Diseño de un sistema de coordinación para enjambres de robots móviles heterogéneos

JOHN JAIRO CHAVEZ CHAVEZ

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

UNIVERSIDAD NACIONAL DE COLOMBIA

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE SISTEMAS E INDUSTRIAL

BOGOTÁ, D.C

Diseño de un sistema de coordinación para enjambres de robots móviles heterogéneos

JOHN JAIRO CHAVEZ CHAVEZ

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

TESIS PRESENTADA PARA OPTAR AL TÍTULO DE

M.SC EN SISTEMAS Y COMPUTACIÓN

DIRECTOR

JONATAN GÓMEZ PERDOMO, PH.D.

CODIRECTOR

ERNESTO CÓRDOBA NIETO, PROFESOR

LÍNEA DE INVESTIGACIÓN

SISTEMAS INTELIGENTES

UNIVERSIDAD NACIONAL DE COLOMBIA

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE SISTEMAS E INDUSTRIAL

BOGOTÁ, D.C

Título en español

Diseño de un sistema de coordinación para enjambres de robots móviles heterogéneos

Title in English

Coordination System Design for heterogeneous swarm of mobile robots

Resumen:

Este trabajo muestra el diseño automático de comportamientos para un enjambre heterogéneo de robots utilizando programación genética. Para ello, se propone el desarrollo de una plataforma computacional que incluye un formato de descripción para la especificación de robots y sus características, comportamientos primitivos que poseen los robots, y tareas que realiza el enjambre de robots. Los comportamientos son construidos con programación genética componiendo y ajustando árboles de expresión que son validados en un simulador basado en física. Como parte de la validación de la plataforma computacional se diseñan e implementan comportamientos de agregación para un grupo de tres robots móviles simulados y su despliegue en tres robots del Laboratorio Fábrica experimental de la Universidad Nacional de Colombia sede Bogotá .

Abstract:

This research work studies the automatic design of behaviors for an heterogeneous swarm of robots using genetic programming. In order to build behaviors for robots automatically a computational platform is proposed. The proposed platform is composed by three major components. The first component is a description format which allows to specify robot properties, basic behaviors and tasks. The second component is a genetic programming implementation along with a physics-based simulator, this component builds in an automatic way expression trees which represent robot behaviors. The final component is a behaviors assignment module to deploy expression trees on real robots. In order to validate the proposed platform robot behaviors are built for three simulated mobile robots and their deployment in three real robots in a manufacturing environment at Universidad Nacional de Colombia.

Palabras clave: Robótica de enjambres, programación genética, comportamientos, diseño automático, coordinación.

Keywords: Swarm robotics, genetic programming, behaviors, automatic design, coordination.

Nota de aceptación

_____

Jurado

_____

Jurado

_____

Director

_____

Codirector

Bogotá, D.C., Noviembre 24 de 2017,

Dedicado a

A mi Padre, porque sus promesas siempre se cumplen.

Agradecimientos

A los profesores Jonatan Gómez y Ernesto Córdoba por su paciencia y apoyo durante el desarrollo de este trabajo.

A los estudiantes, Ingenieros y directivos del grupo de investigación DIMA UN por su valiosa colaboración y soporte durante el desarrollo de este trabajo.

Índice General

Índice de tablas

Índice de Figuras

Introducción

Los robots son usados en diversos ámbitos y su aplicación se ha extendido tanto a escenarios como la industria (manufactura y militar), a la exploración espacial y otros más. Las prestaciones de los sistemas robotizados han motivado la integración y coordinación de robots para solucionar problemas y atender las crecientes demandas en diversos sectores de la sociedad. Estos conjuntos de robots se conocen como sistemas Multi-robot (MRSs, por sus siglas en inglés), sistemas en los que sus elementos interactúan entre si y con el entorno para cumplir un objetivo dentro de un proceso bien definido. Los robots pueden ser de diferentes tipos: robots manipuladores (seriales, paralelos o híbridos), robots móviles (homogéneos y/o heterogéneos), y máquinas automáticas. Un tipo particular de MRSs es aquel en el que el conjunto de robots está integrado únicamente por robots móviles (MMRSs, por sus siglas en inglés). La investigación en coordinación de MMRSs ha sido un área activa desde la década de los 80 [19], investigación cuyo principal objetivo es el desarrollo de mecanismos de coordinación que permitan el despliegue de grupos de robots para solucionar problemas del mundo real, tal como lo es el transporte de materiales y productos en celdas de manufactura.

