UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE INGENIERÍA ELECTRÓNICA

SINTONIZACIÓN DE UN CONTROLADOR PID MEDIANTE PSO APLICADO A UN SISTEMA BALL AND PLATE REALIMENTADO CON VISIÓN ARTIFICIAL

Trabajo de titulación previo a la obtención del
Título de Ingeniero Electrónico

AUTOR: Darwin Kevin Torres Guañuna

TUTOR: William Manuel Montalvo López

Quito, Ecuador

2022

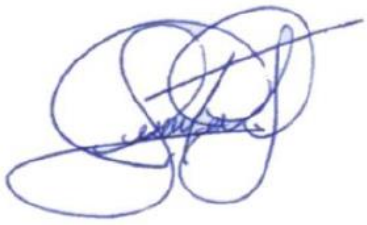
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Darwin Kevin Torres Guañuna con documento de identificación N° 0502869118, manifiesto que:

Soy autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 17 de marzo del año 2022

Atentamente,



_____

Darwin Kevin Torres Guañuna
0502869118

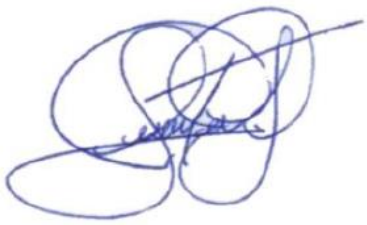
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Darwin Kevin Torres Guañuna con documento de identificación No. 0502869118, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del proyecto técnico de titulación : "Sintonización de un controlador PID mediante PSO aplicado a un sistema ball and plate realimentado con visión artificial", el cual ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 17 de marzo del año 2022

Atentamente,



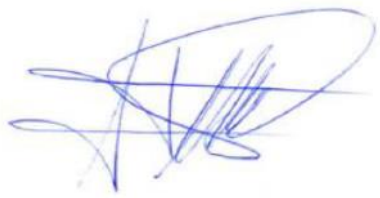
Darwin Kevin Torres Guañuna
0502869118

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, William Manuel Montalvo López con documento de identificación N° 1712789989, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: SINTONIZACIÓN DE UN CONTROLADOR PID MEDIANTE PSO APLICADO A UN SISTEMA BALL AND PLATE REALIMENTADO CON VISIÓN ARTIFICIAL, realizado por Darwin Kevin Torres Guañuna con documento de identificación N° 0502869118 , obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 17 de marzo del año 2022

Atentamente,



Ing. William Manuel Montalvo López M.Sc
1712789989

DEDICATORIA

Éste proyecto está dedicado a toda mi familia, que siempre ha logrado apoyarme de alguna u otra forma para poder seguir cada vez adelante y en especial a mis padres los cuales han permanecido constantemente brindándome todo su apoyo, aún en los tiempos más difíciles, tanto en lo académico como en el ámbito personal siendo un pilar importante en mi vida, a mis hermanos por estar al pendiente de todos mis pasos. Y sobre todo a mi abuelito José, quien supo apoyarme económicamente en mi primera etapa universitaria ya que estaba convencido de las ganas de estudiar que tenía.

Darwin Torres

AGRADECIMIENTO

Agradezco al ingeniero Hamilton Núñez por haberme apoyado con el tema de tesis, brindado con ideas y sugerencias que valían la pena cuando otros docentes solo hacían caso omiso, Al ingeniero William Montalvo por el tiempo y dedicación sobre las correcciones y sugerencias del proyecto de tesis, siendo responsable y puntual en sus acciones. Al ingeniero William Oñate por su preocupación y sugerencias precisas que supo brindar en momentos de incertidumbre.

Darwin Torres

ÍNDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE ANEXOS

RESUMEN

Para la sintonización de un controlador PID se implementó un algoritmo de optimización de partículas o PSO (Particle Swarm Optimization), cuyo objetivo es encontrar los coeficientes proporcional, integral y derivativo e introducirlos de forma autónoma en un controlador PID en lazo cerrado, esto permite controlar el movimiento de la bola hasta dirigirla a la posición deseada, conocida como set point, el sistema ball and plate posee un algoritmo de visión artificial que permite realimentar la posición de la bola sobre el plano mediante la captura, análisis y procesamiento de las imágenes obtenidas en la ejecución del programa. Para el desarrollo de este proyecto se utilizó el lenguaje de programación de python en el programa visual studio, en el cual se ejecuta una interfaz gráfica donde se observa un plano en 2D donde contiene el movimiento de la bola, la simulación del sistema se puede efectuar mediante un host local creado en el algoritmo del programa, se realiza cálculos de errores usando la integral del error absoluto (IEA) de las gráficas obtenidas en tiempo real de la posición vs tiempo entre el método tradicional de Ziegler-Nichols y el método realizado en este proyecto usando PSO.

ABSTRACT

For the tuning of a PID controller, a particle optimization algorithm or PSO (Particle Swarm Optimization) was implemented, whose objective is to find the proportional, integral and derivative coefficients and introduce them autonomously in a closed-loop PID controller, allowing to control the movement of the ball to direct it to the desired position, known as set point, the ball and plate system has an artificial vision algorithm that allows to feedback the position of the ball on the plane through the capture, analysis and processing of the images obtained in the execution of the program. For the development of this project the python programming language was used in the visual studio program, a graphical interface was created where a 2D plane containing the movement of the ball is observed, the system simulation can be executed through a local host created in the program algorithm, error calculations were performed using the integral of the absolute error (IEA) of the graphs obtained in real time of the position vs time between the traditional method of Ziegler-Nichols and the method performed in this project using PSO.

INTRODUCCIÓN

En el diseño y construcción de un controlador PID es indispensable calcular los coeficientes proporcional, integral y derivativo mediante tablas y valores aproximados que se obtienen de las gráficas de representación de estabilización del PID, teniendo en cuenta que valores como la ganancia crítica y periodo crítico se obtienen de acuerdo a la experticia del programador.

El problema radica en que si alguna de las variables que interfieren en el proceso cambia como el peso de la bola, diámetro, rozamiento, etc. El PID tiene que ser sintonizado de acuerdo al cambio físico realizado, ya que el método tradicional no es adaptable a cambios mínimos que pueden suscitarse en el proceso. Existen algunos métodos que intentan simular un PID usando inteligencia artificial que son adaptables a diferentes entornos, pero sacrifican el funcionamiento real de un controlador PID.

Para este fin se desarrolla un algoritmo que permita encontrar los valores más aproximados y con mejores resultados para el funcionamiento de un PID usando PSO (Particle Swarm Optimization), siendo éste un algoritmo bio-inspirado que emula el comportamiento social de enjambres, tomando el menor esfuerzo pero con resultados muy óptimos al momento de encontrar una solución a un problema dado.

Finalmente se simulará una interfaz gráfica que permita interactuar con el usuario para estudiar y comprender de mejor manera el uso de este método con el sistema de ball and plate, obteniendo graficas en tiempo real del control de la bola sobre el plano que permiten el cálculo de errores fundamentales para conocer la eficacia del método realizado.

CAPÍTULO 1

ANTECEDENTES

1.1. Planteamiento del problema

A los estudiantes de la Universidad Politécnica Salesiana se les dificulta realizar una sintonización de los parámetros a utilizar en un controlador PID en el sistema ball and plate que existe en el laboratorio de teoría de control, por lo que se llega a la siguiente pregunta ¿Qué técnicas prácticas pueden utilizar los docentes para identificar y guiar mejor a los estudiantes en la sintonización de un PID? Y ¿Qué efecto tiene la limitada enseñanza sobre algoritmos genéticos impartida a nivel universitario en la academia? Además los estudiantes no tienen la experticia necesaria para llegar a una conclusión con los valores necesarios para la sintonización de un controlador PID utilizando métodos empíricos como es el caso de Ziegler-Nichols, entonces, ¿En qué condiciones de experiencia se encuentran los estudiantes que se encuentran realizando sus prácticas en el laboratorio de teoría de control?

1.2. Justificación

El desarrollo de este proyecto tiene una amplia aplicación en problemas prácticos a nivel industrial, ya que busca una sintonización automática de sus parámetros requeridos para el controlador PID, por ejemplo, si en un proceso se decide cambiar de sustancia con la que se trabaja y este resulta ser más lento de procesarse, los operadores ya no tendrán que parar la planta para realizar una nueva sintonización de los parámetros del PID que están utilizando, ya que el PID es uno de los controladores más utilizados debido a su funcionalidad simple y robusta para trabajar en diferentes procesos (Coba W. 2015).

Por otra parte, se pretende apoyar la idea del uso de inteligencia artificial en sistemas de control, según García el uso del mismo ayuda a una autonomía del controlador con la mínima interacción humana posible (García F. 2019). Además, la creación de este proyecto permite recolectar datos de velocidad y tiempo de reacción del controlador que tiene la inteligencia artificial en la sintonización del mismo, así como el análisis con otros métodos para demostrar su efectividad.

1.3. Objetivos

1.3.1. Objetivo general

Optimizar la sintonización de un controlador PID utilizando el método PSO aplicado sobre un sistema de equilibrio ball and plate con realimentación por visión artificial.

1.3.2. Objetivos específicos

- Caracterizar el sistema ball and plate para la determinación de las variables de entrada y salida.
- Utilizar visión artificial para la ubicación de la esfera sobre el plato del ball and plate.
- Implementar la optimización por enjambre de partículas en la sintonización para ajustar un controlador PID aplicado al control del sistema ball and plate.
- Realizar un análisis comparativo de la sintonización del controlador PID entre PSO y Ziegler – Nichols para la comprobación del método con mejor desempeño de equilibrio utilizando como variable de prueba la integral del error absoluto (IEA).

1.4. Beneficiarios

El presente proyecto sirve para facilitar la obtención de los parámetros proporcional, integral y derivativo correctos para el “Ball and Plate Control System” permitiendo incorporar nuevos métodos que faciliten y mejoren el sistema de sintonización para controles PID en el cual serán beneficiados docentes y estudiantes que realicen las prácticas en el laboratorio de control de la Universidad Politécnica Salesiana logrando así una autonomía por parte de los estudiantes y una herramienta de sustento para los docentes.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1 Algoritmos bio-inspirados

Los modelos de computación bio-inspirados o también llamados algoritmos bio-inspirados han tomado mucha popularidad en los últimos 20 años, debido a la gran versatilidad y funcionalidad al momento de resolver problemas del mundo real, además, se han caracterizado por tener resultados prometedores al momento de resolver problemas complejos utilizando comportamientos naturales o sociales para llegar a una solución en común. Como los algoritmos bio-inspirados resultan ser muy flexibles y con excelentes resultados en cuanto a la toma de decisiones se han adaptado en diferentes áreas ocupacionales como en la industria, comercio, medicina, computación, minería, ciencias e ingenierías (Yang, 2014).

Como objetivo fundamental del algoritmo bio-inspirado es encontrar una solución a un problema real mediante secuencias y reglas sencillas, aplicando el mínimo esfuerzo y conocimiento sobre el espacio mediante una búsqueda aleatoria que resulta ser efectiva y sofisticada, obteniendo así mejores resultados.

La forma de operación de este algoritmo se basa en dos métodos de búsqueda, local y global, para mantener una relación entre diversificación e intensificación, la primera se utiliza a nivel global de forma aleatoria para realizar una exploración del espacio que se requiere. Mientras que la segunda se enfoca a nivel local para establecer la solución más prometedora en un lugar en específico (Martínez - Cosío, 2016).

2.2 Algoritmos de optimización por enjambre de partículas

Este modelo de algoritmo fue desarrollado por Eberhart, R. y Kennedy, J. en el año 1995, el motivo de su investigación fue en base al comportamiento colectivo de distintas sociedades de aves y peces, lo cual tuvo como principal objetivo simular a este comportamiento con un enfoque estocástico los patrones de movimiento de estas especies como se aprecia en la Figura 2.1, siendo así un modelo bio-inspirado

computacionalmente en donde cada uno de los individuos toman el nombre de "partículas" (Eberhart y Kennedy, 1995).

Figura 2.1 Comportamiento social de aves

Comportamiento social de aves, Fuente: Foto de freestocks.org en Pexels

2.2.1 Descripción

La optimización por enjambre de partículas o también llamado PSO por sus siglas en inglés (particle swarm optimization) es uno de los métodos más populares en cuanto a los algoritmos bio-inspirados que existen, este método imita el comportamiento colectivo y social de una especie para resolver problemas difíciles mediante información simple de cada individuo, como es el caso de grupos de aves y peces. Este algoritmo a pesar de no trabajar con parámetros de mutación y cruce tiene ciertas similitudes con los algoritmos evolutivos ya que con una población aleatoria buscan la solución más óptima con el pasar de las generaciones (Martínez, 2018).

En el modelo PSO, un enjambre de partículas con una cantidad n de individuos tienen la capacidad de realizar puntos de búsqueda en donde almacenan la mejor posición que estos hayan descubierto y posiblemente tratan de comunicarse entre sí de manera directa o indirecta para una resolución de problemas de búsqueda. En este modelo computacional las partículas nunca mueren (Eberhart y Kennedy, 2001).

2.2.2 Funcionamiento

Para poner en funcionamiento el algoritmo mediante PSO de manera práctica, se debe generar inicialmente una población aleatoria manteniendo su posición desde que son que creadas hasta que son reemplazadas, debido a que deben ser evaluadas cada conjunto de partículas con un cierto criterio de aptitud, finalizado este proceso se obtiene toda la información de exploración del espacio de forma individual y grupal.

Por consiguiente es asignada una velocidad inicial a cada partícula para generar un desplazamiento de las mismas por medio del espacio, luego de varias iteraciones, las partículas son forzadas a una aceleración continua hacia las que proveen mejor información, permitiendo así converger sobre una posible solución (Shi y Eberhart, 1999).

