INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA
EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

U P I I T A

"Simulación de un sistema evolutivo para que un robot cuadrúpedo aprenda a desplazarse tomando como método de aprendizaje los algoritmos genéticos"

Que para obtener el título de

"Ingeniero en Biónica"

Presenta el alumno:

Mejia Sosa Melissa

Asesor(es)
M. en C. Álvaro Anzueto Ríos
Dr. Carlos Artemio Coello Coello

M é x i c o CD. MX., Enero 2021

INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA
EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

U P I I T A

"Simulación de un sistema evolutivo para que un robot cuadrúpedo aprenda a desplazarse tomando como método de aprendizaje los algoritmos genéticos

Que para obtener el título de

"Ingeniero en Biónica"

Presenta el alumno:

Mejia Sosa Melissa

Sinodales

Dr. Blanca Tovar Corona Presidente	M. en E. Cuauhtémoc Valaguez Velázquez Secretario
M. en C. Álvaro Anzueto Ríos Asesor	Dr. Carlos Artemio Coello Coello Asesor

M é x i c o CD. MX., Enero 2021

AGRADECIMIENTOS

Al Instituto Politécnico Nacional por permitirme terminar mis estudios superiores.

Al M. en C. Álvaro Anzueto Ríos que apoyó mi idea de trabajo terminal y me proporcionó las herramientas necesarias para llevarlo a cabo.

Un especial agradecimiento al Dr. Carlos Coello Coello por siempre estar al pendiente de mi trabajo y brindarme los conocimientos necesarios para realizar mi proyecto.

A mi pareja, que estuvo a mi lado en los momentos más difíciles.

A mis abuelas que siempre estuvieron a mi lado cuando más las necesite, gracias a todo su cariño y apoyo hoy culmino mis estudios.

A todos aquellos que creyeron en mí, gracias.

INDICE GENERAL

CAPITULO 1. INTRODUCCION....1

1.1 Antecedentes....2
1.1.1 Algoritmos evolutivos....2
1.1.2 Robótica evolutiva....3
1.1.2.1 Biorobotica evolutiva....3
1.1.2.2 Robótica cognitiva....4
1.1.2.3 Robótica de enjambres....5
1.1.2.4 Robótica modular....5
1.1.2.5 Otras aplicaciones....6

CAPITULO 2. OBJETIVOS....9

2.1 Objetivo General....9
2.2 Objetivos específicos....9

CAPITULO 3. PLANTEAMIENTO DEL PROBLEMA....11

CAPITULO 4. JUSTIFICACION....13

CAPITULO 5. MARCO TEORICO....15

5.1 Neo-Darwinismo....15
5.2 Computación evolutiva....15
5.3 Algoritmo genético....16
5.3.1 Bases biológicas....16
5.3.2 Conceptos de los algoritmos genéticos....17
5.3.3 Estructura del algoritmo....18
5.3.4 Operadores genéticos....20
5.3.4.1 Selección....20
5.3.4.2 Cruza....22
5.3.4.3 Clonación....24
5.3.4.4 Elitismo....24
5.3.4.5 Mutación....24
5.3.4.6 Función de aptitud....25

CAPITULO 6. DESARROLLO EXPERIMENTAL....27

6.1 Elección del diseño....27
6.2 Modelado del diseño....28
6.3 Migración del diseño a CoppeliaSim....28
6.3.1. Agrupar las piezas....30
6.3.2. Crear articulaciones....31
6.3.3. Diseñar modelo dinámico del robot....31
6.3.4. Crear jerarquía....32

6.4 Conexión de CoppeliaSim con Python 35
6.5 Marcha ideal 37
6.6 Algoritmo genético 41
6.6.1 Codificación 41
6.6.2. Individuos 42
6.6.3 Selección 42
6.6.4. Cruza, mutación y clonación 42
6.6.5. Aptitud 43
6.6.5.1 Avance 46
6.6.5.2. Estabilidad 48
6.6.6. Elitismo 49
6.7. Complicaciones durante la simulación 51
6.7.1. Caídas 51
6.7.2. Colisiones 51
CAPITULO 7. ANALISIS E INTERPRETACION DE RESULTADOS 53
7.1 Análisis de resultados 53
7.1.1. Cruza en 2 puntos 54
7.1.1.1 Variando el porcentaje de mutación 54
7.1.1.2 Variando el porcentaje de cruza 61
7.1.1.3. Combinando parámetros 66
7.1.2. Cruza en 1 punto 69
7.1.2.1. Variando porcentajes de mutación 69
7.1.2.2. Variando el porcentaje de cruza 76
7.1.2.3 Combinando parámetros 81
7.2 Interpretación de resultados 84
CAPITULO 8. DISCUSION 87
CAPITULO 9. CONCLUSIONES 89
CAPITULO 10. SUGERENCIAS PARA INVESTIGACIONES FUTURAS 91
CAPITULO 11. BIBLIOGRAFIA 93
ANEXO I. Marcha Ideal 95
ANEXO II. Código de Marcha ideal 97
ANEXO III. Orientaciones de la base del robot. 101
ANEXO IV. Posiciones de la base del robot en el eje “z”. 103
ANEXO V. Algoritmo genético 105

INDICE DE FIGURAS

Figura 1.1: Modelo 3 de columna vertebral biomimética de un tiburón [11]. 4
Figura 1.2: El kit LEGO MindStorms [12]. 4
Figura 1.3: Trayectoria articular del auto-ensamble realizada por un algoritmo genético [14]. 6
Figura 1.4: Modelo de robot cuadrúpedo que aprende a caminar utilizando algoritmos evolutivos [16]. 7
Figura 1.5: Prototipo experimental de mecanismo de marcha, obtenido por algoritmos de evolución diferencial [18]. 7
Figura 5.1: Genotipo, cromosoma y genes [16]. 17
Figura 5.2: Ejemplo de una cadena cromosómica. 17
Figura 5.3: Ejemplo de un gen con 3 bits. 17
Figura 5.4: Diagrama de flujo del funcionamiento básico de un algoritmo genético. 19
Figura 5.5: Cruza de un punto. 23
Figura 5.6: Cruza de dos puntos. 23
Figura 5.7: Cruza uniforme. 24
Figura 5.8: Ejemplo de mutación. 25
Figura 6.1: Modelo de robot cuadrúpedo de la tesis “Robot Cuadrúpedo Inspirado en la Marcha de un Can” [23] 27
Figura 6.2: Vista isométrica del robot cuadrúpedo ensamblado. 28
Figura 6.3: Robot orientado, posicionado y escalado. 29
Figura 6.4: Iconos para posicionar y orientar al robot. 29
Figura 6.5: Robot agrupado. 30
Figura 6.6: a) Posición de una articulación en el modelo del robot cuadrúpedo, b) Articulaciones agregadas al diseño del robot cuadrúpedo. 31
Figura 6.7: Modelo dinámico del robot cuadrúpedo. 32
Figura 6.8: Jerarquia del robot cuadrupedo. 33
Figura 6.9: Marcha Ideal. 40
Figura 6.10: Plano x,y,z respecto a las coordenadas del mundo. 44
Figura 6.11: Base del robot cuadrúpedo. 45

Figura 6.12: Inclinación en orientación Alpha.....45
Figura 6.13: Inclinación en orientación Beta. 46
Figura 6.14: Inclinación en orientación Gamma. 46
Figura 6.15: Posición del robot después de iniciar la simulación. 47
Figura 6.16: Método para analizar el avance del robot. 47
Figura 6.17: Método para asegurar un avance entre ciclos de marcha.....47
Figura 6.18: Método para evaluar la orientación en Alpha y Beta.....48
Figura 6.19: Método para evaluar la orientación en Gamma. 48
Figura 6.20: Diagrama de flujo de la estructura del algoritmo genético.....50
Figura 6.21: Posibles colisiones entre patas. 50
Figura 6.22: Posibles colisiones entre las patas y la base del robot. 50
Figura 6.23: Posibles colisiones del robot cuadrúpedo en el simulador. 51
Figura 7.1: Promedio de aptitud de mejor individuo con mutación de 0.001. 56
Figura 7.2: Aptitud promedio de la población con mutación de 0.001. 56
Figura 7.3: Promedio de aptitud de mejor individuo con mutación de 0.005. 58
Figura 7.4: Aptitud promedio de la población con mutación de 0.005. 58
Figura 7.5: Promedio de aptitud de mejor individuo con mutación de 0.01. 60
Figura 7.6: Aptitud promedio de la población con mutación de 0.01. 60
Figura 7.7: Promedio de aptitud de mejor individuo con cruza de 0.75. 63
Figura 7.8: Aptitud promedio de la población con cruza de 0.75. 63
Figura 7.9: Promedio de aptitud de mejor individuo con cruza de 0.9. 65
Figura 7.10: Aptitud promedio de la población con cruza de 0.9. 65
Figura 7.11: Promedio de aptitud de mejor individuo combinando parámetros. 68
Figura 7.12: Aptitud promedio de la población combinando parámetros. 68
Figura 7.13: Promedio de aptitud de mejor individuo con mutación de 0.001. 71
Figura 7.14: Aptitud promedio de la población con mutación de 0.001. 71
Figura 7.15: Promedio de aptitud de mejor individuo con mutación de 0.005. 73
Figura 7.16: Aptitud promedio de la población con mutación de 0.005. 73
Figura 7.17: Promedio de aptitud de mejor individuo con mutación de 0.01. 75
Figura 7.18: Aptitud promedio de la población con mutación de 0.01. 75
Figura 7.19: Promedio de aptitud de mejor individuo con cruza de 0.75. 78
Figura 7.20: Aptitud promedio de la población con cruza de 0.75. 78
Figura 7.21: Promedio de aptitud de mejor individuo con cruza de 0.9. 80
Figura 7.22: Aptitud promedio de la población con cruza de 0.9. 80
Figura 7.23: Promedio de aptitud de mejor individuo combinando parámetros. 83
Figura 7.24: Aptitud promedio de la población combinando parámetros. 83

INDICE DE TABLAS

Tabla 6.1 Masas del robot cuadrúpedo. 34
Tabla 6.2 Torques del robot cuadrúpedo. 34
Tabla 6.3 Rangos de movimiento de la base del robot en cada orientación. 48
Tabla 7.1 Evolución de la aptitud del mejor individuo con mutación de 0.001. 55
Tabla 7.2 Evolución de la aptitud del mejor individuo con mutación de 0.005. 57
Tabla 7.3 Evolución de la aptitud del mejor individuo con mutación de 0.01 59
Tabla 7.4 Mejor individuo respecto al porcentaje de mutación. 61
Tabla 7.5 Aptitud promedio de la población respecto al porcentaje de mutación. 61
Tabla 7.6 Evolución de la aptitud del mejor individuo con cruza de 0.75. 62
Tabla 7.7 Evolución de la aptitud del mejor individuo con cruza de 0.9. 64
Tabla 7.8 Mejor individuo respecto al porcentaje de cruza. 66
Tabla 7.9 Aptitud promedio de la población respecto al porcentaje de mutación. 66
Tabla 7.10 Evolución de la aptitud del mejor individuo combinando parámetros. 67
Tabla 7.11 Mejores individuos de las pruebas. 69
Tabla 7.12 Aptitud promedio de la población de los mejores individuos de las pruebas. 69
Tabla 7.13 Evolución de la aptitud del mejor individuo con mutación de 0.001. 70
Tabla 7.14 Evolución de la aptitud del mejor individuo con mutación de 0.005. 72
Tabla 7.15 Evolución de la aptitud del mejor individuo con mutación de 0.01. 74
Tabla 7.16 Mejor individuo respecto al porcentaje de mutación. 76
Tabla 7.17 Aptitud promedio de la población respecto al porcentaje de mutación. 76
Tabla 7.18 Evolución de la aptitud del mejor individuo con cruza de 0.75. 77
Tabla 7.19 Evolución de la aptitud del mejor individuo con cruza de 0.9. 79
Tabla 7.20 Mejor individuo respecto al porcentaje de cruza. 81
Tabla 7.21 Aptitud promedio de la población respecto al porcentaje de cruza. 81
Tabla 7.22 Evolución de la aptitud del mejor individuo combinando parámetros. 82
Tabla 7.23 Mejores individuos de las pruebas. 84
Tabla 7.24 Aptitud promedio de la población de los mejores individuos de las pruebas. 84

Tabla 7.25 Mejores combinaciones de cada método. 86

RESUMEN

El presente proyecto muestra la evolución del aprendizaje de un robot cuadrúpedo con tres grados de libertad en cada extremidad, que aprende a desplazarse sin haber sido programado específicamente para realizar dicha acción, con el uso de algoritmos genéticos, dentro de un entorno de simulación.

Primeramente, se realizó el diseño del robot pieza por pieza en SolidWorks para después incorporarlo dentro del ambiente de simulación. La simulación se realizó en CoppeliaSim (llamado anteriormente V-REP) en un espacio plano sin obstáculos. Su objetivo fue mostrar únicamente al robot y la evolución del movimiento.

Se optó por realizar la demostración del algoritmo genético dentro de un entorno de simulación, debido a que requería de una gran cantidad de tiempo, ya que el robot debía de probar cada posible solución generada.

La programación del algoritmo genético se realizó en Python y la comunicación entre el ambiente de programación y la simulación se hizo mediante una interfaz de programación de aplicaciones llamada API por sus siglas en inglés.

El algoritmo genético es una técnica de búsqueda estocástica por lo que se realizaron varias ejecuciones independientes del mismo, variando y combinando los parámetros de control del sistema (número de generaciones, número de individuos, porcentaje de cruza y porcentaje de mutación), a fin de tener una idea más clara de su funcionamiento promedio.

Palabras clave:
Evolución, Algoritmo Genético, Aprendizaje, Robot Cuadrúpedo, Simulación.

ABSTRACT

This project shows the evolution of the learning of a quadruped robot with three degrees of freedom in each limb, that it learns to move without being specifically programmed to carry out this action, with the use of genetic algorithms, within a simulation environment.

First, the design of the robot was carried out piece by piece in SolidWorks, and then it was incorporated into the simulation environment. The simulation was done in CoppeliaSim (previously called V-REP) in a flat space without obstacles. Its objective was to show only the robot and the evolution of its movement.

We decided to perform the demonstration of the genetic algorithm in a simulation environment because of the large amount of time that it requires, since the robot must test every possible solution produced.

The programming of the genetic algorithm was done in Python and the communication between the programming environment and the simulation environment was done with an Application Programming Interface (API).

The genetic algorithm is a stochastic search technique, and therefore, it is necessary to perform several independent runs varying and combining the parameters of the control system (number of generations, number of individuals, crossover and mutation rates) in order to have a more clear idea of its average performance.

Keywords: *Evolution, Genetic Algorithm, Learning, Quadruped Robot, Simulation.*

GLOSARIO

Neodarwinismo: Es un planteamiento sobre la teoría evolucionista, que unifica la teoría evolutiva de Darwin, el seleccionismo y la genética.
Computación evolutiva: Engloba una serie de técnicas inspiradas en la teoría Neodarwiniana.
Técnica heurística: Es un conjunto de pasos que deben de realizarse para identificar en el menor tiempo posible una solucion de alta calidad para un determinado problema.
Convergencia: En la computación evolutiva es la progresión hacia la uniformidad. Por ejemplo, cuando el 95% de los individuos tienen el mismo valor de aptitud.
Bloques constructores: En la computación evolutiva es el conjunto de valores que generan buenas soluciones a un problema.
Algoritmo genético: Métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización, basados en simular el proceso genético de los organismos vivos.
Operadores genéticos: Alteran la composición de los descendientes de una población.
Codificación: Método para representar un conjunto de soluciones en un algoritmo genético.
Cruza: Método para la recombinación de individuos de una población.
Mutación: Método para generar pequeñas variaciones en los individuos recombinados.
Aptitud: Método para evaluar la eficiencia de un individuo.
Robótica evolutiva: Es un área de investigación en la que se utilizan algoritmos genéticos para resolver problemas de robótica.
CoppeliaSim: Anteriormente conocido como V-REP, es un simulador de robots utilizado en la industria, la educación y la investigación.
Motor de físicas: Combina la cinemática y dinámica para obtener un óptimo rendimiento de una simulación en CoppeliaSim.

Modelo dinámico: Diseño simplificado de un modelo de robot que interactúa con el motor de físicas.

CAPITULO 1. INTRODUCCION

En la robótica, la gran mayoría de sus aplicaciones son diseñadas para realizar una o varias acciones en concreto. Un claro ejemplo son los brazos robóticos que se utilizan en algunas fábricas para automatizar y mejorar los procesos de producción, donde se espera que el robot desarrolle una tarea simple y repetitiva dentro de un ambiente estructurado que no rebase las especificaciones para las que fue diseñado. Sin embargo, cada vez es más común que se requieran robots más independientes capaces de moverse y desplazarse en terrenos irregulares y con obstáculos para lograr uno o varios objetivos. Es por esto que el aprendizaje en robots es una manera de adaptarse a distintas situaciones y los algoritmos evolutivos son una alternativa para que los robots puedan aprender por sí mismos. Mediante dicho aprendizaje existe, además, la ventaja de entender mejor el comportamiento inteligente existente en la naturaleza. De igual manera, también puede ser interesante el uso de algoritmos evolutivos para superar daños sufridos en un robot. Por ejemplo, si un robot caminante se ve dañado en una de sus patas podría encontrar por sí mismo la mejor manera de desplazarse sin esa pata.

Los algoritmos evolutivos más populares son los algoritmos genéticos que son métodos adaptativos, generalmente usados en problemas de búsqueda y optimización de parámetros, basados en la reproducción sexual y en el principio de supervivencia del más apto [1]. Mas formalmente, y siguiendo la definición dada por Goldberg, “los algoritmos genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas” [2]. Para alcanzar la solución a un problema se parte de un conjunto inicial de individuos, llamado población, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema. Estos individuos evolucionarán tomando como base los esquemas propuestos por Darwin sobre la selección natural, y se adaptarán en mayor medida tras el paso de cada generación a la solución requerida [3].

Dentro del presente capítulo se encontrarán los antecedentes que validan el proyecto de investigación. En el capítulo 2 se explicará el objetivo general de la investigación, así como los distintos aspectos a investigar. En el capítulo 3 se mostrará de forma clara y precisa el planteamiento del problema. En el capítulo 4 se dará a conocer la justificación que valida la investigación. En el capítulo 5 se dará una revisión a la literatura para que el lector tenga un mayor entendimiento acerca del tema de investigación. En el capítulo 6 se dará a conocer el procedimiento para llevar a cabo el proyecto. En el capítulo 7 se realizará un análisis detallado de los resultados obtenidos. En el capítulo 8 se realizará una pequeña discusión acerca de los resultados obtenidos. En el capítulo 9 se darán las conclusiones finales. Y, finalmente, en el capítulo 10 se darán recomendaciones y sugerencias para investigaciones futuras basadas en el presente proyecto.

1.1 Antecedentes

1.1.1 Algoritmos evolutivos

En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en una técnica de búsqueda basada en la teoría de la evolución y que se conoce como el algoritmo genético [4]. Esta técnica basada en los mecanismos de selección que utiliza la naturaleza, de acuerdo a los cuales, los individuos más aptos de una población son los que tienen mayor probabilidad de sobrevivir al adaptarse más fácilmente a los cambios que se producen en su entorno.

Un investigador de la Universidad de Michigan llamado John Holland fue uno de los primeros en simular el proceso evolutivo en una computadora a principios de los 1960s. Su objetivo era lograr que las computadoras aprendieran por sí mismas. La técnica que inventó Holland originalmente se denominó “planes reproductivos y adaptativos”, pero después de la publicación de su libro *Adaptation in Natural and Artificial Systems* [5], se hizo popular bajo el nombre de “algoritmo genético”.

Durante los últimos años se ha buscado mejorar el desempeño de los algoritmos genéticos, se han propuesto nuevas técnicas de representación, selección y cruza, con resultados alentadores. Por ejemplo, el uso de códigos de Gray y la codificación dinámica han superado algunos de los problemas asociados con la representación de valores reales mediante cadenas binarias [4]. También se han propuesto técnicas adaptativas que varían dinámicamente los parámetros de control (porcentajes de mutación y cruza) [6]. Otras

innovaciones notables son los algoritmos genéticos distribuidos y los algoritmos genéticos paralelos [7].

Hasta el momento, los algoritmos genéticos siguen siendo un área abierta de investigación, especialmente para el análisis de problemas que resultan difíciles de resolver. Así mismo, existen todavía diversos problemas teóricos asociados a su comportamiento y efectividad, que no han podido resolverse [8][9].

1.1.2 Robótica evolutiva

Los algoritmos evolutivos se usan en distintas áreas y una de ellas es la robótica, dando pie a la robótica evolutiva, de la cual se considera a George J. Friedman como el pionero. En su tesis de maestría, Friedman [10] propuso evolucionar una serie de circuitos de control similares a lo que hoy se conoce como redes neuronales, usando lo que él denominaba “retroalimentación selectiva”, en un proceso análogo a la selección natural. Friedman logró agrupar estos circuitos simples para formar circuitos más completos de forma automática, utilizando mutaciones aleatorias y un proceso de selección, especulando que las simulaciones del proceso de reproducción sexual –o cruza- y el de mutación nos conducirían al diseño de máquinas inteligentes.

Actualmente, la robótica evolutiva se divide en diferentes ramas de investigación:

1.1.2.1 Biorobótica evolutiva

Consiste en construir un robot que se asemeje a un animal y a continuación se evoluciona un aspecto de dicho robot para investigar de qué manera puede evolucionar ese aspecto del animal. Un ejemplo es el de Long, que ha trabajado con la evolución de la rigidez de colas artificiales de animales nadadores y la forma en que se orientan en el agua en función de la rigidez de su cola [11].

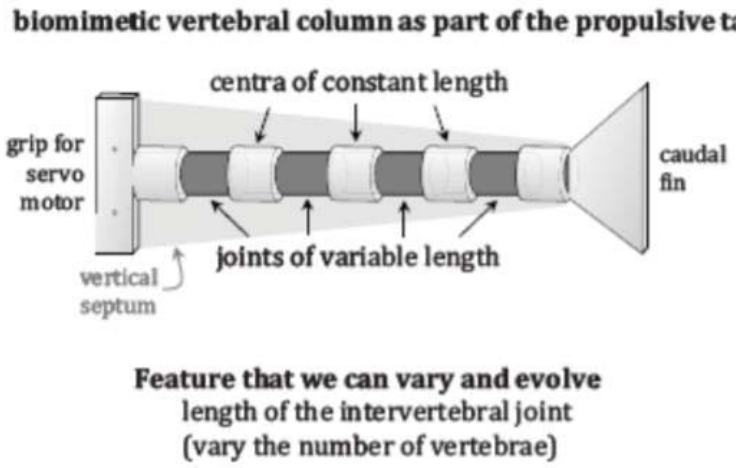

Figura 1.1: Modelo 3 de columna vertebral biomimética de un tiburón [11].

1.1.2.2 Robótica cognitiva

Es la rama de la robótica que se encarga de dotar al robot de inteligencia suficiente como para aprender y razonar cómo comportarse en situaciones complejas. Un ejemplo es el de Miglino quien realizó un proyecto llamado “La robótica como herramienta para la educación” [12], en donde con ayuda de robots físicos implementó los algoritmos genéticos para programarlos como organismos artificiales. Utilizando sencillos kits de ensamblaje, los estudiantes los armaban simulando comportamientos de los animales.

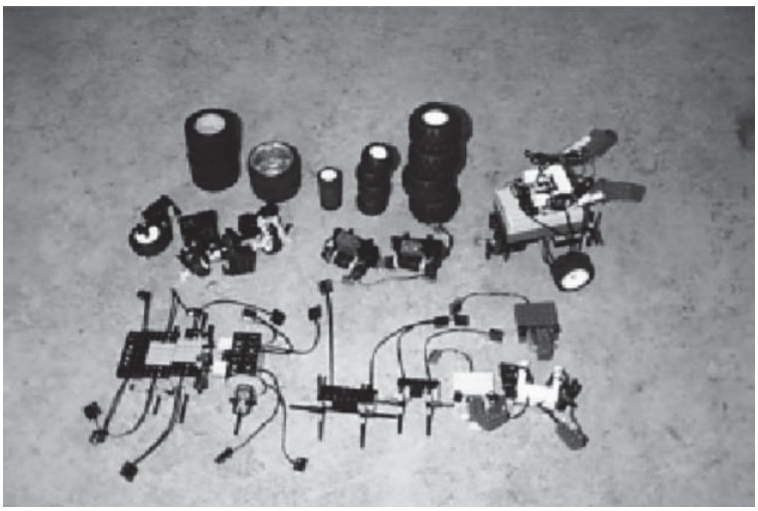

Figura 1.2: El kit LEGO MindStorms [12].

Los niños desarrollaban controladores de robots en el simulador y en cada fase de la evolucion un niño elegia entre diferentes comportamientos del robot mostrados en la pantalla. Cuando estaban satisfechos con el comportamiento del robot simulado, ellos transferian el sistema de control desarrollado a un robot

real LEGO MindStorms, el cual permitía construir un robot con la morfología que los niños elegian.

1.1.2.3 Robótica de enjambres

Este tipo de robótica trabaja con un elevado número de pequeños robots moderadamente simples con el objetivo de trabajar en equipo para conseguir un objetivo en común. Un ejemplo práctico es el de Luke [13] que en un simple entorno depredador / presa buscaba, por medio de enfoques homogéneos y heterogéneos, obtener la interacción de dos algoritmos únicos. Para esto, utilizó distintos métodos de coordinación: ninguno, detección deíctica, en el que solo se detecta si tiene una relación con el agente, por ejemplo, “agente vecino más cercano”, y detección basada en el nombre –detección respecto a la posición del agente basado en su nombre-. Y con ayuda de programación genética realizó 3 estrategias de reproducción: clones, libres y restringidas. Tomando como ejemplo a la gacela y el león, la dificultad para matar a las gacelas se veía directamente afectada por su inteligencia, y la probabilidad de que el león las atrapara se basaba en la evolución del trabajo en equipo de los leones; entre mayor era la cantidad de leones, mejores eran los resultados obtenidos.

1.1.2.4 Robótica modular

Los sistemas robóticos configurables moduladores abordan el diseño, la fabricación, la planificación del movimiento y el control de máquinas cinemáticas autónomas con morfología configurable. Más allá de la actuación, detección y control convencionales que se encuentran típicamente en los robots de morfología fija, los robots auto-reconfigurables también pueden cambiar deliberadamente su propia forma al reorganizar la conectividad de sus partes para adaptarse a nuevas circunstancias, realizar nuevas tareas o recuperarse del daño. Un ejemplo es el de Becerra [14] que realizó un sistema que simulaba el auto-ensamble de un robot modular por medio de la generación de trayectorias, resolviendo su cinemática inversa con algoritmos genéticos.

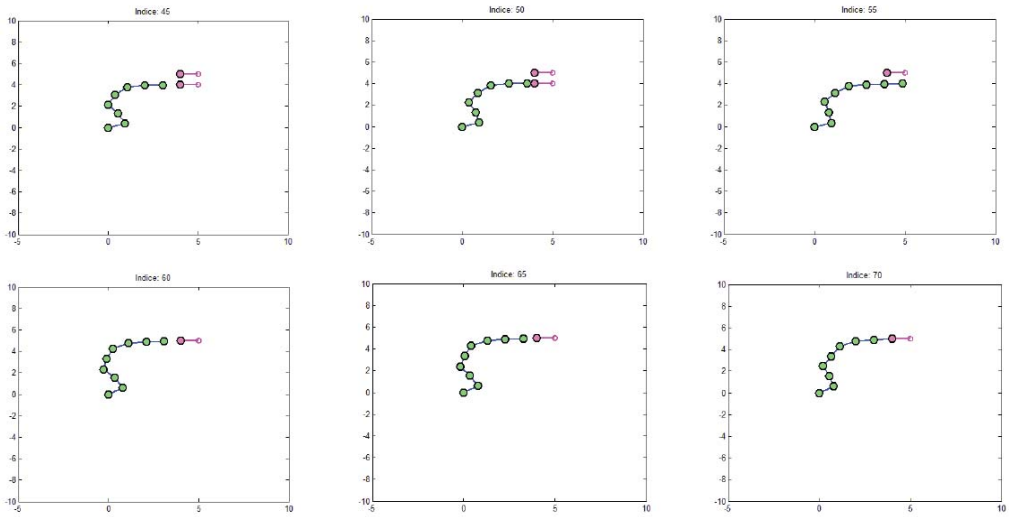

Figura 1.3: Trayectoria articular del auto-ensamble realizada por un algoritmo genético [14].

Actualmente, se encuentran pocas aplicaciones de los robots modulares con algoritmos evolutivos ya que, como menciona Meng en *"Modular Self-Reconfigurale Robot System"* [15], a pesar de las ventajas de los sistemas robóticos auto-reconfigurables modulares, ha sido difícil identificar dominios de aplicación específicos donde se puedan demostrar esos beneficios. Es por ello que, aunque los algoritmos evolutivos son una herramienta capaz para optimizar el comportamiento de un robot modular con una forma prefija, su uso continúa siendo un problema abierto de investigación. Pero se espera que algún día los robots modulares puedan usarse en grandes cantidades para aplicaciones prácticas donde la auto-organización adaptativa y sin supervisión sea posible.

1.1.2.5 Otras aplicaciones

La robótica evolutiva contempla aplicaciones en distintas áreas. Como ejemplo se tienen los trabajos de García [16] y de Jiménez [17] donde se busca que un robot aprenda a caminar utilizando algoritmos evolutivos.

Figura 1.4: Modelo de robot cuadrúpedo que aprende a caminar utilizando algoritmos evolutivos [16].

De igual manera, dentro de los algoritmos evolutivos se encuentran distintos métodos, como lo son los algoritmos de evolución diferencial, implementados en la tesis de maestría de Pantoja [18] para mejorar el diseño del mecanismo de una rodilla para la marcha bípeda.

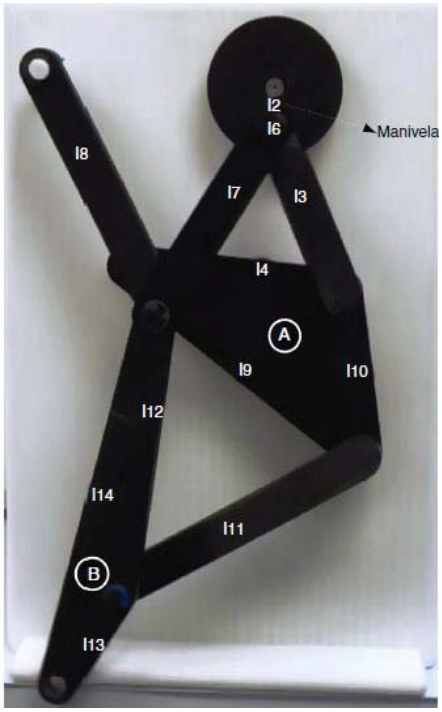

Figura 1.5: Prototipo experimental de mecanismo de marcha, obtenido por algoritmos de evolución diferencial [18].

CAPITULO 2. OBJETIVOS

2.1 Objetivo General

Simular la evolución de un robot cuadrúpedo que aprende a desplazarse tomando como método de aprendizaje los algoritmos genéticos.

2.2 Objetivos específicos

1. Elegir el diseño del robot que se tomará como ejemplo para la aplicación de algoritmos genéticos.

2. Diseñar las piezas del robot en SolidWorks.

3. Ensamblar el diseño del robot.

4. Crear el ambiente de simulación en CoppeliaSim e incorporar el diseño del robot.

5. Realizar la conexión del entorno de simulación con el algoritmo genético por medio de una API.

6. Diseñar el algoritmo genético respecto al diseño del robot que se eligió.

7. Realizar pruebas y mejorar el desempeño del algoritmo genético.

8. Evaluar las mejores soluciones y determinar cuál realiza mejor el desplazamiento del robot.

CAPITULO 3. PLANTEAMIENTO DEL PROBLEMA

El presente trabajo terminal consiste en realizar un sistema capaz de simular los procesos genéticos de los organismos vivos implementados en un robot cuadrúpedo de 3 grados de libertad en cada pata, que aprende a desplazarse sin olvidar las limitaciones mecánicas del diseño. Esto conlleva a desarrollar un algoritmo matemático capaz de interpretar las posibles soluciones como lo hace la naturaleza, midiendo la eficiencia del individuo en la lucha por los recursos [19].

