Universidad Carlos III de Madrid

Localización de robots en mapas 3D mediante CLONALG

Trabajo Fin de Grado

Autor: Enrique Fernández Rodicio
Tutor: Fernando Martín Monar

RESUMEN

El objetivo de este Trabajo Fin de Grado es aplicar el Algoritmo de Selección Clonal (comúnmente conocido como CLONALG) al filtro de localización global de alta precisión para entornos 3D, también llamado RELF-3D, desarrollado por Fernando Martín, Luis Moreno, Santiago Garrido y Dolores Blanco, investigadores de la Universidad Carlos III de Madrid.

Una vez se desarrolle el algoritmo genético y se integre en el programa completo ya existente, se pasará a realizar pruebas para poder comparar los resultados obtenidos con el CLONALG y los obtenidos mediante el algoritmo de Evolución Diferencial, y así poder tener una base para decidir cuál de los dos métodos es más eficaz a la hora de cumplir con su propósito.

En esta memoria, se comenzará con una introducción al trabajo, en la que se expondrán los objetivos del proyecto, se verá el trabajo previo y el estado del arte. En el siguiente apartado, se le proporcionará al lector una pequeña base en el campo de los algoritmos genéticos, para posteriormente explicar con mayor profundidad el método de Evolución Diferencial, comenzando por las bases teóricas y acabando con un estudio acerca del algoritmo implementado en el programa.

A continuación, se llega a la parte central de este trabajo, el CLONALG. Se entrará en profundidad en el funcionamiento del algoritmo, tanto teórica como prácticamente, además de explicar las diferentes partes del código desarrollado. En este apartado también se expondrá el proceso de desarrollo de la solución, las decisiones de diseño y los problemas a los que fue necesario hacer frente para alcanzar la solución.

Por último, se compararán los resultados obtenidos por ambos métodos, además de realizar un análisis de sensibilidad del CLONALG con respecto a una serie de parámetros, y se cerrará la memoria con una conclusión acerca de los resultados obtenidos en la investigación.

ABSTRACT

The objective of this Final Project is to apply the Clonal Selection Algorithm (commonly known as CLONALG) to the Rejection Evolutionary Localization Filter in 3D, also called RELF-3D, developed by Fernando Martín, Luis Moreno, Santiago Garrido and Dolores White, researchers at the Carlos III University of Madrid.

Once the genetic algorithm is developed and integrated into the existing full program, tests will be performed in order to compare the results obtained with CLONALG and those obtained by the algorithm Differential Evolution, so we can have a basis for deciding which of the two methods is more effective in fulfilling its purpose.

In this report, we will begin with an introduction to the work, in which the project objectives will be discussed, previous work and the state of the art will be seen. In the next section, we will provide the reader a small introduction into the field of genetic algorithms to further explain in detail the method of Differential Evolution, beginning with the theoretical bases and ending with a study of the algorithm implemented in the program.

Then we will reach the central part of this work, the CLONALG. It will go into depth on the performance of the algorithm, both theoretically and practically, in addition to explaining the different parts of the code developed. This section describes the development process of the solution and also expose the design decisions and the problems it was necessary to solve to reach the solution.

Finally, the results obtained by both methods, in addition to an analysis of CLONALG sensitivity with respect to a number of parameters are compared, and the report will end with a conclusion about the results of the investigation.

AGRADECIMIENTOS

El Trabajo Final de Grado supone la conclusión de mis estudios en el Grado en Ingeniería de Tecnologías Industriales y el inicio de una nueva etapa en mi formación como estudiante del Máster Universitario en Robótica y Automatización. Llegar hasta este punto no habría sido posible sin el apoyo de una serie de personas, a las que me gustaría dar las gracias.

A mis padres, José Aurelio y Mercedes, y a mis hermanas, Sabela y Marta, por todo el apoyo que me han dado durante estos cuatro años, tanto cuando las cosas iban bien, como en los momentos más difíciles. Siempre han hecho todo lo posible para ayudarme a conseguir llegar hasta el final.

A mis amigos y a mis compañeros de piso, los cuales han tenido que soportar los momentos de frustración que aparecen a lo largo de cualquier proyecto, cuando los resultados no acompañan y no parece verse el fin, y siempre han tenido palabras de apoyo que ayudaron a seguir adelante.

A mis compañeros en la universidad, que me ayudaron a superar numerosas asignaturas a lo largo de la carrera, y en los que siempre me he podido apoyar cuando ha sido necesario.

A mi tutor, Fernando Martín, por estar siempre dispuesto a echarme una mano cuando me encontraba con algún problema durante la realización del proyecto, ayudando así a que haya sido capaz de completar este trabajo y a que haya podido aprender acerca de un tema apasionante.

Por último, a la Universidad Carlos III de Madrid, a los profesores y trabajadores y a todas las personas que, a lo largo de estos cuatro años, han puesto su parte para conseguir que esta haya sido una de las mejores etapas de mi vida.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE ECUACIONES

LISTADO DE ACRÓNIMOS

- **TFG:** Trabajo Fin de Grado
- **CLONALG:** CLONal selection ALGorithm
- **ED:** Evolución Diferencial
- **RELF-3D:** Rejection Evolutionary Localization Filter in Three Dimensions
- **UC3M:** Universidad Carlos III de Madrid
- **GPS:** Global Position System
- **MATLAB:** MATrix LABoratory
- **JIT:** Just In Time
- **MANFRED:** MAN FRiEnDly mobile manipulator
- **SLAM:** Simultaneous Localization And Mapping
- **MCL:** Monte Carlo Localization
- **PSO:** Particle Swam Optimization
- **OBDE:** Opposition-Based Differential Evolution
- **DEGL:** Differential Evolution with Global and Local neighborhoods
- **SADE:** Self Adaptive Differential Evolution
- **JADE:** Adaptive Differential Evolution with Optional External Archive
- **SFLSDE:** Scale Factor for Local Search Differential Evolution
- **AIRS:** Artificial Inmune Recognition System
- **BCA:** B-Cell Algorithm

1. INTRODUCCIÓN AL TRABAJO FIN DE GRADO

- Robótica móvil y navegación

Un robot móvil es aquella máquina automática que puede moverse en un determinado entorno. En función de la forma en la que se mueva se pueden dividir en diferentes categorías: robots rodantes (emplean ruedas para moverse), andantes (emplean dos o más extremidades para desplazarse), reptantes (se arrastran por el suelo), nadadores, voladores…

En el campo de la robótica móvil, se conoce como navegación al guiado de un robot que parte de un punto inicial, a través de un entorno en el que existen ciertos obstáculos a evitar, para alcanzar el punto final deseado. Para que se pueda cumplir con un cierto éxito esta tarea, es necesario poder resolver cuatro grandes problemas [1]:

- **Se necesita proporcionar al robot un mapa del entorno que posea una precisión adecuada**. Este puede haber sido realizado de antemano, o generado por el propio robot, empleando para ello los diversos sensores con los que cuenta, al mismo tiempo que avanza. Esta técnica es conocida como Simultaneous Localization And Mapping, o SLAM, que busca que el robot pueda ser capaz de generar el mapa de su entorno y, al mismo tiempo, pueda localizarse con éxito dentro de ese mapa. De entre todas las técnicas empleadas para su solución, las más efectivas son las basadas en técnicas probabilísticas, que parten del teorema de Bayes, para poder tener en cuenta las distintas fuentes de incertidumbre que van apareciendo a lo largo del problema. Los principales algoritmos en este campo son el Filtro Extendido de Kalman, los mapas de ocupación de celdillas o la solución factorizada del filtro de Bayes [2].

- **El robot debe ser capaz de localizarse dentro del mapa del entorno.** Si para generar dicho mapa se empleó la técnica de SLAM, el punto anterior y este se resuelven de forma simultánea. Si el mapa fue generado de forma externa, debe emplearse un sistema que permita al robot conocer su posición. En este punto es en el que se va a centrar este trabajo. Las diferentes opciones que se presentan a la hora de resolver este problema se discutirán con mayor profundidad en la sección 2 de esta memoria, dedicada al estudio del estado del arte en este campo.

- **El robot debe ser capaz de elegir una ruta libre de obstáculos para poder desplazarse desde el punto inicial al punto final.** Para resolver este problema, se puede tener en cuenta las características de movimiento del robot a emplear o se puede buscar una solución basada simplemente en el mapa del entorno y los puntos inicial y final. Dicha planificación de ruta se puede plantear como un

problema de optimización, expresando los parámetros relevantes en forma de grafo (conjunto de puntos, también llamados vértices o nodos, unidos por líneas, llamadas aristas, que permiten expresar la relación entre elementos de un conjunto) y, por medio de un algoritmo, buscar minimizar la función de coste asociada a uno de los caminos de dicho grafo [3].

- **Por último, el robot debe ser capaz de seguir la ruta que planificó anteriormente.** Esto supone resolver la forma de adecuar dicha ruta a los parámetros que caracterizan el movimiento del robot (en caso de no haberlo hecho a la hora de panificar la ruta), además de poder mantener un seguimiento de su localización a lo largo del desplazamiento. Además, debe seguir recibiendo información del entorno a través de los sensores para poder reaccionar adecuadamente a la aparición de un obstáculo imprevisto.

- Introducción al trabajo y estructura de la memoria

Este estudio está enmarcado en la parte de la navegación conocida como localización, y busca aportar una solución al problema de la localización global del robot dentro de un entorno conocido. Esta solución consiste en hallar su posición y orientación dentro de un entorno conocido, sin tener ningún dato inicial acerca de estas coordenadas. Con este trabajo, se busca ofrecer una solución alternativa a los resultados obtenidos mediante el Rejection Evolutionary Localization Filter in 3D [4], desarrollado por investigadores de la Universidad Carlos III de Madrid, el cual emplea el método de Evolución Diferencial, que imita la Teoría de la Evolución de Darwin para resolver problemas de optimización. Se tratará más en profundidad este algoritmo en la sección 2.2.

Para lograr este objetivo, se va a trabajar con el Algoritmo de Selección Clonal, también conocido como CLONALG, propuesto por primera vez por Leandro N. De Castro y Fernando J. Von Zuben [5]. Este algoritmo está basado en el funcionamiento del sistema inmunológico humano y en la Teoría de la Selección Clonal, según la cual, el propio organismo es capaz de seleccionar aquellos linfocitos que han probado ser útiles contra un determinado antígeno, y clonarlos para obtener nuevos individuos a partir de un proceso de mutación, mientras que aquellos que no han probado ser útiles son reemplazados por nuevos individuos. Esto provoca que todo el sistema obtenga una gran capacidad de adaptación y una velocidad de respuesta creciente ante cada nuevo ataque de antígenos ya conocidos.

El trabajo previo desarrollado por Fernando Martín, Luis Moreno, Santiago Garrido y Dolores Blanco [4] emplea, para la resolución del problema de localización en un robot móvil, información obtenida a partir de los sensores que posee el robot Man Friendly mobile manipulator 2, o MANFRED-2, al proporcionar el tipo de datos necesario para esta aplicación, por lo que esta investigación (al partir de ese mismo trabajo) trabajará sobre los

mismos sensores. Se entrará con mayor profundidad en sus especificaciones en la sección 1.2.

La estructura de esta memoria es la que sigue:

- Una introducción al Trabajo Fin de Grado en la que se exponen los objetivos de este y las motivaciones que influyeron en la realización de este proyecto. Además, también se incluye una sección en la que se explicarán en profundidad las herramientas empleadas y una sección en la que se detalla el presupuesto y la planificación del proyecto.

- Una sección dedicada al estudio del estado del arte y de los estudios previos en los que se basa este trabajo. Veremos las distintas familias de algoritmos que se pueden emplear para solucionar el problema de la localización global de un robot, como pueden ser los métodos basados en distribuciones bayesianas, los basados en métodos de optimización y los híbridos. Se comentarán algunos de los trabajos en este campo.

- Una sección dedicada a proporcionar al lector ciertos conocimientos en el campo de los algoritmos genéticos. Se proporcionará una pequeña base teórica para comprender su funcionamiento, entrando con mayor profundidad en el método de Evolución Diferencial, método empleado en el RELF-3D para obtener la posición del robot. En esta sección se estudiará el algoritmo desarrollado por el grupo de investigadores de la UC3M para conocer a fondo su funcionamiento.

- La sección central de este trabajo está dedicada en exclusiva al CLONALG. Se comenzará dando a conocer los Sistemas Inmunes Artificiales y la Teoría de Selección Clonal, que explica el funcionamiento del sistema inmunológico, y a partir de la cual se desarrolló dicho método. Se presenta también el algoritmo desarrollado para obtener la solución deseada y se comentará el proceso de diseño seguido para alcanzar dicha solución, además de los problemas a los que se fue haciendo frente durante la investigación.

- Una sección de resultados experimentales. En ella se expondrán los datos recogidos empleando ambos métodos, ED y el CLONALG. Se llevará a cabo un análisis de sensibilidad con el CLONALG, comprobando su respuesta ante la variación de distintos parámetros del algoritmo. Por último, se compararán los resultados obtenidos para conocer cuál es la mejor solución.

- Para finalizar, se llevará a cabo una pequeña conclusión donde se resumirán los resultados obtenidos y se cerrará este trabajo.

1.1. Objetivos y motivación del Trabajo Fin de Grado

Dentro de la robótica móvil, uno de los primeros problemas a resolver es el desarrollo de sistemas de localización más y más precisos. Para esta tarea no es posible emplear las herramientas que se aplican en otros campos, como puede ser la localización por Global Position System, o GPS, debido a que no presentan una precisión adecuada a la escala en la que se mueve esta rama de la robótica, o que no es apta para todas las situaciones, como en entornos subterráneos donde no se pueda captar la señal, si volvemos al ejemplo del GPS, por lo que es necesario desarrollar nuevos sistemas que cumplan esta función. De esta necesidad surge la motivación que impulsa esta investigación, que busca aplicar un nuevo algoritmo de optimización para comprobar si es posible mejorar los resultados obtenidos previamente empleando el RELF-3D. Este proyecto presenta dos objetivos, expuestos a continuación:

- Aplicar el Algoritmo de Selección Clonal, conocido como CLONALG, al proyecto RELF-3D para conseguir la correcta localización de un robot situado en un entorno conocido. Se comprobará como los diferentes parámetros configurables en el algoritmo afectan al éxito que el filtro presenta a la hora de enfrentarse a la tarea impuesta, además de comprobar el coste computacional del mismo.

- Una vez el algoritmo ha sido desarrollado e implementado con éxito en el programa ya existente, el siguiente objetivo busca establecer una comparación efectiva entre los resultados obtenidos por medio del CLONALG y los que se obtuvieron empleando el algoritmo con el que trabaja el RELF-3D, basado en el método conocido como Evolución Diferencial.

Con el cumplimiento de estos dos objetivos, este trabajo expondrá las ventajas e inconvenientes de ambos métodos, aportando datos experimentales, apoyados con gráficas para facilitar su rápida comprensión, de forma que se pueda afirmar con claridad cuál de los dos métodos cumple de forma más efectiva con su cometido, y si se debe emplear el CLONALG como método de resolución del problema de localización, o por el contrario, descartarlo como posibilidad y continuar explorando las diferentes opciones que permitan obtener la mejor solución posible.

1.2. Herramientas empleadas

Para efectuar esta investigación, el trabajo de programación se ha realizado sobre el software Matrix Laboratory, comúnmente denominado MATLAB, desarrollado por la compañía Mathworks Inc, mientras que el programa trabaja con el mismo tipo de

información que proporcionan los sensores implementados sobre el robot MANFRED-2, desarrollado por la UC3M.

MATLAB es un programa utilizado para la realización de cálculos con vectores y matrices, además de permitir realizar gráficos en dos y tres dimensiones, entre otras cosas. Fue desarrollado inicialmente por Cleve Moler en 1984 como una forma de agrupar subrutinas escritas en Fortran para poder emplearlas de forma más cómoda en cursos de álgebra lineal y análisis numérico [6]. Emplea su propio lenguaje de programación, el cual puede resultar muy rápido en algunas operaciones, cuando las funciones a ejecutar están escritas en código nativo y presentan los tamaños óptimos, aprovechando así sus capacidades de vectorización. A partir de la versión 6.5 se incorporó un acelerador JIT (Just in time) para mejorar la velocidad de ejecución. [7]

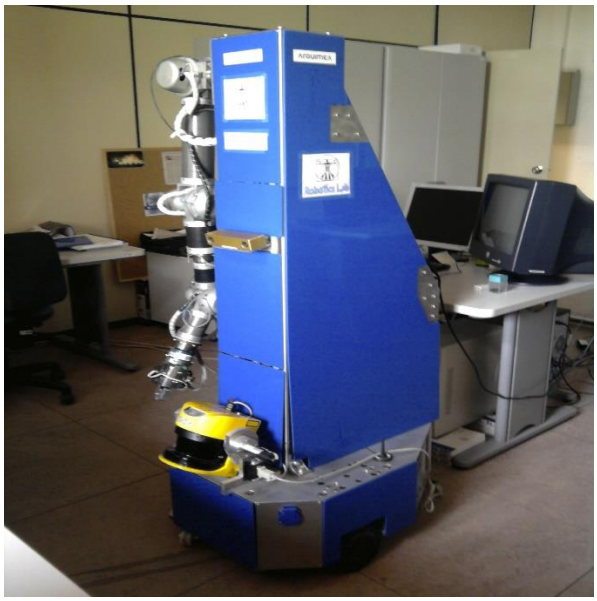

Figura 1 Foto del robot MANFRED-2

MANFRED-2, como se observa en la Figura 1, es un manipulador móvil usado en la investigación y desarrollo en el campo de los robots móviles. Está pensado para poder trabajar en entornos en los que se precisaría de la capacidad de manipulación de un humano, siendo capaz además de moverse de forma autónoma dentro de un mapa conocido. Está diseñado de forma modular, a imagen de los robots que se usan en exploración espacial y satélites de comunicaciones, formados por varios subsistemas unidos para que el sistema completo funcione. Además se le pueden añadir nuevos instrumentos en función de la aplicación que se le quiera dar. Presenta 8 grados de libertad (dos de ellos en la base y los otros seis en el brazo antropomórfico). Algunas de las tareas que puede hacer son las siguientes: Abrir y cruzar puertas, esquivar obstáculos y manipular objetos [8].

Para este trabajo en concreto se van a emplear una serie de sensores que ya fueron empleados en el desarrollo del RELF-3D, y son los siguientes [4]:

- Un telémetro (SICK PLS) que proporciona información en 2D acerca del entorno. Añadiéndole un motor que permita girar este escáner, se pasa a obtener una medida 3D. En dirección horizontal (cada una de las medidas 2D que se toman), la resolución máxima es de 0,25°, mientras que su amplitud es de 190°. En la dirección vertical, la resolución es de 1°. Además se ha implementado otro escáner simulado, que permite realizar trece escaneados verticales con una resolución de 5°, cada uno de ellos con sesenta y una medidas separadas por 3°

- Un encoder óptico modelo HEDS-5540 que nos proporciona la información odométrica, aunque presenta una precisión muy baja y un error que se va acumulando con el tiempo.

