UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA ELECTROMECÁNICA

LÍNEA TERMINAL EN MECATRÓNICA

Campus San Juan del Río

MÓDULO EMBEBIDO PARA EL PROCESAMIENTO DE MICRO ALGORITMOS GENÉTICOS

TESIS

Que para obtener el título de:

Ingeniero Electromecánico

Presenta:

Silvia Karina Martínez Luna

Dirigido por:

Dr. Arturo Yosimar Jaen Cuellar

Co-dirigido por:

Dr. Roque Alfredo Osornio Ríos

RESUMEN

En esta tesis se presenta un módulo IP-core utilizando micro algoritmos genéticos, se hace uso de ellos debido a que es un método de optimización moderno con grandes ventajas en cuento a los métodos clásicos. Haciendo un análisis de cómo es que funcionan estos algoritmos por medio de su principio: la supervivencia del más fuerte, es decir, la selección natural. Sin embargo, lo que se diferencia es que éste es un micro algoritmo genético, como ha de suponerse, se lleva a cabo con pocos recursos computacionales pues el conjunto de parámetros a analizar es de menor cantidad pero se espera la misma eficiencia. Además, se lleva a cabo las simulaciones de la programación, se migra el código a una FPGA para después mostrar su funcionalidad mediante dos casos de estudio: la función de Bohachevsky y en un motor en vacio. En base a las pruebas realizadas se demuestra la eficiencia y robustez de utilizar el módulo con las alternativas de los operadores de cruza y mutación.

(**Palabras clave**: algoritmos genéticos, micro algoritmos genéticos, FPGA)

AGRADECIMIENTOS

Agradeciendo a mi papá Marco A. C. Martínez y a mi mamá Silvia Luna por apoyarme en cada aspecto de mi vida, uno de ellos es en mis estudios y concederme la dicha de seguir estudiando hasta obtener mi profesión. A mi hermana Ana Karen por ser una incondicional hermana y mi mejor amiga.

A mi asesor el Dr. Arturo Yosimar por su paciencia hacia mí, por estarme guiando, por los comentarios, las observaciones, el apoyo, las correcciones y mejoras para sacar delante de la mejor manera posible además de con éxito mi tesis.

Al Dr. Roque A. Osornio por las revisiones y comentarios hacia mi trabajo para mejorarlo y obtener el mejor resultado.

A mis amigos y compañeros por brindarme su amistad estos cuatro años llenos de trabajos, tareas, proyectos, exposiciones.

También a la Universidad Autónoma de Querétaro por abrirme las puertas a la Facultad de Ingeniería Electromecánica. A mis queridos profesores que me alentaron y motivaron para seguir con esta hermosa carrera, apoyándome y otorgándome de sus conocimientos.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

Existen muchos métodos modernos de optimización, conocidos como técnicas heurísticas, los cuales realizan una búsqueda dentro de un espacio de diseño por el resultado más óptimo de algún problema específico. A diferencia de las técnicas convencionales basadas en gradientes los métodos heurísticos no requieren aplicar matemática avanzada para poder proporcionar un resultado óptimo, ya que se basan en procedimientos que mimetizan el comportamiento de la naturaleza.

Por ello es que se han implementado nuevos métodos de optimización los cuales están enfocados en ciertas conductas y características biológicas, moleculares, sistemas neuronales, etc. Algunos de éstos son: redes neuronales, colonia de hormigas, enjambre de partículas, algoritmos genéticos, entre otros.

En esta tesis se trabaja con los algoritmos genéticos (AG), los cuales están basados en el principio de la selección natural, la teoría de Darwin sobre la supervivencia del más apto. Se caracterizan porque el algoritmo parte de una población inicial de individuos, donde cada individuo es una posible solución al problema que se esté trabajando, dependiendo del tamaño de la población inicial se tendrá una mayor diversidad de posibles soluciones, su forma de búsqueda es de manera aleatoria por medio de dicha población inicial, en la cual se le van aplicando diferentes operadores para obtener la solución óptima. Este método tiene la ventaja de abarcar un gran número de posibilidades ante la solución y así tener una gran diversidad de soluciones, descartando las menos aptas, que perecerán, y creando la mejor solución. Además su metodología hace que la solución sea buscada de manera global y no local. Los algoritmos genéticos estándares (AGE), tienen la desventaja de aplicar poblaciones aproximadamente de 50 o más individuos, además de hacer un

gran número de iteraciones para conseguir la solución óptima, ocasionando que la carga computacional de procesamiento sea pesado y tardado.

Sin embargo, en este trabajo no se realizará un AGE, se aplicarán los micro algoritmos genéticos (MAG), ellos solo son aplicados a una población muy pequeña de individuos, por lo general entre cuatro a seis individuos, y con ello se alcanzará la solución óptima en un número de iteraciones mucho menor a los AGE. La única desventaja es que la convergencia ante la solución puede ser errónea si sucede precipitadamente.

Es así como se trabajó en la implementación de un módulo embebido (IP-*core*) para implementar este método de optimización moderno por medio del uso de una FPGA (Field Programmable Gate Array, Arreglo de Compuertas Programables en Campo). Comprobando la robustez del módulo en la aplicación de él mismo en dos casos: en la función Bohachevsky y en un motor en vacio, realizando una gran variedad de pruebas para diferentes puntos de cruza y mutación.

El contenido de la tesis se describe a continuación: en el capítulo uno se presenta la descripción del problema y la justificación del porque se implementará dicho tema, haciendo un planteamiento general de lo que se pretende realizar. En el capítulo dos se muestra la definición de lo que es un algoritmo genético, como funciona, los operadores que lo integran como son la reproducción, el cruce y la mutación, también es mencionado el concepto de un MAG, comparando la diferencia entre las dos nociones, además de información necesaria para entender los conceptos y así realizar la programación e implementación del proyecto. Posteriormente para el capítulo tres, se exhiben los pasos realizados y desarrollados para establecer la ejecución de un MAG aplicado a una función tridimensional y un motor en vacio, con todo lo que conlleva el desarrollo de la metodología. Además, en el capítulo cuatro se realizan el análisis y se muestran los resultados obtenidos. Finalmente para el capítulo cinco se hacen aclaraciones y análisis de las pruebas realizadas a

través de las conclusiones, sugiriendo algunas mejoras o posibles cambios, para trabajos posteriores que estén interesados en continuar con dicho proyecto.

1.1 Antecedentes

Aunque las raíces de la computación evolutiva se remontan a los primeros días de la informática, los propios algoritmos genéticos fueron inventados en la década de 1960 por John Holland. Su razón para estudiar estos sistemas fue más allá de un deseo de un mejor algoritmo de búsqueda y optimización. Tales métodos fueron (y aún son) considerados útiles abstracciones para el estudio de la evolución misma en ambos escenarios naturales y artificiales (Coley, 1999).

Es así como inició el gran esmero por utilizar y desarrollar estos algoritmos genéticos en la aplicación de diferentes entornos. Se han desarrollado muchos trabajos e investigaciones aplicando los AG algunas de la áreas donde se han aplicado son en redes eléctricas, informática, robótica, en prótesis, en aeroespacial, en estructuras, en controladores PID, etc.

En la Universidad Autónoma de Querétaro se han desarrollado trabajos de estructuras de casas de madera (Hurtado, 2011), optimización de costos de losas de concreto (Angeles, 2010), un diseño estructural de armaduras en tres direcciones (Astudillo, 2010), calibración de un modelo climático para invernadero (Cruz, 2007), un diseño óptimo de una microviga para mejorar su sensibilidad (Cruz Peréz, 2009), todos aplicando un algoritmo genético estándar para poder desarrollar dichos propósitos, pero ninguno de ellos está enfocado al tema de procesos industriales.

Los trabajos más destacados y afines al tema para ésta tesis son el de la tesis llamada *“Diseño e implementación de un algoritmo genético en FPGA para sintonización de controladores PID*” (Ortiz, 2011), el cual expone las

ventajas de utilizar este tipo de algoritmos, donde desarrolla un programa en Matlab para después migrarlo a una FPGA, pero esto es solamente simulado, se menciona que se aplicó a un motor CD pero no se aplica a un sistema mecatrónico como una fresa, para ser más específicos a un servomotor.. Además que utiliza un algoritmo genético estándar y no un micro algoritmo genético como será utilizado en éste trabajo.

También se puede encontrar un sin fin de artículos donde en la mayoría de ellos tratan de el uso de un controlador PID empleando algoritmos genéticos. En algunos de ellos explican solamente lo que son los algoritmos genéticos y como se desarrollan, otros exponen varios métodos de optimización comparados entre si y demuestra los grandes resultados que aportan los AG. En otros casos, utilizan un controlador PID, haciendo las simulaciones y destacando que es mejor sintonizar los parámetros por este medio que por métodos convencionales.

Se han desarrollado trabajos con MAG que su enfoque está basado en una muy pequeña población (Coello & Pulido, 2001) y con ella aún se mantiene la diversidad y se llega a la solución, solo que para evitar la prematura convergencia no se aplica el mismo desarrollo de los operadores para ambas situaciones, es decir llevan a cabo diferentes criterios. Éstos son realizados en medios simulados.

Un artículo de gran impacto y antecedente directo del trabajo que se realizará es el de Jaen et al., (2013), el cual por medio de una FPGA obtiene la sintonización de un controlador PID y analiza el resultado del control. No obstante, la prueba se desarrolla fuera de línea, obteniendo las ganancias y después introduciéndolas al sistema de control y hasta la fecha no hay una implementación hardware. Retomando dicho trabajo, en la propuesta de ésta tesis se desarrollará ese módulo en hardware en forma robusta, donde el usuario tenga la libertad de elegir una serie de parámetros según sus

necesidades en vez de que este módulo ya tenga esos parámetros predeterminados fijos sin opción a cambiar.

Con lo anterior se destaca la importancia de que los MAG sean aplicados como métodos de optimización modernos, teniendo la ventaja de tener una diversidad de soluciones para llevar a la solución óptima y global. La ventaja de un micro algoritmo genético, es que convergirá con una población menor y más rápido, concibiendo la solución en menores iteraciones, pues la respuesta o solución óptima se obtendrá con un número de población considerablemente pequeño y destacando que los operadores empleados son muy sencillos, en cuanto a su matemática, debido que no se emplearán cálculos diferenciales o de gradientes. Además se aplican a modelos no lineales, a diferencia de las técnicas tradicionales, ya que su método de búsqueda la realiza en todas la diversidad de posibles soluciones, y es así como llega a la solución global sin estancarse en la primer solución local que encuentre evitando una prematura y errónea solución. Aunado a todo ello se espera el desarrollo de un módulo embebido que permita tener un desempeño adecuado en las aplicaciones buscando que se puedan aplicar en tiempo real.

1.2 Objetivos

1.2.1 Objetivo General

Diseñar y desarrollar un IP-core embebido para procesamiento de micro algoritmos genéticos mediante lenguaje de descripción de hardware demostrando su uso funcionalidad en la sintonización de controladores en sistemas mecatrónicos.

1.2.2 Objetivos Particulares

-Revisar la literatura referente a algoritmos genéticos, para poder dominar el tema y los conceptos que lo definen, revisando fuentes o referencias de calidad como lo son tesis, tesinas, libros, artículos, etc.

-Desarrollar en software el módulo de procesamiento genético simulado, en el programa de MATLAB para poder concretar los conceptos teóricos y aterrizarlos en el entorno de la programación

-Diseñar el diagrama general de arquitectura de procesamiento genético, con el uso de esquemas y mapas conceptuales que ilustren los módulos principales por desarrollar para tener en claro lo que se desarrollará posteriormente.

-Realizar la descripción en hardware de la arquitectura propuesta, usando un programa de simulación VHDL que permita verificar la correcta aplicación del módulo.

-Desarrollar la descripción de las macros para integración del módulo en hardware con el procesador xq16v7, agregando a la librería ya existen las implementaciones que son necesarias para poder usar el módulo como tal con el llamado de las macros.

-Implementar el sistema embebido en la optimización de un proceso simple, que es por medio de la función de Bohachevsky, permitiendo la verificación del módulo junto con las opciones alternativas añadidas.

-Implementar el sistema embebido y aplicarlo en un proceso mecatrónico industrial, el servomotor de una fresadora, validando el módulo MAG.

1.3 Descripción del problema

Los controladores tradicionales son complicados debido al uso de gradientes por desarrollar en cuanto a la matemática se refiere, y así requerir computacionalmente más recursos. Además para poder sintonizar éstos parámetros es necesario realizar los cálculos y después reajustar las ganancias.

Los algoritmos de búsqueda clásicos son útiles en la búsqueda de la solución de funciones lineales, continuas y diferenciales. Estos métodos son analíticos y hacen uso de las técnicas del cálculo diferencial en la localización de los puntos óptimos. Pero las técnicas clásicas tienen un alcance limitado debido a que algunos problemas prácticos implican funciones no lineales.

Actualmente, se cuentan con controladores que necesitan ser ajustados dependiendo de la planta, es decir, sus ganancias deben ser sintonizadas, pero estas ganancias serán siempre fijas ante la misma planta. Sin embargo, todo proceso (planta), siempre está variando, ya sea debido a la carga, al desgaste, etc. teniendo modificaciones, ocasionando que sea necesario hacer el reajuste de estas ganancias y volver a realizar los cálculos para poder determinar dichos valores.

Por ello se han implementado nuevos métodos de optimización, los cuales están enfocados en ciertas conductas y características biológicas, moleculares, neuronales, etc. La mayoría de estos métodos requiere solo el valor de la función y no sus derivadas. Es así como un MAG se aplicará para el desarrollo de la sintonización de un controlador.

Además, este algoritmo, no se llevará a cabo solamente en software también se implementará en hardware. Debido a que en software es muy lento para realizar el proceso y nos limita para poder ser aplicado en tiempo real. Es así como se desarrolla el módulo en hardware, debido a que una tarjeta FPGA, trabaja a grandes velocidades por operación permitiéndonos realizar las pruebas en tiempo real. También cabe destacar que las pruebas realizadas en software se realizan fuera de línea.

1.4 Justificación

El hecho de aplicar un MAG como medio para encontrar una solución óptima tiene la ventaja que gracias a su método se obtiene la solución óptima

global, en cambio los métodos tradicionales solo trabajan sobre el resultado local según su punto de partida cayendo en una solución que no es la global.

Por otro lado, se empleará dicho algoritmo de manera real, es decir, no solamente se hará en software (simulaciones), también se usará en un proceso real para demostrar su eficiencia (en hardware) en este caso en el servomotor de una fresadora. Siendo un módulo como una unidad de soporte en el procesador xq16v7 (procesador que es desarrollado en la misma institución), es decir, esta herramienta será un módulo robusto el cual se podrá manipular cuando se necesite de su empleo, por ejemplo, en un lazo de control cerrado, será la parte que determine el valor de las ganancias para el control, la sintonización.

Un enfoque primordial es a la necesidad de generar un MAG para agilizar el tiempo del proceso, debido a que se han tenido casos los cuales su tiempo de ejecución ha sido tardado, hasta llevarse incluso días en el proceso para la respuesta final (Cruz Peréz, 2009). Por ello, se quiere agilizar el tiempo de ejecución.

Otro punto muy importante por llevar a cabo éste trabajo, es que se da seguimiento a un trabajo desarrollado en la misma facultad, donde se implementa e integra una manera más robusta de poder utilizar el IP-core. Además con la implementación que se ejecutaron en los MAG, se llevaron a cabo la diversidad de poder elegir diferentes puntos de cruza y mutación que son desarrollados en el proceso para así analizar cuáles son las mejores formar de obtener el resultado óptimo.

1.5 Planteamiento general

De entrada es necesario tener en claro lo que se pretende llevar a cabo en conjunto para desarrollar los pasos de la metodología a seguir. Es así como se tiene un planteamiento de forma muy general y la metodología se vuelve en

lo específico. Así pues, de forma general lo que se desea es tener una función la cual se desea encontrar los valores óptimos para su mejor desempeño de la misma. Por ello en la figura 1 tenemos como elemento primordial o entrada, por así decirlo, la función que deseamos optimizar.

Posteriormente se tiene una interfaz gráfica la cual llevará el control del programa, permitirá comunicarse con el microprocesador por medio de USB. Básicamente la interfaz gráfica controla el proceso iterativo, pero el microprocesador realiza las operaciones en si del propio algoritmo, como lo son el operador de cruce, mutación y ordenamiento de los individuos.

En el compilador se tienen las librerías y elementos necesarios para desarrollar los operadores correctamente, mandando efectuar al módulo MAG que se desarrolló en el microprocesador.

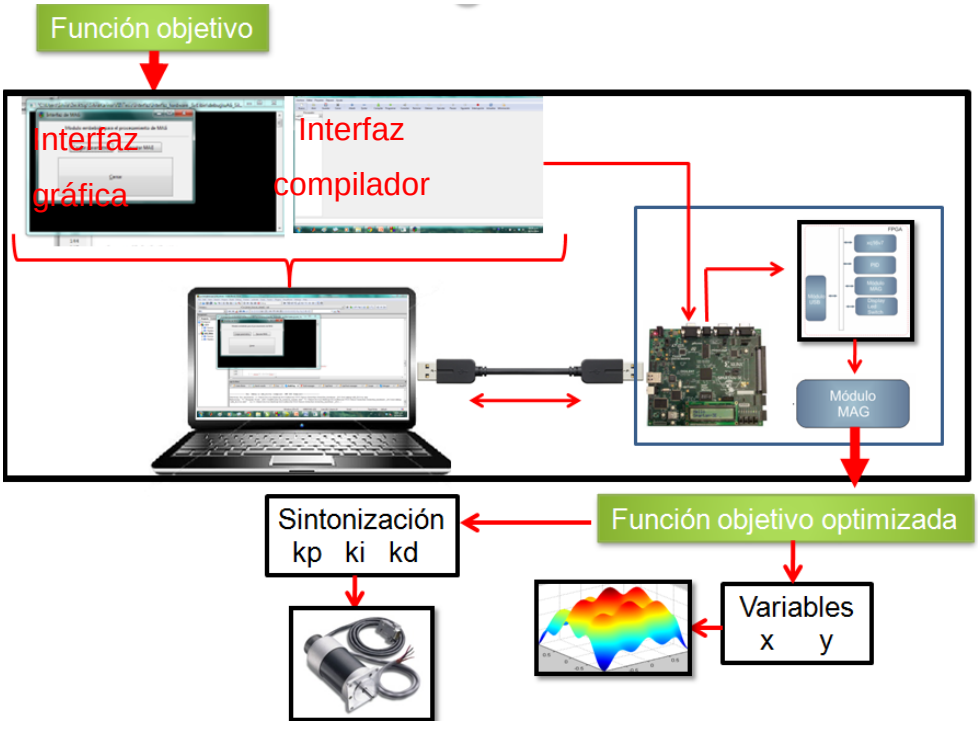


Fig. 1 Planteamiento general a grandes rasgos

Como resultado entre la interacción de la interfaz gráfica con el microprocesador, el resultado arrojado de la última generación serán los valores optimizados de la función objetivo. En los casos finales mostrados en la Fig. 1 como aplicación de todo el algoritmo se tiene a una función tridimensional y a la sintonización de un controlador PID, se quiere obtener el valor máximo y las mejores ganancias para el control, respectivamente.

De tal manera que ahora la metodología a seguir para poder llevar a cabo el módulo MAG se aprecia en la Fig. 2 que está en forma muy general. A continuación se describen los bloques de forma breve.

Es necesario revisar la literatura sobre algoritmos genéticos y micro algoritmos genéticos, para entender el concepto y los principios de operación de tales algoritmos, para posteriormente desarrollar un programa que emplee dichos algoritmos, en éste caso se realizará en MATLAB y así familiarizarse con los operadores, además de concretar los conceptos en el entorno de programación.

Con el código anterior desarrollado en MATLAB, ahora se puede desarrollar en hardware, con un lenguaje de descripción en ello, la implementación del módulo de MAG para el sistema embebido del procesador xq16v7. Realizando sus respectivas simulaciones e implementaciones de las macros para la librería del módulo de MAG y así desarrollar el código junto con la interfaz gráfica.

Finalmente se aplicará el módulo de MAG del sistema embebido en la búsqueda del punto global óptimo máximo de la función de Bohachevsky, además de realizar varias pruebas ante diferentes puntos de cruza y mutación. Posteriormente en la sintonización de las ganancias PID de un servomotor sin carga para darle una validez sólida al módulo. Analizando los resultados y obteniendo las conclusiones necesarias para determinar la funcionalidad y versatilidad del mismo.

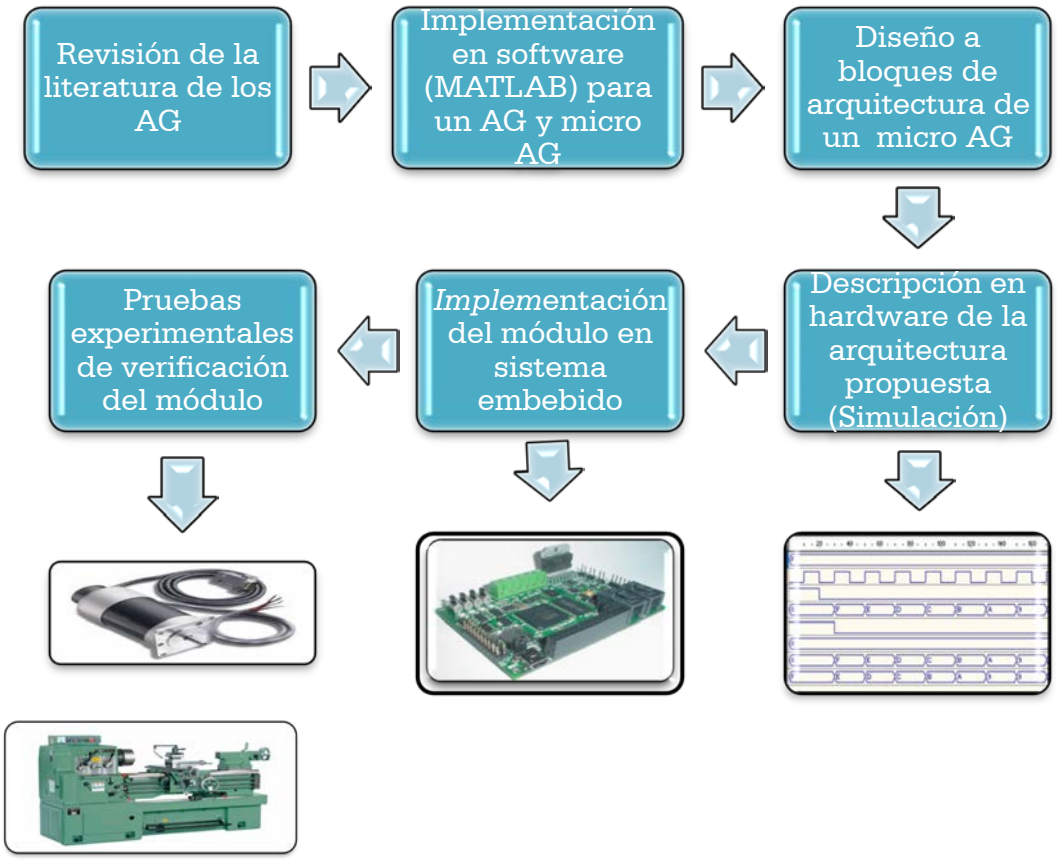


Fig. 2 Metodología general de la tesis

2. Fundamentación teórica

2.1 Estado del Arte

Charles Darwin (1809-1882), autor de la teoría de la evolución, Gregor Mendel (1822-1844), autor de los principios básicos de la teoría de transmisión hereditaria, Hugo de Varis (1848-1935) autor de la teoría de mutación de un gen, Walter Sutton (1877-1916) y Teodor Boveri (1862-1915) autores de la teoría del cromosoma. Son todos científicos quienes pusieron la fundación para el presente estado de conocimiento el cual llega a ser una inspiración para los creadores de la teoría de los algoritmos genéticos (AG). Los primero estudios que pueden ser reconocidos como una conexión con la esta teoría es la investigación hecha por Barricelli, Box, Fraser Friedberg, Friedman, Bledsoe y Bremermann. Sin embargo, es John Holland quien es generalmente reconocido como el creador de la teoría de los algoritmos genéticos. In los 60's del siglo previo John Holland formuló sobre las dos próximas décadas con su equipo, la actualmente conocida forma de los AG.

El rápido crecimiento del interés y el consiguiente desarrollo de la investigación en AG está datada a finales de los años 70's y 80's. Escritos actualmente reconocidos como clásicos De Jong, Goldberg, Davis fueron publicados. Las primeras conferencias fueron organizadas, asociaciones se establecieron y los primeros artículos estrictamente sobre el tema de AG fueron publicados.

Simultáneamente con el desarrollo de los AG, se está avanzado en los métodos relacionados, los cuales también se originan de la corriente evolutiva. Estos métodos comprenden:

- Programación evolutiva (PE) – método planeado por Lawrence Fogel;

- Estrategias de evolución (EE) – método planeado por Ingo Rechemberg y Hans-Paul Schwefel;
- Programación genética (PG) – método planeado por John Korza, aunque su estudio se deriva de la investigación de Smith y Cramer.

Actualmente todos los métodos originados de la corriente evolutiva a menudo se conocen comúnmente como métodos de la computación evolutiva. Esto no significa, que estos métodos se fusionaron en uno solo. Ya que descienden de la misma fuente, estos métodos a menudo inspiraron uno al otro, no obstante, ellos usualmente continúan para ser desarrollados individualmente (Gwiazda, 2007).