Cada posible solución será asociada como un individuo dentro del sistema, lo que significa que cada individuo se interpretará como una posible solución de avance. Cada uno de ellos tendrá un ajuste de acuerdo a su aptitud con respecto de la solución que representa, lo que significa que, el ajuste va a variar respecto a cuánto se desplaza, sobreviviendo solo aquellos que representen los mejores resultados para el sistema. En otras palabras, sobrevivirán los individuos que presenten el mejor desplazamiento posible.

Cabe mencionar que el robot no tendrá conocimiento alguno de su cinemática, lo que hace aún más interesante el análisis profundo de los resultados que vaya mostrando el algoritmo genético. Una solución aceptable depende directamente del diseño del algoritmo genético que se implemente, ya que cada algoritmo puede llegar a variar de una manera considerable de un problema a otro. Todo el sistema se expondrá por simulación, a fin de facilitar la muestra del algoritmo ya que requiere una cantidad considerable de tiempo para poder visualizar el aprendizaje del sistema.

CAPITULO 4. JUSTIFICACION

Los algoritmos evolutivos tienen muchas aplicaciones. Como se ha mencionado anteriormente, una de ellas es la robótica, la que da pie a la robótica evolutiva, la cual busca crear máquinas que tengan un comportamiento autónomo y adaptativo al mismo tiempo. Es por esto que el presente trabajo terminal busca ser un aporte a la robótica evolutiva en nuestro país. Así mismo, se busca probar métodos distintos a los convencionales, con el objetivo de comprobar la efectividad que tienen los algoritmos genéticos al implementarlos en robots, esperando que los resultados sean un avance hacia mejores sistemas de aprendizaje dentro de la industria robótica. Adicionalmente, con los conocimientos obtenidos en investigaciones previas se muestra que la robótica evolutiva es una creciente área de investigación con muchas aplicaciones futuras, por lo que se espera que al finalizar el proyecto se logre desarrollar una aportación al área, la cual permita desarrollos más sofisticados en el futuro.

CAPITULO 5. MARCO TEORICO

5.1 Neo-Darwinismo

Los algoritmos evolutivos no solo engloban la teoría de la evolución propuesta por Charles Darwin [3] sino también el seleccionismo de August Weismann y la genética de Gregor Mendel creando de esta manera el paradigma Neo-Darwiniano [20]. Este paradigma establece que la historia de la vasta mayoría de la vida de nuestro planeta puede ser explicada a través de un puñado de procesos estadísticos que actúan sobre y dentro de las poblaciones y especies: la reproducción, la mutación, la competencia y la selección [20]. La reproducción es una propiedad obvia de todas las formas de vida de nuestro planeta, pues de no contar con un mecanismo de este tipo, la vida misma no existiría. La mutación es un error de copiado del ADN de las especies, el cual es aleatorio. De igual manera, al contarse con una cantidad finita de espacio en la Tierra, garantiza la existencia de la competencia. La selección es una consecuencia natural de la competencia. Por lo tanto, la evolución es el resultado de procesos probabilísticos, que interactúan entre sí, en las poblaciones, de generación en generación.

5.2 Computación evolutiva

La computación evolutiva o algoritmos evolutivos se refiere al estudio de los fundamentos y aplicaciones de ciertas técnicas heurísticas basadas en los principios de la teoría Neo-Darwiniana de la evolución natural. Los algoritmos evolutivos permiten abordar problemas complejos de la ciencia y la ingeniería que involucran perturbaciones caóticas, alta no linealidad y espacios de búsqueda muy accidentados. Los algoritmos tradicionales, en general, no pueden resolver este tipo de problemas [1].

En términos generales, para simular un proceso evolutivo en una computadora se requiere lo siguiente [20]:

- Codificar las estructuras que se replican, (estas estructuras son nuestros individuos).
- Operaciones que afecten a los “individuos” (los operadores genéticos).
- Una función de aptitud, (indica qué tan bueno es un individuo con respecto a los demás).
- Un mecanismo de selección.

Los algoritmos evolutivos se clasifican en 3 distintas categorías, las cuales, aunque inicialmente se originaron de manera independiente, ahora juntas componen la computación evolutiva:

- Algoritmos genéticos.
- Estrategias evolutivas.
- Programación evolutiva.

El trabajo terminal como se menciona en el título del proyecto se lleva a cabo empleando los algoritmos genéticos, por lo que, se indagará a profundidad únicamente en esa categoría. Para conocer de manera introductoria las otras 2 categorías se recomienda consultar *Introducción a los algoritmos genéticos y la programación genética* [19].

5.3 Algoritmo genético

5.3.1 Bases biológicas

El ácido desoxirribonucleico (ADN) es una macro-molécula doblemente trenzada que tiene una estructura helicoidal, que contiene el material genético fundamental de todos los organismos vivos. Dentro del ADN se encuentran los genes que son la unidad de herencia. A la colección total de genes se le denomina genoma y el genotipo es la composición genética del organismo –la información contenida en el genoma- (ver Figura 5.1) [20]. Los cromosomas se encuentran en el núcleo de las células y son responsables de la transmisión de información genética. Cada gen es capaz de ocupar solo una región en particular de un cromosoma. En algunas ocasiones se pueden presentar formas alternativas del gen, a los cuales se les llama alelos.

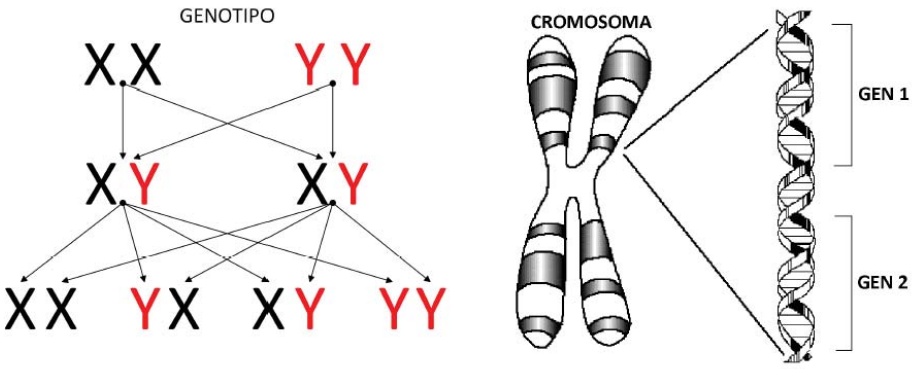

Figura 5.1: Genotipo, cromosoma y genes [16].

La apariencia visible y la funcionalidad de un individuo están contenidas en el genotipo y se conoce como fenotipo. En los fenotipos se producen ocasionalmente alteraciones que se conocen como mutaciones. Las mutaciones son variaciones de uno o más genes y tienen un carácter aleatorio [16]. Si la mutación producida da muestras de tener buenas características, permanecerá en las siguientes generaciones del proceso evolutivo, a través de la selección del individuo. De lo contrario, si la mutación no da muestras de tener buenas características, desaparecerá en las próximas generaciones. La aptitud de un individuo se define como la probabilidad de que éste viva para reproducirse (viabilidad), o como una función del número de descendientes que éste tiene (fertilidad). La selección es el proceso mediante el cual algunos individuos en una población son seleccionados para reproducirse, basados en su aptitud [20]. Se denomina reproducción al proceso de crear a un nuevo individuo, ya sea de manera sexual o asexual.

5.3.2 Conceptos de los algoritmos genéticos

Un cromosoma en los algoritmos genéticos es una estructura de datos que contiene una cadena de parámetros o genes (ver Figura 5.2).

1	0	0	1	1	0	0	0	1

Figura 5.2: Ejemplo de una cadena cromosómica.

Un gen es una subsección de un cromosoma, que generalmente codifica un solo parámetro y un solo parámetro puede tener asociado varios bits (ver Figura 5.3).

1	0	0	1	1	0	0	0	1

Figura 5.3: Ejemplo de un gen con 3 bits.

El genotipo es la codificación de los parámetros que representan una solución del problema que se busca resolver. La codificación puede ser binaria, de números reales, entera, códigos de gray, etc. Un ejemplo es el de la Figura 5.2 en el que se muestra una codificación binaria. Un alelo es un bit del cromosoma. En una codificación binaria, un alelo es un 1 o un 0.

El fenotipo es la decodificación del cromosoma, que se utiliza para obtener su valor respecto a la función aptitud. Tomando como ejemplo la Figura 5.3, tras decodificar el gen de 3 bits, nos da un valor de 5 respecto a una función aptitud creada.

Un individuo es un miembro de una población que representa una de las posibles soluciones al problema. Cada individuo está representado por medio de una cadena cromosómica como se puede visualizar en la Figura 5.2.

La aptitud de un individuo es el valor qué indica que tan buena es una solución para un problema dado, respecto al resto de la población.

Una generación es la creación de una nueva población por medio de operadores de reproducción. En los algoritmos genéticos la reproducción, al igual que en los seres vivos, se realiza al pasar la información de padres a hijos y para lograr esto se utilizan 3 operadores fundamentales [20]:

- Cruza
- Mutación
- Reordenamiento

La cruza es el operador que forma un nuevo cromosoma, combinando partes de sus padres. La mutación es el operador que realiza pequeñas alteraciones a los nuevos cromosomas, modificando los genes de uno o de ambos padres. El operador de reordenamiento cambia de posición los genes de un cromosoma, esperando facilitar la producción de bloques constructores. Los bloques constructores son los conjuntos de genes que generan mejores resultados en la función de aptitud y, a su vez, mejores individuos. Existe una técnica llamada elitismo en la que el individuo más apto no se cruza con nadie, hasta que surge otro individuo mejor que él, que lo desplace. Esto asegura que la aptitud máxima de la población nunca se reducirá de una generación a otra.

5.3.3 Estructura del algoritmo

El funcionamiento general de un algoritmo genético se muestra a continuación en la Figura 5.4.

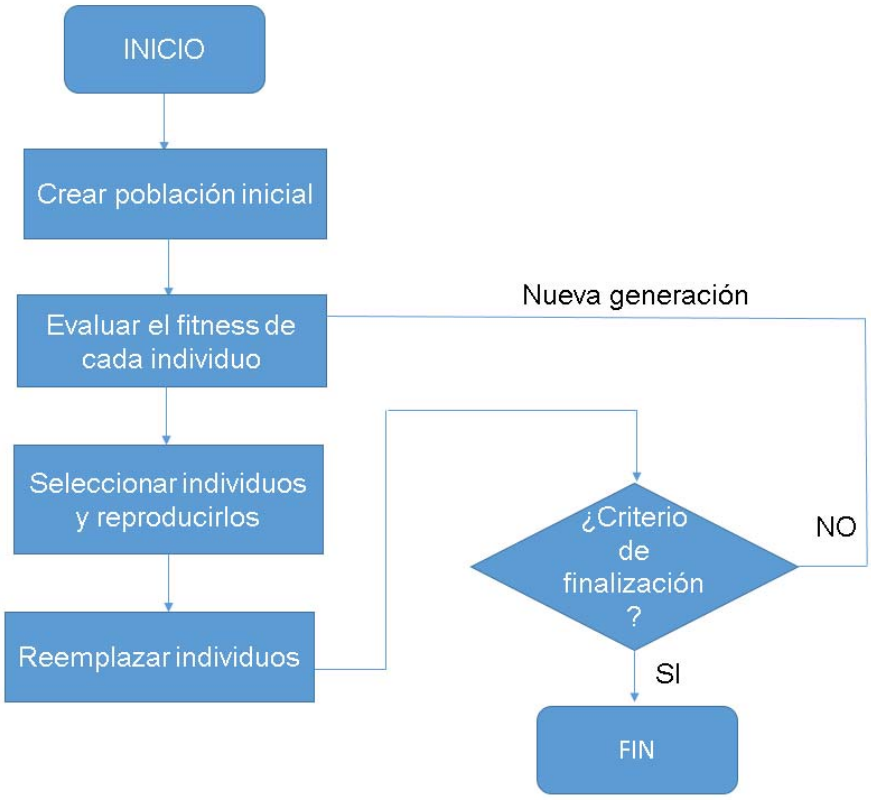

Figura 5.4: Diagrama de flujo del funcionamiento básico de un algoritmo genético.

Los criterios de paro más usuales de un algoritmo genético son:

- Los mejores individuos de la población representan soluciones suficientemente buenas para resolver el problema en cuestión.
- La población ha convergido. Un gen ha convergido cuando el 95% de la población tiene el mismo valor para él. Cuando esto ocurre, la media de la aptitud de la población se aproxima a la aptitud del mejor individuo [19].
- Se alcanza el número de generaciones máximo especificado.

En el caso de haber elitismo, el cromosoma pasa a la siguiente generación sin ser alterado por ningún operador genético.

A continuación, comienza a generarse la nueva población con base en la aplicación de los operadores genéticos de cruza y/o clonación.

- Cruza: Reproducción de tipo sexual.
- Clonación: Reproducción de tipo asexual.

El porcentaje de cruza se delimita por medio de una probabilidad Pc, la cual generalmente tiene un valor de entre 60% a 90%. Cuando no se efectúa la cruza los padres se clonan directamente a la siguiente generación.

Una vez generados los nuevos individuos se realiza la mutación con una probabilidad Pm, la cual por lo general oscila entre el 1% y el 10%. La cantidad de individuos se delimita desde un inicio, por lo que, para ubicar a los descendientes generados, previamente se han de eliminar otros individuos de la población. Para esto existen métodos para saber qué individuos es más conveniente eliminar.

5.3.4 Operadores genéticos

Cada uno de los operadores genéticos que integran a un algoritmo genético tienen técnicas que permiten tener un mejor funcionamiento del mismo. A continuación, se mostrarán las técnicas que se utilizan específicamente para codificación binaria, ya que, aunque existen otros tipos de codificaciones, el presente proyecto se realizó con números binarios.

5.3.4.1 Selección

Los algoritmos de selección se encargan de elegir qué individuos disponen de oportunidades de reproducirse y cuáles no. Al igual que en la naturaleza, se le otorga un mayor número de oportunidades de reproducción a los individuos más aptos. En el algoritmo genético este proceso de selección suele realizarse de forma probabilística, es decir, aun los individuos menos aptos tienen una cierta oportunidad de sobrevivir, ya que, de lo contrario en pocas generaciones la población se volvería homogénea.

También existen técnicas de selección determinísticas que pueden evitar problemas como el de predominancia de ciertos individuos o bien que se tienda a repartir la búsqueda por un mayor espacio, sin dejar de buscar en la mejor zona [19]. En el presente proyecto se discutirán únicamente las técnicas de selección más frecuentemente utilizadas.

- **Selección por ruleta**

Es un método de selección proporcional muy simple y fácil de implementar que fue propuesto por DeJong [21]. La idea consiste en asignar a cada individuo de la población una porción de una ruleta. El tamaño de la porción asignada a cada individuo será una consecuencia directa de su función aptitud, de tal forma que la suma de todos los porcentajes será la unidad. Por lo tanto, la asignación de una porción más grande será para los individuos más aptos y los menos aptos, tendrán una porción de ruleta más pequeña.

Para seleccionar un individuo, basta con generar un número aleatorio en el intervalo entre, 0 y 1, e ir sumando las proporciones de cada individuo hasta que se sobrepase el número aleatorio generado. De esta manera, se determina el individuo escogido por la ruleta.

El algoritmo de la ruleta es el siguiente [20]:

- Calcular la suma de valores esperados T.
- Repetir N veces (N es el tamaño de la población):
    - Generar un número aleatorio r entre 0.0 y T.
    - Ciclar a través de los individuos de la población sumando los valores esperados hasta que la suma sea mayor o igual a r.
    - El individuo que haga que esa suma exceda el límite es el seleccionado.

El valor esperado se calcula de la siguiente manera:

$$V_{esp} = f\_i \,/\, avg\_f$$

Donde:

f_i = Es la aptitud de un individuo.
avg_f = Es el promedio de aptitudes de toda la población.

El método de selección por ruleta, presenta el problema de que a medida que aumenta el tamaño de la población se vuelve más ineficiente. Además, presenta el inconveniente de que el peor individuo puede ser seleccionado más de una vez [19].

- **Selección por torneo**

Existen dos versiones de selección mediante torneo:

- Torneo determinístico
- Torneo probabilístico

En el torneo determinístico se selecciona al azar “p” individuos (generalmente p=2), se comparan con base en su aptitud y el ganador del “torneo” es el individuo más apto el cual pasa a la siguiente generación. Debe barajarse la población un total de “p” veces para seleccionar “N” padres, donde N es el tamaño de la población.

La versión probabilística, únicamente difiere en el paso de selección del ganador del torneo. En lugar de escoger siempre al individuo más apto, se genera un número aleatorio en el intervalo de 0 y 1. Si es mayor que un parámetro “p” -el cual es fijo en todo el proceso evolutivo-, se escoge el individuo más apto; en caso contrario, se elige al menos apto. Usualmente, “p” toma valores en el rango $0.5 < p <= 1$ [19].

A lo largo del proceso evolutivo se pueden variar los individuos que participan en cada torneo y esto a su vez modifica la presión de selección. Esto significa que, si la presión de selección es elevada, los peores individuos apenas tendrán oportunidad de reproducción, y en caso contrario, cuando la cantidad de participantes en el torneo disminuye, los peores individuos tendrán más oportunidades de ser seleccionados.

5.3.4.2 Cruza

Una vez seleccionados los individuos, éstos son recombinados para reproducirse y crear de esta manera la siguiente generación. La cruza es una reproducción del tipo sexual, con una probabilidad que ronda el 90% en la mayoría de los casos.

La idea principal de la cruza es crear una descendencia con la posibilidad de que los genes heredados sean precisamente los causantes de la aptitud de los padres, o en un mejor caso, obtener una aptitud mayor a la que cada uno de los padres por separado podrían obtener.

Para el presente proyecto de investigación se consideran las tres técnicas básicas de cruza descritas a continuación:

- **Cruza de un punto**

Es la técnica más sencilla de cruza, propuesta por Holland [5]. Selecciona a dos individuos, a los cuales se les cortan sus cromosomas por un punto seleccionado aleatoriamente, con el cuidado de no seleccionar un borde del cromosoma, porque de lo contrario no se produciría ninguna división del mismo. Se generan dos segmentos, se intercambian un segmento cada uno, para finalmente tener dos nuevos descendientes (ver Figura 5.5). De esta manera, ambos descendientes heredan información genética de los padres.

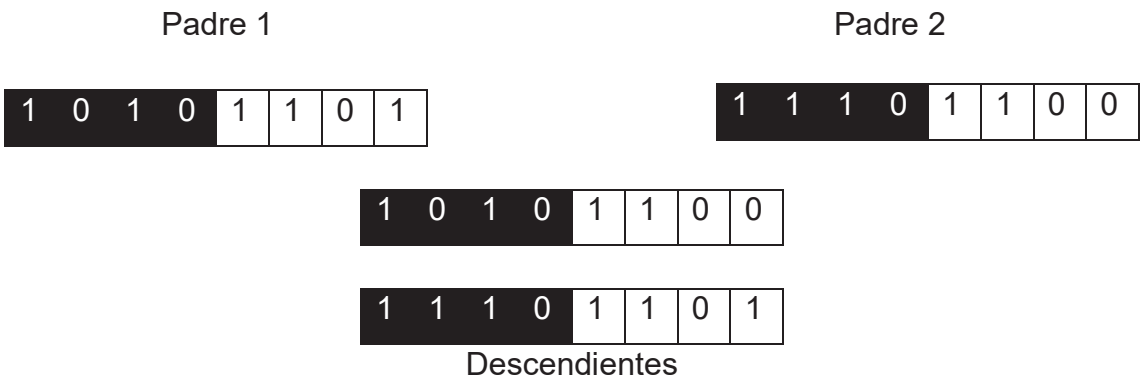

Figura 5.5: Cruza de un punto.

- **Cruza de dos puntos**

DeJong [21] fue el primero en implementar una cruza de n puntos, como una generalización de la cruza de un punto. Concluyendo que el valor más eficiente es n=2. En este caso, se realizan dos cortes que generan tres segmentos, con el debido cuidado de que el corte no coincida con el extremo de los cromosomas. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre (ver Figura 5.6).

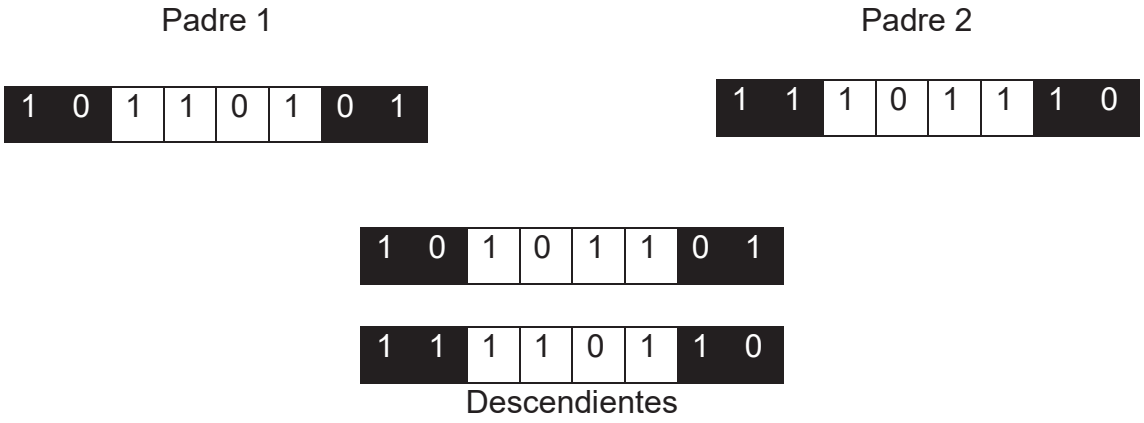

Figura 5.6: Cruza de dos puntos.

- **Cruza uniforme**

En esta técnica se hace uso de una máscara de cruce con valores binarios. La máscara de cruce puede no permanecer fija durante todo el proceso evolutivo. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si, por el contrario, hay un 0, el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres (ver Figura 5.7). El número efectivo de puntos de cruce es L/2 [19], siendo L la longitud del cromosoma.

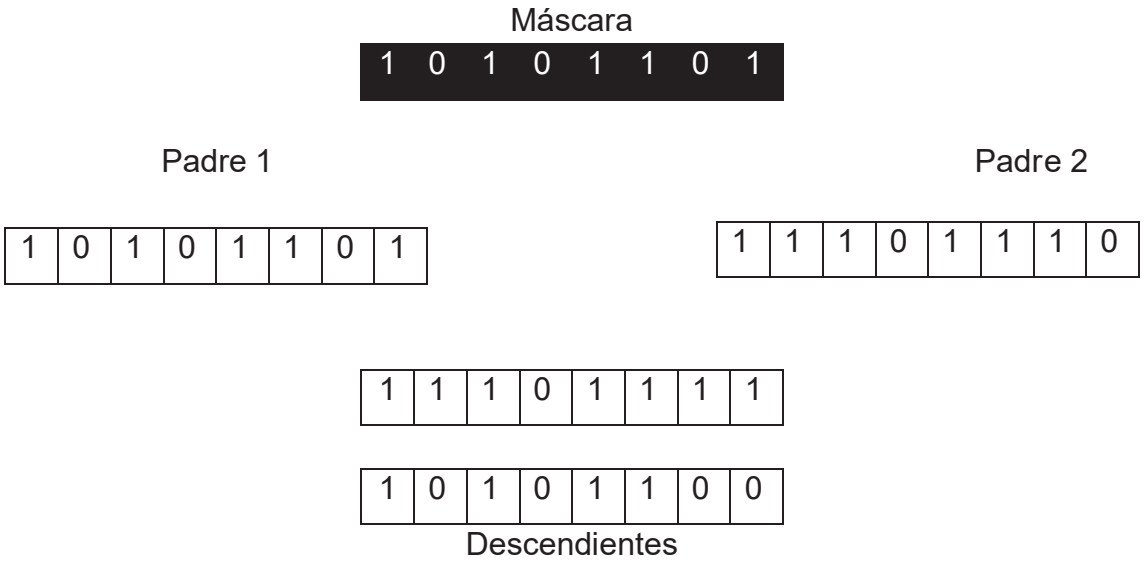

Figura 5.7: Cruza uniforme.

5.3.4.3 Clonación

A diferencia de la cruza, se trata de una estrategia de reproducción asexual, que consiste en copiar sin cambios un individuo a la nueva generación. El porcentaje de clones en un algoritmo genético es reducido, esto para evitar el riesgo de una convergencia prematura de la población hacia ese individuo.

5.3.4.4 Elitismo

Es una variante del operador de clonación que consiste en copiar únicamente al mejor o los mejores individuos de una generación a la siguiente. Esto garantiza que la aptitud máxima no disminuya durante las generaciones. Así mismo, el elitismo debe usarse en algoritmos genéticos a fin de garantizar convergencia [22].

5.3.4.5 Mutación

Ya que se han seleccionan dos individuos de la población para realizar la cruza, uno de los descendientes o ambos, se mutan con una probabilidad Pm, la cual generalmente suele ser baja, menor a 10%. En la práctica se suelen recomendar porcentajes de mutación de entre 0.01 y 0.1 para la representación binaria [20]. Esto se debe a que la varianza de las aptitudes de la población suele reducirse al producirse una mutación. Sin embargo, se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.

La mutación consiste en la modificación aleatoria de uno o más genes del cromosoma del individuo. En el caso de trabajar con codificaciones binarias consiste en simplemente negar un bit (ver Figura 5.8).

La mutación en el algoritmo genético busca imitar de esta manera el comportamiento que se da en la naturaleza, pues cuando se genera la descendencia siempre se produce algún tipo de error, por lo general sin mayor trascendencia, en el paso de la carga genética de padres a hijos [19].

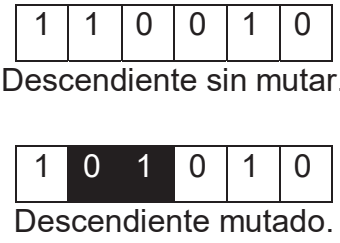

Figura 5.8: Ejemplo de mutación.

5.3.4.6 Función de aptitud

La función de aptitud refleja la adecuación de un individuo al problema. Se le asigna un valor numérico a dicha aptitud, de acuerdo a una función definida por el usuario [16]. Si se piensa en lo que ocurre en la naturaleza, podría decirse que el ajuste representa las posibilidades de que un individuo llegue a la edad adulta y se reproduzca. Esas posibilidades serán en función de la adecuación de dicho individuo al medio en cuanto a lucha por los recursos se refiere. Cabe mencionar que para cada problema la función de aptitud es distinta, ya que debe de adaptarse al problema y fungir como un mecanismo que refleja qué tan buena es una solución de la población.

CAPITULO 6. DESARROLLO EXPERIMENTAL

6.1 Elección del diseño

Para poder implementar el algoritmo genético, se eligió al robot de la tesis “Robot Cuadrúpedo Inspirado en la Marcha de un Can” [23], cuyo modelo que muestra en la Figura 6.1. El modelo presenta tres grados de libertad en cada extremidad, en donde cada articulación es representada por un servomotor, habiendo 12 articulaciones en total. El robot genera un movimiento similar al de un can. Por lo tanto, el algoritmo genético además de que se realizó con el fin de que aprendiera a desplazarse, se buscó que evolucionara a un movimiento similar.

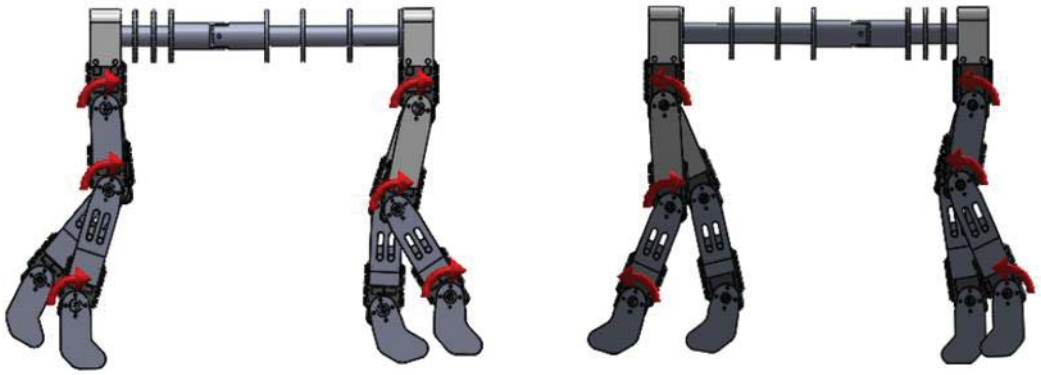

Figura 6.1: Modelo de robot cuadrúpedo de la tesis “Robot Cuadrúpedo Inspirado en la Marcha de un Can” [23]

6.2 Modelado del diseño

El modelo del robot cuadrúpedo se diseñó en SolidWorks. El diseño de las piezas y sus dimensiones pueden encontrarse en la tesis del modelo elegido [23]. Cabe mencionar que a algunas piezas se les hicieron algunos pequeños cambios respecto a los modelos originales por falta de cotas, y los servomotores se obtuvieron en una plataforma llamada GrabCAD [24]. En la Figura 6.2 se muestra una vista isométrica del robot ensamblado.

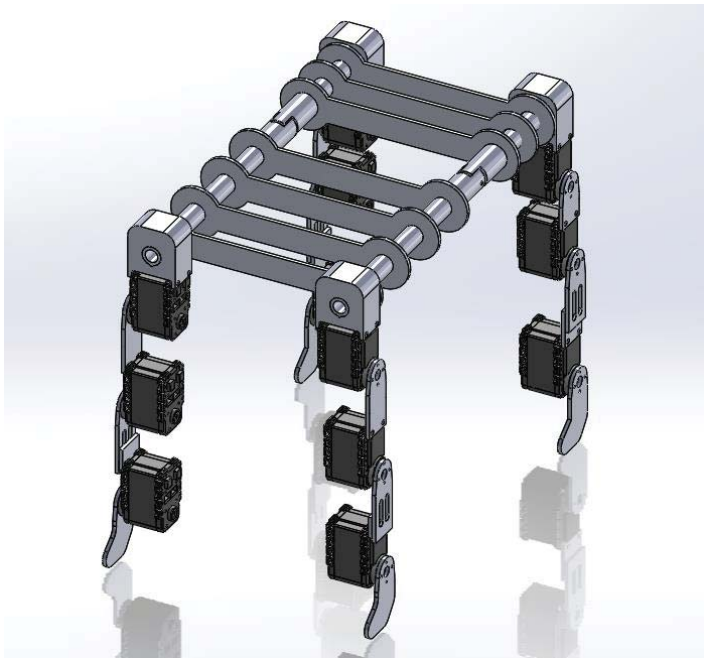

Figura 6.2: Vista isométrica del robot cuadrúpedo ensamblado.

6.3 Migración del diseño a CoppeliaSim

CoppeliaSim utiliza mallas triangulares para describir y mostrar formas. Es por esto, que CoppeliaSim solo importa formatos que describan objetos como mallas triangulares.

Desde SolidWorks es posible convertir el robot ensamblado a mallas triangulares por medio de dos métodos:

1. Convertir el modelo a formato ".URDF": Este formato permite importar todo el modelo ya ensamblado con una posición y orientación preestablecida, es decir, tal como se muestra el modelo del robot cuadrúpedo en la Figura 6.2. Para poder utilizar este formato es necesario descargarlo y agregarlo a SolidWorks. El archivo de descarga

y las instrucciones pueden encontrarse en *SolidWorks to URDF Exporter* [25].

2. Convertir el modelo a formato ".STL": Es un formato de archivo 3D, el cual permite importar el modelo completo al simulador como en el caso anterior, pero va a requerir posicionar, orientar y en algunos casos, escalar el modelo.

Primero se convirtió el modelo a formato ".URDF", pero no funcionó, a pesar de que se probaron distintas versiones de SolidWorks. Es por ello que se utilizó el segundo método el cual funcionó correctamente. Para poder importar el modelo a CoppeliaSim se debe de seguir la siguiente ruta:

File > Import > Mesh

A continuación, se abrirá una pantalla que permitirá escalar y orientar al robot. La Figura 6.3 muestra al modelo del robot cuadrúpedo en el simulador.