2. ESTADO DEL ARTE

A la hora de enfrentarse a la resolución del problema de la localización global en robots móviles, se pueden encontrar distintas familias de algoritmos con este fin. Entre ellas podemos destacar los métodos basados en el concepto de probabilidad bayesiana y los métodos que plantean la localización como un problema de optimización. Por otra parte, también existen técnicas que permiten el mapeo del entorno al mismo tiempo que se busca la localización del robot, conocidas como SLAM. Además se han desarrollado métodos que combinan varias de estas técnicas para lograr la localización.

2.1. Soluciones basadas en la aplicación del teorema de Bayes

El Teorema de Bayes expone que la probabilidad que la probabilidad de que se dé un suceso aleatorio A conociendo que otro suceso B ya ha ocurrido depende de la distribución de probabilidad condicional de B si A ha ocurrido y de la distribución de probabilidad marginal de A.

En estos métodos, el robot busca estimar la probabilidad a posteriori (probabilidad condicional de un evento A sobre B, que se asigna después de tener en cuenta las evidencias existentes) acerca del espacio de posibles soluciones. Esta distribución varía en función de los datos que se conocen acerca del problema, datos que pueden ser medidas de los sensores que captan la posición del robot en un instante determinado, o la información odométrica (datos obtenidos de un sensor en movimiento para obtener información acerca de dicho movimiento).

El primer paso para determinar esta probabilidad a posteriori de forma recursiva consiste en aplicar el teorema de Bayes al último elemento del vector Y_t, en el cual se almacenan los datos que proporcionan los sensores. Suponiendo que la medida z_t (última medida obtenida por los sensores del robot) es independiente de las medidas previas, dado el estado x_t (entendiendo por estado la posición y orientación del robot en un instante dado), se puede observar en la ecuación 1 como se calcularía la probabilidad, donde $p(x_t/Y_t)$ es la probabilidad de que se dé el estado x_t dadas las medidas almacenadas en el vector *Y* en el mismo instante de tiempo, $p(z_t/x_t)$ representa la probabilidad de que se dé la medida *z* dado el estado *x* en el instante *t*, $p(x_t/Y_{t-1})$ es la probabilidad de que se dé el estado *x* en el instante *t* dadas las medidas almacenadas en *Y* en el instante anterior, y $p(z_t/Y_{t-1})$ es la probabilidad de que se dé la observación z en el instante *t* dadas las medidas almacenadas en *Y* en el instante anterior.

$$p(x_t/Y_t) = \frac{p(z_t/x_t)p(x_t/Y_{t-1})}{p(z_t/Y_{t-1})}$$

Ecuación 1. Probabilidad de que se cumpla el estado x dado Y en el instante t

En el segundo paso, se tiene que comprobar como varía el estado x para el instante t+1 conociendo las medidas almacenadas en el vector Y en el instante t y el estado x para ese mismo instante. Esto se consigue integrando para todo el espacio de los números reales la probabilidad de que se dé el estado x_{t+1} si se conoce el estado anterior x_t y los datos procedentes del movimiento del robot en ese instante, conocidos como u_t, y la posibilidad de que se dé x_t conociendo el vector de medidas Y_t, suponiendo que x_{t+1} es independiente de Y_t, con lo que cumpliría la propiedad de Markov. Esto se ve en la ecuación 2.

$$p(x_{t+1}/Y_t) = \int_{R^n} p(x_{t+1}/x_t, u_t) p(x_t/Y_t) dx_t$$

Ecuación 2. Probabilidad de que se cumpla el estado x en el instante t+1 dado Y en el instante t

Por lo tanto, de cada uno de los elementos de la población con los que se trabaja se puede extraer una probabilidad a posteriori con los datos conocidos en el instante actual. Después, a partir de esta probabilidad se obtiene una estimación de la posición futura del robot.

Dentro de los métodos de localización bayesianos podemos señalar una serie de algoritmos agrupados dentro de lo que se conoce como Markov Localization Methods, donde destacan el método conocido como Monte Carlo Localization, de ahora en adelante MCL, el conocido como Grid-based Localization Method y el Topological Markov Method. Además también existe una aplicación del filtro de Kalman para la resolución del problema de localización cuando se conoce la posición inicial del robot.

- ### Filtro de Kalman

Cuando se conoce la posición inicial del robot, el filtro de Kalman supone una forma realmente precisa de mantener el control sobre esta posición, suponiendo que esta siga una distribución Gaussiana unimodal, donde la media representa la posición del robot, mientras que la varianza representa la incertidumbre existente.

Cuando el robot se desplaza, la distribución también cambia, en función de dicho movimiento, mientras que la varianza se incrementa según el modelo matemático que representa la odometría del robot. Existen varios modelos distintos de este filtro en función de la forma en que se introducen en la estimación de la posición las nuevas medidas captadas por los sensores., mientras que la mayoría coinciden en la forma de modelar el movimiento.

La aplicación de un filtro de Kalman ofrece una gran robustez cuando lo único que se busca es seguir la posición del robot durante el movimiento. Sin embargo no ofrece una solución al problema de la localización global, y si se da el caso de que el algoritmo pierde la

posición del robot en algún momento, no sería capaz de recuperarla. Además pierde efectividad si la representación del espacio de soluciones no sigue una distribución Gaussiana, o el error proveniente de los sensores de medida no sigue esta distribución.

Ejemplos del uso del filtro de Kalman se puede encontrar en el trabajo de J. Leonard y H. Durrant-Whyte [9], en el que desarrollan un método de localización que busca la concordancia entre puntos del mapa del entorno geométricamente parametrizables y reconocibles y los datos obtenidos mediante los sensores implementados en el robot. R. Negenborn [10] realizó un estudio acerca de las ventajas e inconvenientes de la aplicación de filtros de Kalman para la resolución del problema de localización en robots móviles. E. Kiriy y M. Buehler [11] aplicaron el filtro extendido de Kalman en la localización de un cortacésped para un campo de golf, con la ayuda de sensores de visión.

- Markov Localization Methods

Estos métodos son un caso especial de estimaciones probabilísticas aplicadas a la localización de robots móviles. Parten de la hipótesis de Markov, que supone que el entorno permanecerá sin cambios durante todo el proceso, por lo que lo único que afectará a los sensores del robot es su propia localización. Mantienen una distribución de probabilidad sobre el espacio de soluciones, en vez de trabajar exclusivamente con una posible solución. Después, en función de los datos que se van recogiendo a partir del movimiento de los sensores, se aumenta la probabilidad de los puntos que coincidan con estos datos, mientras que se baja en los que no coinciden.

Por lo general, las dos principales diferencias que existen entre los distintos algoritmos que se basan en la hipótesis de Markov son la forma de representar el espacio de soluciones y la forma de calcular las probabilidades condicionales del estado del robot en función de los datos odométricos y de las medidas de los sensores del robot en función del estado actual.

a) Topological Markov Method

En este método, se representa el espacio de soluciones atendiendo a las características topológicas del entorno en el cual se intenta localizar al robot. De esta forma, se emplean los puntos más característicos del entorno como referencia para calcular la probabilidad a posteriori de los estados del robot.

J. Košecká y F. Li [12] propusieron un método de localización y mapeo simultáneo (SLAM) en el que obtienen un modelo topológico del entorno empleando técnicas de visión y comparan varias técnicas de reconocimiento para ver cuál es más efectiva.

b) Grid-based Localization Method

Según este método, el espacio de soluciones se modela como una cuadrícula en 3D, donde la resolución espacial (tamaño de la cuadrícula) y angular caracterizan la precisión que es posible obtener con este método. Cada capa de la cuadrícula representa todas las posibles soluciones que presentan la misma orientación, mientras que cada celda representa una posible posición del robot. A cada una de las celdas se le asigna un valor que representa la probabilidad de que el robot se encuentre en ese estado.

Uno de los inconvenientes de este método es que la resolución, y por lo tanto la precisión, debe ser fijada de antemano, al discretizar el espacio de soluciones para obtener la cuadrícula. Además, no todas las medidas se pueden procesar en tiempo real, por lo que se podría descartar información valiosa acerca del estado del robot.

Se puede encontrar un buen ejemplo de este método en el trabajo desarrollado por D. Fox, W. Burgard y S. Thrun [13], en el que se presenta una aplicación de la localización de Markov para entornos dinámicos.

c) Método Monte Carlo

El método de localización conocido como Monte Carlo [14] emplea un filtro de partículas para, dado un entorno conocido, representar una distribución de estados similares, donde cada partícula representa un estado, entendiendo por estado una posible posición del robot. Al igual que muchos de los métodos probabilísticos, los métodos Monte Carlo asumen que los estados cumplen la propiedad de Markov, por lo que cada estado es solo función del estado inmediatamente anterior. Esto solo se cumple si el entorno permanece sin cambios durante todo el proceso de localización.

Comienza con una distribución uniforme de las partículas sobre el espacio de soluciones, considerando que existe la misma probabilidad de que el robot se encuentre en cualquiera de ellas. Una vez el robot comienza a moverse y se obtienen datos odométricos, el filtro varía la posición de las partículas para obtener la nueva posición del robot una vez completado este movimiento. Esto se consigue aplicando a cada partícula el mismo movimiento que realiza el robot.

Cuando se recibe una nueva medida de los sensores, se comprueba si las soluciones actuales encajan con la medida recibida, y en función de esto se eliminan aquellas partículas que no coinciden con estas medidas y se generan unas nuevas que estén más cerca de la solución buscada. Después de repetir este proceso un par de veces, el conjunto de partículas acaba convergiendo alrededor de la posición real del robot.

Las principales características de la versión básica de este método son la no parametricidad (no está basado en familias parametrizadas de distribuciones de

probabilidad), lo que permite que trabaje con diversos tipos de distribuciones de probabilidad distintas, y reduce el uso de memoria, comparado con los métodos basados en cuadrícula (grid-based localization methods), ya que solo depende de la cantidad de partículas y no del tamaño del mapa.

Un ejemplo de la aplicación del MCL es el trabajo desarrollado por F. Dellaert, D. Fox, W. Burgard y S. Thrun [15], donde se estudia su uso para la localización de robots móviles en entornos interiores, o en el trabajo de R. Kümmerle, R. Triebel, P. Pfaff y W. Burgard [16], quienes aplican el MCL para resolver el problema de localización cuando el robot se encuentra en un entorno exterior.

2.2. Soluciones basadas en métodos de optimización

Estos métodos plantean el problema de la localización como la solución a un problema de optimización. Para ello, se debe definir una función matemática que valore cada uno de los estados que forman el espacio de soluciones, en función de su calidad como solución. Después se buscará el estado que proporcione el valor óptimo para dicha función. Dentro de esta categoría podemos encontrar los filtros basados en Evolución Diferencial, de los que se hablará con mayor profundidad en la sección 3, y los filtros conocidos como Particle Swam Optimization filters, o PSO.

El método de optimización por enjambre de partículas (PSO), es una técnica heurística de optimización que se basa en el comportamiento de las abejas. Fue desarrollado inicialmente por J. Kennedy y R. Ebehart [17] en 1995.

Se comienza por generar una población formada por posibles soluciones que se van desplazando por el espacio de soluciones del problema en cuestión. Para cada una de las partículas, se le asigna un movimiento inicial aleatorio, que esté contenido dentro de los límites del problema. Después se inicializa la mejor posición de cada partícula, la mejor velocidad de cada una y la mejor posición en global.

Cuando ya se tiene la población inicial, se entra en el bucle principal del algoritmo. Se selecciona una de las partículas y, para cada una de las dimensiones del problema, se recalcula la velocidad de la partícula en función de dos números aleatorios. En función de esta velocidad se varía la posición de la partícula. Cuando ya se conoce la nueva posición, se calcula el valor que devuelve la función de la que se pretende obtener la solución óptima y, en caso de que este valor mejore el obtenido a partir de la mejor posición de esa partícula hasta el momento, la nueva posición pasa a sustituir a esta. Una vez se ha repetido este proceso para todos los elementos de la población, se comprueba si alguna de las soluciones actuales mejora la considerada como mejor solución global, y si se da el caso, se almacena la nueva mejor posición. Este ciclo se repetirá hasta que se cumplan las condiciones de parada consideradas para la aplicación en cuestión.

F. Jovan, R. Y. S. Dhiemas, M. Satki Alvissalim, W. Jatmiko, M. I. Fanany, A. Febrian, K. Sekiyama y T. Fukuda [18] realizaron un estudio para comprobar la robustez de este método, aplicando la técnica de PSO a varios robots móviles con sistema de comunicación inalámbrica. R. Havangi, M. Ali Nekoui y M. Teshnehlab [19] buscaron solucionar la perdida de diersidad en la población en lo métodos basados en filtros de partículas empleando la técnica de PSO.

2.3. Simultaneous Location And Mapping (SLAM)

SLAM es una técnica que permite a un robot autónomo obtener un mapa del entorno a la vez que se sitúa en él. Aunque existen varios posibles planteamientos de este método, los que mejores resultados han demostrado obtener son aquellos que emplean técnicas probabilísticas, basadas en el Teorema de Bayes.

Una de las soluciones más extendidas consiste en la aplicación de un filtro extendido de Kalman. En este método, el estado del robot consiste en su posición en ese instante y en las medidas recogidas por los sensores.

Lo primero es obtener una estimación de la posición en el que se encontrará el robot en el siguiente instante, a partir de los datos obtenidos acerca del movimiento del robot. Lo siguiente es estimar las medidas del entorno que se obtendrán en este nuevo estado. Una vez el robot ha completado el movimiento, se toman las medidas reales y se compara con la estimación anterior. Por último, se actualiza el vector que almacena el estado del robot en función de las estimaciones realizadas y las medidas recogidas.

S. Se y D. L. J. Little [20] plantean un método de SLAM basado en técnicas de visión que utiliza puntos reconocibles de forma natural en el mapa, en vez de generar estos puntos de forma artificial. Y. J. Lee y S. Sung [21] también utilizan técnicas de visión, en este caso, combinadas con encoders para obtener la información odométrica.

3. ALGORITMOS GENÉTICOS. EVOLUCIÓN DIFERENCIAL

Se conoce como algoritmo al conjunto de pasos organizados que muestran el proceso a seguir para obtener la solución a un determinado problema. Su aplicación se encuentra en múltiples disciplinas, como son las matemáticas, la lógica, las ciencias de la computación y otras disciplinas relacionadas con estas. Entre los distintos tipos de algoritmos encontramos los algoritmos genéticos, una de las principales líneas de investigación en el campo de la inteligencia artificial.

3.1. Introducción a los algoritmos genéticos

Los algoritmos genéticos reciben ese nombre porque están basados en la evolución de las especies y en la selección natural, postulada por Darwin en la Teoría de la Evolución. Según esta teoría, en una población de individuos, aquellos que puedan ser capaces de adaptarse a los cambios que se presenten en el entorno serán los individuos que sobrevivan. Esta adaptación se debe a una serie de variaciones en los genes de dichos individuos, los cuales se transmiten a sus descendientes al reproducirse.

Imitando el funcionamiento de la naturaleza, estos algoritmos generan una población de elementos que se encuentran contenidos en el espacio de soluciones del problema que se pretende resolver, y cuyos genes representan las distintas variables de decisión en dicho problema. Después, se somete a esta población a una serie de cambios aleatorios a través de procesos de mutación y de recombinación genética, y por último se lleva a cabo una selección de los mejores individuos en función de los parámetros definidos. Tras estos pasos se obtiene una nueva generación de individuos que presentará una solución mejor, o, en el peor de los casos, igual de acertada, al problema en cuestión.

De esta forma, con cada nueva generación se mantendrá en la población a aquellos individuos que representen una solución más acertada, y se continuará optimizando la solución en las zonas del espacio de soluciones que ocupan estos elementos. Esto provocará que con el tiempo, la solución del problema converja a un punto óptimo. En la Figura 2 se puede observar el pseudocódigo para un algoritmo genético simple.

```
1
2   function algoritmo_genetico()
3
4       POP_INICIAL
5       COSTE_POP_INICIAL
6
7       while (condicion_de_parada)
8           SELECCION_PROGENITORES
9           CRUCE
10          MUTACION
11          COSTE_DESCENDENCIA
12          INTRODUCCION_NUEVA_GEN
13      end
14  end
15
```

Figura 2 Pseudocódigo del Algoritmo Genético Simple

- Desarrollo del algoritmo

La población inicial por lo general se forma dando un valor aleatorio a cada uno de los genes de los individuos de la población, aunque existen otros métodos, como por ejemplo aplicar técnicas de optimización para generar una población inicial con mejores individuos. De esta forma se conseguiría reducir el tiempo necesario para llegar a la solución, aunque también podría provocar que se converja alrededor de un óptimo local y no uno global (convergencia prematura).

Lo siguiente es calcular el coste de cada uno de los individuos de la población. Se conoce como coste al valor numérico asociado a cada elemento que se utiliza para poder comparar dos soluciones entre sí. Para obtener este valor, necesitamos definir una función objetivo que estudie cada uno de los individuos y que sea capaz de determinar con precisión su capacidad para resolver el problema planteado.

Cuando ya se tiene construida una población inicial y a cada elemento se le ha asignado su coste, se entra en el bucle principal del algoritmo. La condición de parada se definirá en función de nuestras necesidades, pero por lo general suele ser un número máximo de iteraciones a realizar o que el algoritmo haya convergido alrededor de una solución.

Dentro de este bucle tendrá lugar la creación de una nueva generación de individuos, descendientes de los anteriores. Para ello, lo primero que se debe hacer es seleccionar los individuos que actuarán como progenitores del nuevo elemento. Por lo general, los elementos con mejor coste serán los que tengan más posibilidades de reproducirse, aunque siempre se mantiene una cierta aleatoriedad para que exista una diversidad en el espacio de soluciones. Por lo general se pueden considerar tres métodos distintos de selección [22].

El primero es el llamado método ruleta, o método de selección proporcional. Según este, la probabilidad que tiene cada elemento de convertirse en progenitor de la siguiente generación es directamente proporcional a su coste. Después, la selección se produce de manera aleatoria, en función de estos valores.

El método de selección por ranking calcula la probabilidad de reproducirse que tiene una determinada solución a través de un valor numérico que indique su adaptación al medio, en vez de su coste. Esta adaptación puede ser calculada de diversas maneras, entre ellas destaca el ranking lineal. Consiste en ordenar la población en función de su coste y después asignarle un valor en función de su posición en el ranking, valor que variará de forma lineal entre los individuos. Esto es muy útil cuando se tiene un individuo que destaca por encima del resto con mucha diferencia o cuando los individuos presentan costes muy similares, evitando una convergencia prematura en el primer caso o una convergencia excesivamente lenta en el segundo caso.

Por último, el método de selección por torneo consiste en hacer “competir” entre ellos a varios individuos (por lo general dos). Se genera un número aleatoriamente entre 0 y 1 y se

compara con un valor umbral. Si está por debajo, se elige al elemento con mejor coste, y si está por encima, se elige al que tiene peor coste.

Una vez se tienen seleccionados los progenitores, se cruzan para generar un nuevo individuo, a base de coger parte de las características de uno de los progenitores, y otra parte del otro. A nivel práctico, lo que se hace es intercambiar parte de la información contenida en uno de los elementos a cruzar, con la contenida en el otro. Al igual que el de selección, el proceso de cruce se puede llevar a cabo de varias formas distintas [22], [23].

