepoints.append(time()-start\_time)

current\_time = timepoints[-1]

*# when time's up, kill the collect+plot loop*

**if** timepoints[-1] > duration:

run=**False**

*# if the try statement throws an error, just do nothing*

**except**: **pass**

ser.close()

workbook = xlsxwriter.Workbook('motor.xlsx')

worksheet = workbook.add\_worksheet()

worksheet.write('A1', 'time')

worksheet.write('B1', 'output')

worksheet.write('C1', 'input')

k = 1

Ts = 0.22848

ydata\_par = ydata

ydata\_impar = yydata

**for** output, entry **in** zip(ydata\_par, ydata\_impar):

worksheet.write(k, 0, Ts \* k)

worksheet.write(k, 1, output)

worksheet.write(k, 2, entry)

k += 1

workbook.close()

## **Elección del modelo de machine learning y definición de sus restricciones para abstraer el sistema**

:

x = pd.read\_excel('motor.xlsx')

L = len(x)

q1 = pd.DataFrame()

q1['input1'] = pd.Series(x['input'], index=x.index)

q1['y1'] = pd.Series(x['output'], index=x.index)

q1 = q1.shift(periods=-1, freq=**None**, axis=0)

*#q1 = q1.drop(q1.index[0])*

x['input1'] = pd.Series(q1['input1'], index=x.index)

x['y1'] = pd.Series(q1['y1'], index=x.index)

q2 = pd.DataFrame()

q2['input2'] = pd.Series(x['input'], index=x.index)

q2['y2'] = pd.Series(x['output'], index=x.index)

q2 = q2.shift(periods=-2, freq=**None**, axis=0)

*#q2 = q2.drop(q2.index[0:2])*

x['input2'] = pd.Series(q2['input2'], index=x.index)

x['y2'] = pd.Series(q2['y2'], index=x.index)

q3 = pd.DataFrame()

q3['input3'] = pd.Series(x['input'], index=x.index)

q3['y3'] = pd.Series(x['output'], index=x.index)

q3 = q3.shift(periods=-3, freq=**None**, axis=0)

x['input3'] = pd.Series(q3['input3'], index=x.index)

x['y3'] = pd.Series(q3['y3'], index=x.index)

q4 = pd.DataFrame()

q4['input4'] = pd.Series(x['input'], index=x.index)

q4['y4'] = pd.Series(x['output'], index=x.index)

q4 = q4.shift(periods=-4, freq=**None**, axis=0)

x['input4'] = pd.Series(q4['input4'], index=x.index)

x['y4'] = pd.Series(q4['y4'], index=x.index)

*'''*

*q5 = pd.DataFrame()*

*q5['input5'] = pd.Series(x['input'], index=x.index)*

*q5['y5'] = pd.Series(x['output'], index=x.index)*

*q5 = q5.shift(periods=-5, freq=None, axis=0)*

*x['input5'] = pd.Series(q5['input5'], index=x.index)*

*x['y5'] = pd.Series(q5['y5'], index=x.index)*

*'''*

x = x.drop(x.index[17849:17853])

*#y = x['antihorapos']*

y = pd.DataFrame()

y['output'] = x['output']

x = x.drop('output', axis=1)

x = x.drop('time', axis=1)

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.4)

**from** **sklearn.linear\_model** **import** ElasticNet

model = ElasticNet()

model.fit(x\_train, y\_train)

model.score(x\_test,y\_test)

predicted = model.predict(x\_test)

**import** **matplotlib.pyplot** **as** **plt** %matplotlib inline plt.hist([predicted, y\_test['output']])

## **Implementación del controlador PID discreto sobre el modelo de machine learning y definición de su evaluación de desempeño**

**def** evaluate(controller):

controller = ''.join(map(str,controller))

*# PID controller*

\_kp = int(controller[0:10],2)

\_ki = int(controller[10:20],2)

\_kd = int(controller[20:30],2)

kp = 0 + \_kp \* ( (10 - 0) / (pow(2,10) - 1) )

ki = 0 + \_ki \* ( (10 - 0) / (pow(2,10) - 1) )

kd = 0 + \_kd \* ( (10 - 0) / (pow(2,10) - 1) )

*# =====================================================================================================*

pid1 = 0

pid2 = 0

pid3 = 0

pid4 = 0

error0 = 0

error1 = 0

error2 = 0

y1 = 0

y2 = 0

y3 = 0

y4 = 0

E = []

setPoint = 20 *#800*

Ts = 0.22848

**for** k **in** np.arange(1000):

q1 = kp + 0.5 \* Ts \* ki + (kd / Ts)

q2 = Ts \* ki \* 0.5 - kp - (2 \* kd / Ts)

q3 = kd / Ts

pid0 = pid1 + q1 \* error0 + q2 \* error1 + q3 \* error2

pid4 = pid3

pid3 = pid2

pid2 = pid1

pid1 = pid0

error1 = error0

error2 = error1

**if** (pid0 > 255):

pid0 = 255

**elif** (pid0 < 0):

pid0 = 0

**else**:

pid0 = pid0

*#d = {'pwm': [pid0], 'pwm1': [pid1], 'y1': [y1], 'pwm2': [pid2], 'y2': [y2]}*

d = {'pwm': [pid0], 'pwm1': [pid1], 'y1': [y1], 'pwm2': [pid2], 'y2': [y2], 'pwm3': [pid3], 'y3': [y3], 'pwm4': [pid4], 'y4': [y4]}