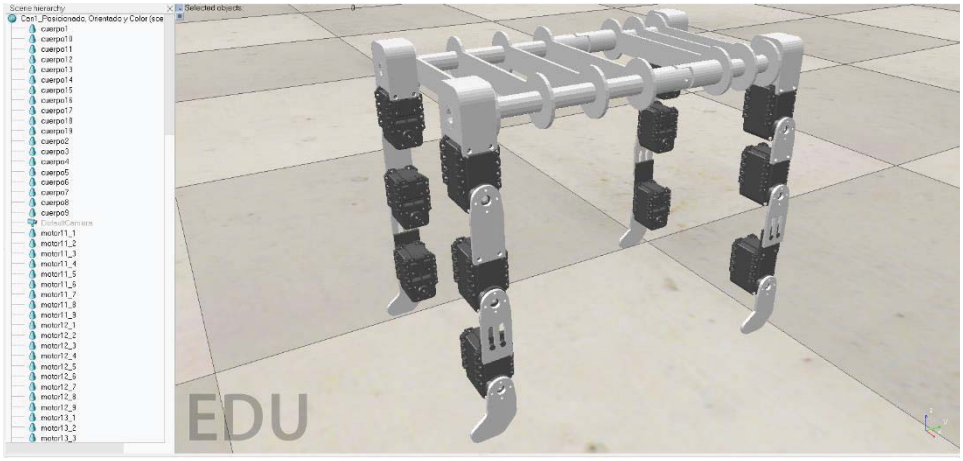

Figura 6.3: Robot orientado, posicionado y escalado.

CoppeliaSim posiciona automáticamente al robot en el simulador respecto a la orientación y escala seleccionada, siendo posible modificarlas posteriormente. La Figura 6.4 muestra los iconos que representan a la posición y orientación respectivamente. Estos pueden encontrarse en el menú principal del simulador.

Figura 6.4: Iconos para posicionar y orientar al robot.

Al importar el diseño del robot cuadrúpedo a CoppeliaSim, se realizaron los siguientes pasos:

- Agrupar las piezas.
- Crear articulaciones.
- Diseñar un modelo dinámico del robot.
- Crear jerarquía.

Solo hasta después de realizar los pasos mencionados, CoppeliaSim reconocerá al diseño como un robot.

6.3.1. Agrupar las piezas

La Figura 6.3 muestra una lista con todas las piezas que contiene el robot; es necesario realizar agrupaciones de las piezas para que puedan moverse en conjunto. La Figura 6.5 muestra las piezas agrupadas adecuadamente.

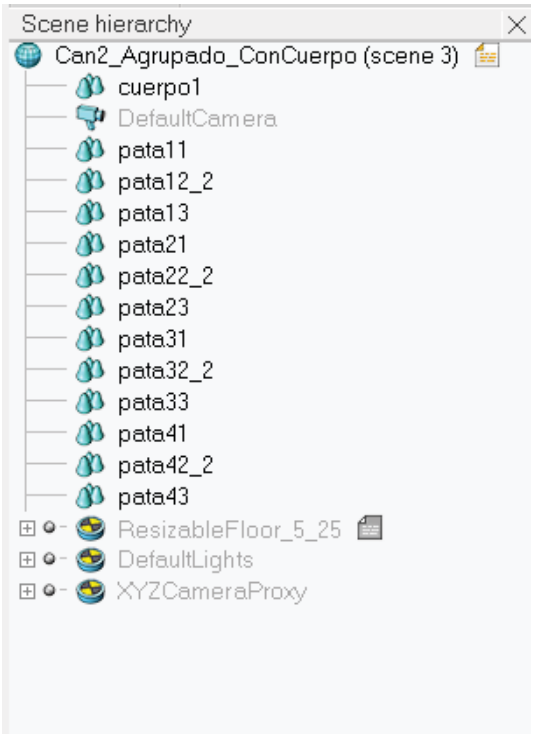

Figura 6.5: Robot agrupado.

Para facilitar el reconocimiento de cada conjunto de piezas, se nombran a las extremidades delanteras pata1 y pata2, a las extremidades traseras pata3 y pata4, y a la base del robot cuerpo1.

La agrupación se realizó respecto al número de pata y de servomotor. Por ejemplo, la pata11 representa la pata1 y el servomotor 1, siendo el servomotor 1 el que está más cercano a la base del robot y el servomotor 3 el más lejano a la base del robot.

6.3.2. Crear articulaciones

Las articulaciones permiten que el robot realice los movimientos efectuados por cada agrupación de piezas. Es por ello que se agregó una articulación a cada servomotor como se muestra en la Figura 6.6a.

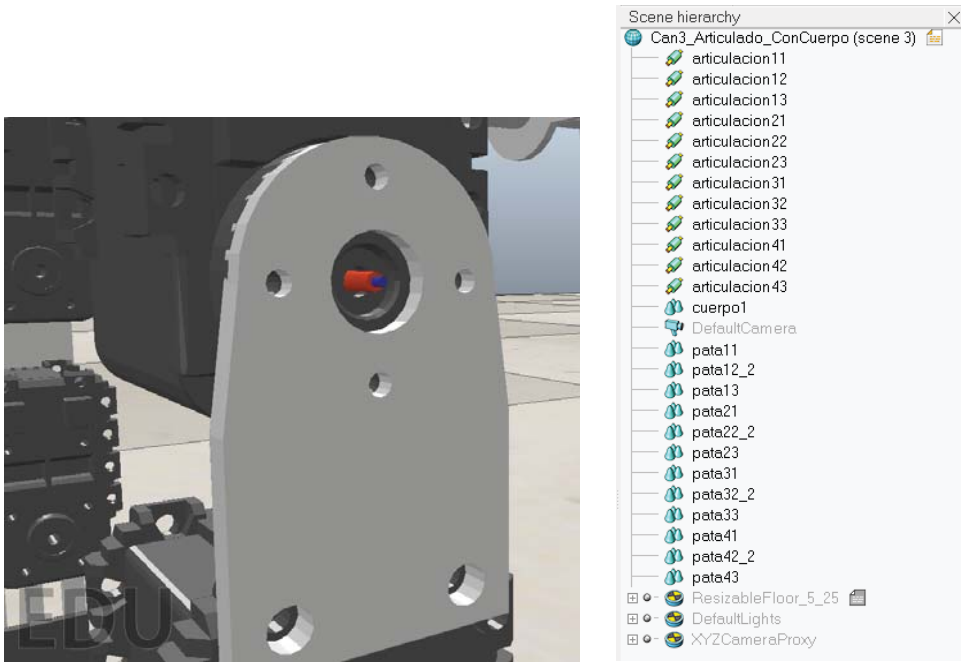

Figura 6.6: a) Posición de una articulación en el modelo del robot cuadrúpedo, b) Articulaciones agregadas al diseño del robot cuadrúpedo.

La articulación 11 que se muestra en la Figura 6.6b representa la pata 1 unida a la articulación 1, siendo la articulación 1 la que está en el servomotor más cercano a la base del robot y la articulación 3 la que está en el servomotor más lejano a la base del robot.

6.3.3. Diseñar modelo dinámico del robot

CoppeliaSim cuenta con un motor de físicas, el cual permite que el robot realice movimientos análogos a los que realizaría si éste fuera construido físicamente, considerando la cinemática y dinámica de cada movimiento. Es por ello que, para que CoppeliaSim pueda generar los movimientos del modelo del robot cuadrúpedo elegido, se debe diseñar el mismo robot con formas más simples ya sean círculos, cuadrados o esferas. También es posible diseñarlo por medio de formas convexas, las cuales generan formas parecidas a las piezas originales como es el caso de las patas en color verde de la Figura 6.7.

Realizar el diseño con formas más simples reduce el consumo de memoria del simulador, ya que como se mencionó anteriormente, CoppeliaSim utiliza

mallas triangulares para describir y mostrar formas, y el modelo original requiere gran cantidad de mallas por pieza, lo que se traduce en una mayor cantidad de memoria del simulador y esto en consecuencia, ralentiza en gran medida a la simulación. El modelo dinámico del robot cuadrúpedo se muestra en la Figura 6.7.

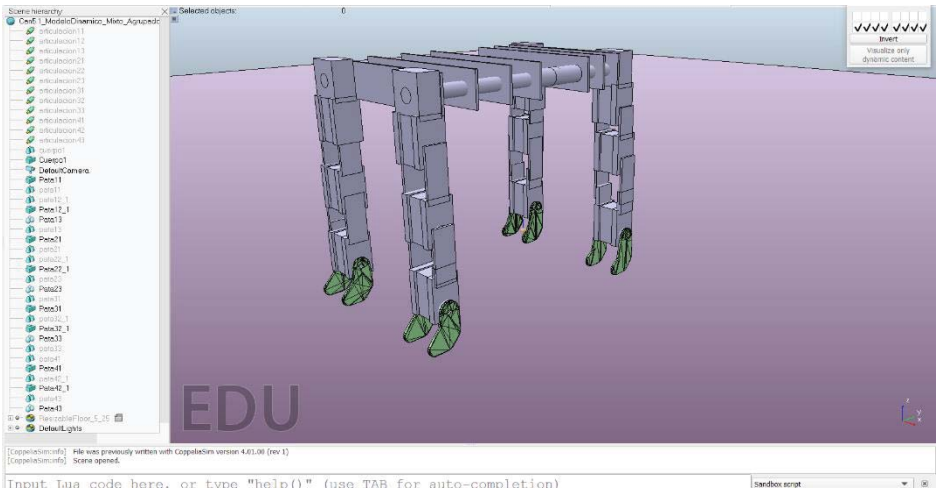

Figura 6.7: Modelo dinámico del robot cuadrúpedo.

Cabe mencionar que, a pesar de que las formas convexas demandan más memoria, éstas se utilizaron debido a que mejoran la interacción que existe entre las patas y el piso, ya que son más similares al modelo original. De igual manera, hay algunas configuraciones que deben llevarse a cabo. CoppeliaSim cuenta con un tutorial que explica a detalle las propiedades con las que debe cumplir el modelo dinámico y cómo realizarlas [26].

El modelo de la Figura 6.7 se mantuvo oculto a lo largo de las pruebas, el motor de físicas lo utilizó para realizar los cálculos físicos de los movimientos, para que a su vez pudieran visualizarse en el modelo original del robot.

6.3.4. Crear jerarquía

Se debe realizar una jerarquía de la agrupación de las piezas, esto con el fin de que el simulador reconozca el orden y el nivel de importancia de cada pieza. La jerarquía diseñada se muestra en la Figura 6.8. La agrupación llamada “Cuerpo1” es la base del robot y la primera articulación de cada pata es llamada “padre”. Las articulaciones siguientes se llaman “hijos”. Cada que el padre generé un movimiento, los hijos lo realizarán de igual manera. Analogamente, si un hijo realiza un movimiento, los hijos de menor jerarquía seguirán su movimiento.

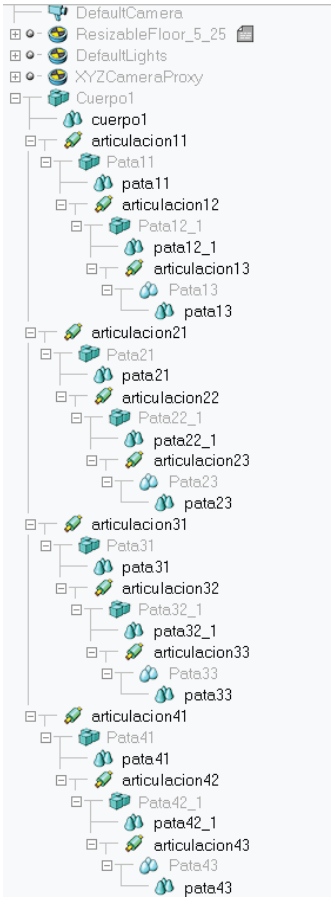

Figura 6.8: Jerarquia del robot cuadrúpedo.

CoppeliaSim cuenta con un extenso tutorial donde explica a detalle todos los puntos que son necesarios para generar un modelo adecuado en el simulador [27].

Finalmente, para tener listo el modelo del robot cuadrúpedo en el simulador, se agregaron las masas y los torques del robot de la tesis de la que se tomó el modelo [23], esto con el fin de ser fiel al modelo original en medida de lo posible. La Tabla 6.1 muestra una comparación de las masas del modelo original respecto a las del modelo en el simulador.

Tabla 6.1 Masas del robot cuadrúpedo.

	Masa (Kg)	
Eslabón	**Modelo original**	**Modelo en simulación**
Pata11	0.00324	0.004361
Pata12_2	0.0098475	0.007144
Pata13	0.003861	0.01020
Pata21	0.00324	0.004361
Pata22_2	0.0098475	0.007144
Pata23	0.003861	0.01020
Pata31	0.00252	0.003055
Pata32_2	0.010238	0.00909
Pata33	0.0044064	0.01129
Pata41	0.00252	0.003055
Pata42_2	0.010238	0.00909
Pata43	0.0044064	0.01129
Cuerpo1	0.08238618	0.133
Servomotor	0.0546	0.02774

	Masa Total (Kg)
Modelo original	0.011406606
Modelo en simulación	0.0137253

Los nombres de las piezas que se muestran en la Tabla 6.1 corresponden a los nombres del modelo dinámico del robot en el simulador.

Las masas del modelo original que se muestran en la Tabla 6.1 no consideran a las baterías ni a los actuadores.

La Tabla 6.2 muestra una comparación de los torques del modelo original respecto a los del modelo en el simulador.

Tabla 6.2 Torques del robot cuadrúpedo.

Torques (Nm)	**Modelo original**	**Modelo en simulación**
articulacion11	0.287	0.630
articulacion12	0.764	0.630
articulacion13	0.472	0.650
articulacion21	0.287	0.630
articulacion22	0.764	0.630
articulacion23	0.472	0.650
articulacion31	0.939	0.730
articulacion32	1.01	0.730
articulacion33	0.508	0.750
articulacion41	0.939	0.730
articulacion42	1.01	0.730
articulacion43	0.508	0.750

Los torques fueron modificados a partir de gran variedad de pruebas realizadas, encontrándose que los más apropiados para el ambiente de simulación son los presentados en la Tabla 6.2.

6.4 Conexión de CoppeliaSim con Python

Inicialmente se había considerado realizar la conexión por medio de ROS -del inglés ROS: Robot Operating System-, el cual permite en términos generales, realizar una conexión entre diversos lenguajes de programación con el fin de crear software para robots. Sin embargo, pero al tener más información de las funcionalidades del simulador CoppeliaSim se tuvo conocimiento del concepto API –del inglés API: Application Programming Interface-, la cual es una interfaz de programación de aplicaciones que permite realizar conexiones entre distintos ambientes de programación entre los que se encuentra Python. En otras palabras, es un conjunto de rutinas que permiten el acceso a distintas funciones de un determinado lenguaje de programación. Es por ello que se optó por realizar la conexión por medio de una API, la cual facilita el proceso de conexión debido a que pertenece al simulador que se está utilizando.

Para poder realizar la conexión, se estudió *Remote API Functions* [28] en la que se enlistan todas las funciones que se pueden utilizar para ejecutar CoppeliaSim desde Python y a su vez, se describe el procedimiento que se requiere para llevar a cabo cada función adecuadamente.

Para habilitar la API remota en el lado del servidor, es decir, desde CoppeliaSim hay dos formas diferentes:

- Servicio de servidor API remoto continuo: Con este método, las funciones de la API remota se ejecutan siempre en el lado del servidor, incluso si la simulación no se está ejecutando.
- Servicio de servidor API remoto temporal: El usuario tiene el control de cuando se inicia o se detiene el servicio.

En las pruebas iniciales se consideró realizar un servicio de servidor API remoto temporal por la simpleza y facilidad de implementación, pero debido a que el algoritmo genético requirió de gran cantidad de tiempo de ejecución, fue necesario configurar un servicio de servidor API remoto continuo, esto debido a su estabilidad y facilidad para poder detener y reiniciar la simulación automáticamente, propiedad que fue necesaria a lo largo de las pruebas. Para ello fue necesario configurar el archivo llamado *remoteApiConnections.txt* que se encuentra en la carpeta de CoppeliaSim de la siguiente forma:

portIndex1_debug = true

De esta manera, el servicio se inicia automáticamente cada que se abre CoppeliaSim en el puerto 19997, por lo que al iniciar la conexión desde el lado del cliente fue necesario conectarse a dicho puerto. Para más detalles acerca de la conexión del lado del servidor consultar *Remote API Server Side* [29].

Para iniciar una conexión desde el cliente, fue necesario buscar los siguientes elementos en la carpeta de CoppeliaSim:

- sim.py
- simConst.py
- remoteApi.dll

Los archivos deben copiarse en la carpeta donde se guardarán los programas de Python y al crear un programa se debe llamar a *import sim* para cargar la biblioteca. Un ejemplo del código que debe de ejecutarse en Python para establecer una conexión con CoppeliaSim es de la siguiente manera:

```
Import sim
sim.simxFinish(-1)
ClientID=sim.simxStart('127.0.0.1', 19997, True, True, 2000, 5)
if (ClientID == 0):
        print("Se ha establecido una conexion")
        sim.simxFinish(ClientID)
else
        print("Error en la conexion")
sim.delete()
```

El código se inicia mandando a llamar a la biblioteca sim y se cierran todas las conexiones que podrían estar previamente abiertas. Se crea un cliente llamado ClientID y finalmente se pregunta si se ha realizado la conexión. De ser así, se podrá iniciar el código de las acciones que se van a ejecutar en el simulador. De lo contrario, simplemente indica que no se ha generado una conexión y termina el programa. Para más detalles acerca de la conexión del lado del cliente consultar *Remote API Client Side* [30].

Inicialmente se había planteado realizar el proyecto en Linux, considerando que esto facilitaría y aceleraría el funcionamiento del algoritmo genético. Esto es debido a que proyectos pasados han funcionado adecuadamente en dicho sistema operativo, pero al realizar las pruebas de conexión, se encontró que fallaba en gran cantidad de ocasiones y que era bastante inestable. Por ello, se migró todo el proyecto a Windows y se utilizó Anaconda para generar los programas en Python. Las pruebas resultaron exitosas, generando una conexión estable.

6.5 Marcha ideal

Para que el algoritmo genético pudiera ejecutarse adecuadamente fue necesario tener datos que ayudaran a la evolución del algoritmo y de esta forma pudiera converger a una solución óptima. Como además se buscaba generar movimientos similares a los de un can, se diseñó una marcha similar a la de la tesis “Robot cuadrúpedo inspirado en la marcha de un can” [23]. Con la ayuda de un video donde se muestra el desplazamiento de dicho robot [31], se pudo realizar una marcha similar, la cual se nombró “Marcha Ideal” y se muestra en la Figura 6.9, por medio de varias imágenes consecutivas, que se visualizan de izquierda a derecha.

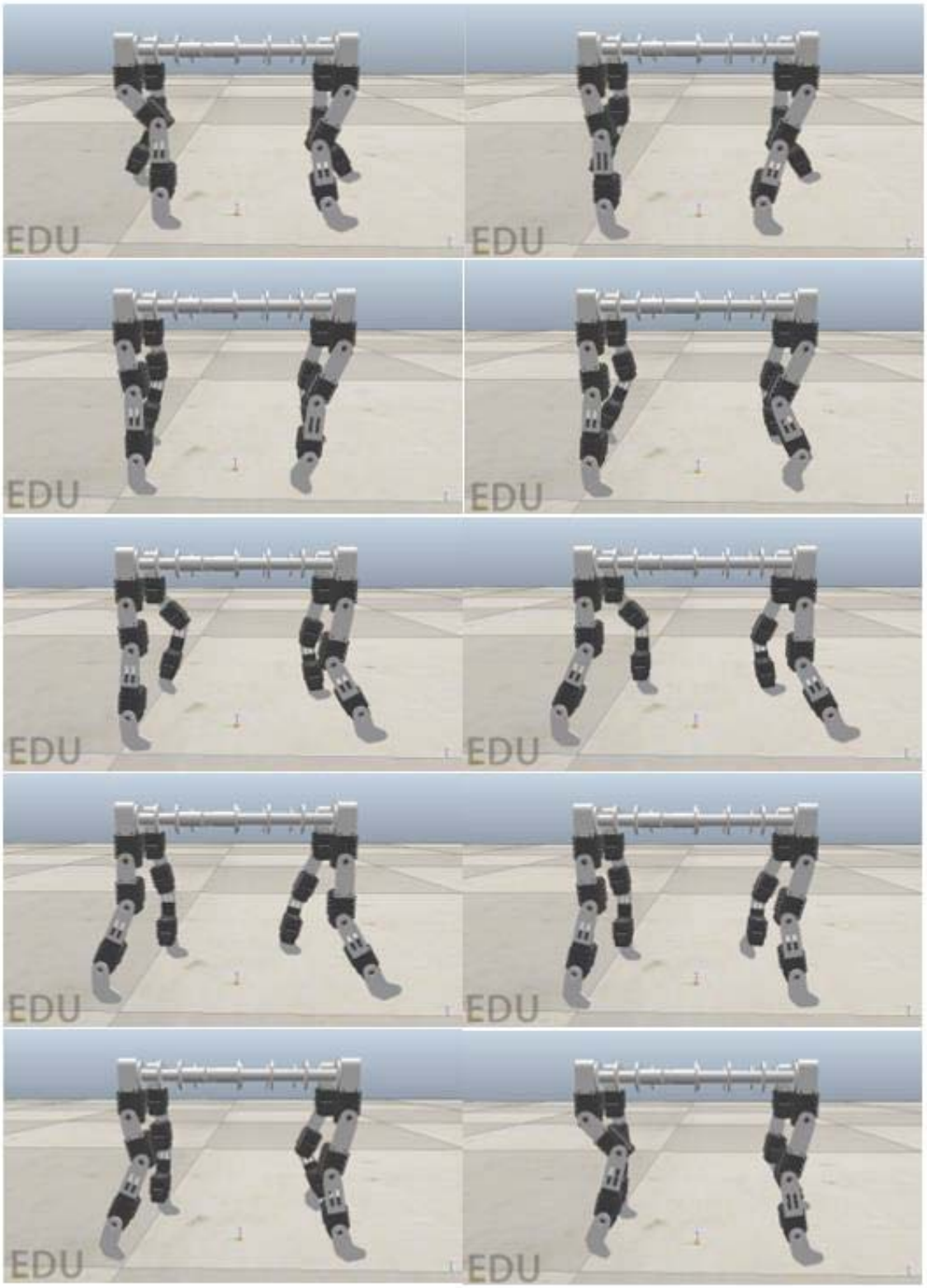

Figura 6.9: Marcha Ideal.

La Marcha Ideal cuenta con 10 pasos en total. La tabla con las posiciones de cada servomotor durante un ciclo de la Marcha Ideal se puede encontrar en el Anexo I.

En el siguiente enlace se puede visualizar un fragmento del video del robot original realizando su desplazamiento y la versión que se creó para el modelo del robot en el simulador.

https://youtu.be/ScgemME5qXw

El código que se realizó para generar la Marcha Ideal que se muestra en el video, se encuentra en el Anexo II.

6.6 Algoritmo genético

El objetivo principal del algoritmo genético es que, a partir de una población inicial aleatoria, el modelo del robot cuadrúpedo elegido aprenda a desplazarse dentro de un ambiente de simulación, generando un movimiento similar al de un can.

6.6.1 Codificación

Holland en su libro *Adaptation in Natural and Artificial Systems* [5] justifica por qué es más común usar una codificación binaria. Holland argumenta que, al hacer una comparación respecto de otro tipo de representación –por ejemplo, decimal-, la codificación binaria cuenta con más esquemas que favorecen la diversidad e incrementa la probabilidad de formar bloques constructores [4]. Es decir, la codificación binaria tiene una mayor probabilidad de que se produzca un incremento en la aptitud de cada generación y esto se traduce a un mejor desempeño del algoritmo genético. Es por ello que el proyecto adoptó una codificación binaria, donde cada cromosoma representa una marcha del can.

Una marcha como se ha mencionado anteriormente, cuenta con 10 pasos en total. Por cada paso, cada uno de los 12 servomotores se colocará en una posición. Por lo tanto, al final de la marcha se habrán obtenido un total de 120 posiciones entre todos los servomotores.

Cada posición de un servomotor está representada por un número decimal y para poder convertir este número a un número binario, es importante primero determinar la cantidad de bits que éste representa. Para ello es importante saber los rangos de movimiento de cada servomotor.

La marcha ideal es un conjunto de 120 números decimales, en donde en ninguna de sus posiciones se genera un número mayor a 60 ni menor a -60.

Es por ello que este rango es el que se consideró para obtener la cantidad de bits por cada número decimal.

La siguiente fórmula muestra cómo obtener el número de bits de un número decimal:

$$log_2(x)$$

Donde x es la cantidad total de valores que se encuentran en el rango establecido, que en este caso es de -60 a 60 siendo un total de 121 números. El resultado es 6.91, valor que debe ser redondeado hacia arriba, dando un valor de 7 bits por número decimal y al multiplicar por las 120 posiciones, da un total de 840 bits por cada ciclo de marcha.

6.6.2. Individuos

Cada individuo está representado por un cromosoma y cada cromosoma contiene 120 genes. A su vez, cada gen contiene 7 alelos; es decir, cada individuo representa un ciclo de marcha, el cual contiene 120 posiciones de los servomotores y cada posición está representada por 7 bits. Esto quiere decir que cada individuo tiene 840 bits en total.

6.6.3 Selección

Se utilizó el método de selección por torneo determinístico, debido a que este método aseguró que siempre se seleccionaran a los mejores individuos, elevando la probabilidad de tener una mejor aptitud con el paso de las generaciones.

6.6.4. Cruza, mutación y clonación

El algoritmo genético parte de una población inicial de números decimales aleatorios dentro de los rangos establecidos y para poder realizar la cruza y mutación, fue necesario convertir estos números decimales a sus valores equivalentes en su versión binaria, utilizando la siguiente fórmula:

$$Num\_bin = \left(\frac{Num\_dec - q_{min}}{q_{max} - q_{min}}\right) * \ 2^l$$

Donde:

Num_bin = Número decimal equivalente a su versión binaria.
Num_dec = Número decimal.
qmin = Valor mínimo de movimiento de las articulaciones.
qmax = Valor máximo de movimiento de las articulaciones.
l = Cantidad de bits por número decimal.

Es necesario utilizar esta fórmula debido a que no se pueden convertir números negativos a números binarios.

- Cruza: La cruza se realizó con los métodos de cruza en un punto y cruza en dos puntos, con una probabilidad de 0.6, 0.75 y 0.9.

- Mutación: La mutación se realizó intercambiando un bit de 1 a 0 y viceversa, con una probabilidad de 0.001, 0.005 y 0.01.

- Clonación: En los casos en donde no hubo cruza, se colocaron directamente a los padres en la siguiente generación.

Para continuar con el proceso del algoritmo genético, se realizó la conversión de binario a decimal con la siguiente fórmula:

$$Num\_dec = \left(\frac{Num\_bin}{2^{l}}\right) * (\, q_{max} - q_{min}) + q_{min}$$

Donde:

Num_dec = Número decimal.
Num_bin = Número decimal equivalente a su versión binaria.
qmin = Valor mínimo de movimiento de las articulaciones.
qmax = Valor máximo de movimiento de las articulaciones.
l = Cantidad de bits por número decimal.

6.6.5. Aptitud

La función de aptitud permite orientar al algoritmo para saber cuál de sus individuos es mejor y procurar que persista y mejore con el tiempo. Es por ello que la función de aptitud debe evaluarse respecto a qué tan parecidos son los individuos a la marcha ideal que se creó anteriormente, cuestión que no fue sencillo de solucionar.

Para obtener una función de aptitud óptima, se hicieron múltiples pruebas variando los parámetros del algoritmo genético.

Los valores de la función de aptitud se establecieron en el rango de 0 a 1, tomando como mejor individuo al que más se acercara a 1. Para que un individuo fuera una buena solución se consideraron dos cuestiones:

- Avanza
- El movimiento es estable

Cada uno de los puntos mencionados anteriormente representaron el 50% de la solución.

- **Posición**

Los movimientos del robot se analizaron en el simulador respecto a los planos x,y,z. El robot en el simulador está orientado respecto a los ejes que se muestran en la esquina inferior derecha de la Figura 6.10, a los que CoppeliaSim llama “coordenadas del mundo”.

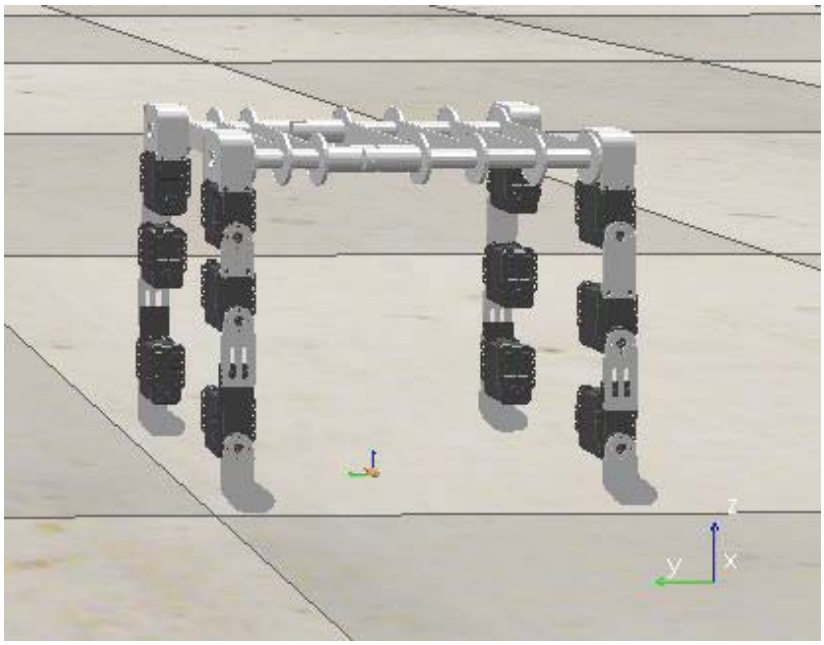

Figura 6.10: Plano x,y,z respecto a las coordenadas del mundo.

Para que el robot realice un avance debe de desplazarse a lo largo del eje “y” en dirección negativa y para asegurarse de que avance en línea recta debe de mantenerse en el eje “x” en valores cercanos a cero. El eje “z” no interfiere en el proceso de avance del robot.

- **Orientación**

Para que el movimiento sea estable se considera la orientación del robot, específicamente de la base del robot llamada “Cuerpo1”, la cual contempla al siguiente conjunto de piezas que se muestran en la Figura 6.11.

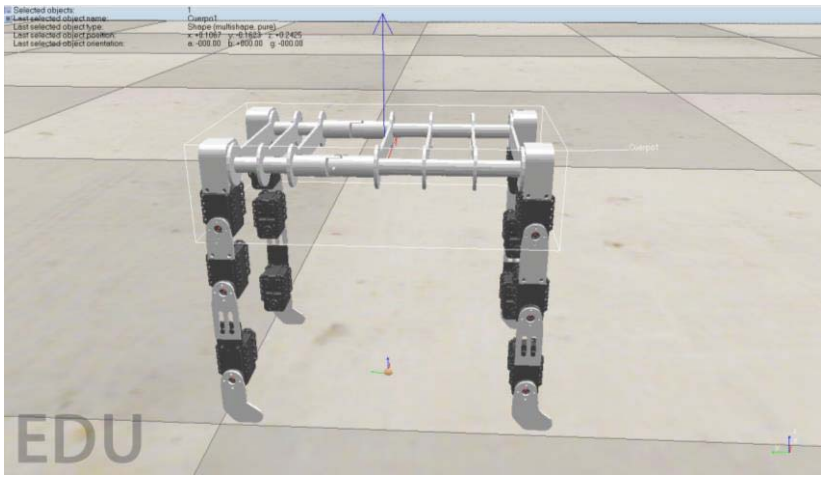

Figura 6.11: Base del robot cuadrúpedo.

Como se muestra en la Figura 6.11 la orientación no considera a las patas del robot. Las orientaciones en CoppeliaSim están dadas por Alpha, Beta y Gamma. Para un mayor entendimiento en las Figuras 6.12, 6.13 y 6.14 se muestra una cierta inclinación del robot respecto a cada orientación mencionada.