Aplicando matemáticamente se puede generar un conjunto de partículas en donde se les asigna una vector de posición como se muestra en la Ecuación 2.1.

$$x_i = [x_1 \quad x_2 \quad x_n] \tag{2.1}$$

En donde cada partícula debe ser inicializada con un valor de posición aleatorio uniforme con sus respectivos límites superiores e inferiores en el área a desarrollarse. Como segundo parámetro fundamental es la inicialización de un vector de velocidad para las partículas creadas como se observa en la Ecuación 2.2.

$$v_i = [v_1 \quad v_2 \quad v_n] \tag{2.2}$$

Este vector v_i debe ser inicializado en cero, además, tanto como para v_i y x_i deben tomar valores pertenecientes a los reales.

También existen dos parámetros más que si bien no se utilizan para la creación de las partículas son indispensables como memoria de almacenamiento para cada itinerantica. De esta manera se tiene la mejor posición registrada por cada individuo y

la mejor posición registrada de manera global de todas las partículas como se aprecia en las Ecuaciones 2.3 y 2.4 respectivamente.

$$pBest_i = [pBest_1 \quad pBest_2 \quad pBest_n] \tag{2.2}$$

$$gBest_i = [gBest_1 \quad gBest_2 \quad gBest_n] \tag{2.3}$$

Después de haber inicializado las variables de almacenamiento de las partículas, el valor del vector $pBest_i$ toma los valores de la posición inicial de x_i ($pBest_i = x_i$). La actualización de posición y velocidad de cada partícula es modificada constantemente por cada iteración realizada y matemáticamente el algoritmo se describe de la siguiente forma:

2.2.3 Actualización de vector velocidad

$$v_i(t+1) = w * v_i(t) + c_1 * r_1\big(pBest_i(t) - x_i(t)\big) + c_2 * r_2\big(gBest_i - x_i(t)\big) \tag{2.4}$$

Donde:

- $v_i(t)$: Actualización de Velocidad de la partícula
- w: Factor de inercia
- c_1 , c_2 : Constantes de aceleración
- r_1 , r_2 : Parámetros aleatorios (0 y 1)

En la Ecuación 2.5, el factor de inercia suele ser asignado por el programador de acuerdo al nivel de experticia y si el algoritmo tiende a tener una rápida aceleración es necesario establecer una condición de velocidad máxima declarando una variable v_{max} tomando en cuenta que no debe ser demasiado alto ni tan corto debido a que no suelen converger. Por otra parte los valores de $c_1 \, y \, c_2$ son necesarios para el movimiento de la partícula donde mediante estudios, se ha establecido que $c_{1,2}$ puede tomar un valor máximo de 4, entonces, $c_1 + c_2 = 4$ donde no es necesario que ambas variables sean iguales (Del Valle, 2008).

Los parámetros aleatorios $r_1 \, y \, r_2$ o también conocidos como parámetros de aceleración estocásticos vienen definidos por dos valores aleatorios que pueden ser 0

y 1, esto permite un comportamiento más realista al enjambre creado logrando simular de mejor manera las actitudes sociales de las aves y peces, al igual que los valores de v_i y x_i son actualizados en cada iteración (Yang, 2019).

2.2.4 Actualización de vector posición

Con la Ecuación 2.6 se actualiza el vector posición luego de haber actualizado la velocidad de cada partícula.

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.5}$$

Una vez actualizada la posición se evalúa la siguiente condición ($x_i < pBest_i$), y si la cumple, el valor de x_i pasa a ser memorizado en $pBest_i$ ($pBest_i = x_i$), esto se actualiza constantemente hasta cumplir con la condición de paro general ($pBest_i < gBest_i$) actualizando el mejor $gBest_i$ de salida.

2.2.5 Diagrama de flujo y algoritmo

El diagrama de flujo representado en la Figura 2.2 expresa las características más importantes para el funcionamiento del PSO, se obvia subprocesos debido a la extensión del programa.

Figura 2.2 Diagrama de flujo de PSO

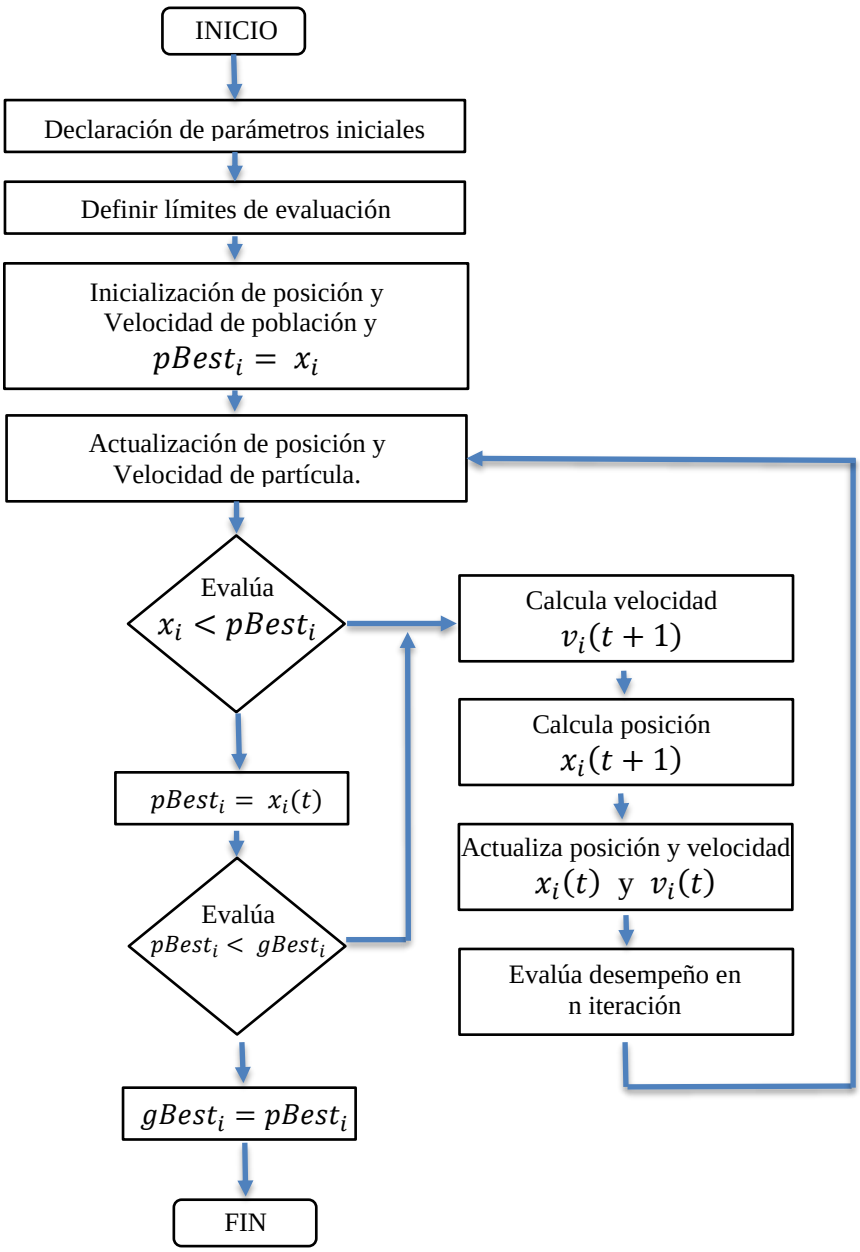
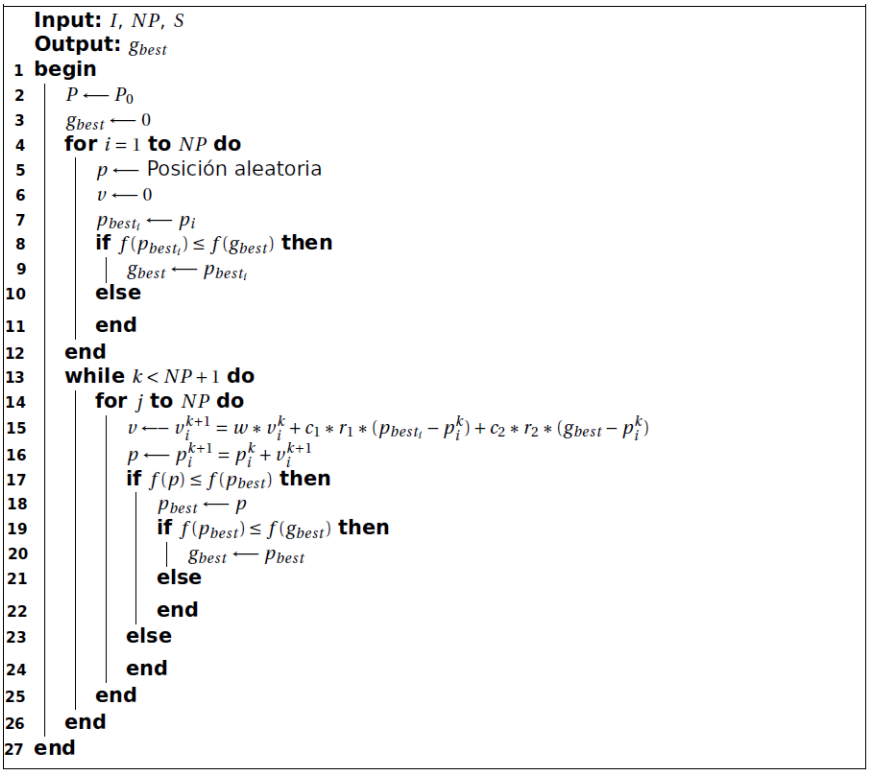


Diagrama de flujo de PSO, Elaborado por: Darwin Torres

Establecido el diagrama de flujo del PSO, en la Figura 2.3 se presenta un modelo base del algoritmo que se utiliza para desarrollar la optimización por enjambre de partículas.

Figura 2.3 Algoritmo de PSO

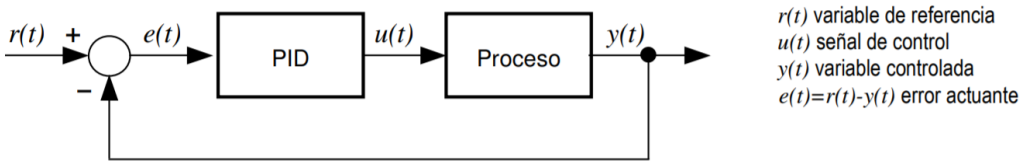


Algoritmo de PSO, Fuente: (Martínez, 2018)

2.3 Controlador PID

Un control PID es un mecanismo que se utiliza comúnmente en sistemas industriales por su alta fiabilidad debido a que calcula el error $e(t)$ entre el valor deseado y el valor medido. El algoritmo de este tipo de control utiliza 3 parámetros fundamentales para su desempeño siendo estas una acción proporcional, integral y derivativa. La interacción de estas 3 acciones permite ajustar el proceso por medio de uno o varios elementos de control en un valor deseado, como por ejemplo temperatura exacta en un horno, posicionamientos de válvulas, potencias en calentadores, y más (Coba, 2015).

Figura 2.4 Controlador PID



Esquema básico de un controlador PID, Fuente: (Coba, 2005)

En definitiva la ganancia proporcional depende del error actual, el tiempo de integración de los errores pasados y el tiempo de derivación en los errores a futuro.

Expresando matemáticamente el PID para la Figura 2.4 se obtiene:

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{d}{dt}e(t)\right] \tag{2.6}$$

Y para la Ecuación 2.8 de error se tiene:

$$e(t) = y_{ref}(t) - y(t) \tag{2.7}$$

2.3.1 Acción proporcional

La acción proporcional tiene la capacidad de responder al error de manera inmediata con la desventaja de acumular oscilaciones manera gradual si se tiene un error relativamente pequeño, está representada por la Ecuación 2.9.

$$u(t) = K_p e(t) \tag{2.8}$$

2.3.2 Acción integral

En esta acción el error se va acercando a cero, haciendo que error de salida sea lo más cercana al valor de referencia en el transcurso del tiempo.

$$u(t) = \frac{K_p}{T_i}\int_0^t e(\tau)d\tau \tag{2.9}$$

2.3.3 Acción derivativa

Por último al agregar esta función, permite que el controlador actúe antes de que se presente el error como tal, sin que existan cambios considerables en el sistema, en definitiva resulta ser como una acción de predicción de errores.

$$u(t) = K_p T_d\frac{d}{dt}e(t) \tag{2.10}$$

2.4 Sistemas ball and plate

Hoy en día existe una amplia gama de sistemas ball and plate tanto comerciales como artesanales (creados por estudiantes y universidades) los cuales permiten crear, analizar e implementar diferentes tipos de controles con la finalidad de desarrollar avances tecnológicos (Araujo, 2018).

2.4.1 Sistema comercial

La empresa Feedback Instruments realiza diferentes prototipos para analizar sistemas de teoría de control, uno de ellos es el sistema ball and plate de la Figura 2.5 que funciona con la plataforma NI LabView, básicamente permite controlar una esfera en una superficie plana y mantener su posición además tiene la capacidad de visualizar la esfera mediante una cámara web USB (Feedback Instruments).