El primer método, conocido como cruce de un punto, consiste en seleccionar un punto en la cadena de genes que caracterizan a un elemento de la población e intercambiar todos los genes a la izquierda de ese punto. Existe una variación de este método que consiste en elegir varios puntos e intercambiar los genes situados a izquierda y derecha de cada punto. Otra forma de cruzar los dos progenitores consiste en escoger de forma aleatoria para cada gen a uno de los progenitores. Es lo que se conoce como cruce uniforme. También existen otros métodos específicos para un tipo de problema en concreto.

Cuando ya se ha generado la descendencia, se emplea un mecanismo de mutación para inducir ciertas variaciones en los genes de las posibles soluciones para obtener otras nuevas. Este proceso va ganando en importancia a medida que la población va convergiendo. Esta mutación se puede llevar a cabo de múltiples formas en función de los requisitos del problema. Lo más sencillo es mutar cada gen de forma aleatoria e independiente, mientras que formas más complejas tendrían en cuenta más variables y relaciones para aplicar restricciones a esta mutación.

Después de esto, los nuevos elementos se introducen en la nueva población. Para ello, se puede eliminar el peor individuo de la generación anterior cada vez que se quiere introducir un nuevo elemento, se pueden generar individuos hasta llegar a un valor prefijado y sustituirlos por los elementos que presenten peor adaptación en la generación anterior, o eliminar soluciones de forma aleatoria para hacer espacio a las nuevas (este último método presenta el inconveniente de que se podría estar eliminando una solución muy buena, aunque por otra parte también permite mantener una mayor diversidad de individuos en la población).

Cuando se ha completado la creación de una nueva generación, el proceso vuelve a empezar con la selección de nuevos progenitores, el cruce y la mutación. Esto continuará hasta que se cumpla la condición de parada.

- ### Ventajas e inconvenientes de los algoritmos genéticos

La ventaja principal de este tipo de algoritmo es que presenta una buena solución en un tiempo no excesivo y que puede ser aplicada a una gran multitud de campos, ya que no

precisa tener un conocimiento específico acerca del problema a resolver. Aunque, debido a esto, suelen ofrecer resultados más pobres frente a los que se pueden conseguir empleando técnicas específicas propias de la disciplina en el que se incluye el trabajo a realizar.

Otras ventajas de los algoritmos genéticos son las siguientes [24], [25]:

- Trabajan al mismo tiempo con una población de soluciones más amplia, lo que permite explorar al mismo tiempo diferentes regiones del espacio de soluciones, aumentando la probabilidad de alcanzar un óptimo.
- Debido también al trabajar de forma simultánea en varios puntos, son menos propensos a converger alrededor de un óptimo local. Esto hace que destaquen trabajando en problemas en los que la función objetivo presenta una gran cantidad de óptimos locales, es discontinua, introduce una gran cantidad de ruido o varía con el tiempo.

Algunas desventajas son que pueden presentar una convergencia alrededor de un punto que no resulte la solución óptima (una convergencia prematura). Esto se produce cuando entra en la población un elemento que representa un óptimo local demasiado pronto, lo que disminuye la diversidad de la población, al suponer una gran mejora con respecto a los otros individuos de la población. También es posible que tarde demasiado en obtenerse dicha convergencia, o que incluso no llegue a producirse, debido a una mala selección de los parámetros del algoritmo.

- ### Casos en los que conviene aplicarlos y consideraciones a tener en cuenta

Estos algoritmos permiten obtener una solución adecuada en aquellos problemas que se pueden expresar con un modelo matemático y cuyas posibles soluciones se encuentren dentro de un espacio acotado y puedan ser comparadas entre sí empleando una función matemática para obtener un valor que nos indique como de eficaz es esa solución. Es necesario definir esta función de forma muy precisa, ya que en caso de no ser así, la solución que se obtenga puede no ser la que se estaba buscando.

Además de elegir una función objetivo adecuada, es necesario seleccionar los parámetros del algoritmo con el mismo cuidado, como son el tamaño de población o la tasa de mutación.

Una población demasiado pequeña puede suponer que solo se busque la solución en una región demasiado pequeña del espacio de soluciones, lo que provocaría o bien que no se encontrase la solución óptima, o que se tardase demasiado en encontrarla. Por el contrario, un tamaño de población excesivamente grande trabajará con más posibles soluciones al mismo tiempo, aumentando las posibilidades de obtener el óptimo, pero al mismo tiempo aumenta el coste computacional, lo que vuelve a suponer un aumento

excesivo del tiempo. Elegir un buen tamaño de población permitirá obtener soluciones aceptables en un periodo no excesivo.

En cuanto a la agresividad del operador de mutación, también es necesario mantener un control exhaustivo, puesto que puede representar la diferencia entre permitir mutar a los individuos para mejorar la población, o que dejen de representar una buena solución. Si el mecanismo de mutación opera de forma excesivamente suave, las mutaciones inducidas en la población podrían ser demasiado pequeñas como para que representen una mejora respecto a la generación anterior. Si actúa de forma demasiado agresiva, podría convertir una solución que se encuentra cercana al óptimo, en un individuo que represente una mala solución al problema a representar.

Algunas aplicaciones de los algoritmos las podemos encontrar en la resolución de problemas de optimización, donde destacan especialmente, la programación automática, el aprendizaje máquina, la economía, los sistemas inmunes artificiales, la ecología, la genética de poblaciones, la evolución y el aprendizaje, o los sistemas sociales [23].

3.2. Evolución Diferencial. Bases teóricas

La Evolución Diferencial (de ahora en adelante, ED) es un algoritmo genético empleado para la resolución de problemas de optimización que se caracteriza por almacenar los nuevos individuos de la población en un vector de prueba y compararlos con los elementos ya presentes en la población para ver cuales deben pasar a pertenecer a la nueva población. Fue presentado por primera vez en un artículo firmado por Kenneth Price y Rainer Storn [26] en 1997.

Este método comienza generando una población inicial de tamaño NP con elementos extraídos del espacio de soluciones del problema a resolver. Dichos individuos se denominan en ED vectores objetivo. Cada uno de estos elementos se representa mediante un vector que contiene los valores de las distintas variables del problema, también llamadas genes. Para generar esta población inicial, se dan valores aleatorios contenidos entre los límites superior e inferior para cada una de las variables. Este proceso se puede observar en la ecuación 3, donde *g* es el número de la generación (0 o 1, dependiendo de cómo se quiera numerar a la generación inicial), *i* es el elemento de la población y *j* es la variable dentro de dicho elemento. Esta será la generación inicial.

$$x_{i,j}^{g} = x_{j_{min}} + (rand[0,1] * (x_{j_{max}} - x_{j_{min}}))$$

Ecuación 3. Generación de los elementos de la población inicial

Cuando ya se ha generado la población inicial, se calcula el coste asociado a cada uno de los elementos de la población, esto es, el valor asociado a cada individuo que permite

realizar una comparación entre ellos y conocer cual resulta una mejor solución al problema planteado.

Después se somete a un proceso de mutación y recombinación para dar lugar a los nuevos individuos que competirán con los ya existentes por un lugar en la siguiente generación.

El mecanismo de mutación más corriente empleado consiste en generar NP vectores, conocidos como vectores aleatorios ruidosos, a partir de tres vectores objetivo extraídos de la población, según se muestra en la ecuación 4, donde *v* es el vector aleatorio ruidoso, x_A, x_B, y x_C son los vectores objetivo y *F* es la tasa de mutación. Lo normal es que se escojan tres vectores al azar, pero existen variaciones de este mecanismo donde los vectores se generan a partir de los mejores individuos de la población. De esta forma, se reduce la diversidad en la población, ya que los nuevos elementos estarán formados a partir de los mejores de la generación anterior, pero se conseguirá aumentar las posibilidades de que el nuevo elemento mejore los presentes.

$$v_i^g = x_A^g + F * (x_B^g - x_C^g)$$

Ecuación 4. Mecanismo de mutación ED

La tasa de mutación, *F*, es el parámetro que regula la agresividad del mecanismo de mutación. Puede tomar un valor real contenido en el intervalo [0,1]. Es una de las claves para conseguir alcanzar la solución deseada, por lo que se debe tratar de ajustar adecuadamente, ya que en caso de que sea muy pequeño, el vector mutado será muy similar al vector x_A, mientras que si es muy grande puede provocar que el elemento mutado se convierta en una mala solución.

Una vez se han credo los vectores aleatorios ruidosos, se someten a un proceso de recombinación. A través de este proceso se construyen los vectores prueba que contendrán a los elementos que competirán con los individuos de la generación anterior. Cada uno de los genes del vector prueba se obtendrá eligiendo aleatoriamente entre los genes presentes en el vector mutado y en el elemento de la población que corresponde (el primer vector prueba se generará eligiendo entre el primer elemento de la población y el primer vector mutado, y así sucesivamente). Esto se puede observar en la ecuación 5, donde *g* es la generación con la que se trabaja, *i* es la posición del elemento en la población, *j* es el gen del elemento en cuestión y *CR* es la tasa de recombinación.

$$p_{i,j}^g \begin{cases} v_{i,j}^g & si\ rand[0,1] \leq CR \\ x_{i,j}^g & si\ rand[0,1] > CR \end{cases}$$

Ecuación 5. Mecanismo de recombinación ED

La tasa de recombinación es una variable que puede tomar un valor contenido en el intervalo [0,1]. Al igual que ocurre con la tasa de mutación, es necesario ajustar este

parámetro para poder lograr los resultados esperados. Una tasa de recombinación muy alta implica que por lo general, los vectores prueba van a estar compuestos mayoritariamente por los genes de los vectores obtenidos por medio de mutación, con lo que se podrían perder genes con un muy buen valor presentes en el elemento de la generación actual, mientras que si la tasa es muy baja, la mayoría de genes procederán del elemento original, con lo que el vector de prueba presentará un coste muy similar al del elemento original, ralentizando la obtención de una buena solución.

Cuando se ha generado un vector de prueba, se calcula su coste y se llev0a a cabo el proceso de selección, donde se compara este vector con el elemento de la población que ocupa la misma posición para ver cuál de los dos formará parte de la próxima generación. El proceso de selección se puede ver en la ecuación 6.

$$x_i^{g+1} \begin{cases} p_i^g & si\ cost\left(p_i^g\right) > cost(x_i^g) \\ x_i^g & si\ cost\left(p_i^g\right) \leq cost(x_i^g) \end{cases}$$

Ecuación 6. Proceso de selección ED

Una vez se ha obtenido una nueva generación de individuos, se vuelve a comenzar el proceso de mutación-recombinación-selección. Este ciclo continuará hasta que se cumplan las condiciones requeridas para finalizar, que por lo general suele ser haber alcanzado un número determinado de iteraciones, o que la población haya convergido alrededor de una solución.

- Diferentes métodos basados en ED [24], [25]

A partir del algoritmo básico, se han desarrollado diferentes modificaciones del método de ED. Aunque se pueden clasificar atendiendo a diferentes criterios, el criterio general atiende a la forma de fijar los distintos valores para los parámetros del algoritmo, pudiendo encontrar métodos adaptivos, que emplean diferentes técnicas para que el propio algoritmo pueda fijar el mejor valor para ellos, métodos que combinan técnicas adaptivas con otras técnicas, conocidos como métodos híbridos, o métodos no adaptivos. En la Figura 3 se puede observar una clasificación de los distintos métodos explicados a continuación.

$$ED \begin{cases} métodos\ adaptivos: \begin{cases} SADE \\ JADE \end{cases} \\ métodos\ híbridos: SFLSDE \\ métodos\ no\ adaptivos: \begin{cases} ED\ básico \\ OBDE \\ DEGL \end{cases} \end{cases}$$

Figura 3. Clasificación de los distintos métodos de ED

1. **Evolución Diferencial Basada en Oposición (OBDE)**: En esta versión, el algoritmo trabaja al mismo tiempo con dos soluciones opuestas en el espacio de soluciones. Por pura estadística, al generar una población inicial, si se elige un punto al azar y el punto opuesto, hay un 50% de posibilidades de que la solución que forma parte de la población esté más cerca del óptimo buscado. De esta forma, se elige entre cada solución y su opuesta la que presente un mejor coste, y esa será la que pase a formar parte de la población. En la ecuación 7 se ve el proceso para el opuesto de un punto, donde x_{min} y x_{max} son el valor mínimo y máximo que puede tomar la variable x.

$$x' = x_{max} + x_{min} - x$$

Ecuación 7. Ecuación del punto opuesto

2. **Evolución Diferencial con vecindades Globales y Locales (DEGL):** Utiliza un mecanismo de mutación que parte de los mejores individuos, en vez de formar los vectores mutados con elementos aleatorios. Además dicho vector se forma como combinación lineal de dos vectores, uno formado con el método habitual entre todas las soluciones, lo que vendría siendo la vecindad global, mientras que el otro vector se genera con los elementos pertenecientes a la zona con forma de anillo que rodea el individuo a mutar. Esta combinación lineal está controlada por una variable, perteneciente al intervalo [0,1], que determina si se da preferencia a la explotación de la zona que rodea al punto a mutar (si es cercana a 1) o a la exploración de nuevas zonas en el espacio de soluciones (si es cercana a 0). Este parámetro puede ser establecido por el usuario o fijado mediante el empleo de alguna técnica auto-adaptiva, técnica que permite al propio algoritmo fijar el valor óptimo del parámetro en base a una serie de pruebas que el mismo realiza.

3. **Evolución Diferencial Auto-Adaptiva (SADE):** Esta modificación del método básico propone permitir que sea el algoritmo el que controle los diferentes parámetros que controlan su correcto funcionamiento, además de los mecanismos de mutación y recombinación empleados para generar los elementos que buscan un hueco en la población. Lleva un recuento de los éxitos y fracasos que ha conseguido cada combinación de opciones y, en función de esto, elige cual es la mejor estrategia para la siguiente generación.

4. **Evolución Diferencial Adaptiva con Archivo Opcional Externo (JADE):** Varía el mecanismo de mutación añadiendo dos términos de diferencia entre vectores, en vez de uno. En el primer término se resta el elemento del que se parte al mejor individuo de la actual generación, mientras que en el segundo término se elige un elemento de la población, al que se le resta otro individuo, que puede salir de la población o de una memoria en la que se almacenan aquellos individuos que se

descartaron en la fase de selección. Este mecanismo de mutación se puede observar en la ecuación 8. En esta ecuación, x_A^g es el elemento a mutar, x_{best}^g es el mejor elemento de la población, x_B^g es uno de los elementos de la población, x_C^g es el elemento que se puede elegir entre la población y los individuos no seleccionados y v_i^g es el vector mutado. Además, este método aplica los mecanismos de auto-adaptación presentes en el método SADE.

$$v_i^g = x_A^g + F * \left(x_{best}^g - x_A^g\right) + F * \left(x_B^g - x_C^g\right)$$

Ecuación 8. Mecanismo de mutación DE/current-to-best/1

5. **Evolución Diferencial con Búsqueda Local del Factor de Escala (SFLSDE):** Se puede definir como un algoritmo memético. Un algoritmo memético combina varias ideas tomadas de diferentes técnicas metaheurísticas, tanto basadas en población (mantienen una población de posibles soluciones) como basadas en búsqueda local, lo que constituye la principal característica de estos algoritmos. Se centra en la adaptación de varios parámetros durante la etapa de mutación, en especial la tasa de mutación, F, empleando para ello dos algoritmos de búsqueda local.

3.3. Implementación del algoritmo en el RELF-3D

Para el RELF-3D se ha desarrollado un algoritmo basado en la versión estándar del método de ED [4]. La población inicial con la que se comienza a trabajar ha sido generada en un archivo externo al algoritmo y pasada como parámetro. En la Figura 4 se puede observar el pseudocódigo para este algoritmo.

Algorithm 1 alg_genet_3d

```
1:  function alg_genet_3D(...)
2:      for i = 1 : N_P do
3:          estimated_dist_3d(i) ← dist_est_3d(...)
4:          cost(i) ← fitness_3d(estimated_dist_3d(i), real_dist_3d)
5:      end for
6:      while (CONVERGENCE CONDITIONS) do
7:          for i = 1 : N_P do
8:              MUTATION
9:              CROSSOVER
10:             SELECTIONwithTHRESHOLDING
11:             estimated_dist_3d(i) ← dist_est_3d(...)
12:             cost(i) ← fitness_3d(estimated_dist_3d(i), real_dist_3d(i))
13:                     ▷ cost function value calculation for next generation
14:         end for
15:         DISCARDING
16:         [error, ind_best] ← min(cost)
17:         bestmem ← pop(ind_best)
18:         conv_conditions_checking(...)
19:     end while
20: end function          ▷ return bestmem, error and population
```

Figura 4. Pseudocódigo para el algoritmo empleado en el RELF-3D [7]

En el problema de la localización lo que se pretende es encontrar la posición y la orientación del robot, por lo que en el primero de los genes de cada individuo se almacena su coste, los tres siguientes representan sus coordenadas en cada uno de los tres ejes, y el último representa su posición.

El algoritmo comienza por calcular el coste asociado a cada uno de los elementos de dicha población. Para esto, primero es necesario obtener las medidas del sensor laser simulado asociadas a la posición y orientación del individuo en cuestión, empleando para ello una función específica. Esta función transforma los datos del sensor en incrementos para cada uno de los ejes utilizando la orientación del elemento con el que se trabaja, para cada una de las distintas medidas que realiza el láser. Una vez se tienen estos incrementos, se va modificando la posición del elemento hasta que se encuentra un obstáculo. En ese momento se calcula la distancia que existe entre ese obstáculo y el punto inicial, y se almacena en un vector llamado dist_3d.

Después, en función de la versión del algoritmo con la que se trabaje, se emplea una de las dos funciones objetivo desarrolladas. Ambas funciones comparan las medidas obtenidas con el sensor laser con las medidas asociadas al elemento seleccionado.

La primera consiste en el cálculo de la suma de los errores al cuadrado, función conocida como L2-Norm. En la ecuación 9 se puede ver esta función. En dicha ecuación, z_t es el conjunto de medidas que el sensor del robot toma en el instante t, ${\hat{z}_t}^j$ es el conjunto de medidas estimadas por el robot para el elemento con el que se trabaja, ${x_t}^j$ es el elemento de la población con el que se está trabajando, y ${\sigma_e}^2$ es la varianza del error de medida.

$$L2\left({x_t}^j - \hat{x}_t\right) = \sum_{i=0}^{Ns} \frac{(z_{t,i} - {\hat{z}_{t,i}}^j)^2}{2{\sigma_e}^2}$$

Ecuación 9. Calculo de la función L2-Norm

Esta es la función empleada en la mayor parte de los casos, aunque presenta algunas desventajas. Debido al número, precisión y rango de los sensores empleados en la toma de medidas, se limita la posibilidad de distinguir entre varias posiciones, lo que provoca que el algoritmo pueda converger a varios máximos globales. Por otra parte, debido al mapa empleado en el estudio, en el cual se encuentran varias habitaciones con geometrías similares, provoca que existan varias soluciones válidas para la función L2-Norm. Además, debido a que esta función no trabaja con todos los datos que se conocen, el resultado obtenido es bastante inestable, y el filtro puede moverse entre varios mínimos locales de forma brusca. Por último, esta función no proporciona unos resultados tan buenos si el modelo que representa el ruido en las medidas no se conoce, o está mezclado con otras distribuciones.