2.2 Introducción a los algoritmos genéticos

En recientes años, algunos métodos de optimización son conceptualmente diferentes de las técnicas de programación tradicionales. Estos métodos son etiquetados como métodos de optimización modernos o no tradicionales la mayoría de estos métodos están basados en ciertas características y conductas biológicas, moleculares y sistemas neurobiológicos (Rao, 2009).

Los algoritmos genéticos son algoritmos numéricos de optimización inspirados por la selección natural y la genética natural. El método es en general capaz de ser aplicado a un extremadamente amplio rango de problemas. Están siendo usados para ayudar a resolver problemas prácticos a diario (Coley, 1999).

Los algoritmos genéticos son radicalmente diferentes de los métodos tradicionales. Por ejemplo, los AG no usan gradientes, consecuentemente, ellos son aplicados a una amplia clase para problemas de optimización (Chong & Zak, 2001).

La idea de usar una población de soluciones para resolver problemas de optimización de ingeniería prácticos fue considerada varias veces durante los 1950's y 1960's. Sin embargo, fue en esencia inventado por John Holland. Su libro de 1975, *Adaptation in Natural and Artificial Systems*, es particularmente valorado por su enfoque visionario. Otros, por ejemplo De Jong, en un artículo titulado *Genetic Algorithms are not Function Optimizers*, ha tenido mucho interés para recordar que la AG son potencialmente mucho más que un método robusto para estimar una serie de parámetros desconocidos dentro de un modelo de sistema físico (Coley, 1999).

2.3 Algoritmos genéticos

Algoritmos Genéticos (AG) son técnicas de optimización basadas en la simulación de los fenómenos que tienen lugar en la evolución de las especies haciendo una adaptación a un problema de optimización (Meng & Song, 2007).

Los AG difieren fuertemente en la concepción de otros métodos de búsqueda, incluidos los métodos de optimización tradicionales y otros métodos de búsqueda estocásticos. La diferencia básica es que mientras que otros métodos siempre procesan puntos únicos en el espacio de búsqueda, los AG mantienen una población de soluciones potenciales. Además constituyen una clase de métodos de búsqueda, especialmente adecuado para la resolución óptima de problemas complejos (Renner & Ekárt, 2002).

Los AG son inicializados con una población supuesta (aleatoria). Usualmente hay un seleccionador al azar (*random*). El algoritmo típico usa tres operadores: selección o reproducción, cruce y mutación para dirigir a la población a converger en la solución optima global.

Típicamente, estas aproximaciones iniciales son codificaciones binarias (o cadenas) de las verdaderas variables. Esta inicial población es entonces procesada por los tres principales operadores (Coley, 1999).

2.3.1 Operadores genéticos

Un ciclo de reproducción, cruce y mutación y la evaluación del valor de la función aptitud (*fitness)* es conocida como una generación. Si el criterio de convergencia es satisfactorio, la población es iterativamente operada por los tres operadores y el resultado de la nueva población es evaluada por la función fitness. El proceso es continuamente a través de varias generaciones hasta que la convergencia del criterio este satisfecha y el proceso esté terminado (Rao, 2009).

2.3.1.1 Reproducción

La reproducción es el primer operador aplicado a la población para seleccionar las mejores cadenas (diseños) de la población para formar una *mating pool*, este término se refiere a la cantidad de individuos que se deben tener siempre en cada generación definidos desde un principio, es como un recipiente que contiene esos individuos y siempre debe estar con esa cantidad de individuos. Estas cadenas son llamadas cromosomas (Chong & Zak, 2001). El operador reproducción es también llamado el operador de selección porque selecciona las mejores cadenas de la población. Este operador es usado para recolectar las cadenas arriba del promedio de la actual población e insertar sus múltiples copias en la mating pool basada sobre un proceso probabilístico (Rao, 2009).

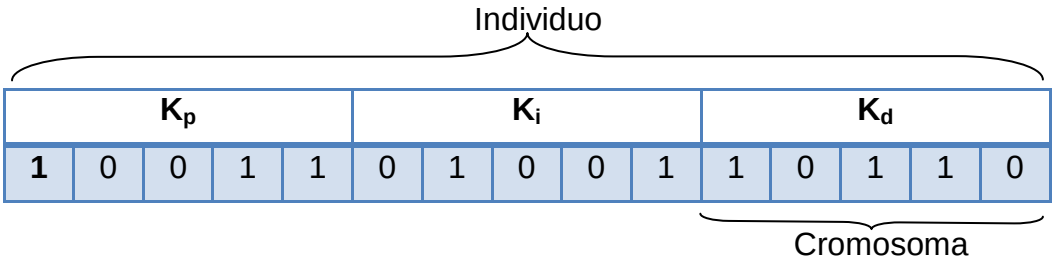


Fig. 3 Haciendo una analogía con los parámetros de un PID, distinguimos se distingue lo que es un individuo y un cromosoma

A cada individuo le corresponde un valor de la función objetivo, referida a la aptitud del cromosoma.

Para la selección de este operador se pueden utilizar varios medios, los más utilizados son por la rueda de la ruleta y por torneo.

Por la rueda de la ruleta, cada espacio es asignado a un individuo. Algunos individuos pueden ser asignados a múltiples espacios. El número de espacios asociados con cada individuo es proporcional a su aptitud. Donde nosotros hacemos girar la ruleta *N* veces hasta que la mating pool contenga todo los *N* cromosomas (Chong & Zak, 2001).

Y en el de torneo, se seleccionan un par de individuos de manera aleatoria, se compara su valor de aptitud y se toma el que tenga la mejor de todos. Se repite el proceso hasta que la mating pool contenga los *N* individuos (Chong & Zak, 2001).

Es importante mencionar la diferencia entre la función objetivo y la función aptitud o fitness.

La función objetivo se utiliza para proporcionar una medida de cómo los individuos se han desempeñado en el dominio del problema. En el caso de un problema de minimización, los individuos más aptos tendrán el valor numérico más bajo de la función objetivo asociada.

Otra función que es la función de aptitud se utiliza normalmente para transformar el valor de la función objetivo en una medida de la aptitud relativa. Esta asignación es siempre necesaria cuando la función objetivo es para ser minimizada como los valores más bajos de la función objetivo corresponden a los individuos más aptos (Nazir et al., 2009).

2.3.1.2 Cruce

Después de la reproducción, el operador de cruce es implementado. El propósito del cruce es crear nuevas cadenas por intercambiar información entre cadenas de la mating pool. Se eligen dos cadenas (individuos) de manera

aleatoria de la mating pool generada por el operador reproducción y algunas de las porciones de estas cadenas son intercambiadas entre ellas.

Los dos individuos seleccionados para la participación del cruce son conocidos como padres y los individuos generados por este operador con conocidos como hijos o descendencia (Rao, 2009). Haciéndolo de manera gráfica y más clara en la Fig. 4.

Padre 1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Padre 2	1	1	1	1	1	1	1	1	1	1

Hijo 1	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Hijo 2	1	1	1	0	0	0	0	0	0	0

Fig. 4 Se muestra el punto de cruce y la descendencia que se obtuvo

Los hijos generados pudieran ser mejores que los padres. Si ellos son mejores, contribuirán a converger más rápido. Por otro lado, si son peores no ayudarían al éxito del resultado, ya que no sobrevivirá en la siguiente generación.

Así que el efecto del cruce puede ser tanto útil como perjudicial. Por ellos al igual que los padres son elegidos de la mating pool aleatoriamente, la probabilidad de que los individuos se les aplique el cruce es p_c. (Chong & Zak, 2001). Los valores típicos de p_c están entre 0.4 y 0.9 (Coley, 1999). Así solamente se les aplicará a una pequeña parte de la población el cruce, manteniendo a los de forma original.

2.3.1.3 Mutación

El cruce es el principal operador por el cual nuevos individuos con mejores valor de aptitud son creados para la nueva generación. El operador de mutación es aplicado a los nuevos individuos con una específica pequeña probabilidad de mutación, p_m. El operador de mutación cambia un digito binario

a 1 ó 0. Es decir, un número aleatorio entre 0 y 1 es generado/elegido. Si el número aleatorio es más pequeño que p_m, entonces el número binario es cambiado de 1 a 0 o de 0 a 1. Por otro lado, el digito binario no es cambiado.

El propósito de la mutación es generar una cadena en el vecindario de la cadena actual, de este modo conseguimos una investigación local alrededor de la solución actual, para salvaguardar contra una prematura perdida de importante material genético en una posición en particular, y mantener la diversidad en la población.

Cadena original	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Cadena con mutación	0	0	0	1	0	0	0	0	0	0

Fig. 5 Ejemplo del operador mutación

2.4 Micro algoritmos genéticos

Un micro algoritmo genético se diferencia de los estándares debido al número de individuos empleados, es por ellos que se llama micro, porque solamente trabaja con una pequeña población, y el número de iteraciones es menor. El número de individuos en una pequeña población es considerado a partir de 3 o 4 individuos (Coello & Pulido, 2001).

El tamaño de la población es el factor clave en el funcionamiento genético de optimización del micro-algoritmo. Así se ahorran recursos computacionales en comparación de los AG estándares debido a que en los estándares se tiene un número grande de individuos y tienen que ser analizados en cada etapa de los operadores, realizando muchas iteraciones, mientras que con unos cuantos individuos la carga computacional es menor.

El proceso para realizar los operadores no es de la misma forma que un AG estándar. Son aplicados de diferente manera debido a que solo se tienen

4 individuos, y si se le aplican los operadores como en un AG estándar se puede perder información y llevar a la respuesta incorrecta.

2.5 Lenguaje de descripción en Hardware (VHDL)

VHDL se define del lenguaje de descripción de hardware VHSIC (*Very High Speed Integrated Circuit)* y HDL (*hardware description language).* Fue originalmente patrocinado por el Departamento de Defensa de E.U. y más tarde se transfirió a la IEEE (Institute of Electrical and Electronics Enginners). El lenguaje es formalmente definido por la norma 1076 de la IEEE (Chu, 2008).

VHDL es una notación formal destinada para su uso en todas las fases de la creación de sistemas electrónicos. Debido a que es a la vez de lectura mecánica y legible, es compatible con el desarrollo, la verificación, la síntesis y prueba de diseños de hardware, la comunicación de los datos de diseño de hardware, y el mantenimiento, la modificación, y la adquisición de hardware (IEEE, 2008).

Existen diversos lenguajes descriptivos que han sido desarrollados en los últimos años, pero indudablemente el lenguaje que más difusión ha tenido y que se utiliza mayormente es el VHDL.

Las características principales que hacen del VHDL el lenguaje universal de descripción de circuitos son el ser un lenguaje estándar definido como tal por el IEEE y que los proveedores del paquete tienen que seguir el estándar, haciendo que los diseños sean portátiles a cualquier plataforma.

Existen diversos proveedores de VHDL y todos ellos tienen sus características propias en cuanto al entorno gráfico del usuario, sin embargo, todos ellos son compatibles con la definición estándar y por lo tanto, una descripción de circuitos hecha bajo un entorno, puede ser llevada a otro sin cambios sustanciales.

Se deben diferenciar perfectamente dos aspectos importantes de las herramientas de diseño: el entorno gráfico al usuario y el algoritmo de síntesis. El entorno gráfico al usuario son todos los elementos de desplegado, ayudas y ventanas para el manejo de los proyectos. El algoritmo de síntesis es la interpretación y ejecución del proyecto en sí. Dependiendo del proveedor del programa, el entorno puede cambiar de uno a otro, pero el algoritmo de síntesis es el mismo para todos.

Una de las ventajas del lenguaje VHDL es que permite un diseño jerárquico, es decir, que se pueden establecer diversos niveles de abstracción para el diseño que van desde la definición de compuertas básicas hasta sistemas complejos, pasando por bloques funcionales y unidades operativas.

El último proceso de diseño es la síntesis, donde se toma el resultado de la compilación y se convierte en un mapa de interconexiones entre los diversos elementos que contiene un circuito programable donde se vacía .la información del proyecto. Antes de decidir sobre la fabricación o programación del circuito, siempre es conveniente realizar una simulación del mismo para poder verificar su correcta definición, y sobre todo, que realice la tarea para la cual fue creado (Troncoso, 2007).

2.6 Arreglo de compuertas programables en campo (FPGA)

Desde teléfonos inteligentes a equipos médicos, de microondas a los sistemas de frenado de antibloqueo. Los proyectos de sistemas embebidos modernos desarrollan máquinas de computación que se han convertido en una parte integral de nuestra sociedad. Para desarrollar estos productos, los ingenieros emplean un amplio rango de herramientas y tecnología para ensamblar sistemas embebidos de componentes hardware y software. Un componente, el FPGA (Field-Programmable Gate Array); es llegando a ser más importante hoy en día.

Informalmente, un FPGA puede ser pensado como una "pizarra en blanco" en la que cualquier circuito digital puede ser configurado, es decir, puede ser configurado después de la fabricación del dispositivo, instalado en un producto o, en algunos casos, incluso después de que el producto ha sido enviado al consumidor. Esto hace que el dispositivo FPGA fundamentalmente sea diferente de otros dispositivos de circuitos integrados. En definitiva, un FPGA proporciona "hardware" programable para el desarrollador de sistemas embebidos (Sass & Schmidt, 2010).

2.6.1 ¿Qué es un FPGA?

En el nivel más alto, los FPGAs son chips de silicio reprogramables. Al utilizar bloques de lógica pre-construidos y recursos para ruteo programables, usted puede configurar estos chips para implementar funcionalidades personalizadas en hardware sin tener que utilizar una tablilla de prototipos o un cautín. Sólo deberá desarrollar tareas de cómputo digital en software y compilarlas en un archivo de configuración o bitstream que contenga información de cómo deben conectarse los componentes. Además, los FPGAs son completamente reconfigurables y al instante toman una nueva “personalidad” cuando usted compila una diferente configuración de circuitos. Anteriormente sólo los ingenieros con un profundo entendimiento de diseño de hardware digital podían trabajar con la tecnología FPGA. Sin embargo, el aumento de herramientas de diseño de alto nivel está cambiando las reglas de programación de FPGAs, con nuevas tecnologías que convierten los diagramas a bloques gráficos, o hasta el código ANSI C a circuitos de hardware digital.

La adopción de chips FPGA en la industria ha sido impulsada por el hecho de que los FPGAs combinan lo mejor de los ASICs y de los sistemas basados en procesadores. Ofrecen velocidades temporizadas por hardware y fiabilidad, pero sin requerir altos volúmenes de recursos para compensar el gran gasto que genera un diseño personalizado de ASIC. El silicio reprogramable tiene la misma capacidad de ajustarse que un software que se ejecuta en un

sistema basado en procesadores, pero no está limitado por el número de núcleos disponibles. A diferencia de los procesadores, los FPGAs llevan a cabo diferentes operaciones de manera paralela, por lo que éstas no necesitan competir por los mismos recursos. Cada tarea de procesos independientes se asigna a una sección dedicada del chip, y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan otros procesos (Thompson, 2012).

Las FPGAs se introdujeron en 1985 por la Compañía de Xilinx. Desde entonces, muchos diferentes FPGAs han sido desarrollados por una serie de empresas: Actel, Altera, Plessey, Plus, Advanced Micro Devices (AMD), QuickLogjc, Algotronix, entre otras. Consta de una matriz bidimensional de bloques lógicos que se puede conectar con recursos generales de interconexión. La interconexión comprende segmentos de alambre, donde los segmentos pueden ser de varias longitudes. Presente en la interconexión son interruptores programables que sirven para conectar los bloques lógicos a los segmentos de alambre, o un segmento de cable a otro. Los circuitos lógicos se implementan en el FPGA mediante la partición de la lógica en bloques lógicos individuales y luego la interconexión de los bloques como sea necesario en los interruptores.

Para facilitar la aplicación de una amplia variedad de circuitos, es importante que un FPGA sea tan versátil como sea posible. Esto significa que el diseño de los bloques lógicos, junto con la de los recursos de interconexión, debe facilitar la aplicación de un gran número de circuitos lógicos digitales (Brown, J. Francis, Rose, & Vranesik, 1992).

2.6.2 Los Cinco Beneficios Principales de la Tecnología FPGA

Rendimiento – Aprovechando del paralelismo del hardware, los FPGAs exceden la potencia de cómputo de los procesadores digitales de señales (DSPs) rompiendo el paradigma de ejecución secuencial y logrando más en

cada ciclo de reloj. BDTI, una destacada firma analista que realiza evaluaciones de referencia, lanzó evaluaciones mostrando cómo los FPGAs pueden entregar significativamente más potencia de procesamiento por dólar que una solución de DSP, en algunas aplicaciones2. El controlar entradas y salidas (E/S) a nivel de hardware ofrece tiempos de respuesta más veloces y funcionalidad especializada que coincide con los requerimientos de una aplicación.

Tiempo en llegar al mercado – La tecnología FPGA ofrece flexibilidad y capacidades de rápido desarrollo de prototipos para enfrentar los retos de que un producto se libere tarde al mercado. Usted puede probar una idea o un concepto y verificarlo en hardware sin tener que pasar por el largo proceso de fabricación por el que pasa un diseño personalizado de ASIC. Posteriormente podrá implementar cambios y realizar iteraciones de un diseño FPGA en cuestión de horas en vez de semanas. También existe hardware comercial listo para usarse (COTS) con diferentes tipos de E/S ya conectados a un chip FPGA programable por el usuario. El aumento en disponibilidad de herramientas de software de alto nivel disminuye la curva de aprendizaje con niveles de abstracción. Estas herramientas frecuentemente incluyen importantes núcleos IP (funciones pre-construidas) para control avanzado y procesamiento de señales.

Precio – El precio de la ingeniería no recurrente de un diseño personalizado ASIC excede considerablemente al de las soluciones de hardware basadas en FPGA. La fuerte inversión inicial de los ASICs es fácilmente justificable para los fabricantes de equipos originales que embarcan miles de chips por año, pero muchos usuarios finales necesitan la funcionalidad de un hardware personalizado para decenas o cientos de sistemas en desarrollo. La misma naturaleza programable del silicio implica que no hay precio de fabricación o largo plazos de ejecución de ensamblado. Los requerimientos de un sistema van cambiando con el tiempo, y el precio de

cambiar incrementalmente los diseños FPGA es insignificante al compararlo con el precio de implementar cambios en un ASIC antes de su lanzamiento.

Fiabilidad – Mientras que las herramientas de software ofrecen un entorno de programación, los circuitos de un FPGA son una implementación segura de la ejecución de un programa. Los sistemas basados en procesadores frecuentemente implican varios niveles de abstracción para auxiliar a programar las tareas y compartir los recursos entre procesos múltiples. El software a nivel driver se encarga de administrar los recursos de hardware y el sistema operativo administra la memoria y el ancho de banda del procesador. El núcleo de un procesador sólo puede ejecutar una instrucción a la vez, y los sistemas basados en procesadores están siempre en riesgo de que sus tareas se obstruyan entre sí. Los FPGAs, que no necesitan sistemas operativos, minimizan los retos de fiabilidad con ejecución paralela y hardware preciso dedicado a cada tarea.

Mantenimiento a largo plazo – Como se mencionó anteriormente, los chips FPGA son actualizables en campo y no requieren el tiempo y el precio que implica rediseñar un ASIC. Los protocolos de comunicación digital por ejemplo, tienen especificaciones que podrían cambiar con el tiempo, y las interfaces basadas en ASICs podrían causar retos de mantenimiento y habilidad de actualización. Los chips FPGA, al ser reconfigurables, son capaces de mantenerse al tanto con modificaciones a futuro que pudieran ser necesarias. Mientras el producto o sistema se va desarrollando, usted puede implementarle mejoras funcionales sin la necesidad de invertir tiempo rediseñando el hardware o modificando el diseño de la tarjeta (Thompson, 2012).

2.7 Control

2.7.1 Control en lazo cerrado

Un sistema que mantiene una relación determinada entre la salida y la entrada de referencia, comparándolas y usando la diferencia como medio de

control, se denomina *sistema de control realimentado*. Un ejemplo sería el sistema de control de temperatura de una habitación. Midiendo la temperatura real y comparándola con la temperatura de referencia (temperatura deseada), el termostato activa o desactiva el equipo de calefacción o de enfriamiento para asegurar que la temperatura de la habitación se mantiene en un nivel confortable independientemente de las condiciones externas.

Los sistemas de control realimentados no se limitan a la ingeniería, sino que también se encuentran en diversos campos ajenos a ella. Por ejemplo, el cuerpo humano es un sistema de control realimentado muy avanzado. Tanto la temperatura corporal como la presión sanguínea se conservan constantes mediante una realimentación fisiológica. De hecho, la realimentación realiza una función vital: hace que el cuerpo humano sea relativamente insensible a las perturbaciones externas, permitiendo que funcione de forma adecuada en un entorno cambiante.

Los sistemas de control realimentados se denominan también sistemas de *control en lazo cerrado*. En la práctica, los términos control realimentado y control en lazo cerrado se usan indistintamente. En un sistema de control en lazo cerrado, se alimenta al controlador la señal de error de actuación, que es la diferencia entre la señal de entrada y la señal de realimentación (que puede ser la propia señal de salida o una función de la señal de salida y sus derivadas y/o integrales), con el fin de reducir el error y llevar la salida del sistema a un valor deseado. El término control en lazo cerrado siempre implica el uso de una acción de control realimentado para reducir el error del sistema.

2.7.2 Controladores automáticos

Un controlador automático compara el valor real de la salida de una planta con la entrada de referencia (el valor deseado), determina la desviación y produce una señal de control que reduce la desviación a cero o a un valor pequeño. La manera en la cual el controlador automático produce la señal de control se denomina acción de control. La Fig. 6 es un diagrama de bloques de

un sistema de control industrial que consiste en un controlador automático, un actuador, una planta y un sensor (elemento de medición). El controlador detecta la señal de error, que por lo general, está en un nivel de potencia muy bajo, y la amplifica a un nivel lo suficientemente alto. La salida

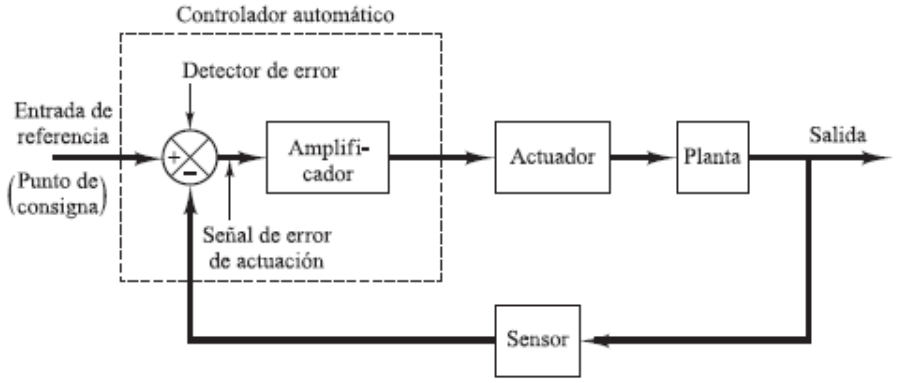


Fig. 6 Diagrama de bloques de un sistema de control industrial, formado por un controlador automático, un actuador, una planta y un sensor.

de un controlador automático se alimenta a un actuador, como un motor o una válvula neumáticos, un motor hidráulico o un motor eléctrico. (El actuador es un dispositivo de potencia que produce la entrada para la planta de acuerdo con la señal de control, a fin de que la señal de salida se aproxime a la señal de entrada de referencia.) El sensor, o elemento de medición, es un dispositivo que convierte la variable de salida en otra variable manejable, como un desplazamiento, una presión o un voltaje, que pueda usarse para comparar la salida con la señal de entrada de referencia. Este elemento está en la trayectoria de realimentación del sistema en lazo cerrado. El punto de ajuste del controlador debe convertirse en una entrada de referencia con las mismas unidades que la señal de realimentación del sensor o del elemento de medición.

2.7.3 Clasificación de los controladores industriales

Los controladores industriales se clasifican, de acuerdo con sus acciones de control, como:

- Controladores proporcionales

- Controladores integrales
- Controladores proporcionales-integrales
- Controladores proporcionales-derivativos
- Controladores proporcionales-integrales-derivativos

La mayoría de los controladores industriales emplean como fuente de energía la electricidad o un fluido presurizado, como el aceite o el aire. Los controladores también pueden clasificarse, según el tipo de energía que utilizan en su operación, como neumáticos, hidráulicos o electrónicos. El tipo de controlador que se use debe decidirse basándose en la naturaleza de la planta y las condiciones de operación, incluyendo consideraciones tales como seguridad, costo, disponibilidad, fiabilidad, precisión, peso y tamaño.

- **Acción de control proporcional (P).** Para un controlador con acción de control proporcional, la relación entre la salida del controlador u(t) y la señal de error e(t) es:

$$u(t) = K_P e(t)$$

o bien, en cantidades transformadas por el método de Laplace,

$$\frac{U(s)}{E(s)} = K_p$$

Donde K_p es una ganancia proporcional ajustable. Cualquiera que sea el mecanismo real y la forma de la potencia de operación, el controlador proporcional es, en esencia, un amplificador con una ganancia ajustable.