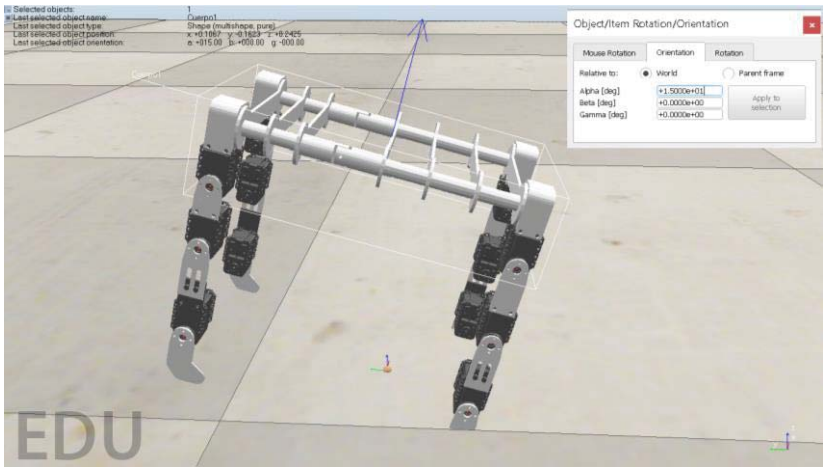

Figura 6.12: Inclinación en orientación Alpha.

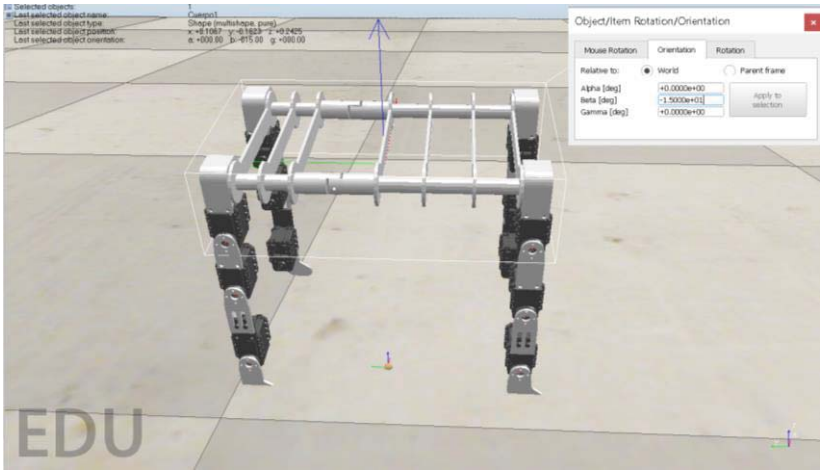

Figura 6.13: Inclinación en orientación Beta.

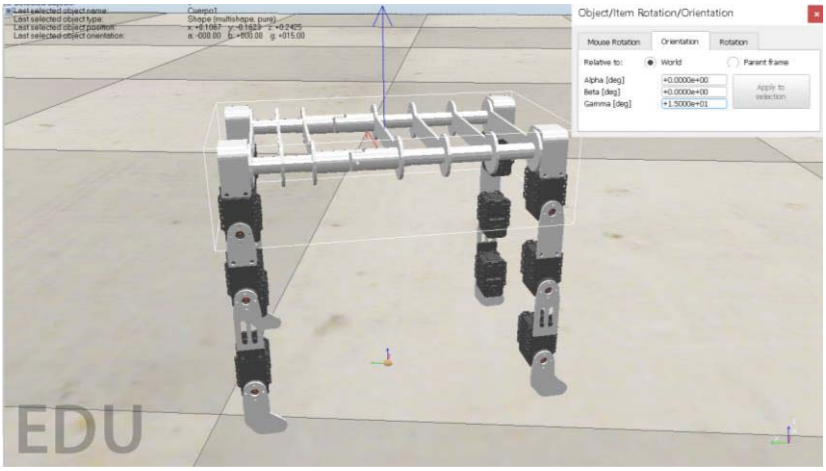

Figura 6.14: Inclinación en orientación Gamma.

Para que el movimiento sea estable, las tres orientaciones deben de ser cercanas a cero grados.

6.6.5.1 Avance

Como se mencionó anteriormente, para realizar la conexión de Python con CoppeliaSim se utilizó un servicio de servidor API remoto continuo, el cual hizo que el robot adoptara la posición de la Figura 6.15 antes de iniciar cada marcha, lo cual no permitió tomar la posición inicial como referencia.

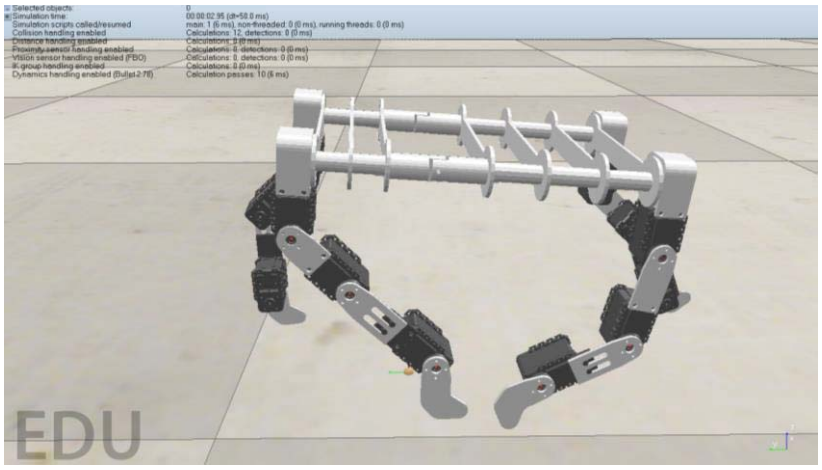

Figura 6.15: Posición del robot después de iniciar la simulación.

Al momento de iniciar los movimientos de una marcha, el robot regresaba a la posición de la Figura 6.11 y realizaba los movimientos adecuadamente sin mayores complicaciones. Es por esto que se optó por considerar la posición del robot después del primer paso como la posición inicial, y a partir de ésta evaluar su desplazamiento. Esto aseguró que el robot, después del primer movimiento, tendría un desplazamiento hacia adelante. Para asegurar que el robot continuara avanzando, cuando llegaba al sexto paso, éste se tomaba como posición inicial y a partir de éste, se evaluaba el resto de la marcha.

1	2	3	4	5	6	7	8	9	10

Figura 6.16: Método para analizar el avance del robot.

El método de la Figura 6.16 solo toma en cuenta 2 posiciones iniciales durante la prueba del ciclo de marcha, siendo esto lo más óptimo para la función de aptitud.

Para asegurar un correcto desplazamiento entre ciclos de marcha, fue necesario agregar el primer paso al final de la marcha, como se muestra en la Figura 6.17, con el fin de analizar la transición del décimo paso al primer paso.

1	2	3	4	5	6	7	8	9	10	1

Figura 6.17: Método para asegurar un avance entre ciclos de marcha.

Al agregar un paso más, el tamaño del individuo aumentó de 120 a 132 posiciones. Este aumento de posiciones únicamente se considera para evaluar la aptitud del individuo sin afectar el resto de los operadores genéticos.

El avance representó el 50% de la aptitud total del individuo, y para determinar si la marcha cumplía con este porcentaje se creó un contador, el cual sumaba 1 cada vez que no había un avance. Por lo tanto, si el contador era mayor a cero se tomaba como una marcha que no generaba avance.

6.6.5.2. Estabilidad

Para poder evaluar la estabilidad del robot se utilizaron las orientaciones Alpha, Beta y Gamma ya mencionadas anteriormente. Se realizó un análisis de las orientaciones que tiene la base del robot en un ciclo de la Marcha Ideal. Los resultados pueden encontrarse en el Anexo III. De esta forma se obtuvieron los rangos de movimiento de cada orientación, los cuales se muestran en la Tabla 6.3.

Tabla 6.3 Rangos de movimiento de la base del robot en cada orientación.

Orientación	Rango de movimiento
Alpha	-5° a 5°
Beta	-5° a 5°
Gamma	-10° a 10°

El método que se utilizó para evaluar las orientaciones Alpha, Beta y Gamma en un ciclo de marcha, fue creando 3 contadores, donde cada uno representó a una orientación. Cada contador sumaba 1 o 3 cada vez que la base del robot tenía una posición que no estaba dentro de los rangos establecidos de las orientaciones. La Figura 6.18 muestra la forma en que se evaluó la orientación en Alpha y Beta.

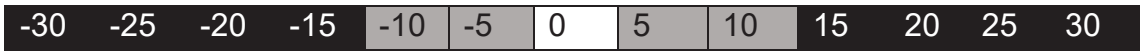

Figura 6.18: Método para evaluar la orientación en Alpha y Beta.

El contador sumaba 1 si la posición de la base del robot se encontraba en el rango de -5 a -10 o de 5 a 10, y en el caso de que la posición era menor a -10 o mayor a 10 el contador sumaba 3. Esto le permitió al algoritmo genético identificar con mayor facilidad a un individuo que generaba un movimiento caótico.

En el caso de la orientación en Gamma, el contador sumaba 1 si la posición de la base del robot se encontraba en el rango de -10 a -15 o de 10 a 15, y en el caso en el que la posición era menor a -15 o mayor a 15 el contador sumaba 3, como se muestra en la Figura 6.19.

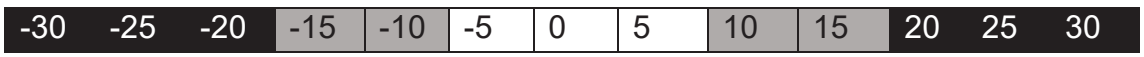

Figura 6.19: Método para evaluar la orientación en Gamma.

Siguiendo con el análisis de la estabilidad, se analizaron las posiciones en el eje “z” durante un ciclo de la Marcha Ideal. Los resultados pueden encontrarse en el Anexo IV. De estos resultados se obtuvieron los rangos en los que se encontraba la base del robot en el eje “z” durante la marcha, los cuales son de

0.226 a 0.2361 metros. Se creó un contador para evaluar el eje “z”, y cada que la posición de la base del robot sobrepasaba los rangos establecidos, el contador sumaba 1.

Por cada paso se pueden generar máximo 10 errores: 3 errores del contador de la orientación Alpha, 3 errores del contador de la orientación Beta, 3 errores del contador de la orientación Gamma y 1 error del contador de la posición en el eje “z”, habiendo un máximo de 110 errores por ciclo de marcha.

La estabilidad representa el 50% de la aptitud de un individuo y el avance el otro 50%. Para mostrar la aptitud de cada individuo en una escala de 0 a 1, siendo 0 una aptitud con 110 errores y siendo 1 una aptitud con cero errores, fue necesario realizar una normalización de los parámetros. La normalización se realizó con la siguiente fórmula:

$$Vn = \frac{(X - X_{\min})(R_2 - R_1)}{X_{max} - X_{min}} + R_1$$

Donde:

Vn = Valor normalizado.
X = Valor a normalizar.
Xmin = Rango mínimo de X.
Xmax = Rango máximo de X.
R1 = Valor mínimo normalizado.
R2 = Valor máximo normalizado.

Debido a que una aptitud de 1 es igual a cero errores, Xmax es igual a cero y a su vez Xmin igual a 110.

Cabe mencionar que cada que el contador del eje “z” o el contador de la orientación Gamma eran mayores a cero, se reducía al 90% la aptitud final. Esto se hizo para asegurar que solo los individuos con mayor estabilidad y un mejor avance obtuvieran una mayor aptitud.

6.6.6. Elitismo

El elitismo es una variable que copia al individuo con la mayor aptitud de una generación a la siguiente. Esto se hace con el fin de que nunca se pierda el mejor individuo ni que se disminuya el valor de la aptitud al paso de las generaciones.

El siguiente diagrama de flujo muestra la estructura del algoritmo genético. Para mayor información del funcionamiento del algoritmo, consultar el código que se encuentra en el Anexo V.

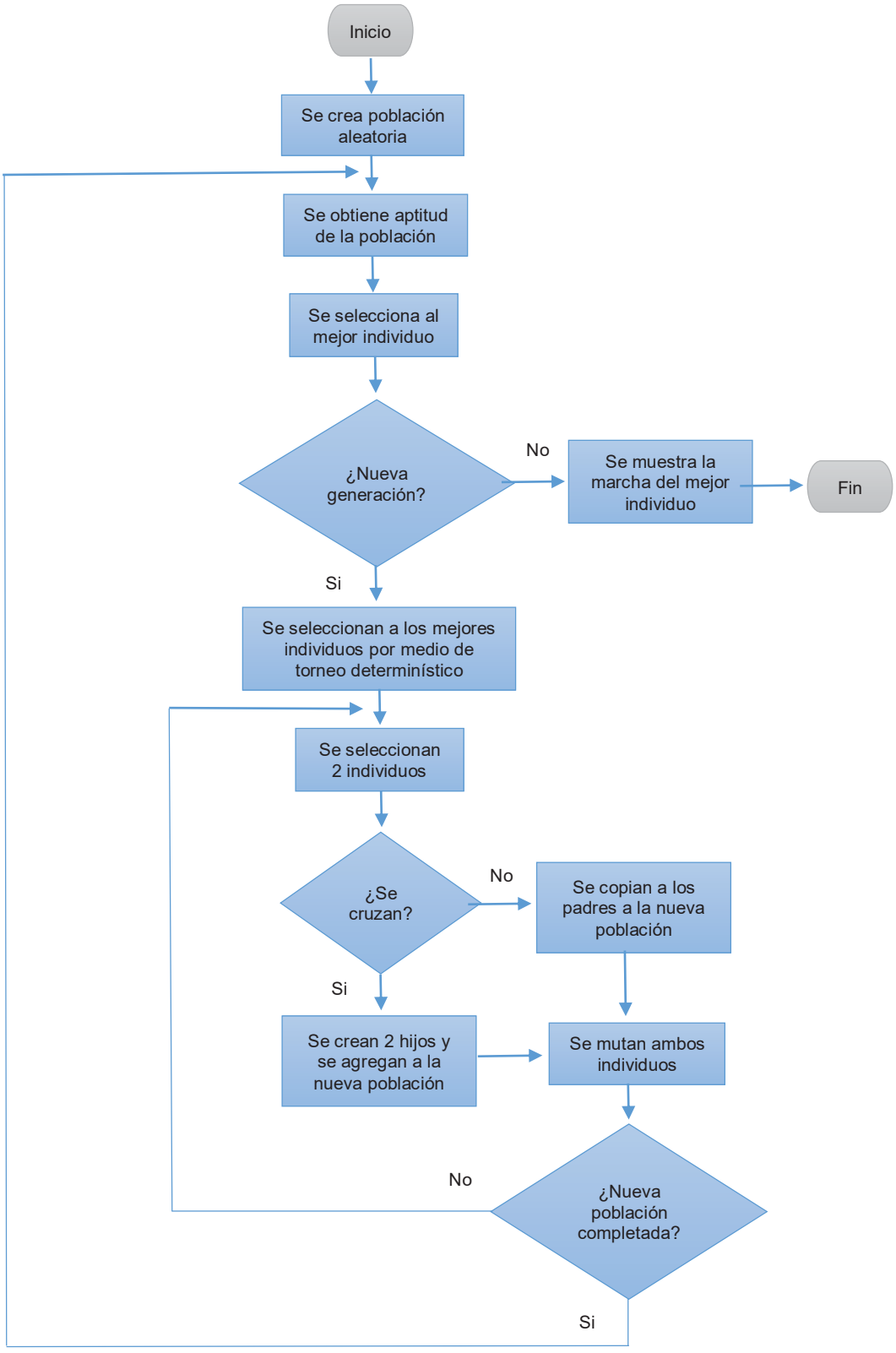

Figura 6.20: Diagrama de flujo de la estructura del algoritmo genético.

Debido a que solo se generan nuevos valores en los individuos al inicio del algoritmo genético, todos los individuos se deben de mutar sin importar si se hayan cruzado o no, como se muestra en la Figura 6.20. Esto se hace con el fin de generar nuevos valores a lo largo de las generaciones, ya que la cruza únicamente crea combinaciones de los valores dados en la población inicial.

6.7. Complicaciones durante la simulación

6.7.1. Caídas

Durante las pruebas se observó que cada vez que se caía el robot, no era posible regresarlo a su posición inicial, debido al realismo de la simulación. Es por esto que la solución fue reiniciar la simulación automáticamente, permitiendo que el robot regresara a su posición inicial y de esta manera continuar evaluando la marcha de los siguientes individuos.

Para reiniciar la simulación se utilizó el siguiente código.

```
sim.simxStopSimulation(ClientID, sim.simx_opmode_oneshot)
time.sleep(1)
sim.simxStarSimulation(ClientID, sim.simx_opmode_oneshot)
time.sleep(0.5)
```

Para que la conexión se reestablezca adecuadamente entre cada reinicio de la simulación, es necesario que pasen al menos 0.5 segundos después de haber detenido la simulación y también al iniciarla.

6.7.2. Colisiones

Debido a que los rangos de movimiento que tiene cada articulación son de -60 a 60 grados, es posible que se generen colisiones, por lo que fue importante darle a conocer a CoppeliaSim las distintas combinaciones de colisiones que se podían generar, ya que, de no hacerlo, la simulación se quedaría bloqueada.

La Figura 6.21 muestra las posibles colisiones que se pueden generar entre las patas del robot.

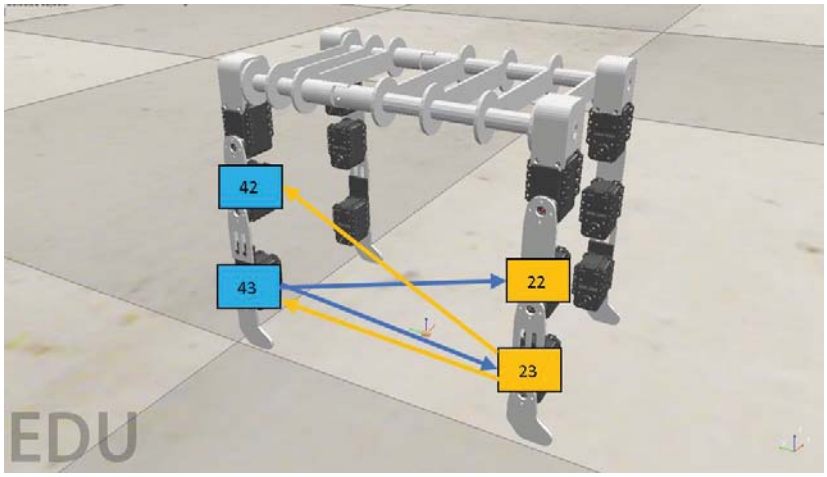

Figura 6.21: Posibles colisiones entre patas.

La pata43 puede colisionar con la pata22 y la pata23, y la pata23 puede colisionar con la pata42 y la pata 43. Lo mismo sucede en las otras dos patas.

Las colisiones también se pueden generar entre las patas y la base del robot, como se muestra en la Figura 6.22.

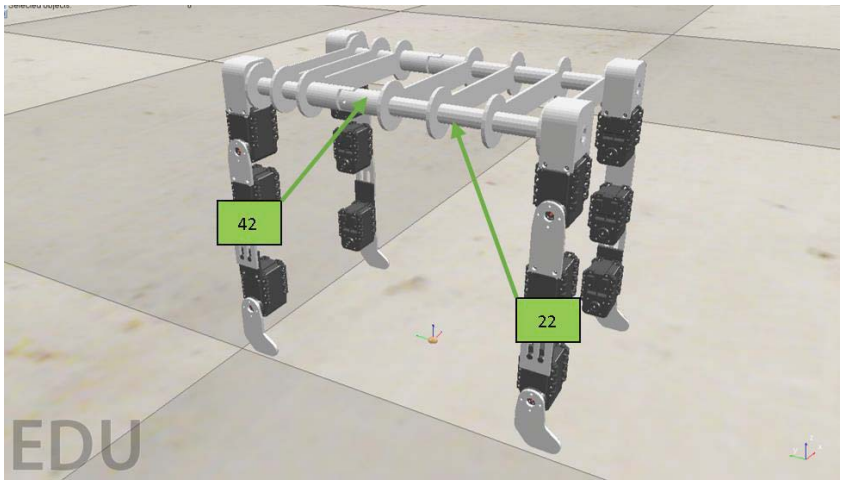

Figura 6.22: Posibles colisiones entre las patas y la base del robot.

Para que Python reconozca las colisiones que se pueden generar, fue necesario primero agregarlas y nombrarlas en el simulador, como se muestra en la Figura 6.23.

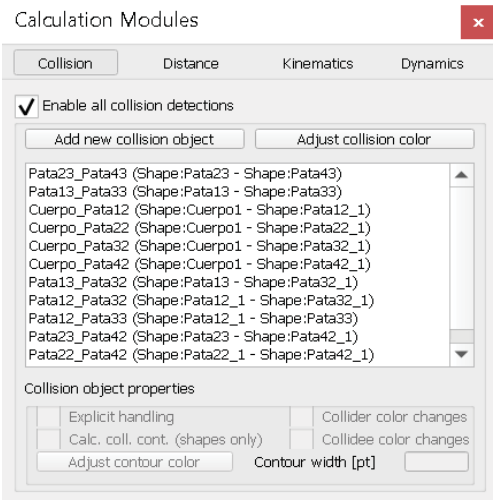

Figura 6.23: Posibles colisiones del robot cuadrúpedo en el simulador.

De esta manera, Python pudo reconocer el conjunto de articulaciones que podrían llegar a colisionarse y continuar ejecutando los siguientes movimientos sin mayor problema.

CAPITULO 7. ANALISIS E INTERPRETACION DE RESULTADOS

7.1 Análisis de resultados

Las pruebas se iniciaron a partir de los valores recomendados por DeJong para los parámetros de control [21]:

Tipo de cruza: 2 puntos.
Tamaño de población: 50 individuos
Porcentaje de cruza: 0.60
Porcentaje de mutación: 0.001

Se inició con el método de cruza en 2 puntos, variando los porcentajes de cruza y mutación individualmente. De igual manera, se repitió el proceso con el método de cruza en 1 punto, para definir cuál es la combinación de parámetros más efectiva entre ambos métodos.

Debido a que el algoritmo genético es una técnica de búsqueda estocástica, se realizaron 3 pruebas de cada combinación de parámetros para obtener su funcionamiento promedio.

De cada prueba que se llevó a cabo, se obtuvo la aptitud del mejor individuo y la aptitud promedio de la población de cada generación, esto con el fin de evaluar el desempeño del algoritmo genético considerando a toda la población.

7.1.1. Cruza en 2 puntos

7.1.1.1 Variando el porcentaje de mutación

Los cambios en el porcentaje de mutación se realizaron con los siguientes parámetros de control:

Número de generaciones: 50
Número de individuos: 50
Porcentaje de cruza: 0.6

- **Mutación 0.001**

La Tabla 7.1 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.1 Evolución de la aptitud del mejor individuo con mutación de 0.001.

Generación	Prueba	Aptitud	Promedio
1	1	0.6890	0.6987
	2	0.7509	
	3	0.6563	
5	1	0.78	0.7557
	2	0.7690	
	3	0.7181	
10	1	0.7981	0.7921
	2	0.8127	
	3	0.7654	
15	1	0.8272	0.82
	2	0.8309	
	3	0.8018	
20	1	0.8309	0.8272
	2	0.8381	
	3	0.8127	
25	1	0.8381	0.8345
	2	0.8418	
	3	0.8236	
30	1	0.8527	0.8454
	2	0.8527	
	3	0.8309	
35	1	0.8672	0.8563
	2	0.8563	
	3	0.8454	
40	1	0.8672	0.8587
	2	0.8563	
	3	0.8527	
45	1	0.8672	0.8587
	2	0.8563	
	3	0.8527	
50	1	0.8672	0.8612
	2	0.86	
	3	0.8563	

La Figura 7.1 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

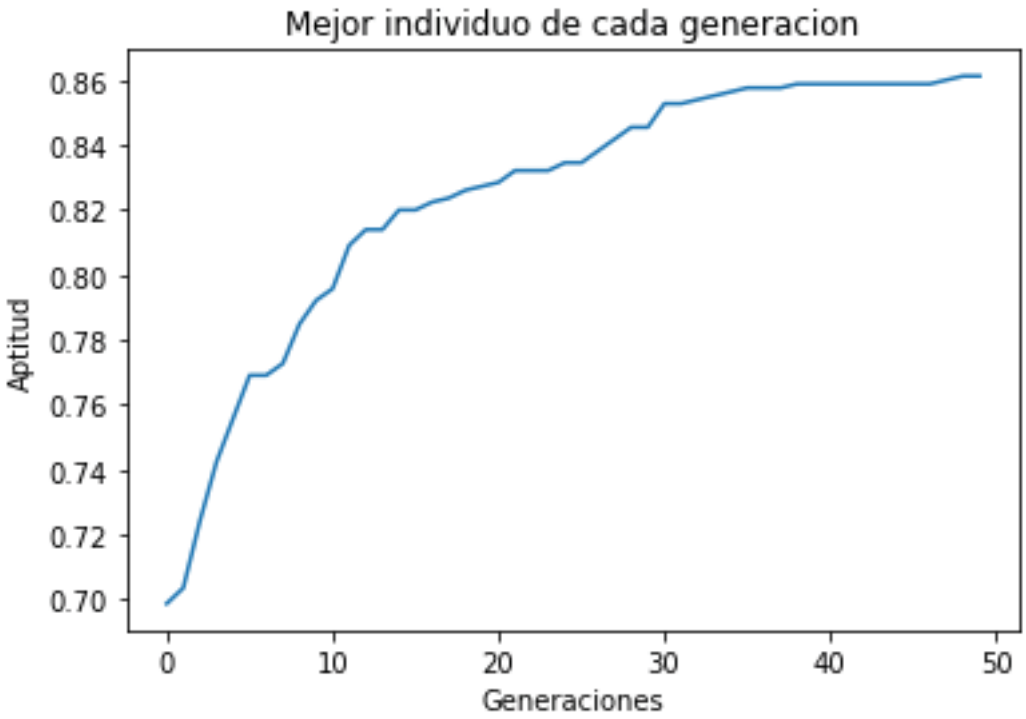

Figura 7.1: Promedio de aptitud de mejor individuo con mutación de 0.001.

La Figura 7.2 muestra la aptitud promedio de toda la población en cada generación.

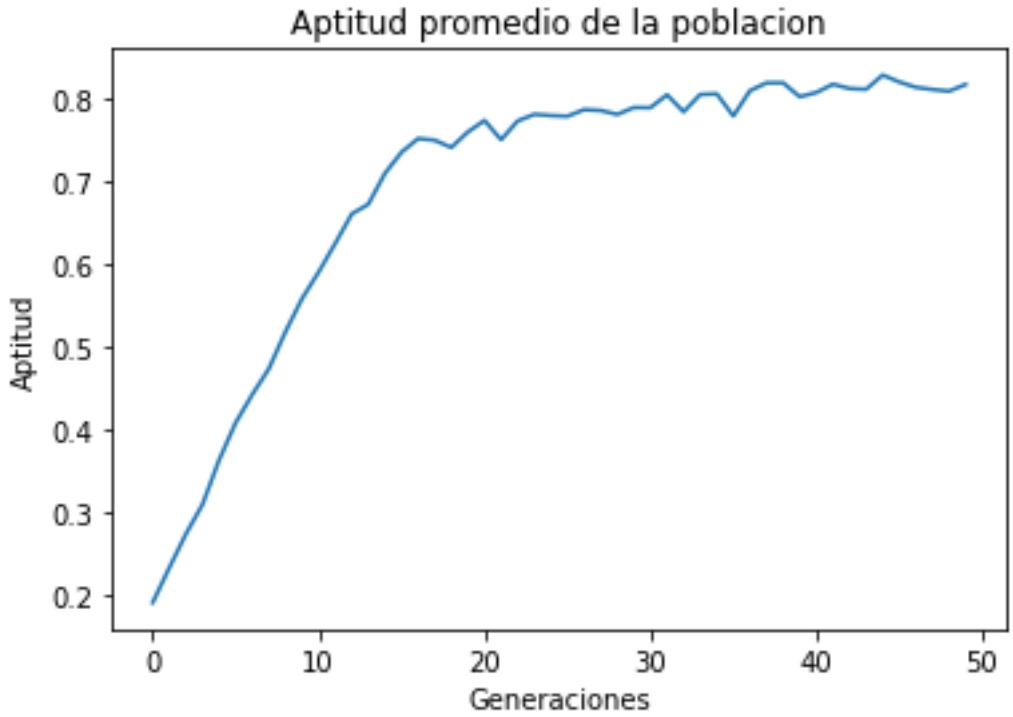

Figura 7.2: Aptitud promedio de la población con mutación de 0.001.

- **Mutación 0.005**

La Tabla 7.2 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.2 Evolución de la aptitud del mejor individuo con mutación de 0.005.

Generación	Prueba	Aptitud	Promedio
1	1	0.6563	0.7084
	2	0.74	
	3	0.7290	
5	1	0.8090	0.7836
	2	0.8127	
	3	0.7290	
10	1	0.8163	0.8115
	2	0.8345	
	3	0.7836	
15	1	0.8236	0.8272
	2	0.8490	
	3	0.8090	
20	1	0.8490	0.8515
	2	0.8527	
	3	0.8527	
25	1	0.8563	0.8684
	2	0.8745	
	3	0.8745	
30	1	0.8563	0.8721
	2	0.8818	
	3	0.8781	
35	1	0.86	0.8733
	2	0.8818	
	3	0.8781	
40	1	0.86	0.8733
	2	0.8818	
	3	0.8781	
45	1	0.8636	0.8745
	2	0.8818	
	3	0.8781	
50	1	0.8636	0.8745
	2	0.8818	
	3	0.8781	

La Figura 7.3 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

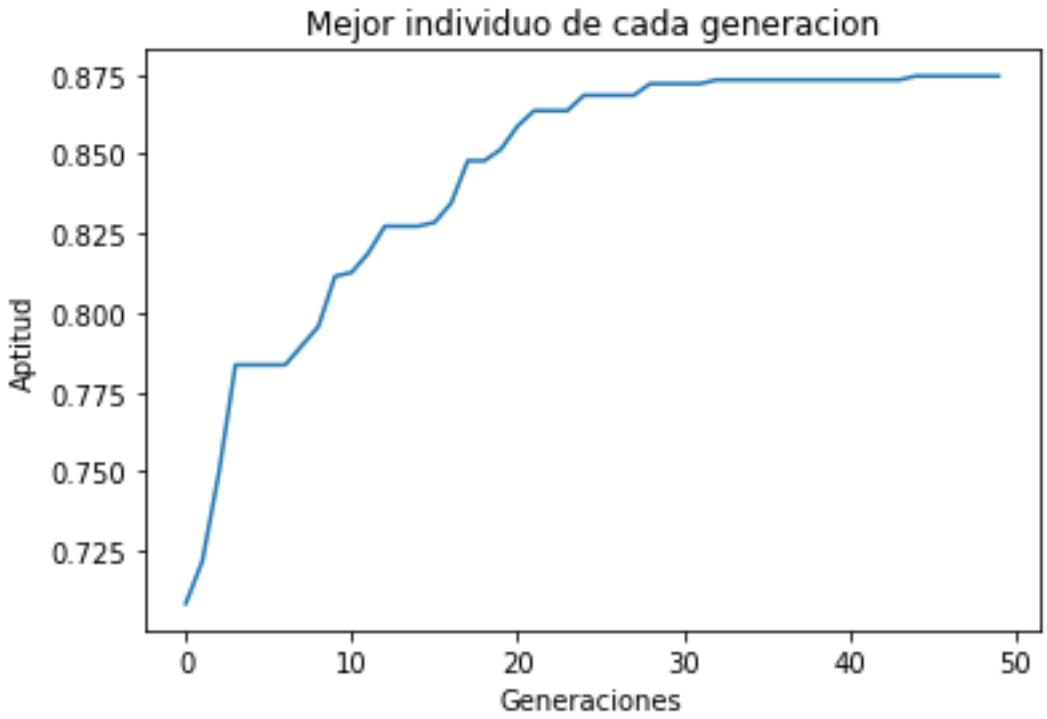

Figura 7.3: Promedio de aptitud de mejor individuo con mutación de 0.005.