El resultado final de un proceso de diseño para coordinar un grupo de robots es el conjunto de reglas que le indican a cada miembro del conjunto de robots cómo debe comportarse para resolver una tarea específica en un contexto o entorno de aplicación. En general, los entornos de aplicación o despliegue de los sistemas multi-robot son, en algún grado, no estructurados y altamente dinámicos, lo que implica que los comportamientos generados para cada robot durante el proceso de diseño sean robustos, flexibles y escalables al conjunto de robots. Se han propuesto varios mecanismos para la coordinación de MMRSs que se agrupan en dos grandes tipos de aproximaciones: técnicas de coordinación fuerte y técnicas de coordinación bionspiradas [11].

Por un lado las técnicas de coordinación fuerte dividen el problema de coordinación en dos partes: planeación y asignación de tareas [4], [5], [20]; en la planeación se busca descomponer una tarea en subtareas simples [12] que pueden ser realizadas por los robots; la asignación trata el problema de asignar las subtareas a los robots de forma óptima [16], siendo este el problema más abordado y de donde han surgido aproximaciones basadas en mercados [10], [5], basadas en comportamientos [21], [24], basadas en emociones [1], mecanismos de reputación [27], y muchos otros [8], [32], [17], [15], [14].

Por el otro, las técnicas bioinspiradas utilizan abstracciones de los comportamientos observados en enjambres naturales tales como las sociedades de insectos y otros sistemas biológicos, y modelos evolutivos para diseñar software de control para cada miembro del conjunto de robots. Este tipo de técnicas tratan con grupos definidos por

un gran número de robots, generalmente homogéneos, con capacidades de cómputo y percepción limitadas [2].

La robótica de enjambres es una técnica bioinspirada que aborda el problema de coordinación extrapolando el concepto de auto-organización observado en enjambres naturales [51]. Aquí, y de la misma forma en que ha sido observado en las sociedades de insectos, se espera que el orden en el conjunto de robots surja como resultado de las interacciones entre los elementos de dicho conjunto, y que el grupo de robots exhiba las propiedades de adaptabilidad, robustez, y escalabilidad observadas en los enjambres naturales. Sin embargo, desde el punto de vista de ingeniería, el diseño de un sistema multi-robot auto-organizado no es una tarea trivial, ya que el comportamiento del sistema debe ser modelado en términos de la dinámica de las interacciones individuo-individuo -entorno, interacciones que deben ser codificadas en los controladores de los robots. Estos controladores establecen relaciones entre los sistemas de percepción y el sistema de efectores de los robots, las cuales definen las interacciones entre los robots y su entorno para resolver el problema objetivo. Determinar cuáles son las reglas de interacción a nivel individual para producir un patrón de auto-organización deseado es conocido como el problema de diseño [51].

Una de las formas de representar tales interacciones entre los robots ha sido el uso de máquinas de estados finitos probabilistas [17], [14], redes neuronales [18], física virtual [28], [30], para codificar controladores que encapsulan los comportamientos de los individuos del enjambre. Esta aproximación ha permitido coordinar grupos de robots para realizar tareas observadas en sociedades de animales. Por ejemplo, se han desarrollado comportamientos para tareas de forrajeo con conjuntos de robots con tolerancia a fallas [13]. Siguiendo la misma línea, se han definido comportamientos para agrupar robots homogéneos en zonas determinadas (agregación) [29] y segregación [25]. Utilizando técnicas de física virtual se han coordinado robots para la formación de patrones espaciales específicos [28]. De igual forma, se han desarrollado comportamientos de patrones de movimiento colectivo y segregación con un conjunto heterogéneo de robots [31].

Uno de los factores que ha dificultado el desarrollo de controladores para resolver tareas más complejas, es la carencia de métodos de diseño formales y automatizados. La mayoría del tiempo, el software de control es diseñado y producido por un experto humano a través de un arduo proceso experimental difícil de escalar. Para resolver este problema, han surgido algunos sistemas para el diseño automático de software de control que utilizan técnicas de evolución artificial [33], [18], técnicas de optimización por partículas (PSO) [22] y aprendizaje por refuerzo [24]. La meta del diseño automático es encontrar, sin o con poca intervención humana, las interacciones robot-robot y robot-entorno, codificadas en un programa de control, que permitan al conjunto de robots resolver un problema en un entorno determinado, o dicho de otra forma, hacer que el conjunto de robots se auto-organice.