En ciertas situaciones, es preferible obtener el coste a partir de la función L1-Norm, que calcula el error absoluto, aunque si esta función no fuese derivable, sería necesario aplicar

programación lineal para obtener una solución. Se recomienda su uso cuando los errores presentan una distribución de Laplace, Cauchy o normal contaminadas. La L1-Norm se puede observar en la ecuación 10.

$$L1\left({x_t}^j - \hat{x}_t\right) = \sum_{i=0}^{Ns} \frac{\left|z_{t,i} - {\hat{z}_{t,i}}^j\right|}{\sigma_e}$$

Ecuación 10. Función objetivo L1-Norm

Una vez se ha calculado el coste de cada individuo, el cual se almacena en el primer gen de cada individuo de la población, se entra en el bucle principal del algoritmo. Para cada uno de los elementos se lleva a cabo un proceso de mutación, cruce y selección.

Se comienza por seleccionar de forma aleatoria a tres de los individuos de la población, de tal forma que no se elija el mismo elemento más de una vez, y que ninguno de los tres coincida con el elemento sobre el que se está trabajando.

Después se genera un vector prueba. Este puede estar formado o bien por el individuo correspondiente o generado a partir de los tres elementos elegidos anteriormente. Para ver cuál de los dos casos se da, se recurre al mecanismo de cruce. Se genera una variable aleatoria y se compara con el valor de la constante de recombinación, o CR. Si este valor aleatorio (que debe estar dentro del rango de valores de CR) es menor que la constante de recombinación, se pasa a crear el vector prueba a través del mecanismo de mutación. Si es mayor, el vector prueba coincidirá con el elemento original.

En el RELF-3D se han implementado dos versiones distintas del mecanismo de mutación. La primera consiste en el mecanismo básico del método ED, conocido como DE/rand/1, en el que el vector prueba se genera a partir de la ecuación 3, vista en la sección 3.2, utilizando los tres individuos de forma aleatoria. La segunda versión emplea la misma ecuación, solo que en este caso, se busca cuál de los tres elementos es el mejor y a ese se le añade la diferencia de los otros dos, multiplicada por la tasa de mutación.

Cuando se ha generado el vector prueba, y se ha comprobado que este entra dentro de los límites del mapa, se calcula su coste y se compara con el del elemento que se pretende sustituir. De entre los dos, el que presente menor coste pasará a formar parte de la nueva generación. En el mecanismo de sustitución se permite la opción de utilizar una banda de rechazo del 2%. Esta banda provoca que se descarten no solo los vectores prueba que sean peores que el elemento con el que se compara, si no que se descartarán también todos aquellos que no mejoren a dicho elemento al menos en un 2% de su coste.

Por último se presenta también la opción de utilizar un mecanismo de descarte, que busca aumentar la velocidad del algoritmo. Para ello, se descarta un número aleatorio de las peores soluciones existentes y se sustituyen por elementos aleatorios pertenecientes a la mitad de la población donde se almacenan los mejores individuos. Este mecanismo es más

importante en las primeras generaciones, mientras que a medida que la población converge alrededor de la solución pierde importancia.

Este ciclo se repite hasta que se cumple alguna de las condiciones de convergencia que se han definido para este algoritmo. En este caso se tienen en cuenta dos casos: que se alcance el límite superior de iteraciones, o que se alcance la convergencia total de la población.

4. CLONALG: ALGORITMO DE SELECCIÓN CLONAL

El Algoritmo de Selección Clonal, conocido comúnmente como CLONALG, es un tipo de algoritmo genético que se encuadra dentro de la línea de investigación conocida como Sistemas Inmunes Artificiales.

4.1. Introducción a los Sistemas Inmunes Artificiales

Los sistemas inmunes artificiales son sistemas inteligentes que se basan en los principios que rigen el funcionamiento del sistema inmunológico de los animales vertebrados, utilizando las habilidades de aprendizaje y de memoria inmune para alcanzar la solución del problema que se intenta resolver.

La investigación acerca de estos sistemas comenzó en los años 80 con el trabajo de Farmer, Packard, Perelson [27]. En él se muestra el modelo creado a partir de la Teoría de la Selección Clonal y se compara con la técnica de aprendizaje automático desarrollada por Holland [28]. Posteriormente, se abrieron varias líneas de investigación dentro de este campo, entre las que destacan las siguientes cuatro: Clonal Selection Algorithms, Negative Selection Algorithms, Inmune Network Algorithms y Dendritic Cells Algorithms

La familia de algoritmos conocida como Clonal Selection Algorithms adapta la Teoría de la Selección Clonal para la resolución de problemas de optimización y de reconocimiento de patrones, centrándose especialmente en los aspectos de la teoría que comparten características comunes con la Teoría de la Evolución, propuesta por Charles Darwin. Al igual que en la evolución, los mejores individuos son seleccionados y se reproducen a través del proceso de clonación. La mutación de los clones amplía la diversidad de la población de individuos. Tres algoritmos destacan dentro de esta categoría:

1. **Clonal Selection Algorithm (CLONALG):** Propuesto por Leandro N. de Castro y Fernando J. Von Zuben [5], dio comienzo a este campo de investigación, aplicando la Teoría de la Selección Clonal para la resolución de problemas de reconocimiento de patrones y de optimización. En la sección 4.2 se hablará más en detalle de este algoritmo.

2. **Artificial Inmune Recognition System (AIRS):** Desarrollado por A. Watkins, J. Timmis y L. Boggess [29], consiste en un sistema de aprendizaje automático supervisado (el algoritmo recibe una serie de datos en los que cada entrada se presenta junto a la salida deseada, y genera una función que permite obtener el resultado buscado a partir de su correspondiente entrada). Imita la capacidad de los anticuerpos de reconocer a un determinado antígeno y unirse a él para obtener la relación entre los datos de entrada y salida del sistema y poder generar la función correspondiente. Los elementos de la población son expuestos a una serie de datos de entrenamiento. Los individuos que reciban un mayor

estimulo serán seleccionados para ser clonados y mutaos, y posteriormente estos clones competirán entre sí por un puesto en la población.

3. **B-Cell Algorithm (BCA):** En este algoritmo, la población está formada por vectores de 64 bits que almacenan números con doble precisión codificados en binario. Cada uno de estos vectores representa uno de los linfocitos B del sistema inmune. De cada uno de estos vectores se saca un número de clones que suele corresponderse con el tamaño de la población (aunque no siempre tiene que ser el mismo). Estos clones se mantienen en memorias independientes para cada grupo creado a partir del mismo elemento. Después se elige un clon al azar y se aplican cambios aleatorios a cada uno de los elementos contenidos en ese vector, para generar un nuevo vector y mantener la diversidad. Una vez hecho esto, se les aplica a cada una de las células B el mecanismo de hipermutación somática contigua, una característica distintiva de este algoritmo. Por último se compara cada clon mutado con el elemento original para ver si debe sustituir a este en la población. Se puede obtener más información sobre este método en el trabajo de J. Kelsey y de J. Timmis [30].

En el estudio del sistema inmune se conoce como selección negativa al mecanismo que permite eliminar a aquellos linfocitos que, debido a su estructura genética, se detecta a sí mismo, o a los tejidos del propio organismo, como un antígeno. Esta eliminación se produce por lo general durante la etapa de maduración de dicho linfocito.

El algoritmo basado en este mecanismo fue desarrollado en principio por S. Forrest, A. Perelson, L. Allen y R. Cherukuri [31] en 1994 como un método para detectar manipulaciones de datos ocasionadas por virus informáticos. Este algoritmo comienza por generar una serie de cadenas de datos propios codificados en binario que definen el estado normal del sistema. Después genera una serie de detectores preparados para reconocer datos totalmente diferentes de estas cadenas de datos. De esta forma, al presentar nuevos datos a alguno de los detectores, este puede clasificarlos como propios del sistema o extraños (siempre que alguno de los detectores se active se sabrá que se ha producido una variación en alguno de los datos del sistema).

Si se exigiese que el dato en cuestión coincidiese en su totalidad con alguno de los detectores, se necesitaría generar una gran cantidad de detectores para asegurar que alguno de ellos fuese capaz de coincidir con el dato en cuestión. Además si los datos presentasen una longitud excesiva, complicaría la obtención de detectores que no coincidan en ningún punto con dicho dato. Estas dos cuestiones provocarían un aumento de la potencia de cálculo necesaria para ejecutar el algoritmo.

Para solucionar esto, se va a considerar que el detector coincide con una cadena de datos en concreto cuando exista una coincidencia de los datos almacenados en un determinado número de posiciones contiguas.

Otra clase de sistemas inmunes artificiales son los algoritmos basados en la Teoría de la Red Inmune, propuesta por primera vez por el inmunólogo Niels Jerne [32]. Esta teoría presenta al sistema inmune como una red de linfocitos que interactúan entre sí, y de moléculas con regiones variables que les permite enlazarse no solo con elementos ajenos al propio organismo, sino con otras regiones para formar una red. Debido a esto, en ausencia de estímulos externos, la actividad llevada a cabo por el sistema inmune consistiría en las relaciones entre las distintas moléculas, lo que activaría el proceso de selección clonal, al igual que ocurriría ante la presencia de un patógeno.

El funcionamiento básico de este algoritmo es similar al del CLONALG. Se comienza por generar una serie de clones a partir de los mejores individuos de una población generada de forma aleatoria. El número de clones a obtener será directamente proporcional a la afinidad del elemento a clonar. Estos clones serán sometidos a un proceso de hipermutación inversamente proporcional a su afinidad. A partir de aquí es donde este algoritmo difiere del CLONALG. Una vez mutados los clones, se eligen los mejores para mantener almacenados en una memoria. De estos elementos se eliminarán aquellos cuya afinidad con el antígeno a detectar y con los otros clones de la memoria no supere un valor determinado. Los clones supervivientes se introducen de nuevo en la población. Posteriormente se comprueba la afinidad existente entre cada pareja posible de anticuerpos en la población y se eliminan aquellos cuya afinidad no alcance el valor umbral. Por último se generan nuevos anticuerpos de forma aleatoria y se introducen en la población.

Dentro de los algoritmos que aplican esta teoría podemos encontrar el aiNet, desarrollado por de Castro y Von Zuben [33]. Posteriormente, de Castro y Timmis [34] presentaron una modificación de este método llamada Opt-aiNet que se caracteriza por determinar de forma automática el tamaño de la población, por combinar una búsqueda local y global, por unos criterios de parada mejor definidos y por poder encontrar y mantener óptimos locales estables.

La última línea de investigación que se puede encontrar dentro de los Sistemas Inmunes Artificiales es la conocida como Dendritic Cell Algorithms. Uno de los primeros trabajos en este campo fue el de J. Greensmith, U. Aickelin y S. Cayzer [35]. El principal campo de aplicación de este algoritmo es la detección de anomalías.

Este tipo de algoritmos están basados en la conocida como Teoría del Peligro, según la cual, el sistema inmune no reconoce las diferentes células como propias del organismo o externas, sino que distingue como seguro o perjudicial, en función de una serie de señales que los tejidos dañados generan.

Este algoritmo comienza creando una población de células dendríticas, que se considerarán como células sin madurar. Cada una de estas células almacenará una serie de antígenos provenientes de una fuente externa. Después comprobará las señales presentes en la zona de donde proviene el antígeno, distinguiendo si son señales de peligro, de que

todo está bien, de inflamación o de presencia de moléculas asociadas a un determinado grupo de patógenos. En función de estas señales de entrada, calcula tres señales de salida distintas. La primera indica que esa célula ha pasado al estado maduro, la segunda indica que ha pasado al estado semi-maduro y la tercera es el nivel de moléculas que han sido estimuladas dentro de la población por la célula sobre la que se está trabajando. Si este nivel supera un cierto valor umbral, esta célula se traslada a una nueva población, y se clasifica en función de las dos primeras señales. Si el nivel de la señal de salida que indica que la célula está madura es mayor que el de la señal que indica que está semi-madura, esa célula pasará a considerarse como célula madura, si no, pasará a considerarse como célula semi-madura.

Cuando se ha realizado este proceso para todas las células dendríticas de la población, se pasa a presentarles a las células maduras y semi-maduras los distintos datos que se quieren clasificar como seguros o dañinos. Si un determinado dato se presenta a un mayor número de células semi-maduras, se considerará como seguro, si se presenta a un mayor número de células maduras, se considerará como dañino.

4.2. CLONALG: Algoritmo de Selección Clonal

El CLONALG fue desarrollado por Leandro N. de Castro y Fernando J. Von Zuben [5] en un artículo publicado en junio de 2003 para ofrecer inicialmente un método para el reconocimiento de patrones y aprendizaje automático, y posteriormente fue adaptado para ofrecer una solución a problemas de optimización. Este algoritmo recibe su nombre de la Teoría de la Selección Clonal, teoría que rige el funcionamiento del sistema inmunológico humano. Esta teoría explica cómo se produce y controla la respuesta inmune a un estímulo provocado por un antígeno determinado.

- Teoría de la Selección Clonal

Esta teoría fue postulada por los inmunólogos Niels Jerne, Frank Burnett y David Talmage. Jerne presentó en 1954 la teoría de que los linfocitos se encuentran presentes en el cuerpo anteriormente a la aparición de un determinado antígeno. Esta aparición lo que provoca es la selección de aquel linfocito capaz de producir el anticuerpo asociado a ese antígeno para ser clonado de forma que se pueda hacer frente a la infección. Posteriormente, en 1957, Burnett y Talmage completaron esta teoría proponiendo el concepto de memoria inmunológica, según el cual, se observa una distinción de dos tipos de linfocitos. El primero es clonado de forma masiva para hacer frente a la infección inmediatamente. El otro se almacena para poder ser clonado en caso de que el mismo antígeno sea detectado.

Un linfocito es una célula linfática generada en la médula ósea y que posteriormente se desplaza a otros órganos linfoides. Responden ante el estímulo generado por un elemento

extraño. Se dividen en dos tipos, linfocitos B (que se encargan de la generación de anticuerpos) y linfocitos T, que se encargan de la detección de antígenos asociados a moléculas asociadas a su vez a un grupo de genes ubicados en el brazo corto del cromosoma 6.

Los linfocitos B se generan en la médula ósea a partir de células indiferenciadas. Para ello se realizan todas las posibles combinaciones de genes, obteniendo así un linfocito inespecífico, capaz de reconocer a cualquier molécula, incluso las pertenecientes al propio organismo, por lo que es necesario descartar las que cumplan esta última característica.

Cuando el sistema inmune responde a un determinado antígeno, los linfocitos que presentan una mayor afinidad con este se le unen, provocando la generación de células plasmáticas capaces de secretar diferentes anticuerpos. Además de esto, también se generan una serie de linfocitos B del mismo tipo para mantener en la memoria inmune. De esta forma, si se produce una nueva exposición al mismo antígeno, estos linfocitos pueden generar anticuerpos con una mayor afinidad frente a este antígeno. Dicha afinidad seguirá aumentando con cada infección posterior. Además, los linfocitos adaptados para responder ante cierto tipo de antígeno no solo aumentarán su afinidad con respecto a este, sino también con todos los que presenten una estructura semejante.

Si se compara la primera respuesta al antígeno con la respuesta ante un segundo encuentro, se puede observar que esta última presenta un menor tiempo de respuesta, una mayor tasa de replicación de las células plasmáticas, y la presencia de anticuerpos de mayor afinidad. Además se necesita una menor cantidad de antígenos para provocar esta segunda respuesta.

Para mantener una diversidad en la población, se emplean dos mecanismos. Por una parte, tras llevar a cabo la clonación de los linfocitos más adaptados, se les somete a un procedimiento de hipermutación. Esto es necesario para conseguir un rápido crecimiento de los clones, pero sin embargo si se le provocan mutaciones excesivas a un mismo elemento se podrían perder las características que convierten a ese linfocito en un candidato para mantener en el sistema. Por ello, el proceso de hipermutación se lleva a cabo de forma inversamente proporcional a la afinidad del anticuerpo en cuestión. De esta forma, cuando presentan una afinidad muy baja, las mutaciones son más severas para tratar de que esta afinidad aumente, mientras que para células con afinidad más alta, las mutaciones se reducen para evitar que el linfocito pierda su utilidad.

De manera paralela al proceso de hipermutación, se produce también lo que se conoce como selector de receptores. De esta forma, se eliminan aquellos receptores del linfocito B que presentan una baja afinidad y se sustituyen por nuevos receptores. Así se evita tener que eliminar la célula a completo.

Para mantener la diversidad en la población, se elimina una cierta cantidad de los peores elementos presentes, y se sustituyen por nuevas células aleatorias.

- Bases teóricas del CLONALG

La versión inicial del CLONALG se desarrolló para el reconocimiento de patrones. En estos problemas, se puede comprobar la existencia de dos distintas poblaciones. La primera es una población de antígenos, que consiste en los elementos que se pretenden reconocer. La segunda es la población de anticuerpos, los elementos con los que se va a trabajar para resolver el problema en cuestión.

La población de anticuerpos se puede dividir en varios subgrupos. Por una parte están los elementos a mantener a lo largo de las generaciones, debido a la afinidad que presentan con respecto a los anticuerpos a reconocer. Entre estos individuos podemos agrupar a los mejores individuos, que son los primeros candidatos a ser clonados. También están los elementos no seleccionados para la clonación, los peores de los cuales serán sustituidos por nuevos anticuerpos generados de forma aleatoria. La cantidad de anticuerpos a sustituir puede variar desde unos pocos elementos hasta llegar a sustituir a todos los individuos que no se clonen.

Se comienza por seleccionar uno de los antígenos y se le pone en contacto con todos los anticuerpos que forman la población. De esta forma se obtiene para cada elemento la afinidad que presenta frente a ese antígeno y se almacena ese valor en un vector.

De entre todos los elementos se seleccionan individuos que presentan una mayor afinidad ante ese antígeno en concreto. Estos elementos son mutados de forma proporcional a su afinidad, de tal manera que del mejor individuo se obtiene la mayor cantidad de clones, mientras que del peor de los seleccionados se sacarán solo dos o tres clones. El conjunto de los clones formarán una nueva población, que será con la que se va a trabajar en las siguientes etapas. En la ecuación 11 se puede observar la manera de calcular el número de clones a sacar de cada individuo. En esa ecuación, n es el número de elementos a clonar, β es un factor de escala que permite variar la cantidad de clones a realizar y NP es el total de la población.

$$Nc = \sum_{i=1}^{n} round\left(\frac{\beta * NP}{i}\right)$$

Ecuación 11. Calculo del número de clones a obtener de cada individuo

El factor de escala, β, controla el número de clones a obtener a partir de cada uno de los elementos seleccionados. Para las primeras generaciones interesa que tenga un valor bajo (0.1 es un valor bastante común). Así se consigue mantener una diversidad en la población cuando aún no se está cerca del óptimo. Según los individuos se vayan acercando a la

solución deseada, puede irse aumentando este valor, para favorecer una búsqueda local en el entorno del mejor individuo. Si el valor de β es muy alto desde el inicio, podría producirse una convergencia prematura de la población (sin contar los elementos nuevos que se introducen en cada generación), mientras que si se mantiene bajo durante demasiadas generaciones se puede retrasar la obtención del óptimo, al no generar suficientes clones del elemento que se encuentra más cerca de la solución buscada.