Figura 2.5 Sistema ball and plate

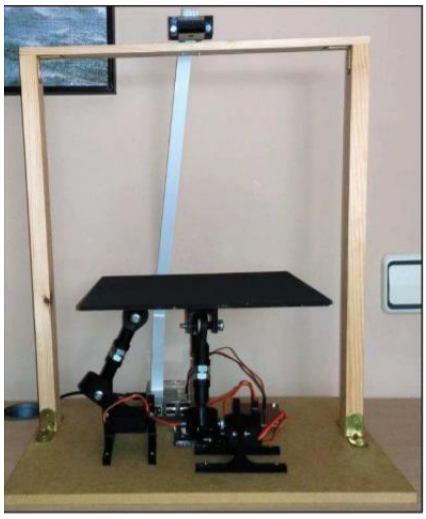
Sistema de comercialización ball and plate, Fuente: https://www.leybold-shop.com/ball-and-plate-system-33-240.html

2.4.2 Sistema no comercial

Por otra parte existen prototipos realizados por estudiantes, universidades y personas en general que desarrollan maneras eficientes para su funcionamiento mediante

mecanismos ingeniosos que permiten el movimiento de la superficie en los dos ejes principales para su equilibrio como se observa en la Figura 2.6.

Figura 2.6 Sistema no comercial ball and plate



Sistema no comercial Ball and Plate, Fuente: (Sánchez, 2017)

2.5 Visión artificial

La visión artificial es una emulación de la capacidad visual de los seres vivos para entender o interpretar la imagen capturada y así poder desenvolverse en su entorno, debido a esto se ha incorporado en diferentes áreas con gran rapidez como en la biología, medicina, robótica, agricultura, seguridad y la industria (Davies, 2012).

Para interpretar una imagen capturada de un mundo tridimensional a uno bidimensional (Fu y González, 1988) es necesario seguir con estos pasos:

- Captura
- Pre-procesamiento
- Segmentación
- Descripción
- Reconocimiento
- Interpretación

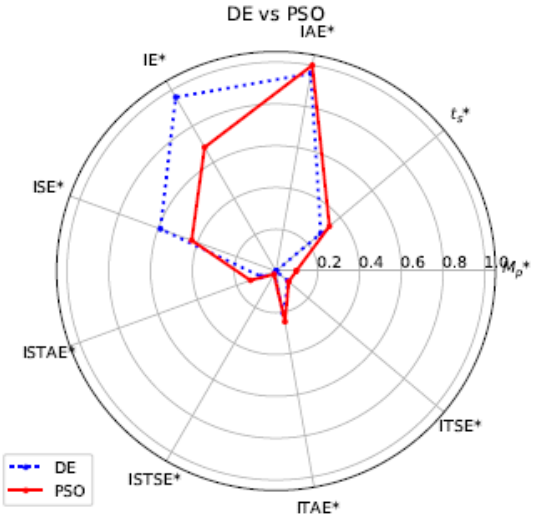
Con la utilización de técnicas adecuadas no es necesario seguir toda la secuencia, debido a que algunas imágenes resultan ser más complejas que otras, después de haber obtenido, procesado y analizado la información adquirida, el algoritmo realiza una toma de decisiones en base a las imágenes digitales generadas.

2.6 Proyectos similares

2.6.1 Aplicación 1

El proyecto denominado ALGORITMOS BIOINSPIRADOS PARA LA SINTONIZACION DE SISTEMAS DE CONTROL CON RETARDO fue desarrollado por el Ing. René Martínez en México en el año 2018, en el documento señala los problemas que existen al controlar un sistema con variables retardadas que en la mayoría de casos la sintonización no se puede modelar, debido a esto, se ha implementado métodos meta heurísticos como son los sistemas computacionales bio-inspirados.

Figura 2.7 Comparación entre DE y PSO



Comparación de resultado promedio entre DE y PSO, Fuente: (Martínez, 2018)

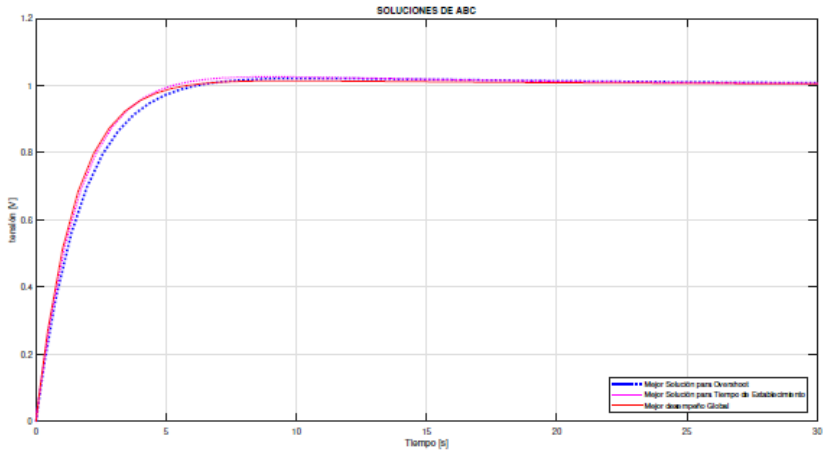
Martínez en su investigación llegó a la conclusión que la implementación de algoritmos meta heurísticos establecen una optimización de sintonización para los

controladores PID, además, según la Figura 2.7 existe una ventaja en DE en ciertas características con respecto al PSO.

2.6.2 Aplicación 2

El proyecto DISEÑO E IMPLEMENTACION DE CONTROLADORES DE TEMPERATURA APLICANDO LOS ALGORITMOS DE OPTIMIZACIÓN BIOINSPIRADOS COLONIA DE ABEJAS ARTIFICIALES Y ENJAMBRE DE PARTÍCULAS, fue desarrollado por la Ing. Fernanda Bisarrea de la universidad de las fuerzas armadas en el año 2019. En él explica los exitosos resultados de la implementación de los algoritmos bio-inspirados en controladores de temperatura, en el trabajo investigativo propone el uso y diseño de un PID mediante algoritmos de enjambre de partículas y colonia de abejas artificiales.

Figura 2.8 Respuesta de GA



Respuesta del controlador obtenido mediante GA, Fuente: (Bisarrea, 2019)

En la Figura 2.8 se muestra la salida correspondiente al control donde se puede apreciar una salida ligeramente amortiguada, demostrando así que la utilidad del algoritmo bio-inspirado.

2.6.3 Aplicación 3

El trabajo realizado por el Ing. Diego Pacheco, SINTONIZACIÓN DE LOS PARÁMETROS DE UN CONTROLADOR FUZZY LOGIC, BASADA EN EL

ALGORITMO DE OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS UTILIZADO EN EL SISTEMA DE GESTIÓN ENERGÉTICA DE UNA MICRORED ELECTROTÉRMICA. Desarrollado en la universidad de las fuerzas armadas en el año 2019, detalla el incremento a nivel mundial de la demanda energética, y para afrontar este problema nace la iniciativa de micro redes MG.

El objetivo era reducir los costos de facturación y minimizar el perfil de red intercambiado con la red eléctrica, todo esto aplicando algoritmos de optimización por enjambre de partículas para la sintonización de los parámetros de un controlador con lógica difusa (Pacheco, 2019).

Figura 2.9 Diagrama de flujo de PSO implementado

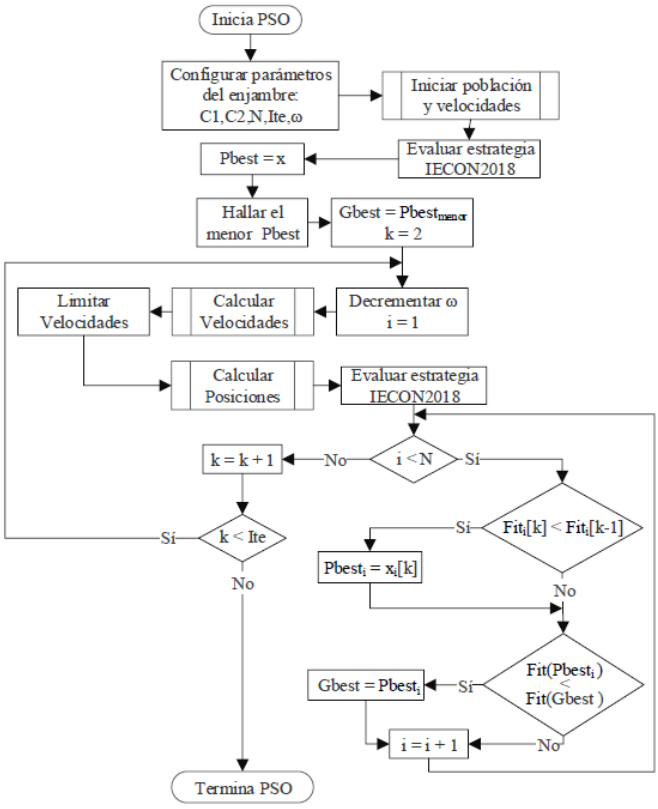


Diagrama de flujo de PSO implementado, Fuente: (Pacheco, 2019)

El algoritmo PSO de la Figura 2.9, según Pacheco consiguió mejorar los parámetros de calidad, además, el tiempo de convergencia del PSO obtuvo una mejora de 69.96 horas, es decir un menor tiempo para el resultado aceptable.

CAPITULO 3

DESARROLLO

En el presente capítulo se detalla la construcción de un sistema ball and plate en un ambiente virtual para simular el funcionamiento de un algoritmo PSO, Se han aplicado métodos y formas no tradicionales para el funcionamiento de visión artificial, las cuales se precisan de forma detallada su algoritmo.

3.1 Funcionamiento

El objetivo del proyecto es encontrar los valores de los coeficientes proporcional, integral y derivativo de forma automática usando PSO por lo que al ejecutar la clase principal inmediatamente se envía un host local donde se encuentra la plataforma, esfera (con posición aleatoria), contenedores de datos, y los botones principales para usar el programa. Al presionar el botón “usar PSO” el algoritmo comienza a encontrar la mejor solución que se acerque al set point deseado, después los valores son colocados automáticamente en los contenedores de datos del PID, esto permite controlar el movimiento y estabilización de la esfera, la cual su posición esta realimentada con la visión artificial incorporada.

3.2 Montaje virtual

El sistema ball and plate esta simulado en python, por lo que se detalla el código necesario para la simulación del mismo dando lugar a un sistema interactivo e intuitivo para uso educativo.

Para el funcionamiento del programa se utiliza una clase principal y 3 subclases que permiten realizar el PSO, PID y el procesamiento de imágenes simultáneamente. Para ello se utilizan varias librerías en python que permiten la interacción usuario-maquina mediante un host local.

3.2.1 Librería bootstrap

Bootstrap es una librería que permite el diseño de entornos web en el cual se puede agregar todo tipo contenidos como ingresar texto, sliders, tablas, advertencias, etc. Existen plantillas sencillas para aplicaciones básicas así como también ambientes más complejos de paga.

3.2.2 Canvas

Canvas es una herramienta que se utiliza para la creación de la plataforma en el cual la esfera tendrá el movimiento en el eje cartesiano. Una vez descargada e instalada la librería antes mencionada se procede a ingresar a la clase html para crear la plataforma con la siguiente línea de código:

```
[...]
<div class="col-md-6"><canvas id="lienzo" width="500" height="500"
                    style="border: 1px solid blanchedalmond; cursor:
pointer;">Su navegador no soporta canvas :( </canvas></div>
 [...]
```

Donde el canvas adopta el nombre de “lienzo” el cual posee un ancho y alto de 500 con color sólido “blanchedalmond”.

3.2.3 Etiquetas y entradas de parámetros PID

Para generar las etiquetas de los parámetros del PID así como también los contenedores donde se representaran los datos obtenidos del PSO se utiliza lo siguiente:

```
[...]
<form id="formulario_x">
      <div class="form-group">
         <label for="kpx">Coef. Proporcional [Kp]</label>
         <input type="number" class="form-control" id="kpx"
step="0.000000001" min="0" value="0">
      </div>
      <div class="form-group">
          <label for="kix">Coef. Integral [Ki]</label>
          <input type="number" class="form-control" id="kix"
step="0.000000001" min="0" value="0">
          <small id="kix_e" class="form-text text-muted">0</small>
       </div>
       <div class="form-group">
           <label for="kdx">Coef. Derivativo [Kd]</label>
           <input type="number" class="form-control" id="kdx"
step="0.000000001" min="0" value="0">
           <small id="kdx_e" class="form-text text-muted">0</small>
       </div>
[...]
```

Se utilizan 3 contenedores de datos para representar los valores del coeficiente proporcional, integral y derivativo, así como también sus respectivas etiquetas, tanto del eje X como del eje Y. Los contenedores son de tipo entrada de datos numérico ya que una vez que el PSO calcule los valores, estos son colocados de forma automática para el funcionamiento del PID, además tiene la funcionalidad de ingresar de forma manual cada valor del coeficiente para ver el comportamiento de cada parámetro en el PID. El valor mínimo a colocar es cero en todos los casos.

3.2.4 Botón actualizar

Tanto en los formularios X, Y se agrega un botón de actualización de datos, este permite ingresar el valor de los contenedores de los coeficientes a la clase de PID

```
[...]
<form id="formulario_x">
<button type="submit" class="btn btn-primary">Actualizar</button>
</form>
 [...]
```

3.2.5 Botón PSO y reiniciar

Se coloca el botón “Utilizar PSO” que permite llamar a la clase PSO y ejecutarla para que posteriormente ingrese los valores en los contenedores explicados en el punto

3.2.3. Y el botón reiniciar permite reestablecer todos los valores iniciales para permitir ejecutar el programa por otra ocasión.

```
[...]
<div class="row text-center mb-2 mt-4">
    <div class="col-md-12">
    <a href="#" onclick="PSO(); return false;" class="btn btn-
info">Utilizar PSO</a>
    <a href="#" onclick="Reiniciar(); return false;" class="btn btn-
success">Reiniciar</a>
     </div>
</div>
 [...]
```

3.3 Algoritmo de optimización de enjambre de partículas (PSO)

Este algoritmo permite encontrar mínimos y máximos para establecer la respuesta más cercana a la deseada, usando el comportamiento natural de ciertas especies, haciéndola muy fiable y rápida para encontrar la mejor solución a problemas complejos del mundo real.