El diseño automático de software de control a través de evolución artificial, también conocido como robótica evolutiva, es una herramienta bioinspirada que utiliza técnicas de computación evolutiva para estructurar y sintonizar controladores para cada robot del enjambre. La robótica evolutiva apalanca el diseño incremental del sistema como un conjunto, es decir, el surgimiento de un comportamiento global a partir de esquemas de control básicos. La aplicación de evolución artificial puede explorar soluciones, que no son evidentes, a través de la interacción de conjuntos de poblaciones durante el proceso evolutivo. Diferentes estrategias de evolución han sido utilizadas para la construcción automática de controladores, entre estas se encuentran: algoritmos genéticos [35], co-evolución [39] y programación genética [40]. Sin embargo, existe un déficit en herramientas computacionales que integren todos los elementos requeridos y faciliten la construcción automática de comportamientos utilizando técnicas de computación evolutiva para enjambres heterogéneos de robots.

El diseño automático de comportamientos utilizando técnicas de evolución artificial requiere de la especificación de un mecanismo de representación de los comportamientos, de forma tal que puedan aplicarse las etapas del proceso evolutivo. Tal mecanismo de representación debe poseer la expresividad necesaria para abarcar un amplio rango de comportamientos, comportamientos a los que se les pueda cuantificar su efectividad mediante un sistema de validación.

En la actualidad existe una gran variedad de plataformas de simulación de robots y plataformas de simulación basadas en agentes [8]. Sin embargo, dichas plataformas no incluyen en su núcleo central: i. las herramientas de comunicación y medición automáticas requeridas en la etapa de validación del proceso evolutivo, ii. los mecanismos de especificación de las propiedades del enjambre y iii. La forma de describir el problema objetivo que se desea resolver.

Por todo lo expuesto hasta este punto, la investigación científica de aproximaciones bioinspiradas con diseño automático es una oportunidad valiosa para el desarrollo de sistemas para la coordinación de sistemas multi-robot, con potencial aplicación a problemas del mundo real como vigilancia, transporte coordinado, búsquedas en áreas de desastre, exploración espacial, logística en hospitales y aeropuertos, y transporte de materiales en entornos de manufactura.

En este trabajo se aborda el diseño automático de software de control para enjambres de robots utilizando técnicas evolutivas, se explora el uso de programación genética para la síntesis o inducción de comportamientos para un grupo heterogéneo de robots (que realizan tareas de forma cooperativa, son especificados en un formato de descripción, y validados en un entorno de simulación), se desarrolla una plataforma computacional que facilita el proceso de diseño utilizando las técnicas expuestas, y se generan comportamientos para resolver tareas seleccionadas de la literatura, así como su despliegue en tres robots móviles en una celda de manufactura experimental.

Las principales contribuciones de esta tesis son.

1. Se desarrolló un entorno computacional, para el diseño automático y despliegue de los comportamientos para un conjunto heterogéneo de robots móviles, que proporciona una aproximación general para la auto-organización basada en interacciones locales.

2. Se desarrolló un formato de descripción que permite: especificar las propiedades físicas, sensores y efectores de los robots; describir tareas, que realiza un conjunto de robots, en términos de objetivos y restricciones; describir un conjunto de comportamientos primitivos que poseen los robots, a partir de los cuales se construyen los comportamientos de auto-organización.

3. Se simuló un conjunto de robots y un entorno de operación, especificados en el formato de descripción, en un simulador basado en física con el fin de recrear las condiciones encontradas en entornos físicos reales y así validar los comportamientos construidos en un proceso de diseño automático.

4. Se construyeron comportamientos de auto-organización para un conjunto de robots simulados, utilizando programación genética, para realizar tareas de agregación espacial. Finalmente, por medio de la plataforma computacional, se desplegaron los comportamiento de auto-organización para la agregación de tres robots móviles del Laboratorio Fábrica Experimental de la Universidad Nacional de Colombia.