La Figura 7.4 muestra la aptitud promedio de toda la población en cada generación.

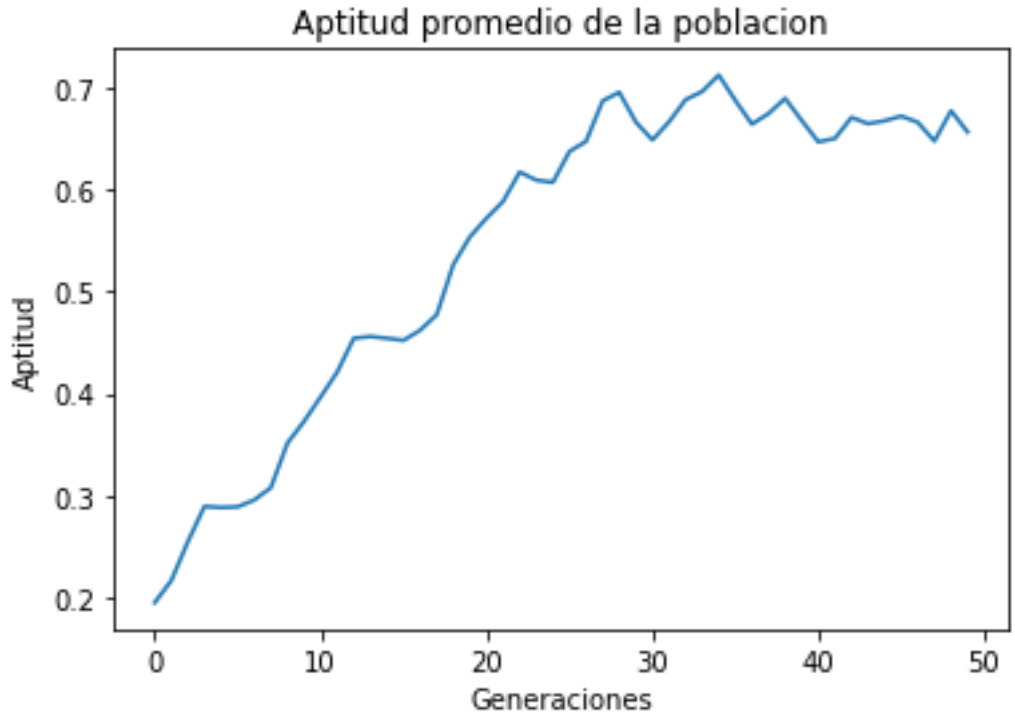

Figura 7.4: Aptitud promedio de la población con mutación de 0.005.

- **Mutación 0.01**

La Tabla 7.3 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.3 Evolución de la aptitud del mejor individuo con mutación de 0.01.

Generación	Prueba	Aptitud	Promedio
1	1	0.6236	0.6769
	2	0.7072	
	3	0.7	
5	1	0.7618	0.7703
	2	0.7618	
	3	0.7872	
10	1	0.8163	0.8018
	2	0.7981	
	3	0.7909	
15	1	0.8345	0.8139
	2	0.8018	
	3	0.8054	
20	1	0.8345	0.8309
	2	0.8236	
	3	0.8345	
25	1	0.86	0.8466
	2	0.8418	
	3	0.8381	
30	1	0.86	0.8478
	2	0.8418	
	3	0.8418	
35	1	0.8636	0.8527
	2	0.8490	
	3	0.8454	
40	1	0.8636	0.8539
	2	0.8490	
	3	0.8490	
45	1	0.8636	0.8539
	2	0.8490	
	3	0.8490	
50	1	0.8636	0.8563
	2	0.8563	
	3	0.8490	

La Figura 7.5 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

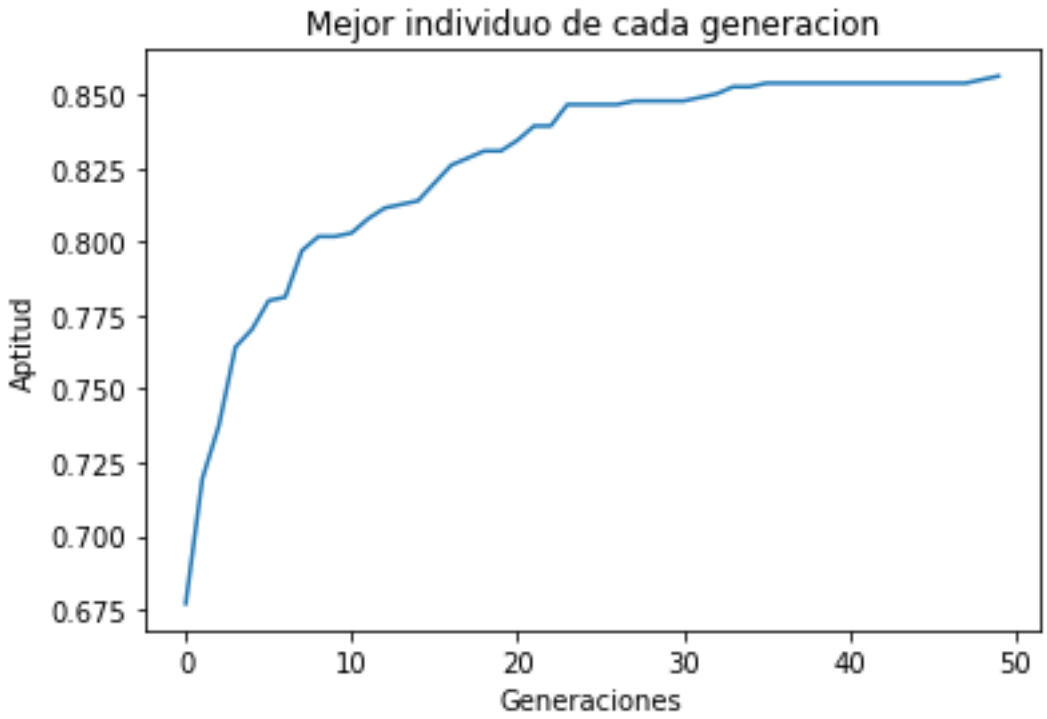

Figura 7.5: Promedio de aptitud de mejor individuo con mutación de 0.01.

La Figura 7.6 muestra la aptitud promedio de toda la población en cada generación.

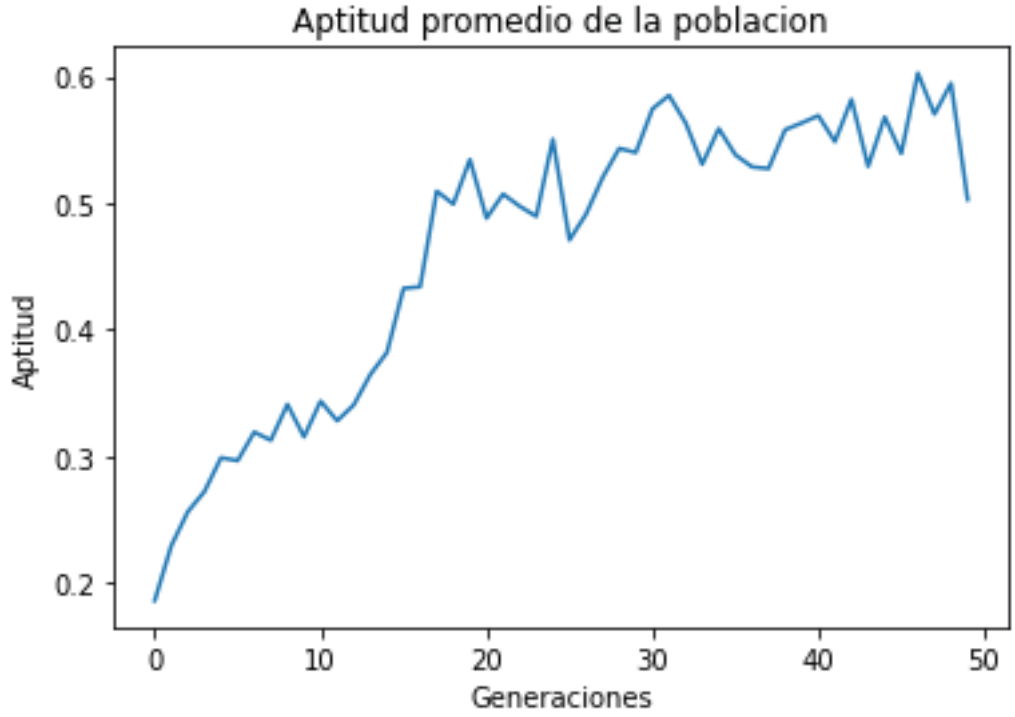

Figura 7.6: Aptitud promedio de la población con mutación de 0.01.

La Tabla 7.4 muestra una comparación del mejor individuo según el porcentaje de mutación.

Tabla 7.4 Mejor individuo respecto al porcentaje de mutación.

Mutación	Mejor aptitud
0.001	0.8612
0.005	0.8745
0.01	0.8563

La Tabla 7.5 muestra la aptitud promedio de la población en la primera y última generación según el porcentaje de mutación, y asi como la mejora de aptitud que hubo entre ellas.

Tabla 7.5 Aptitud promedio de la población respecto al porcentaje de mutación.

Mutación	Aptitud promedio de la población		Mejora
	Primera generación	Última generación	
0.001	0.1910	0.8176	0.6266
0.005	0.1952	0.6561	0.4609
0.01	0.1861	0.5028	0.3167

El porcentaje mutación que genera un individuo con una mayor aptitud no siempre produce una mayor aptitud promedio en la población, debido a que generalmente entre mayor mutación se tenga, la aptitud de la población, en general, tiende a disminuir. Sin embargo, a su vez, esto permite probar mayor cantidad de combinaciones, y en algunos casos generar un mejor individuo, como fue el caso del porcentaje de mutación de 0.005.

7.1.1.2 Variando el porcentaje de cruza

Los cambios en el porcentaje de cruza se realizaron con los siguientes parámetros de control:

Número de generaciones: 50
Número de individuos: 50
Porcentaje de mutación: 0.001

Las pruebas que se realizaron al variar el porcentaje de mutación, se llevaron a cabo con un porcentaje de cruza de 0.60. Debido a esto, las variaciones de cruza iniciaron a partir de 0.75, pero al realizar la comparación de resultados se consideraron los valores dados en la Tabla 7.1.

- **Cruza 0.75**

La Tabla 7.6 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.6 Evolución de la aptitud del mejor individuo con cruza de 0.75.

Generación	Prueba	Aptitud	Promedio
1	1	0.7181	0.7618
	2	0.82	
	3	0.7472	
5	1	0.7690	0.7896
	2	0.82	
	3	0.78	
10	1	0.8272	0.8260
	2	0.82	
	3	0.8309	
15	1	0.8418	0.8321
	2	0.82	
	3	0.8345	
20	1	0.8672	0.8563
	2	0.8490	
	3	0.8527	
25	1	0.8818	0.8648
	2	0.86	
	3	0.8527	
30	1	0.8854	0.8696
	2	0.8636	
	3	0.86	
35	1	0.8854	0.8757
	2	0.8745	
	3	0.8672	
40	1	0.8890	0.8781
	2	0.8745	
	3	0.8709	
45	1	0.8890	0.8793
	2	0.8781	
	3	0.8709	
50	1	0.8890	0.8830
	2	0.8818	
	3	0.8781	

La Figura 7.7 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

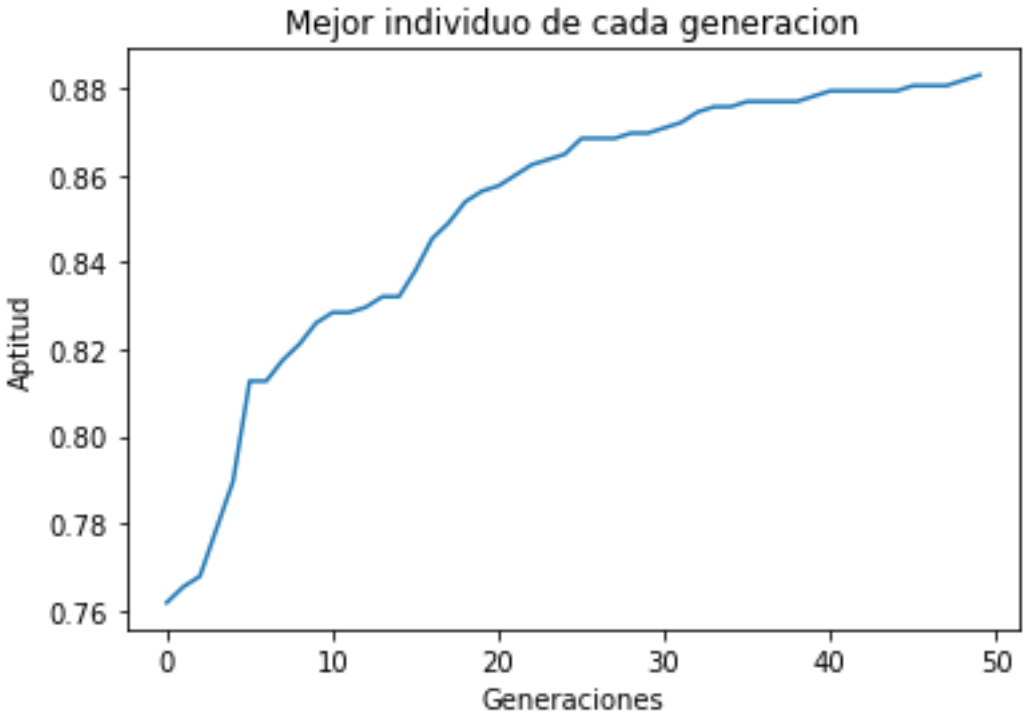

Figura 7.7: Promedio de aptitud de mejor individuo con cruza de 0.75.

La Figura 7.8 muestra la aptitud promedio de toda la población en cada generación.

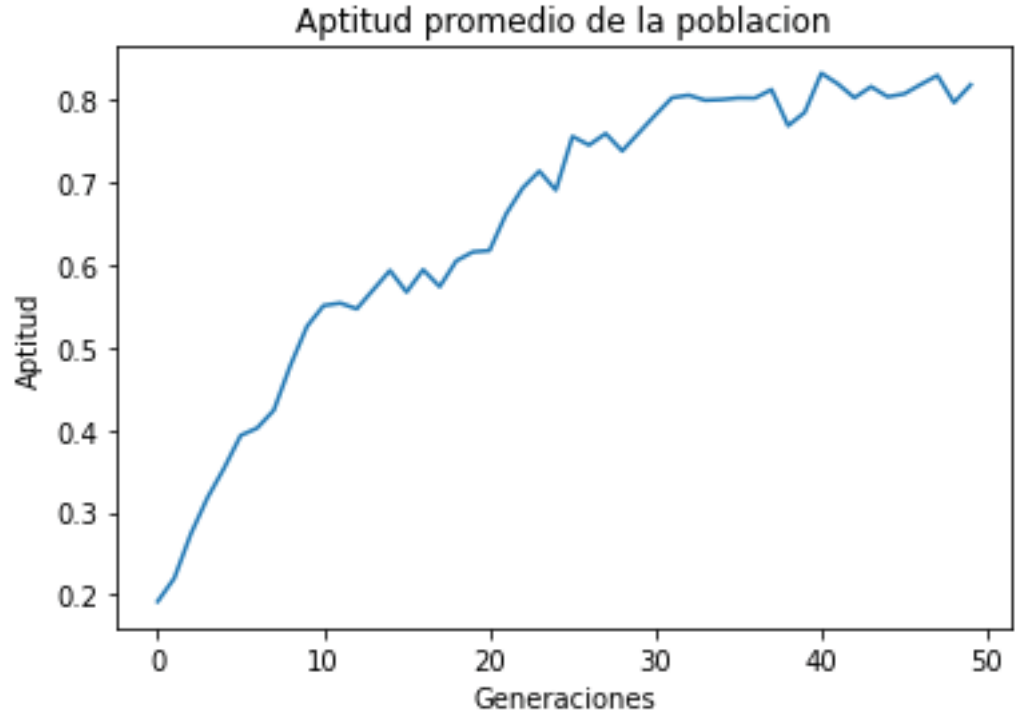

Figura 7.8: Aptitud promedio de la población con cruza de 0.75.

- **Cruza de 0.90**

La Tabla 7.7 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.7 Evolución de la aptitud del mejor individuo con cruza de 0.9.

Generación	Prueba	Aptitud	Promedio
1	1	0.6672	0.6684
	2	0.6672	
	3	0.6709	
5	1	0.78	0.7581
	2	0.7509	
	3	0.7436	
10	1	0.8090	0.7993
	2	0.8272	
	3	0.7618	
15	1	0.8345	0.8139
	2	0.8454	
	3	0.7618	
20	1	0.8381	0.8466
	2	0.86	
	3	0.8418	
25	1	0.8563	0.8587
	2	0.8672	
	3	0.8527	
30	1	0.8636	0.8660
	2	0.8709	
	3	0.8636	
35	1	0.8709	0.8696
	2	0.8709	
	3	0.8672	
40	1	0.8709	0.8721
	2	0.8745	
	3	0.8709	
45	1	0.8781	0.8745
	2	0.8745	
	3	0.8709	
50	1	0.8781	0.8781
	2	0.8818	
	3	0.8745	

La Figura 7.9 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

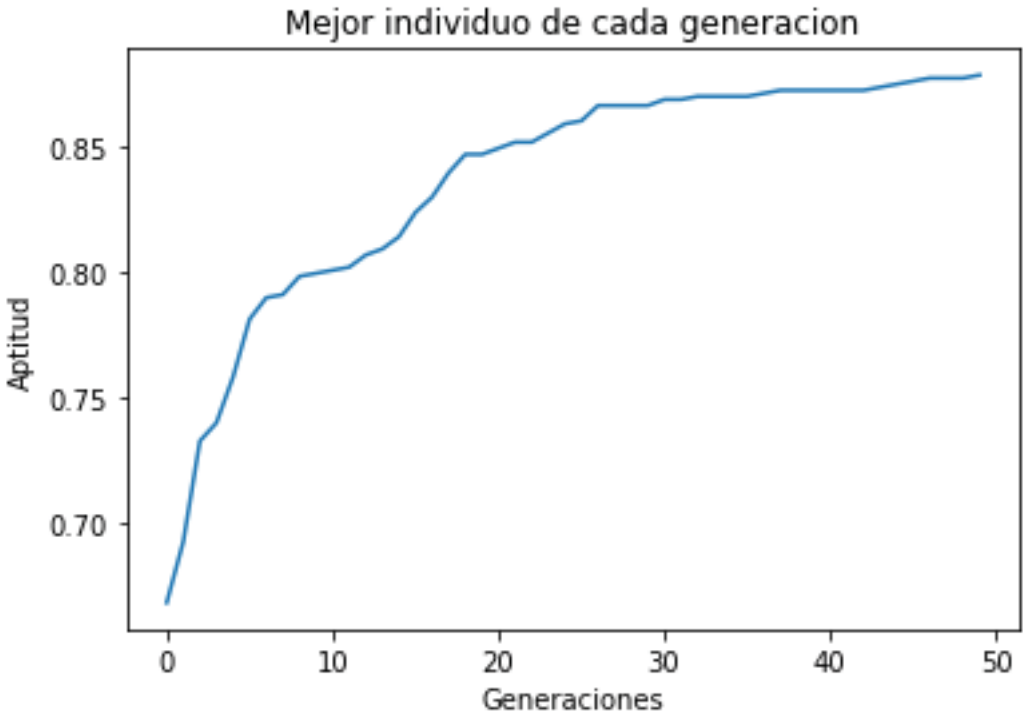

Figura 7.9: Promedio de aptitud de mejor individuo con cruza de 0.9.

La Figura 7.10 muestra la aptitud promedio de toda la población en cada generación.

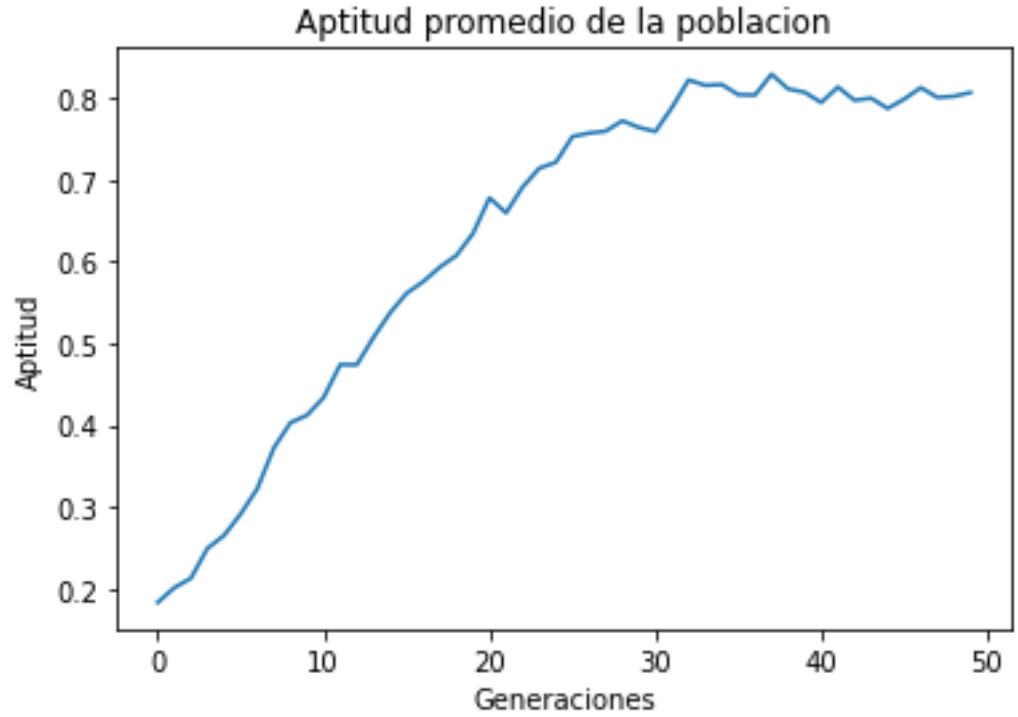

Figura 7.10: Aptitud promedio de la población con cruza de 0.9.

La Tabla 7.8 muestra una comparación del mejor individuo según el porcentaje de cruza.

Tabla 7.8 Mejor individuo respecto al porcentaje de cruza.

Cruza	Mejor aptitud
0.6	0.8612
0.75	0.8830
0.9	0.8781

La Tabla 7.9 muestra la aptitud promedio de la población en la primera y última generación según el porcentaje de cruza, y asi como la mejora de aptitud que hubo entre ellas.

Tabla 7.9 Aptitud promedio de la población respecto al porcentaje de mutación.

Cruza	Aptitud promedio de la población		Mejora
	Primera generación	Última generación	
0.6	0.1910	0.8176	0.6266
0.75	0.1919	0.8185	0.6266
0.9	0.1832	0.8068	0.6236

El porcentaje de cruza que generó al individuo con mayor aptitud fue el de 0.75, y a su vez, generó la mayor aptitud promedio en la población, debido a que se utilizó el mínimo porcentaje de mutación.

7.1.1.3. Combinando parámetros

Al obtener el porcentaje de cruza y mutación que generaban una mayor aptitud de manera individual, se optó por realizar una prueba combinando ambos porcentajes:

Generaciones: 50
Individuos: 50
Porcentaje de mutación: 0.005
Porcentaje de cruza: 0.75

La Tabla 7.10 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.10 Evolución de la aptitud del mejor individuo combinando parámetros.

Generación	Prueba	Aptitud	Promedio
1	1	0.7209	0.7230
	2	0.7363	
	3	0.7036	
5	1	0.8090	0.7787
	2	0.7436	
	3	0.7836	
10	1	0.8127	0.7945
	2	0.7618	
	3	0.8090	
15	1	0.8272	0.8151
	2	0.7945	
	3	0.8236	
20	1	0.8527	0.8418
	2	0.8418	
	3	0.8309	
25	1	0.8527	0.8418
	2	0.8418	
	3	0.8309	
30	1	0.8527	0.8430
	2	0.8418	
	3	0.8345	
35	1	0.86	0.8527
	2	0.8527	
	3	0.8454	
40	1	0.8672	0.8575
	2	0.8527	
	3	0.8527	
45	1	0.8672	0.86
	2	0.8527	
	3	0.86	
50	1	0.8709	0.8660
	2	0.86	
	3	0.8672	

La Figura 7.11 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

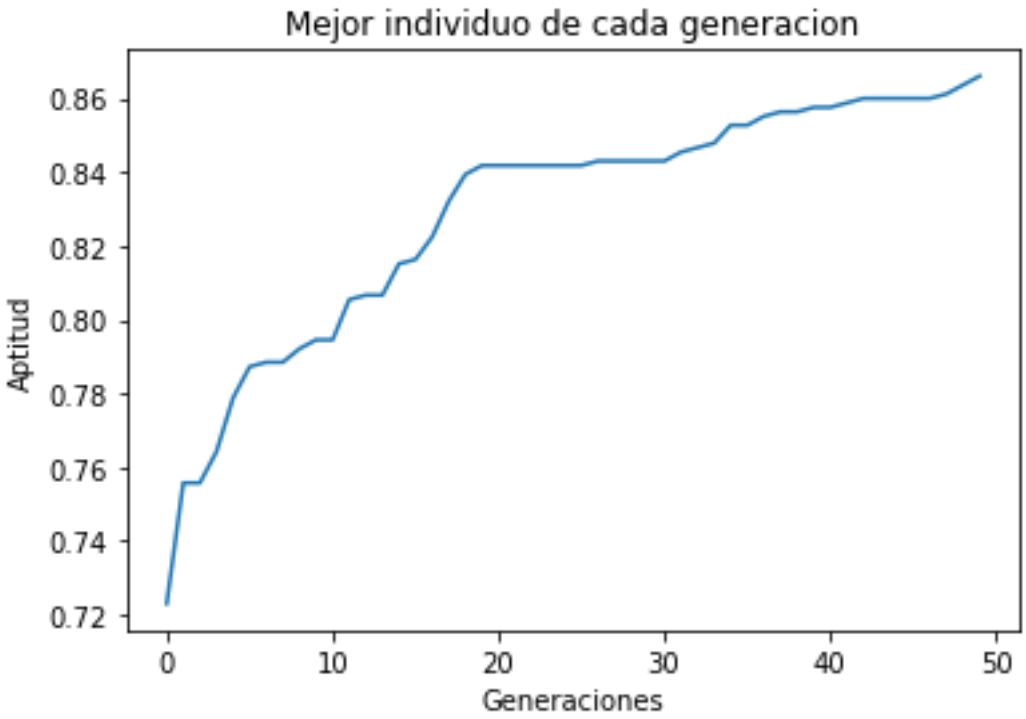

Figura 7.11: Promedio de aptitud de mejor individuo combinando parámetros.

La Figura 7.12 muestra la aptitud promedio de toda la población en cada generación.

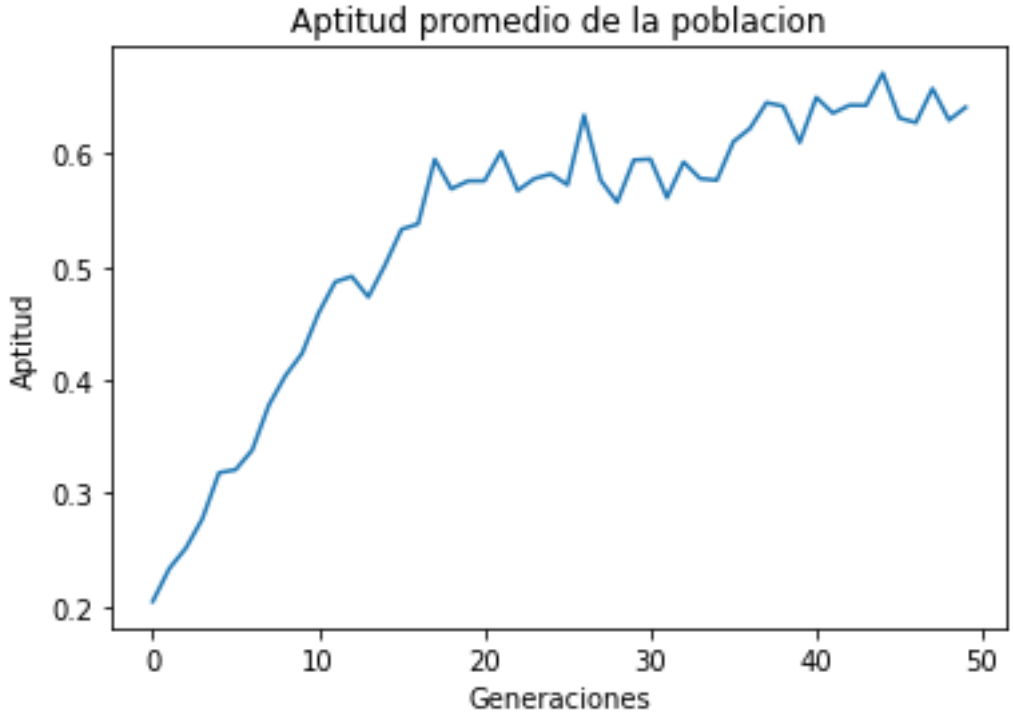

Figura 7.12: Aptitud promedio de la población combinando parámetros.

La Tabla 7.11 muestra una comparación del mejor individuo de las pruebas realizadas al variar la cruza, la mutación y la combinación de ambos parámetros.

Tabla 7.11 Mejores individuos de las pruebas.

Mutación	Cruza	Mejor aptitud
0.005	0.6	0.8745
0.001	0.75	0.8830
0.005	0.75	0.8660

La Tabla 7.12 muestra la aptitud promedio de la población en la primera y última generación de los mejores individuos de las pruebas realizadas al variar la cruza, la mutación, la combinación de ambos parámetros y asi como la mejora de aptitud que hubo entre ellas.

Tabla 7.12 Aptitud promedio de la población de los mejores individuos de las pruebas.

Mutación	Cruza	Aptitud promedio de la población		Mejora
		Primera generación	Última generación	
0.005	0.6	0.1952	0.6561	0.4609
0.001	0.75	0.1919	0.8185	0.6266
0.005	0.75	0.2041	0.6411	0.437

Se concluye que, para el método de cruza en 2 puntos, la combinación de parámetros que generó al individuo con mayor aptitud es con un porcentaje de mutación de 0.001 y un porcentaje de cruza de 0.75.

7.1.2. Cruza en 1 punto

7.1.2.1. Variando porcentajes de mutación

Los cambios en el porcentaje de mutación se realizaron con los siguientes parámetros de control:

Número de generaciones: 50
Número de individuos: 50
Porcentaje de cruza: 0.6

- **Mutación 0.001**

La Tabla 7.13 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.13 Evolución de la aptitud del mejor individuo con mutación de 0.001.