- **Acción de control integral (I).** En un controlador con acción de control integral, el valor de la salida del controlador *u*(*t*) se cambia a una razón proporcional a la señal de error *e*(*t*). Es decir,

$$\frac{du(t)}{dt} = K_i e(t)$$

o bien

$$u(t) = K_i \int_0^t e(t)dt$$

donde *Ki* es una constante ajustable. La función de transferencia del controlador integral es

$$\frac{U(s)}{E(s)} = \frac{K_i}{s}$$

- **Acción de control proporcional-integral (PI).** La acción de control de un controlador proporcional-derivativa (PD) se define mediante

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt$$

o la función de transferencia del controlador es

$$\frac{U(s)}{E(s)} = K_p \left(1 + \frac{1}{T_i s}\right)$$

donde *Ti* se denomina *tiempo integral*.

- **Acción de control proporcional-derivativa (PD).** se define mediante:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt}$$

o la función de transferencia es

$$\frac{U(s)}{E(s)} = K_p (1 + T_d s)$$

donde *Td* es el *tiempo derivativo*.

- **Acción de control proporcional-integral-derivativa (PID).** La combinación de la acción de control proporcional, la acción de control

integral y la acción de control derivativa se denomina acción de control proporcional-integral-derivativa. Esta acción combinada tiene las ventajas de cada una de las tres acciones de control individuales. La ecuación de un controlador con esta acción combinada está dada por

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

o la función de transferencia es

$$\frac{U(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$

donde *Kp* es la ganancia proporcional, *Ti* es el tiempo integral y *Td* es el tiempo derivativo. El diagrama de bloques de un controlador proporcional-integral-derivativo aparece en la Fig. 7. (Ogata, 2010).

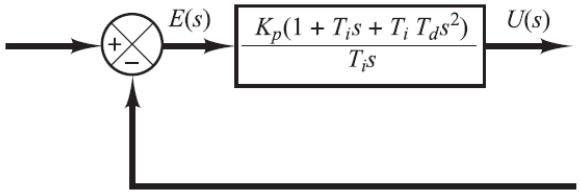


Fig. 7 Diagrama a bloques de un controlador PID

2.7.4 Sistemas de segundo orden

En esta sección, se obtendrá la respuesta de un sistema de control típico de segundo orden para una entrada escalón, rampa e impulso. Aquí se considera un servomotor como ejemplo de un sistema de segundo orden.

Servosistema. El servosistema que se muestra en la Fig. 8(a) consiste en un controlador proporcional y elementos de carga (elementos de inercia y fricción viscosa). Se supone que se desea controlar la posición de salida *c* de

forma que siga a la posición de entrada *r*. La ecuación para los elementos de carga es

$$J\ddot{c} + B\dot{c} = T$$

donde *T* es el par producido por el controlador proporcional de ganancia *K*. Tomando la transformada de Laplace a ambos lados de esta última ecuación, suponiendo condiciones iniciales nulas, se obtiene

$$Js^2C(s) + BsC(s) = T(s)$$

Por tanto, la función de transferencia entre *C*(*s*) y *T*(*s*) es

$$\frac{C(s)}{T(s)} = \frac{1}{s(Js + B)}$$

Utilizando esta función transformada, la Fig. 8(a) se puede redibujar como se muestra en la Fig. 8(b), que se puede modificar como se muestra en la Fig. 8(c). La función de transferencia en lazo cerrado se obtiene entonces como

$$\frac{C(s)}{R(s)} = \frac{K}{Js^2 + Bs + K} = \frac{k/J}{s^2 + (B/J)s + (K/J)}$$

Tal sistema en el que la función de transferencia en lazo cerrado posee dos polos se denomina sistema de segundo orden. (Algunos sistemas de segundo orden pueden contener uno o dos ceros.)

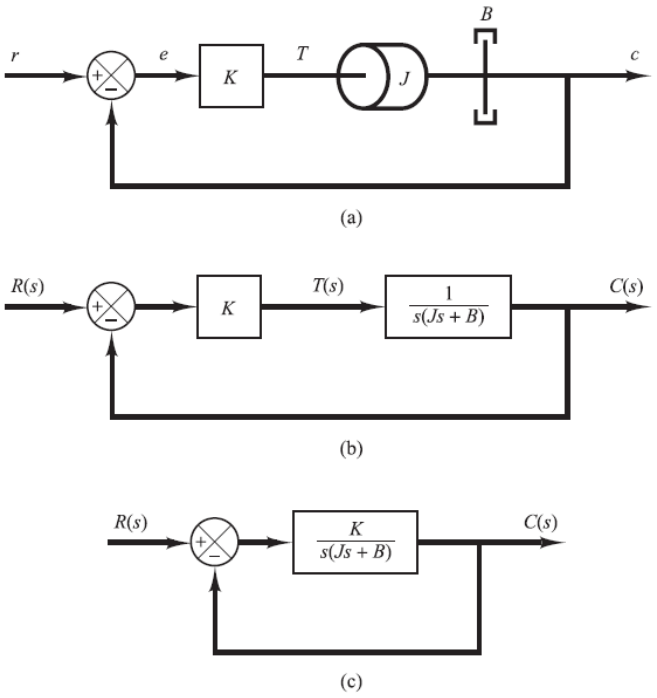


Fig. 8 (a) Servosistema; (b) diagrama de bloques; (c) diagrama de bloques simplificado.

Respuesta escalón de sistemas de segundo orden. La función de transferencia en lazo cerrado del sistema de la Fig. 8(c) es:

$$\frac{C(s)}{R(s)} = \frac{K}{Js^2 + Bs + K}$$

que puede reescribirse como

$$\frac{C(s)}{R(s)} = \frac{\frac{K}{J}}{\left[s + \frac{B}{2J} + \sqrt{\left(\frac{B}{2J}\right)^2 - \frac{K}{J}}\right]\left[s + \frac{B}{2J} - \sqrt{\left(\frac{B}{2J}\right)^2 - \frac{K}{J}}\right]}$$

Los polos en lazo cerrado son complejos si B^2-4JK<0, y son reales si B^2-4JK≥0. En el análisis de la respuesta transitoria, es conveniente escribir

$$\frac{K}{J} = \omega_n^2, \qquad \frac{B}{J} = 2\zeta\omega_n = 2\sigma$$

donde σ se denomina *atenuación*; ω_n, *frecuencia natural no amortiguada*, y ζ, *factor de amortiguamiento relativo* del sistema. El factor de amortiguamiento relativo ζ es el cociente entre el amortiguamiento real *B* y el amortiguamiento crítico $B_c=2\sqrt{JK}$ o bien

$$\zeta = \frac{B}{B_c} = \frac{B}{2\sqrt{JK}}$$

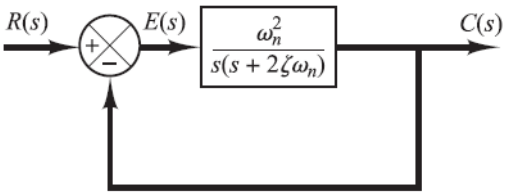


Fig. 9 Sistema de Segundo orden

En términos de ζ y ω_n, el sistema de la Fig. 8(c) se convierte en el que aparece en la Fig. 9, y la función de transferencia en lazo cerrado *C*(*s*)/*R*(*s*) obtenida mediante la ecuación

$$\frac{C(s)}{R(s)} = \frac{K}{Js^2 + Bs + K}$$

Se escribe como

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + \ 2\zeta\omega_n\ s + \ \omega_n^2}$$

Esta forma se denomina *forma estándar* del sistema de segundo orden.

El comportamiento dinámico del sistema de segundo orden se describe a continuación en términos de dos parámetros ζ y ω_n. Si 0< ζ <1, los polos en lazo cerrado son complejos conjugados y se encuentran en el semiplano izquierdo del plano *s*. El sistema, entonces, se denomina subamortiguado y la respuesta transitoria es oscilatoria. Si ζ=0, la respuesta transitoria no se

amortigua. Si ζ=1, el sistema se denomina críticamente amortiguado. Los sistemas sobreamortiguados corresponden a ζ>1.

Ahora se obtendrá la respuesta del sistema que aparece en la Fig. 9 para una entrada escalón unitario. Se considerarán tres casos diferentes: el subamortiguado (0<ζ<1), el críticamente amortiguado (ζ=1) y el sobreamortiguado (ζ>1).

1) *Caso subamortiguado* (0<ζ<1): en este caso, *C*(*s*)/*R*(*s*) se escribe como

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{(s + \zeta\omega_n + j\omega_d)(s + \zeta\omega_n - j\omega_d)}$$

donde $\omega_d = \omega_n\sqrt{1-\zeta^2}$. La frecuencia ω_d se denomina *frecuencia natural amortiguada*. Para una entrada escalón unitario, *C*(*s*) se escribe como

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{(s^2 + \ 2\zeta\omega_n\ s + \ \omega_n^2)\ s}$$

Obteniendo la transformada inversa de Laplace de la ecuación anterior es

$$c(t) = 1 - e^{-\zeta\omega_n t}\left(cos\omega_d t + \frac{\zeta}{\sqrt{1-\zeta^2}}\ sen\omega_d t\ \right)$$

$$= \boxed{1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}} sen\left(\omega_d t + tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right)}, \qquad para\ t \geq 0$$

Este resultado se obtiene directamente usando una tabla de transformadas de Laplace. A partir de la última ecuación se observa que la frecuencia de oscilación transitoria es la frecuencia natural amortiguada ω_d y que, por tanto, varía con el factor de amortiguamiento relativo ζ. La señal de error para este sistema es la diferencia entre la entrada y la salida, y es

$$c(t) = r(t) - c(t) = e^{-\zeta\omega_n t}\left(cos\omega_d t + \frac{\zeta}{\sqrt{1-\zeta^2}}\, sen\omega_d t\right), \qquad para\ t \geq 0$$

Esta señal de error presenta una oscilación sinusoidal amortiguada. En estado estacionario, o en *t*=∞, no existe un error entre la entrada y la salida.

Si el factor de amortiguamiento relativo ζ es igual a cero, la respuesta se vuelve no amortiguada y las oscilaciones continúan indefinidamente. La respuesta *c*(*t*) para el caso del amortiguamiento cero se obtiene sustituyendo ζ=0 en la ecuación, lo cual produce

$$c(t) = 1 - cos\omega_d t\,, \qquad para\ t \geq 0$$

Por tanto, a partir de la ésta ecuación, se establece que ω_n representa la frecuencia natural no amortiguada del sistema. Es decir, u*n* es la frecuencia a la cual el sistema oscilará si el amortiguamiento disminuyera a cero. Si el sistema lineal tiene cualquier cantidad de amortiguamiento, no se puede observar experimentalmente la frecuencia natural no amortiguada. La frecuencia que se observa es la frecuencia natural amortiguada ω_d, que es igual a $\omega_n\sqrt{1-\zeta^2}$. Esta frecuencia siempre es menor que la frecuencia natural no amortiguada. Un aumento en ζ reduciría la frecuencia natural amortiguada ω_d. Si ζ aumenta más de la unidad, la respuesta se vuelve sobreamortiguada y no oscilará.

2) *Caso críticamente amortiguado* (ζ=1): si los dos polos de *C*(*s*)/*R*(*s*) son casi iguales, el sistema se aproxima mediante uno críticamente amortiguado.

Para una entrada escalón unitario, *R*(*s*)=1/*s* y *C*(*s*) se escribe como

$$C(s) = \frac{\omega_n^2}{(s+\omega_n)^2 s}$$

La transformada inversa de Laplace de ecuación se encuentra como

$$\boxed{c(t) = 1 - e^{-\omega_n t}(1 + \omega_n t)}, \qquad para\ t \geq 0$$

3) *Caso sobreamortiguado* (ζ>1): en este caso, los dos polos de *C*(*s*)/*R*(*s*) son reales negativos y diferentes. Para una entrada escalón unitario, *R*(*s*)=1/*s* y *C*(*s*) se escriben como

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{\left(s + \zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1}\right)\left(s + \zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1}\right)s}$$

La transformada inversa de Laplace de la ecuación es:

$$\begin{aligned} c(t) = 1 &+ \frac{1}{2\sqrt{\zeta^2 - 1}\left(\zeta + \sqrt{\zeta^2 - 1}\right)} e^{-\left(\zeta + \sqrt{\zeta^2 - 1}\right)\omega_n t} \\ &- \frac{1}{2\sqrt{\zeta^2 - 1}\left(\zeta - \sqrt{\zeta^2 - 1}\right)} e^{-\left(\zeta - \sqrt{\zeta^2 - 1}\right)\omega_n t} \\ &= \boxed{1 + \frac{\omega_n}{2\sqrt{\zeta^2 - 1}}\left(\frac{e^{-s_1 t}}{s_1} - \frac{e^{-s_2 t}}{s_2}\right)}, \qquad para\ t \geq 0 \end{aligned}$$

donde $s_1 = \left(\zeta + \sqrt{\zeta^2 - 1}\right)\omega_n$ y $s_2 = \left(\zeta - \sqrt{\zeta^2 - 1}\right)\omega_n$. Por tanto, la respuesta *c*(*t*) incluye dos términos exponenciales que decaen.

Cuando ζ es apreciablemente mayor que la unidad, uno de los dos exponenciales que decaen disminuye mucho más rápido que el otro, por lo que el término exponencial que decae más rápido puede pasarse por alto (corresponde a una constante de tiempo más pequeña). Es decir, si -s_2 se localiza mucho más cerca del eje $j\omega$ que -s_1 (lo cual significa que $|s_2| \ll |s_1|$), para una solución aproximada se puede no considerar -s_1. Esto se permite debido a que el efecto de -s_1 en la respuesta es mucho más pequeño que el de -s_2, ya que el término que incluye s_1 en la ecuación anterior se descompone mucho más rápido que el término que tiene a s_2. Una vez desaparecido el término exponencial que decae más rápido, la respuesta es similar a la de un sistema de primer orden, y *C*(*s*)/*R*(*s*) se aproxima mediante:

$$\frac{C(s)}{R(s)} = \frac{\zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1}}{\left(s + \zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1}\right)s}$$

La respuesta del tiempo *c*(*t*) es, entonces,

$$c(t) = 1 - e^{-\left(\zeta - \sqrt{\zeta^2 - 1}\right)\omega_n t}, \qquad para\ t \geq 0$$

Esto proporciona una respuesta escalón unitario aproximada cuando uno de los polos de *C*(*s*)/*R*(*s*) puede pasarse por alto.

La Fig. 10 contiene una familia de curvas *c*(*t*) con diversos valores de ζ, donde la abscisa es la variable adimensional $\omega_n t$ *t*. Las curvas sólo son funciones de ζ y se obtienen a partir de las tres ecuaciones encerradas en un recuadro con anterioridad. El sistema descrito mediante estas ecuaciones estaba inicialmente en reposo.

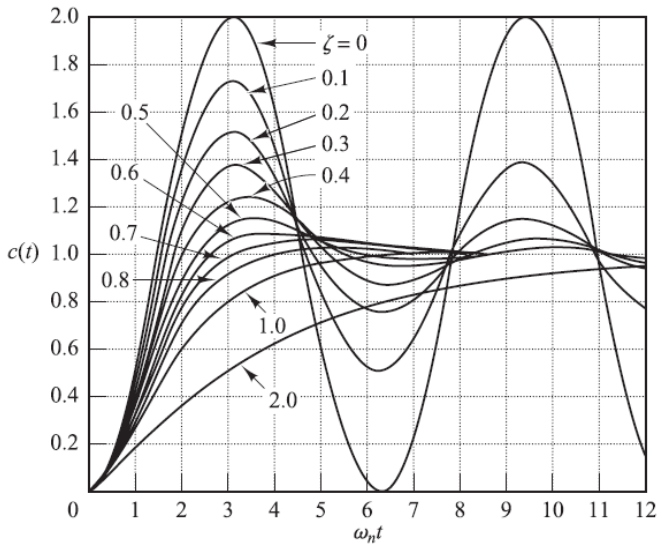


Fig. 10 Curvas de respuesta a escalón unitario del sistema mostrado en la Fig. 9

Obsérvese que los dos sistemas de segundo orden que tienen el mismo ζ pero diferente ω_n presentarán la misma sobreelongación y mostrarán el mismo patrón oscilatorio. Se dice que tales sistemas tienen la misma estabilidad relativa.

Es importante observar que, para los sistemas de segundo orden, cuyas funciones de transferencia en lazo cerrado son diferentes de las obtenidas mediante la ecuación principal de un sistema de segundo orden, las curvas de respuesta escalón se ven muy distintas de las que aparecen en la Fig. 10.

En la Fig. 10 se observa que un sistema subamortiguado con ζ entre 0.5 y 0.8 se acerca al valor final con mayor rapidez que un sistema críticamente amortiguado o sobreamortiguado. Entre los sistemas que responden sin oscilación, un sistema críticamente amortiguado presenta la respuesta más rápida. Un sistema sobreamortiguado siempre es lento para responder a las entradas.

Definiciones de las especificaciones de respuesta transitoria. En muchos casos prácticos, las características de desempeño deseadas del sistema de control se especifican en términos de cantidades en el dominio del tiempo. Los sistemas que pueden almacenar energía no responden instantáneamente y presentan respuestas transitorias cada vez que están sujetos a entradas o perturbaciones.

Con frecuencia, las características de desempeño de un sistema de control se especifican en términos de la respuesta transitoria para una entrada escalón unitario, puesto que esta es fácil de generar y es suficientemente drástica. (Si se conoce la respuesta a una entrada escalón, es matemáticamente posible calcular la respuesta para cualquier entrada.)

La respuesta transitoria de un sistema para una entrada escalón unitario depende de las condiciones iniciales. Por conveniencia al comparar respuestas transitorias de varios sistemas, es una práctica común usar la condición inicial estándar de que el sistema está en reposo al inicio, por lo cual la salida y todas las derivadas con respecto al tiempo son cero. De este modo, las características de respuesta se comparan con facilidad.

La respuesta transitoria de un sistema de control práctico muestra con frecuencia oscilaciones amortiguadas antes de alcanzar el estado estacionario. Al especificar las características de la respuesta transitoria de un sistema de control para una entrada escalón unitario, es común especificar lo siguiente:

Tiempo de retardo, t_d

Tiempo de subida, t_r

Tiempo pico, t_p

Sobreelongación, M_p

Tiempo de asentamiento, t_s

Estas especificaciones se definen enseguida y aparecen en forma gráfica en la Fig. 11.

Tiempo de retardo t_d: el tiempo de retardo es el tiempo requerido para que la respuesta alcance la primera vez la mitad del valor final.

Tiempo de subida, t_r: el tiempo de subida es el tiempo requerido para que la respuesta pase del 10 al 90%, del 5 al 95% o del 0 al 100% de su valor final. Para sistemas subamortiguados de segundo orden, por lo general se usa el tiempo de subida de 0 a 100%. Para sistemas sobreamortiguados, suele usarse el tiempo de levantamiento de 10 a 90%.

Tiempo pico, t_p: el tiempo pico es el tiempo requerido para que la respuesta alcance el primer pico de sobreelongación.

Sobreelongación máxima (porcentaje), M_p: la máxima sobreelongación es el máximo valor del pico de la curva de respuesta, medido a partir de la unidad. Si el valor final en estado estacionario de la respuesta es diferente de la unidad, es frecuente utilizar el porcentaje de sobreelongación máxima. Se define mediante

$$Porcentaje\ de\ sobreelongación\ máxima = \frac{c(t_p) - c(\infty)}{c(\infty)} * 100\%$$

La cantidad de sobreelongación máxima (en porcentaje) indica de manera directa la estabilidad relativa del sistema.

Tiempo de asentamiento, t_s: El tiempo de asentamiento es el tiempo que se requiere para que la curva de respuesta alcance un rango alrededor del valor final del tamaño especificado por el porcentaje absoluto del valor final (por lo general, de 2 o 5%). El tiempo de asentamiento se relaciona con la mayor constante de tiempo del sistema de control. Los objetivos del diseño del sistema en cuestión determinan qué criterio de error en porcentaje utilizar.

Las especificaciones en el dominio del tiempo que se han proporcionado son muy importantes, ya que casi todos los sistemas de control son sistemas en el dominio del tiempo; es decir, deben presentar respuestas de tiempo aceptables. (Esto significa que el sistema de control debe modificarse hasta que la respuesta transitoria sea satisfactoria).

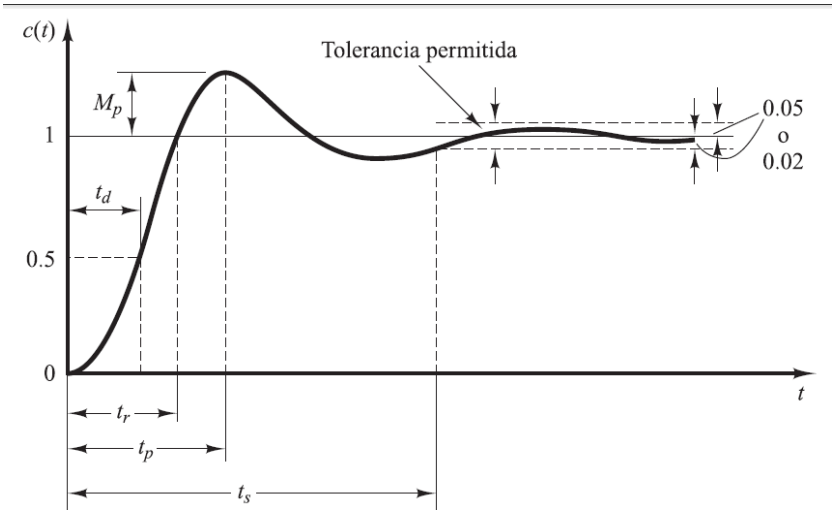


Fig. 11 Curva de respuesta a escalón unitario td, tr, tp, Mp y ts

Obsérvese que todas estas especificaciones no se aplican necesariamente a cualquier caso determinado. Por ejemplo, para un sistema

sobreamortiguado no se aplican los términos tiempo pico y sobreelongación máxima. (En los sistemas que producen errores en estado estacionario para entradas escalón, este error debe conservarse dentro de un nivel de porcentaje especificado (Ogata, 2010).

3. Metodología

Como se visualizó en el planteamiento general del capítulo uno por medio de un esquema, ahora se volverá a retomar el planteamiento general para volverle a dar un enfoque más dinámico y comprensible. Posteriormente se detallará en secciones el trabajo realizado para dar a cabo los resultados finales.

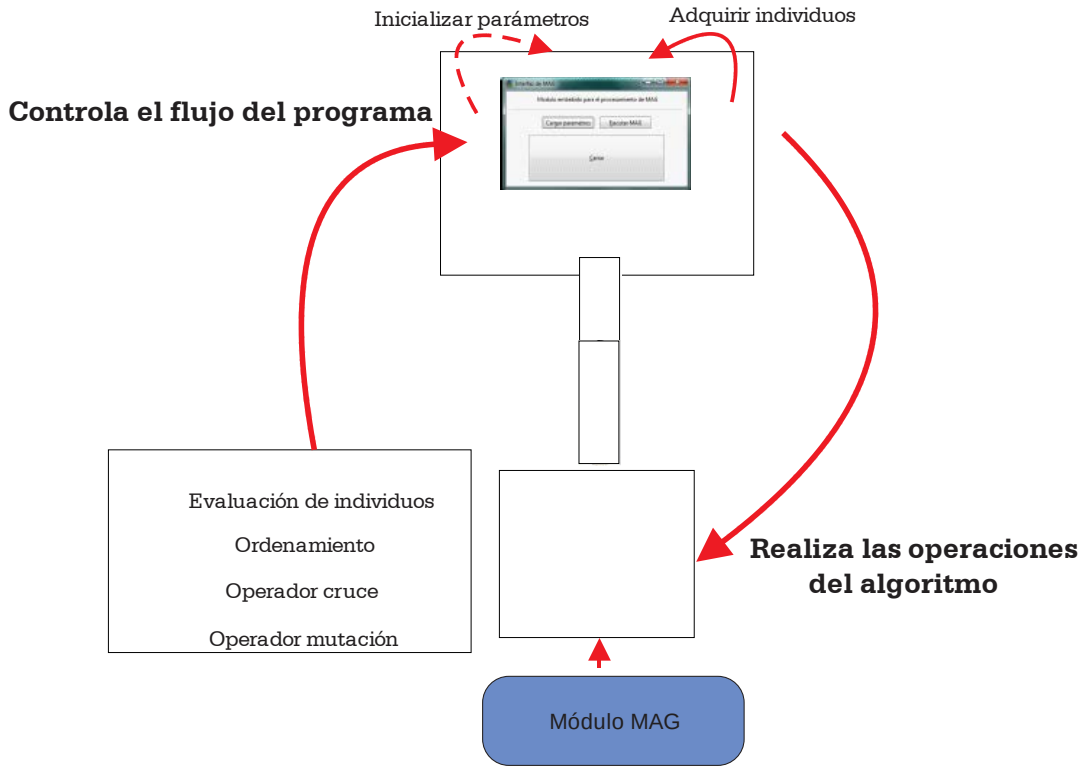


Fig. 12 Planteamiento general iterativo

El flujo del proceso completo es el mostrado en la Fig. 12, donde se aprecia un flujo iterativo. Se empieza con la inicialización de los parámetros que solo ocurre al principio del algoritmo, y ocurre en la interfaz gráfica en la PC. En la misma interfaz se mandan llamar a los individuos que están en el microprocesador, en el se ejecutan la evaluación de los individuos (función de desempeño), el ordenamiento de mejor al peor y las operaciones de mutación y cruce, son enviados de vuelta a la interfaz para que los almacene y la interfaz

es la encargada de llevar el proceso, es decir, en ésta se tiene el control del algoritmo en sí. Y se vuelve iterativo hasta que según los parámetros iniciales, el número de generaciones haya terminado.

Después de tener un conocimiento general de lo que se tuvo que desarrollar, la metodología de como se realizó cada proceso está en ésta sección.