\_y = model.predict(pd.DataFrame(data=d))

y = \_y[0]

**if** (y > 252): *#1023):*

y = 252 *#1023*

**elif** (y < 0): *#30):*

y = 0 *#30*

**else**:

y = y

y1 = y

y2 = y1

y3 = y2

y4 = y3

error0 = setPoint - y

E.append(error0)

total\_error = sum(map(abs, E))

print([total\_error])

**return** (total\_error),

## **Sintonización del controlador PID por medio de algoritmos genéticos**

**from** **deap** **import** base, creator

*# Crear clase FITNESS: ( Nombre de la clase, Delegando la clase como fitness, Posibles valores min y max del fitness )*

creator.create("ControllerFitness", base.Fitness, weights=(-1.0,))

*# Crear clase CROMOSOMA: ( Nombre de la clase, type de los valores de la clase, Establecer fitness con la clase anterior )*

creator.create("Controller", list, fitness=creator.ControllerFitness)

**import** **random**

**from** **deap** **import** tools

IND\_SIZE = 30 *# ki bits + kp bits + kd bits*

*# Functions to initialize populations from individuals*

toolbox = base.Toolbox()

*# Crear el ATRIBUTO de cromosomas ( Nombre del registro para el toolbox, Función para crear un cromosoma aleatorio )*

toolbox.register("bit\_attribute", random.randint, 0, 1) *# random.choice, '10')*

*# Crear el CROMOSOMA ( Nombre del registro, , La clase cromosoma, La clase atributo, Tamaño de la población )*

*# Cuando se llame a toolbox.individual(), este llamará a tool.initRepeat()*

toolbox.register("controller", tools.initRepeat, creator.Controller, toolbox.bit\_attribute, n=IND\_SIZE)

*# Crear la POBLACIÓN ( Nombre del registro, , Type del cromosoma, El cromosoma )*

toolbox.register("population", tools.initRepeat, list, toolbox.controller)

toolbox.register("mate", tools.cxTwoPoint)

toolbox.register("mutate", tools.mutUniformInt, low=0, up=1, indpb=0.1)

toolbox.register("select", tools.selTournament, tournsize=3)

toolbox.register("evaluate", evaluate)

**def** main():

pop = toolbox.population(n=40)

*# pop[0:100] = prev*

CXPB, MUTPB, NGEN = 0.7, 0.1, 40

*# Evaluate the entire population*

fitnesses = list(map(toolbox.evaluate, pop))

**for** ind, fit **in** zip(pop, fitnesses):

ind.fitness.values = fit

minFit = (1,)

*#while minFit > (0.02,):*

**for** g **in** range(NGEN):

*# Select the next generation individuals*

offspring = toolbox.select(pop, len(pop))

*# Clone the selected individuals*

offspring = list(map(toolbox.clone, offspring))

*# Apply crossover and mutation on the offspring*

**for** child1, child2 **in** zip(offspring[::2], offspring[1::2]):

**if** random.random() < CXPB:

toolbox.mate(child1, child2)

**del** child1.fitness.values

**del** child2.fitness.values

**for** mutant **in** offspring:

**if** random.random() < MUTPB:

toolbox.mutate(mutant)

**del** mutant.fitness.values

*# Evaluate the individuals with an invalid fitness*

invalid\_ind = [ind **for** ind **in** offspring **if** **not** ind.fitness.valid]

fitnesses = list(map(toolbox.evaluate, invalid\_ind))

minFit = min(fitnesses)

print("=START=================================================================")

print(fitnesses)

print("=END===================================================================")

**for** ind, fit **in** zip(invalid\_ind, fitnesses):

ind.fitness.values = fit

*# The population is entirely replaced by the offspring*

pop[:] = offspring

**return** pop

kx = main()

**for** bits **in** kx:

controller = ''.join(map(str,bits))

*# PID controller*

\_kp = int(controller[0:10],2)

\_ki = int(controller[10:20],2)

\_kd = int(controller[20:30],2)

kp = 0 + \_kp \* ( (1 - 0) / (pow(2,10) - 1) )

ki = 0 + \_ki \* ( (1 - 0) / (pow(2,10) - 1) )

kd = 0 + \_kd \* ( (1 - 0) / (pow(2,10) - 1) )

print([\_kp,\_ki,\_kd,kp,ki,kd])

controller = ''.join(map(str,kx[1]))