Este documento está estructurado de la siguiente forma: en el capítulo I se realiza un estado del arte de los conceptos, fundamentos y trabajos previos en el área de la robótica de enjambres. En el capítulo II se propone un diseño e implementación para una plataforma computacional que permite: i. la generación automática de comportamientos de un conjunto heterogéneo de robots, plataforma en la que es posible describir a cada uno de los miembros de un conjunto de robots, las tareas que realiza el conjunto de robots y los comportamientos primitivos que posee cada robot. ii. generar comportamientos complejos a partir de comportamientos primitivos y iii. simular un conjunto de robots y validar los comportamientos generados durante el ciclo evolutivo. En el capítulo III se presenta el proceso de generación automática de comportamientos, en el que se utiliza programación genética para la síntesis o inducción de programas para el enjambre de robots, se definen tanto la representación de los programas o comportamientos, como los operadores genéticos y las funciones de aptitud. En el capítulo IV se describe el uso de la plataforma computacional para el diseño automático de programas para coordinar a tres robots móviles en una tarea de agregación espacial y el despliegue de dichos comportamientos en robots reales. Finalmente, se exponen las conclusiones y se proponen algunas actividades a realizar en trabajos futuros.

CAPÍTULO 1

Estado del arte

1.1. Introducción

La robótica de enjambres es un parte de la robótica que aborda el diseño de estrategias de coordinación para grupos grandes de robots tomando como inspiración los comportamientos observados en enjambres naturales. Esta surgió de la aplicación de la inteligencia de enjambres a los sistemas multi-robot con la intención de crear una forma de coordinar robots buscando que estos exhiban las mismas propiedades de robustez y flexibilidad observados en los enjambres naturales. La idea principal en esta aproximación es que, el comportamiento del enjambre está determinado por las interacciones que se dan entre los miembros del enjambre y el entorno [45]. Desde el punto de vista de la ingeniería, se desea encontrar las interacciones que hagan que un enjambre de robots solucione problemas del mundo real, tales como, la exploración colaborativa, transporte de material en entornos de manufactura, etc.

Se ha realizado una serie de investigaciones para encontrar los mecanismos que permiten el diseño automático de comportamientos para enjambres de robots. Una de estas aproximaciones es el uso de evolución artificial para construir y sintonizar controladores que codifican las reglas de cada individuo del enjambre que conducen a la auto-organización. En la robótica de enjambres evolutiva (como se le conoce a esta aproximación), cada controlador es codificado en un genotipo al cual se aplican leves modificaciones (mutaciones) o combinaciones (cruces) de forma iterativa hasta encontrar posibles soluciones. Otras aproximaciones utilizan estrategias de optimización similares tales como la optimización por partículas o la física virtual.

Un aspecto importante en todas estas aproximaciones a la coordinación de enjambres de robots es la necesidad de validar los comportamientos durante el proceso de construcción. Los simuladores basados en física son una herramienta que ha sido utilizada como medio de validación, ya que estos sistemas permiten recrear las propiedades de los robos reales y del entorno, sin los gastos de construir un gran número de robots reales.

El presente capítulo tiene como objetivo exponer los conceptos y fundamentos detrás de la robótica de enjambres evolutiva como técnica para la construcción automática de comportamientos para enjambres de robots, y cómo la auto-organización puede ser generada, en un grupo de robots, construyendo de forma automática programas de control utilizando técnicas de computación evolutiva y herramientas de simulación.

Este capítulo está organizado de la siguiente manera. La sección II expone los conceptos básicos de los sistemas multi-robot, en la sección III se presenta la aplicación de la inteligencia de enjambres a los sistemas multi-robot. En la sección IV se ilustra cómo la evolución artificial ha permitido diseñar comportamientos de auto-organización para robots. En la sección V se indican sistemas de simulación como herramientas de validación en enjambres de robots. Finalmente, en la sección VI se presenta cómo la programación genética ha sido utilizada para coordinar enjambres de robots.

1.2. Sistemas multi-robot

Un sistema multi-robot es una colectividad de robots desplegada para resolver tareas complejas en entornos desconocidos [3], [23]. Desde su aparición, estos sistemas han atraído el interés de muchos investigadores, unos ven en ellos la oportunidad para desarrollar, desde el punto de vista de la ingeniería, complejos sistemas robóticos con propiedades de versatilidad y robustez [19], y otros ven en estos sistemas una forma de estudiar los sistemas cognitivos. Los sistemas multi-robot surgieron como una respuesta a las limitaciones y dificultades encontradas cuando se intentó aplicar un único sistema robótico para resolver problemas complejos en entornos dinámicos [51]. Como es de esperar, un robot monolítico debe contener todos las herramientas necesarias para resolver el problema lo que incrementa la complejidad del sistema, y cualquier falla afecta de forma crítica el desempeño del robot. Por el contrario, una colectividad de robots ofrecen (por lo menos conceptualmente), una especie de paralelismo en la ejecución, versatilidad (heterogeneidad del conjunto), y redundancia dada por el uso de múltiples robots. Sin embargo, estos beneficios traen consigo un costo: la coordinación del grupo de robots en torno a la ejecución colectiva de una tarea es inherentemente compleja. Las interacciones entre los robots y el entorno, y la dinámica misma del entorno son difíciles de modelar y adaptar para resolver un problema determinado. Una de las aproximaciones para abordar estas dificultades surgió de las observaciones, y extrapolación a los sistemas multi-robot, de los enjambres naturales y sus propiedades de organización descentralizada: la robótica de enjambres [51].