3.3.1 Subclase partícula

En esta subclase se inicializan las variables necesarias para el funcionamiento del PSO, así como la implementación de métodos que permiten la evaluación y actualización de la posición y actualización de la velocidad de la partícula.

En el método “init” se empieza con la inicialización de todas las variables que se va a utilizar

```
[...]
class Particle:

    def __init__(self, limites, numero_variables, opcion,
fitness_inicial):
        self.limites = limites
        self.opcion = opcion
        self.posicion_particula = []
        self.velocidad_particula = []
        self.mejor_posicion_local_particula = []
        self.fitness_mejor_posicion_local_particula =
fitness_inicial
        self.fitness_posicion_particula = fitness_inicial
        self.numero_variables = numero_variables
 [...]
```

En el mismo método se incorporan la función random para inicializar la posición y la velocidad de la partícula de forma aleatoria comprendida entre límites de -1 a 1, posteriormente se guardan en las variables posición_partícula y velocidad_partícula respectivamente.

```
[...]
for i in range(self.numero_variables):
            self.posicion_particula.append(random.uniform(self.limit
es[i][0], self.limites[i][1]))
            self.velocidad_particula.append(random.uniform(-1, 1))
 [...]
```

El algoritmo PSO tiene la capacidad de encontrar soluciones mínimas o máximas de acuerdo al problema planteado, para desarrollar este proyecto se utiliza la opción de mínimos ya que nos permite converger a una sola respuesta siendo esta la más aproximada a la solución requerida. Para encontrar los mínimos se utiliza el siguiente algoritmo en la función evaluar, siendo -1 la opción que permite hallar lo mencionado.

```
[...]
def evaluar(self, funcion):
        self.fitness_posicion_particula =
funcion(self.posicion_particula)
        if self.opcion == -1:
            if self.fitness_posicion_particula <
self.fitness_mejor_posicion_local_particula:
                self.mejor_posicion_local_particula =
self.posicion_particula
                self.fitness_mejor_posicion_local_particula =
self.fitness_posicion_particula
[...]
```

En el método actualizar_velocidad se crea dos variables randomicas que permite generar los valores de aceleración de las partículas, para r1 y r2 los valores deben estar delimitados entre 0 y 1, la función random permite crear por defecto estos límites.

```
[...]
def actualizar_velocidad(self, mejor_posicion_global, parametros):
        for i in range(self.numero_variables):
            r1 = random.random()
            r2 = random.random()
[...]
```

Continuando con el método se tiene los parámetros w, c1 y c2 los cuales son introducidos por el programador de acuerdo a la experticia del mismo. Y descomponiendo la Ecuación 3.1 se tiene:

$$v_i(t+1) = w * v_i(t) + c_1 * r_1\big(pBest_i(t) - x_i(t)\big) + c_2 * r_2\big(gBest_i - x_i(t)\big) \quad (3.1)$$

$$Velocidad\ Cognitiva = c_1 * r_1\big(pBest_i(t) - x_i(t)\big) \quad (3.2)$$

$$Velocidad\ Social = c_2 * r_2\big(gBest_i - x_i(t)\big) \quad (3.3)$$

$$Velocidad\ particula = w * velocidad\ particula + velocidad\ cognitiva + velocidad\ social \quad (3.4)$$

```
[...]
w, c1, c2 = parametros

            velocidad_cognitiva = c1 * r1 *
(self.mejor_posicion_local_particula[i] -
self.posicion_particula[i])
            velocidad_social = c2 * r2 * (mejor_posicion_global[i] -
self.posicion_particula[i])
            self.velocidad_particula[i] = w *
self.velocidad_particula[i] + velocidad_cognitiva + velocidad_social
 [...]
```

Una vez encontrada la velocidad se prosigue en la actualización de la posición de la partícula, para ello se utiliza la Ecuación 3.5.

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (3.5)$$

```
[...]
def actualizar_posicion(self):
        for i in range(self.numero_variables):
            self.posicion_particula[i] = self.posicion_particula[i]
+ self.velocidad_particula[i]
 [...]
```

En el mismo método se establece condiciones para que la posición de la partícula no exceda los límites entre 0 y 1, donde si es menor a 0 tomara el valor de cero y si es mayor a 1 tomara el valor de uno.

```
[...]
if self.posicion_particula[i] > self.limites[i][1]:
                self.posicion_particula[i] = self.limites[i][1]
if self.posicion_particula[i] < self.limites[i][0]:
                self.posicion_particula[i] = self.limites[i][0]
 [...]
```

3.3.2 Subclase PSO

Después de haber establecido los parámetros fundamentales como la posición y velocidad de cada partícula, así como también el comportamiento social. Ahora es indispensable establecer el número de partículas y las iteraciones que se necesiten para encontrar el resultado deseado en la función ingresada. Para ello se ingresa un nuevo método con nuevas variables.

```
[...]
class PSO:

    def __init__(self, funcion, limites, numero_particulas,
iteraciones, parametros, opcion = -1, numero_variables = 2):
        self.funcion = funcion
        self.limites = limites
        self.numero_particulas = numero_particulas
        self.iteraciones = iteraciones
        self.numero_variables = numero_variables
        self.__opcion = opción
        self.__parametros = parametros
        self.__fitness_mejor_posicion_global_particula =
self.fitness_inicial
        self.__mejor_posicion_global_particula = []
 [...]
```

En este método se crea un nuevo vector “enjambre_particulas” para posteriormente crear una lista con los mejores resultados, además de dos bucles; número de partículas e iteraciones que realiza el programa usando la opción de mínimos.

```
[...]
 def evaluar(self):
        enjambre_particulas = []
        for i in range(self.numero_particulas):
            enjambre_particulas.append(Particle(self.limites,
self.numero_variables, self.__opcion, self.fitness_inicial))
        A = []
 [...]
```

En el segundo método se establece el rango de las iteraciones para que puedan ser evaluadas en la función dada.

```
[...]
for i in range(self.iteraciones):
            for j in range(self.numero_particulas):
                enjambre_particulas[j].evaluar(self.funcion)
                if self.__opcion == -1:
                    if
enjambre_particulas[j].fitness_posicion_particula <
self.__fitness_mejor_posicion_global_particula:
                        self.__mejor_posicion_global_particula =
list(enjambre_particulas[j].posicion_particula)
                        self.__fitness_mejor_posicion_global_particu
la = float(enjambre_particulas[j].fitness_posicion_particula)
 [...]
```

Y por último se actualiza la mejor velocidad y posición de la partícula guardándola como “mejor_posicion_global_particula”

```
[...]
for j in range(self.numero_particulas):
                enjambre_particulas[j].actualizar_velocidad(self.__m
ejor_posicion_global_particula, self.__parametros)
                enjambre_particulas[j].actualizar_posicion()

            A.append(self.__fitness_mejor_posicion_global_particula)
 [...]
```

3.4 Algoritmo de control (PID)

Para lograr el objetivo es necesario contar con un control en lazo cerrado para alcanzar el estado de salida deseado. Para esto se inicializa las variables necesarias en la clase PID detalladas en las siguientes líneas de código.

```
[...]
class PID:
    def __init__(self, Kp, Ki, Kd, tau, lim_min, lim_max):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.tau = tau
        self.lim_min = lim_min
        self.lim_max = lim_max
        self.integrator = 0.0
        self.differentiator = 0.0
        self.prev_error = 0.0
        self.prev_measurement = 0.0

    def set_gains(self, Kp, Ki, Kd, tau):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.tau = tau
        self.differentiator = 0
[...]
```

Para la ejecución del algoritmo PID se crea un método de operación del mismo, en este se comienza a realizar la diferencia entre set point y el valor medido para conocer el error producido en ese instante de tiempo partiendo de la Ecuación 2.8.

```
[...]
def compute(self, setpoint, measurement, T):
        # Error
        error = setpoint - measurement
[...]
```

Obtenido el error, se prosigue a multiplicarlo por la variable Kp la cual será enviada por la clase PSO.

```
[...]
# Proporcional
        proportional = self.Kp * error
[...]
```

Como la integral del error es la acumulación del error en el tiempo, se multiplica la suma del error actual y el error anterior con la constante integral en función del tiempo T. Se añade el producto de la mitad del tiempo debido a que todo el programa se ejecuta

con retardo de 0.5 del tiempo en general para visualizar de mejor manera el funcionamiento del programa.

```
[...]
# Integral
        self.integrator = self.integrator + 0.5 * self.Ki * T *
(error + self.prev_error)
 [...]
```

Es necesario colocar límites mínimos y máximos de acuerdo al tamaño de la superficie, se evalúa de acuerdo a la variable proporcional, esto para que las oscilaciones no vayan a exceder el tamaño de la superficie.

```
[...]
# Limites de la integral
        limMinInt = 0
        limMaxInt = 0
        if (self.lim_max > proportional):
            limMaxInt = self.lim_max - proportional
        else:
            limMaxInt = 0

        if (limMinInt < proportional):
            limMinInt = self.lim_min -proportional
        else:
            limMinInt = 0
 [...]
```

Para limitar la integral se evalúa la variable “Integrator” con respecto a los nuevos límites internos “limMaxInt” y “limMinInt” para que no superen dichos valores.

```
[...]
 # Limitar la integral
        if (self.integrator > limMaxInt):
            self.integrator = limMaxInt
        elif (self.integrator < limMinInt):
            self.integrator = limMinInt
[...]
```

Para la derivada se evalúa la tasa de cambio del error con respecto al tiempo, para lo cual se resta la medida de la posición de la bola con la medida anterior, se multiplica con el coeficiente derivativo y se divide con respecto al tiempo con un porcentaje de amortiguamiento.

```
[...]
# Derivativa con filtro
        self.differentiator = (2.0 * self.Kd * (measurement -
self.prev_measurement) + (2.0 * self.tau - T) * self.differentiator)
/ (2.0 * self.tau + T)
 [...]
```

Para terminar se suma las 3 operaciones (proporcional, integral y derivativa) para obtener la salida deseada.

```
[...]
# Calcular salida del PID
        pid = proportional + self.integrator + self.differentiator
        return pid

 [...]
```

3.5 Algoritmo de visión artificial

Para determinar la posición de la esfera en el plano 2D de la simulación es necesario obtener imágenes por algún medio, sin embargo para este proyecto debido a que la luz se refleja de varias fuentes no es posible usar una cámara web, para esto se ha desarrollado un plano secundario donde se dibuja a la par con el plano de la simulación, guardándolo como imagen dentro de los archivos del proyecto.

Dentro de la clase OpenCv se define el tamaño del plano que tiene dimensiones 500x500, la funciones ones permite crear el fondo de color negro. Seguido dibujamos la esfera con los valores X, Y derivados de la clase principal, donde se almacenan las posiciones en tiempo real del programa. Y por último con la función imwrite se guarda el dibujo con el nombre de image.png para ser procesada después.

```
[...]
class OpenCV:
    def image(x, y):
        x= int(x) + 250
        y= int(y) + 250
        img = np.ones((500,500,1),np.uint8)*0
        cv2.circle(img, (x, y), 10, (255, 0, 0), -1)
        cv2.imwrite('image.png', img)
 [...]
```

En la misma clase pero en el método analiza se llama a la imagen copia del programa para convertirlo en escala de grises mediante la función cvtColor.

```
[...]
def analiza():
        frame = cv2.imread('image.png')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        img = cv2.medianBlur(gray, 5)
[...]
```

En el mismo método se halla la esfera en el plano mediante la función HoughCircles, la cual mediante la imagen transformada a escala de grises y parámetros de gradiente, radio, y distancia puede localizar la esfera dándonos como resultado un vector de posición en x, y.

```
[...]
 circles = cv2.HoughCircles(image=img, method=cv2.HOUGH_GRADIENT,
dp=0.9, minDist=2, param1=10, param2=10, minRadius=1, maxRadius=15)
 [...]
```

Para finalizar la obtención de la posición de la esfera, se calcula el centro de la esfera con el comando uint16. Retornando el valor de Center para la ejecución del programa.

```
[...]
center = (0, 0)
        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
                center = (i[0], i[1])
        return center
[...]
```

3.6 Algoritmo de ejecución principal o clase madre

Esta clase permite ejecutar todas las subclases detalladas anteriormente, las cuales serán llamadas en ciertos métodos para resolver el objetivo principal. El primer método define la función con la que trabaja el sistema, se ha colocado 3 tipos de paraboloides que representan el movimiento tentativo de sistemas de ball and plate, en esta simulación se puede representar cualquier modelo matemático creado por otros proyectos de ball and plate.

```
[...]
def definir_funcion(O):
    x = O[0]
    y = O[1]
    #z = (x) ** 2 +  (y) ** 2
    #z = (y - x) ** 2 + (y - x ** 2) ** 2
    z = (x - y**2) ** 2 + (y - x ** 2) ** 2
    return z
[...]
```

Se inserta el set point deseado, como el tamaño del plano es de 500 x 500 se coloca el valor de 250 en el vector para estabilizar la esfera en el centro de la superficie. Además, se llama a la función random para la posición, esto permite inicializar la posición de la esfera en cualquier parte del plano de forma aleatoria.

```
[...]
set_point = [250, 250]
position = randomicos()
[...]
```

Para el control PID del proyecto definimos 4 tipos de control, dos para el eje x y dos para el eje y, siendo un total de 4 PID que intervienen en el programa debido a que cada eje tiene parte positiva y parte negativa con respecto el punto central del plano.