Como desarrollo general a seguir para la implementación de un módulo de micro algoritmos genéticos es necesario realizar una serie de pasos para el manejo de tema y así la ejecución del mismo en un sistema embebido. Como ya se había mostrado antes el esquema general es el mostrado en la Fig. 13.

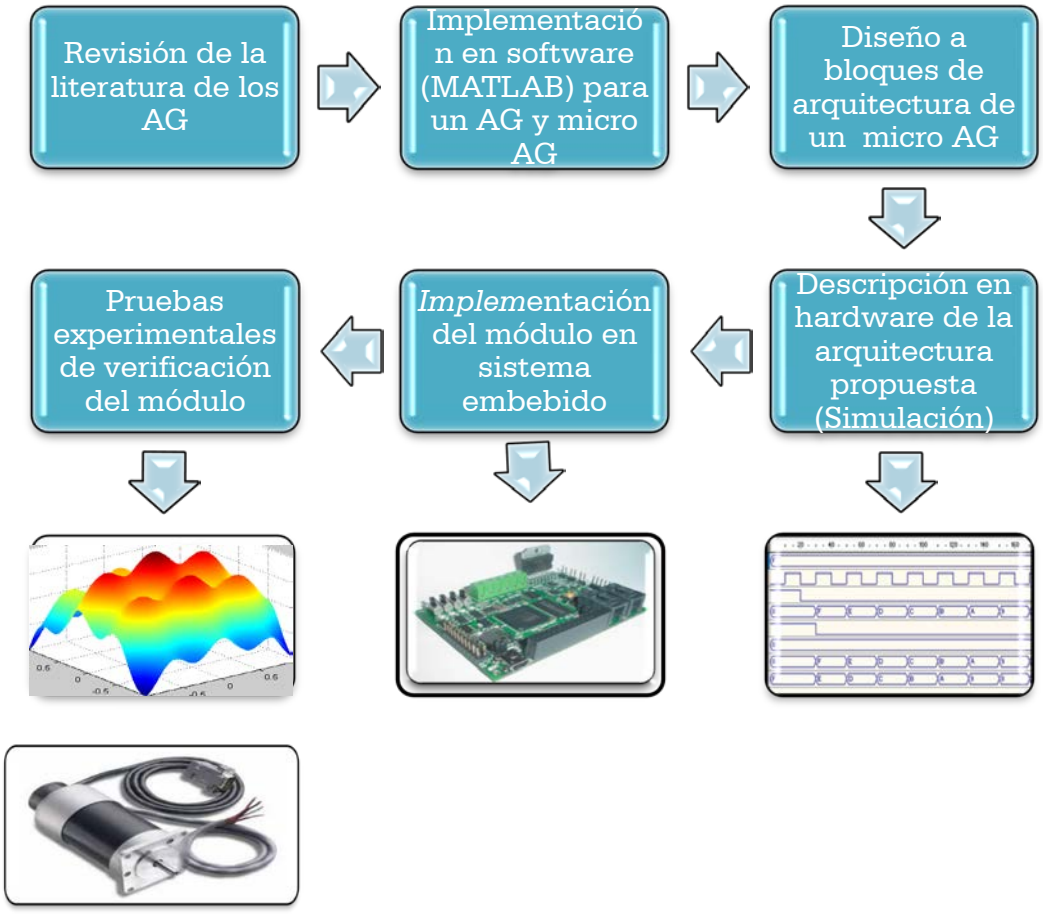


Fig. 13 Esquema general de la metodología

El primer módulo del esquema es documentarse, es decir, conocer del tema, los conceptos, el proceso, ¿Cómo funcionan?, ¿Para qué?, etc. Es necesario leer y familiarizarse acerca del tema para poder desarrollar posteriormente el proceso que lleva a cabo el algoritmo en código y después implementarlo en el sistema embebido.

Al conocer el tema es necesario primero saber los orígenes, en este caso son los algoritmos genéticos estándares, al entender los mismos y cómo funcionan podemos realizar modificaciones, innovaciones, evoluciones del algoritmo para tratar de mejorarlo, aplicando así los MAG.

3.1 Implementación en software (MATLAB) para un AG y MAG

Para afianzar los conocimientos del proceso iterativo del algoritmo fue necesario implementarlo primero en software y así comprender por completo el algoritmo en el medio de programación. La sección 3.1.1 y 3.1.2 describen los parámetros, consideraciones y valores que se utilizaron para la realización en éste trabajo. Además de ser más claro para la comprensión del mismo.

3.1.1 Desarrollo general de un algoritmo genético estándar

Un algoritmo genético estándar (AGE) funciona con una población de individuos grande (en este caso se usaron 100 individuos), que son generados aleatoriamente, después se evalúa cada uno de ellos, teniendo una función que es la que determina el valor de desempeño (función de Bohachevsky), de cada individuo según deseemos sea un máximo o un mínimo (un máximo). Posteriormente se elige a los mejores en la etapa de selección, en este caso por medio de la selección torneo, en la cual se elegirán aleatoriamente dos individuos, se comparan y el que tenga el mejor desempeño pasa a la siguiente generación y se realizarán los torneos necesarios hasta llenar los 100 individuos.

Fig. 14 Diagrama general del proceso de un AGE

Teniendo ahora una nueva generación con los mejores individuos, se aplica el operador de cruce, donde aleatoriamente se elegirán a dos individuos (padres) y aleatoriamente se decidirá un punto de cruza para el intercambio de información, es decir de bits, creando un nuevo individuo (descendencia) y así crear una nueva generación. Solo se emplea el operador de cruza si con un random de 0 a 1 (usando 0.6), entrará al rango de probabilidad de cruza que se propuso, sino este operador no se aplica y pasa solo uno de los dos padres intacto a la siguiente generación. A esta nueva generación se le emplea el operador de mutación, donde si está dentro del rango de probabilidad (0.1), se realiza la mutación y si no pasa indemne el individuo a la nueva generación. Si se le aplica el operador, se elige aleatoriamente un punto de cruza donde se cambio el valor del bit seleccionado, si es 1 se vuelve 0 y viceversa. Se pueden apreciar los siguientes pasos reducidos en la Fig. 14 y Fig. 15.

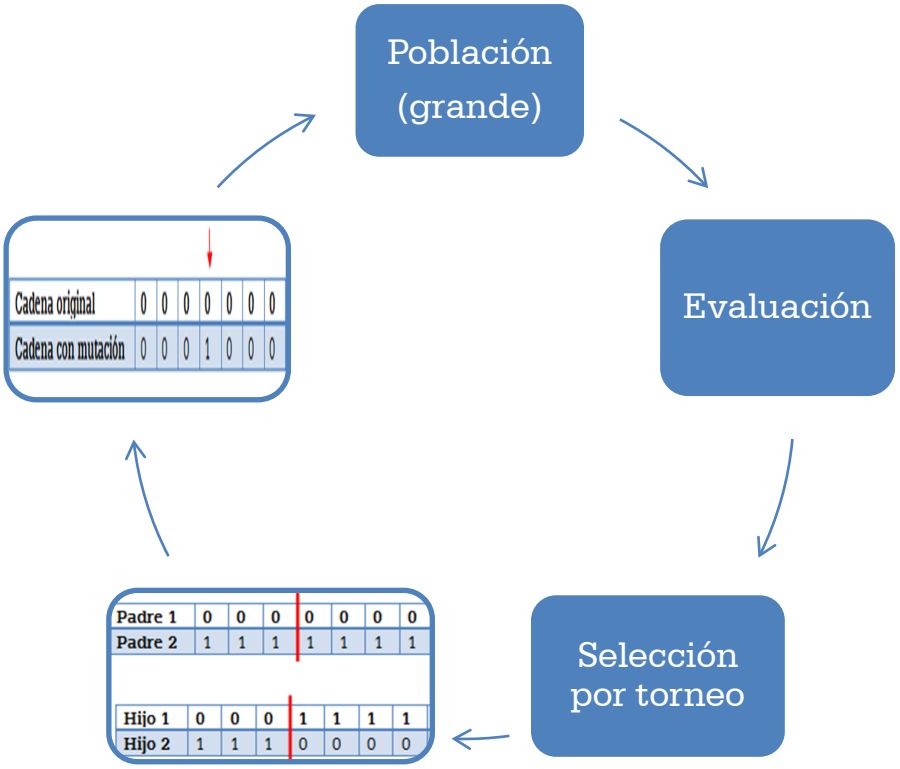


Fig. 15 Proceso del desarrollo del algoritmo genético

Entendiendo este proceso se prosiguió a realizar lo que es un micro algoritmo genético, el cual es muy parecido a uno estándar, sin embargo, difiere en la cantidad de individuos (4 individuos para este trabajo) y además los operadores no se emplean de la misma forma.

Para ser precisos, se desarrolló un micro algoritmo genético con cuatro individuos, estos son distribuidos equidistantemente según sean los rangos elegidos en el cual pueda contener la solución, ya que si se generan aleatoriamente se puede caer en una solución errónea o local, es decir convergir prematuramente. También se les evalúa su desempeño, con alguna función o parámetro que se quiera ser evaluado. En este caso no se puede realizar una selección por torneo debido a que solamente se tiene cuatro individuos y se llegaría a una solución prematura además de equivocada, en lugar de ello se realiza un ordenamiento por burbuja (ordenándose del mejor al

peor individuo o viceversa dependiendo la situación deseada). Como proceso iterativo se puede apreciar a grandes rasgos en la Fig. 16.

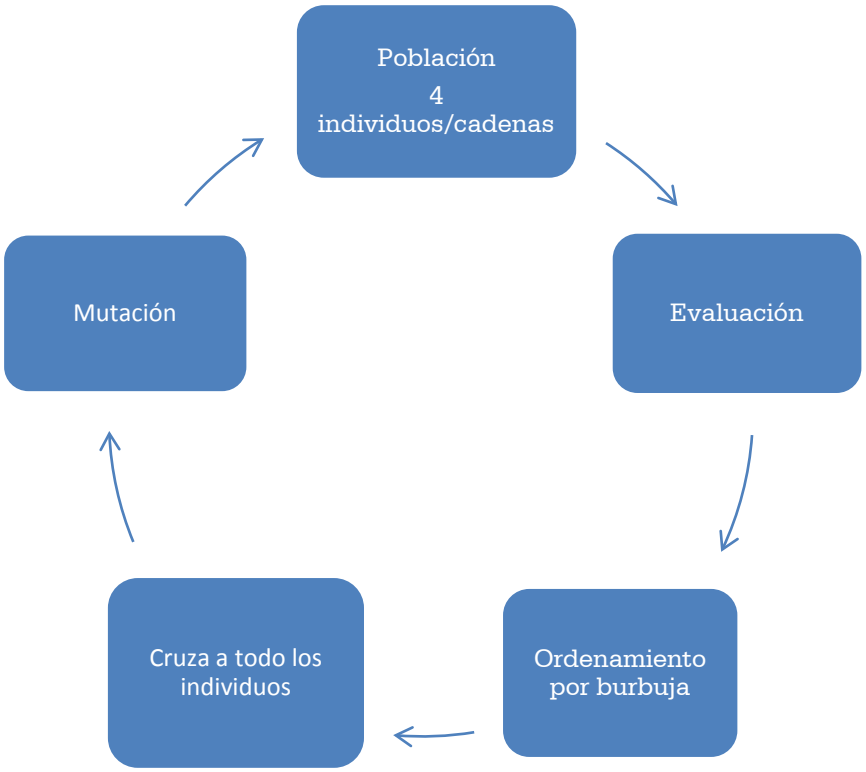


Fig. 16 Diagrama de un algoritmo micro-genético

Después en el operador cruza, al mejor individuo se le deja intacto y pasa directo a la siguiente generación, al segundo individuo de la siguiente generación se crea con la cruza del primero y el segundo mejores, para el tercero es la cruza entre el primero y el tercero de los mejores y el cuarto se crea aleatoriamente para generar diversidad de soluciones (Fig. 17). Y finalmente solo se aplicará el operador de cruza cuando de manera aleatoria se elija un rango entre 0-1, si entra en el rango de probabilidad de mutación, al individuo se le aplica el operador si no pasa intacto. Y así se crea una nueva generación en la cual se volverá la generación inicial y así el proceso es iterativo hasta el punto que se cumpla el número de generaciones.

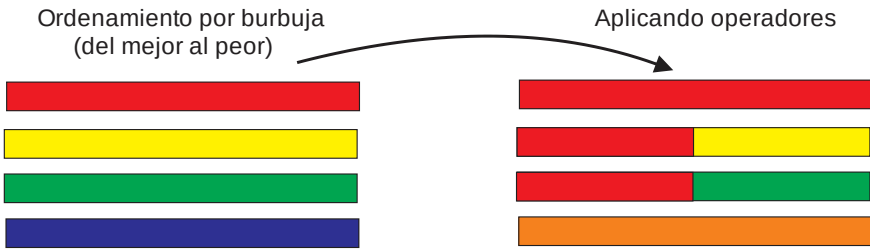


Fig. 17 Descripción gráfica de la etapa de los operadores para MAG

3.2 Planteamiento en de la estructura en software

Además de conocer y entender en qué consiste un algoritmo genético, es indispensable desarrollar el código de un AGE, así no solamente se quedaría en teoría, se llevaría a cabo a la práctica para el completo entendimiento del algoritmo. En las secciones anteriores se describe cómo es que se desarrollaron los códigos para un AGE y un MAG.

A grandes rasgos lo que comprende el código para un AGE y un MAG son los siguientes aspectos:

- Inicialización de parámetros
- Creación de la población inicial
- Evaluación de individuos
- Operador de selección/ ordenamiento burbuja
- Operador cruce
- Operador mutación
- Almacenamiento del mejor individuo
- Asignación de población final a población inicial
- Reinicio del proceso

En ese orden es como se llevó a cabo la secuencia del código para un AGE y la única diferencia del MAG es que el lugar de aplicar el operador de selección se utiliza el de ordenamiento burbuja.

3.3 Diseño a bloques de arquitectura de un MAG

Con el planteamiento general se tiene una noción de lo que se debe realizar de manera general y a partir de ahí buscar la manera específica en cuanto a cada parte para ser programada y tener todo el conjunto de partes. Una de ellas es el diseño a bloques de las operaciones que realiza el algoritmo en sí, sin tener todo el desarrollo iterativo del MAG, de ello se encargará la interfaz gráfica. El diseño a bloques quedó de la forma en que se muestra en la Fig. 18, explicada a continuación.

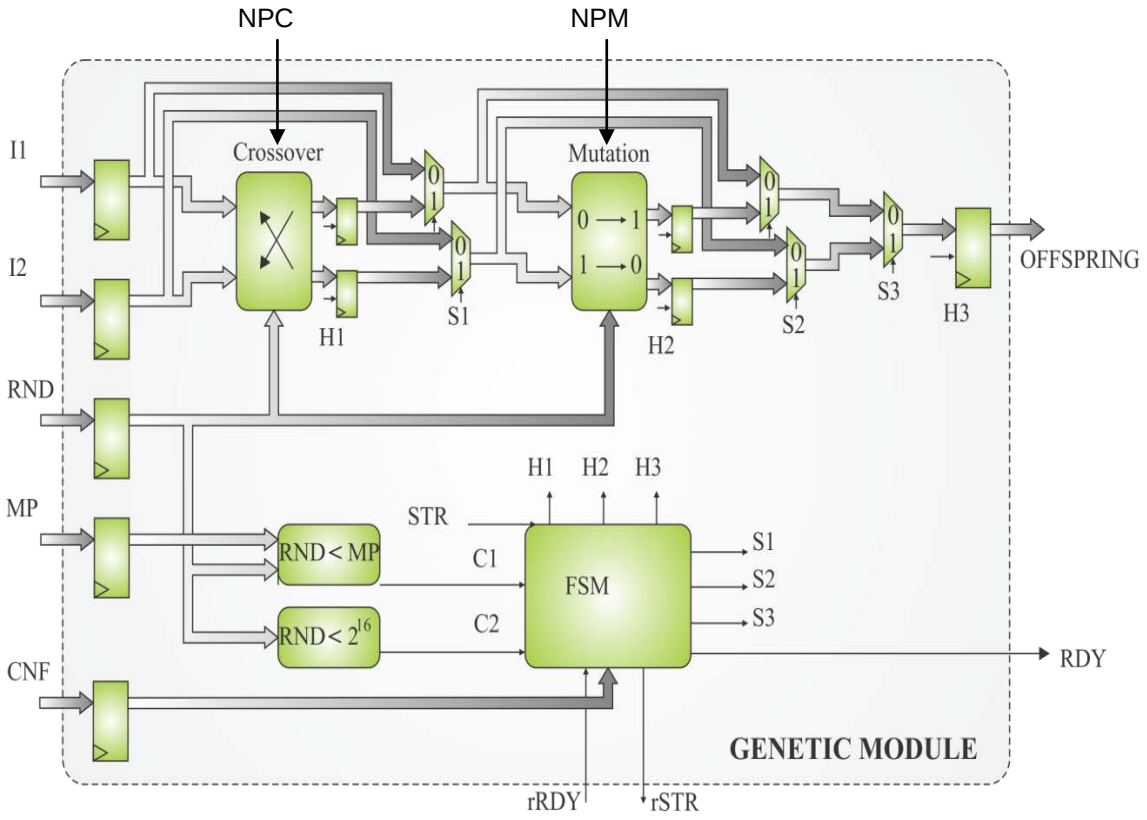


Fig. 18 Propuesta a bloques

En la Fig. 18 se tienen cinco entradas principales y una salida principal. I1 e I2 indican los individuos que serán modificados, es decir, se les aplicará un proceso de MAG. Mientras que RND es indispensable para poder generar los números aleatorios, pues la tarjeta SPARTAN no tiene generador de aleatorios.

MP es la probabilidad de mutación que es un parámetro que se introduce por el usuario según desee, de 0 a 1, 0 indica que no queremos que se muten y el 1 que a I1 e I2 se les aplique el operador de mutación. Finalmente CNF es la configuración que se desea aplicar según el proceso, éste permite el control de la máquina de estados (FSM) para poder determinar si se aplica cruza, mutación o ninguna según sea el caso. También es importante mencionar que rRDY es una señal que indica que el generador de aleatorios está listo y el rSTR da la señal de mandar pedir un aleatorio, RDY solo dice que la máquina de estados ha terminado de realizar su función.

Otras dos señales muy importantes son NPC y NPM, número o cantidad de puntos de cruce y número de puntos de mutación, respectivamente. Generalmente se ha utilizado un solo punto de cruce y mutación. Sin embargo, como complemento hacia éste módulo de MAG se realizará la opción de hasta 3 puntos de cruce y 4 puntos de mutación. Así que ahora una variante al diseño es que el usuario pueda elegir entre varios puntos de cruza y de mutación.

Padre 1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Padre 2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Hijo 1	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
Hijo 2	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**

Fig. 19 Cantidad de puntos de cruce a elegir

La Fig. 19 muestra que el número de puntos posibles será de tres, haciendo un ejemplo sencillo para su mejor entendimiento. Recalcando que solo serán tres puntos de cruza por individuo, el usuario solo controlará la cantidad de puntos de cruza que desea, aleatoriamente los puntos de cruce se realizarán.

También la cantidad de puntos de mutación será variable, según elija el usuario y aleatoriamente serán elegidos para el intercambio de bit. En la Fig. 20 se ilustra que serán posibles hasta cuatro puntos de mutación como máximo.

Cadena original	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Cadena con mutación	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Fig. 20 Cantidad de puntos de mutación

Haciendo un enfoque a los bloques de cruce y mutación, la dinámica es según el número de puntos deseados para realizar en los operadores de cruce y mutación.

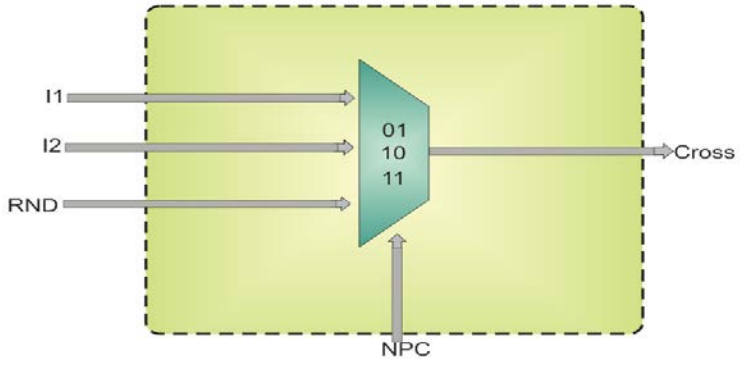


Fig. 21 Módulo del operador de cruza

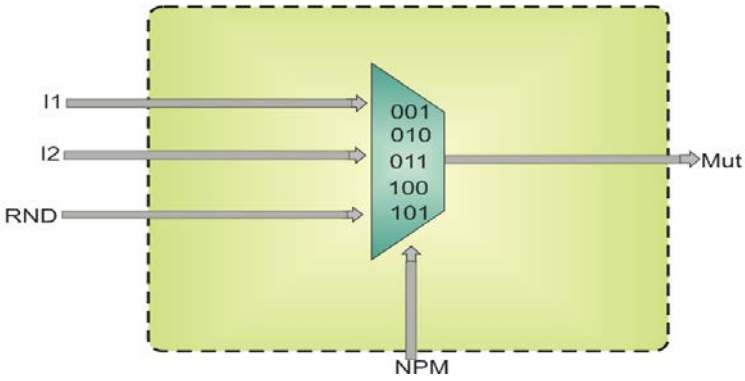


Fig. 22 Módulo del operador mutación

En la Fig. 21 se contempla el bloque del operador de cruza, el cual es modificado según la configuración de NPC que se es seleccionada por medio de una macro de la librería desarrollada en hardware y solamente es posible seleccionar hasta un número de puntos de cruza de tres. De igual forma para la Fig. 22, NPM es la configuración para poder determinar hasta un máximo de cinco puntos de cruza. En ambas imágenes se tienen las señales del individuo uno (I1), del individuo dos (I2) para que a éstos se les aplique dicho operados y el número aleatoria (RND) para poder hacer los el punto de cruza y/o mutación.

3.4 Descripción en hardware de la arquitectura propuesta

En el FPGA se tienen varios módulos tal como el del microprocesador (xq16v7) y el módulo MAG, solo para hacerlo de manera ilustrativo se pueden observar en la Fig. 23 como se utilizan varios módulos o se cuentan con varios módulos principales y todos son manipulados por medio de un bus y la dirección que le corresponde a cada uno de ellos. Algunos otros módulos que se utilizan son el del PID y el de comunicación USB.

Específicamente el módulo a tratar es el del MAG. El módulo MAG contiene varios sub-módulos que logran en conjunto realizar las operaciones del micro algoritmo genético, los elementos principales son el sub-módulo RND, que es el encargado de realizar los números aleatorios, el sub-módulo FSM, que es la máquina de estados, encargada de realizar todo el proceso y darle orden y seguimiento al módulo de MAG uniendo todos los otros sub-módulos.

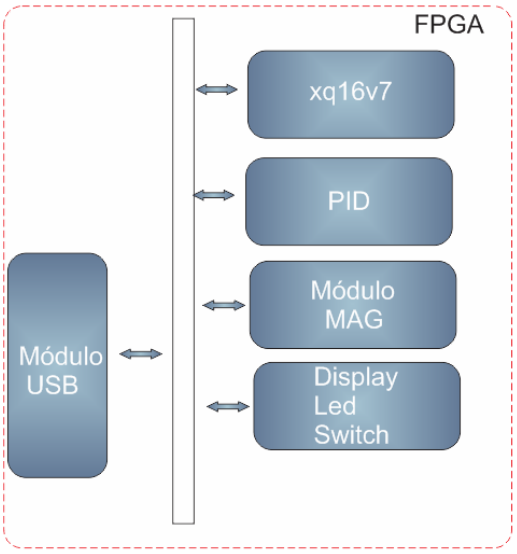


Fig. 23 Algunos módulos que integran el microprocesador

El módulo MAG interactúa con el del microprocesador por medio de la dirección, es decir, que el microprocesador sabe que debe mandar llamar al MAG debido a su dirección (ADDR) y solo hay un dato de entrada (DI) y uno de salida (DO). La Fig. 24 muestra ilustrativamente la conexión.

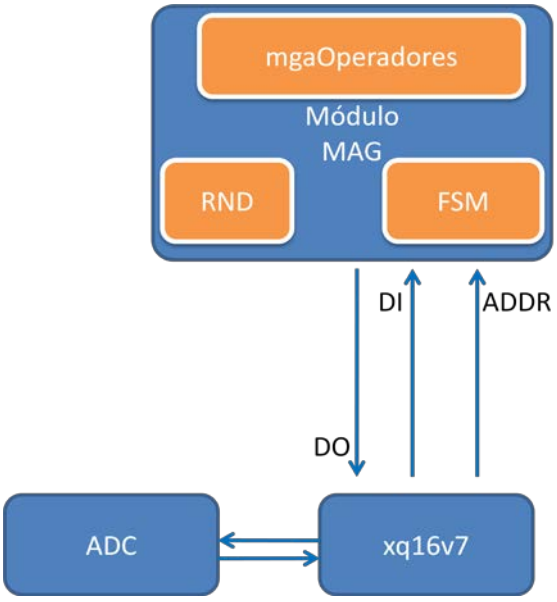


Fig. 24 Módulo MAG interactuando con el microprocesador

En DI entran las cinco entradas principales antes explicadas del módulo MAG, que aunque sea una sola, está perfectamente direccionada para cada una (I1, I2, CNF, MP, RND). Y la salida es solamente una (OFFSPRING). Además en la figura 18 también muestra una interacción con el ADC, el cual se encarga de convertir una señal analógica a digital.