Esta nueva población se somete a un proceso de hipermutación. Cada uno de los elementos se modifica de manera inversamente proporcional a su afinidad. Esto se realiza para evitar que un anticuerpo que presenta una buena afinidad pueda pasar a convertirse en una mala solución. De la misma forma, los individuos que presentan una baja afinidad mutan de forma más agresiva, para tener más posibilidades de obtener un anticuerpo mutado con una afinidad alta. Se puede observar el proceso para el cálculo de la tasa de mutación para cada uno de los elementos en la ecuación 12, donde ρ es un factor de escala y f es la afinidad del individuo en cuestión.

$$tasa\ mutación(i) = \exp(-\rho * f)$$

Ecuación 12. Cálculo de la tasa de mutación para cada elemento

La tasa de decaimiento, ρ, controla la variación de la tasa de mutación según la población se va acercando a una solución. Si se le da un valor muy bajo, provocaría que la tasa de mutación fuese muy alta para todos los elementos, lo que haría que mutasen en exceso y, para elementos con una afinidad buena, podría convertirlos en malas soluciones. Por otra parte, si el valor de ρ es demasiado alto, los valores obtenidos para la tasa de mutación serían muy bajos, por lo que al principio las mutaciones serían muy leves, con lo que la mejora de los individuos mutados sería pequeña, y a medida que la población se fuese acercando a la solución, las mutaciones podrían llegar a desaparecer. Lo mejor sería buscar una tasa de mutación alta al inicio, y que fuese disminuyendo a medida que se va llegando a la solución buscada, sin llegar nunca a disminuir por debajo de un límite.

Una vez se ha llevado a cabo el proceso para la totalidad de la población de clones y se ha recalculado la afinidad de cada uno de los elementos mutados, se selecciona al mejor de ellos y se compara con el correspondiente elemento presente en la memoria. El que presente la mejor afinidad de los dos, formará parte del nuevo grupo de anticuerpos, mientras que el otro será descartado.

Por último, se eliminan los peores elementos de la población y se sustituyen por elementos nuevos que se generan de forma aleatoria. Con esto se completa una generación. El proceso se seguirá repitiendo hasta que se satisfaga la condición de parada, que por lo general suele ser alcanzar un determinado número de generaciones.

Posteriormente se adaptó la estrategia descrita más arriba, para que el CLONALG pudiese hacer frente a problemas de optimización. Para ello hay que realizar una serie de modificaciones en los pasos a seguir.

En primer lugar, en la versión del CLONALG para optimización no existe ningún antígeno a reconocer, sino que el objetivo es obtener el valor óptimo que se puede extraer de una determinada función objetivo. De esta forma, cada anticuerpo representa una posible solución perteneciente al espacio de soluciones del problema a resolver. En segundo lugar, cuando se lleva a cabo el proceso de hipermutación para el conjunto de la población de clones, se eligen los n mejores elementos mutados para comparar con los individuos que se seleccionaron en un principio para ser clonados.

4.3. Desarrollo de la solución e implementación del algoritmo

En este proyecto se ha trabajado sobre la versión básica del CLONALG. A lo largo del proceso de programación se han realizado varias versiones del algoritmo. Se comenzó desarrollando por partes el código a ser implementado en MATLAB, buscando la forma de llevar a la práctica los pasos señalados por de Castro y Von Zuben [5].

En la primera versión del algoritmo, el usuario indicaba el número n de elementos que se seleccionaban para ser clonados, el factor de escala β para calcular el número de clones a obtener a partir de cada individuo y el número d de elementos nuevos a introducir en cada generación, además del tamaño de población, el límite superior de iteraciones a realizar y el error a introducir en las medidas.

Una función externa al propio algoritmo se encargaba de generar de forma aleatoria una población de posibles soluciones factibles al problema de la localización de un robot en un entorno conocido. Estas soluciones estaban formadas por el coste del elemento en cuestión, las coordenadas respecto a los tres ejes y la orientación respecto al eje z (considerando que el robot se mueve tan solo en los ejes x e y). En la variable D se almacenará el número de genes que componen la solución.

Para cada uno de los elementos de la población se calculaba su coste utilizando para ello la función objetivo L2-Norm, descrita en la sección 3.3 de este trabajo. Este coste era almacenado en la primera posición del vector que representaba a dicho individuo. Una vez hecho esto, se ordenaba la población en función de su coste, de forma que los elementos con menor coste ocupaban las primeras posiciones.

Hecho esto, se pasaba a calcular la cantidad de clones a obtener para cada uno de los n primeros individuos de la población. Esto se realizaba en tres pasos. Primero se aplicaba la ecuación 11 al primer elemento de la población. Después se almacenaba este valor en la posición correspondiente del vector num_clones. Por último se incrementaba la variable

Nc sumándole el valor obtenido en el primer paso. Este proceso se repite para los n elementos.

Una vez terminado, Nc almacena el tamaño total que tendrá la población de clones, mientras que en cada posición del vector num_clones se guarda el número de clones a obtener del individuo correspondiente. Este vector tiene un tamaño $n + 1$ y en la posición 1 se almacena un 0. Esto será necesario durante el proceso de clonación.

Antes de empezar con esto, se crean dos matrices distintas, la primera para almacenar la población de clones antes del proceso de mutación (Cj), y la segunda para almacenar los clones mutados (Cj_mut). A partir de aquí se entra en el bucle principal del programa. Para este algoritmo se ha seleccionado como criterio de parada un límite de generaciones.

Lo primero a realizar para cada nueva generación es la clonación de los mejores individuos. Para esto se emplean dos bucles anidados. El bucle exterior variará entre 1 y n, y permitirá elegir el miembro de la población a clonar, además de calcular el punto de la matriz Cj donde se almacenarán los clones de ese elemento. El bucle interior variará entre 1 y el número de clones a obtener a partir del individuo seleccionado y, para cada uno de esos valores, copiará el elemento de la población en esa posición de Cj.

El siguiente paso es calcular la tasa de mutación para cada uno de los clones. A aquellos que hayan sido clonados a partir del mismo elemento se les aplicará la misma tasa, que será directamente proporcional al coste del individuo en cuestión (cuanto más bajo sea el coste, mejor será esa solución, por lo que las mutaciones que se le provoquen deben ser menores).

Para el cálculo de la tasa de mutación se aplica la ecuación 12, pero con una pequeña modificación. La exponencial se calculará para $\frac{\rho}{f}$, en vez de para $\rho * f$, donde ρ es la constante que controla el decaimiento de la tasa de mutación y f es el coste. Esto se debe a que esta aplicación busca la minimización del coste, por lo que las tasas más bajas deben estar asociadas a los costes más bajos. A la constante ρ se le ha dado un valor de 690, dicho valor se calculó experimentalmente para que la tasa de mutación variase entre un valor cercano a 1 para los puntos con costes muy altos, y alrededor de 0,01 para elementos con un coste muy bajo.

La siguiente etapa es el paso crítico en este proceso, el mecanismo de hipermutación. Para este proyecto se han probado seis operadores distintos para llevar a cabo la mutación de los clones. Los cuatro primeros son bastante simples y parten de la misma base, mientras que los otros dos aplican distribuciones normales para el cálculo de la mutación a la que se debe someter a cada elemento.

El primer operador desarrollado es el más simple de los seis. A cada una de las variables de decisión del problema se le suma o resta un valor aleatorio en el rango de la variable en cuestión, multiplicado por la tasa de mutación correspondiente. Se muestra en la ecuación

13, donde Cj_{mut} es el clon mutado, Cj es el clon antes de ser sometido al proceso, i indica el elemento sobre el que se está trabajando ($i \in [1, Nc]$), j indica el gen del individuo sobre el que se está trabajando ($j \in [2, D+1]$) y b es un valor aleatorio contenido en el rango sobre el que se mueve la variable a mutar.

$$Cj_{mut}(i,j) = Cj(i,j) \pm tasa_{mut}(i) * b$$

Ecuación 13. Operador de mutación básico

Los tres siguientes son variaciones de este. El primero de ellos consiste en sumarle o restarle al gen a mutar el producto de la tasa de mutación por un valor aleatorio en el rango de la variable que se está mutando y dividirlo por la generación actual. De esta forma a medida que se van creando nuevas generaciones, las mutaciones a las los individuos están sometidas son cada vez menores. Este operador se muestra en la ecuación 14, donde $iter\_actual$ indica la generación sobre la que se está trabajando.

$$Cj_{mut}(i,j) = Cj(i,j) \pm \frac{tasa_{mut}(i) * b}{iter\_actual}$$

Ecuación 14. Primera variación del operador de mútación básico

El siguiente mecanismo completa al anterior multiplicando el numerador por el tamaño de la población de clones. De esta forma la agresividad de la mutación estará directamente relacionada con el número de clones a obtener en total. En la ecuación 15 se puede ver este mecanismo.

$$Cj_{mut}(i,j) = Cj(i,j) \pm \frac{tasa_{mut}(i) * b * Nc}{iter\_actual}$$

Ecuación 15. Segunda variación del operador de mutación

El último de los operadores de este grupo consiste en una variación del mecanismo anterior. El tamaño de la población de clones se le multiplica al denominador, en vez de al numerador, por lo que la agresividad de la mutación que sufrirá cada individuo es inversamente proporcional al tamaño de la población de clones, en vez de ser directamente proporcional. Se muestra este mecanismo en la ecuación 16. Tanto este método como el anterior han sido desarrollados a partir del trabajo de J. C. Herrera [36].

$$Cj_{mut}(i,j) = Cj(i,j) \pm \frac{tasa_{mut}(i) * b}{iter\_actual * Nc}$$

Ecuación 16. Tercera variación del operador de mutación

Los otros dos mecanismos que se han probado emplean distribuciones probabilísticas para generar las mutaciones en cada uno de los elementos. El primero de ellos se puede ver

en la ecuación 17. Se cambia el término que se le suma o resta al elemento original, por el producto entre la tasa de mutación y el valor de la distribución gaussiana con media 0 y varianza 1. Este mecanismo fue presentado en el libro escrito por K. M. Woldemariam [37].

$$Cj_{mut}(i,j) = Cj(i,j) \pm tasa_{mut}(i) * N(0,1)$$

Ecuación 17. Primer mecanismo de mutación basado en la distribución gaussiana

El otro operador fue presentado por S. Chittineni, A. N. S. Pradeep, D. Godavarthi, S. C. Satapathy, S. M. Krishna y P. V. G. D. P. Reddy [38] y está basado también en la distribución gaussiana.

$$\tau_1 = \left(\sqrt{2 * \sqrt{D}}\right)^{-1}, \quad \tau_2 = \left(\sqrt{2 * D}\right)^{-1},$$

Ecuación 18. Cálculo del paso total y del paso individual de mutación

$$\theta(i,j) = \theta(i,j) * e^{\left(\tau_1 * N(0,1) + \tau_2 * N_j(0,1)\right)}$$

Ecuación 19. Calculo del paso de mutación para el gen j del elemento i

$$Cj_{mut}(i,j) = Cj(i,j) + \theta(i,j) * N_j(0,1)$$

Ecuación 20. Segundo mecanismo de mutación basado en la distribución gaussiana

Después de comprobar el funcionamiento de cada uno de estos mecanismos de mutación, se decidió optar por el propuesto en la ecuación 14, al ser el operador que, siendo bastante simple, ofreció los mejores resultados bajo los parámetros definidos para estos ensayos.

Tras modificar uno de los genes del individuo, se comprueba que, una vez mutado, continúe dentro de los límites del problema. Cuando se ha completado la mutación de un elemento, se calcula su afinidad y se almacena en el primer gen de dicho individuo.

El siguiente paso es comparar los n mejores clones mutados con los n elementos que se seleccionaron inicialmente. En esta primera versión del algoritmo se comparaba el mejor clon con el mejor elemento, el segundo mejor clon con el segundo mejor elemento, y así sucesivamente hasta comparar los n clones. De cada pareja, aquel con el coste más bajo sería introducido de nuevo en la población.

Por último, se generan d nuevos individuos que sustituirán a los peores elementos de la población, y se calcula su coste. Se vuelve a ordenar la población en función de su coste, se calculan los distintos errores y se muestra por pantalla los resultados para esa generación. Todo este proceso se repetirá hasta alcanzar la generación límite.

Esta primera versión presentaba una serie de problemas. En primer lugar, si $n + d < NP$, esto quería decir que había una parte de la población sobre la que no se trabajaba. Por otra parte, al existir siempre una parte de la población que se renovaba en cada generación,

nunca se llegaba a apreciar una completa convergencia de la población. Otro problema se daba al comparar los clones mutados con los elementos originales. Al comparar el mejor con el mejor, el segundo mejor con el segundo mejor y demás, no se estaba asegurando que los mejores individuos entrasen en la población, ya que el segundo mejor clon podía ser peor que el segundo mejor elemento, pero aun así seguir siendo mejor que los otros elementos de la población.

En la segunda versión del algoritmo se introdujeron algunos cambios para poder corregir estos fallos. Lo primero que se buscó solucionar fue la existencia de puntos sobre los que no se llegara a trabajar. Se eliminó la variable d, con la que el usuario introducía el número de individuos. En esta versión, se generaba un número aleatorio N, que sería la cantidad de clones mutados a introducir de nuevo en la población. El resto de elementos serían eliminados y sustituidos por nuevos individuos.

En cuanto a la manera de seleccionar quienes debían formar parte de la nueva generación, se buscó la manera de conseguir que fuesen los mejores individuos globales los que sobreviviesen. Para ello, se introducen en un mismo vector los N mejores clones, tras el proceso de mutación, y los N mejores individuos de la población. Este vector se ordena en función del coste, y los N mejores elementos almacenados en este vector son introducidos de nuevo en la población.

Esta segunda versión del algoritmo solucionaba alguno de los problemas que mostraba la primera versión, pero seguía habiendo algunos fallos que tratar, como buscar la convergencia de la población.

La tercera versión del algoritmo busca arreglar este problema. Esta vez, N deja de ser un valor aleatorio y pasa a valer $N = 0.3 * NP$, redondeando al entero más cercano. Además, para conseguir la convergencia de la población, este valor irá creciendo al ir aumentando el número de generaciones, hasta llegar a reemplazar a toda la población. Sin embargo no resultó una buena solución, ya que la población convergería independientemente de haber llegado a una solución o no. Por lo tanto se decidió que N aumentase al ir acercándose el mejor individuo a la solución deseada.

Al comenzar a realizar pruebas para obtener los resultados experimentales, se observaron una serie de problemas que no habían aparecido hasta ese momento. Por una parte, como N tenía un valor fijo, podían darse casos en los que Nc fuese menor que N, lo que provocaba un error que detenía la ejecución del algoritmo. Para solucionar esto, se añadió una condición después de calcular Nc. Si se daba el caso de que el tamaño de la población de clones era menor que el número de elementos a introducir de nuevo en la población, entonces N pasaba a valer Nc.

Por último, se introdujo una última modificación en el algoritmo. Como se explicó en la sección anterior, el parámetro β controla el número de clones a obtener de cada individuo.

En un principio era el usuario quien decidía su valor. Para mejorar los resultados se decidió fijar el valor inicial de β en 0.1, para ir aumentándolo a medida que la población se acerca a la solución óptima. De esta forma, al principio se creaban pocos clones de cada individuo para mantener la diversidad en la población. Según se llegaba al óptimo, cada vez se iban creando más clones de los mejores individuos, para lograr la convergencia de la población alrededor de estos elementos. El fragmento de código que calculaba Nc pasó a situarse dentro del bucle principal del programa y su valor se recalcula para cada nueva generación.

De esta forma se llegó a la solución definitiva, en la que los parámetros que controlaba el usuario eran el tamaño de población, el límite de generaciones, el error a introducir en las medidas, la posición y orientación del robot y el número de elementos a seleccionar. El resto de los parámetros del algoritmo se irán modificando a lo largo de la ejecución, para buscar el funcionamiento óptimo. En la Figura 5 se puede ver el pseudocódigo para el algoritmo final.

```
3   function CLONALG ()
4       for i=1:NP
5           CALCULO_COSTE_PARA_POBLACION(i)
6       end
7       while (CONDICIONES_DE_CONVERGENCIA)
8           CALCULO_Nc
9           RELLENAR_Cj
10          CALCULO_tasa_mut
11          for i=1:NP
12              MUTAR_Cj
13              CALCULO_COSTE_PARA_Cj_mutado
14              SELECCION_ENTRE_Cj_Y_Cj_mutado
15          end
16          ELIMINAR_PEORES_ELEMENTOS_POBLACION
17          INTRODUCIR_NUEVOS_ELEMENTOS
18          CALCULO_COSTE_NUEVOS_ELEMENTOS
19          bestmem <- pob_ordenada(1,:)
20          COMPROVAR_COND_CONVERGENCIA
21      end
22  end
```

Figura 5. Pseudocódigo para CLONALG

5. RESULTADOS EXPERIMENTALES

En esta sección se mostrarán los resultados obtenidos empleando ambos métodos para poder llevar a cabo una correcta comparación entre ED y CLONALG. Se realizarán varias pruebas sobre ambos algoritmos para comprobar su capacidad para alcanzar una solución válida, además del tiempo necesario para ello y los efectos que tienen sobre ellos los distintos parámetros que los caracterizan.

El límite que separará los experimentos exitosos de los fallidos será que el coste del peor individuo de la población sea menor o igual a $3 \times \frac{Ns}{2}$, para Ns=100. Esto se debe a que cada una de las medidas obtenidas mediante el sensor laser se puede estimar empleando una distribución normal, o distribución gaussiana. A la hora de calcular el coste de un individuo, se suman todas estas medidas laser, por lo que el resultado pasará a seguir una distribución chi-cuadrado, para la cual, el valor que se espera obtener cuando el 95% de los elementos se encuentran dentro de la desviación típica de la distribución es Ns, y se obtiene de una tabla. Se emplea $3 \times \frac{Ns}{2}$ para tener un límite menos estricto a la hora de considerar el éxito de un experimento.

Lo primero será comprobar la capacidad de los algoritmos para localizar con éxito el robot, sin tener en cuenta ninguna limitación de iteraciones, en una sola etapa. Se considera como una etapa lo que transcurre entre dos medidas distintas del escáner 3D. De esta forma se puede comprobar si los dos métodos son capaces de llegar a una buena solución si se les da el tiempo suficiente o si se elige una población lo bastante amplia como para que sea posible explorar con profundidad varias regiones del espacio de soluciones, aumentando las posibilidades de llegar a un óptimo.