1.3. Robótica de enjambres

La robótica de enjambres surgió de la aplicación de la inteligencia de enjambres a los sistemas multi-robot [51]. El término inteligencia de enjambres fue utilizado inicialmente para hacer referencia al estudio de los autómatas celulares, posteriormente, el término denotó un conjunto de técnicas de optimización bioinspirada (Ant Colony Optimization, Particle Swarm Optimization) y estudios de sociedades de insectos [54].

La robótica de enjambres estudia cómo un grupo de robots autónomos puede ejecutar colectivamente una tarea sin un mecanismo de control centralizado. Dorigo et al [47] definieron cuatro características importantes de la robótica de enjambres.

La primera característica es el número de robots: si bien, no definen un número específico de robots, los mecanismos de coordinación bajo esta aproximación deben coordinar un conjunto grande de robots. Investigaciones en este campo han reportado el uso de entre 20 [9] y 1000 robots [53].

La segunda característica tiene que ver con las propiedades de los robots: la robótica de enjambres apalanca la coordinación de grupos homogéneos de robots, es decir, robots con propiedades y capacidades muy similares a nivel de las interacciones. Sin embargo, los grupos heterogéneos no están excluidos de esta aproximación (se han reportado investigaciones con grupos heterogéneos de robots [7], [46]). Adicionalmente, se considera que los robots son autónomos y se encuentran en un entorno con el cual interactúan.

La tercera característica se refiere al tipo de tareas que son abordadas por el enjambre: la tarea debe requerir una colectividad para ser resuelta, o por lo menos debe mejorarse de forma considerable la solución comparada con la aplicación de un sólo robot.

Finalmente, la cuarta característica tiene que ver con la escalabilidad del conjunto: los robots deben poseer capacidades limitadas (locales) de percepción del entorno y de comunicación. Mecanismos de coordinación centralizado e intercambio de conocimiento global afectan considerablemente el desempeño a medida que el número de robots aumenta. Escalabilidad es una de las premisas en el diseño de mecanismos de coordinación para enjambres de robots.

Los estudios en robótica de enjambres han sido motivados por tres propiedades identificadas por Sahin [45]: robustez, flexibilidad y escalabilidad. En este contexto, la robustez hace referencia a la capacidad de tolerar fallas en los individuos del enjambre, la flexibilidad es la capacidad que posee el enjambre para trabajar en un amplio rango de entornos y aplicaciones. Por su parte, la escalabilidad se refiere a la capacidad para mantener o mejorar el desempeño del enjambre a medida que crece el número de robots.

Se han propuesto varias taxonomías para clasificar la gran cantidad de investigaciones que se han realizad en el área [48], [49]. Brambilla et al [2] agrupan los trabajos en dos categorías: métodos y comportamientos colectivos. Dentro de la categoría de métodos se encuentran los métodos de análisis y diseño. En los métodos de diseño se encuentran los métodos de diseño basados en comportamientos en los que, de forma iterativa, se estudia, implementa y mejora el comportamiento para cada robot, tomando como base (la mayoría de las veces) modelos matemáticos de los comportamientos de los enjambres naturales. Los otros métodos son los métodos de diseño automático.

Los métodos de diseño automático, en la robótica de enjambres, buscan establecer un marco formal para el análisis y diseño de enjambres de robots, sin embargo, aún no hay

un método formal aceptado para establecer las reglas a nivel del individuo que produzcan el patrón deseado a nivel de enjambre. Dentro de los métodos de diseño automático se encuentran las aproximaciones de aprendizaje reforzado [24] y la robótica evolutiva [2]. En el aprendizaje reforzado se busca que los robots aprendan un comportamiento por medio de la interacción con el entorno recibiendo realiamentación positiva o negativa por sus acciones.