3.5 *Implem*entación del módulo en sistema embebido

Primero es necesario entender a grandes rasgos como se está trabajando en el sistema embebido. Anteriormente se ha mostrado el planteamiento general del sistema embebido siendo iterativo o dinámico, en la Fig. 25 muestra el sistema embebido que está integrado por distintos módulos comunicados a través de un bus, ya no de manera iterativa. Cada módulo tiene un dato de entrada y un dato de salida. Sin embargo, aunque solo maneje un dato de entrada y salida, los datos de cada módulo como el del USB, PID, el de MAG son definidos por una dirección única.

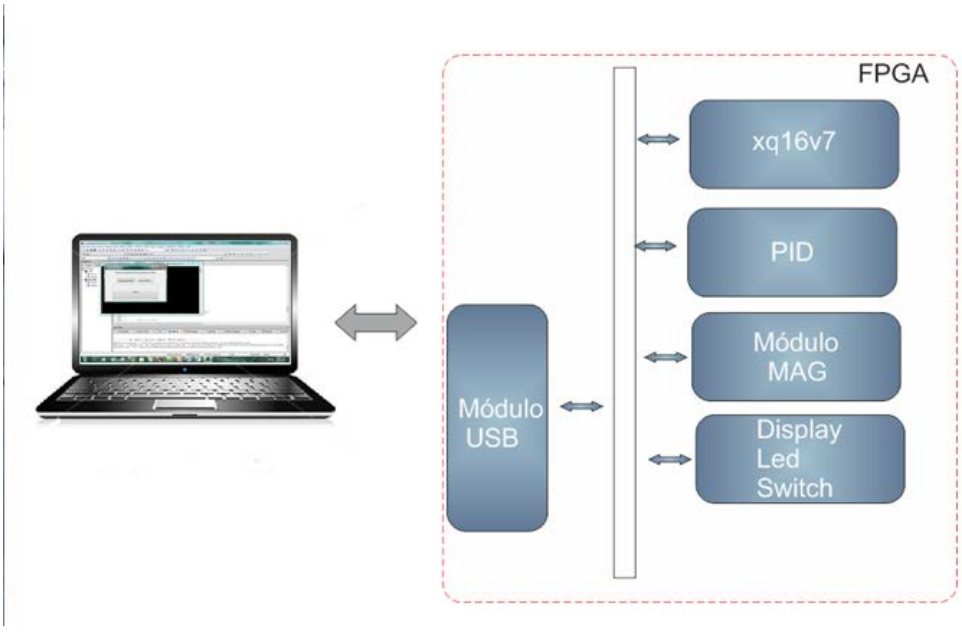


Fig. 25 Sistema embebido integrado simple

El sistema interactúa con el módulo y la PC por medio de una interfaz gráfica y a través de la conexión USB. Es importante volver a mencionar que todo se comunica/controla por medio de un bus de direcciones.

```
Register map:

   Name    Address Type    Description
   SETUP   0x00    R/W     Module setup
   I1      0x01    R/W     Individual 1 for genetic operation
   I2      0x02    R/W     Individual 2 for genetic operation
   OFFSP   0x03    R       Offspring generated from I1 and I2
   MP      0x04    R/W     Mutation probability
   RND     0x05    R       Pseudo random number generated


Register details:

 SETUP:    15-12  11  10  9    8  7     6  5      4    3   2   1    0   bit
           0000   -- NPM --  --NPC--  --BUSY--  rSTR  ----CNF----  STR   |
```

Fig. 26 Registros principales de la librería de MAG

Dentro del microprocesador se desarrolló un código para la interacción de los módulos con el sistema. Para el módulo MAG se define una serie de registros, como se muestra en la Fig. 26 en la parte superior y cada registro tiene un puerto de salida, además, de que el conjunto de registros tienen una dirección base (Fig. 27). En la parte inferior de la Fig. 26 se tiene como es que están configurados los registros, cada registro tiene una dirección que permite activar los bits correspondientes según su posición. (Fig. 28)

```
#ifndef LIBRARY_CODE
/* Port definition */
port (uGM_ctrl)  = 0x00 + uGM_BASE_ADDRESS;
port (uGM_I1)    = 0x01 + uGM_BASE_ADDRESS;
port (uGM_I2)    = 0x02 + uGM_BASE_ADDRESS;
port (uGM_Offsp) = 0x03 + uGM_BASE_ADDRESS;
port (uGM_MP)    = 0x04 + uGM_BASE_ADDRESS;
port (uGM_Random)= 0x05 + uGM_BASE_ADDRESS;
#endif
```

Fig. 27 Dirección de los puertos de salida de los registros

Por ello son necesarias las banderas para poder manipular el control de los bloques de la arquitectura en VHDL, como por ejemplo la de la máquina de estados. Las banderas de NPC y NPM nos permiten elegir la cantidad de puntos de cruza o mutación según se elija. Las demás son banderas de control para la máquina de estados (Fig. 28), como lo es pedir un aleatorio (uGM_RND), o el seleccionar que solo se realice mutación (uGM_M), cruza (uGM_C) o ambas (uGM_CM). Como se dijo con anterioridad, también se muestran las direcciones de cada bandera que activan un bit, cambiando la configuración del módulo como se aprecia en la parte inferior de la Fig. 26.

```
/* Control flags */
#define uGM_STR      0x01
#define uGM_C        0x02
#define uGM_M        0x04
#define uGM_CM       0x08
#define uGM_RND      0x10
#define UGM_MSK      0x20
#define RND_MSK      0x40
#define NPC1         0x80
#define NPC2         0x100
#define NPC3         0x180
#define NPM1         0x200
#define NPM2         0x400
#define NPM3         0x600
#define NPM4         0x800
```

Fig. 28 Algunas banderas de la librería de MAG

Así es como se tiene un control para la configuración según se desee realizar o utilizar cada una de esas banderas. Se le da un control a la máquina de estados y un seguimiento correcto al proceso del algoritmo.

```
/* Macro definitions */
#define Set_Crossover1(a,b)  uGM_ctrl = uGM_C|NPC1;\
                             uGM_I1   = a;\
                             uGM_I2   = b

#define Set_Crossover2(a,b)  uGM_ctrl = uGM_C|NPC2;\
                             uGM_I1   = a;\
                             uGM_I2   = b

#define Set_Crossover3(a,b)  uGM_ctrl = uGM_C|NPC3;\
                             uGM_I1   = a;\
                             uGM_I2   = b

#define Set_Mutation1(a,x)   uGM_ctrl = uGM_M|NPM1;\
                             uGM_I1   = a;\
                             uGM_MP   = x;\
                             uGM_I2   = 0
```

Fig. 29 Definición de algunos macros de la librería de MAG

Se definieron macros, que son una clase de etiquetas que representan una secuencia de código, es decir, al llamar a la macro que es como se sustituyera una serie de instrucciones, desde el procesador se mandan llamar las macros que dicen “ejecuta” las instrucciones correspondientes. Por ejemplo, en la Fig. 29 se tiene el *macro Set_Crossover1(a, b)* la cual al escribirla en el código, será reemplazada por toda su secuencia de instrucciones que son las que se encuentran del lado derecho de la macro (enfrente). Explicando las instrucciones dicen que la configuración (uGM_ctrl) cambiará a alto para ejecución de la cruza de un solo punto (NPC1) y que lo que se haya introducido como (*a*) se convertirá en el primer individuo y el segundo individuo será (*b*).

Respecto al trabajo desarrollado en PC, se desarrolló una interfaz gráfica de usuario en c++ y librerías JTK, ésta permite la comunicación en línea con el microprocesador. Consta de tres botones: cargar parámetros, ejecutar MAG y cerrar mostrados en la Fig. 30. Primero se define en la interfaz una secuencia de comandos que controlan el flujo del programa en el sistema embebido. La computadora controla las acciones que se ejecutarán en el procesador y el microprocesador desempeña las operaciones del mismo algoritmo, logrando así el proceso iterativo del MAG.

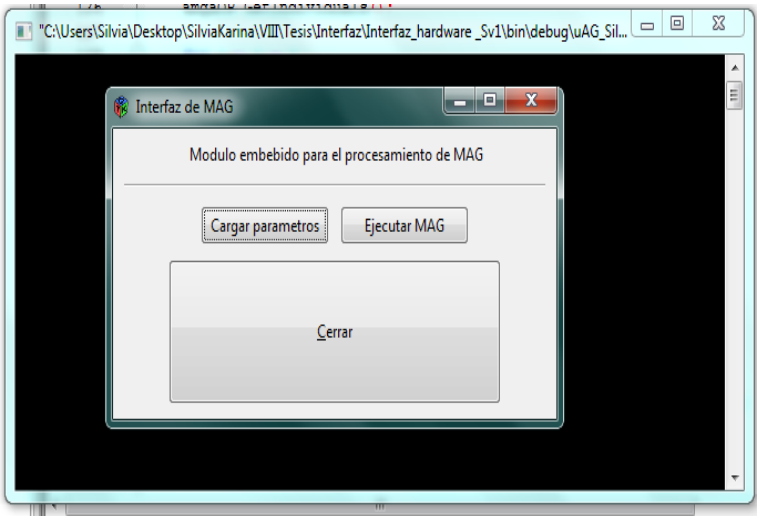


Fig. 30 Interfaz gráfica

Al presionar le botón de *cargar parámetros* lo que hace es adquirir la cantidad de iteraciones que se van a realizar, los rangos en los que puede estar la solución y la probabilidad de mutación. Posteriormente se presiona el botón de *ejecutar MAG*, donde primero solicita los individuos iniciales al microprocesador diciendo que los mande a la interfaz, como ya se había mencionado son cuatro individuos distribuidos equidistantemente (ya están fijos en el microprocesador). En la interfaz se decodifican los individuos recibidos y son evaluados por la función objetivo (función de Bohachevsky), regresando el desempeño al microprocesador para que ahí se realice el orden de mayor a menor de los individuos y después se aplican los operadores. El proceso se realizará hasta que el número degeneraciones haya finalizado. Es así como el programa, la interfaz, controla el flujo del sistema embebido.

En el compilador del microprocesador se definen los individuos distribuidos equidistantemente, como los registros del microprocesador es de 16 bit, se distribuye 65535 entre los cuatro individuos, por ellos es que es equidistante y posteriormente cuando se manden a la PC se decodificaran en los rangos propuestos como parámetros.

En la Fig. 31 se muestra la interfaz del compilador del microprocesador, donde se maneja la librería del módulo desarrollado y se utilizan la macros. Aquí se encuentran los individuos iniciales, se lleva a cabo el ordenamiento del desempeño del mejor al peor, también se realizan los operadores de cruza y mutación.

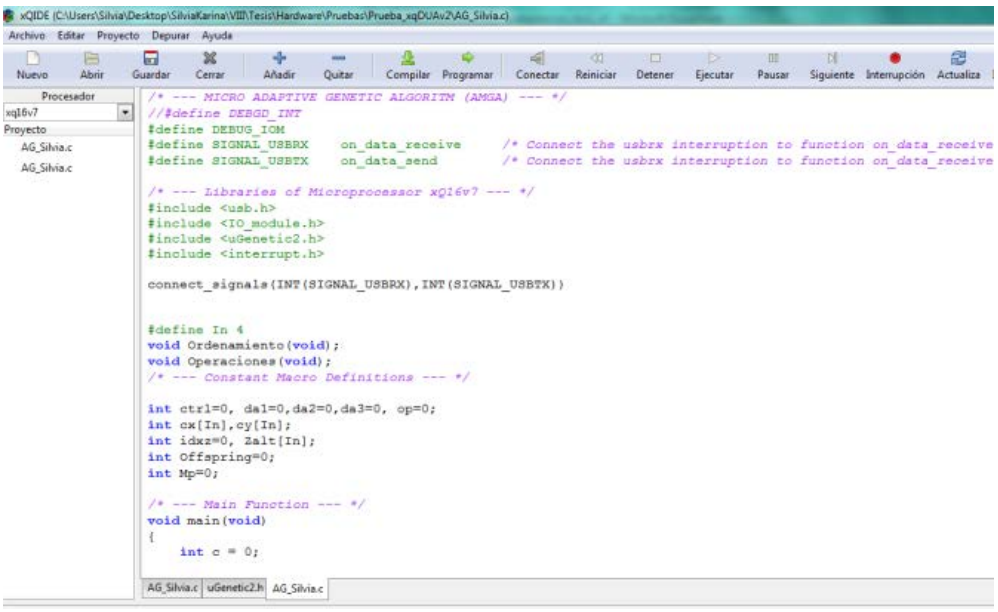


Fig. 31 Compilador del microprocesador

Resumiendo, a grandes rasgos la PC realiza los cálculos del desempeño de los individuos y envía el desempeño al microprocesador para que realice todo el proceso, el valor del desempeño de la función de Bohachevsky fue necesario realizarlo en la interfaz pues cuenta de operaciones complejas que no puede realizar el microprocesador como son senos y cosenos, así solamente el micro desarrolla todo el proceso del algoritmo y se tiene una comunicación en línea. Pero para la función de desempeño de la sintonización de control del PID, si fue posible colocarla en el microprocesador. La Fig. 32 muestra la conexión de PC con la tarjeta por medio del USB.

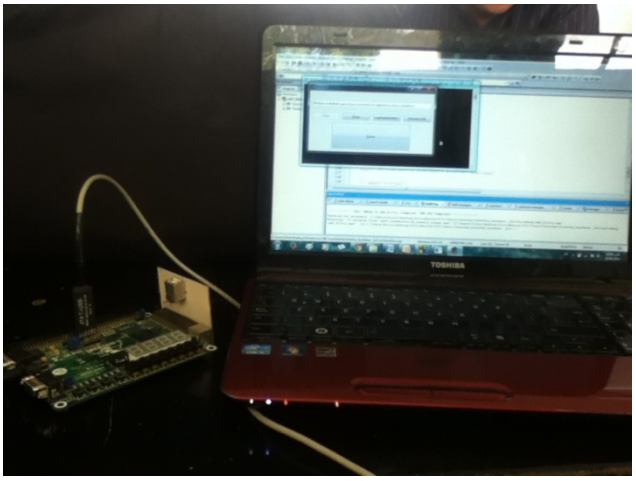


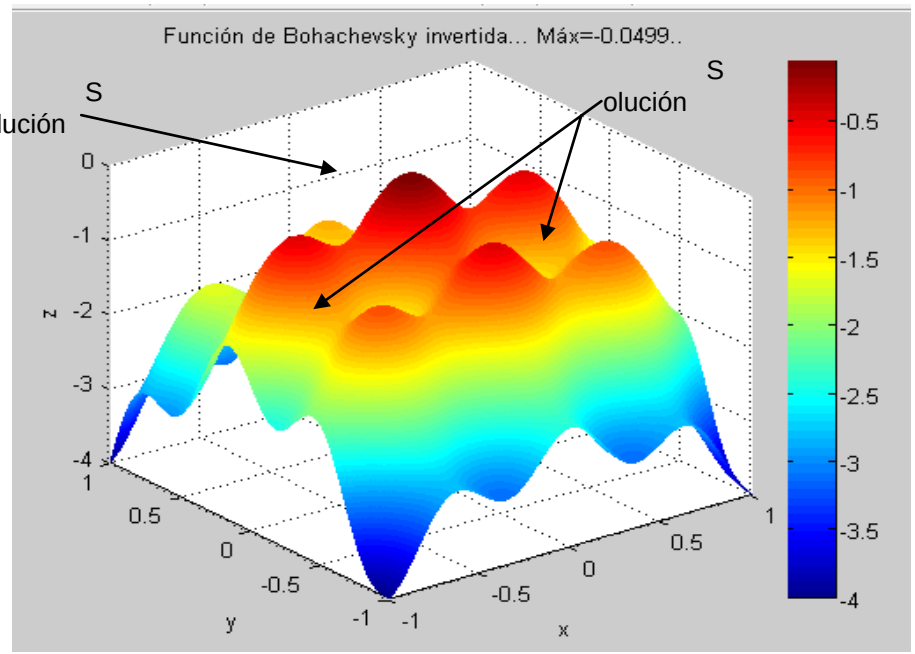
Fig. 32 Comunicación pc-microprocesador en físico

3.6 Casos de estudio y los pasos que se realizaron para efectuar las Pruebas experimentales de verificación del módulo de MAG

A continuación las dos sub-secciones siguientes mostrarán como fueron realizadas las pruebas para validar el módulo de MAG. Se explicará lo que se hizo para ejecutar la experimentación de cada caso de estudio como lo son el de la función de Bohachevsky y el control de un servomotor. Hasta la sección de resultados del capítulo 4, se darán a conocer sus resultados de cada uno. Enfatizando nuevamente, en estas secciones solo se mostraran los pasos que se realizaron para el desarrollo de cada prueba.

3.6.1 Primer caso de estudio: función Bohachevsky

Se escogió la función de Bohachevsky (Fig. 33), que permite visualizar como en una función no lineal (tridimensional), se puede tener una gran cantidad de soluciones locales y únicamente una solución global, que es el valor que deseamos encontrar. La solución global es destacada en color rojo intenso y las locales en un amarillo-anaranjado.



$$f(x,y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\sin(3\pi y) + 0.7$$

Fig. 33 Función de Bohachevsky

Las variables X y Y son las que deben ser óptimas para encontrar el punto global máximo (Z). Al hablar del rango de la posible solución se refiere al rango que puede estar X y Y para encontrar su solución óptima de cada una de ellas. En la Fig. 33 el rango para X es de -1 a 1, al igual que para Y. no siempre son iguales los rangos, en este caso resulto ser el mismo. La función de desempeño que es la que nos permitirá encontrar el punto máximo global es la función mostrada en la parte inferior de la Fig. 33. En ella se hace claro las variables X y Y y la función f(x,y) es igual a Z, la altura en este caso.

Al haber aclarado cual es el caso de estudio, la función de Bohachevsky. Primeramente cabe señalar que como parámetros iniciales se realizaron 150 iteraciones por cada prueba realizada y se efectuaron más de 200 pruebas.

Para validar el módulo del MAG por medio de la función de Bohachevsky, se hicieron 20 pruebas por cada modificación en el código, es

decir, se realizaron 20 pruebas con un punto de cruce y un punto de mutación, otras 20 pruebas con dos puntos de cruza y cuatro puntos de mutación, etc. También se realizaron pruebas con la probabilidad de mutación fija y con la probabilidad variable.

Se presenta en la Fig. 34 el sistema con las pruebas que se realizaron para la función de Bohachevsky, donde se tiene la PC, la tarjeta FPGA (microprocesador) y la conexión por medio de USB

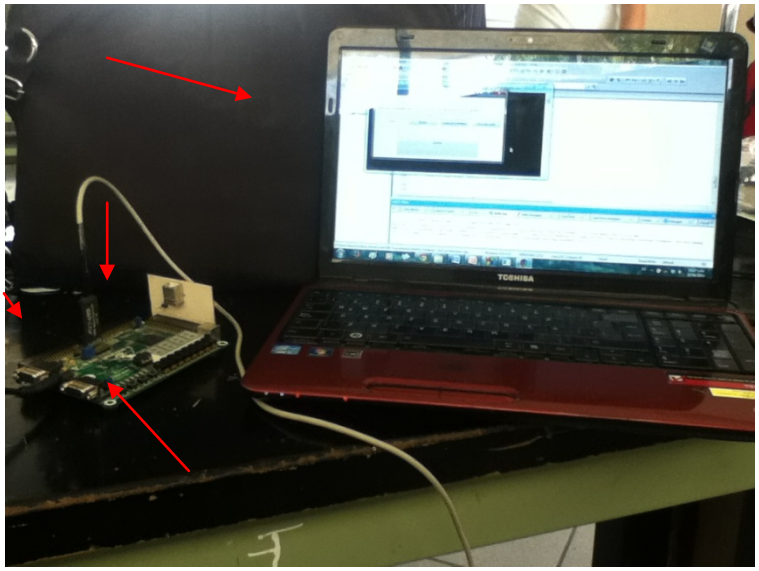


Fig. 34 Sistema para la función de Bohachevsky

Fue necesario verificar cada paso del método para estar seguro que el resultado final sería el correcto. Por lo tanto, las primeras pruebas fueron que el micro estuviera realizando bien el ordenamiento del desempeño de los cuatro individuos. Por ello, se utilizaron los switches y los Led para monitorear el proceso y asegurar que lo realizará adecuadamente.

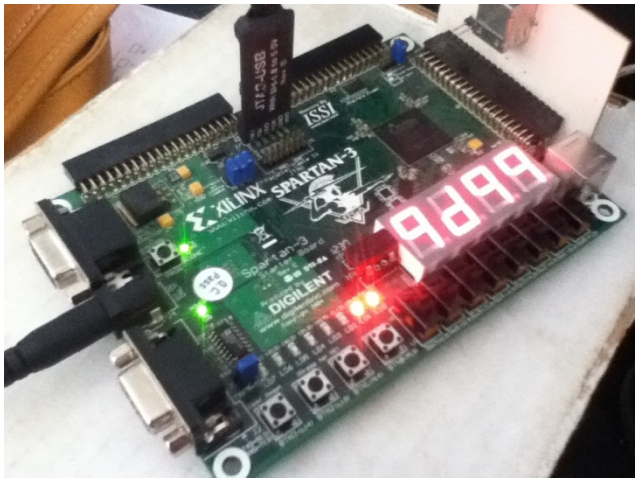


Fig. 35 Monitoreo del ordenamiento burbuja con Led y switch

Para la realización de las diferentes pruebas, los datos que se cambiaron fueron los señalados en la Fig. 36.

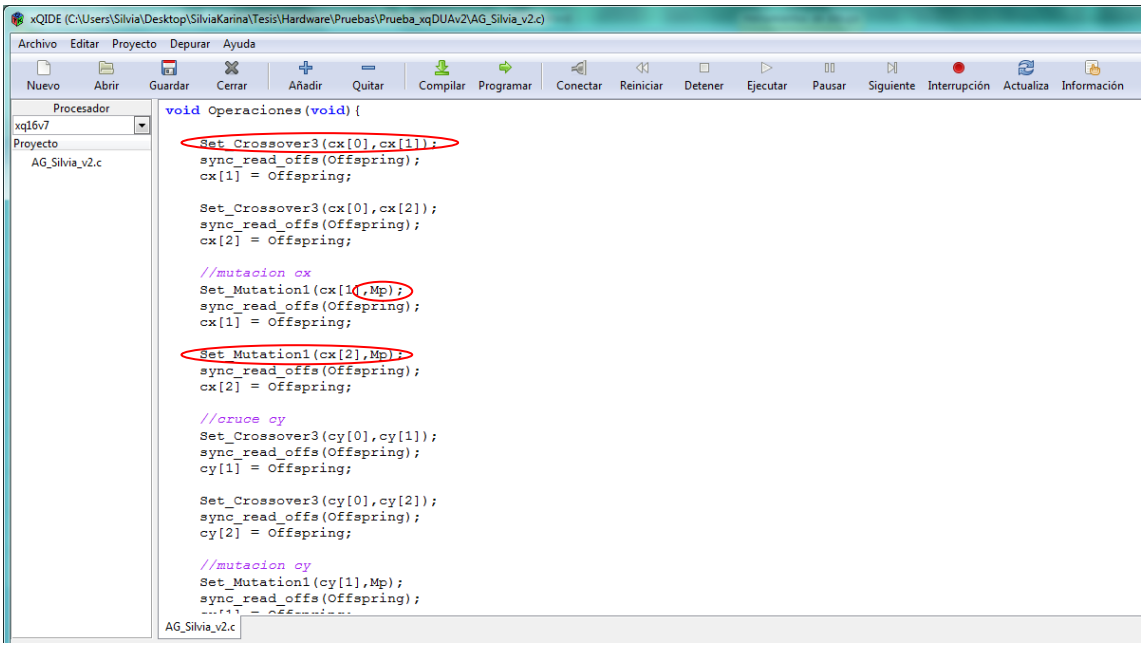


Fig. 36 Muestra de los datos a modificar par a las pruebas

Se utilizó la consola para monitorear la respuesta para cada iteración, donde al comprobar que se realizaba correctamente, los datos se guardaron en un archivo txt (Fig. 37) para ser graficados y ver los resultados.