Generación	Prueba	Aptitud	Promedio
1	1	0.7545	0.7339
	2	0.7727	
	3	0.6745	
5	1	0.7909	0.7872
	2	0.8090	
	3	0.7618	
10	1	0.8127	0.8103
	2	0.8345	
	3	0.7836	
15	1	0.8163	0.8139
	2	0.8418	
	3	0.7836	
20	1	0.8345	0.8309
	2	0.8454	
	3	0.8127	
25	1	0.8563	0.8490
	2	0.8490	
	3	0.8418	
30	1	0.86	0.8539
	2	0.86	
	3	0.8418	
35	1	0.86	0.8587
	2	0.8636	
	3	0.8527	
40	1	0.8672	0.8612
	2	0.8636	
	3	0.8527	
45	1	0.8672	0.8636
	2	0.8636	
	3	0.86	
50	1	0.8745	0.8660
	2	0.8636	
	3	0.86	

La Figura 7.13 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

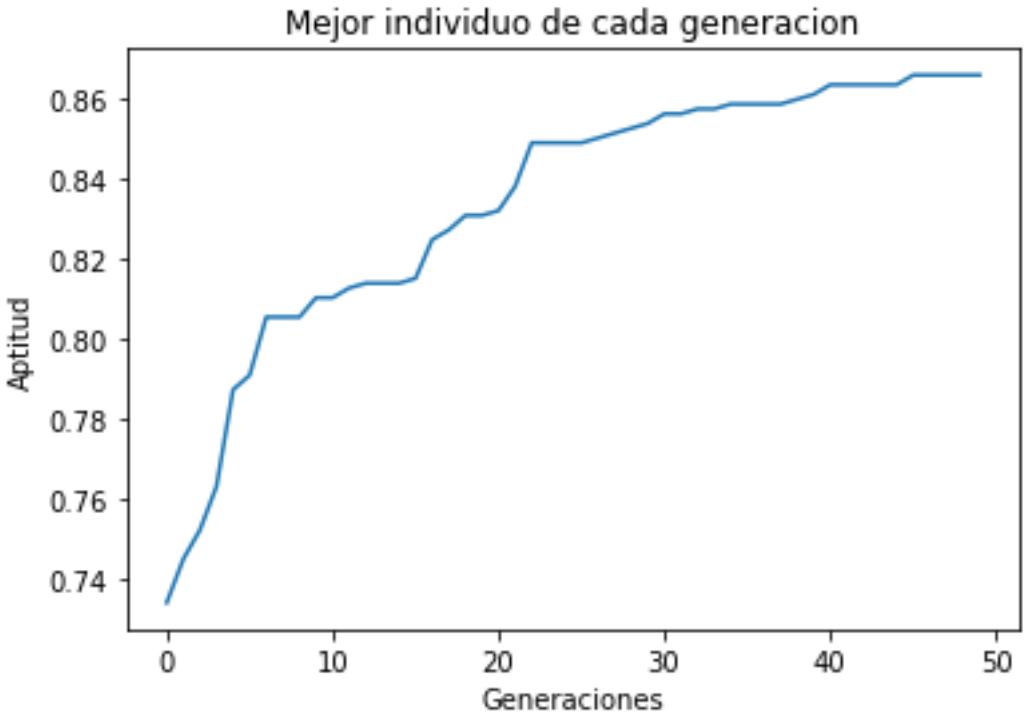

Figura 7.13: Promedio de aptitud de mejor individuo con mutación de 0.001.

La Figura 7.14 muestra la aptitud promedio de toda la población en cada generación.

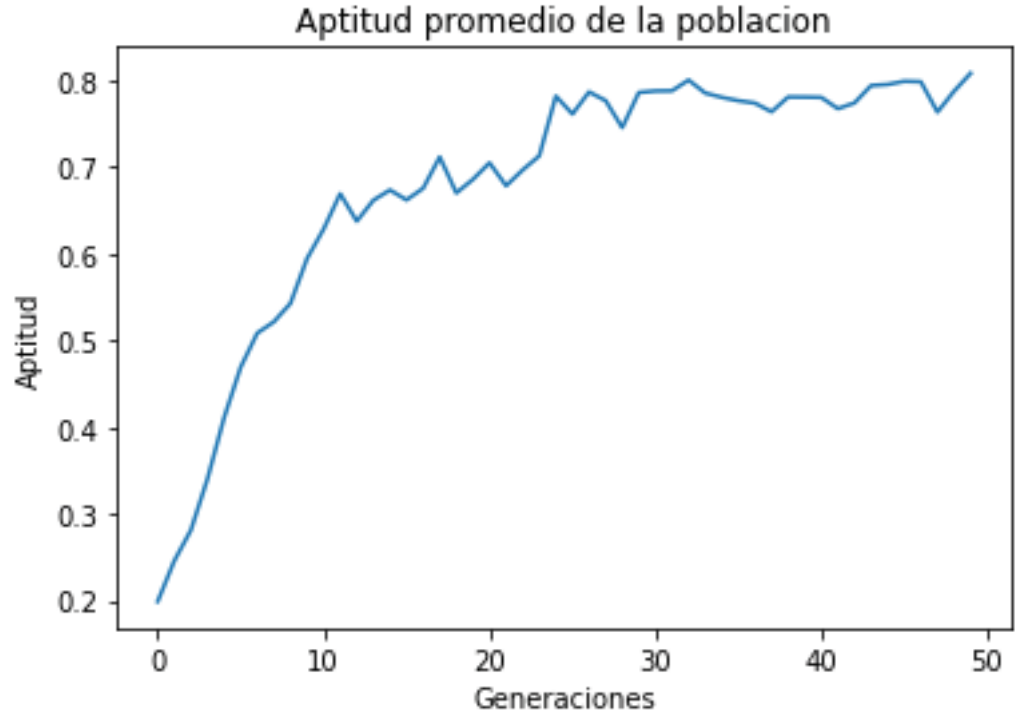

Figura 7.14: Aptitud promedio de la población con mutación de 0.001.

- **Mutación 0.005**

La Tabla 7.14 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.14 Evolución de la aptitud del mejor individuo con mutación de 0.005.

Generación	Prueba	Aptitud	Promedio
1	1	0.7763	0.6975
	2	0.7036	
	3	0.6127	
5	1	0.8163	0.7787
	2	0.7581	
	3	0.7618	
10	1	0.8527	0.8042
	2	0.7981	
	3	0.7618	
15	1	0.8527	0.8321
	2	0.82	
	3	0.8236	
20	1	0.8527	0.8345
	2	0.8236	
	3	0.8272	
25	1	0.86	0.8515
	2	0.8527	
	3	0.8418	
30	1	0.8709	0.8563
	2	0.8563	
	3	0.8418	
35	1	0.8709	0.8587
	2	0.8563	
	3	0.8490	
40	1	0.8709	0.8612
	2	0.8563	
	3	0.8563	
45	1	0.8709	0.8660
	2	0.8563	
	3	0.8709	
50	1	0.8709	0.8660
	2	0.8563	
	3	0.8709	

La Figura 7.15 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

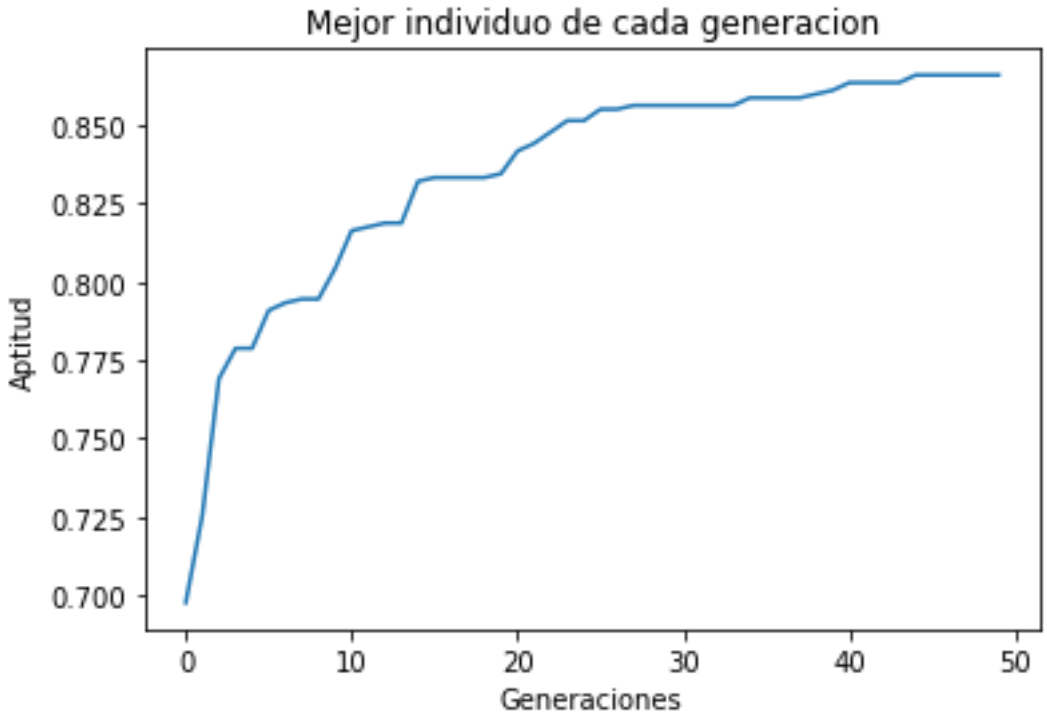

Figura 7.15: Promedio de aptitud de mejor individuo con mutación de 0.005.

La Figura 7.16 muestra la aptitud promedio de toda la población en cada generación.

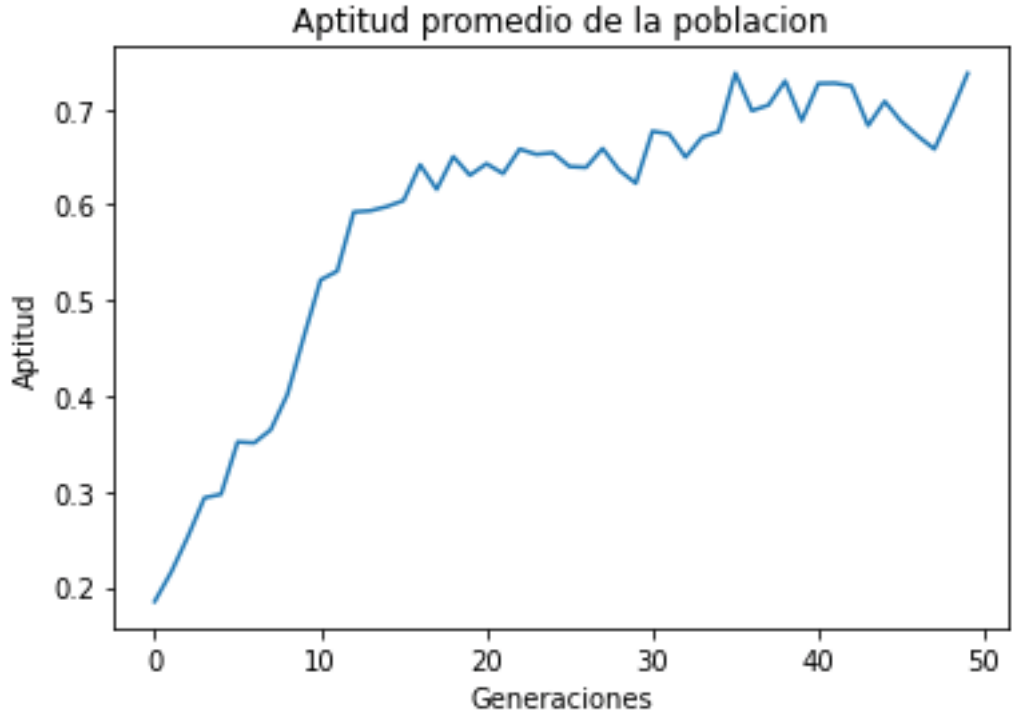

Figura 7.16: Aptitud promedio de la población con mutación de 0.005.

- **Mutación 0.01**

La Tabla 7.15 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.15 Evolución de la aptitud del mejor individuo con mutación de 0.01.

Generación	Prueba	Aptitud	Promedio
1	1	0.5981	0.7012
	2	0.7145	
	3	0.7909	
5	1	0.7509	0.7812
	2	0.7872	
	3	0.8054	
10	1	0.8127	0.8078
	2	0.8018	
	3	0.8090	
15	1	0.86	0.8369
	2	0.8236	
	3	0.8272	
20	1	0.8672	0.8406
	2	0.8272	
	3	0.8272	
25	1	0.8709	0.8515
	2	0.8272	
	3	0.8563	
30	1	0.8709	0.8551
	2	0.8345	
	3	0.86	
35	1	0.8818	0.8672
	2	0.86	
	3	0.86	
40	1	0.8818	0.8709
	2	0.86	
	3	0.8709	
45	1	0.8818	0.8709
	2	0.86	
	3	0.8709	
50	1	0.8818	0.8709
	2	0.86	
	3	0.8709	

La Figura 7.17 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

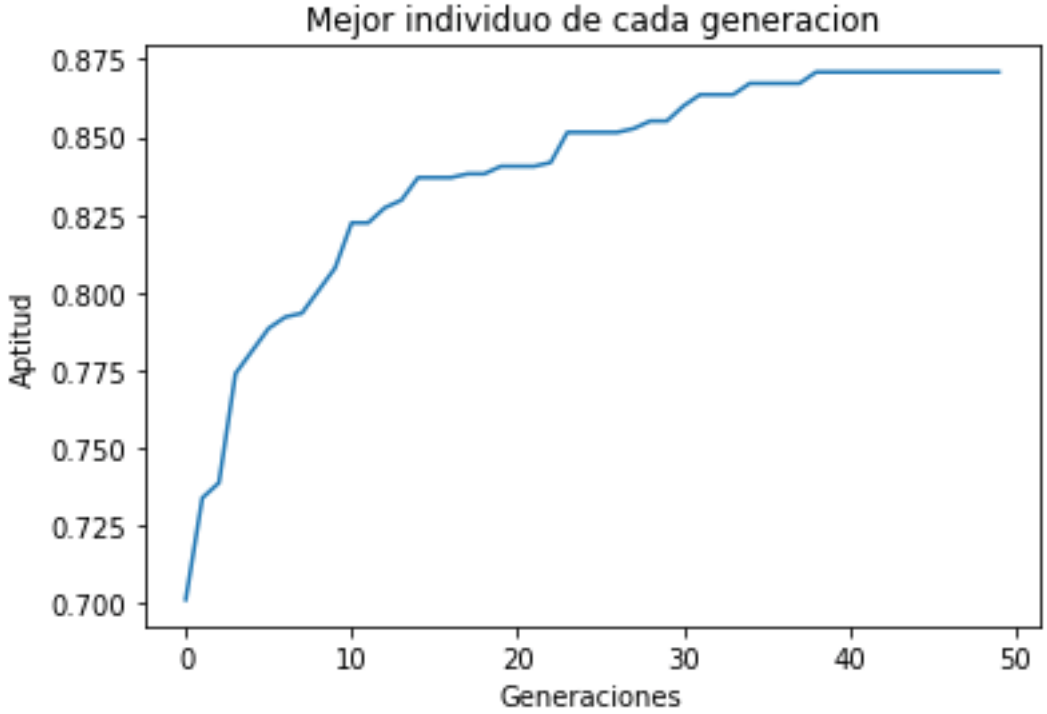

Figura 7.17: Promedio de aptitud de mejor individuo con mutación de 0.01.

La Figura 7.18 muestra la aptitud promedio de toda la población en cada generación.

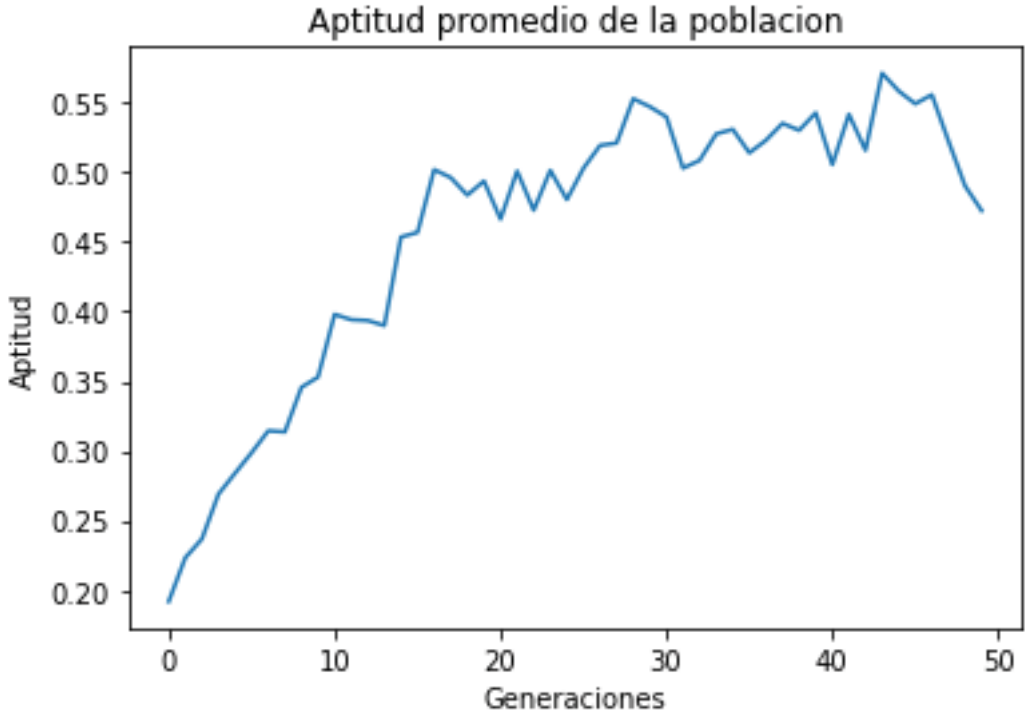

Figura 7.18: Aptitud promedio de la población con mutación de 0.01.

La Tabla 7.16 muestra una comparación del mejor individuo según el porcentaje de mutación.

Tabla 7.16 Mejor individuo respecto al porcentaje de mutación.

Mutación	Mejor aptitud
0.001	0.8660
0.005	0.8660
0.01	0.8709

La Tabla 7.17 muestra la aptitud promedio de la población en la primera y última generación según el porcentaje de mutación, y asi como la mejora de aptitud que hubo entre ellas.

Tabla 7.17 Aptitud promedio de la población respecto al porcentaje de mutación.

Mutación	Aptitud promedio de la población		Mejora
	Primera generación	Última generación	
0.001	0.1989	0.8079	0.609
0.005	0.1849	0.7374	0.5525
0.01	0.1918	0.4725	0.2807

El aumento de mutación afecta en mayor medida al método de cruza en 1 punto, pero a su vez el porcentaje de mutación de 0.01 obtuvo al individuo con mayor aptitud.

7.1.2.2. Variando el porcentaje de cruza

Los cambios en el porcentaje de cruza se realizaron con los siguientes parámetros de control:

Número de generaciones: 50
Número de individuos: 50
Porcentaje de mutación: 0.001

Las pruebas que se realizaron al variar el porcentaje de mutación, se llevaron a cabo con un porcentaje de cruza de 0.60. Debido a esto, las variaciones de cruza iniciaron a partir de 0.75, pero al realizar la comparación de resultados se consideró a la Tabla 7.13

- **Cruza 0.75**

La Tabla 7.18 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.18 Evolución de la aptitud del mejor individuo con cruza de 0.75.

Generación	Prueba	Aptitud	Promedio
1	1	0.7509	0.7060
	2	0.7036	
	3	0.6636	
5	1	0.8272	0.7909
	2	0.7981	
	3	0.7472	
10	1	0.8636	0.8272
	2	0.8054	
	3	0.8127	
15	1	0.8709	0.8430
	2	0.8236	
	3	0.8345	
20	1	0.8709	0.8478
	2	0.8309	
	3	0.8418	
25	1	0.8781	0.8587
	2	0.8454	
	3	0.8527	
30	1	0.8818	0.86
	2	0.8454	
	3	0.8527	
35	1	0.8818	0.8648
	2	0.86	
	3	0.8527	
40	1	0.8818	0.8648
	2	0.8636	
	3	0.86	
45	1	0.8854	0.8733
	2	0.8709	
	3	0.8636	
50	1	0.8854	0.8757
	2	0.8781	
	3	0.8636	

La Figura 7.19 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

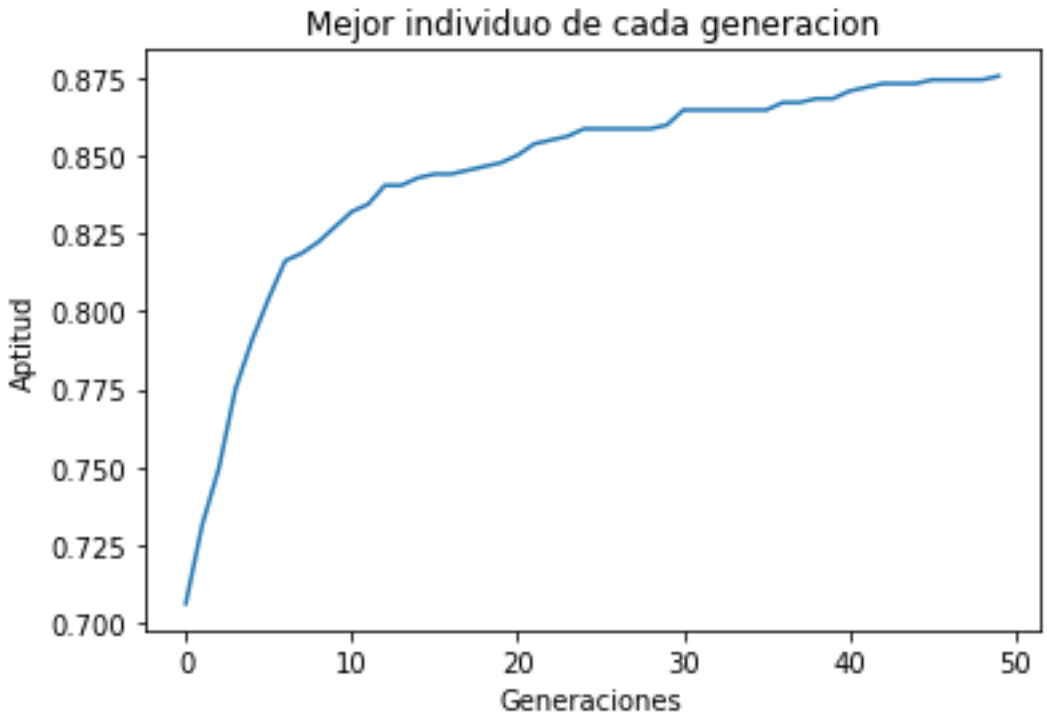

Figura 7.19: Promedio de aptitud de mejor individuo con cruza de 0.75.

La Figura 7.20 muestra la aptitud promedio de toda la población en cada generación.

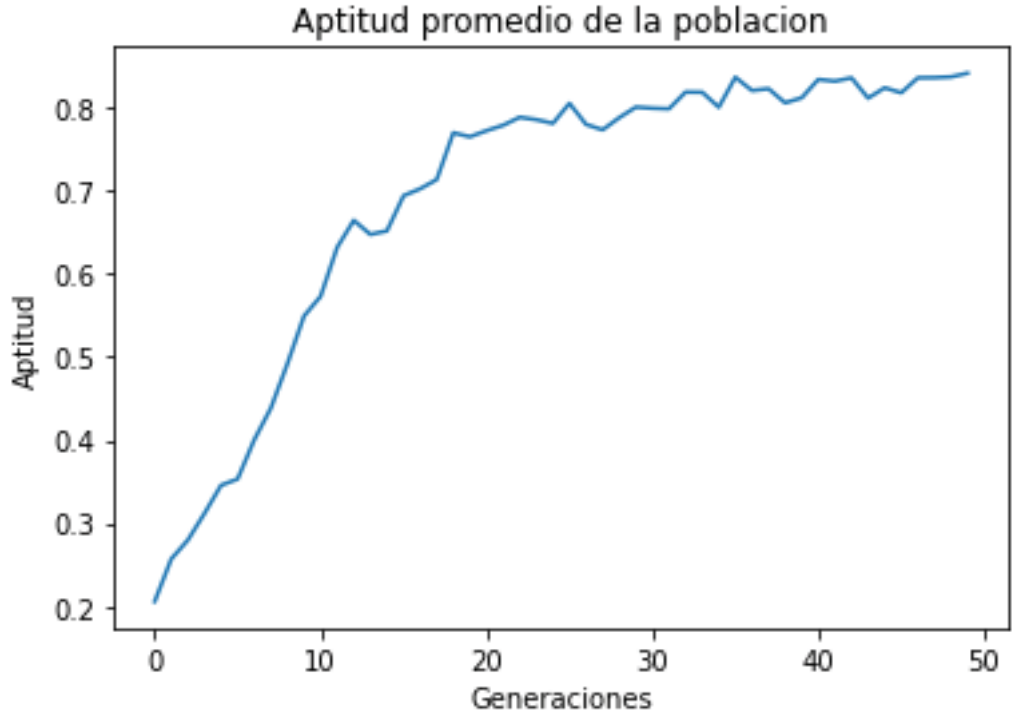

Figura 7.20: Aptitud promedio de la población con cruza de 0.75.

- **Cruza 0.9**

La Tabla 7.19 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.19 Evolución de la aptitud del mejor individuo con cruza de 0.9.

Generación	Prueba	Aptitud	Promedio
1	1	0.5690	0.6393
	2	0.6963	
	3	0.6527	
5	1	0.7727	0.7630
	2	0.7727	
	3	0.7436	
10	1	0.8127	0.8115
	2	0.8236	
	3	0.7981	
15	1	0.8345	0.8369
	2	0.8381	
	3	0.8381	
20	1	0.8490	0.8515
	2	0.8563	
	3	0.8490	
25	1	0.86	0.8587
	2	0.86	
	3	0.8563	
30	1	0.86	0.8624
	2	0.8672	
	3	0.86	
35	1	0.8636	0.8672
	2	0.8745	
	3	0.8636	
40	1	0.8672	0.8709
	2	0.8818	
	3	0.8636	
45	1	0.8672	0.8733
	2	0.8818	
	3	0.8709	
50	1	0.8709	0.8769
	2	0.8854	
	3	0.8745	

La Figura 7.21 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

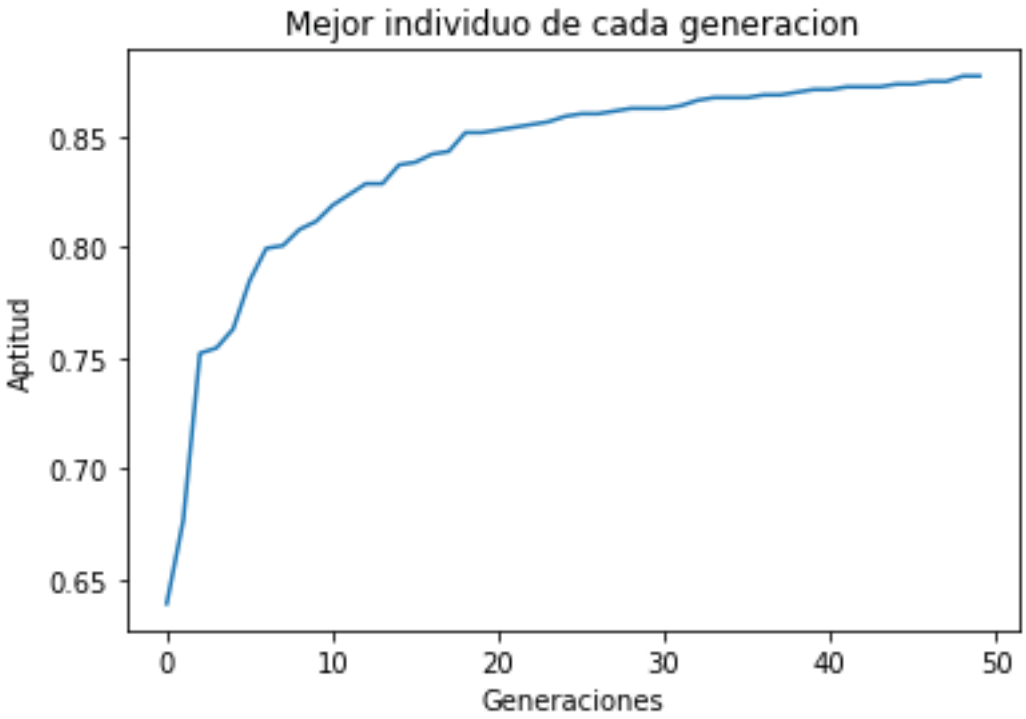

Figura 7.21: Promedio de aptitud de mejor individuo con cruza de 0.9.

La Figura 7.22 muestra la aptitud promedio de toda la población en cada generación.

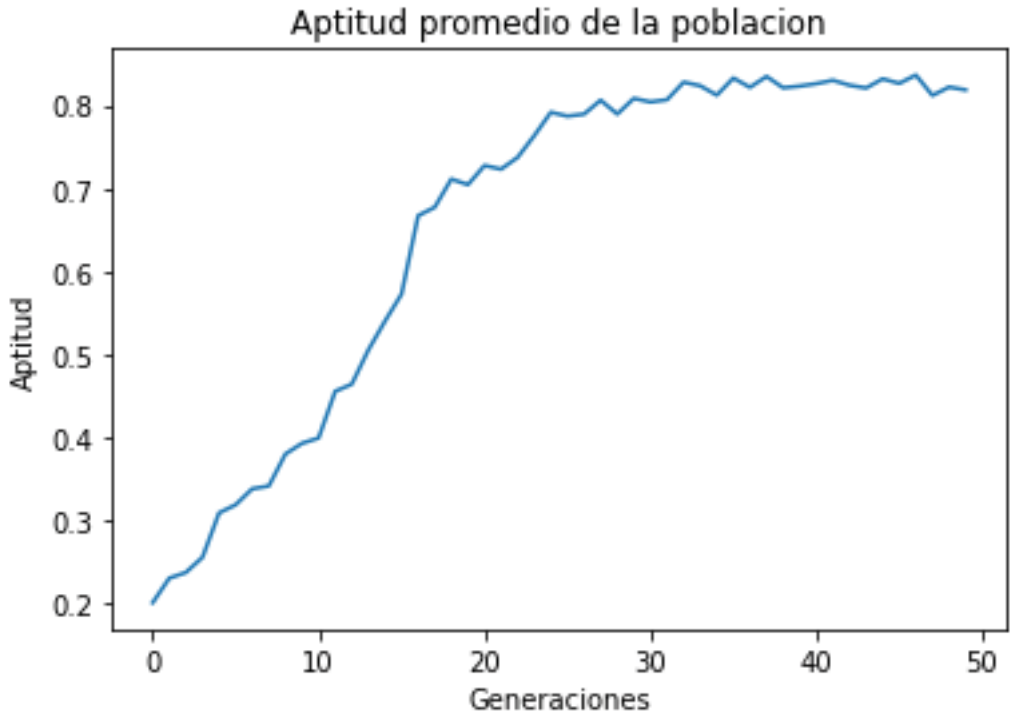

Figura 7.22: Aptitud promedio de la población con cruza de 0.9.

La Tabla 7.20 muestra una comparación del mejor individuo según el porcentaje de cruza.

Tabla 7.20 Mejor individuo respecto al porcentaje de cruza.

Cruza	Mejor aptitud
0.6	0.8660
0.75	0.8757
0.9	0.8769

La Tabla 7.21 muestra la aptitud promedio de la población en la primera y última generación según el porcentaje de cruza, y asi como la mejora de aptitud que hubo entre ellas.

Tabla 7.21 Aptitud promedio de la población respecto al porcentaje de cruza.

Cruza	Aptitud promedio de la población		Mejora
	Primera generación	Última generación	
0.6	0.1989	0.8079	0.609
0.75	0.2057	0.8417	0.636
0.9	0.2004	0.8188	0.6184

La cruza en un punto al tener el mínimo de mutación para las variaciones del porcentaje de cruza, hace que la población en general sea más homogénea. Es por ello que se requiere el mayor porcentaje de cruza para poder realizar la búsqueda en un mayor espacio.

7.1.2.3 Combinando parámetros

Al obtener el porcentaje de cruza y mutación que generaban una mayor aptitud de manera individual, se optó por realizar una prueba combinando ambos porcentajes:

Generaciones: 50
Individuos: 50
Porcentaje de mutación: 0.01
Porcentaje de cruza: 0.9

La Tabla 7.22 muestra la evolución de la aptitud del mejor individuo al paso de las generaciones.

Tabla 7.22 Evolución de la aptitud del mejor individuo combinando parámetros.

Generación	Prueba	Aptitud	Promedio
1	1	0.3054	0.5696
	2	0.6781	
	3	0.7254	
5	1	0.7836	0.7642
	2	0.7290	
	3	0.78	
10	1	0.82	0.8042
	2	0.8018	
	3	0.7909	
15	1	0.8490	0.8333
	2	0.8345	
	3	0.8163	
20	1	0.8527	0.8527
	2	0.8709	
	3	0.8345	
25	1	0.86	0.8587
	2	0.8709	
	3	0.8454	
30	1	0.8672	0.8660
	2	0.8818	
	3	0.8490	
35	1	0.8672	0.8660
	2	0.8818	
	3	0.8490	
40	1	0.8709	0.8684
	2	0.8818	
	3	0.8527	
45	1	0.8709	0.8684
	2	0.8818	
	3	0.8527	
50	1	0.8709	0.8684
	2	0.8818	
	3	0.8527	

La Figura 7.23 muestra una gráfica del promedio de la aptitud del mejor individuo de cada generación.