La robótica evolutiva es una técnica para el diseño de robots (tanto hardware como software) que utiliza técnicas de evolución artificial. En robótica evolutiva el componente que se desea desarrollar es codificado en un genotipo utilizando algún tipo de representación, posteriormente una población de genotipos es sometida a un proceso iterativo en el que se le aplican operaciones de combinación y mutación hasta encontrar el componente adecuado. Este método ha demostrado ser efectivo para el diseño de sistemas robóticos y ha sido aplicado al diseño de enjambres de robots [35]. La aplicación de técnicas de computación evolutiva al diseño de enjambres de robots dio lugar a la robótica de enjambres evolutiva [6], [51].

1.4. Robótica de enjambres evolutiva

La evolución artificial ha sido aplicada en el diseño de comportamientos para enjambres de robots. Algunos de los trabajos pioneros en esta área incluyen los experimentos realizados por Werner et al [54], en los que aplicaron coevolución para el estudio de comportamientos presa-depredador en poblaciones de individuos simples; Reynolds [55] y su trabajo de diseño del sistema de control de un grupo de criaturas para escapar de un depredador, y más recientemente, Byington et al [50] desarrollaron controladores para un enjambre de robots que realizan locomoción cooperativa utilizando algoritmos genéticos; Suthambut et al [35] evolucionaron controladores para fútbol de robots con redes neuronales artificiales; por su parte Larik et al [38] evolucionaron estrategias para fútbol de robots.

El algoritmo de evolución artificial utilizado en la robótica de enjambres tiene la estructura que se presenta en la figura 1.1. El algoritmo trabaja con poblaciones de individuos (comportamientos). En el primer paso se crea una población inicial de comportamientos de forma aleatoria. Posteriormente, se evalúa cada comportamiento, en este paso se asigna cada comportamiento a los robots y se determina su desempeño de acuerdo con una métrica definida por el usuario. En el tercer paso se verifica si se ha cumplido con los criterios de parada, los criterios más comunes son alcanzar un número de generaciones determinado o llegar al valor de desempeño establecido. En el cuarto paso se selecciona un conjunto de comportamientos para participar en las operaciones que producen la nueva generación, la selección se realiza con base en el valor de desempeño de cada comportamiento obtenido en el paso de evaluación. En el quinto

paso se aplican sobre los comportamientos seleccionados operaciones de cruce y mutación, la operación de cruce produce dos comportamientos nuevos combinando partes tomadas aleatoriamente de dos comportamientos seleccionados; el operador de mutación cambia una parte, de forma aleatoria, de un comportamiento seleccionado. Los comportamientos obtenidos por cruce y mutación sustituyen a la población actual y se convierten en la población a la que se aplican nuevamente los pasos tres al cinco.

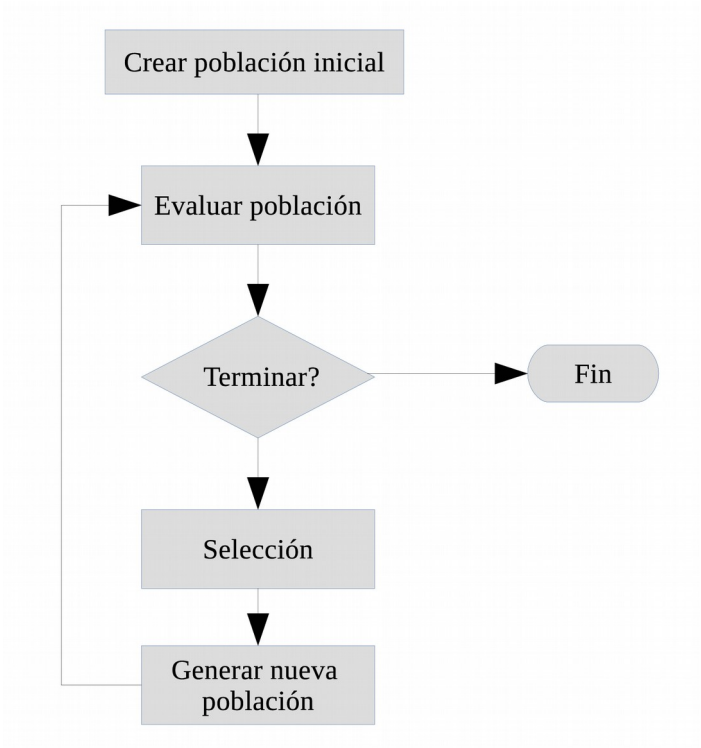


Figura 1.1. Algoritmo básico de evolución artificial utilizado en robótica evolutiva.