*# PID controller*

\_kp = int(controller[0:10],2)

\_ki = int(controller[10:20],2)

\_kd = int(controller[20:30],2)

kp = 0 + \_kp \* ( (1 - 0) / (pow(2,10) - 1) )

ki = 0 + \_ki \* ( (1 - 0) / (pow(2,10) - 1) )

kd = 0 + \_kd \* ( (1 - 0) / (pow(2,10) - 1) )

print([\_kp,\_ki,\_kd,kp,ki,kd])

volatile unsigned int cuenta = 0;

int potAnaPin = 0; // pin A0

int PwmPin = 9; // pin 9

int encoderDIGPin = 2;

unsigned long t1, t2;

float t, f, f2;

int Sp, n=0;

float pidVER = 0;

float pid0 = 0;

float pid1 = 0;

float pid2 = 0;

float error0 = 0;

float error1 = 0;

float error2 = 0;

float y1 = 0;

float y2 = 0;

float kp = 0.3225806451612903; //0.07820136852394917; //0.5034213098729228;

float ki = 1; //1.2512218963831867; ///0.9843597262952102;

float kd = 0.009775171065493646; //0.17595307917888564; //0.028347996089931573;

float Ts = 0.22848; //0.1;

float q1 = kp + 0.5 \* Ts \* ki + (kd / Ts);

float q2 = Ts \* ki \* 0.5 - kp - (2 \* kd / Ts);

float q3 = kd / Ts;

float c1=0;

void setup() {

Serial.begin(115200);

pinMode(PwmPin, OUTPUT); // sets the pin as output

pinMode(encoderDIGPin, INPUT);

SREG = (SREG & 0b01111111); //Desabilitar interrupciones

TIMSK2 = TIMSK2|0b00000001; //Habilita la interrupcion por desbordamiento

TCCR2B = 0b00000111; //Configura preescala para que FT2 sea de 7812.5Hz

SREG = (SREG & 0b01111111) | 0b10000000; //Habilitar interrupciones //Desabilitar interrupciones

}

void loop() {

t1 = pulseIn(encoderDIGPin, HIGH);

t2 = pulseIn(encoderDIGPin, LOW);

t = t1 + t2;

if (t == 0) {

f = 500000;

} else {

f = 500000/t;

}

if (f > 1000) {

f = 0;

}

f = f \* 17 / 8;

f = 0.0609 \* f + f2 \* 0.9391;

f2 = f;

if (t == 0) {

f = 0;

}

analogWrite(PwmPin, pid0);

//Sp0 = analogRead(potAnaPin) / 8; // Read potenciometer signal

//error0 = Sp0 - f;

}

ISR (TIMER2\_OVF\_vect){ // función que se llama cada 32.64ms

cuenta++;

if(cuenta > 300) { // CAMBIANDO EL SETPOINT CADA 150 cuentas = 150 \* 32.64ms

Sp = random(0, 252);

cuenta=0;

}

if(cuenta % 7 == 0) { // MUESTREO CADA 7 cuentas = 7 \* 32.64mfloat pid0 = pid1 + q1 \* error0 + q2 \* error1 + q3 \* error2;

error0 = Sp - f;

pid0 = pid1 + q1 \* error0 + q2 \* error1 + q3 \* error2;

pid2 = pid1;

pid1 = pid0;

error2 = error1;

error1 = error0;

if (pid0 > 255)

{

pidVER = pid0;

pid0 = 255;

}

else if (pid0 < 0)

{

pid0 = 0;

}

else

{

pid0 = pid0;

}

//float c0 = 0.0609 \* pid0 + 0.9391 \* c1;

//c1 = c0;

//analogWrite(PwmPin, pid0);

Serial.print(Sp);

Serial.print(" ");

Serial.print(f);

Serial.println(" ");

}

}

# **DISCUCIÓN**

## **Elección la señal de entrada para el sistema motor DC - encoder**

## **Obtención de un conjunto de datos experimentales en tiempo discreto**

## **Elección del modelo de machine learning y definición de sus restricciones para abstraer el sistema**

## **Implementación del controlador PID discreto sobre el modelo de machine learning y definición de su evaluación de desempeño**

## **Sintonización del controlador PID por medio de algoritmos genéticos**

# **CONCLUSIONES**

## **Elección la señal de entrada para el sistema motor DC - encoder**

## **Obtención de un conjunto de datos experimentales en tiempo discreto**

## **Elección del modelo de machine learning y definición de sus restricciones para abstraer el sistema**

## **Implementación del controlador PID discreto sobre el modelo de machine learning y definición de su evaluación de desempeño**

## **Sintonización del controlador PID por medio de algoritmos genéticos**

# **SUGERENCIAS**

# **BIBLIOGRAFÍA**

# **ANEXOS**