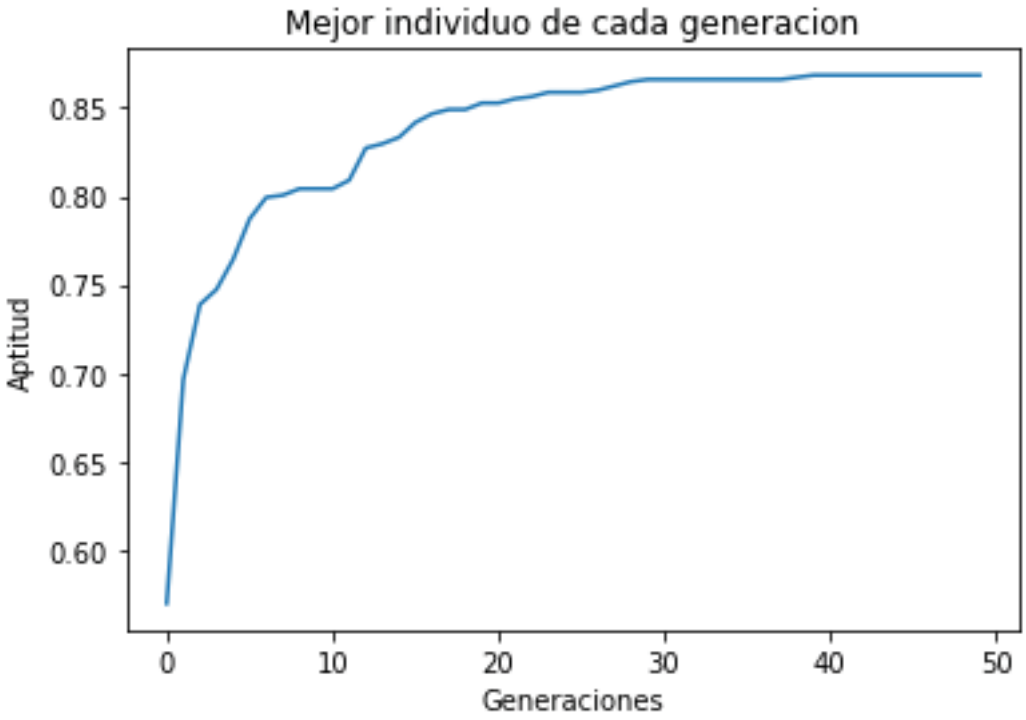

Figura 7.23: Promedio de aptitud de mejor individuo combinando parámetros.

La Figura 7.24 muestra la aptitud promedio de toda la población en cada generación.

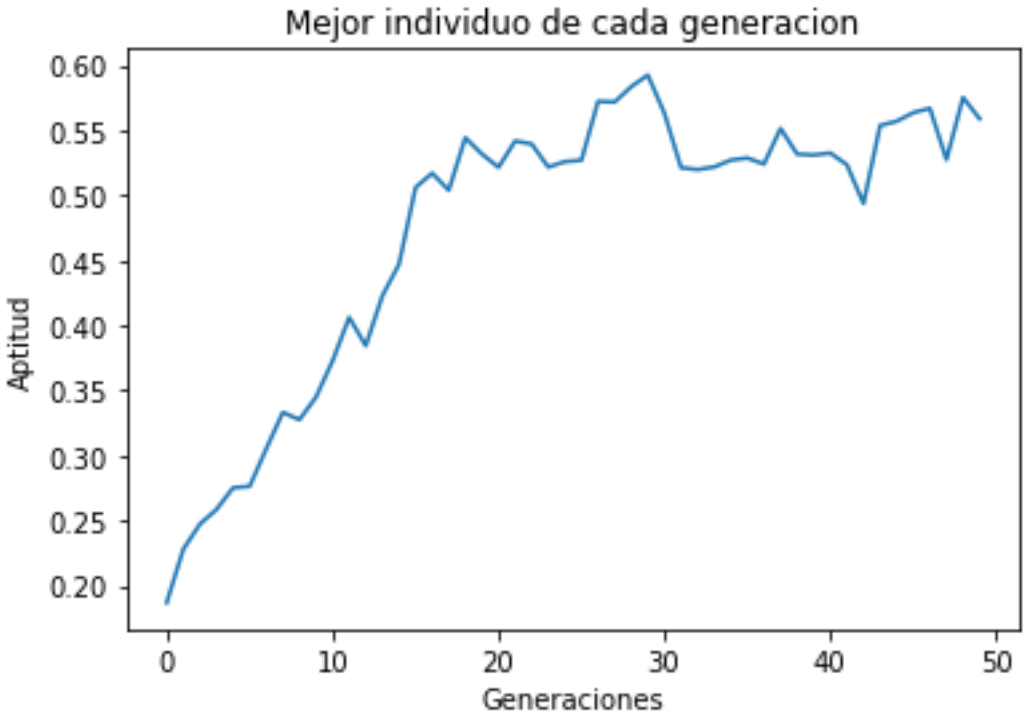

Figura 7.24: Aptitud promedio de la población combinando parámetros.

La Tabla 7.23 muestra una comparación del mejor individuo de las pruebas realizadas al variar la cruza, la mutación y la combinación de ambos parámetros.

Tabla 7.23 Mejores individuos de las pruebas.

Mutación	Cruza	Mejor aptitud
0.01	0.6	0.8709
0.001	0.9	0.8769
0.01	0.9	0.8684

La Tabla 7.24 muestra la aptitud promedio de la población en la primera y última generación de los mejor individuos de las pruebas realizadas al variar la cruza, la mutación, la combinación de ambos parámetros y la mejora de aptitud que hubo entre ellas.

Tabla 7.24 Aptitud promedio de la población de los mejores individuos de las pruebas.

Mutación	Cruza	Aptitud promedio de la población		Mejora
		Primera generación	Última generación	
0.01	0.6	0.1918	0.4725	0.2807
0.001	0.9	0.2004	0.8188	0.6184
0.01	0.9	0.1871	0.5587	0.3716

En la Tabla 7.24 se comprueba que, a mayor porcentaje de mutación, los bloques constructores que generaban una buena aptitud en la mayoría de los casos se perdían.

Se concluye que, para el método de cruza en 1 punto, la combinación de parámetros que generó al individuo con mayor aptitud es con un porcentaje de mutación de 0.001 y un porcentaje de cruza de 0.9.

7.2 Interpretación de resultados

La obtención de la aptitud de cada individuo requirió un tiempo en promedio de 7.2 segundos, por lo que, al tener una población de 50 individuos, cada generación tomó 6 minutos. Por lo tanto, realizar una prueba de 50 generaciones requirió un tiempo total de 5 horas.

Obtener el promedio de aptitud de la población permitió visualizar la forma en que la mutación altera en gran medida a los bloques constructores que generaban una buena aptitud. Debd tenerse en mente que entre mayor sea el porcentaje de mutación, aumenta la probabilidad de disminuir o afectar negativamente a la aptitud. Sin embargo, esto a su vez permitió probar aquellas

combinaciones que un aumento en el porcentaje de cruza no habría podido lograr.

De igual manera, la aptitud promedio de la población permitió ver cómo influye el método de cruza que se esté utilizando ya que, en el caso del método de cruza en 1 punto la población se hacía homogénea con mayor facilidad. Esto puede ser contraproducente si no se encuentra una buena aptitud, ya que las variaciones de padres a hijos son muy pocas, y esto a su vez retrasa en gran medida la evolución de los individuos. Pero en el caso de obtener una buena aptitud, el método de cruza en 1 punto permite mejorar la aptitud de los individuos apenas afectando a los bloques constructores, ya que muchas veces para incrementar la aptitud de un individuo, solo se requieren ligeros cambios.

Debido a esto, los mejores resultados con el método de cruza en 1 punto fueron con el mínimo porcentaje de mutación (0.001) y el máximo porcentaje de cruza probada (0.9). Esto permitió cruzar a la mayoría de los individuos creando ligeros cambios en su aptitud, sin interferir con grandes porcentajes de mutación que afectarían los resultados, y que para este método solo retrasarían el aprendizaje.

El método de cruza en 2 puntos permite tener un mayor espacio de búsqueda del mejor individuo, debido a que hay mayor cantidad de combinaciones. Esto en algunos casos puede ser contraproducente, ya que demasiada variedad puede perder la aptitud de un buen individuo. Es por ello que la combinación de porcentaje de mutación de 0.001 y el porcentaje de cruza de 0.75 fueron los que dieron mejores resultados. Esto se debe a que un alto porcentaje de cruza hace que se pierda con mayor facilidad una buena aptitud y un porcentaje de 0.6 no permite realizar una buena busqueda. Y un mínimo porcentaje de mutación permitió crear aquellas combinaciones que no se lograrían con un porcentaje de cruza de 0.75.

Debido a que un individuo contiene un ciclo de marcha, que equivalen a 10 pasos, un ligero cambio puede afectar drásticamente a la aptitud de un individuo positivamente o negativamente. Es por ello que las combinaciones de parámetros de cruza en 1 punto y 2 puntos, anteriormente mencionadas, dieron los mejores resultados. Esto se debe a que realizar ligeros cambios en el individuo en el mayor espacio de búsqueda que fue posible, afectaba en menor medida a la aptitud promedio de la población, permitiendo a su vez, obtener a los individuos con mayores aptitudes. La Tabla 7.25 muestra el valor máximo de aptitud que se obtuvo en cada método.

Tabla 7.25 Mejores combinaciones de cada método.

Método	Porcentaje de mutación	Porcentaje de cruza	Aptitud
Cruza en 1 punto	0.001	0.9	0.8769
Cruza en 2 puntos	0.001	0.75	0.8830

El número de individuos lo que hace es aumentar la probabilidad de que haya una mejor aptitud, permitiendo crear más combinaciones. El número de generaciones permite crear nuevas combinaciones a partir de la población anterior. Aumentar el número de individuos no beneficia tanto al desempeño del algoritmo genético como lo hace el número de generaciones.

Debido a esto, se contempló aumentar el número de generaciones hasta obtener la combinación de parámetros que diera mejores resultados. En el siguiente enlace se muestra un video de la evolución de la marcha a lo largo de 100 generaciones con el método que genero una mayor aptitud.

En el video se muestra, cada 10 generaciones a algunos individuos obteniendo su valor de aptitud. Al final de cada generación se visualiza la marcha del mejor individuo de esa población. Para las generaciónes 80 y 90 no se muestra al mejor individuo debido a que se mantiene el mismo individuo desde la generación 70.

https://youtu.be/IKREBqRdiag

En el siguiente enlace se muestra un video de los individuos que lograron los mejores movimientos de todas las pruebas realizadas.

https://youtu.be/6NAk1ZfinpI

CAPITULO 8. DISCUSION

La robótica evolutiva tiene un largo camino por recorrer. Los proyectos que se mencionan en el estado del arte son inicios de un área que está en actual crecimiento. El presente proyecto representó un pequeño avance para la biorobótica evolutiva, que es la rama que busca generar movimientos similares a seres vivos, como en el caso de Long [11] quien trabajó con colas nadadoras y la evolución de su movimiento. El robot cuadrúpedo que se eligió para el proyecto logró generar movimientos similares a los de la Marcha Ideal, pero para obtener mejores resultados se requiere de más estudio del movimiento y tal vez de otras formas de analizar el problema.

Trabajos como el de Jiménez [17] y el de García [16] tambien utilizaron robots cuadrúpedos para aprender a desplazarse utilizando métodos de la computación evolutiva. A diferencia del presente proyecto, Jiménez se centró en realizar un avance simple con un robot de dos grados de libertad en cada extremidad utilizando una codificación decimal y diferentes métodos de cruza y mutación, que le dieron resultados satisfactorios.

Dado que en el presente proyecto el robot cuadrúpedo cumplió con el objetivo de aprender a desplazarse adecuadamente con una codificación binaria y usando distintos métodos de cruza y mutación a los utilizados por Jiménez, se concluye que los algoritmos genéticos son viables para la creación de robots más independientes.

CAPITULO 9. CONCLUSIONES

El algoritmo genético que se desarrolló para el presente proyecto, dio resultados satisfactorios debido a una adecuada evaluación de cada individuo. Tener una buena función de aptitud requirió un claro entendimiento del problema y un buen conocimiento del robot cuadrúpedo que se estaba utilizando.

Debido a esto, sin importar el método de cruza que se haya probado los resultados fueron aceptables. Es por ello, que fue necesario realizar un análisis sobre las combinaciones de parámetros que generaban un mejor desempeño del algoritmo genético.

Y aunque no se llegó a un movimiento lo suficientemente similar al de un can, se comprueba su efectividad para aprender a realizar movimientos más simples.

Sin embargo, aunque se hayan obtenido valores satisfactorios de los métodos probados y la manera de analizar un ciclo de marcha, no se descarta la idea de que haya otras formas de obtener mejores resultados. Sin embargo, debido a los alcances del presente proyecto, se concluye que se obtuvo la mejor solución posible con el tiempo establecido y los conocimientos obtenidos a lo largo del proceso del proyecto.

CAPITULO 10. SUGERENCIAS PARA INVESTIGACIONES FUTURAS

Partiendo de los resultados obtenidos en el presente Trabajo Terminal, se han visualizado las siguientes líneas de actuación, que podrían mejorar o ampliar los alcances del proyecto:

- Realizar el proyecto en un programa de simulación diferente para hacer una comparación de tiempos respecto a CoppeliaSim y definir cuál es más eficiente. Los tiempos y la forma de obtener los datos del simulador son fundamentales para un óptimo desempeño del algoritmo genético.

- Cambiar la codificación de los individuos, por ejemplo, a una representación decimal, lo que conlleva realizar pruebas con métodos distintos de selección, cruza y mutación. Realizar comparaciones con el algoritmo genético con codificación binaria y definir qué tipo de representación es más eficaz.

- Derivar la función de aptitud con otro método de análisis.

- Probar distintos métodos de la computación evolutiva, tales como la programación genética o las estrategias evolutivas. Esto con el fin de verificar qué método es más eficiente en robots que aprenden a desplazarse.

- El proyecto se desarrolló en un ambiente de simulación plano y sin obstáculos. Se podrían realizar cambios en el ambiente para poder ver cómo se adaptaría a lugares cambiantes y con obstáculos.

- Comprobar la eficiencia de los algoritmos evolutivos cuando el robot cuadrúpedo tiene fallas en alguna de sus extremidades y ver si es capaz de adaptarse a dicho cambio y seguir desplazándose.

CAPITULO 11. BIBLIOGRAFIA

[1] D. B. Fogel, “What is evolutionary computation?”, *IEEE SPECTRUM,* vol. 37, pp. 26-32, February 2000.

[2] D. Goldberg, *Genetic algorithms in search, optimization and machine learning,* Addison-Wesley Publishing Company, 1989.

[3] C. Darwin, *On the origin of species by means of natural selection, or the preservation of favored races in the struggle for life*, New York: D. Appleton and Company,1859.

[4] C. A. Coello, “Introducción a los Algoritmos Genéticos”, *Soluciones Avanzadas. Tecnologías de Información y Estrategias de Negocios.* No. 17, pp. 5-11, enero 1995.

[5] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press: Ann Arbor, 1975.

[6] A. Minguez, “Ingeniería Avanzada para Sistemas de Control de Ruido Acústico mediante Técnicas Adaptativas”, tesis doctoral, Departamento de señales, sistemas y radiocomunicaciones, UPM, España, Madrid, 1998.

[7] M. Srinivas, M. Lalit. “Genetic Algorithms: A Survey”, IEEE Computer, pp. 17-26, Jun 1994.

[8] J. E. Gregory, Rawlins, *Fundations of Genetic Algorithms,* Morgan Kaufmann Publishers, 1991.

[9] Whitley, L. Darrell, *Fundations of Genetic Algorithms 2,* Morgan Kaufmann Publishers, 1993.

[10] G. J. Friedman, “Slective Feedback Computers for Engineering Synthesis and Nervous System Analogy”, M.S. thesis, University of California, Los Angeles, February 1956.

[11] J. Long, *Darwin’s devices: What evolving robots can teach us about the history of life and the future of tecnology*, basic books, 2012.

[12] O. Miglino, H. H. Lund, M. Cardaci, “La robótica como herramienta para la educación”, Departmente of Computer Science, pp. 1-22, 1995.

[13] L. Spector, S. Luke, *Evolving teamwork and coordination with genetic programming,* Proceedings of the First Annual Conference on Genetic Programming, MIT Press, Cambridge, 1996.

[14] M. Becerra, “Diseño y simulación de un robot modular autoconfigurable”, tesis de licenciatura, Academia de mecatrónica, IPN-UPIITA, CDMX, 2008
[15] Meng. Y, Zhang. Y, Jin. Y, “Modular Self-Reconfigurable Robot Systems”, *IEEE Robotics and Automation Magazine,* vol. 6, pp. 43-52, March 2007.
[16] C. García, Bellerin, “Diseño e implementación de un cuadrúpedo que “aprende” a caminar mediante algoritmos genéticos”, tesis de licenciatura, Memoria Laboratorio Robótica, UPM, España, Madrid, 2013.
[17] A. L. Jiménez, “Sistema de aprendizaje evolutivo para un robot caminante”, tesis de licenciatura, Departamento de automática, ingeniería electrónica e informática industrial, UPM, España, Madrid, 2015.
[18] J. S. Pantoja, M.G. Villareal, “Síntesis optima de un mecanismo para la marcha bípeda utilizando evolución diferencial”, Revista internacional de métodos numéricos para calculo y diseño en ingeniería, pp. 16, abril 2016.
[19] M. Gestal, D. Rivero, J.R. Rabuñal, J. Dorado, A. Pazos, *Introducción a los algoritmos genéticos y la programación genética*. Madrid: Universidade da Coruña, 2010.
[20] “Introducción a la computación evolutiva”, Departamento de Computación, CINVESTAV, primavera 2020.
[21] A. K. De Jong, “An Analysis of the Behavior of a Class of Genetic Adaptative Systems”, PhD thesis, University of Michigan, 1975.
[22] Günter Rudolph, Convergence Analysis of Canonical Genetic Algorithms, *IEEE Transactions on Neural Networks,* 5:96–101, January 1994.
[23] B. Lopez, J. Mendoza, “Robot cuadrúpedo inspirado en la marcha de un can”, tesis de licenciatura, Academia de biónica, IPN-UPIITA, CDMX, 2017.
[24] GrabCAD (s.f.). GRABCAD. [Online]. Available: https://grabcad.com/
[25] ROS (s.f.). SolidWorks to URDF Exporter. [Online]. Available: http://wiki.ros.org/sw_urdf_exporter
[26] CoppeliaSim (s.f.). Shape Dynamics properties. [Online]. Available: https://www.coppeliarobotics.com/helpFiles/en/shapeDynamicsProperties.htm
[27] CoppeliaSim (s.f.). Building a clean model tutorial. [Online]. Available: https://www.coppeliarobotics.com/helpFiles/en/buildingAModelTutorial.htm
[28] CoppeliaRobotics. (s.f.). Remote API Functions (Python). [Online]. Available: https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm
[29] CoppeliaRobotics. (s.f.). Enabling the remote API – server side. [Online]. Available:
https://www.coppeliarobotics.com/helpFiles/en/remoteApiServerSide.htm
[30] CoppeliaRobotics. (s.f.). Enabling the remote API – client side. [Online]. Available:
https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm
[31] J. Mendoza. (2017, septiembre 28). Quadruped robot. [Online]. Available: https://www.youtube.com/watch?v=fqLrywuhdVY&feature=youtu.be&fbclid=IwAR1SaV_KTVtiZavcVuWEndtZ1KDuLkDDecez3zmBXCu4PVsCXJGw5oR8p

ANEXO I. Marcha Ideal

Posiciones	PASOS									
	1	2	3	4	5	6	7	8	9	10
Articulacion11	0	0	-15	-15	-15	-25	-25	-25	-25	-10
Articulacion12	30	30	25	25	25	35	10	20	50	40
Articulacion13	-5	-5	-5	5	5	15	15	-30	-60	-15
Articulacion21	-25	-25	-25	-25	-10	0	0	-15	-15	-15
Articulacion22	35	10	20	50	40	30	30	25	25	25
Articulacion23	15	15	-30	-60	-15	-5	-5	-5	5	5
Articulacion31	0	0	5	15	45	45	15	10	5	5
Articulacion32	-25	-25	-20	-40	-60	-45	-15	-10	-10	-10
Articulacion33	25	25	5	5	-10	0	10	0	10	10
Articulacion41	45	15	10	5	5	0	0	5	15	45
Articulacion42	-45	-15	-10	-10	-10	-25	-25	-20	-40	-60
Articulacion43	0	10	0	10	10	25	25	5	5	-10

ANEXO II. Código de Marcha Ideal

```
Import sim

motor11=[0, 0, -15, -15, -15, -25, -25, -25, -25, -10]
motor12=[30, 30, 25, 25, 25, 35, 10, 20, 50, 40]
motor13=[-5, -5, -5, 5, 5, 15, 15, -30, -60, -15]
motor21=[-25, -25, -25, -25, -10, 0, 0, -15, -15, -15]
motor22=[35, 10, 20, 50, 40, 30, 30, 25, 25, 25]
motor23=[15, 15, -30, -60, -15, -5, -5, -5, 5, 5]
motor31=[0, 0, 5, 15, 45, 45, 15, 10, 5, 5]
motor32=[-25, -25, -20, -40, -60, -45, -15, -10, -10, -10]
motor33=[25, 25, 5, 5, -10, 0, 10, 0, 10, 10]
motor41=[45, 15, 10, 5, 5, 0, 0, 5, 15, 45]
motor42=[-45, -15, -10, -10, -10, -25, -25, -20, -40, -60]
motor43=[0, 10, 0, 10, 10, 25, 25, 5, 5, -10]

#Inicia programa y se abre conexión

print("Iniciando...")
sim.simxFinish(-1)
clientID=sim.simxStart('127.0.0.1',19997,True,True,2000,5) #Se conecta con CoppeliaSim
if (clientID == 0):
    print("Conectado al servidor API remoto correctamente")
    joints={}
    for i in range (1,5):
        for j in range (1,4):
            jointname="articulacion{}{}".format(i,j)
            joint="art{}{}".format(i,j)
            err1,joints[joint]=sim.simxGetObjectHandle(clientID, jointname,    sim.simx_opmode_oneshot_wait)

     #Python reconoce el conjunto de piezas del simulador

    #---------------------------------------------------------------------------------------------------------------
    err1,body=sim.simxGetObjectHandle(clientID, "Cuerpo1", sim.simx_opmode_oneshot_wait)
    err1,pata1=sim.simxGetObjectHandle(clientID, "Pata13", sim.simx_opmode_oneshot_wait)
    err1,pata2=sim.simxGetObjectHandle(clientID, "Pata23", sim.simx_opmode_oneshot_wait)
    err1,pata3=sim.simxGetObjectHandle(clientID, "Pata33", sim.simx_opmode_oneshot_wait)
    err1,pata4=sim.simxGetObjectHandle(clientID, "Pata43", sim.simx_opmode_oneshot_wait)
    err1,ref=sim.simxGetObjectHandle(clientID, "Referencia", sim.simx_opmode_oneshot_wait)
    #---------------------------------------------------------------------------------------------------------------

    pasos = 0 #Inicializa el numero de pasos
    marcha = 0 #Inicializa cantidad de ciclos de marcha que se ejecutaran

    while (marcha<3):
        while (pasos<10):

            #Avanzan todas las patas para realizar movimiento

            #---------------------------------------------------------------------------------------------------------------------------------------------------
            sim.simxPauseCommunication(clientID, 1) #Se pausa la comunicacion para poder mover varias juntas a la vez
            sim.simxSetJointTargetPosition(clientID, joints['art11'], motor11[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art12'], motor12[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art13'], motor13[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art21'], motor21[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art22'], motor22[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art23'], motor23[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art31'], motor31[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art32'], motor32[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art33'], motor33[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art41'], motor41[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art42'], motor42[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art43'], motor43[pasos]*np.pi/180, sim.simx_opmode_oneshot)
            sim.simxPauseCommunication(clientID, 0) #Se reestablece la comunicacion
            #---------------------------------------------------------------------------------------------------------------------------------------------------

            #Pregunta si cada articulación ha llegado a la posición deseada

            #---------------------------------------------------------------------------------------------------------------
            err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_streaming)
            while True:
                err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_buffer)
```

```
    if (abs(v-1*(motor11[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor12[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor13[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor21[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor22[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor23[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor31[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor32[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor33[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor41[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor42[pasos]*np.pi/180))<0.5*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_buffer)
    if (abs(v-1*(motor43[pasos]*np.pi/180))<0.5*np.pi/180):
        break

pasos=pasos+1
#-----------------------------------------------------------------------------------------------------------------
```

```
#Regresa a posiciones iniciales

#-----------------------------------------------------------------------------------------------------------------------
sim.simxPauseCommunication(clientID, 1)
sim.simxSetJointTargetPosition(clientID, joints['art11'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art12'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art13'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art21'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art22'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art23'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art31'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art32'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art33'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art41'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art42'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art43'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxPauseCommunication(clientID, 0)

err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break
```

```
    err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    #Se cierra conexion
    sim.simxFinish(clientID)
    #-----------------------------------------------------------------------------------------------------------------


else:
    print("No se pudo conectar")
```

ANEXO III. Orientaciones de la base del robot.

Para obtener las orientaciones en cada paso, se realizaron varias ejecuciones de la Marcha Ideal y se obtuvo el promedio.

Base del robot	Orientaciones (°)		
	Alpha	**Beta**	**Gamma**
Paso 1	1.41904	-0.23466	-7.57158
Paso 2	1.83842	1.78546	-5.36772
Paso 3	1.29136	0.0394	-6.3765
Paso 4	1.63542	-0.21044	-5.10194
Paso 5	2.00308	-0.91308	-4.9984
Paso 6	1.47716	-0.0551	-3.03788
Paso 7	1.51194	-1.9718	-5.2365
Paso 8	1.26018	-0.27864	-4.417
Paso 9	1.61158	-0.13376	-5.67988
Paso 10	2.11454	0.50416	-5.7717

ANEXO IV. Posiciones de la base del robot en el eje “z”.

Para obtener las posiciones del eje “z” en cada paso, se realizaron varias ejecuciones de la Marcha Ideal y se obtuvo el promedio.

Base del robot	Posiciones (m)
	Eje “z”
Paso 1	0.2262
Paso 2	0.2296
Paso 3	0.2359
Paso 4	0.2345
Paso 5	0.2345
Paso 6	0.2263
Paso 7	0.2295
Paso 8	0.2359
Paso 9	0.2345
Paso 10	0.2345