Lo siguiente que se debe comprobar es la capacidad del robot de localizarse cuando se añaden al problema datos acerca del movimiento. En este caso, debería ser más sencillo para el robot hallar su posición, ya que tendría más información con la que trabajar. Se probará la efectividad de los algoritmos limitando primero el número de etapas, usando una mayor cantidad de iteraciones por etapa. Esto es beneficioso en zonas con mayor cantidad de obstáculos, porque al localizarse en menos etapas, el robot recorre menos distancia antes de estar correctamente localizado, aunque aumentaría el tiempo que se necesita para poder completar una sola etapa, lo que haría que entre dos movimientos transcurriese mucho más tiempo. A continuación se limitará el número de iteraciones por etapa, lo que provocará que el tiempo que se tarde en completar una etapa sea menor, aunque serán necesario un mayor número de etapas para obtener una localización satisfactoria, lo que supone un riesgo para la integridad del robot y de lo que le rodea, ya que recorrerá mayor distancia antes de situarse en el mapa.

Para ambos experimentos, se comprobará como cada uno de los parámetros del algoritmo afecta a su capacidad de obtener una solución adecuada. Todos los resultados

serán presentados en tablas, apoyados con gráficas que faciliten la comprensión de estos al lector. Además, se medirá el tiempo que tarda cada algoritmo en obtener una solución, en función del tamaño de población y del número de iteraciones a realizar, para ambos métodos, mientras que para el CLONALG se añadirá el número de elementos a clonar.

Se ha diseñado un banco de pruebas en MATLAB para simplificar la realización de estos experimentos. El RELF-3D se integra dentro de este sencillo programa, que permite funcionar en dos modalidades distintas.

El primer modo de funcionamiento permite ejecutar el algoritmo seleccionado durante el número de veces deseado, generando para cada intento un punto distinto dentro del mapa. El tamaño de población, el error en las medidas y el número de iteraciones a realizar es el mismo para todas las ejecuciones del algoritmo. Cuando finaliza, calcula el porcentaje de casos que cumplen con el criterio que se ha tomado para considerar al robot correctamente localizado y muestra el tiempo que se ha tardado en realizar todos los experimentos.

El segundo modo permite llevar a cabo un análisis de sensibilidad del algoritmo seleccionado en función del tamaño de población y del número de iteraciones máximas a realizar. Se le pide al usuario con que versión del algoritmo quiere trabajar y cuantas veces quiere que se realice cada prueba. Después, para una población de 10 individuos y un límite de 10 iteraciones, comienza ejecutando dicho algoritmo el número de veces seleccionado, generando puntos aleatorios para cada ejecución, y proporcionando el porcentaje de casos en los que se localizó con éxito al robot. Después se pasa a aumentar el número de individuos a 50 y en las siguientes pruebas el tamaño de población continuará creciendo de 50 en 50 hasta llegar al límite, que se ha fijado en 500. Cuando se hayan probado todos los tamaños de población con el límite de 10 iteraciones, se cambiará el valor de este límite y pasará a valer 50, empezando de nuevo las pruebas para todos los tamaños de población. Después, el límite superior de iteraciones seguirá aumentando de 50 en 50 hasta llegar a 500, valor seleccionado como tope.

Además, ambos modos de funcionamiento permiten seleccionar el número máximo de etapas durante las cuales se llevará a cabo la localización. En caso de ser mayor que 1, generará direcciones aleatorias para el movimiento del robot, comprobando siempre que dicho movimiento no se salga del mapa, ni impacte contra un obstáculo.

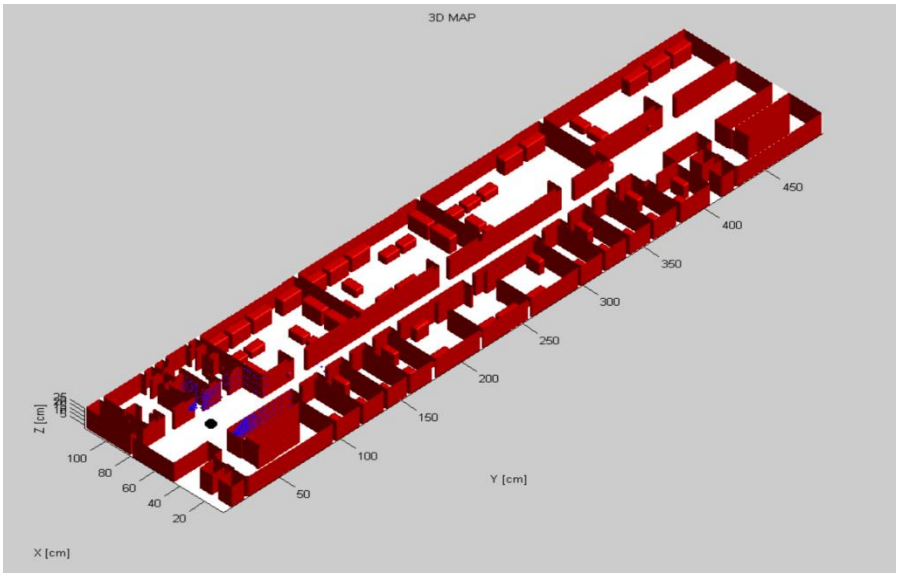

Figura 6. Mapa 3D de la UC3M empleado en el experimento

El mapa a emplear, mostrado en la Figura 6, es una simulación de una de las zonas de despachos y pasillos de la UC3M. Se ha generado empleando la técnica conocida como mapa de ocupación de celdillas, la cual genera un mapa del entorno a partir de variables binarias uniformemente espaciadas que indican la presencia de obstáculos en ese punto. Fue diseñado primero en 2D y luego extruido para obtener un entorno 3D. El mapa se divide en celdillas cúbicas de 12,1 cm de arista, y tiene unas dimensiones de $500 \times 119 \times 25$ celdillas. Las coordenadas del robot en el mapa se indican en celdillas, mientras que el error de posición se da en cm.

5.1. Resultados obtenidos mediante ED

Con el método de Evolución Diferencial los parámetros necesarios para el funcionamiento del algoritmo son la posición y orientación del punto real a encontrar, el error que se le añadirá a los resultados, el tamaño de población y el número de iteraciones. Se va a trabajar sobre la versión del algoritmo que incorpora el mecanismo de Tresh-Disc, ya que, como se puede observar en el trabajo previo sobre el RELF-3D [4], ha demostrado proporcionar los mejores resultados. De las cuatro versiones del algoritmo que presentan este mecanismo, se eligió la que emplea como función de coste la basada en el cálculo de la suma de los errores al cuadrado (L2-norm), y en la que la mutación de los individuos se genera de forma aleatoria, y no trabajando a partir del mejor individuo, lo que nos permite explorar una zona más amplia del espacio de soluciones y evita en ciertos casos la convergencia prematura del algoritmo.

a) Localización en una sola etapa

Como ya se explicó previamente, el primero de los experimentos consiste en comprobar la capacidad del algoritmo para proporcionar una solución válida. En caso de que la localización sea un éxito, se incrementa en uno la variable true. Si no consigue localizarse, se incrementará la variable false en uno. El porcentaje de éxitos se obtendrá dividiendo los casos exitosos entre el total y multiplicando el resultado por 100.

Se ejecutará 20 veces el algoritmo, lo cual proporcionará una resolución del 5%. Para este ensayo, se decidió trabajar con una población de 200 individuos, ya que con ese tamaño se obtuvieron buenos resultados en las pruebas previas, y se le permitió trabajar durante 500 iteraciones.

Este experimento se realizó 10 veces aplicando la misma configuración para poder estudiar si existe una repetitividad entre resultados y proporcionar de esta forma un conocimiento más preciso acerca de las capacidades que muestra este método. Los resultados obtenidos se muestran en la tabla 1.

Experimento	Porcentaje de éxitos
1	70%
2	65%
3	65%
4	45%
5	60%
6	55%
7	70%
8	55%
9	60%
10	70%

Tabla 1. Porcentaje de éxitos obtenido para 500 iteraciones, 200 individuos y 20 posiciones distintas

Dado que el experimento 4 presenta un resultado bastante inferior a los demás, y que este resultado no se vuelve a repetir, se puede considerar que bajo estas condiciones, el método de ED proporcionará un porcentaje de éxitos comprendido entre el 55% y el 70%. No son unos resultados especialmente altos, aunque hay que tener en cuenta que no se está empleando los datos que se obtienen del movimiento del robot, lo que limita la efectividad del método.

Este experimento se ha repetido para distintos valores tanto del tamaño de población como del límite superior de iteraciones. De esta forma se puede observar como cada uno de estos dos parámetros afectan al porcentaje de éxitos obtenido. Los valores elegidos para ambas variables son 10, 50, 100, 150, 200, 250, 300, 350, 400, 450 y 500. Los resultados se pueden observar en las tablas 2 a 12.

Nº Iteraciones	NP	Éxitos
10	10	0%
	50	0%
	100	0%
	150	0%
	200	0%
	250	0%
	300	0%
	350	0%
	400	0%
	450	0%
	500	0%

Tabla 2. Porcentaje de éxitos para 10 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
50	10	0%
	50	0%
	100	0%
	150	0%
	200	0%
	250	0%
	300	0%
	350	0%
	400	0%
	450	0%
	500	0%

Tabla 3. Porcentaje de éxitos para 50 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
100	10	0%
	50	0%
	100	20%
	150	5%
	200	5%
	250	0%
	300	15%
	350	15%
	400	5%
	450	15%
	500	5%

Tabla 4. Porcentaje de éxitos para 100 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
150	10	0%
	50	5%
	100	25%
	150	40%
	200	35%
	250	30%
	300	30%
	350	45%
	400	25%
	450	40%
	500	40%

Tabla 5. Porcentaje de éxitos para 150 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
200	10	0%
	50	0%
	100	40%
	150	55%
	200	45%
	250	40%
	300	55%
	350	65%
	400	40%
	450	60%
	500	60%

Tabla 6. Porcentaje de éxitos para 200 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
250	10	0%
	50	10%
	100	25%
	150	50%
	200	65%
	250	50%
	300	35%
	350	55%
	400	55%
	450	75%
	500	50%

Tabla 7. Porcentaje de éxitos para 250 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
300	10	0%
	50	15%
	100	20%
	150	30%
	200	45%
	250	40%
	300	60%
	350	50%
	400	70%
	450	75%
	500	75%

Tabla 8. Porcentaje de éxitos para 300 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
350	10	0%
	50	10%
	100	30%
	150	65%
	200	45%
	250	55%
	300	60%
	350	65%
	400	70%
	450	80%
	500	65%

Tabla 9. Porcentaje de éxitos para 350 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
400	10	0%
	50	5%
	100	20%
	150	15%
	200	65%
	250	50%
	300	70%
	350	65%
	400	70%
	450	45%
	500	75%

Tabla 10. Porcentaje de éxitos para 400 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
450	10	0%
	50	0%
	100	15%
	150	50%
	200	60%
	250	45%
	300	65%
	350	45%
	400	70%
	450	80%
	500	80%

Tabla 11. Porcentaje de éxitos para 450 iteraciones, variando NP

Nº Iteraciones	NP	Éxitos
500	10	0%
	50	15%
	100	30%
	150	35%
	200	55%
	250	60%
	300	70%
	350	65%
	400	75%
	450	70%
	500	70%

Tabla 12. Porcentaje de éxitos para 500 iteraciones, variando NP

De estas tablas se observa que el porcentaje de éxitos es directamente proporcional a ambas variables, tanto al número de iteraciones como al tamaño de población. Aun así, esta relación no es totalmente lineal, ya que se observan casos en los que un número menor de individuos ofrece un porcentaje mayor de casos en los que se ha localizado con éxito el robot.

En la Figura 7 se puede observar una gráfica que representa el porcentaje de éxitos frente al tamaño de población elegido. Cada uno de los colores representa un límite máximo de iteraciones distinto.

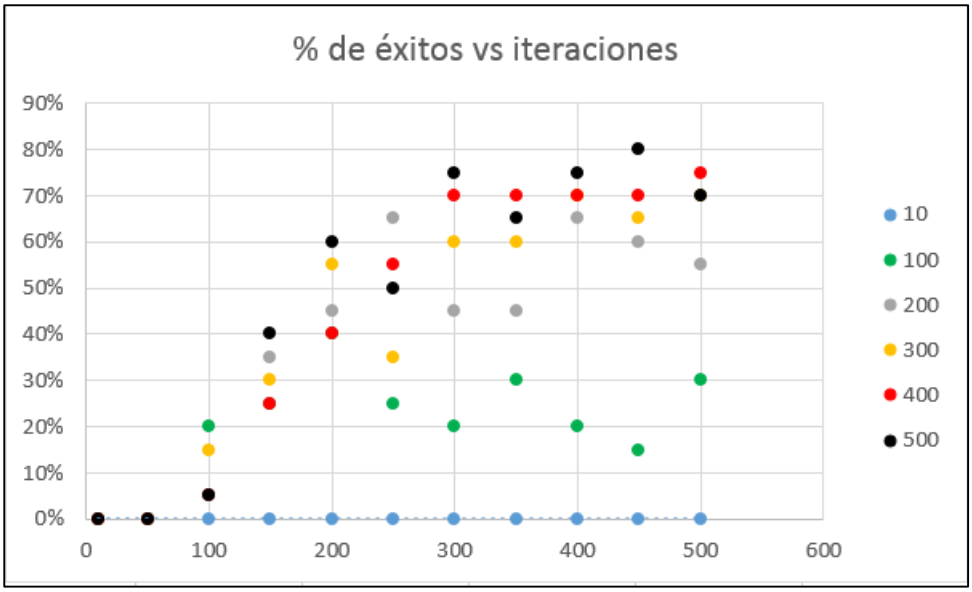

Figura 7. Porcentaje de éxitos frente al límite superior de iteraciones

Viendo la gráfica de la figura 7, se puede observar que entre ambas variables y el porcentaje de éxitos existe una relación directa. Respecto al límite superior de iteraciones, el porcentaje de éxitos suele ser mayor a medida que aumentan las iteraciones, aunque no siempre se cumple esto, ya que se pueden dar casos en los que un aumento en el límite de iteraciones no se corresponde con una mejora en los resultados. Esto se debe a una cierta aleatoriedad en los resultados, ya que si la población converge con rapidez hacia una zona del espacio de soluciones donde no se encuentre el óptimo global, el hecho de dejar correr el algoritmo durante más o menos iteraciones podría no conseguir que la búsqueda se desplace hacia dicho óptimo, haciendo que la localización no resulte satisfactoria.

En el caso del tamaño de población, se ve con mayor claridad que un aumento en el tamaño de población se corresponde con una mejora en los resultados, en especial para tamaños más pequeños. Sin embargo, a partir de una población de 200 individuos esta mejora no se nota con tanta claridad, ya que el porcentaje de éxitos alcanza el rango máximo que se ha conseguido obtener mediante este algoritmo. También se puede ver que la linealidad de los resultados aumenta con el tamaño de población, por eso para NP altos, al aumentar el límite superior de iteraciones mejoran los resultados y se observan menos casos en los que esta relación no se cumpla.

b) Localización en varias etapas

A continuación se comprobará de nuevo el porcentaje de éxitos que el algoritmo muestra a la hora de localizar la posición del robot, pero en este caso se considera que el robot está en movimiento. Esto supone que ya no se debe completar en una sola etapa, y

además se espera que los resultados sean mejores, ya que se añade información proveniente del movimiento.

Para la aplicación que se pretende dar a este algoritmo, interesa limitar el número de etapas en las que se alcanza un resultado, ya que si se permite que el robot avance durante demasiado tiempo sin encontrar su posición, podría llegar a chocar con un obstáculo. Por eso se va a fijar el límite de etapas en 5. Se calculará para diferentes tamaños de población y del límite máximo de iteraciones el porcentaje de casos en los que el robot se localiza con éxito. El criterio para determinar el éxito o fracaso del experimento será el mismo que en el primer experimento. Los tamaños de población serán 10, 100 y 200, mientras que los límites máximos de iteraciones serán 10, 100, 200, 300, 400 y 500. Los resultados se presentan en las tablas 13 a 18.

Nº de iteraciones	NP	% de éxitos
10	10	5
	100	0
	200	0

Tabla 13. Porcentaje de éxitos para 10 iteraciones

Nº de iteraciones	NP	% de éxitos
100	10	15
	100	30
	200	75

Tabla 14. Porcentaje de éxitos para 100 iteraciones

Nº de iteraciones	NP	% de éxitos
200	10	0
	100	25
	200	55

Tabla 15. Porcentaje de éxitos para 200 iteraciones

Nº de iteraciones	NP	% de éxitos
300	10	0
	100	30
	200	60

Tabla 16. Porcentaje de éxitos para 300 iteraciones

Nº de iteraciones	NP	% de éxitos
400	10	5
	100	25
	200	65

Tabla 17. Porcentaje de éxitos para 400 iteraciones

Nº de iteraciones	NP	% de éxitos
500	10	5
	100	25
	200	70

Tabla 18. Porcentaje de éxitos para 500 iteraciones

Para las primeras tablas, donde el límite superior de iteraciones está por debajo de 300, se obtienen mejores resultados empleando varias etapas en la localización en vez de una sola, lo cual es lógico, al emplear toda la información que posee el robot, tanto la extraída de los sensores de posición como la obtenida a partir del movimiento. Sin embargo, para 400 y 500 iteraciones, los resultados son muy similares a los que se obtienen utilizando solo una etapa. Esto se debe a que si se deja correr el algoritmo durante un número significativo de iteraciones, será capaz de converger en una sola etapa, por lo que es prácticamente irrelevante el número de etapas durante las que ocurra la localización.

c) Coste computacional del algoritmo

A la hora de lograr la localización satisfactoria de un robot móvil, uno de los principales problemas que se necesita solucionar es conseguir dicha localización en un periodo de tiempo lo suficientemente corto. Este tiempo depende de varios factores, destacando las características geométricas del entorno, el tamaño de población o el límite máximo de iteraciones.

En este apartado se va a comprobar el tiempo que el algoritmo tarda en obtener una solución, independientemente del resultado obtenido, y como los distintos parámetros afectan a este tiempo. Para comprobar el efecto del tamaño de población, se fijará el límite superior de iteraciones en 100 y se irá variando NP entre 10, 100, 200, 300, 400 y 500. Los resultados se pueden observar en la tabla 19.

Límite de iteraciones	NP	Tiempo (s)
100	10	0,17
	100	35,71
	200	43,79
	300	82,64
	400	123,66
	500	178,88

Tabla 19. Tiempos de ejecución para 100 iteraciones y varios valores de NP

Para ver como varía el tiempo de ejecución con respecto al límite superior de iteraciones se va a repetir el proceso, pero esta vez será el tamaño de la población el que se fijará en 100 mientas que el número de iteraciones será el que varíe. Se pueden observar los resultados en la tabla 20.

NP	Límite de iteraciones	Tiempo (s)
100	10	2,79
	100	36,32
	200	62,62
	300	41,82
	400	67,5
	500	51,36

Tabla 20.Tiempos de ejecución para NP 100 y varios límites de iteraciones

En ambos casos, el tiempo necesario para completar una etapa aumenta al aumentar tanto el límite máximo de iteraciones como el tamaño de población. Es con la variación de este último con la que se observan mayores incrementos. Para buscar la razón de esto se emplea una de las herramientas que proporciona MATLAB para poder conocer los cuellos de botella del proceso y ver como se reparte el tiempo entre los distintos archivos del programa. Esto se puede observar en la Figura 8.