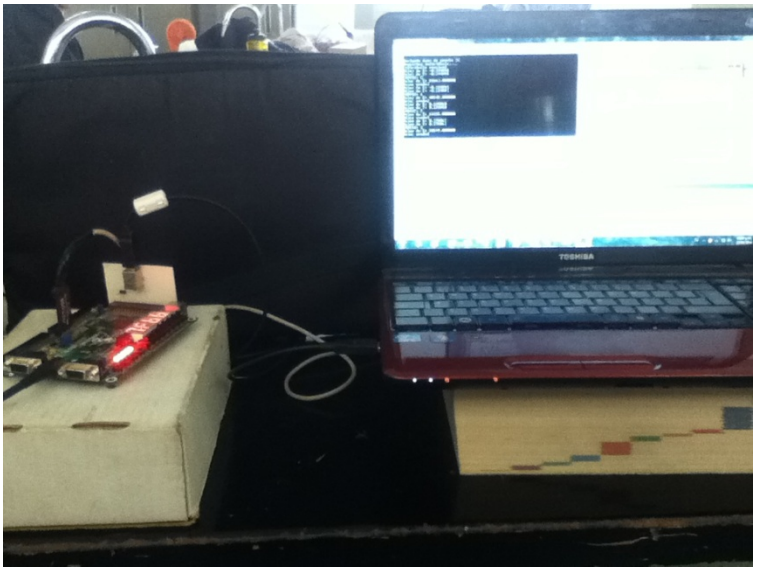


Fig. 37 Monitoreo por consola

En la consola se apreciaba el número de iteración, la evaluación de cada individuo, su desempeño, entre otros. (Fig. 38)

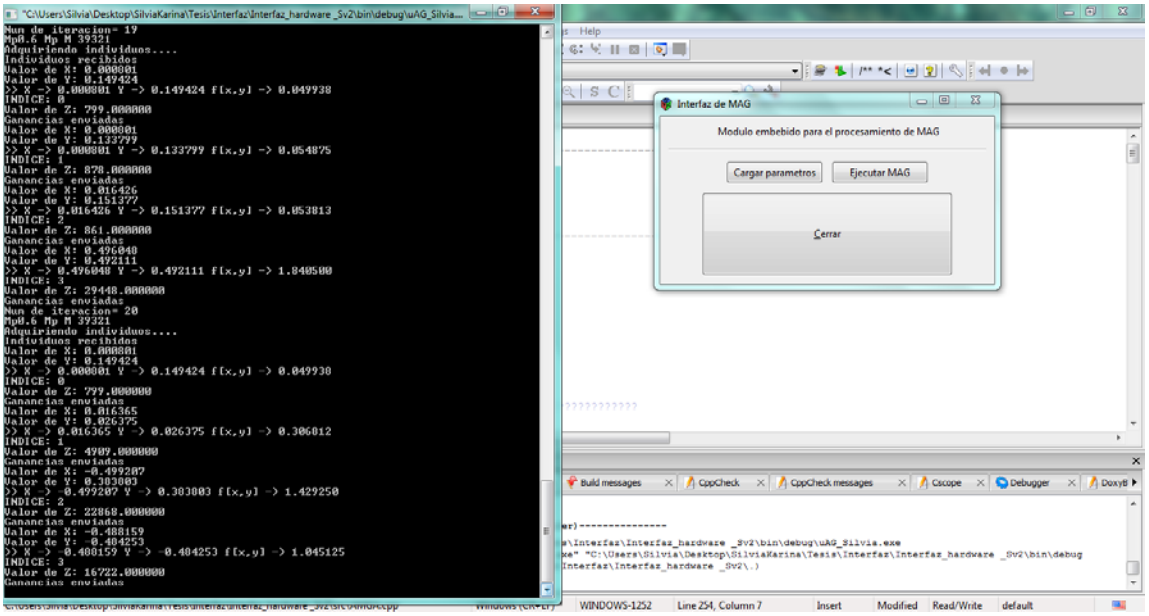


Fig. 38 Datos que mostraba la consola

Fig. 39 Proceso completo para la función de Bohachevsky

Finalmente se realizaron las 150 iteraciones por cada prueba realizada, monitoreando únicamente en la consola el número de iteración (Fig. 40) y así determinar que el proceso terminó correctamente (Fig. 39).

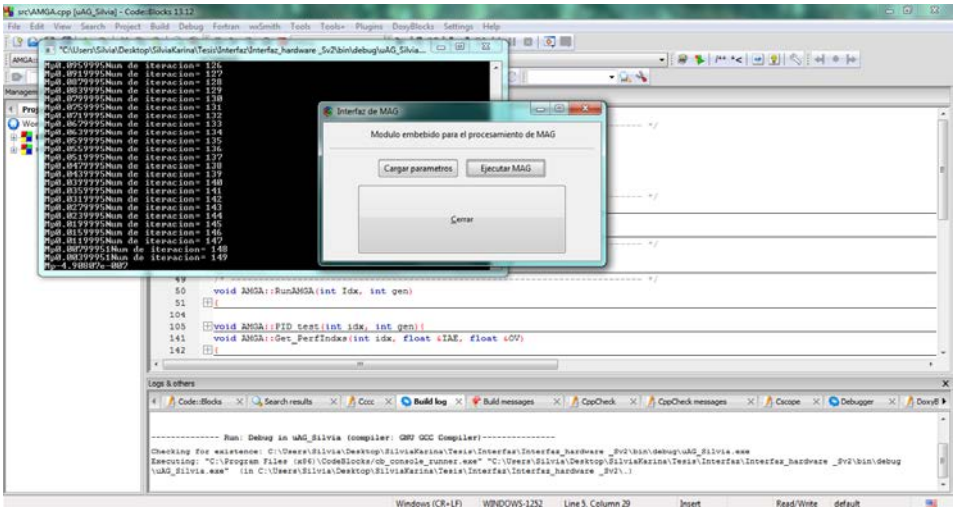


Fig. 40 Proceso terminado mostrado en consola

Los resultados se mostrará en el capitulo y sección correspondiente a este caso de estudio.

3.6.2 Segundo caso de estudio: sintonización de un controlador PID

Es de gran importancia que el módulo MAG sea probado en un sistema mecatrónico, es por ello las pruebas necesarias se llevarán a cabo en un

servomotor, el cual es el modelo que se utiliza en una fresadora. Es así como la aplicación del módulo se desarrollará en un sistema de control a lazo cerrado.

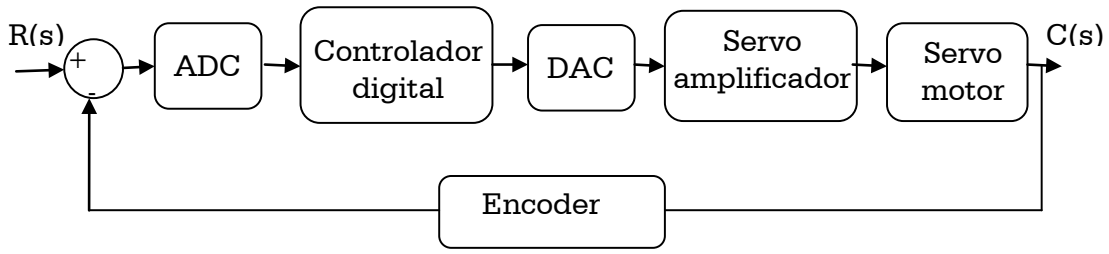


Fig. 41 Modelado de sistema para servoposicionamiento

El módulo MAG será el encargado de efectuar la sintonización de un controlador PID, en otras palabras, encontrará el valor óptimo de las ganancias kp, ki y kd (proporcional, integral y derivativo, respectivamente). No obstante, es necesario el ensamble de todo el sistema de control como se muestra en el modelado a bloques de la Fig. 41. Como referencia se utilizará la aplicación del mismo encoder, es decir, el motor al dar una vuelta realiza 4000 cuentas, es así como la referencia será a un número muy pequeño, se usó 200 como referencia. Los convertidores ADC y DAC, así como el control digital, se encuentran en la FPGA.

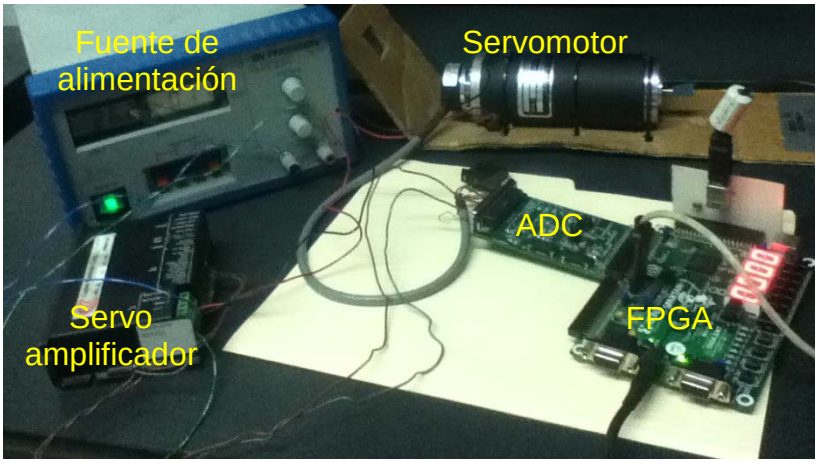


Fig. 42 Sistema físico de control a lazo cerrado

Para la parte de implementación en físico es observable en la Fig. 42 y en la Fig. 43. En la primera se enuncia que es cada elemento, se aprecia la fuente de alimentación que es utilizada para alimentar el servoamplificador, el servomotor, las tarjetas FPGA y el ADC. En la Fig. 43 se observa el sistema embebido integrado completo, junto con la PC.

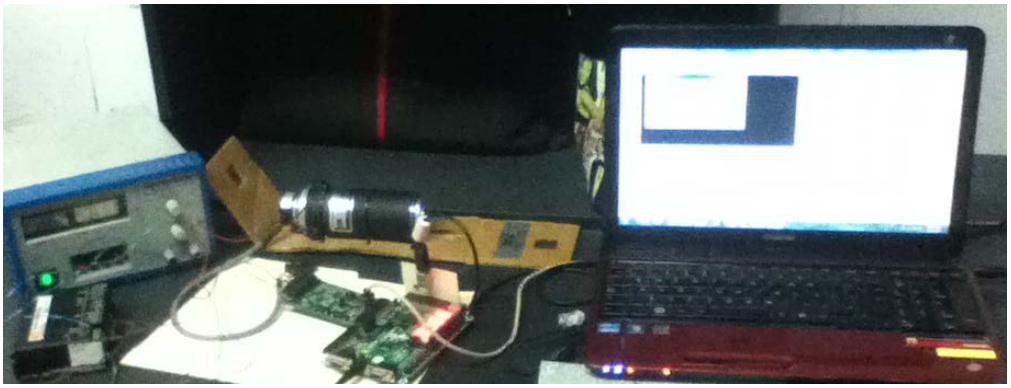


Fig. 43 Sistema embebido completo en físico

En estas pruebas experimentales, se agregaron varias funciones a la parte de la interfaz y en el compilador. También se mandaron llamar nuevos módulos como el del encoder y el PID, para poder mover el servomotor y realizar las pruebas. Cada prueba dura 3 segundos y es almacenada la respuesta ante cada prueba de sintonización de las ganancias generadas para analizarla. Se generan *N* números de pruebas por cada individuo, es decir, si se realizaron 60 iteraciones habrá 60 pruebas con el individuo 1, 60 con el individuo 2 y así sucesivamente. Las mejores pruebas están como se ha mencionado en el primer individuo.

4. RESULTADOS

4.1 Resultados de la implementación en software

Al realizar un AGE en Matlab, es decir en software, se asientan las bases teóricas y se dominan los principios básicos. Posteriormente alcanzado el objetivo de un AGE, se implementa el MAG, el cual es el enfoque de dicha tesis. Se hizo el desarrollo de los dos algoritmos, AGE y MAG, con el caso de la función de Bohachevsky. Se llevaron a cabo varias pruebas para determinar si el MAG es viable y los resultados son mostrados en la Tabla 1.

Tabla 1 Comparación entre el valor de los parámetros de un AGE y un MAG

Parámetros	AGE	MAG
Tamaño de la población	100	4
Número de generaciones	300	150
Probabilidad de cruza	0.6	1
Probabilidad de mutación	0.1	0.6
Variables de diseño (optimización)	X , Y	
Rango de diseño	[-0.5 0.5] , [-0.5 0.5]	
Función objetivo	$f(x,y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\sin(3\pi y) + 0.7$	

En la Tabla 1 se pueden comparar los parámetros más importantes en el algoritmo. En un AGE se utilizaron a cien individuos y en un MAG solamente son cuatro, al tener una reducción en la cantidad de individuos el proceso se vuelve menos iterativo, es decir, solamente en cada etapa (operador) se evalúan a cuatro individuos, haciendo más ligera la carga computacional.

También el número de iteraciones dice cuantas veces se repetirá el proceso, mientras menos sean, más rápido se genera el resultado e igualmente se aligera la carga computacional.

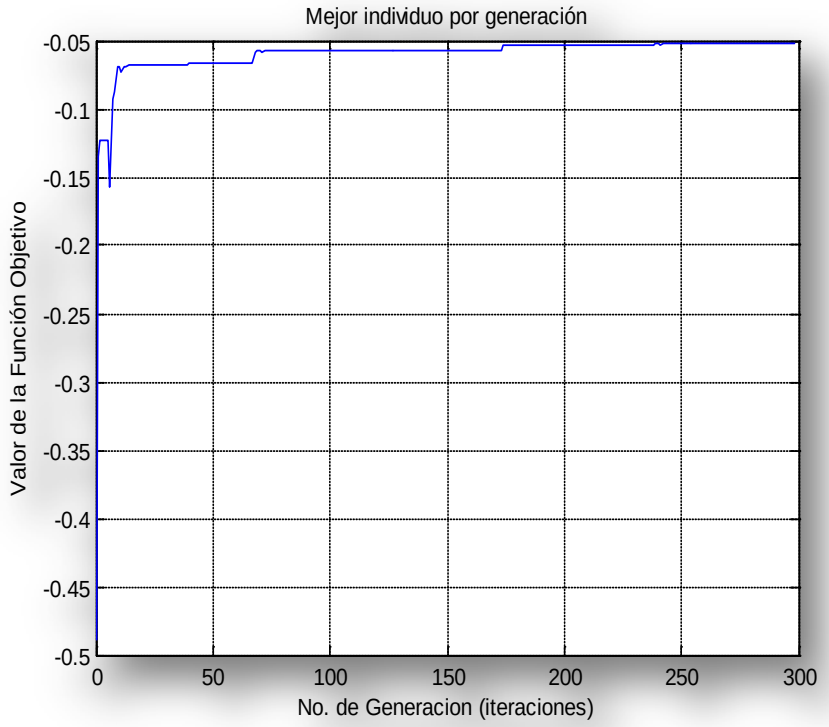
En las siguientes gráficas de la Fig. 44 a), b), c) se muestra el resultado de un AGE, donde en el eje de las x, son el numero de generaciones y el eje de las y es el valor de desempeño de el mejor individuo por generación.

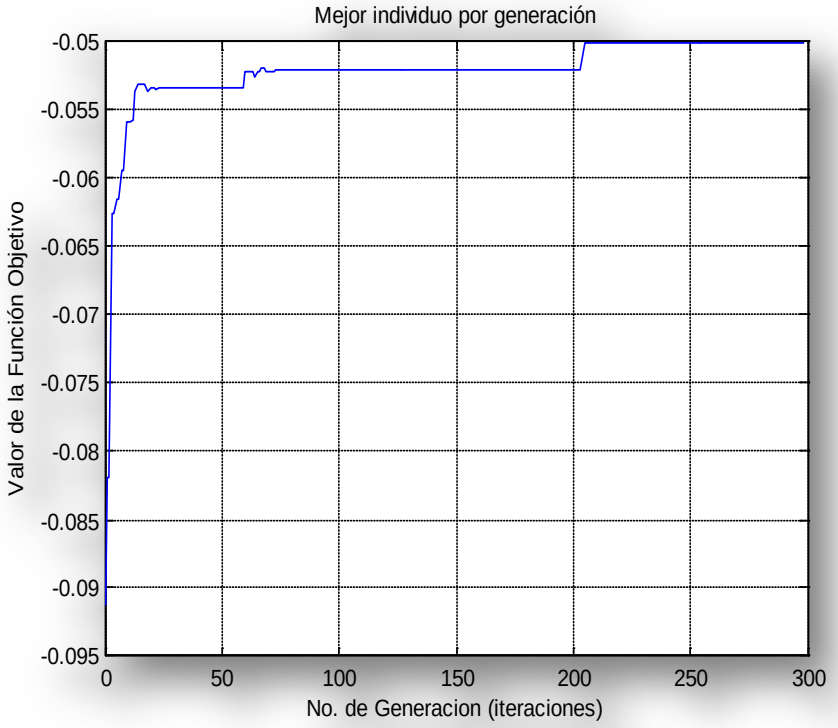
Entonces al tener a mejores individuos en cada generación, las nuevas generaciones tendrán mejores desempeños, así hasta convergir a la solución global.

Cabe mencionar que el resultado óptimo no se puede asegurar en que número de generación llegará a la solución final debido a que el valor de los individuos es aleatorio y en la mayoría del proceso existen randoms para la toma de decisiones, volviéndose impredecible.

En la Fig. 44 se parecía que en eje de las abscisas se tiene el número de generaciones, es decir, el número de veces que se repite todo el proceso. Por otro lado, para el eje de las ordenadas se muestra el valor de desempeño del mejor individuo por generación, es decir, de los cien individuos evaluados solo el mejor se muestra en la gráfica y ese permanece para la siguiente generación.

Es así como se puede observar que en la Fig. 44 a), Fig. 44 b), y en la Fig. 44 c) se alcanzan la solución óptima en un diferente número de generación, ya sea, en la iteración 10 o hasta después de la 200.


Mejor individuo por generación
Valor de la Función Objetivo
No. de Generacion (iteraciones)
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
-0.45
-0.5
0
50
100
150
200
250
300


Mejor individuo por generación
Valor de la Función Objetivo
No. de Generacion (iteraciones)
-0.05
-0.055
-0.06
-0.065
-0.07
-0.075
-0.08
-0.085
-0.09
-0.095
0
50
100
150
200
250
300

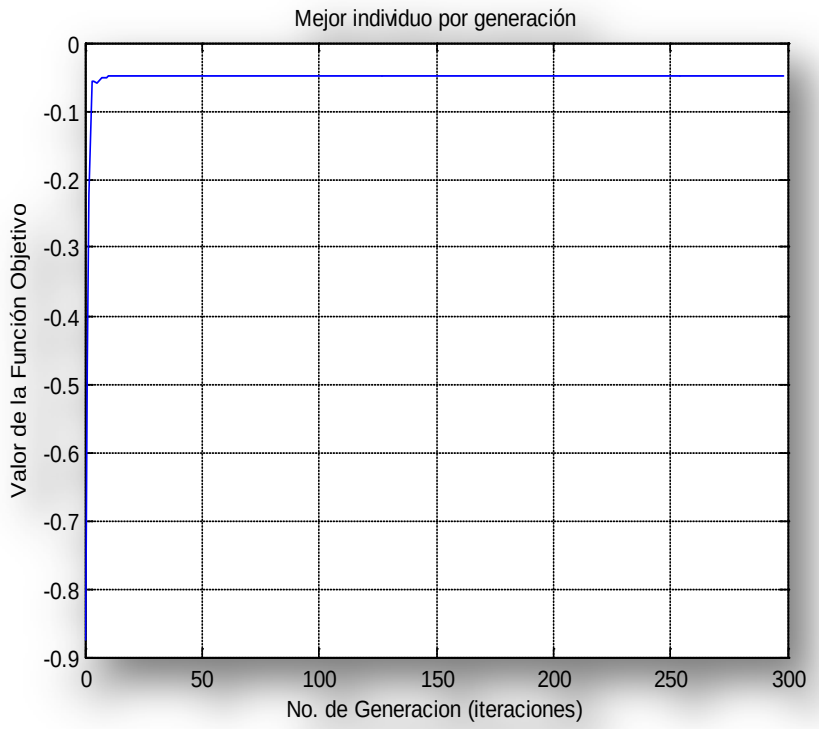
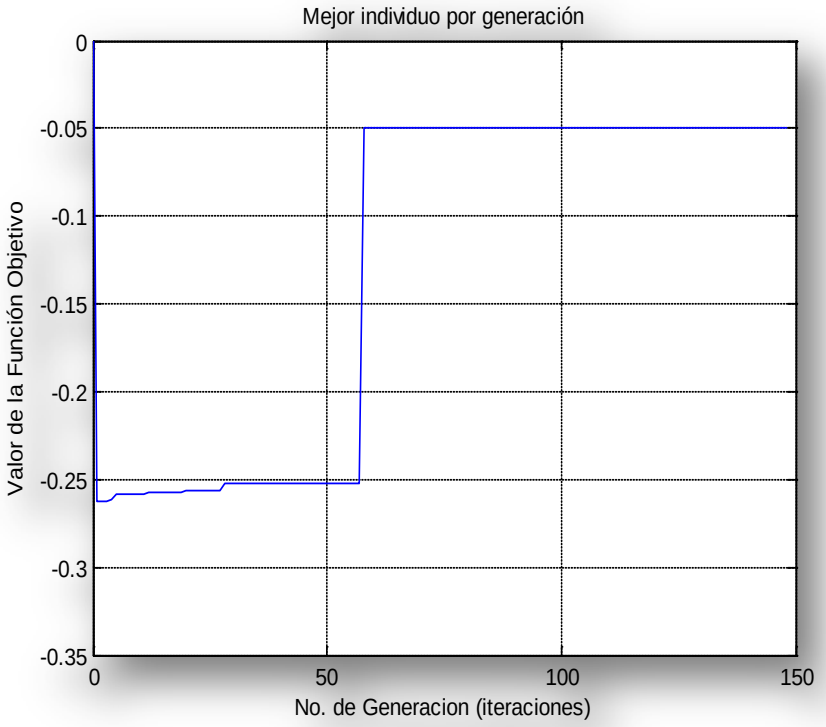
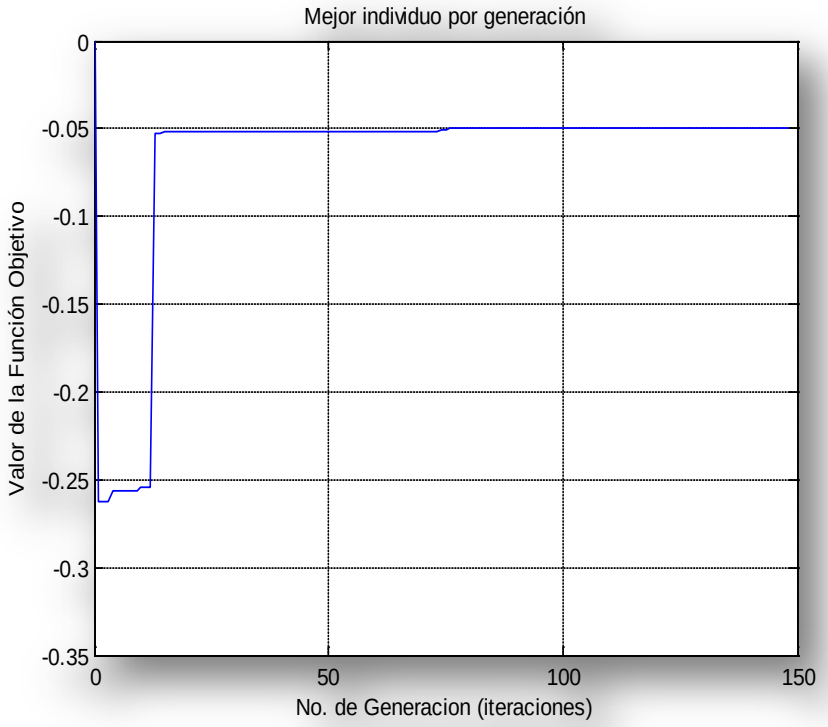


Fig. 44 Resultados del mejor individuo por generación en un AGE

Por otro lado, los MAG se comportan similares que los AGE solamente que con menos generaciones e individuos. Para la Fig. 45 a), Fig. 45 b) y Fig. 45 c) es de igual forma para el eje de las abscisas donde se tiene el número de generaciones, es decir, el número de veces que se repite todo el proceso y para el eje de las ordenadas se muestra el valor de desempeño del mejor individuo por generación, que en ésta caso como solo son cuatro individuos y el primero siempre es el mejor, siempre se muestra el primero de cada generación. De igual forma es apreciable que el resultado se obtiene en una iteración diferente. Además, solo son 150 iteraciones muchas menos que en el AGE, en el cual son 300. También el primer individuo tiene el mismo valor de desempeño porque éste se propone fijo para no caer en un error prematuro.


Mejor individuo por generación
Valor de la Función Objetivo
No. de Generacion (iteraciones)
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
0
50
100
150


Mejor individuo por generación
Valor de la Función Objetivo
No. de Generacion (iteraciones)
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
0
50
100
150

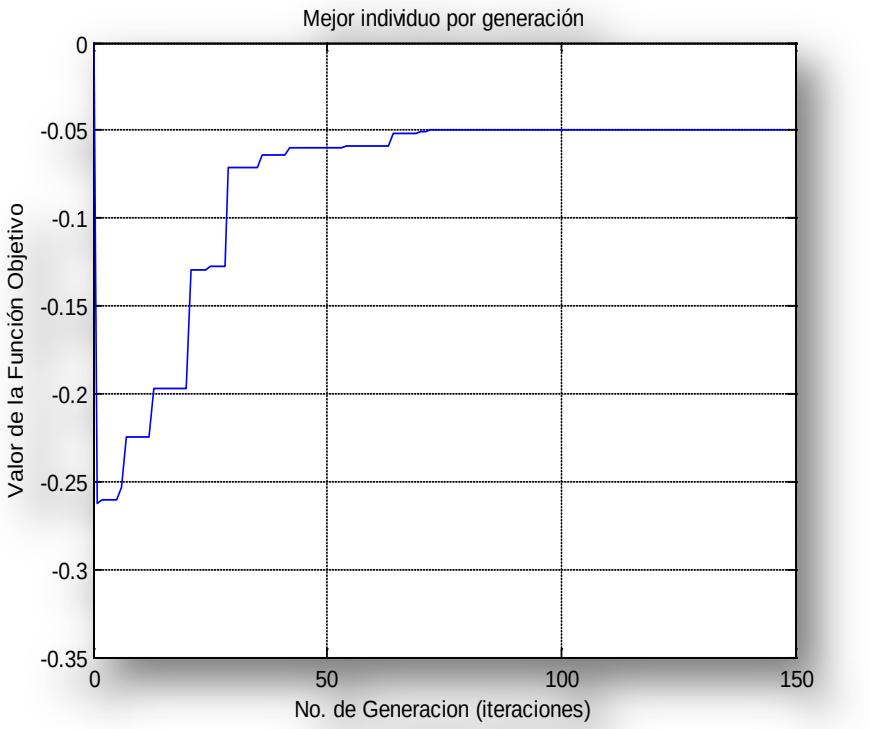


Fig. 45 Resultados del mejor individuo por generación en un MAG

Es así como la Fig. 44 y Fig. 45 muestran las diferencias en los resultados para ambos métodos, AGE y MAG respectivamente. Además de enfatizar que de igual forma al tener una población muy pequeña arroja los mismos resultados que al tener un gran número de individuos y en menos iteraciones.