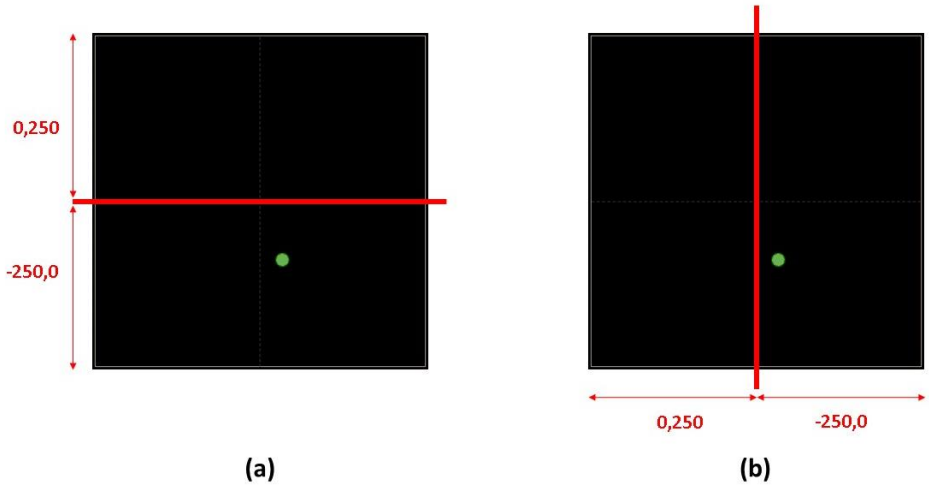
```
[...]
pidx = [0, 0, 0] # pid para control en el eje x
pidy = [0, 0, 0] # pid para control en el eje y

valoresx = []
valoresy = []

control_xa = PID(pidx[0], pidx[1], pidx[2], 0.1, 0, 250)
control_xb = PID(pidx[0], pidx[1], pidx[2], 0.1, -250, 0)
control_ya = PID(pidy[0], pidy[1], pidy[2], 0.1, 0, 250)
control_yb = PID(pidy[0], pidy[1], pidy[2], 0.1, -250, 0)
[...]
```

Como se puede apreciar en la Figura 3.1(a) la parte superior de la línea roja representa al control_xa con límites de 0 a 250 y para el control_xb se representa en la parte inferior de la Figura 3.1(a), la misma lógica pertenece para el eje de la coordenadas Y teniendo su control_ya y control_yb.

Figura 3.1 Referencia superficie Ball and Plate



Sistema de referencia de plano Ball and Plate, Elaborado por: Darwin Torres

Una vez inicializado los vectores pidx y pidy se lleva a cabo la introducción de datos para su funcionamiento, en la primera forma el usuario puede ingresar valores de forma manual para observar el comportamiento del PID, y la segunda forma se ingresa de manera automática mediante la ejecución del algoritmo PSO cuyo objetivo es encontrar los parámetros más óptimos para el correcto funcionamiento del PID en el sistema ball and plate.

En el método setPID se toma las variables de los contenedores creados en el punto 3.2.3, los cuales los cuales serán almacenados en los vectores pidx y pidy para su uso en las variables control xa, xb, ya y yb.

```
[...]
def setPID():
       pidx[0] = float(request.form['kpx'])
    pidx[1] = float(request.form['kix'])
    pidx[2] = float(request.form['kdx'])

    pidy[0] = float(request.form['kpy'])
    pidy[1] = float(request.form['kiy'])
    pidy[2] = float(request.form['kdy'])

    control_xa.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_xb.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_ya.set_gains(pidy[0], pidy[1], pidy[2], 0.1)
    control_yb.set_gains(pidy[0], pidy[1], pidy[2], 0.1)
[...]
```

En el método setPSO llama a la subclase PSO para encontrar los valores necesarios para el PID usando los parámetros función, límites, número de partículas, iteraciones, parámetros y opción=-1.

```
[...]
def setPSO():
    pidx[0] = abs(PSO(definir_funcion, [(-1, 1), (0.5, 0.988)], 12,
100, [0.2, 1, 2]).evaluar()[0])
    pidx[1] = abs(PSO(definir_funcion, [(-1, 1), (0.01, 0.00001)],
12, 100, [0.2, 1, 2]).evaluar()[0])
    pidx[2] = abs(PSO(definir_funcion, [(-1, 1), (-0.01, 0.00001)],
12, 100, [0.5, 1, 2]).evaluar()[0])

    pidy[0] = abs(PSO(definir_funcion, [(-1, 1), (0.3, 0.988)], 12,
100, [0.12, 1, 2]).evaluar()[0])
    pidy[1] = abs(PSO(definir_funcion, [(-1, 1), (0.01, 0.0001)],
12, 100, [0.2, 1, 2]).evaluar()[0])
    pidy[2] = abs(PSO(definir_funcion, [(-1, 1), (-0.01, 0.00001)],
12, 100, [0.5, 1, 2]).evaluar()[0])

    control_xa.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_xb.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_ya.set_gains(pidy[0], pidy[1], pidy[2], 0.1)
    control_yb.set_gains(pidy[0], pidy[1], pidy[2], 0.1)

[...]
```

Como se puede apreciar se ingresan los parámetros pertinentes para la ejecución del programa; para la función se llama al método definir_funcion que se introdujo anteriormente, los limites se encuentran valores entre -1 y 1 con decimales desde 0.001 hasta 0.00001, trabaja con 12 partículas en iteraciones y se ingresa los valores de w, c1 y c2 donde la suma c1+c2 de no debe ser mayor a 4 y tampoco es necesario que sean valores iguales, se ha colocado 1 y 2 respectivamente.

```
[...]
# (class) PSO(funcion, limites, numero_particulas, iteraciones,
parametros, opcion=-1, numero_variables=2)
PSO(definir_funcion, [(-1, 1), (0.5, 0.988)], 12, 100, [0.2, 1,
2]).evaluar()[0])
[...]
```

Para controlar la esfera en el plano se relaciona la posición random inicial con el error calculado (set point – posición de visión artificial) en cada instante del tiempo, en el método control se utiliza la función time para el tiempo, y se llama al método analiza

de la subclase opencv para obtener los valores de posx y posy, almacenándolos en valores x, y respectivamente.

```
[...]
def control():

    while (True):
        if (calcular == 1):
            tiempo = time.time()
            canvas.image(position[0], position[1])
            posx, posy = canvas.analiza()

            valoresx.append(posx)
            valoresy.append(posy)
[...]
```

En el mismo método, se utiliza otro método llamado compute de la subclase PID para calcular el error generado entre la posición aleatoria y los datos obtenidos por la visión artificial tanto para el eje X como para el eje Y.

```
[...]
if ((posx != 0) and (posy != 0)):
        error_xa = control_xa.compute(set_point[0], posx, muestreo)
        error_xb = control_xb.compute(set_point[0], posx, muestreo)
        error_ya = control_ya.compute(set_point[1], posy, muestreo)
        error_yb = control_yb.compute(set_point[1], posy, muestreo)
 [...]
```

Por último se adiciona la posición actual más el error calculado para mover la esfera en la dirección deseada, caso contrario de surgir algún error la esfera tendrá una posición aleatoria nuevamente.

```
[...]
position[0] += error_xa
                position[0] += error_xb
                position[1] += error_ya
                position[1] += error_yb
            else:
                position = randomicos()
            muestreo = time.time() - tiempo
 [...]
```

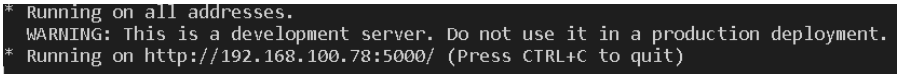
CAPÍTULO 4

PRUEBAS Y RESULTADOS

4.1 Procedimiento para la utilización del simulador ball and plate

1. Se ejecuta la clase principal del programa realizado en Python.
2. Ya inicializado el programa se indica la dirección IP a la que se debe acceder indicado en la Figura 4.1.

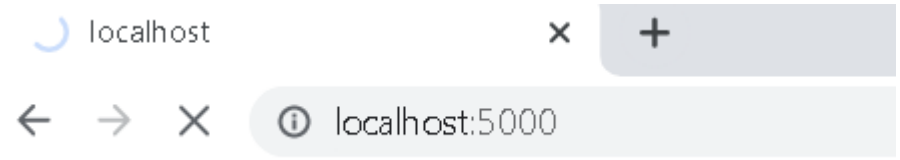
Figura 4.1 Dirección del Host local



Elaborado por: Darwin Torres

3. Como se observa en la Figura 4.2 se abre el navegador de preferencia y se coloca la dirección IP anterior o también se puede acceder colocando localhost:5000 en el buscador.

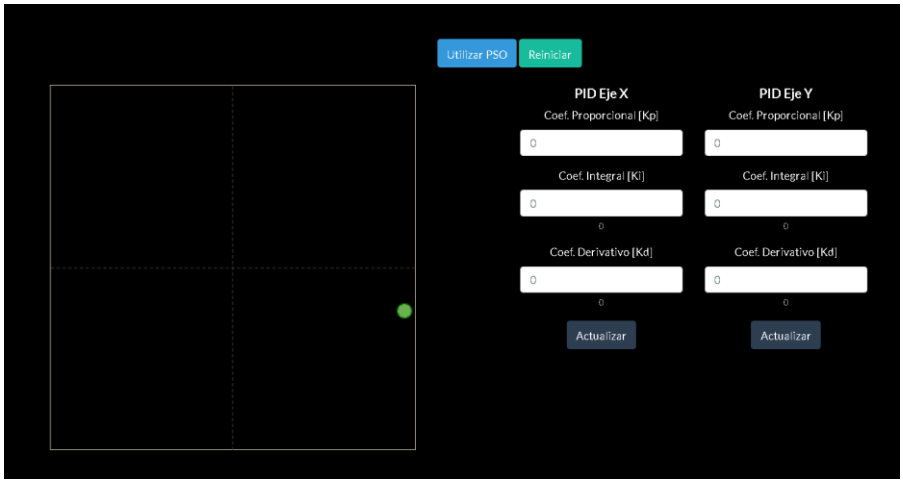
Figura 4.2 Búsqueda del host local



Elaborado por: Darwin Torres

4. Al cargar la dirección mencionada en el punto anterior, se despliega la interfaz de la Figura 4.3, la bola de color verde siempre tendrá una posición diferente cada vez que se ejecute el programa.

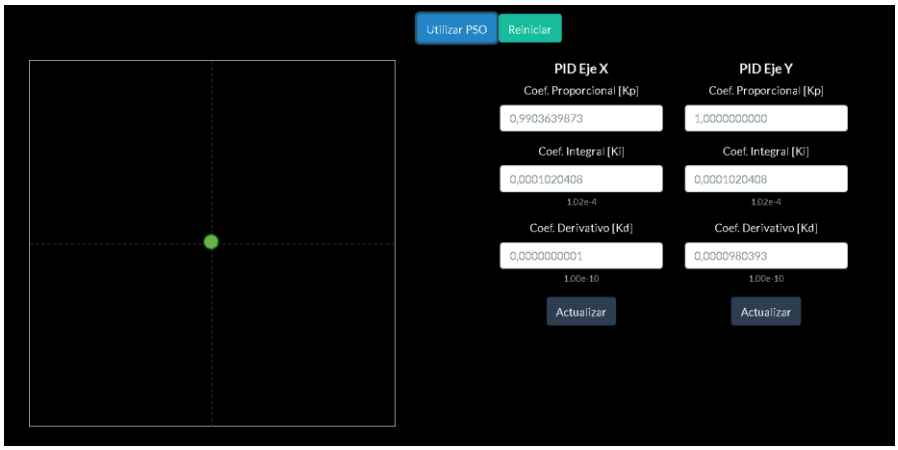
Figura 4.3 Interfaz gráfica



Elaborado por: Darwin Torres

5. Para comenzar a utilizar el PSO, se debe presionar el botón “Utilizar PSO” y automáticamente encontrará los valores del PID y se desplegaran en los contenedores vacíos. Simultáneamente la bola comenzará a moverse hasta llevar al punto de set point como se aprecia en la Figura 4.4.

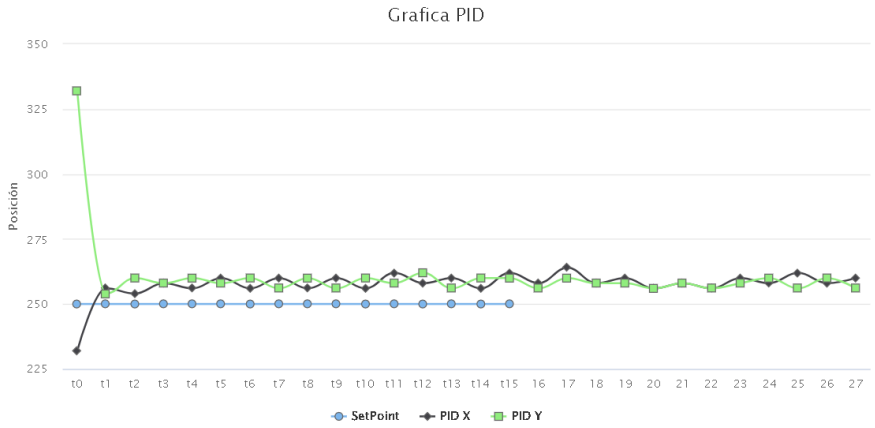
Figura 4.4 Uso de PSO en interfaz gráfica



Elaborado por: Darwin Torres

6. Luego de unos 5 segundos se despliega una gráfica indicado en la Figura 4.5 que representa la posición Vs tiempo de ambos controladores en el eje X y Y.

Figura 4.5 Gráfica de estabilización



Elaborado por: Darwin Torres

7. Si se presiona el botón “Reiniciar”, la bola se coloca en otra posición aleatoria reiniciando el programa desde cero como se muestra en la Figura 4.6.