Profile Summary

Generated 22-Sep-2014 07:03:24 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
dist_est_3d	10102	239.273 s	239.273 s	
testbench	1	259.866 s	8.268 s	
initialization	1	4.485 s	4.485 s	
fitness_3d	10100	1.602 s	1.602 s	
random	71119	5.655 s	1.522 s	
strmatch	71119	1.388 s	1.388 s	
cell.strmatch	71119	3.113 s	1.187 s	
unidrnd	71119	1.020 s	0.925 s	
iscellstr	142238	0.538 s	0.538 s	
alg_genet_3D	1	247.020 s	0.538 s	
stats\private\statsizechk (MEX-function)	71119	0.095 s	0.095 s	
local_3D_real	1	247.113 s	0.046 s	
testbench>inicio_pob	1	0 s	0.000 s	
sortrows	100	0 s	0.000 s	
sortrowsc (MEX-function)	100	0 s	0.000 s	

Figura 8. Tiempo empleado en cada una de las funciones que conforman el RELF-3D empleando el método de ED

La columna "Self Time" muestra el tiempo que cada función consume por sí misma, sin tener en cuenta el tiempo que consumen las funciones anidadas dentro de esta. En esta columna se puede comprobar que el principal cuello de botella del proceso se encuentra en la llamada a la función dist_est_3d. Dicha función estima las medidas que el láser obtendría en el caso de que el robot estuviese en una determinada localización. Se calcula para cada uno de los elementos de la población durante la generación de la población inicial y durante cada una de las iteraciones del algoritmo. Esto es lo que provoca que el tamaño de población sea en este caso el parámetro crítico a la hora de optimizar el tiempo de ejecución, ya que el resto de tiempos son casi despreciables frente a este (el tiempo empleado en la función testbench no se debe tener en cuenta, ya que este es el banco de pruebas diseñado para la toma de resultados experimentales). Además de las llamadas a esta función dentro del bucle principal del algoritmo, se emplea también fuera de este para calcular el coste de la población inicial.

Por último, se puede observar que no para todos los casos un aumento en el número de iteraciones supone un aumento en el tiempo de ejecución. Esto se debe a una pronta convergencia del algoritmo. Además, el ordenador que se utiliza para sacar los datos también afecta al tiempo requerido para completar la ejecución del algoritmo.

5.2. Resultados obtenidos mediante CLONALG

Para el CLONALG se llevarán a cabo los mismos experimentos que ya fueron realizados sobre el método de Evolución Diferencial. De esta forma se tendrá una base sólida para poder establecer una comparación. Los parámetros de control son los mismos que para el algoritmo de ED, además del número de clones a realizar para cada individuo. La función objetivo con la que se va a trabajar es la L2-Norm.

a) Localización en una sola etapa

Al igual que para el algoritmo de ED, el primer experimento que se lleva a cabo es comprobar la capacidad del CLONALG para ofrecer una solución válida al problema de la localización. Se ha fijado el límite de iteraciones máximo en 500, aunque se trabajará con una población menor de individuos. Esto es debido a que, tras la fase de clonación, el número crece principalmente en función del tamaño elegido para la población inicial, por lo que si a NP se le da un valor de 100, y se eligen 5 elementos para clonar, el tamaño de la población después de generar los clones pasa a ser de 195 individuos.

También existe un crecimiento de la población de clones en función del número de individuos a seleccionar, aunque este es un factor secundario frente al tamaño de la población inicial, ya que, según el valor de NP, se alcanzará un valor de n tal que, si se sigue aumentando, la población de clones solo aumentará en uno o dos individuos.

Para la primera prueba se ha seleccionado un tamaño inicial de población de 100 individuos, y se comprobará el éxito obtenido para tres valores distintos de n, que son 5, 15 y 25. Los resultados obtenidos se muestran en la tabla 21.

n	Experimento	% de éxitos
5	1	30
	2	30
	3	20
	4	25
	5	40
15	1	20
	2	35
	3	15
	4	20
	5	5
25	1	20
	2	10
	3	15
	4	20
	5	15

Tabla 21. Porcentaje de éxitos para NP=100, 500 iteraciones y varios valores de n

El porcentaje de éxitos empleando el CLONALG bajo estas condiciones se sitúa alrededor del 25%, porcentaje bastante bajo, aun cuando no se está utilizando los datos odométricos en la determinación de la localización del robot. También se puede observar que se obtienen mejores resultados si se seleccionan 5 individuos para clonar que los que se obtienen seleccionando 15 o 25. Esto se debe principalmente a dos motivos.

A la hora de analizar las soluciones que ofrecen de forma experimental ambos métodos, se ha observado que existe una ligera dispersión de los resultados entre ejecuciones sucesivas del algoritmo. Al fijar el número de experimentos en 20 se consigue una resolución del 5%, por lo que la diferencia entre un 20% o un 25% consiste tan solo en un experimento fallido o exitoso. Esta dispersión de resultados parece ser mayor cuando se trabaja con el CLONALG.

Por otra parte, aunque es cierto que al seleccionar un mayor número de individuos hay más posibilidades de llegar a la solución, los principales factores que marca la efectividad del método son el límite superior de iteraciones y el tamaño de población, por lo que puede darse el caso de que incrementar el valor de n no suponga una mejora en los resultados.

Posteriormente se ha realizado un análisis de sensibilidad sobre el CLONALG, para ver como los distintos parámetros de control afectan a los resultados que se pueden obtener con este método. Al igual que en el ensayo anterior, los tres valores de n elegidos son 5, 15 y 25. El tamaño de población presentará los valores 10, 50, 100, 150 y 200 y el límite superior de iteraciones tomará los valores 10, 50, 100, 150, 200, 250, 300, 350, 400, 450 y 500. Los resultados obtenidos se muestran en las tablas 22 a 54.

n	Nº de iteraciones	NP	% de éxitos
5	10	10	0
		50	0
		100	0
		150	0
		200	0

Tabla 22. Porcentaje de éxitos para 10 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	50	10	0
		50	0
		100	10
		150	0
		200	5

Tabla 23. Porcentaje de éxitos para 50 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	100	10	0
		50	5
		100	15
		150	10
		200	20

Tabla 24. Porcentaje de éxitos para 100 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	150	10	0
		50	0
		100	10
		150	20
		200	15

Tabla 25. Porcentaje de éxitos para 150 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	200	10	5
		50	10
		100	15
		150	5
		200	20

Tabla 26. Porcentaje de éxitos para 200 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	250	10	10
		50	10
		100	10
		150	30
		200	5

Tabla 27. Porcentaje de éxitos para 250 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	300	10	0
		50	10
		100	15
		150	20
		200	5

Tabla 28. Porcentaje de éxitos para 300 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	350	10	5
		50	5
		100	15
		150	30
		200	10

Tabla 29. Porcentaje de éxitos para 350 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	400	10	10
		50	10
		100	5
		150	20
		200	10

Tabla 30. Porcentaje de éxitos para 400 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	450	10	5
		50	0
		100	40
		150	20
		200	15

Tabla 31. Porcentaje de éxitos para 450 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
5	500	10	20
		50	10
		100	30
		150	10
		200	15

Tabla 32. Porcentaje de éxitos para 500 iteraciones y seleccionando 5 individuos

n	Nº de iteraciones	NP	% de éxitos
15	10	50	0
		100	0
		150	0
		200	5

Tabla 33. Porcentaje de éxitos para 10 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	50	50	5
		100	15
		150	10
		200	10

Tabla 34. Porcentaje de éxitos para 50 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	100	50	10
		100	5
		150	10
		200	20

Tabla 35. Porcentaje de éxitos para 100 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	150	50	5
		100	5
		150	15
		200	10

Tabla 36. Porcentaje de éxitos para 150 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	200	50	5
		100	20
		150	25
		200	15

Tabla 37. Porcentaje de éxitos para 200 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	250	50	0
		100	15
		150	15
		200	15

Tabla 38. Porcentaje de éxitos para 250 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	300	50	10
		100	15
		150	25
		200	35

Tabla 39. Porcentaje de éxitos para 300 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	350	50	10
		100	15
		150	5
		200	20

Tabla 40. Porcentaje de éxitos para 350 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	400	50	15
		100	10
		150	10
		200	0

Tabla 41. Porcentaje de éxitos para 400 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	450	50	20
		100	5
		150	25
		200	15

Tabla 42. Porcentaje de éxitos para 450 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
15	500	50	15
		100	20
		150	25
		200	20

Tabla 43. Porcentaje de éxitos para 500 iteraciones y seleccionando 15 individuos

n	Nº de iteraciones	NP	% de éxitos
25	10	50	0
		100	0
		150	0
		200	0

Tabla 44. Porcentaje de éxitos para 10 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	50	50	5
		100	0
		150	5
		200	15

Tabla 45. Porcentaje de éxitos para 50 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	100	50	5
		100	20
		150	15
		200	25

Tabla 46. Porcentaje de éxitos para 100 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	150	50	10
		100	15
		150	40
		200	25

Tabla 47. Porcentaje de éxitos para 150 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	200	50	30
		100	10
		150	25
		200	25

Tabla 48. Porcentaje de éxitos para 200 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	250	50	0
		100	20
		150	20
		200	10

Tabla 49. Porcentaje de éxitos para 250 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	300	50	15
		100	25
		150	15
		200	40

Tabla 50. Porcentaje de éxitos para 300 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	350	50	15
		100	0
		150	15
		200	25

Tabla 51. Porcentaje de éxitos para 350 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	400	50	15
		100	15
		150	20
		200	30

Tabla 52. Porcentaje de éxitos para 400 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	450	50	20
		100	0
		150	15
		200	25

Tabla 53. Porcentaje de éxitos para 450 iteraciones y seleccionando 25 individuos

n	Nº de iteraciones	NP	% de éxitos
25	500	50	10
		100	20
		150	25
		200	40

Tabla 54. Porcentaje de éxitos para 500 iteraciones y seleccionando 25 individuos

En este caso, la mejora en los resultados depende sobre todo del límite superior de iteraciones, y no tanto del tamaño de población, a diferencia del método de ED. Esto es debido a que la población sobre la que trabaja el CLONALG es la obtenida a partir del proceso de selección y clonación de los mejores individuos, lo que supone que va a ser mayor, generalmente, que el tamaño de población definido por el usuario.

Por otra parte, también se vuelve a observar lo que se comentaba antes acerca de la influencia del número de individuos que se van a seleccionar para ser clonados. Aunque los experimentos realizados con $n = 25$ presentan unos resultados ligeramente superiores a los que se realizaron usando un valor menor, esta mejoría no es realmente significativa, comparada con la influencia del tamaño de población y el límite de iteraciones.

En general, los resultados obtenidos trabajando con el CLONALG son bastante bajos, presentando como máximo un 40% de éxitos a la hora de localizar el robot en una sola etapa.

b) Localización en varias etapas

Al igual que se hizo para el algoritmo de ED, el siguiente experimento a realizar sobre el CLONALG es comprobar la capacidad que este método presenta para obtener una correcta localización cuando esta se lleva a cabo en varias etapas. Se ha fijado el valor de n en 5, los límites de iteraciones serán 10, 100, 200, 300, 400 y 500 y el valor de NP cambiará entre 10, 100 y 200. Los resultados se pueden observar en las tablas 55 a 60.

Límite de iteraciones	NP	% de éxitos
10	10	0
	100	15
	200	5

Tabla 55. Porcentaje de éxitos para 5 etapas, n=5 y 10 iteraciones

Límite de iteraciones	NP	% de éxitos
100	10	15
	100	15
	200	35

Tabla 56. Porcentaje de éxitos para 5 etapas, n=5 y 100 iteraciones

Límite de iteraciones	NP	% de éxitos
200	10	20
	100	20
	200	30

Tabla 57. Porcentaje de éxitos para 5 etapas, n=5 y 200 iteraciones

Límite de iteraciones	NP	% de éxitos
300	10	15
	100	10
	200	35

Tabla 58. Porcentaje de éxitos para 5 etapas, n=5 y 300 iteraciones

Límite de iteraciones	NP	% de éxitos
400	10	20
	100	25
	200	35

Tabla 59. Porcentaje de éxitos para 5 etapas, n=5 y 400 iteraciones

Límite de iteraciones	NP	% de éxitos
500	10	20
	100	15
	200	50

Tabla 60. Porcentaje de éxitos para 5 etapas, n=5 y 500 iteraciones

Como era de esperar, los resultados recogidos en este experimento mejoran en todos los casos a los obtenidos cuando no se tiene en cuenta la información acerca del movimiento del robot. Sin embargo, sigue presentando un porcentaje de éxitos muy bajo, alcanzando en el mejor de los casos un 50% de éxitos, lo que no permite que este método suponga una forma fiable de obtener la posición del robot.

c) Coste computacional del algoritmo

El último experimento que se va a llevar a cabo para el CLONALG es el cálculo del tiempo necesario para completar una ejecución del algoritmo. Se variará el límite superior de iteraciones, el número de elementos a seleccionar para ser clonados y el tamaño de población inicial. Pero primero es necesario tener en cuenta una serie de factores.

Para empezar, se va a comenzar con un estudio de los cuellos de botella para este algoritmo, empleando la misma herramienta que ya se usó para el método de ED. En la Figura 9 se puede ver el resultado de este proceso.

Profile Summary

Generated 23-Sep-2014 08:07:37 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
dist_est_3d	10102	490.090 s	490.090 s	
local_3D_real	1	651.855 s	149.460 s	
initialization	1	6.541 s	6.541 s	
fitness_3d	10100	2.841 s	2.841 s	
init_clonalg	1	2.320 s	2.320 s	
clonalg	1	493.349 s	0.561 s	
sortrows	301	0.039 s	0.033 s	
sortrowsc (MEX-function)	301	0.006 s	0.006 s	
local_3D_real>inicio_pob	1	0.003 s	0.003 s	

Figura 9. Distribución del tiempo de ejecución para el CLONALG

Al igual que ya ocurría para el método de ED, el cuello de botella se encuentra en la llamada a la función dist_est_3d, por lo que de nuevo, el tamaño de población va a suponer un parámetro importante a la hora de reducir el tiempo de ejecución. Para el CLONALG, el efecto es aún mayor, ya que la población efectiva con la que trabaja el algoritmo se incrementa en casi el doble, si se fija el valor de n en 5. Por ejemplo, seleccionando una población de 200 individuos, la población de clones tendrá un tamaño de 388 individuos. Este efecto se incrementa según aumenta el número de elementos que son seleccionados para ser clonados. Además, la velocidad con la que el algoritmo lleva a cabo cada iteración disminuye según se va acercando a la solución.

Dicho esto, a continuación se muestran los tiempos obtenidos para el CLONALG. En la tabla 62 se muestran los resultados recogidos fijando el tamaño de población en 100 y variando el límite superior de iteraciones. En la tabla 64 se ven los tiempos obtenidos si se fija el límite de iteraciones en 100 y se varía el tamaño de población.

NP	n	Límite de iteraciones	Tiempo (s)
100	5	10	2,56
		100	27,53
		200	46,32
		300	183
		400	566,91
		500	360,93
	15	10	2,69
		100	91,6
		200	201,96
		300	124,05
		400	100,7
		500	269,99

	25	10	6,73
		100	61,33
		200	123,67
		300	84,48
		400	235,3
		500	220,22

Tabla 61. Tiempos obtenidos fijando el tamaño de población en 100 y variando el límite de iteraciones

Límite de iteraciones	n	NP	Tiempo (s)
100	5	10	8,09
		100	61,29
		200	49,63
		300	150,6
		400	188,4
		500	162,52
	15		
		100	115,087
		200	56,75
		300	74,99
		400	114,006
		500	505,90
	25		
		100	234,26
		200	72,91
		300	123,16
		400	215,37
		500	249,81

Tabla 62. Tiempos obtenidos fijando el límite de iteraciones en 100 y variando el tamaño de población

5.3. Comparación de resultados obtenidos

Según los resultados de los distintos experimentos, el algoritmo de Evolución Diferencial supone una solución mucho más eficiente al problema de la localización de robots móviles. Cuando se trata de lograr esta localización empleando una sola etapa, el CLONALG presenta resultados ligeramente mejores para un tamaño de población de 10 individuos, aunque siguen siendo resultados muy bajos, alrededor de un 5% o un 10%. Para tamaños de población superiores, el método de ED presenta unos resultados muy superiores. Al final, se puede concluir que para este experimento, el algoritmo de ED es superior, ya que presenta un porcentaje máximo de éxitos del 80%, frente al 40% que ofrece el CLONALG.

Para ambos algoritmos, uno de los motivos que explica los bajos resultados se encuentra en el mapa del entorno. Como se puede observar en la Figura 6, al inicio de la sección de resultados

experimentales, el entorno en el que se realiza la investigación presenta un alto nivel de simetrías. Existen numerosas habitaciones que presentan la misma geometría, por lo que si el robot se encontrase en una de dichas habitaciones, le costaría descubrir en cuál de ellas está, ya que las medidas de los sensores serían muy similares entre ellas.

En la tabla 63 se pueden observar los errores de posición y orientación obtenidos con ambos algoritmos para varios puntos del mapa. Los parámetros empleados son 200 iteraciones, 200 individuos, un error en la medida (e_{dist}) del 1% y, en el caso del CLONALG, se seleccionarán 25 individuos para clonar. $e_p\ ED$ y $e_p\ C$ representan el error de posición obtenido para el algoritmo de Evolución Diferencial y para el CLONALG, respectivamente, mientras que $e_o\ ED$ y $e_o\ C$ representan los errores de orientación. X, Y, y Z son las coordenadas del punto seleccionado y Θ es la orientación.

Θ (°)	X (cm)	Y (cm)	Z (cm)	e_{dist} (%)	$e_p\ ED\ (cm)$	$e_o\ ED$ (°)	$e_p\ C\ (cm)$	$e_o\ C$ (°)
0	60	50	15	1	0,61	0,0001	0,603	0,0001
125	30	200	20	1	0,11	0,06	3,83	0,606
275	73	213	9	1	0,07	0,018	0,28	0,027
16	83	120	11	1	0,16	0,004	0,803	0,04
60	38	267	7	1	0,44	0,0053	1,406	0,28

Tabla 63. Errores obtenidos para una serie de puntos, empleando ambos algoritmos

De la tabla 63 se extrae la conclusión de que, en el caso de que se consiga localizar al robot con éxito, para ambos algoritmos se presenta una precisión de milímetros, llegando a ser de 1 o 2 centímetros en el caso más alejado, mostrando así la buena precisión que ofrece este método.

Cuando se aumenta el número de etapas que se pueden realizar para lograr la localización, ambos métodos mejoran los resultados obtenidos, pero aun así, el algoritmo de ED sigue presentando unos resultados mejores, un 75% frente al 50% que proporciona el CLONALG. Si se juntan los resultados obtenidos con este experimento y con el anterior, se puede concluir que el CLONALG no supone un método eficaz a la hora de llevar a cabo la tarea de la localización.