4.2 Descripción en hardware de la arquitectura propuesta

Por lo que se refiere al diagrama de bloques visto en la sección de la metodología, el diagrama general, se tiene en especial a los de la Fig. 21 y Fig. 22, que son los del operador de cruce y de mutación, respectivamente. Es necesario implementarlos en la estructura de VHDL, pero antes se necesita realizar simulaciones que demuestren que los resultados arrojados serán los correctos. Por lo que se han hecho las simulaciones al respecto de las

modificaciones añadidas, como son la cantidad de puntos de cruce y puntos de mutación.

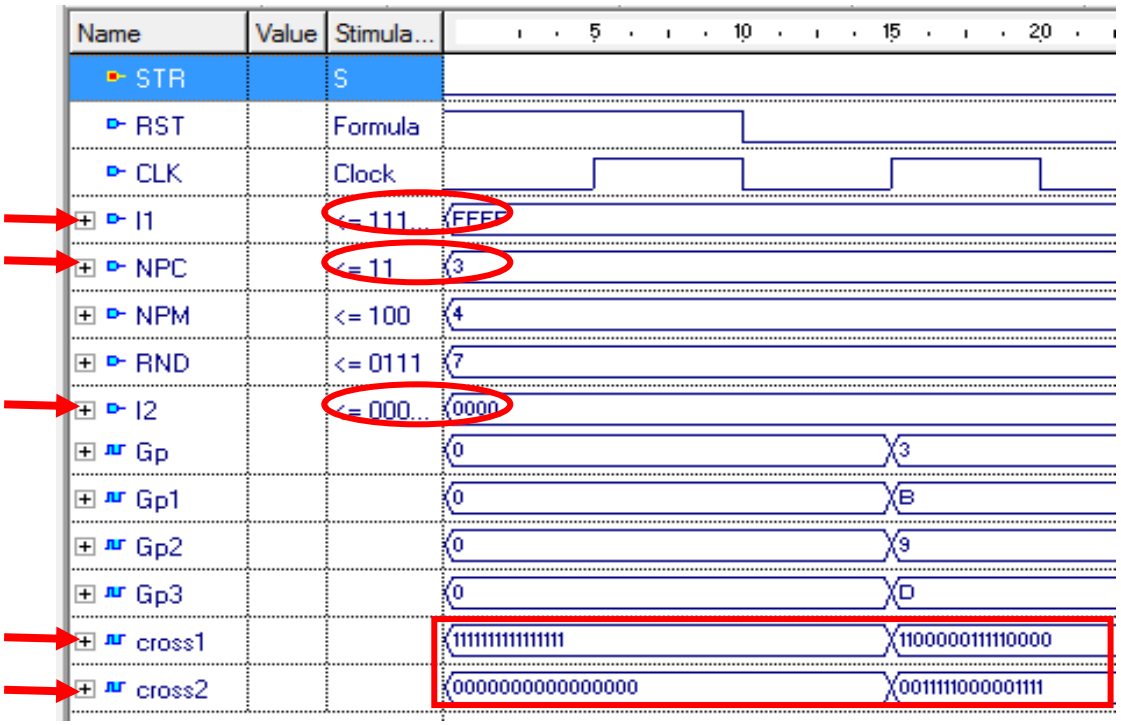
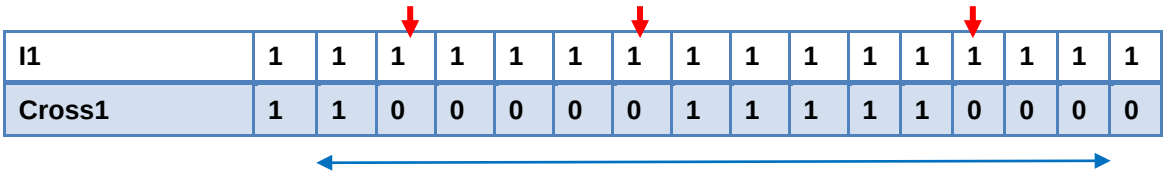


Fig. 46 Simulación de tres puntos de cruce

En la Fig. 46 se demuestra cómo sería el resultado de la cruza si se eligieran tres puntos. Explicando la simulación anterior, se tiene que un individuo (I1) tiene 16 bits y todo ellos son unos (FFFF) y un segundo individuo (I2) con todo sus bits en ceros (0000), para poder hacer más claro el resultado de cruce, se eligieron tres puntos de cruce (NPC) y como número aleatorio se escogió el siete (RND), así como solo se puede pedir un número random por iteración, se hizo un arreglo de bits para que pudieran haber tres tipos de randoms diferentes con un solo random (Gp, Gp2, Gp3). Realizando las operaciones necesarios de cruza de extremos con medios o como fuera la situación dependiendo la cantidad de puntos a cruzar. Es así como cross1 y cross2, tiene marcados sus puntos y efectivamente se pueden apreciar el

intercambio de bits. Para una mejor vista se muestra en la Tabla 2, donde los puntos de cruza son Gp=3, Gp2=9 y Gp3=D (13 en decimal).

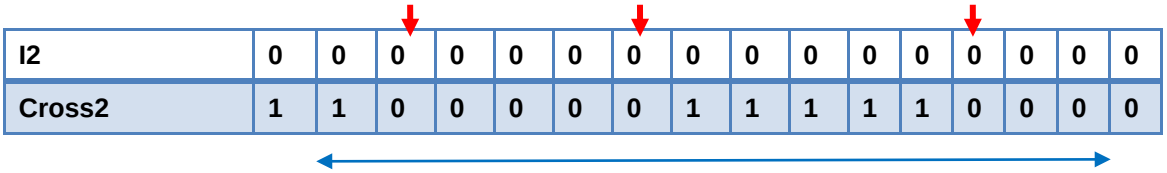
Tabla 2 Acercamiento del I1 aplicandole el operador cruce



I1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Cross1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0

Es lo mismo para el caso del segundo individuo pero que éste contiene solamente ceros, los puntos de cruza serán los mismos al igual que el intercambio (Tabla 3). Sencillamente la diferencia es que las cruzas serán las contrarias.

Tabla 3 Acercamiento del I2 aplicandole el operador cruce



I2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cross2	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0

Cuando se haya realizado la cruza, solamente uno de los dos individuos pasará a la siguiente generación y este será elegido por medio de un número aleatorio, así pasará el uno (I1) o pasará el dos (I2).

Del mismo modo se efectuó la simulación para la implementación del operador de mutación y los resultados se muestran en la Fig. 47.

En contraste con el de cruce los diferentes randoms utilizados son Gp, Gp1, Gp2, y Gp3, se tuvo una cantidad de cuatro puntos de mutación (NPM), posteriormente de tres, y al realizar el cambio de bit a su opuesto, tenemos marcados los resultados en mut1 y mut2.

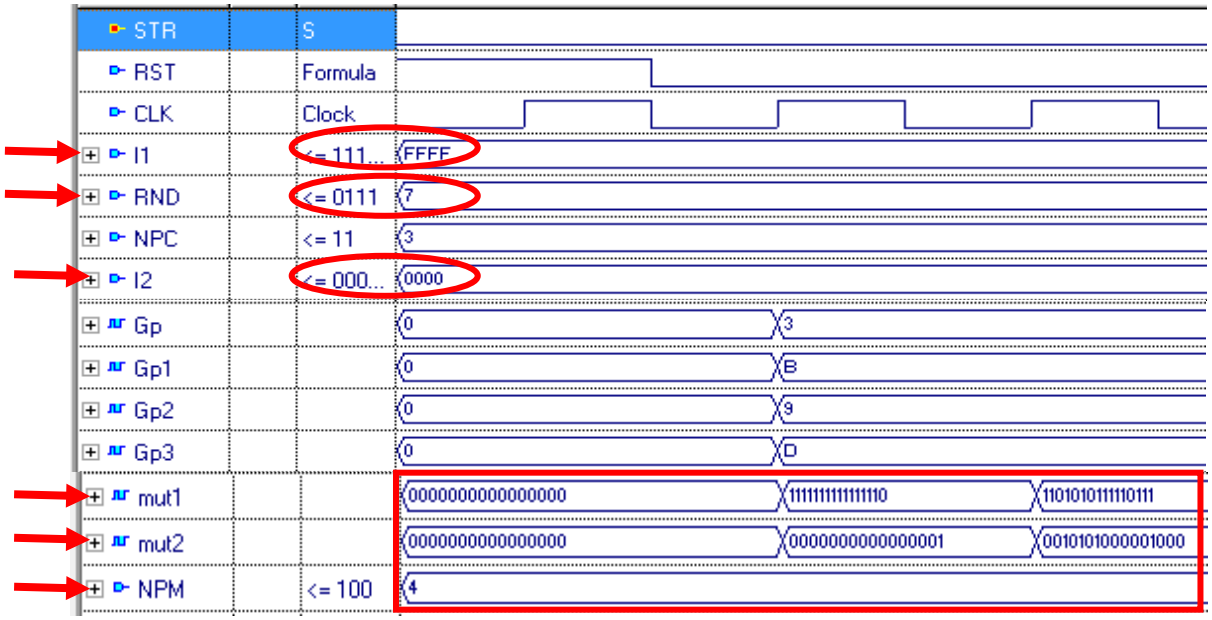


Fig. 47 Simulación de la mutación

Aunque se puede observar un pequeño desfase de un ciclo de reloj, debido a que realiza la mutación en el flanco ascendente y esto sucede hasta un ciclo después. Para hacerlo más claro los resultados de los cambios de bits por los cuatro puntos de mutación se ven en la Tabla 4 y Tabla 5, donde son los puntos máximos de posibles.

Tabla 4 Acercamiento del I1 aplicándole el operador mutación

I1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Mut1	1	1	0	1	0	1	0	1	1	1	1	1	0	1	1	1

Tabla 5 Acercamiento del I2 aplicándole el operador mutación

I2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mut2	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0

4.3 Resultados experimentales de verificación del módulo

Con respecto a ésta sección, permite mostrar los resultados obtenidos en las diferentes pruebas experimentales, resumiendo y ordenando la información de una gran cantidad de pruebas por medio de tablas, haciendo más fácil y entendible los resultados.

4.3.1 Resultados experimentales con la función Bohachevsky

Acerca de las pruebas realizadas para la validación del módulo de MAG aplicado a la función de Bohachevsky, en la Tabla 6 se muestran los resultados en concreto. Se aprecia que por cada prueba realizada se llevaron a cabo veinte repeticiones. En las pruebas que sean distinguidas por un asterisco (*) al lado, indica que en ellas no se modificó la probabilidad de mutación, es decir que siempre se mantuvo constante, en éste caso 0.6. Mientras que en las demás por cada generación que pasaba se iba decreciendo gradualmente para no ocasionar cambios tan drásticos, debido a que se supone ya se está llegando al valor final deseado.

Tabla 6 Resumen de resultados

No. de prueba	Puntos de cruce	Puntos de mutación	Repeticiones realizadas	No. de repeticiones con:		
				0.049	0.05-0.0509	0.051-0.06
1*	1	1	20	13	7	0
2*	2	2	20	4	15	1
3*	2	1	20	8	11	1
4*	3	3	20	7	11	2
5*	2	4	20			
6*	3	4	20			
7	1	1	20	10	8	2
8	2	3	20	6	10	4
9	2	1	20	11	9	0

10	3	3	20	4	8	8**
11	3	1	20	16	4	0
12	3	4	20			

El valor que alcanza la respuesta óptima es el de 0.0499, así en la tabla anterior se muestra cuantas repeticiones llegaron al resultado correcto, las que estuvieron casi cerca y las más alejadas. Entonces se tiene que la combinación más adecuada para garantizar un resultado deseado es la de 3 puntos de cruza con un punto de mutación. Y la combinación con peores resultados fueron los de tres puntos de cruce y tres de mutación. También se obtuvieron mejores resultados reduciendo la probabilidad de mutación para cada iteración.

Recordando que los individuos a encontrar eran X y Y, donde éstos al ser evaluados por la función objetivo, en éste caso la función de Bohachevsky, proporcionaban el valor de desempeño, Z, y éste permitía realizar el ordenamiento y encontrar el punto más alto. Así se manejan las variables en las gráficas siguientes de ésta sección.

Para la gráfica de la Fig. 48 y de las posteriores a ella, se tiene en el eje de las abscisas las generaciones o el número de veces que se repitió el proceso para llegar al resultado final. En el eje de las ordenadas es el valor de desempeño, se sabe que el valor a alcanzar es de 0.0499.

Se observa como en la gráfica de la Fig. 48 y de la Fig. 49, el resultado se obtuvo correctamente solo que en diferente número de iteración.

Hubo ocasiones que el resultado óptimo se encontraba en la iteración cuatro (Fig. 50) y otras inclusive fueron hasta la última iteración (Fig. 51).

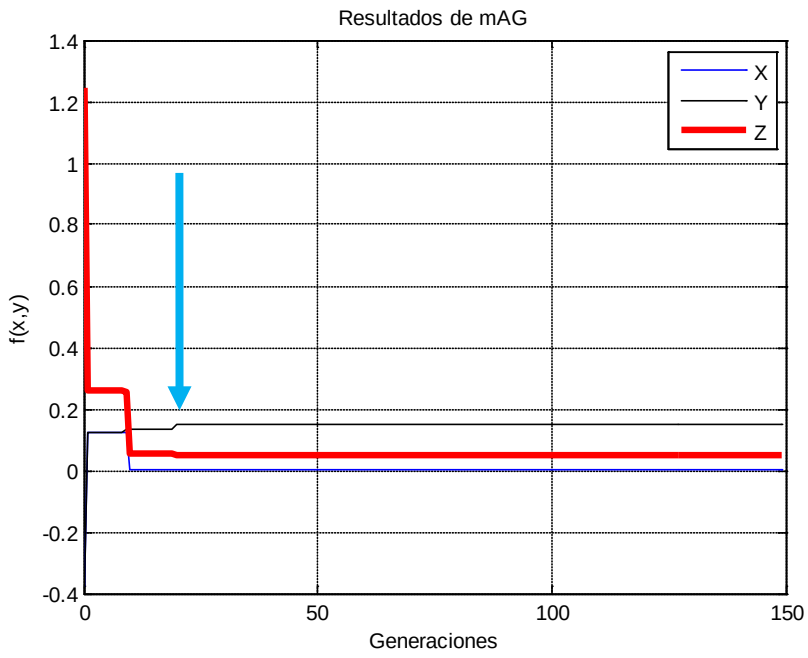


Fig. 48 Gráfica con un resultado en las primeras iteraciones

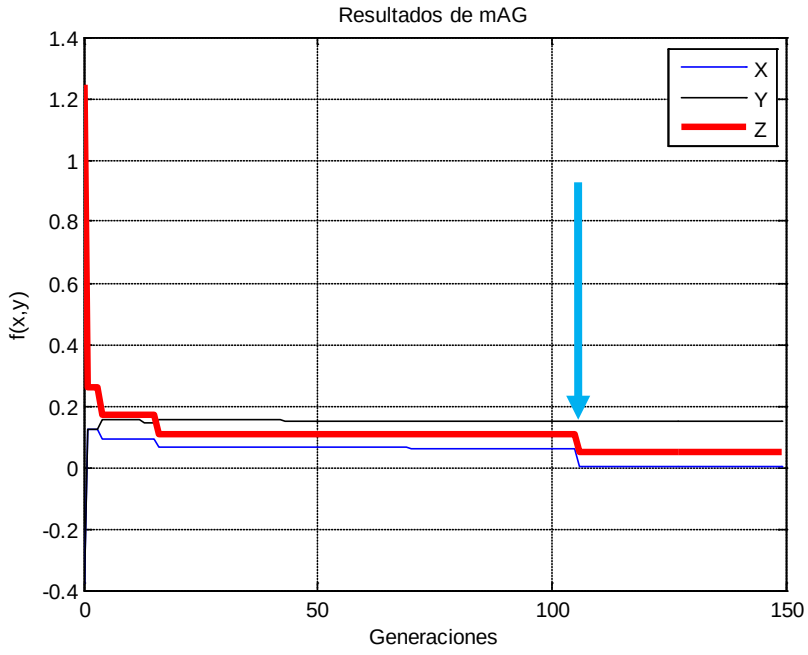


Fig. 49 Gráfica con un resultado en las primeras iteraciones

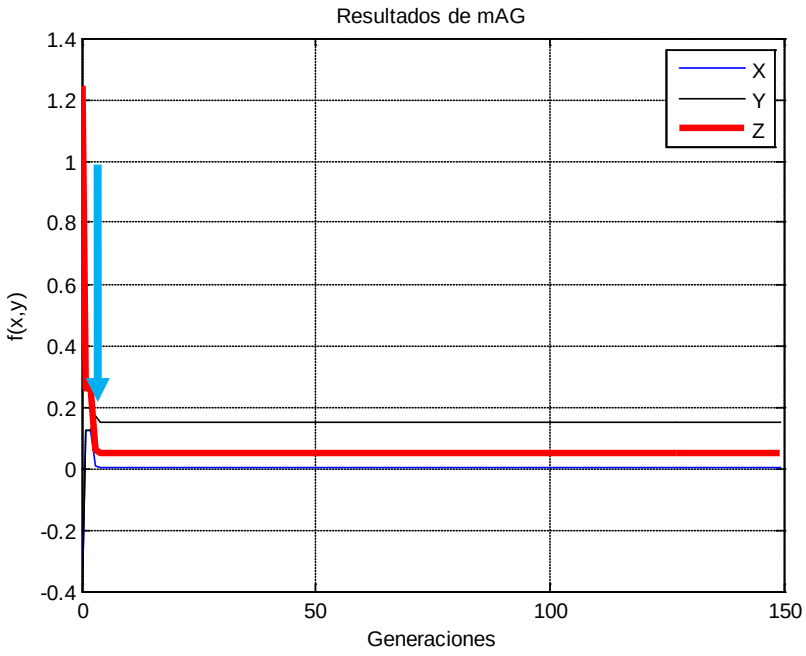


Fig. 50 Se llegó al resultado de 0.049938 en la cuarta iteración

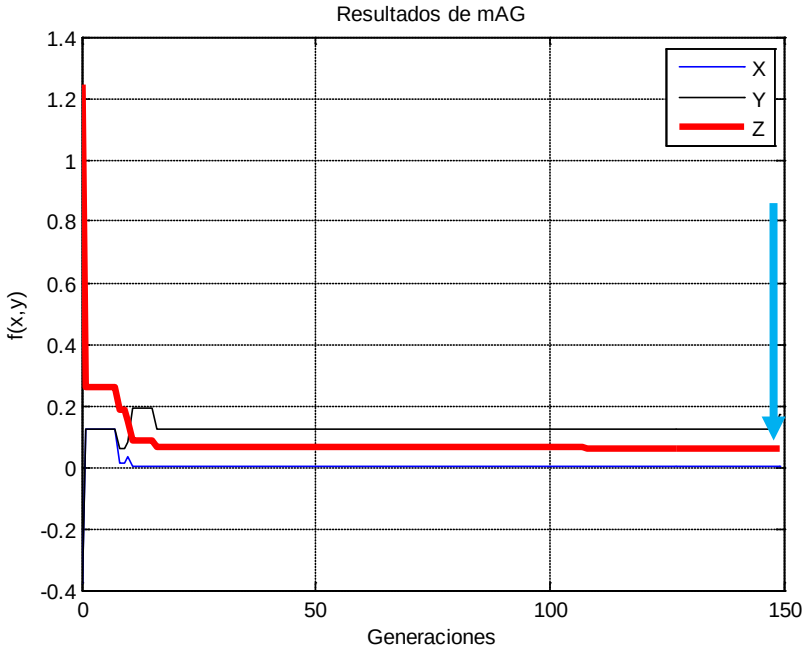


Fig. 51 Se obtuvo Z=0.049938 en la iteración 150

Mostrando la diversidad de encontrar la respuesta a diferente número de generación debido al aleatorio en los puntos de cruce y mutación.

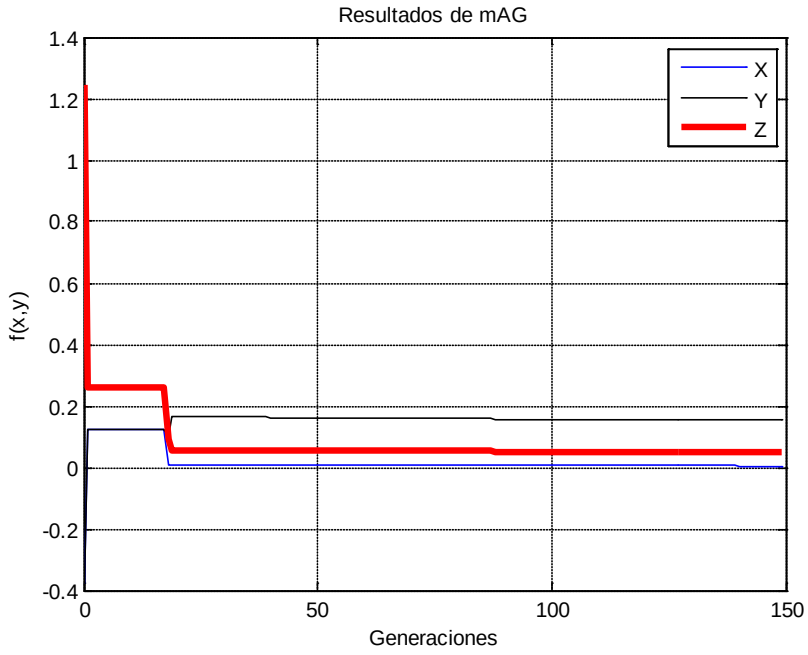


Fig. 52 Peor desempeño con 0.0590125 como resultado final

Es importante destacar que donde se obtuvieron los peores resultados fue en la prueba número 10 (3 puntos de cruce y tres de mutación), se alcanzó el desempeño más lejano de 0.059125 (Fig. 52), todos los demás más lejanos se encontraban entre 0.051, la mayoría, y escasamente hubo 0.052 y 0.053 quizás no se note el alejamiento de la respuesta en la gráfica porque son valores muy pequeños y aunque fue el más lejano, está bastante cerca del resultado.

4.3.2 Resultados experimentales con un servomotor

Se realizaron una gran cantidad de pruebas experimentales en un servomotor, estas pruebas son denotadas en la Tabla 7, la cual muestra el intervalo de pruebas realizadas con un número especifico de generaciones (Núm. de Gen.), la probabilidad de mutación que se aplicó (Mp), al igual que la cantidad de puntos de cruce (NPC) y de mutación (NPM).

Tabla 7 Resultados de las pruebas del servomotor

No. prueba	Elementos por prueba	Núm. de Gen	Mp	NPC	NPM	Sobrepaso	Sin sobrepaso	No llegaron a la referencia
1	1-20	100	0.6	3	1	15	5	0
2	21-40	80	0.6	3	3	14	5	1
3	41-60	80	0.4	3	1	16	4	0
4	61-80	80	0.4	3	3	19	0	1
5	81-100	80	0.99	3	4	17	2	1
6	101-120	70	0.99	3	1	17	2	1
7	121-130	70	0.6	3	1	7	2	1
8	131-140	70	0.6	2	4	8	2	0
9	141-160	70	0.99	3	4	11	6	3
10	161-170	70	0.99	3	1	10	8	2
11	171-190	80	0.6	1	1	10	9	1

En la Tabla 7 se muestra que los peores resultados debido a que no llegaron a la referencia fueron con las pruebas de los puntos de mutación y de cruce más altos (3 y 4 respectivamente), además de tener un número de probabilidad de 0.99, es decir que casi a todos los individuos se les aplicaba los operadores de cruce y mutación. Esto se debe a que tenemos gran diversidad de posibles soluciones y por ende se pierde en el camino de encontrar a la solución óptima. Y los mejores resultados fueron obtenidos con tres puntos de cruce y uno de mutación, ya que se generó en el cruce una diversidad a las soluciones pero en la mutación solo se altera de forma considerable para generar una solución óptima.

Tabla 8 Resultados promedio de las pruebas

prueba	num. Gen	Mp	NPC	NPM	kp	ki	kd	OV	IAE
1	100	0.6	3	1	267.93	8775.41	4.11	1677352.96	9900.84
2	80	0.6	3	3	268.85	8568.13	4.23	2855861.51	12813.90
3	80	0.4	3	1	270.10	8863.66	4.18	1729395.81	10045.90
4	80	0.4	3	3	237.60	9056.66	4.09	1692970.23	9896.03
5	80	0.99	3	4	256.33	8427.41	4.08	2427616.67	11733.06
6	70	0.99	3	1	258.43	9039.90	4.01	1736895.78	10112.42
7	70	0.6	3	1	236.19	8919.49	4.14	1739659.57	10028.42
8	70	0.6	2	4	261.81	9040.79	3.96	1804957.02	10002.95
9	70	0.99	3	4	271.67	8547.54	3.86	1808143.71	10742.65
10	70	0.99	3	1	272.95	7896.50	3.99	1885806.17	10796.01
11	80	0.6	1	1	254.00	8312.39	3.47	2039419.34	10872.27

En la Tabla 8 nos señala el promedio de cada conjunto de pruebas que resulto de las ganancias kp, ki y kd. Además de mostrar los valores de las funciones objetivo que fueron importantes para evaluar el mejor desempeño.