Figura 4.6 Uso de reinicio

Uso de reinicio en Interfaz gráfica, Elaborado por: Darwin Torres

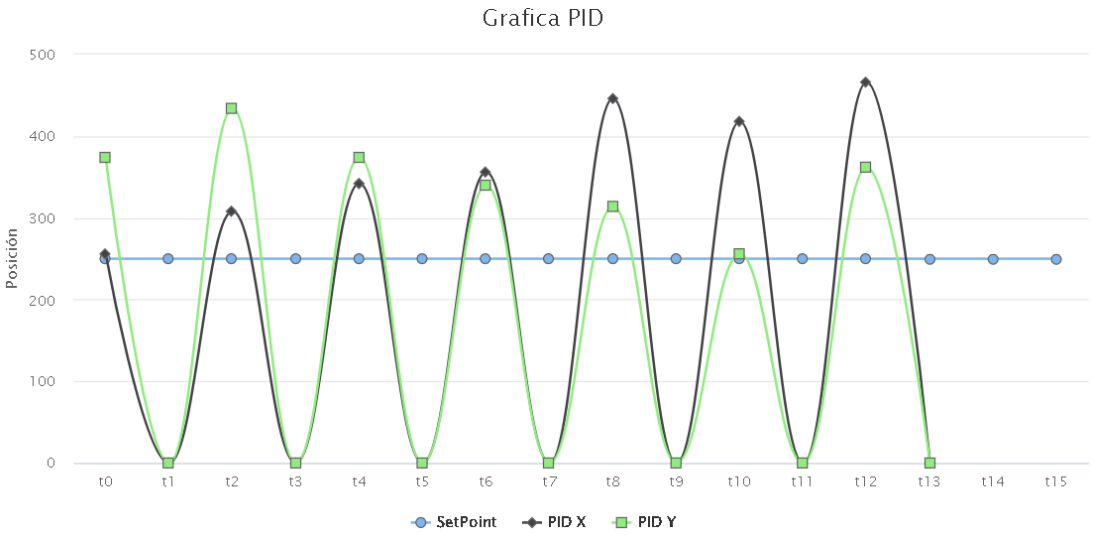
8. En la interfaz se puede introducir valores por el usuario de tal manera se puede estudiar el comportamiento de una sola variable en el programa. Como ejemplo se ha colocado un valor en el coeficiente Kp del eje X indicado en la Figura 4.7.

Figura 4.7 Datos manuales

Ingreso de datos manuales en Interfaz gráfica, Elaborado por: Darwin Torres

9. En el mismo ejemplo se observa que la bola no se estabiliza como se indica en la Figura 4.8 por lo que no tiene sus coeficientes completos para su buen funcionamiento, en la gráfica se representa una oscilación que tiende a incrementar su error en ambas salidas.

Figura 4.8 Desestabilización PID



Elaborado por: Darwin Torres

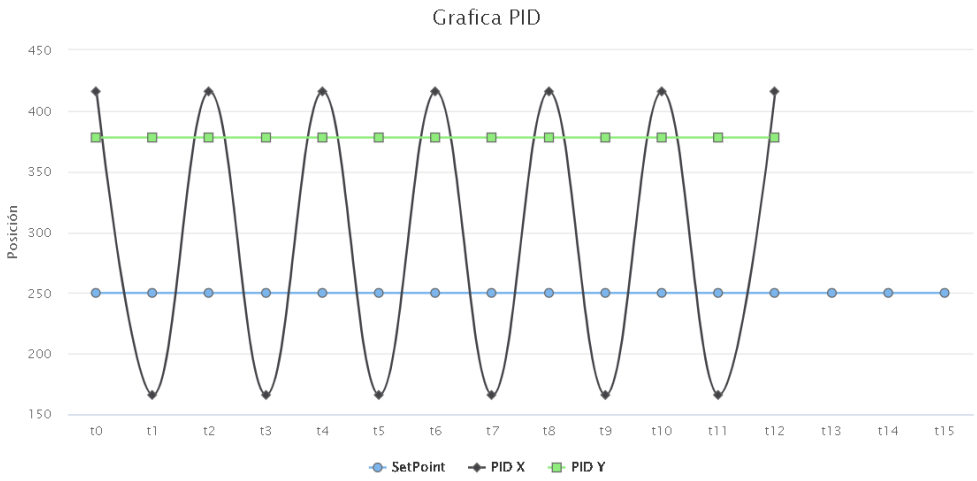
4.2 Comparación entre método PSO y Ziegler-Nichols

Para realizar la comparativa de la integral del error absoluto entre ambos métodos es necesario conocer el tiempo de estabilización del PID en Ziegler-Nichols, además se necesita calcular los valores de los coeficientes e introducirlos de manera manual en el programa para hallar su gráfica y realizar el análisis.

4.2.1 Método de Ziegler-Nichols

En el programa se ingresa un valor en el coeficiente proporcional en el que las oscilaciones sean estables, la constante I, D deben ser anuladas para lograr este procedimiento. En este caso se ha encontrado un valor de 9 para la ganancia critica (Ku) del eje X, como se observa en la Figura 4.9.

Figura 4.9 Oscilación estable de PID



Elaborado por: Darwin Torres

Se usa la Tabla 4.1 de Ziegler-Nichols en lazo cerrado para encontrar la constante proporcional, el tiempo integral y derivativo.

Tabla 4.1 Tabla de Ziegler-Nichols

Control	Kp	τi	τd
PID	$0.6Ku$	$0.5Pu$	$0.125Pu$

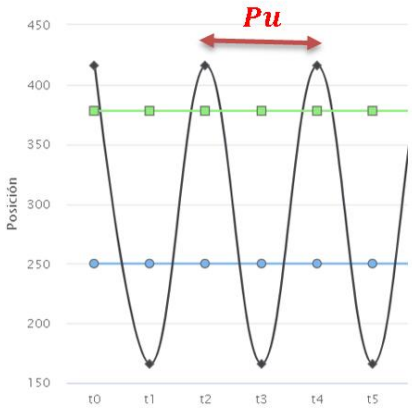
Elaborado por: Darwin Torres

Donde:

- Kp es la constante proporcional
- τi es el tiempo integral
- τd es el tiempo derivativo
- Ku es la ganancia critica
- Pu es el periodo critico

Partiendo de la Figura 4.9 se mide el tiempo que tarda una oscilación en repetirse como se muestra en la Figura 4.10.

Figura 4.10 Medición de datos



Elaborado por: Darwin Torres

Como se puede observar en la Figura 4.10 el tiempo aproximado de Pu es de 1.9s y reemplazando los valores de $Ku\ y\ Pu$ en la Tabla 4.1, se obtiene la Tabla 4.2 con los valores definidos de $Kp, \tau i\ y\ \tau d$.

Tabla 4.2 Tabla de tiempo integral y derivativo

Control	Kp	τi	τd
PID	5.4	0.95	0.2375

Elaborado por: Darwin Torres

Partiendo de la Ecuación (2.7) se tiene:

$$G(s) = Kp\left[1 + \frac{1}{T_i s} + T_d s\right]$$

Donde:

- $Kp = Kp$
- $Ki = \frac{Kp}{Ti}$
- $Kp = Kp(Td)$

Obteniendo los valores finales para la sintonización del PID.

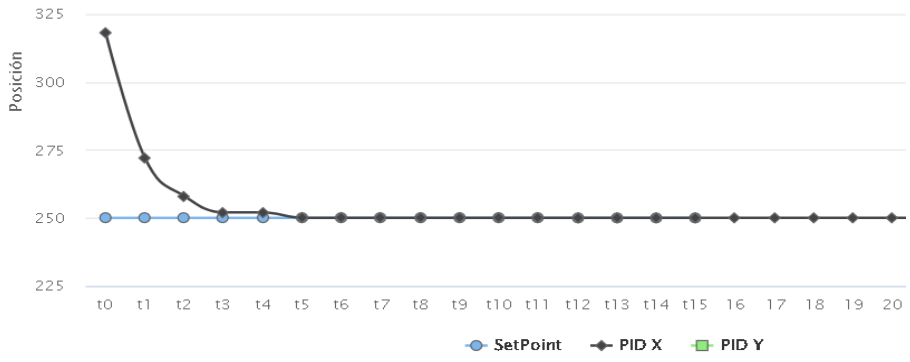
Tabla 4.3 Tabla de coeficientes de PID

Control	Kp	Ki	Kd
PID	5.4	5.68	1.28

Elaborado por: Darwin Torres

Al ingresar los datos de la Tabla 4.3 en el programa de forma manual se obtiene la Figura 4.11, donde el segundo 5 ya se observa una estabilización con un valor de 253 en la posición del plano aproximadamente.

Figura 4.11 Comportamiento de PID con Ziegler-Nichols



Elaborado por: Darwin Torres

4.2.2 Cálculo de error en Ziegler-Nichols

La integral del error absoluto (IEA) es utilizada para medir la cantidad del error que se produce en algún sistema, para lo cual se tiene:

$$Error = Sp - Pv \tag{4.1}$$

$$IEA = \int_0^{Tp} |Error| dt \tag{4.2}$$

Donde:

- Tp es el tiempo aproximado de estabilización
- Sp es el Set Point
- Pv es el valor más probable medido

Partiendo del análisis de la Figura 4.11 se obtiene que:

$$Tp = 5s$$

$$Sp = 250$$

$$Pv = 253$$

Calculando el error con la Ecuación 4.1 se tiene:

$$Error = Sp - Pv$$

$$Error = 250 - 253$$

$$Error = -3$$

$$IEA = \int_0^{Tp} |Error| dt$$

$$IEA = \int_0^{5} |-3| dt$$

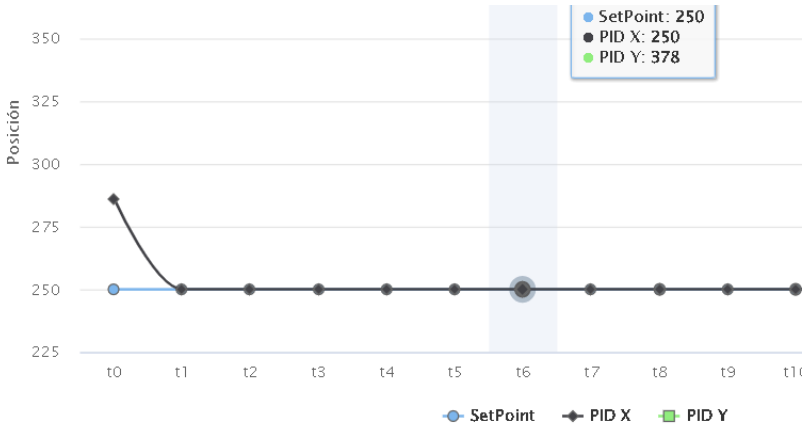
$$IEA = 3t|_0^5$$

$$IEA = (3 * 5) = 15$$

4.2.3 Cálculo de error del PID usando PSO

Para encontrar los datos necesarios para el cálculo del error es necesario ejecutar el programa usando PSO, automáticamente halla los datos de los coeficientes que mejor estabilizan la bola en el plano, teniendo como resultado:

Figura 4.12 Comportamiento de PID con PSO



Elaborado por: Darwin Torres

Partiendo del análisis de la Figura 4.12 se obtiene que:

$$Tp = 1s$$

$$Sp = 250$$

$$Pv = 251$$

Calculando el error con la Ecuación 4.1 se tiene:

$$Error = Sp - Pv$$

$$Error = 250 - 251$$

$$Error = -1$$

$$IEA = \int_0^{Tp} |Error| dt$$

$$IEA = \int_0^1 |-1| dt$$

$$IEA = 1t|_0^1$$

$$IEA = (1 * 1) = 1$$

Como se puede observar en la Figura 4.11 y la Figura 4.12 a simple vista el tiempo de estabilización del PID usando PSO es 5 veces menor que el PID con Ziegler-Nichols y el error es 3 veces menor respectivamente, se tiene los siguientes resultados en las Tablas 4.4 y 4.5.

Tabla 4.4 Tabla de cálculo de error

Set Point	Ziegler-Nichols	PSO
250	253	251
Error	-3	-1
% Error	1.2%	0.4%

Elaborado por: Darwin Torres

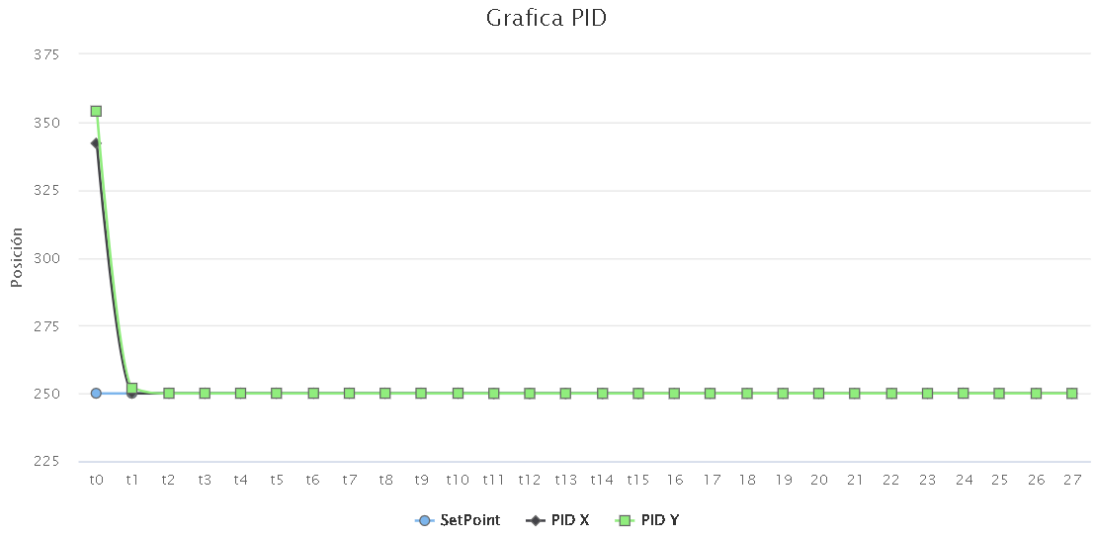
Tabla 4.5 Tabla de error absoluto

IEA		
	Ziegler-Nichols	PSO
Tp	5s	1s
error	-3	-1
IEA	15	1

Elaborado por: Darwin Torres

El análisis del error se realizó solo en el eje X en ambos casos, debido a que la gráfica resultante del PID en el eje Y es prácticamente la misma como se puede apreciar en la Figura 4.13, dando así los mismos resultados en los cálculos de la integral del error absoluto.