ANEXO V. Algoritmo genético

```
import random
import math
import sim
import numpy as np
import time

#Programa principal
#-----------------------------------------------------------------------------------------------------------
def AlgoritmoGenetico(Generaciones, Individuos, Pc, Pm):
    Genes = 120
    Elitismo = 0
    MejorMovimiento = 0
    Generacion = 0
    Poblacion = PoblacionInicial(Individuos, Genes)

    while (Generacion < Generaciones or Elitismo >= 1):
        print("Inicia generacion numero: ", Generacion + 1)
        MejorAptitud, Poblacion, MejorIndividuo, Aptitud = Seleccion(Poblacion, Genes)
        if (MejorAptitud > Elitismo):
            Elitismo = MejorAptitud
            MejorMovimiento = MejorIndividuo
        Generacion = Generacion + 1
        Poblacion = Cruza(Poblacion, Genes, Pc, Pm, MejorMovimiento)

    print("El algoritmo genetico ha finalizado")
    print("Aptitud de mejor individuo", Elitismo)
    Marcha(MejorMovimiento, Genes)  #Se muestra mejor movimiento


#Población Inicial
#-----------------------------------------------------------------------
def PoblacionInicial(Individuos, Genes):
    Poblacion = []
    individuo = []
    for i in range (Individuos):
        for j in range (Genes):
            Aleatorio = random.randint(-60, 59) #7 Bits
            individuo.append(Aleatorio)
        Poblacion.append(individuo)
        individuo = []
    return Poblacion


#Selección de individuos por torneo determinístico
#-----------------------------------------------------------------------------------------------
def Seleccion(Poblacion, Genes):
    NuevaPoblacion = []
    MejorAptitud = 0
    MejorIndividuo = 0
    Individuos = len(Poblacion)

    #Se obtiene aptitud de la poblacion
    Aptitud = EvaluarAptitud(Poblacion, Genes)

    for i in range(Individuos):
        if (MejorAptitud < Aptitud[i]):
            MejorAptitud = Aptitud[i]
            MejorIndividuo = Poblacion[i]
        individuo1 = random.randint(0, Individuos-1)
        individuo2 = random.randint(0, Individuos-1)

        if (Aptitud[individuo1] < Aptitud[individuo2] ):
            NuevaPoblacion.append(Poblacion[individuo2])
        else:
            NuevaPoblacion.append(Poblacion[individuo1])

    Poblacion = NuevaPoblacion
    return MejorAptitud, Poblacion, MejorIndividuo, Aptitud
```

```
#Obtención de aptitud de la población
#-------------------------------------------------------------------------------------------------------------
def EvaluarAptitud(Poblacion, Genes):
    Individuos = len(Poblacion)
    Aptitud = []
    for i in range(Individuos):
        Posiciones = Poblacion[i]

        #Obtencion de datos para evaluar aptitud
        Zeta, Alpha, Beta, Gamma, AuxBody = Marcha(Posiciones)

        ZetaIndividuo = Zeta
        AlphaIndividuo = Alpha
        BetaIndividuo = Beta
        GammaIndividuo= Gamma
        Promedio = ZetaIndividuo + AlphaIndividuo + BetaIndividuo + GammaIndividuo
        Normalizacion = ((Promedio - 110) / -110)*0.5
        if (GammaIndividuo>0 or ZetaIndividuo>0):
            Normalizacion = Normalizacion * 0.8
        if (AuxBody == 0):
            Normalizacion = Normalizacion + 0.5
        Aptitud.append(Normalizacion)

    return Aptitud


#Conversión de números binarios a números decimales
#-----------------------------------------------------------------------
def Traducir(individuo, Genes):
    Posiciones = []
    for j in range(Genes):
        Decimal = int(individuo[j],2)
        Bin_Dec = math.floor((Decimal/2**7) * 120 - 60)
        Posiciones.append(Bin_Dec)
    return Posiciones


#Cruza, clonación y mutación de los individuos
#-----------------------------------------------------------------------------------------------
def Cruza(Poblacion, Genes, Pc, Pm, MejorMovimiento):

    #Conversion de números decimales a números binarios

    Individuos = len(Poblacion)
    individuo = []
    poblacion_bin = []
    for i in range (Individuos):
        for j in range (Genes):
            Dec_Bin = math.ceil(((Poblacion[i][j]+60)/120) * (2**7))
            Binario = ''
            while (Dec_Bin // 2 !=0):
                Binario = str(Dec_Bin % 2) + Binario
                Dec_Bin = Dec_Bin // 2
            Binario = str(Dec_Bin) + Binario
            while (len(Binario)<7):
                Binario = '0' + Binario
            individuo.append(Binario)
        poblacion_bin.append(individuo)
        individuo = []

  #Cruza en 2 puntos

  Poblacion = poblacion_bin
    NuevaPoblacion = []
    for i in range(0,Individuos-1,2):
        Padre = Poblacion[i]
        Madre = Poblacion[i+1]

        ProbC = np.random.binomial(1,Pc,1)
        if (ProbC == 1):
            PuntoA = random.randint(1,840)
            PuntoB = random.randint(1,840)

            Padre_unido = ''
            for i in range(len(Padre)):
```

```
        Padre_unido = Padre_unido + Padre[i]
    Madre_unido = ''
    for i in range(len(Madre)):
        Madre_unido = Madre_unido + Madre[i]

    Hijo1_unido = ''
    Hijo2_unido = ''

    if (PuntoA < PuntoB):
        Hijo1_unido = Padre_unido[0:PuntoA] + Madre_unido[PuntoA:PuntoB] + Padre_unido[PuntoB:840 + 1]
        Hijo2_unido = Madre_unido[0:PuntoA] + Padre_unido[PuntoA:PuntoB] + Madre_unido[PuntoB:840 + 1]
    else:
        Hijo1_unido = Padre_unido[0:PuntoB] + Madre_unido[PuntoB:PuntoA] + Padre_unido[PuntoA:840 + 1]
        Hijo2_unido = Madre_unido[0:PuntoB] + Padre_unido[PuntoB:PuntoA] + Madre_unido[PuntoA:840 + 1]

    Hijo1=[]
    for i in range (0, len(Hijo1_unido),7):
        Hijo1.append(str(Hijo1_unido[i:i+7]))

    Hijo2=[]
    for i in range (0, len(Hijo2_unido),7):
        Hijo2.append(str(Hijo2_unido[i:i+7]))

else:

    #No hay cruza

    Hijo1 = Padre
    Hijo2 = Madre

#Mutacion

 #Hijo1

union1 = ''
for i in range(len(Hijo1)):
    union1 = union1 + Hijo1[i]
cambio1 = ''
for i in range(len(union1)):
    mutacion1 = np.random.binomial(1,Pm,1)
    if (mutacion1 == 1):
        if (union1[i] == '0'):
            cambio1 = cambio1 + '1'
        else:
            cambio1 = cambio1 + '0'
    else:
        cambio1 = cambio1 + union1[i]

Hijo1Mutado=[]
for i in range (0, len(cambio1),7):
    Hijo1Mutado.append(str(cambio1[i:i+7]))
Hijo1 = Hijo1Mutado

 #Hijo2

union2 = ''
for i in range(len(Hijo2)):
    union2 = union2 + Hijo2[i]
cambio2 = ''
for i in range(len(union2)):
    #Mutacion de 0.01 del vector
    mutacion2 = np.random.binomial(1,Pm,1)
    if (mutacion2 == 1):
        if (union2[i] == '0'):
            cambio2 = cambio2 + '1'
        else:
            cambio2 = cambio2 + '0'
    else:
        cambio2 = cambio2 + union2[i]

Hijo2Mutado=[]
for i in range (0, len(cambio2),7):
    Hijo2Mutado.append(str(cambio2[i:i+7]))
Hijo2 = Hijo2Mutado

Hijo1 = Traducir(Hijo1, Genes)
```

```
        Hijo2 = Traducir(Hijo2, Genes)

        NuevaPoblacion.append(Hijo1)
        NuevaPoblacion.append(Hijo2)

    #Agregar al mejor individuo de la poblacion anterior a la nueva poblacion
    NuevaPoblacion.pop()
    NuevaPoblacion.append(MejorMovimiento)
    Poblacion=NuevaPoblacion

    return Poblacion


#Conexión con CoppeliaSim para obtención de datos
#-----------------------------------------------------------------------------------------------------------------------------------------------------------
def Marcha(Posiciones):

    #Prueba de ciclo de marcha

    motor11 = Posiciones[0:1] + Posiciones[12:13] + Posiciones[24:25] + Posiciones[36:37] + Posiciones[48:49] +
              Posiciones[60:61]  + Posiciones[72:73] + Posiciones[84:85] + Posiciones[96:97] + Posiciones[108:109] +
              Posiciones[0:1]
    motor12 = Posiciones[1:2] + Posiciones[13:14] + Posiciones[25:26] + Posiciones[37:38] + Posiciones[49:50] +
              Posiciones[61:62] + Posiciones[73:74] + Posiciones[85:86] + Posiciones[97:98] + Posiciones[109:110] +
              Posiciones[1:2]
    motor13 = Posiciones[2:3] + Posiciones[14:15] + Posiciones[26:27] + Posiciones[38:39] + Posiciones[50:51]
              Posiciones[62:63] + Posiciones[74:75] + Posiciones[86:87] + Posiciones[98:99] + Posiciones[110:111] +
               Posiciones[2:3]
    motor21 = Posiciones[3:4] + Posiciones[15:16] + Posiciones[27:28] + Posiciones[39:40] + Posiciones[51:52] +
               Posiciones[63:64] + Posiciones[75:76] + Posiciones[87:88] + Posiciones[99:100] + Posiciones[111:112] +
              Posiciones[3:4]
    motor22 = Posiciones[4:5] + Posiciones[16:17] + Posiciones[28:29] + Posiciones[40:41] + Posiciones[52:53] +
              Posiciones[64:65] + Posiciones[76:77] + Posiciones[88:89] + Posiciones[100:101] + Posiciones[112:113] +
              Posiciones[4:5]
    motor23 = Posiciones[5:6] + Posiciones[17:18] + Posiciones[29:30] + Posiciones[41:42] + Posiciones[53:54] +
              Posiciones[65:66] + Posiciones[77:78] + Posiciones[89:90] + Posiciones[101:102] + Posiciones[113:114] +
              Posiciones[5:6]
    motor31 = Posiciones[6:7] + Posiciones[18:19] + Posiciones[30:31] + Posiciones[42:43] + Posiciones[54:55] +
               Posiciones[66:67] + Posiciones[78:79] + Posiciones[90:91] + Posiciones[102:103] + Posiciones[114:115]+
               Posiciones[6:7]
    motor32 = Posiciones[7:8] + Posiciones[19:20] + Posiciones[31:32] + Posiciones[43:44] + Posiciones[55:56] +
               Posiciones[67:68] + Posiciones[79:80] + Posiciones[91:92] + Posiciones[103:104] + Posiciones[115:116]+
               Posiciones[7:8]
    motor33 = Posiciones[8:9] + Posiciones[20:21] + Posiciones[32:33] + Posiciones[44:45] + Posiciones[56:57] +
               Posiciones[68:69] + Posiciones[80:81] + Posiciones[92:93] + Posiciones[104:105] + Posiciones[116:117]+
               Posiciones[8:9]
    motor41 = Posiciones[9:10] + Posiciones[21:22] + Posiciones[33:34] + Posiciones[45:46] + Posiciones[57:58] +
               Posiciones[69:70] + Posiciones[81:82] + Posiciones[93:94] + Posiciones[105:106] + Posiciones[117:118]+
               Posiciones[9:10]
    motor42 = Posiciones[10:11] + Posiciones[22:23] + Posiciones[34:35] + Posiciones[46:47] + Posiciones[58:59] +
              Posiciones[70:71] + Posiciones[82:83] + Posiciones[94:95] + Posiciones[106:107] + Posiciones[118:119] +
              Posiciones[10:11]
    motor43 = Posiciones[11:12] + Posiciones[23:24] + Posiciones[35:36] + Posiciones[47:48] + Posiciones[59:60] +
               Posiciones[71:72] + Posiciones[83:84] + Posiciones[95:96] + Posiciones[107:108] + Posiciones[119:120]+
               Posiciones[11:12]

    #IInicia programa y abre conexión

    sim.simxFinish(-1)
    clientID=sim.simxStart('127.0.0.1',19997,True,True,5000,5)
    if (clientID == 0):
        joints={}
        for i in range (1,5):
            for j in range (1,4):
                jointname="articulacion{}{}".format(i,j)
                joint="art{}{}".format(i,j)
                err1,joints[joint]=sim.simxGetObjectHandle(clientID, jointname, sim.simx_opmode_oneshot_wait)

        err1,body=sim.simxGetObjectHandle(clientID, "Cuerpo1", sim.simx_opmode_oneshot_wait)
        err1,pata1=sim.simxGetObjectHandle(clientID, "Pata13", sim.simx_opmode_oneshot_wait)
        err1,pata2=sim.simxGetObjectHandle(clientID, "Pata23", sim.simx_opmode_oneshot_wait)
        err1,pata3=sim.simxGetObjectHandle(clientID, "Pata33", sim.simx_opmode_oneshot_wait)
        err1,pata4=sim.simxGetObjectHandle(clientID, "Pata43", sim.simx_opmode_oneshot_wait)
        err1,ref=sim.simxGetObjectHandle(clientID, "Referencia", sim.simx_opmode_oneshot_wait)
        pasos=0 #Inicializa el numero de pasos
```

```
    Aux=0          #Contador de la posicion en el eje "z"
    AuxA=0         #Contador de la orientación en Alpha
    AuxB=0         # Contador de la orientación en Beta
    AuxC=0         # Contador de la orientación en Gamma
    AuxBody=0      #Contador de la posicion en el eje "y"

    #Colisiones

    ColisionArticulacion = [None] * 12
    err1,ColisionArticulacion[0] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata12',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[1] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata22',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[2] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata32',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[3] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata42',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[4] = sim.simxGetCollisionHandle(clientID, 'Pata23_Pata43',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[5] = sim.simxGetCollisionHandle(clientID, 'Pata13_Pata33',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[6] = sim.simxGetCollisionHandle(clientID, 'Pata13_Pata32',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[7] = sim.simxGetCollisionHandle(clientID, 'Pata12_Pata33',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[8] = sim.simxGetCollisionHandle(clientID, 'Pata12_Pata32',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[9] = sim.simxGetCollisionHandle(clientID, 'Pata23_Pata42',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[10] = sim.simxGetCollisionHandle(clientID, 'Pata22_Pata43',
sim.simx_opmode_oneshot_wait)
    err1,ColisionArticulacion[11] = sim.simxGetCollisionHandle(clientID, 'Pata22_Pata42',
sim.simx_opmode_oneshot_wait)

    while (pasos<11):

        #Avanzan todas las patas para realizar movimiento

        sim.simxPauseCommunication(clientID, 1)
        sim.simxSetJointTargetPosition(clientID, joints['art11'], motor11[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art12'], motor12[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art13'], motor13[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art21'], motor21[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art22'], motor22[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art23'], motor23[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art31'], motor31[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art32'], motor32[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art33'], motor33[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art41'], motor41[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art42'], motor42[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxSetJointTargetPosition(clientID, joints['art43'], motor43[pasos]*np.pi/180, sim.simx_opmode_oneshot)
        sim.simxPauseCommunication(clientID, 0)

        #ARTICULACION11

        T = time.time()
        Inicio = T
        Actual = T
        err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_streaming)
        err5,Colision1 = sim.simxReadCollision(clientID,ColisionArticulacion[0],sim.simx_opmode_streaming)
        err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
        err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
        err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
        err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
        while ((Actual - Inicio)<0.5):
            err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_buffer)
            if (err2 == sim.simx_return_ok):
                err5,Colision1 = sim.simxReadCollision(clientID, ColisionArticulacion[0], sim.simx_opmode_buffer)
                err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
                err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
                err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
                err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
                if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
                    (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
                    if ((abs(v-1*(motor11[pasos]*np.pi/180))<2*np.pi/180) or (Colision1 == True) or (Colision6 == True) or
```

```
            (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
            break
    Actual=time.time()

#ARTICULACION12

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_streaming)
err5,Colision1 = sim.simxReadCollision(clientID,ColisionArticulacion[0],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision1 = sim.simxReadCollision(clientID, ColisionArticulacion[0], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor12[pasos]*np.pi/180))<2*np.pi/180) or (Colision1 == True) or (Colision6 == True) or
                (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

#ARTICULACION13

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_streaming)
err5,Colision1 = sim.simxReadCollision(clientID,ColisionArticulacion[0],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision1 = sim.simxReadCollision(clientID, ColisionArticulacion[0], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor13[pasos]*np.pi/180))<2*np.pi/180) or (Colision1 == True) or (Colision6 == True) or
                (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

# ARTICULACION21

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_streaming)
err5,Colision2 = sim.simxReadCollision(clientID,ColisionArticulacion[1],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision2 = sim.simxReadCollision(clientID, ColisionArticulacion[1], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
```

```
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor21[pasos]*np.pi/180))<2*np.pi/180) or (Colision2 == True) or (Colision5 == True) or
                (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION22

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_streaming)
err5,Colision2 = sim.simxReadCollision(clientID,ColisionArticulacion[1],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision2 = sim.simxReadCollision(clientID, ColisionArticulacion[1], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor22[pasos]*np.pi/180))<2*np.pi/180) or (Colision2 == True) or (Colision5 == True) or
                (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION23

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_streaming)
err5,Colision2 = sim.simxReadCollision(clientID,ColisionArticulacion[1],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision2 = sim.simxReadCollision(clientID, ColisionArticulacion[1], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor23[pasos]*np.pi/180))<2*np.pi/180) or (Colision2 == True) or (Colision5 == True) or
                (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION31

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_streaming)
err5,Colision3 = sim.simxReadCollision(clientID,ColisionArticulacion[2],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision3 = sim.simxReadCollision(clientID, ColisionArticulacion[2], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
```

```
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok)  and (err7 ==sim.simx_return_ok) and
          (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
           if ((abs(v-1*(motor31[pasos]*np.pi/180))<2*np.pi/180) or (Colision3 == True) or (Colision6 == True) or
             (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
               break
    Actual=time.time()

#ARTICULACION32

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_streaming)
err5,Colision3 = sim.simxReadCollision(clientID,ColisionArticulacion[2],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision3 = sim.simxReadCollision(clientID, ColisionArticulacion[2], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor32[pasos]*np.pi/180))<2*np.pi/180) or (Colision3 == True) or (Colision6 == True) or
              (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

#ARTICULACION33

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_streaming)
err5,Colision3 = sim.simxReadCollision(clientID,ColisionArticulacion[2],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision3 = sim.simxReadCollision(clientID, ColisionArticulacion[2], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor33[pasos]*np.pi/180))<2*np.pi/180) or (Colision3 == True) or (Colision6 == True) or
              (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

#ARTICULACION41

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_streaming)
err5,Colision4 = sim.simxReadCollision(clientID,ColisionArticulacion[3],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
```

```
        err5,Colision4 = sim.simxReadCollision(clientID, ColisionArticulacion[3], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor41[pasos]*np.pi/180))<2*np.pi/180) or (Colision4 == True) or (Colision5 == True) or
              (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION42

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_streaming)
err5,Colision4 = sim.simxReadCollision(clientID,ColisionArticulacion[3],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision4 = sim.simxReadCollision(clientID, ColisionArticulacion[3], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
         (err8==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor42[pasos]*np.pi/180))<2*np.pi/180) or (Colision4 == True) or (Colision5 == True) or
              (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION43

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_streaming)
err5,Colision4 = sim.simxReadCollision(clientID,ColisionArticulacion[3],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision4 = sim.simxReadCollision(clientID, ColisionArticulacion[3], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8
            ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor43[pasos]*np.pi/180))<2*np.pi/180) or (Colision4 == True) or (Colision5 == True) or
              (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#Posicion cuerpo

err4,posf=sim.simxGetObjectPosition(clientID, body, ref, sim.simx_opmode_streaming)
while True:
    err4,posf=sim.simxGetObjectPosition(clientID, body, ref, sim.simx_opmode_buffer)
    if (err4==sim.simx_return_ok): #Da tiempo para asegurar de que llegue el dato
        break

if (posf[2]<0.226 or posf[2]>0.2361):
    Aux=Aux + 1
```

```
    if (pasos == 0):
        posi=posf

    else:

        if (posf[1]>posi[1]):
            AuxBody= AuxBody + 1
        if (pasos == 5):
            posi=posf

    #Orientacion

    err4,ori=sim.simxGetObjectOrientation(clientID, body, -1, sim.simx_opmode_streaming)
    while True:
        err4,ori=sim.simxGetObjectOrientation(clientID, body, -1, sim.simx_opmode_buffer)
        if (err4==sim.simx_return_ok):
            oriA=(ori[0]*180)/np.pi
            oriB=(ori[1]*180)/np.pi
            oriC=(ori[2]*180)/np.pi
            break

    if (oriA>5 or oriA<-5):
        AuxA= AuxA + 1
        if (oriA>10 or oriA<-10):
            AuxA= AuxA + 2

    if (oriB>5 or oriB<-5):
        AuxB= AuxB + 1
        if (oriB>10 or oriB<-10):
            AuxB= AuxB + 2

    if (oriC>10 or oriC<-10):
        AuxC= AuxC + 1
        if (oriC>15 or oriC<-15):
            AuxC= AuxC + 2

    pasos=pasos+1


Alpha = AuxA
Beta = AuxB
Gamma = AuxC
Zeta = Aux

#Reinicio de simulacion

if (oriA or oriB or oriC >= 50):
    sim.simxStopSimulation(clientID,sim.simx_opmode_oneshot)
    time.sleep(1)
    sim.simxStartSimulation(clientID,sim.simx_opmode_oneshot)
    time.sleep(0.5)

#Regresa a posiciones iniciales

sim.simxPauseCommunication(clientID, 1)
sim.simxSetJointTargetPosition(clientID, joints['art11'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art12'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art13'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art21'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art22'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art23'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art31'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art32'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art33'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art41'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art42'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art43'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxPauseCommunication(clientID, 0)

err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break
```

```
    err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    sim.simxFinish(clientID)

else:
    print("No se pudo conectar")


return Zeta, Alpha, Beta, Gamma, AuxBody
```

#Muestra marcha de mejor individuo de la poblacion

```
#-----------------------------------------------------------------------------------------------------------------------------------------------------
def MarchaFinal(Posiciones):

    motor11 = Posiciones[0:1] + Posiciones[12:13] + Posiciones[24:25] + Posiciones[36:37] + Posiciones[48:49] +
              Posiciones[60:61]  + Posiciones[72:73] + Posiciones[84:85] + Posiciones[96:97] + Posiciones[108:109]
    motor12 = Posiciones[1:2] + Posiciones[13:14] + Posiciones[25:26] + Posiciones[37:38] + Posiciones[49:50] +
              Posiciones[61:62] + Posiciones[73:74] + Posiciones[85:86] + Posiciones[97:98] + Posiciones[109:110]
    motor13 = Posiciones[2:3] + Posiciones[14:15] + Posiciones[26:27] + Posiciones[38:39] + Posiciones[50:51]  +
              Posiciones[62:63] + Posiciones[74:75] + Posiciones[86:87] + Posiciones[98:99] + Posiciones[110:111]
    motor21 = Posiciones[3:4] + Posiciones[15:16] + Posiciones[27:28] + Posiciones[39:40] + Posiciones[51:52] +
               Posiciones[63:64] + Posiciones[75:76] + Posiciones[87:88] + Posiciones[99:100] + Posiciones[111:112]
    motor22 = Posiciones[4:5] + Posiciones[16:17] + Posiciones[28:29] + Posiciones[40:41] + Posiciones[52:53] +
              Posiciones[64:65] + Posiciones[76:77] + Posiciones[88:89] + Posiciones[100:101] + Posiciones[112:113]
    motor23 = Posiciones[5:6] + Posiciones[17:18] + Posiciones[29:30] + Posiciones[41:42] + Posiciones[53:54] +
              Posiciones[65:66] + Posiciones[77:78] + Posiciones[89:90] + Posiciones[101:102] + Posiciones[113:114]
    motor31 = Posiciones[6:7] + Posiciones[18:19] + Posiciones[30:31] + Posiciones[42:43] + Posiciones[54:55] +
               Posiciones[66:67] + Posiciones[78:79] + Posiciones[90:91] + Posiciones[102:103] + Posiciones[114:115]
    motor32 = Posiciones[7:8] + Posiciones[19:20] + Posiciones[31:32] + Posiciones[43:44] + Posiciones[55:56] +
               Posiciones[67:68] + Posiciones[79:80] + Posiciones[91:92] + Posiciones[103:104] + Posiciones[115:116]
    motor33 = Posiciones[8:9] + Posiciones[20:21] + Posiciones[32:33] + Posiciones[44:45] + Posiciones[56:57] +
               Posiciones[68:69] + Posiciones[80:81] + Posiciones[92:93] + Posiciones[104:105] + Posiciones[116:117]
    motor41 = Posiciones[9:10] + Posiciones[21:22] + Posiciones[33:34] + Posiciones[45:46] + Posiciones[57:58] +
               Posiciones[69:70] + Posiciones[81:82] + Posiciones[93:94] + Posiciones[105:106] + Posiciones[117:118]
    motor42 = Posiciones[10:11] + Posiciones[22:23] + Posiciones[34:35] + Posiciones[46:47] + Posiciones[58:59] +
              Posiciones[70:71] + Posiciones[82:83] + Posiciones[94:95] + Posiciones[106:107] + Posiciones[118:119]
    motor43 = Posiciones[11:12] + Posiciones[23:24] + Posiciones[35:36] + Posiciones[47:48] + Posiciones[59:60] +
               Posiciones[71:72] + Posiciones[83:84] + Posiciones[95:96] + Posiciones[107:108] + Posiciones[119:120]


 #Inicia programa y abre conexión

    print("Inciando marcha de mejor individuo")
    sim.simxFinish(-1)
    clientID=sim.simxStart('127.0.0.1',19997,True,True,5000,5)
    if (clientID == 0):
        print("Conectado al servidor API remoto correctamente")
        joints={}
        for i in range (1,5):
            for j in range (1,4):
                jointname="articulacion{}{}".format(i,j)
                joint="art{}{}".format(i,j)
                err1,joints[joint]=sim.simxGetObjectHandle(clientID, jointname, sim.simx_opmode_oneshot_wait)

        pasos=0      #Inicializa el numero de pasos
        marcha=0     #Inicializa el numero de ciclos de marcha

        #Colisiones

        ColisionArticulacion = [None] * 12
        err1,ColisionArticulacion[0] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata12',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[1] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata22',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[2] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata32',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[3] = sim.simxGetCollisionHandle(clientID, 'Cuerpo_Pata42',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[4] = sim.simxGetCollisionHandle(clientID, 'Pata23_Pata43',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[5] = sim.simxGetCollisionHandle(clientID, 'Pata13_Pata33',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[6] = sim.simxGetCollisionHandle(clientID, 'Pata13_Pata32',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[7] = sim.simxGetCollisionHandle(clientID, 'Pata12_Pata33',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[8] = sim.simxGetCollisionHandle(clientID, 'Pata12_Pata32',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[9] = sim.simxGetCollisionHandle(clientID, 'Pata23_Pata42',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[10] = sim.simxGetCollisionHandle(clientID, 'Pata22_Pata43',
sim.simx_opmode_oneshot_wait)
        err1,ColisionArticulacion[11] = sim.simxGetCollisionHandle(clientID, 'Pata22_Pata42',
sim.simx_opmode_oneshot_wait)
```

```
    while (marcha<5):
        while (pasos<10):

            #Se mueven todas las patas para realizar un paso

            sim.simxPauseCommunication(clientID, 1)
            sim.simxSetJointTargetPosition(clientID, joints['art11'], motor11[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art12'], motor12[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art13'], motor13[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art21'], motor21[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art22'], motor22[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art23'], motor23[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art31'], motor31[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art32'], motor32[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art33'], motor33[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art41'], motor41[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art42'], motor42[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxSetJointTargetPosition(clientID, joints['art43'], motor43[pasos]*np.pi/180,
sim.simx_opmode_oneshot)
            sim.simxPauseCommunication(clientID, 0)

            #ARTICULACION11

            T = time.time()
            Inicio = T
            Actual = T
            err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_streaming)
            err5,Colision1 = sim.simxReadCollision(clientID,ColisionArticulacion[0],sim.simx_opmode_streaming)
            err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
            err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
            err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
            err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
            while ((Actual - Inicio)<0.5):
                err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_buffer)
                if (err2 == sim.simx_return_ok):
                    err5,Colision1 = sim.simxReadCollision(clientID, ColisionArticulacion[0], sim.simx_opmode_buffer)
                    err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
                    err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
                    err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
                    err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
                    if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
                        (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
                        if ((abs(v-1*(motor11[pasos]*np.pi/180))<2*np.pi/180) or (Colision1 == True) or (Colision6 == True) or
                            (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                            break
                Actual=time.time()

            #ARTICULACION12

            T = time.time()
            Inicio = T
            Actual = T
            err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_streaming)
            err5,Colision1 = sim.simxReadCollision(clientID,ColisionArticulacion[0],sim.simx_opmode_streaming)
            err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
            err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
            err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
            err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
            while ((Actual - Inicio)<0.5):
                err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_buffer)
                if (err2 == sim.simx_return_ok):
                    err5,Colision1 = sim.simxReadCollision(clientID, ColisionArticulacion[0], sim.simx_opmode_buffer)
                    err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
                    err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
                    err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
```

```
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
           (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
           if ((abs(v-1*(motor12[pasos]*np.pi/180))<2*np.pi/180) or (Colision1 == True) or (Colision6 == True) or
              (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
              break
    Actual=time.time()

#ARTICULACION13

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_streaming)
err5,Colision1 = sim.simxReadCollision(clientID,ColisionArticulacion[0],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision1 = sim.simxReadCollision(clientID, ColisionArticulacion[0], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
       (err8==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
           if ((abs(v-1*(motor13[pasos]*np.pi/180))<2*np.pi/180) or (Colision1 == True) or (Colision6 == True) or
              (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
              break
    Actual=time.time()

#ARTICULACION21

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_streaming)
err5,Colision2 = sim.simxReadCollision(clientID,ColisionArticulacion[1],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision2 = sim.simxReadCollision(clientID, ColisionArticulacion[1], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
           (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
           if ((abs(v-1*(motor21[pasos]*np.pi/180))<2*np.pi/180) or (Colision2 == True) or (Colision5 == True) or
              (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
              break
    Actual=time.time()

#ARTICULACION22

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_streaming)
err5,Colision2 = sim.simxReadCollision(clientID,ColisionArticulacion[1],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision2 = sim.simxReadCollision(clientID, ColisionArticulacion[1], sim.simx_opmode_buffer)
```

```
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor22[pasos]*np.pi/180))<2*np.pi/180) or (Colision2 == True) or (Colision5 == True) or
              (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION23

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_streaming)
err5,Colision2 = sim.simxReadCollision(clientID,ColisionArticulacion[1],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision2 = sim.simxReadCollision(clientID, ColisionArticulacion[1], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor23[pasos]*np.pi/180))<2*np.pi/180) or (Colision2 == True) or (Colision5 == True) or
                (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION31

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_streaming)
err5,Colision3 = sim.simxReadCollision(clientID,ColisionArticulacion[2],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision3 = sim.simxReadCollision(clientID, ColisionArticulacion[2], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok)  and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor31[pasos]*np.pi/180))<2*np.pi/180) or (Colision3 == True) or (Colision6 == True) or
                (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

#ARTICULACION32

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_streaming)
err5,Colision3 = sim.simxReadCollision(clientID,ColisionArticulacion[2],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_buffer)
```

```
    if (err2 == sim.simx_return_ok):
        err5,Colision3 = sim.simxReadCollision(clientID, ColisionArticulacion[2], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor32[pasos]*np.pi/180))<2*np.pi/180) or (Colision3 == True) or (Colision6 == True) or
                (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

#ARTICULACION33

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_streaming)
err5,Colision3 = sim.simxReadCollision(clientID,ColisionArticulacion[2],sim.simx_opmode_streaming)
err6,Colision6 = sim.simxReadCollision(clientID,ColisionArticulacion[5],sim.simx_opmode_streaming)
err7,Colision7 = sim.simxReadCollision(clientID,ColisionArticulacion[6],sim.simx_opmode_streaming)
err8,Colision8 = sim.simxReadCollision(clientID,ColisionArticulacion[7],sim.simx_opmode_streaming)
err9,Colision9 = sim.simxReadCollision(clientID,ColisionArticulacion[8],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision3 = sim.simxReadCollision(clientID, ColisionArticulacion[2], sim.simx_opmode_buffer)
        err6,Colision6 = sim.simxReadCollision(clientID, ColisionArticulacion[5], sim.simx_opmode_buffer)
        err7,Colision7 = sim.simxReadCollision(clientID, ColisionArticulacion[6], sim.simx_opmode_buffer)
        err8,Colision8 = sim.simxReadCollision(clientID, ColisionArticulacion[7], sim.simx_opmode_buffer)
        err9,Colision9 = sim.simxReadCollision(clientID, ColisionArticulacion[8], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor33[pasos]*np.pi/180))<2*np.pi/180) or (Colision3 == True) or (Colision6 == True) or
                (Colision7 == True) or (Colision8 == True) or (Colision9 == True)):
                break
    Actual=time.time()

#ARTICULACION41

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_streaming)
err5,Colision4 = sim.simxReadCollision(clientID,ColisionArticulacion[3],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
while ((Actual - Inicio)<0.5):
    err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_buffer)
    if (err2 == sim.simx_return_ok):
        err5,Colision4 = sim.simxReadCollision(clientID, ColisionArticulacion[3], sim.simx_opmode_buffer)
        err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
        err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
        err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
        err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
        if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
            (err8==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
            if ((abs(v-1*(motor41[pasos]*np.pi/180))<2*np.pi/180) or (Colision4 == True) or (Colision5 == True) or
              (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                break
    Actual=time.time()

#ARTICULACION42

T = time.time()
Inicio = T
Actual = T
err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_streaming)
err5,Colision4 = sim.simxReadCollision(clientID,ColisionArticulacion[3],sim.simx_opmode_streaming)
err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
```

```
        err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
        while ((Actual - Inicio)<0.5):
            err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_buffer)
            if (err2 == sim.simx_return_ok):
                err5,Colision4 = sim.simxReadCollision(clientID, ColisionArticulacion[3], sim.simx_opmode_buffer)
                err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
                err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
                err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
                err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
                if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
                    (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
                    if ((abs(v-1*(motor42[pasos]*np.pi/180))<2*np.pi/180) or (Colision4 == True) or (Colision5 == True) or
                        (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                        break
            Actual=time.time()

        #ARTICULACION43

        T = time.time()
        Inicio = T
        Actual = T
        err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_streaming)
        err5,Colision4 = sim.simxReadCollision(clientID,ColisionArticulacion[3],sim.simx_opmode_streaming)
        err6,Colision5 = sim.simxReadCollision(clientID,ColisionArticulacion[4],sim.simx_opmode_streaming)
        err7,Colision10 = sim.simxReadCollision(clientID,ColisionArticulacion[9],sim.simx_opmode_streaming)
        err8,Colision11 = sim.simxReadCollision(clientID,ColisionArticulacion[10],sim.simx_opmode_streaming)
        err9,Colision12 = sim.simxReadCollision(clientID,ColisionArticulacion[11],sim.simx_opmode_streaming)
        while ((Actual - Inicio)<0.5):
            err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_buffer)
            if (err2 == sim.simx_return_ok):
                err5,Colision4 = sim.simxReadCollision(clientID, ColisionArticulacion[3], sim.simx_opmode_buffer)
                err6,Colision5 = sim.simxReadCollision(clientID, ColisionArticulacion[4], sim.simx_opmode_buffer)
                err7,Colision10 = sim.simxReadCollision(clientID, ColisionArticulacion[9], sim.simx_opmode_buffer)
                err8,Colision11 = sim.simxReadCollision(clientID, ColisionArticulacion[10], sim.simx_opmode_buffer)
                err9,Colision12 = sim.simxReadCollision(clientID, ColisionArticulacion[11], sim.simx_opmode_buffer)
                if ((err5 == sim.simx_return_ok) and (err6 ==sim.simx_return_ok) and (err7 ==sim.simx_return_ok) and
                    (err8 ==sim.simx_return_ok) and (err9 ==sim.simx_return_ok)):
                    if ((abs(v-1*(motor43[pasos]*np.pi/180))<2*np.pi/180) or (Colision4 == True) or (Colision5 == True) or
                        (Colision10 == True) or (Colision11 == True) or (Colision12 == True)):
                        break
            Actual=time.time()

        pasos=pasos+1
    marcha=marcha+1
    pasos=0


#Regresa a posiciones iniciales

sim.simxPauseCommunication(clientID, 1)
sim.simxSetJointTargetPosition(clientID, joints['art11'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art12'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art13'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art21'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art22'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art23'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art31'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art32'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art33'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art41'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art42'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxSetJointTargetPosition(clientID, joints['art43'], 0*np.pi/180, sim.simx_opmode_oneshot)
sim.simxPauseCommunication(clientID, 0)

err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art11'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break

err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_streaming)
while True:
    err2,v=sim.simxGetJointPosition(clientID, joints['art12'], sim.simx_opmode_buffer)
    if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
        break
```

```
    err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art13'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art21'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art22'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art23'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art31'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art32'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art33'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art41'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art42'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break

    err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_streaming)
    while True:
        err2,v=sim.simxGetJointPosition(clientID, joints['art43'], sim.simx_opmode_buffer)
        if (abs(v-1*(0*np.pi/180))<0.1*np.pi/180):
            break
    sim.simxFinish(clientID)

else:
    print("No se pudo conectar")
```