Para la Tabla 9 se observa el error de cada individuo en su conjunto de pruebas. Se aprecia que los resultados son muy parecidos. Además la desviación estándar de los puntos de cruce de 3 y de 4 puntos de mutación es donde se aprecia mayor rango, esto se debe a que como tiene mayor diversidad de soluciones se tiene un rango más amplio de los valores de las posibles ganancias. También es importante mencionar que estos errores son el promedio de un conjunto de datos. A lo que se refiere es que las soluciones finales si llegaban a la referencia indicada pero en la búsqueda de esta solución los pruebas generadas mostraban un error hasta que llegaba a la referencia.

Muy pocas pruebas como se ve en la Tabla 7 no llegaron a la referencia pero la última prueba de la generación si llego a la referencia algunas con sobre paso y otras sin sobre paso. Sin embargo, son errores generados por muchos datos que al inicio de las iteraciones mostraban grandes errores ya que eran errores exagerados pues apenas se estaba buscando la solución óptima con el algoritmo.

Tabla 9 Promedio individual de cada promedio de un elemento de una prueba de la misma clase

prueba	error	desv error	error ind1	error ind2	error ind3	error ind4	desv estd 1	desv estd 2	desv estd 3	desv estd 4
1-20	23.25	9.05	27.41	26.18	25.49	-7.89	8.15	8.17	8.35	11.53
21-40	26.35	9.51	32.82	27.89	25.63	-11.16	8.27	8.70	9.11	11.98
41-60	22.05	9.10	28.12	26.62	23.90	-4.84	8.21	8.24	8.49	11.44
61-80	19.28	9.30	25.21	24.05	21.20	-5.20	8.22	8.31	8.84	11.82
81-100	25.35	10.15	32.12	19.22	16.58	-33.36	8.48	9.87	10.03	12.22
101-120	21.22	9.24	26.80	25.72	21.84	-8.13	8.24	8.38	8.62	11.70
121-130	21.27	9.19	25.81	23.43	21.26	-9.44	8.28	8.32	8.55	11.60
131-140	24.68	9.74	26.92	25.30	21.65	-24.85	8.23	8.66	9.32	12.74
141-160	13.20	11.40	29.04	1.84	-3.75	-25.67	8.28	12.55	12.59	12.19
161-170	27.28	9.53	31.52	28.57	27.29	-21.73	8.34	8.45	8.63	12.70
171-190	26.23	9.58	28.54	28.47	27.38	-20.53	8.34	8.26	8.33	13.37

En las graficas siguientes se muestran algunas gráficas simbólicas que muestran los diferentes tipos de respuestas que mostraron la última iteración de cada generación, es decir, la solución final generada por el MAG.

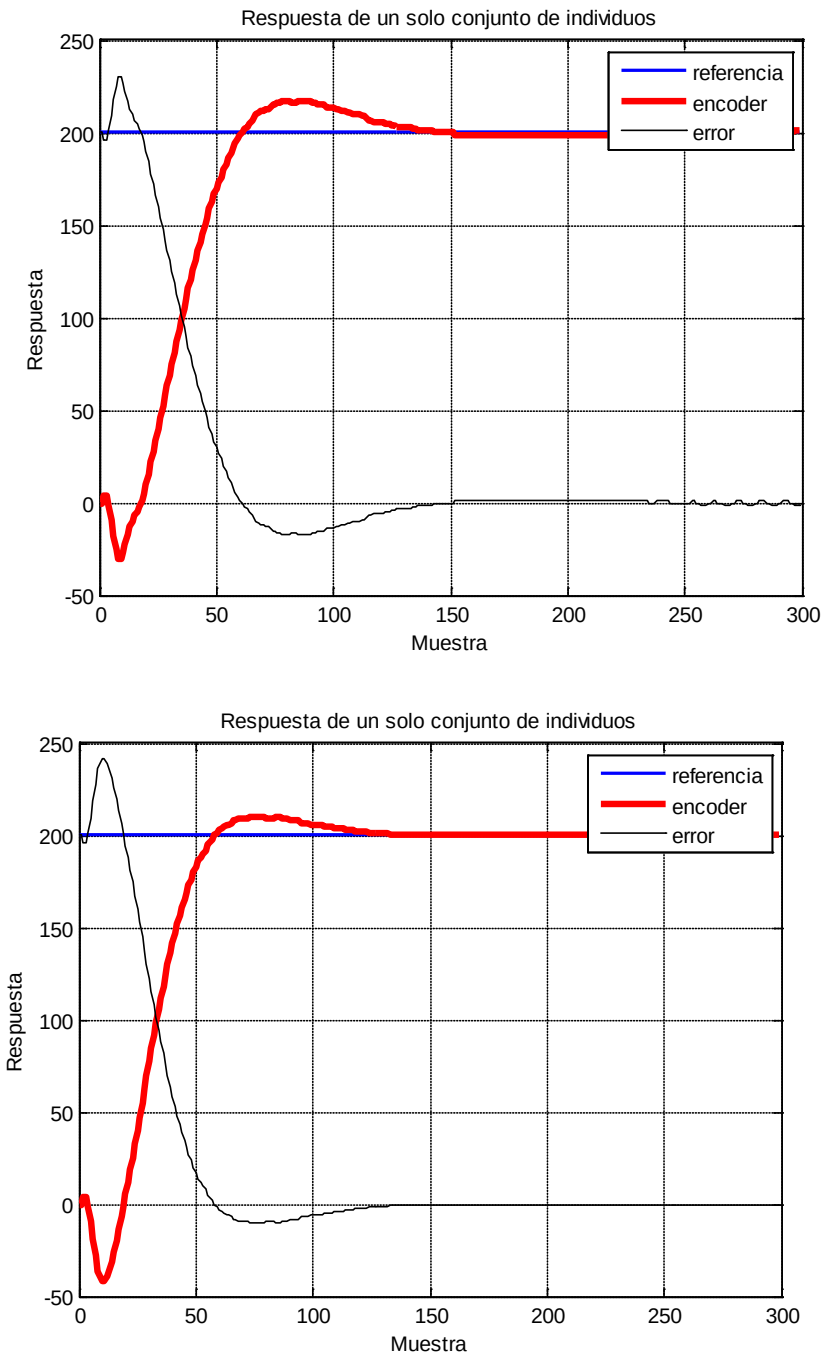


Fig. 53 Gráficas obtenidas de la respuesta de un servomotor aplicando un MAG

En el eje de las X se denota las muestras que fueron tomadas por cada prueba al probar un kp, ki y kd diferentes, y en el eje de las Y, se muestra la respuesta. La señal roja es la respuesta, siendo observable un sistema de

segundo orden. La azul es la referencia a la que se pretende llegar y la negra es el error, donde el error llega a cero. Tanto en la gráfica inferior de la Fig. 53 como en la superior es observable que se llegó a la referencia, el control fue exitoso, solo que se tuvo un sobre paso muy ligero pero aceptable.

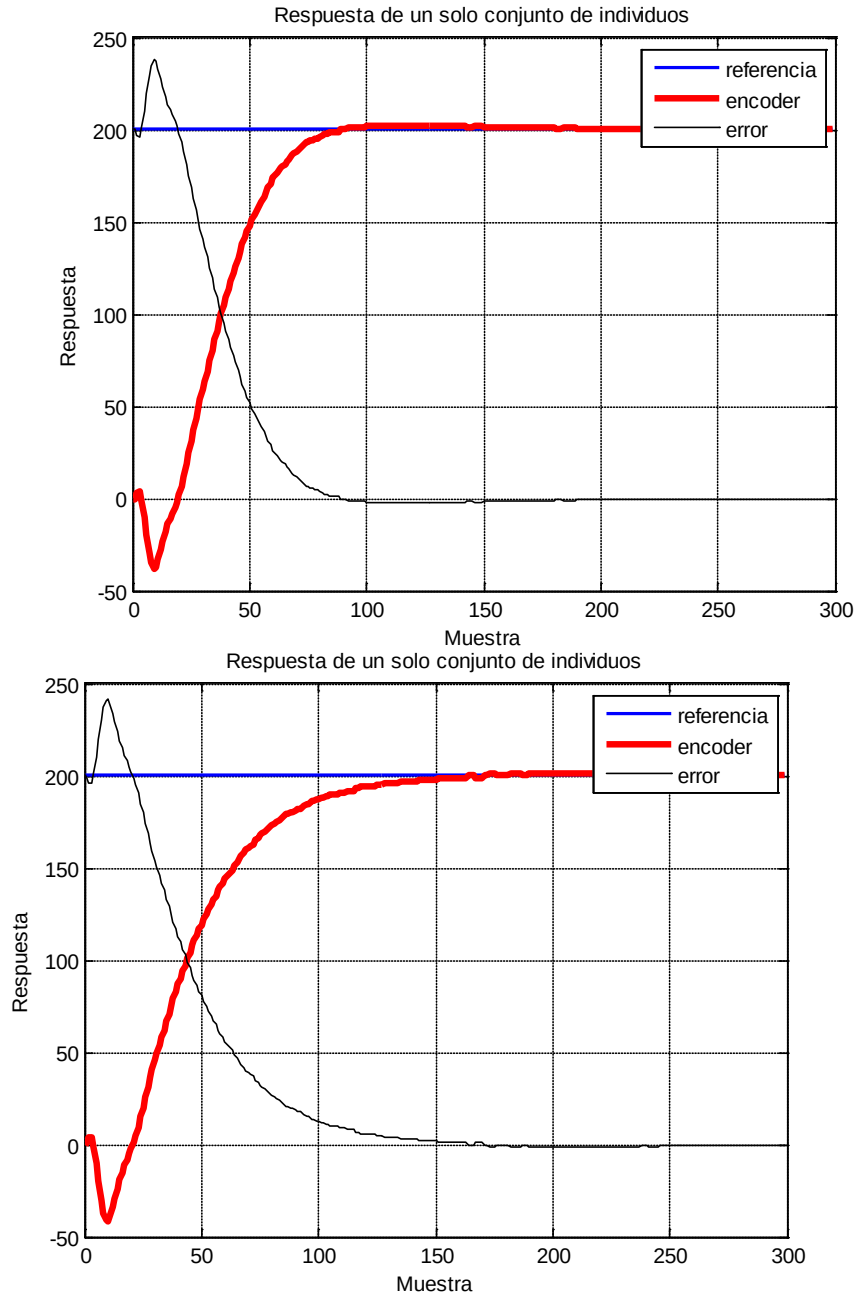


Fig. 54 Respuesta de un servomotor aplicando un MAG sin sobrepaso

A diferencia de la Fig. 53 que se tiene sobrepaso, también hubo respuestas sin sobre paso como se muestran en la Fig. 54. Y son respuestas que llegan a la referencia.

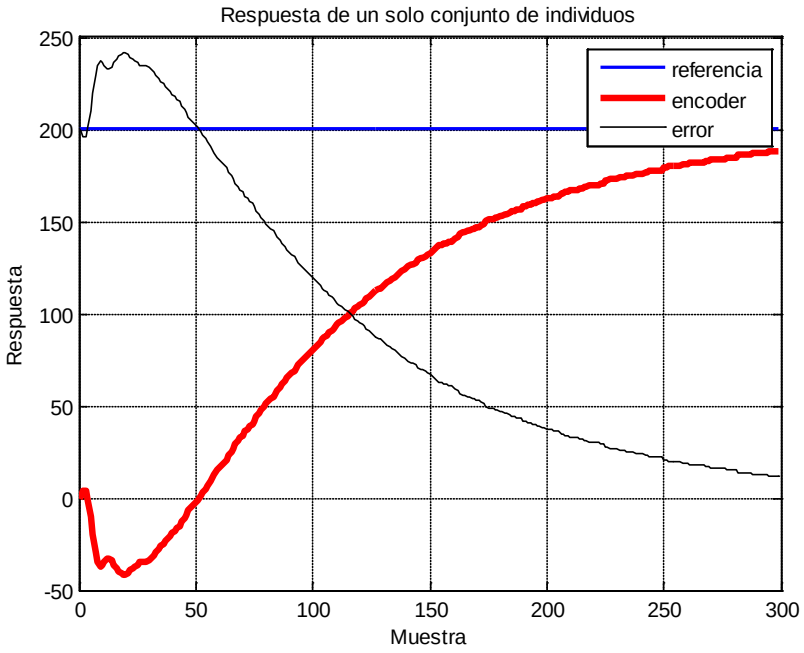


Fig. 55 Respuesta que no llegó a la referencia

Para la Fig. 55 se muestra uno de los pocos casos que no llegaron a la referencia. Es decir, que los valores encontrados por el algoritmo genético de kp, ki y kd, no fueron los indicados. No se llegó a una solución óptima.

Y para los valores de las ganancias kp, ki y kd del contralador PID. Se muestra su desarrollo en el trascurso de las iteraciones, hasta llegar a la iteración final deseada. Esto se muestra en la Fig. 56 con dos ejemplos, en el eje de las X, se ven el número de iteraciones y en el de la Y, es valor de las ganancias. Los valores de las ganancias pueden ser tan diversos o llegar a un punto muy rápido de un valor y permanecer fijo. Ninguna gráfica es igual debido al mecanismo del algoritmo.

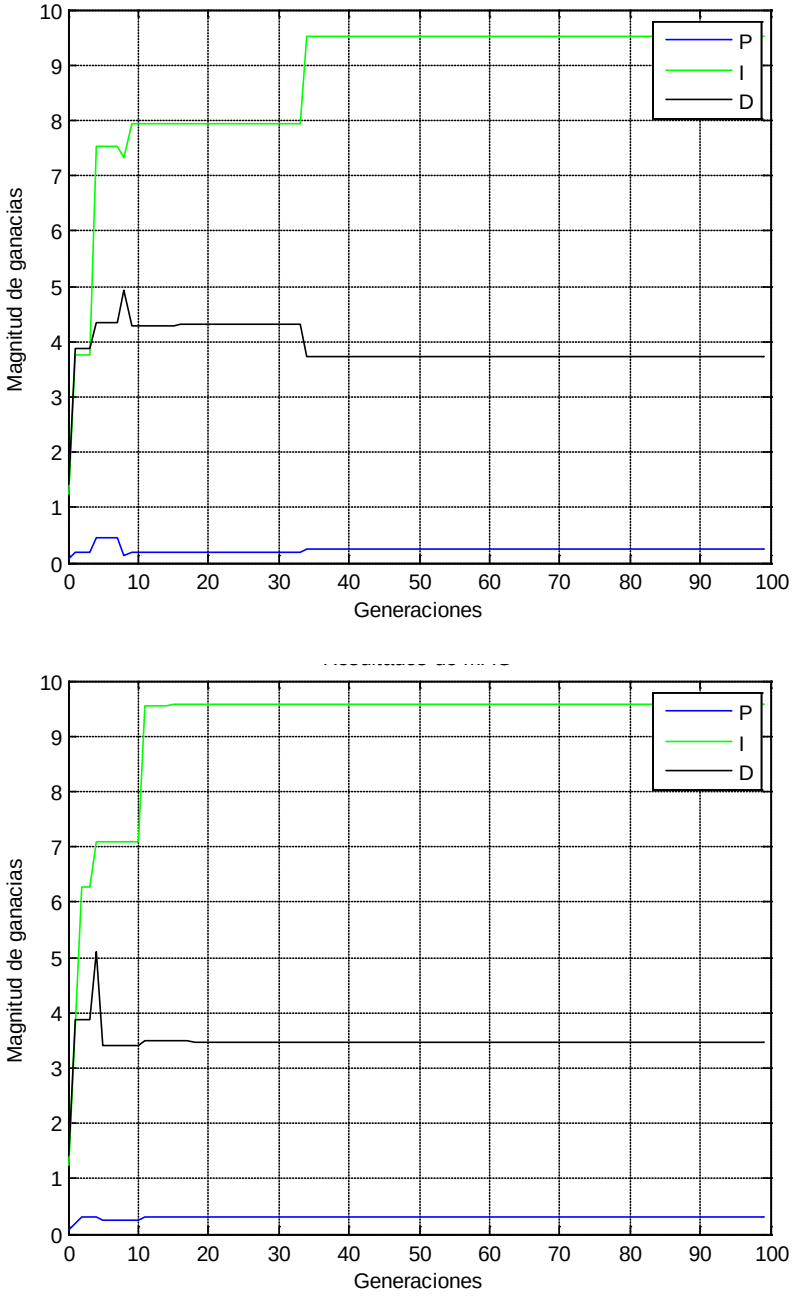


Fig. 56 Valores de las ganancias del controlador con el transcurso de las generaciones

Es importante mencionar que para hacer más apreciable el conjunto de valores de las ganancias, se hizo una escala en P e I, dividiendo su valor real entre mil. Los valores reales si se dicen en las tablas.

5. Conclusiones y prospectivas

5.1 Conclusiones

Los resultados permitieron demostrar que el módulo de MAG si funciona y que es manejable ante diferentes puntos de cruce y de mutación. Este módulo es de impacto tecnológico y de aplicación, pues permite obtener soluciones óptimas ante problemas no lineales. En este caso se demostró su robustez en el uso de máquinas y herramientas, siendo específicos para la sintonización del control de un servomotor y la obtención de los valores máximos de la función de Bohachevsky. Ya que el servomotor es utilizado en máquinas como una fresadora, es tomado como una prueba válida para verificar el módulo de MAG.

5.1.1 Conclusión acerca de los resultados experimentales con la función Bohachevsky

Al realizar un MAG en software fue importante para asentar los conceptos del algoritmo y fue puesta a prueba la función de Bohachevsky. Pero el punto era llevarlo a cabo en hardware Al realizar las pruebas en una función tridimensional con un punto óptimo global y varios puntos locales, era probable que el algoritmo fallara y convergiera en algún punto local. Pero al realizar una distribución equidistante en los valores fijos iniciales de los individuos *X* y *Y*, ofreció una buena respuesta ante la búsqueda de los valores óptimos.

El módulo demostró ser funcional y obtuvo muy buenas respuestas para la localización de los óptimos globales ante dicha función. Además, cabe mencionar que al realizar tantas pruebas ante diferentes puntos de cruce y mutación permitió realizar un análisis ante la situación en general del algoritmo MAG.

Es así como los mejores resultados fueron ante un punto de cruce y un punto de mutación y también al haber tres puntos de cruce y uno de mutación. Pero la mejor combinación fue la de tres puntos de cruce y uno de mutación al

tener una certeza del 80% al llevar al punto óptimo que era el de 0.0499 y el 20% restante solo se alejo del valor óptimo (0.0499) un 2%. Sin embargo, para un punto de cruce y uno de mutación, solo llegó al óptimo en un 65% y se alejo de la respuesta en un 35% con un 2% de margen de error.

Los peores resultados se obtuvieron en los puntos de cruce y mutación con mayor cantidad de puntos a la vez como fueron 2-4, 3-3, 3-4, (cantidad de puntos de cruce y mutación, respectivamente). Esto se debe a que al tener una gran cantidad de puntos de cruce y también de mutación, se crea una gran diversidad y se pierde información, sin llegar a la convergencia. Por ello, al tener tres puntos de cruce ya es suficiente para generar una diversidad a las soluciones y con un solo punto de mutación se logra modificar de manera pequeña pero significativa y así acercarse a la solución, en cambio si se tiene muchos puntos de mutación, se crea una solución muy variada, cayendo en soluciones erróneas.

Finalmente, los resultados del módulo MAG como sistema embebido fueron exitosos, permitiendo encontrar de manera satisfactoria el valor de las variables de forma óptima.

5.1.2 Conclusión acerca de los resultados experimentales con un servomotor

El módulo de forma práctica mostró ser funcional, encuentra las ganancias de un controlador PID de manera exitosa. Se probó el módulo con varias combinaciones de los posibles puntos de cruce y mutación, además se altero la probabilidad de mutación para demostrar la variación de los valores. Para ser más explícitos, las pruebas que fueron más destacadas son con los puntos 3-1, 2-4 y 3-4 (puntos de cruce y mutación, respectivamente). Los dos primero fueron de buenos resultados, todos llegaron a la referencia sin tener problemas, pero al tener 3-4 se tuvo una gran diversidad de soluciones perdiéndose de la respuesta óptima, no convergía.

Y aquellos que llegaron a tener sobrepaso no fue mayor a un 20%, además que se llego a la referencia exitosamente.

Como se mencionó en la conclusión de la sección pasada, los resultados son semejantes. Pero es importante destacar, que al tener un MAG se ahorra mucho tiempo, esto fue más claro en esta sección de pruebas, pues el probar a los cuatro individuos por una generación de 100, por ejemplo, se tiene un total de 400 pruebas, las cuales si se toma un tiempo probar para cada individuo un conjunto de propuesta de kp, ki y kd en el servomotor, y si se recuerda con un AGE serian 100 individuos por cada generaciones pudiendo llegar a ser mas de mil pruebas y eso tomaría mucho tiempo y recursos. Así el MAG ahora tiempo y recursos.

5.2 Prospectivas

Un MAG puede ser empleado para un problema en específico, no se puede emplear a todo tipo de problemas, es importante tenerlo en cuenta. Aun así, es aplicable a una gran cantidad de problemas, por ejemplo, como se realizó en este trabajó para la sintonización de un controlador y los valores de una función tridimensional.

Es posible mejorar el desarrollo del algoritmo, una mejor interfaz o aplicarlo en otros fenómenos físicos para desarrollar mejores conclusiones. También definir un número de generaciones en el que muestre un resultado óptimo ahorrando así más recursos y tiempo de desempeño y no realizar iteraciones innecesarias.

6. REFERENCIAS

Angeles, J. A. 2010. *Optimizacion del costo de las losas de concreto forzado aplicando algoritmos genéticos.*

Astudillo, N. C. 2010. *Algoritmos genéticos aplicados al diseño estructural de armaduras en tres direcciones.*

Brown, S. D., J. Francis, R., Rose, J., & Vranesik, Z. G. 1992. *Field-Programmable Gate Arrays.* Kluwer Academic .PubHshers.

Chong, E. K., & Zak, S. H. 2001. *An introduction to optimization.* New York: John Wiley & Sons.

Chu, P. P. 2008. FPGA prototyping by VHDL examples. New Jersey: WILEY.

Coello Coello, C. A., & Toscano Pulido, G. 2001. A micro-genetic algorithm for multiobjective optimization. *Springer-Verlag* .

Coello, C. A., & Pulido, G. T. 2001. A micro-genetic algorithm for multiobjective optimization. *Springer-Verlag* .

Coley, D. A. 1999. *An Introduction to Genetic Algorithms for Scientists and Engineers.* Singapore: World Scientific.

Cruz Peréz, Á. 2009. *Diseño óptimo de una microviga para mejorar su sensibilidad por medio de algoritmos genéticos.*

Cruz, R. G. 2007. *Algoritmos genéticos para la calibración de un modelo climático para invernadero.*

Gwiazda, T. D. 2007. *Genetic Algorithms Reference Volumen II. Mutation operator for numerical optimization problems.* Polonia: Tomaszugiazda.

Hurtado, A. Z. 2011. *Estructuras de las casas de madera optimizadas con algoritmos genéticos.*

IEEE. 2008, Septiembre. *IEEE PROJECT. P1076 - Standard for VHDL Language Reference Manual.* Retrieved Marzo 5, 2014, from https://standards.ieee.org/develop/project/1076.html

IEEE. 2008, Septiembre. *IEEE STANDARD. 1076-2008 - IEEE Standard VHDL Language Reference Manual.* Retrieved Marzo 5, 2014, from http://standards.ieee.org/findstds/standard/1076-2008.html

Jaen Cuellar, A. Y., Romero Troncoso, R. d., Morales-Velazquez, L., & Osornio-Rios, R. A. 2013. PID-Controller Tuning Optimization with Genetic Algorithms in Servo Systems. *INTECH* .

K. P. Chong, E., & H. Zak, S. 2001. *An introduction to optimization.* New York: John Wiley & Sons.

Meng, X., & Song, B. 2007. Fast Genetic Algorithms used for PID parameter Optimization. *International Conference on Automation and Logistics.*

Nazir, A. J., Gauthman, Surajan, R., & S, B. L. 2009. A simplified Genetic Algorithm for online tuning of PID controller in LabView. *World Congress on Nature & Biologically Inspired Computing.*

Ogata, K. 2010. *Ingeniería de control moderna.* Pearson.

Ortiz, C. R. 2011. *Diseño e implementación de un algoritmo genético en FPGA para sintonizacion de controladores PID.*

Rao, S. S. 2009. *Engineering Optimization. Theory and practice.* New Jersey: John Wiley & Sons.

Renner, G., & Ekárt, A. 2002. Genetic algorithms in computer aided design. *ELSEVIER* .

Sass, R., & Schmidt, A. G. 2010. Embedded Systems Design with Platform FPGAs principles and practices. Morgan Kaufmann.

Thompson, M. 2012. Introduction to FPGA Technology: Top 5 Benefits. *National Instruments* .

Troncoso, R. d. 2007. Electrónica digital y lógica programable. Guanajuato: DR Universidad de Guanajuato.