Figura 4.13 Comportamiento del PID con ambos ejes



Elaborado por: Darwin Torres

CAPÍTULO 5

CONCLUSIONES

Se cumplió de manera exitosa con los objetivos de este proyecto al implementar el algoritmo PSO en un control PID sobre un sistema ball and plate simulado con una interfaz amigable para su funcionamiento, permitiendo realizar nuevos análisis y estudios de nuevos sistemas que permiten encontrar soluciones apropiadas en un menor tiempo posible.

Se caracterizó de manera satisfactoria todas las entradas y salidas de datos que usa el programa y la relación que existe entre ellos para su funcionamiento en la auto-sintonización del PID.

Se implementó la visión artificial como método de realimentación de posición de la bola para el controlador PID mediante el procesamiento de imágenes almacenadas de la captura del plano en el host local.

Se comprobó que la visión artificial resulta ser muy útil para identificar objetos sin importar del material del que esté constituido, sin embargo para la adquisición de imágenes es necesario mantener un entorno apropiado para su correcto funcionamiento ya que con cualquier perturbación o interferencia los datos de posición serán erróneos.

Se logró implementar el algoritmo bio-inspirado para la sintonización de las variables de un controlador PID, permitiendo estudiar y desarrollar análisis comparativo entre el comportamiento, resultado, variables de datos de éste algoritmo con el método de sintonización tradicional de Ziegler-Nichols.

Se realizó pruebas de simulación de un sistema Ball and Plate en 2D, así como del algoritmo PSO y las perturbaciones que el usuario puede añadir al programa, dando como resultado la correcta estabilización de la bola sobre el plano en el set point deseado.

El uso del algoritmo de optimización de enjambre de partículas (PSO) como método para encontrar los coeficientes de sintonización de un controlador PID, demostró mejor desempeño con 5 veces en tiempo de estabilización que el método tradicional de Ziegler-Nichols, trabajando en el mismo proceso bajo las mismas condiciones de equilibrio.

El algoritmo genético bio-inspirado realizado, además de mejorar sus tiempos de estabilización, demostró también tener un porcentaje de error mucho menor que el método tradicional de Ziegler-Nichols, siendo de 0.4 y 1.2% respectivamente. Lo que abre las puertas al uso de este algoritmo a sistemas más avanzados de control.

Los parámetros de w, c1 y c2 que se necesitan para ajustar el algoritmo PSO han sido escogidos de manera empírica para la función del problema ya que no existe manera matemática de obtener un resultado concreto, sin embargo se ha realizado pruebas con otras funciones teniendo excelentes resultados en el momento de la estabilización, demostrando que el algoritmo puede sintonizar varios controles PID con diferentes procesos.

Al analizar los resultados obtenidos del tiempo aproximado de estabilización de la Figura 4.13 en los dos ejes tiende a ser casi instantáneo, estabilizándose en 1 segundo debido a que en la simulación el movimiento representaría a una superficie ideal, existe un margen de error que no puede cuantificarse como lo es en un sistema real, sin embargo, bajo las mismas circunstancias el error es tres veces menor usando PSO que la sintonización por el método tradicional de Ziegler-Nichols.

RECOMENDACIONES

Pese a que los valores de c1 y c2 funcionaron con diferentes funciones ingresadas en el sistema, se podría realizar un análisis más profundo sobre la utilización de estos parámetros para observar las incidencias de los mismos en el algoritmo generado.

En la interfaz creada, se podría crear más contenedores con un panel más robusto donde de se pueda ingresar el set point deseado, sin necesidad de cambiar el código de programación.

Se debe tomar en cuenta que la subclase PSO es utilizada de manera simultánea en 6 líneas de código por lo que se recomienda no exceder el número de iteraciones en la ejecución de la subclase PSO ya que el programa tiende a consumir más recursos ralentizando el procesamiento de imágenes.

Para la realimentación de la posición de la bola en la superficie se puede usar algún otro sistema de adquisición de imágenes como es el caso de cámaras web, a pesar de ello, es necesario establecer un entorno de iluminación idóneo para que no existan brillos que afecten el procesamiento de las imágenes obtenidas.

Se recomienda reiniciar el programa mediante dos formas; la primera usando el botón reiniciar y la segunda actualizando la página web, debido a que los datos del PSO se quedan almacenados en la subclase PID.

Es necesario convertir la imagen obtenida en escala de grises para realizar el análisis respectivo para encontrar la bola en la superficie, además es indispensable coordinar los parámetros necesarios como el diámetro y el punto central de la localización de la bola en la imagen.

Se debe tomar en cuenta que en la interfaz al momento de colocar un punto central este no equivale a 0,0 en el eje de las coordenadas ya que el canvas dibuja la superficie de izquierda a derecha y de arriba hacia abajo, por lo que el punto central equivale a 250,250.

Se recomienda analizar qué tipo de función siendo máximos o mínimos se requiere utilizar para el problema a resolver dado el inconveniente que nunca llega a converger en una sola respuesta cuando la opción es incorrecta.

No existe un número relacionado para colocar las partículas necesarias y el número de iteraciones que interactúen de manera óptima para obtener el resultado, se recomienda empezar a usar cantidades pequeñas e ir subiendo de a poco hasta alcanzar una respuesta favorable.

El factor de inercia colocado en el algoritmo de PSO, tiene diferentes valores en cada uno de los ejes, ya que en la simulación no se puede apreciar con claridad la influencia del mismo se podría realizar un análisis comparativo entre limites de factor de inercia y así brindar un respaldo al programador de que valor aproximado debería utilizar.

REFERENCIAS

Araujo, A. (2018). *Diseño y control de un sistema Ball and Beam con realimentación visual con Raspberry-Pi,* Sevilla. Obtenido de: http://bibing.us.es/proyectos/abreproy/5936/fichero/PFC-5936-ARAUJO.pdf

Coba, W. (Octubre, 2015). *Control PID,* Obtenido de: http://www.frlr.utn.edu.ar/archivos/alumnos/electronica/catedras/38-sistemas-de-control-aplicado/Publicaciones/Control_PID_Enfoque_Descriptivo.pdf

Cova, J. (Diciembre, 2005). *Control PID un enfoque descriptivo,* Obtenido de: http://www.frlr.utn.edu.ar/archivos/alumnos/electronica/catedras/38-sistemas-de-control-aplicado/Publicaciones/Control_PID_Enfoque_Descriptivo.pdf

García, F. (Mayo, 2019). *Inteligencia Artificial. Una perspectiva desde la ficción a la realidad,* Obtenido de Grial: https://repositorio.grial.eu/bitstream/grial/1623/1/Ficcion.pdf

Yang, X. S. (Febrero, 2014). *Nature-inspired optimization algorithms,* Obtenido de: https://www.sciencedirect.com/science/article/pii/B9780124167438000166

Martínez, A. y Cosío, M. (Agosto, 2016). *Algoritmos inspirados en la naturaleza para solucionar problemas difíciles,* Sociedad Mexicana de Inteligencia Artificial, Obtenido de: https://www.researchgate.net/publication/312940534_Algoritmos_inspirados_en_la_naturaleza_para_solucionar_problemas_dificiles

Martínez, R. (2018). *Algoritmos bioinspirados para la sintonización de sistemas de control con retardo,* Baja California, México, Obtenido de: http://cicese.repositorioinstitucional.mx/jspui/handle/1007/2523

Bisarrea, F. (2019). *Diseño e implementación de controladores de temperatura aplicando los algoritmos de optimización bio-inspirados colonia de abejas artificiales*

y enjambre de partículas, Sangolqui, Obtenido de: http://repositorio.espe.edu.ec/xmlui/handle/21000/20815?show=full

Eberhart, R. y Kennedy, J. (1995). *Particle Swarm Optimization,* Australia, Obtenido de: https://ieeexplore.ieee.org/document/488968

Eberhart, R. y Kennedy, J. (2001). *Swarm Intelligence,* Australia, Obtenido de: https://www.elsevier.com/books/swarm-intelligence/eberhart/978-1-55860-595-4

Shi, Y. y Eberhart, R. (1999). *Empirical study of particle swarm optimization.* Washinngton Dc, EEUU. Obtenido de: https://ieeexplore.ieee.org/document/785511

Del Valle, Y., Veneyagamoorthy, G., Mohagheghi, J. y Harley, R. (Abril, 2008). *Particle Swarm Optimization: Basic concepts, Variants and Applications in Power Systems.* Obtenido de: https://ieeexplore.ieee.org/document/4358769

Yang, N., Dey, S. y Xin, -S. (2019). *Springer Tracts in Nature-Inspired Computing*. Obtenido de: https://www.springer.com/series/16134

Sánchez, D. (Mayo, 2017). *Plataformas de bajo coste para experimentación en control,* ESAII. Obtenido de: https://upcommons.upc.edu/bitstream/handle/2117/109547/Plataformas%20de%20bajo%20coste%20para%20experimentaci%C3%B3n%20en%20control.pdf?sequence=1&isAllowed=y

Davies, E. (2012). *Computer and machine visión,* USA. Obtenido de: https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Computer%20and%20Machine%20Vision_%20Theory%2C%20Algorithms%2C%20Practicalities%20%284th%20ed.%29%20%5BDavies%202012-03-19%5D.pdf

Fu, K. y González, R. (1998). *Robótica: Control, detección, visión e inteligencia,* México. Obtenido de: https://dialnet.unirioja.es/servlet/libro?codigo=246791

ANEXOS

Anexo 1. Algoritmo de clase principal de ejecución

```
import time, json, math, threading, random
from flask import Flask, render_template, request, Response
from libs.pid import PID
from libs.pso import PSO
from libs.opencv import OpenCV

app = Flask(__name__)

def randomicos():
    return [math.floor(random.random()*250),
math.floor(random.random()*250)]

def definir_funcion(O):
    x = O[0]
    y = O[1]

    #z = (x) ** 2 +  (y) ** 2
    #z = (y - x) ** 2 + (y - x ** 2) ** 2
    z = (x - y**2) ** 2 + (y - x ** 2) ** 2
    return z


set_point = [250, 250] # posicion central en canvas x, y (width 500,
height 500)
position = randomicos()

pidx = [0, 0, 0] # pid para control en el eje x
pidy = [0, 0, 0] # pid para control en el eje y

valoresx = []
valoresy = []

control_xa = PID(pidx[0], pidx[1], pidx[2], 0.1, 0, 250)
control_xb = PID(pidx[0], pidx[1], pidx[2], 0.1, -250, 0)
control_ya = PID(pidy[0], pidy[1], pidy[2], 0.1, 0, 250)
control_yb = PID(pidy[0], pidy[1], pidy[2], 0.1, -250, 0)

calcular = 0

canvas = OpenCV

@app.route('/')
def inicio():
    return render_template('index.html')
```

```
@app.route('/reset')
def reset():
    global position
    global pidx
    global pidy
    global calcular
    global valoresx
    global valoresy

    position = randomicos()

    pidx = [0, 0, 0] # pid para control en el eje x
    pidy = [0, 0, 0] # pid para control en el eje y
    valoresx = []
    valoresy = []
    calcular = 0

    return json.dumps({'status':'OK'})

@app.route('/setPos', methods=['POST'])
def setPos():
    global valoresy
    global valoresx
    global calcular

    position[0] = float(request.form['X'])
    position[1] = float(request.form['Y'])
    valoresx = []
    valoresy = []
    calcular = 0

    print("[INFO] Posicion ("+str(position[0])+", "+str(position[1])+")
modificada por usuario")
    return json.dumps({'status':'OK'})

@app.route('/setPID', methods=['POST'])
def setPID():
    global control_xa
    global control_xb
    global control_ya
    global control_yb
    global valoresx
    global valoresy
    global calcular

    pidx[0] = float(request.form['kpx'])
    pidx[1] = float(request.form['kix'])
    pidx[2] = float(request.form['kdx'])
```

```
    pidy[0] = float(request.form['kpy'])
    pidy[1] = float(request.form['kiy'])
    pidy[2] = float(request.form['kdy'])

    control_xa.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_xb.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_ya.set_gains(pidy[0], pidy[1], pidy[2], 0.1)
    control_yb.set_gains(pidy[0], pidy[1], pidy[2], 0.1)

    valoresx = []
    valoresy = []

    calcular = 1

    print("[INFO] PID modificada por usuario")
    return json.dumps({'status':'OK'})

@app.route('/setPSO', methods=['POST'])
def setPSO():
    global control_xa
    global control_xb
    global control_ya



    global control_yb
    global valoresx
    global valoresy

    pidx[0] = abs(PSO(definir_funcion, [(-1, 1), (0.5, 0.988)], 12,
100, [0.2, 1, 2]).evaluar()[0])
    pidx[1] = abs(PSO(definir_funcion, [(-1, 1), (0.01, 0.00001)], 12,
100, [0.2, 1, 2]).evaluar()[0])
    pidx[2] = abs(PSO(definir_funcion, [(-1, 1), (-0.01, 0.00001)], 12,
100, [0.5, 1, 2]).evaluar()[0])

    pidy[0] = abs(PSO(definir_funcion, [(-1, 1), (0.3, 0.988)], 12,
100, [0.12, 1, 2]).evaluar()[0])
    pidy[1] = abs(PSO(definir_funcion, [(-1, 1), (0.01, 0.0001)], 12,
100, [0.2, 1, 2]).evaluar()[0])
    pidy[2] = abs(PSO(definir_funcion, [(-1, 1), (-0.01, 0.00001)], 12,
100, [0.5, 1, 2]).evaluar()[0])

    control_xa.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_xb.set_gains(pidx[0], pidx[1], pidx[2], 0.1)
    control_ya.set_gains(pidy[0], pidy[1], pidy[2], 0.1)
    control_yb.set_gains(pidy[0], pidy[1], pidy[2], 0.1)

    valoresx = []
```

```
    valoresy = []

    print("[INFO] PSO calculando valores...")
    return json.dumps({'pidx' : pidx, 'pidy' : pidy, 'status':'OK'})

@app.route('/getPos')
def getPos():
    global position
    global valoresx
    global valoresy

    datosx = ""
    for x in valoresx:
        valor = str(x)
        datosx += valor + "#"

    datosy = ""
    for y in valoresy:
        valor = str(y)
        datosy += valor + "#"
    return json.dumps({'position' : position, 'valoresx': datosx,
'valoresy': datosy, 'status':'OK'})

def control():
    global control_xa
    global control_xb
    global control_ya
    global control_yb
    global position
    global set_point
    global image
    global valoresx
    global valoresy

    muestreo = 0

    while (True):
        if (calcular == 1):
            tiempo = time.time()
            canvas.image(position[0], position[1])
            posx, posy = canvas.analiza()
            # print(posx, posy)
            valoresx.append(posx)
            valoresy.append(posy)
            # print(valoresx, valoresy)

            if ((posx != 0) and (posy != 0)):
                error_xa = control_xa.compute(set_point[0], posx,
muestreo)
```

```
                error_xb = control_xb.compute(set_point[0], posx,
muestreo)

                error_ya = control_ya.compute(set_point[1], posy,
muestreo)
                error_yb = control_yb.compute(set_point[1], posy,
muestreo)

                position[0] += error_xa
                position[0] += error_xb
                position[1] += error_ya
                position[1] += error_yb
            else:
                position = randomicos()
            muestreo = time.time() - tiempo

        time.sleep(0.5)

resp = threading.Thread(target = control)
resp.daemon = True
resp.start()

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=False)
```