Si se comparan los tiempos que cada algoritmo emplea en esta tarea, es necesario primero considerar que para el CLONALG la única condición de convergencia es el límite superior de iteraciones, mientras que para Evolución Diferencial se considera también la convergencia de la población. Esto se debe a que en el CLONALG, el conjunto de la población no va a converger hasta que no se encuentre lo suficientemente cerca de la solución buscada, ya que se siguen añadiendo elementos nuevos a la población en cada iteración. Además, por el proceso de clonación de los mejores individuos, acaba por llenarse la población con clones de un determinado elemento, lo que podría provocar en algunos casos una convergencia prematura. Al final, como es el propio diseñador el que decide para qué momento se va a producir la convergencia (no se puede dar el caso de que uno de los elementos represente la solución buscada y que la población no haya convergido sobre ese individuo) se ha optado por no incluir la convergencia de la población como criterio de parada.

De las tablas de tiempos se puede comprobar que el algoritmo de Evolución Diferencial emplea mucho menos tiempo en completar si tarea que el CLONALG. Además, las variaciones con respecto al tamaño de población y al límite superior de iteraciones. Esto se debe al hecho de que el CLONALG reduce la velocidad a la que ejecuta cada iteración según se va acercando a la solución.

6. CONCLUSIÓN AL TRABAJO FIN DE GRADO

Con este trabajo se pretendía encontrar una alternativa al algoritmo de Evolución Diferencial para resolver el problema de la localización en robots móviles. Para ello se ha desarrollado un método basado en el Algoritmo de Selección Clonal, que aplica la Teoría de la Selección Clonal a la resolución de problemas de optimización.

Se han desarrollado varias versiones del algoritmo, comprobando como afectan a cada una los distintos parámetros y buscando la configuración óptima que permitiese obtener los mejores resultados posibles.

Sin embargo, como se puede observar en la sección de resultados experimentales, no se logró mejorar, ni alcanzar, el nivel de resultados que el método de ED proporciona, por lo que se puede concluir que, tal y como se ha implementado el CLONALG en esta investigación, no supone una alternativa viable al algoritmo de Evolución Diferencial.

Esto no quiere decir que haya que descartar por completo el CLONLAG como método para la resolución de este problema, pero se requiere continuar el trabajo para mejorar el funcionamiento del algoritmo. Las principales vías que deberían seguir las investigaciones posteriores son tres:

- Buscar una mejor configuración de parámetros para optimizar los resultados. La tasa de decaimiento de la mutación y el factor que controla el número de clones a generar resultan claves en el correcto funcionamiento del algoritmo, por lo que desarrollar un método que permitiese emplear en cada momento el valor necesario resultaría muy beneficioso a la hora de mejorar los resultados. Esto se puede implementar de forma adaptiva, en la que el propio algoritmo es capaz de variar el valor de los parámetros según lo requiera, o fijando diferentes combinaciones para encontrar la mejor.

- Desarrollar un nuevo operador de mutación, o mejorar el mecanismo ya existente. La fase de mutación de los clones es el proceso clave a la hora de explorar las regiones del espacio cercanas a los individuos de la población, por lo que lograr que la mutación de los individuos los convierta en mejor solución es el principal problema sobre el que se debería trabajar.

- Por último, se debería buscar la forma de reducir los tiempos de ejecución del algoritmo para obtener una localización precisa en el menor tiempo posible.

BIBLIOGRAFÍA

[1] G. Azkune, Navegación Autónoma. http://cuentos-cuanticos.com/2011/11/12/navegacion-autonoma/

[2] SLAM. http://es.wikipedia.org/wiki/SLAM_(rob%C3%B3tica)

[3] A. González Sieira, M. Mucientes Molina, *"Planificación de movimientos en robótica móvil utilizando retículas de estados"*, Escuela Técnica Superior de Ingeniería, Universidad de Santiago de Compostela, Santiago, 2011.

[4] F. Martín, L. Moreno, S. Garrido y D. Blanco, *"High accuracy global localization filter for 3D enviroments"*, Robotica (Online). Vol. 30, No. 3, pp.363-378, 2012.

[5] L. N. de Castro, F. J. Von Zuben, *"Learning and Optimization Using the Clonal Selection Principle",* IEEE Transactions on Evolutionary Computation, Vol 6, Nº3, June 2003.

[6] MATLAB. http://es.wikipedia.org/wiki/MATLAB

[7] J. García de Jalón, J. Ignacio Rodríguez, J. Vidal, *"Aprenda MATLAB como si estuviera en primero"*, Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid, Madrid, 2005.

[8] Información sobre el robot MANFRED-2 disponible en la página web del Robotics Lab de la UC3M. http://roboticslab.uc3m.es/roboticslab/robot.php?id_robot=5

[9] J. Leonard, H. F. Durrant-Whyte, *"Mobile Robot Localization by Tracking Geometric Beacons"*, IEEE Transactions on Robotic and Automation, Vol 7, No 3, pp. 376-382, June 1996.

[10] R. Negenborn, *"Robot Localization and Kalman Filters On finding your position on a noisy world"*, Utretch University, 2003.

[11] E. Kiriy, M. Buehler, *"Three-State Etended Kalman Filter forMobile Robot Localization"*, Department of Electronic Systems, Aalborg University, Denmark, 2002.

[12] J. Košecká y F. Li, *"Vision Based Topological Markov Localization"*, Department of Computer Science, George Mason University, Fairfax, Virginia, USA.

[13] D. Fox, W. Burgard, S. Thrun, *"Markov Localization for Mobile Robots in Dynamic Environments",* Journal of Artificial Intelligence Research 11, pp. 391-427, 1999.

[14] Monte Carlo Localization. http://en.wikipedia.org/wiki/Monte_Carlo_localization

[15] F. Dellaert, D. Fox, W. Burgard, S. Thrun, *"Monte Carlo Localization for Mobile Robots",* Computer Science Department, Carnegie Mellon University, Pittsburgh, and Institute of Computer Science III, University of Bonn, Bonn.

[16] R. Kümmerle, R. Triebel, P. Pfaff, W. Burgard, *"Monte Carlo Localization in Outdoor Terrains using Multi-Level Surface Maps",* Departament of Computer Science, University of Freiburg, Germany, and Autonomous Systems Lab, Swiss Federal Institute of Technology, Zurich, Switzerland.

[17] J. Kennedy, R.. Eberhart, *"Particle Swam Optimization"*, Proceedings of IEEE International Conference on Neural Networks, vol. IV, pp. 1942–1948, 1995.

[18] F. Jovan, R. Y. S. Dhiemas, M. Satki Alvissalim, W. Jatmiko, M. I. Fanany, A. Febrian, K. Sekiyama, T. Fukuda, *"Real Multiple Mobile Robots Implementation of PSO Algorithm for Odor Source Localization"*, International Conference on Advanced Computer Science and Information Systems, pp. 253-258, 2010.

[19] R. Havangi, M. Ali Nekoui, M. Teshnehlab, *"A Multi Swarm Particle Filter for Mobile Robot Localization"*, IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 2, May 2010.

[20] S. Se, D. L. J. Little, *"Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks"*, The International Journal of Robotics Research, Sage Publications, Vol. 21, No. 8, pp. 735-758, August 2002.

[21] Y. J. Lee, S. Sung, *"Vision Based SLAM for Mobile Robot Navigation Using Distributed Filters"*, Konkuk University, Seoul, South Korea.

[22] R. Caballero, J. Molina, M. Luque, A. Torrico, T. Gómez, *"Algoritmos genéticos para la resolución de problemas de Programación por Metas Entera. Aplicación a la Economía de la Educación",* Departamento de Economía Aplicada, Universidad de Málaga, Málaga.

[23] N. Gil Loñodo, *"Algoritmos Genéticos",* Escuela de Estadística, Universidad Nacional de Colombia, Colombia, 2006

[24] P. D. Gutiérrez, I. Triguero, F. Herrero, "Algoritmos basados en nubes de partículas y evolución diferencial para el problema de optimización continua: un estudio experimental", Actas del VIII Congreso Español sobre Metaheurística, Algoritmos Evolutivos y Bioinspirados (MAEB12), Albacete, pp. 144-156, Febrero 8-10, 2012.

[25] Transparencias de la asignatura de Bioinformática, Tema 10, "Algoritmos Basados en Evolución Diferencial", Curso 2013-2014, Universidad de Granada.

[26] R. Storn, K. Price, *"Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces"*, 1995.

[27] J. D. Farmer, N. H. Packard, A. S. Perelson, *"The Inmune System, Adaptation and Machine Learning"*, Physica, pp. 187-204, North-Holland, Amsterdam, 1986.

[28] J. H. Holland, *"Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-base systems"*, Machine Learnig 2, R. S. Michalski, J. G. Carbonell y T. M. Mitchell, eds. (Morgan Kauffman, Los Alto, in press), cap. 20.

[29] A. Watkins, J. Timmis, L. Boggess, *"Artificial Inmune Recognition System (AIRS): An Inmune-Inspired Supervised Learning Algorithm"*, Genetic Programing and Evolvable Machines 5, pp. 291-317, 2004

[30] J. Kelsey, J. Timmis, *"Inmune Inspired Somatic Contiguous Hypermutation for Function Optimisation"*, Computing Laboratory, University of Kent, Canterbury, Kent, UK.

[31] S. Forrest, A. Perelson, L. Allen, R. Cherukuri, *"Self-Nonself Discrimination in a Computer"*, IEEE Symposium on Research in Security and Privacy, 1994.

[32] N. Jerne, *"Towards a network theory of the immune system"*, Ann. Immunol. (Inst. Pasteur), 125C, pp. 373-389, 1974

[33] L. N. de Castro, F. J. Von Zuben, *"aiNet: an Artificial Inmune Network for Data Analysis"*, in "*Data Mining: A Heuristic Aproach"*, Hussein A. Abbass, Ruhul A. Sarker, and Charles S. Newton, Idea Group Publishing, USA, March de 2001.

[34] L. de Castro, J. Timmis, *"An Artificial Immune Network for Multimodal Function Optimization"*, Proceedings of IEEE Congress on Evolutionary Computation, vol. 1, pp. 699-674, Hawaii, May 2002.

[35] J. Greensmith, U. Aickelin, S. Cayzer, *"Introducing Dendritic Cells as a Novel Inmune-Inspired Algorithm for Anomaly Detection"*, Artificial inmune Systems, Vol. 3627, pp 153-167, 2005.

[36] J. C. Herrera Lozada, *"Sistema Inmune Artificial con Población Reducida para Optimización Numérica"*, Tesis Doctoral, México D.F., Junio 2011.

[37] K. M. Woldemariam, *"Multimodal and Constrained Optimization in Artificial Inmune System"*, Bahir Dar University, Bahir Dar, Ethiopia, 2004.

[38] S. Chittineni, A. N. S. Pradeep, D. Godavarthi, S. C. Satapathy, S. M. Krishna, P. V. G. D. P. Reddy, *"Design of Fixed and Ladder Mutation Factor-Based Clonal Selection Algorithm for Solving Unimodal and Multimodal Functions",* Applied Computational Intelligence and Soft Computing, Vol 2011 (2011), Article ID 210918.

ANEXO A: Planificación y Presupuesto del proyecto

A.1) Planificación del proyecto

A lo largo de la realización de este proyecto se ha ido pasando por diferentes etapas. Se comenzó realizando una investigación acerca del trabajo previo sobre el RELF-3D, y acerca del CLONALG. Posteriormente se comenzó el trabajo de desarrollo de la solución requerida, que fue pasando por varias etapas, al irse desarrollando diferentes versiones del algoritmo. La última etapa del desarrollo coincidió con el inicio de la etapa de experimentación. La última etapa comprende la redacción de la memoria, y se llevó a cabo en paralelo a la experimentación y a las últimas modificaciones del algoritmo.

En la Figura 10 se puede observar la cronología de este proyecto. La evolución del trabajo durante los primeros meses fue limitada, mientras que a partir del mes de Julio se comenzaron a experimentar mayores avances en el trabajo.

Fecha	Descripción
1-2-2014	Se comienza la investigación acerca del RELF-3D y el CLONALG
11-2-2014	Se comienza el estudio del código desarrollado para el RELF-3D
20-4-2014	Se comienza el trabajo de implementación del CLONALG
7-7-2014	1ª Versión del algoritmo terminada
14-7-2014	2ª Versión del algoritmo terminada
19-7-2014	3ª Versión del algoritmo terminada
20-7-2014	Comienzo de la redacción de la memoria
1-8-2014	Se comienza la adquisición de datos experimentales para el algoritmo de ED
11-9-2014	Versión final del algoritmo terminada. Se comienza la adquisición de datos experimentales para el CLONALG
22-9-2014	Fin de la obtención de datos experimentales
22-9-2014	Fin de la redacción de la memoria

Figura 10. Cronología del proyecto

A continuación, en la Figura 11 se presenta el diagrama de flujo del trabajo.

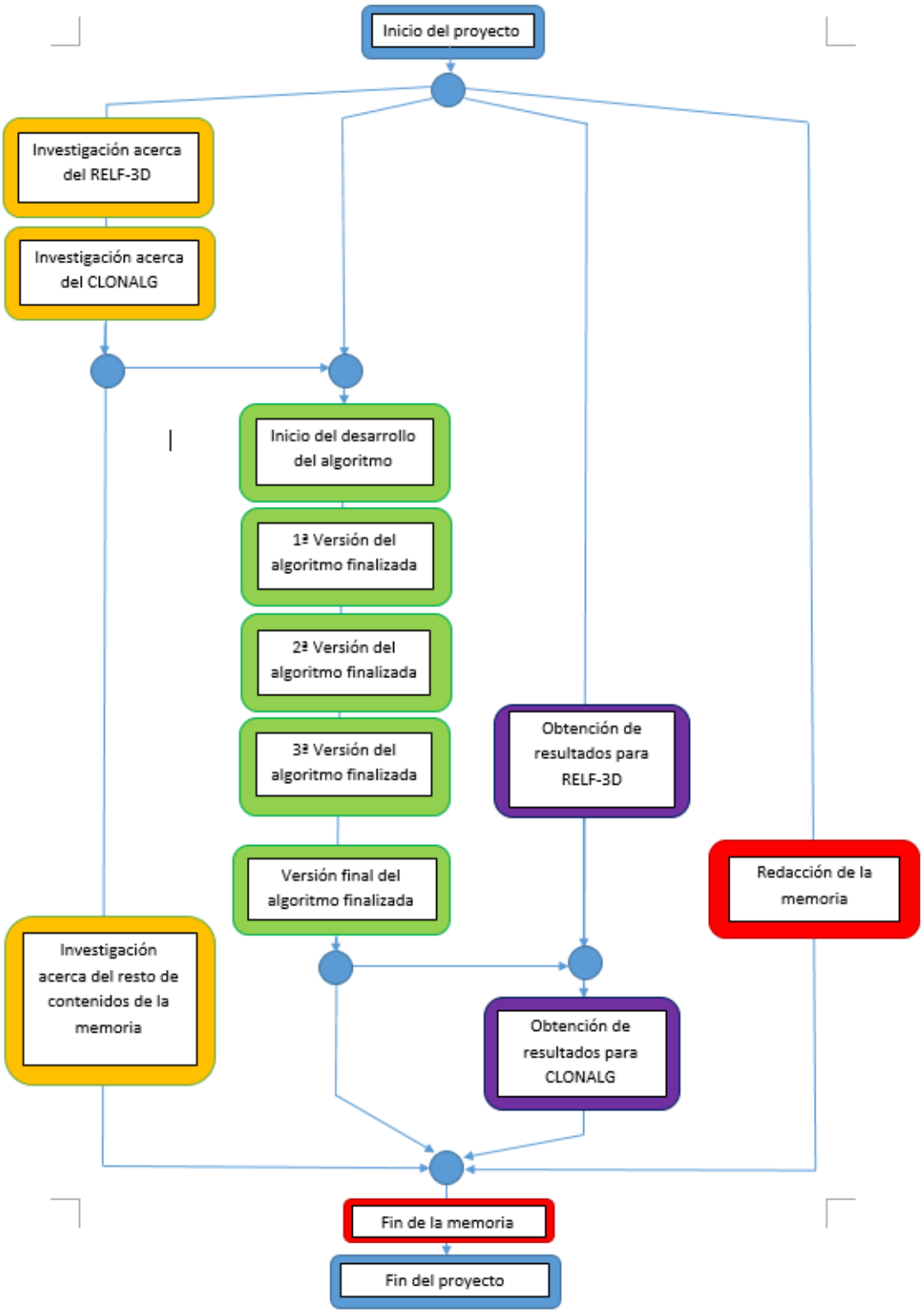

Figura 11. Diagrama de flujo del proyecto

Los recuadros amarillos hacen referencia a las tareas incluidas en la fase de investigación, e incluyen tanto la adquisición de conocimientos para poder implementar la

solución buscada, como la búsqueda, filtrado y comprensión de datos para añadir a la memoria del trabajo.

Los recuadros verdes pertenecen a las tareas propias de la fase de programación, desde el inicio del desarrollo del algoritmo hasta la implementación de la versión final.

Los recuadros morados contienen las tareas pertenecientes a la fase de experimentación. Se incluye una tarea por cada algoritmo sobre el que se trabaja, aunque cada una de ellas se podría dividir en varias subtareas que consisten en los distintos experimentos llevados a cabo sobre cada algoritmo.

Por último, las tareas en el interior de los recuadros rojos son las pertenecientes al proceso de redacción de la memoria del proyecto.

En este trabajo, la secuencia crítica de tareas es la siguiente:

- Investigación acerca del RELF-3D
- Investigación acerca del CLONLAG
- Inicio del desarrollo del algoritmo
- 1ª versión del algoritmo terminada
- 2ª versión del algoritmo terminada
- 3ª versión del algoritmo terminada
- Versión final del algoritmo terminada
- Obtención de resultados para CLONALG
- Fin de la memoria

Se considera secuencia crítica al conjunto de tareas encadenadas cuyo retraso provocaría un retraso en la fecha de fin del proyecto. Cualquiera de estas tareas no puede empezar antes de que la anterior termine, y a su vez restringen el inicio de la siguiente.

A.2) Presupuesto del proyecto

En cuanto al presupuesto del proyecto, al ser un trabajo de investigación, solo se van a tener en cuenta los sueldos tanto del ingeniero encargado del desarrollo del trabajo, que recibirá un salario de 8 € por hora de trabajo, y del tutor encargado de ejercer una cierta supervisión sobre la investigación, el cual recibirá un salario de 25 € por hora.

En cuanto a la duración del proyecto, se separa en varias fases. Entre los meses de febrero y finales de abril, la carga de trabajo asciende a un total de 30 horas. A partir del día 17 de junio, se comenzó a aplicar una jornada de 8 horas diarias a la investigación, de lunes a viernes, y se continuó con este ritmo hasta la fecha de fin del proyecto.

En total se han dedicado a este proyecto 710 horas, lo que supone un coste de 5680 euros. A esto hay que sumarle el salario del tutor. Se considera una carga de trabajo de 8 horas, por lo que el coste total, antes de impuestos se eleva a 5880 euros, lo que después de añadirle el IVA resulta en un total de SIETE MIL CIENTO CATORCE EUROS CON OCHO CENTIMOS.