Anexo 2. Algoritmo de control PID

```
import time
import matplotlib.pyplot as plt

class PID:

    # Inicializacion
    def __init__(self, Kp, Ki, Kd, tau, lim_min, lim_max):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.tau = tau
        self.lim_min = lim_min
        self.lim_max = lim_max
        self.integrator = 0.0
        self.differentiator = 0.0
        self.prev_error = 0.0
        self.prev_measurement = 0.0

    def set_gains(self, Kp, Ki, Kd, tau):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
```

```
        self.tau = tau
        self.differentiator = 0

    def get_gains(self):
        return [self.Kp, self.Ki, self.Kd, self.tau]

    def map(self, x, in_min, in_max, out_min, out_max):
        return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
out_min

    def set_limits(self, lim_min, lim_max):
        self.lim_min = lim_min
        self.lim_max = lim_max

    def compute(self, setpoint, measurement, T):
        # Error
        error = setpoint - measurement
        # Proporcional
        proportional = self.Kp * error
        # Integral
        self.integrator = self.integrator + 0.5 * self.Ki * T * (error
+ self.prev_error)
        # Limites de la integral
        limMinInt = 0
        limMaxInt = 0
        if (self.lim_max > proportional):
            limMaxInt = self.lim_max - proportional
        else:
            limMaxInt = 0

        if (limMinInt < proportional):
            limMinInt = self.lim_min -proportional
        else:
            limMinInt = 0
        # Limitar la integral
        if (self.integrator > limMaxInt):
            self.integrator = limMaxInt
        elif (self.integrator < limMinInt):
            self.integrator = limMinInt
        # Derivativa con filtro
        self.differentiator = (2.0 * self.Kd * (measurement -
self.prev_measurement) + (2.0 * self.tau - T) * self.differentiator) /
(2.0 * self.tau + T)
        # Calcular salida del PID y limitar salida
        pid = proportional + self.integrator + self.differentiator
        if (pid > self.lim_max):
            pid = self.lim_max
        elif (pid < self.lim_min):
            pid = self.lim_min
```

```
        self.prev_error = error
        self.prev_measurement = measurement

        return pid
```

Anexo 3. Algoritmo PSO

```
import random
import time, math

class Particle:

    def __init__(self, limites, numero_variables, opcion,
fitness_inicial):
        self.limites = limites
        self.opcion = opcion
        self.posicion_particula = []  # particle position
        self.velocidad_particula = []  # particle velocity
        self.mejor_posicion_local_particula = []  # best position of
the particle
        self.fitness_mejor_posicion_local_particula =
fitness_inicial  # initial objective function value of the best
particle position
        self.fitness_posicion_particula = fitness_inicial  # objective
function value of the particle position
        self.numero_variables = numero_variables

        for i in range(self.numero_variables):
            self.posicion_particula.append(random.uniform(self.limites[
i][0], self.limites[i][1]))  # generate random initial position
            self.velocidad_particula.append(random.uniform(-1, 1))  #
generate random initial velocity

    def evaluar(self, funcion):
        self.fitness_posicion_particula =
funcion(self.posicion_particula)
        if self.opcion == -1:
            if self.fitness_posicion_particula <
self.fitness_mejor_posicion_local_particula:
                self.mejor_posicion_local_particula =
self.posicion_particula  # update the local best
                self.fitness_mejor_posicion_local_particula =
self.fitness_posicion_particula  # update the fitness of the local best
        if self.opcion == 1:
            if self.fitness_posicion_particula >
self.fitness_mejor_posicion_local_particula:
                self.mejor_posicion_local_particula =
self.posicion_particula  # update the local best
```

```
                self.fitness_mejor_posicion_local_particula =
self.fitness_posicion_particula  # update the fitness of the local best

    def actualizar_velocidad(self, mejor_posicion_global, parametros):
        for i in range(self.numero_variables):
            r1 = random.random()
            r2 = random.random()

            w, c1, c2 = parametros

            velocidad_cognitiva = c1 * r1 *
(self.mejor_posicion_local_particula[i] - self.posicion_particula[i])
            velocidad_social = c2 * r2 * (mejor_posicion_global[i] -
self.posicion_particula[i])
            self.velocidad_particula[i] = w *
self.velocidad_particula[i] + velocidad_cognitiva + velocidad_social

    def actualizar_posicion(self):
        for i in range(self.numero_variables):
            self.posicion_particula[i] = self.posicion_particula[i] +
self.velocidad_particula[i]

            # check and repair to satisfy the upper self.limites
            if self.posicion_particula[i] > self.limites[i][1]:
                self.posicion_particula[i] = self.limites[i][1]
            # check and repair to satisfy the lower self.limites
            if self.posicion_particula[i] < self.limites[i][0]:
                self.posicion_particula[i] = self.limites[i][0]

class PSO:

    def __init__(self, funcion, limites, numero_particulas,
iteraciones, parametros, opcion = -1, numero_variables = 2):
        self.funcion = funcion
        self.limites = limites
        self.numero_particulas = numero_particulas
        self.iteraciones = iteraciones
        self.numero_variables = numero_variables
        self.__opcion = opcion # -1 miniza funcion - 1 maximiza funcion
        self.__parametros = parametros

        self.fitness_inicial = None
        if (self.__opcion == -1):
            self.fitness_inicial = float("inf")
        if (self.__opcion == 1):
            self.fitness_inicial = -float("inf")

        self.__fitness_mejor_posicion_global_particula =
self.fitness_inicial
```

```
        self.__mejor_posicion_global_particula = []


    def evaluar(self):
        enjambre_particulas = []
        for i in range(self.numero_particulas):
            enjambre_particulas.append(Particle(self.limites,
self.numero_variables, self.__opcion, self.fitness_inicial))
        A = []

        for i in range(self.iteraciones):
            for j in range(self.numero_particulas):
                enjambre_particulas[j].evaluar(self.funcion)

                if self.__opcion == -1:
                    if
enjambre_particulas[j].fitness_posicion_particula <
self.__fitness_mejor_posicion_global_particula:
                        self.__mejor_posicion_global_particula =
list(enjambre_particulas[j].posicion_particula)
                        self.__fitness_mejor_posicion_global_particula
= float(enjambre_particulas[j].fitness_posicion_particula)
                if self.__opcion == 1:
                    if
enjambre_particulas[j].fitness_posicion_particula >
self.__fitness_mejor_posicion_global_particula:
                        self.__mejor_posicion_global_particula =
list(enjambre_particulas[j].posicion_particula)
                        self.__fitness_mejor_posicion_global_particula
= float(enjambre_particulas[j].fitness_posicion_particula)
            for j in range(self.numero_particulas):
                enjambre_particulas[j].actualizar_velocidad(self.__mejo
r_posicion_global_particula, self.__parametros)
                enjambre_particulas[j].actualizar_posicion()

            A.append(self.__fitness_mejor_posicion_global_particula)  #
record the best fitness

        return self.__mejor_posicion_global_particula
```

Anexo 4. Algoritmo de visión artificial

```
import cv2, time
import numpy as np

class OpenCV:

    def image(x, y):
```

```
        x= int(x) + 250
        y= int(y) + 250
        img = np.ones((500,500,1),np.uint8)*0
        cv2.circle(img, (x, y), 10, (255, 0, 0), -1)
        cv2.imwrite('image.png', img)

    def analiza():
        frame = cv2.imread('image.png')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # apply a blur using the median filter
        img = cv2.medianBlur(gray, 5)

        # finds the circles in the grayscale image using the Hough
transform
        circles = cv2.HoughCircles(image=img,
method=cv2.HOUGH_GRADIENT, dp=0.9, minDist=2, param1=10, param2=10,
minRadius=1, maxRadius=15)

        center = (0, 0)
        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
                center = (i[0], i[1])

        return center
```

Anexo 5. Algoritmo de desarrollo de interface

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>PID Ball</title>
    <link rel="stylesheet" href="../static/css/bootstrap.min.css">
</head>

<body style="background: black; text-align: center; margin-top: 4em;">
    <div class="container-fluid">

        <div class="row text-center mb-2 mt-4">
            <div class="col-md-12">
                <a href="#" onclick="PSO(); return false;" class="btn
btn-info">Utilizar PSO</a>
```

```
                <a href="#" onclick="Reiniciar(); return false;"
class="btn btn-success">Reiniciar</a>
            </div>
        </div>

        <div class="row mt-4">
            <div class="col-md-6"><canvas id="lienzo" width="500"
height="500"
                    style="border: 1px solid blanchedalmond; cursor:
pointer;">Su navegador no soporta
                    canvas :( </canvas></div>
            <div class="col-md-2" style="color: white;">
                <h5><strong>PID Eje X</strong></h5>
                <form id="formulario_x">
                    <div class="form-group">
                        <label for="kpx">Coef. Proporcional
[Kp]</label>
                        <input type="number" class="form-control"
id="kpx" step="0.000000001" min="0" value="0">
                    </div>
                    <div class="form-group">
                        <label for="kix">Coef. Integral [Ki]</label>
                        <input type="number" class="form-control"
id="kix" step="0.000000001" min="0" value="0">
                        <small id="kix_e" class="form-text text-
muted">0</small>
                    </div>
                    <div class="form-group">
                        <label for="kdx">Coef. Derivativo [Kd]</label>
                        <input type="number" class="form-control"
id="kdx" step="0.000000001" min="0" value="0">
                        <small id="kdx_e" class="form-text text-
muted">0</small>
                    </div>
                    <button type="submit" class="btn btn-
primary">Actualizar</button>
                </form>
            </div>

            <div class="col-md-2" style="color: white;">
                <h5><strong>PID Eje Y</strong></h5>
                <form id="formulario_y">
                    <div class="form-group">
                        <label for="kpy">Coef. Proporcional
[Kp]</label>
                        <input type="number" class="form-control"
id="kpy" step="0.000000001" min="0" value="0">
                    </div>
                    <div class="form-group">
```

```
                        <label for="kiy">Coef. Integral [Ki]</label>
                        <input type="number" class="form-control"
id="kiy" step="0.000000001" min="0" value="0">
                        <small id="kiy_e" class="form-text text-
muted">0</small>
                    </div>
                    <div class="form-group">
                        <label for="kdy">Coef. Derivativo [Kd]</label>
                        <input type="number" class="form-control"
id="kdy" step="0.000000001" min="0" value="0">
                        <small id="kdy_e" class="form-text text-
muted">0</small>
                    </div>
                    <button type="submit" class="btn btn-
primary">Actualizar</button>
                </form>
            </div>
        </div>
    </div>
    <br>
    <div class="row t-4">
        <div class="col-md-12 text-center">
            <div id="grafica" style="width: 900px; height: 450px;
margin: 0 auto;"></div>
        </div>
    </div>
    <br>
    <br>
    <br>

    <script src="../static/js/jquery.min.js"></script>
    <script src="../static/js/bootstrap.min.js"></script>
    <script src="../static/js/highcharts.js"></script>
    <!-- <script src="../static/js/highcharts.js.map"></script>     -->

    <script src="../static/js/principal.js"></script>
</body>

</html>
```