Trianni [51] señaló que el uso de evolución artificial para el diseño de enjambres de robots permite abordar dos aspectos del problema de diseño: el primero es la descomposición del comportamiento de alto nivel del enjambre en los comportamientos de los individuos que lo producen, el segundo aspecto es la implementación de los comportamientos individuales en los controladores de los robots. En la robótica de enjambres evolutiva el controlador y sus parámetros son codificados en el genotipo de los individuos del algoritmo de evolución artificial. Los controladores y sus respectivos parámetros construidos por evolución artificial determinan el comportamiento de cada robot del enjambre.

Dos de los esquemas más utilizados para codificar los controladores y sus parámetros (comportamientos) son: las redes neuronales artificiales [17] y las máquinas de estados finitos probabilísticos [9].

En robótica de enjambres una red neuronal crea una relación entre el sistema de percepción del robot y su sistema de efectores, de este modo, la red es un controlador que reacciona a los estímulos del entorno y determina el comportamiento del robot. En la figura 1.2 se presenta el esquema conceptual de una red neuronal artificial que realiza una transformación del espacio de percepciones (sensores) al espacio de acciones (efectores). La forma en que se realiza dicha transformación está determinada por los conjuntos de valores de las conexiones *L* y *M*. En robótica de enjambres evolutiva, se utiliza el algoritmo de evolución artificial para encontrar los valores *L* y *M* que definen el comportamiento para cada robot. Un individuo de la población a evolucionar está conformado por el conjunto de pesos de una red neuronal artificial predefinida. Utilizando esta aproximación se han producido comportamientos de auto-organización para enjambres de robots que realizan tarea de agregación y forrajeo colectivo [2].

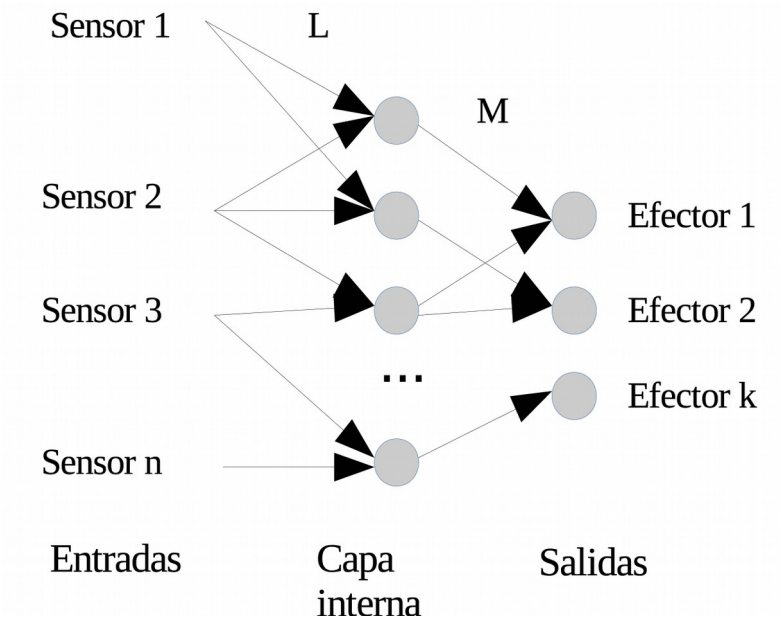


Figura 1.2. Representación de comportamientos con redes neuronales artificiales.

Otro esquema utilizado para representar comportamientos para los miembros del enjambre de robots son las máquinas de estado finitos probabilísticos. A diferencia de las redes neuronales artificiales, los máquinas de estado no realizan transformaciones de bajo nivel entre los sistemas de percepción y de efectores de los robots sino que, definen el comportamiento de un robot como una combinación de módulos o funciones, el flujo entre los módulos está determinado por los valores de transición entre los estados de la máquina. Los módulos o funciones son operaciones que el robot puede realizar, por ejemplo, moverse en una dirección determinada, rotar en cierta dirección o

cualquier otra operación más compleja. En la figura 1.3 se ilustra cómo una máquina de estados puede codificar un comportamiento para un robot.

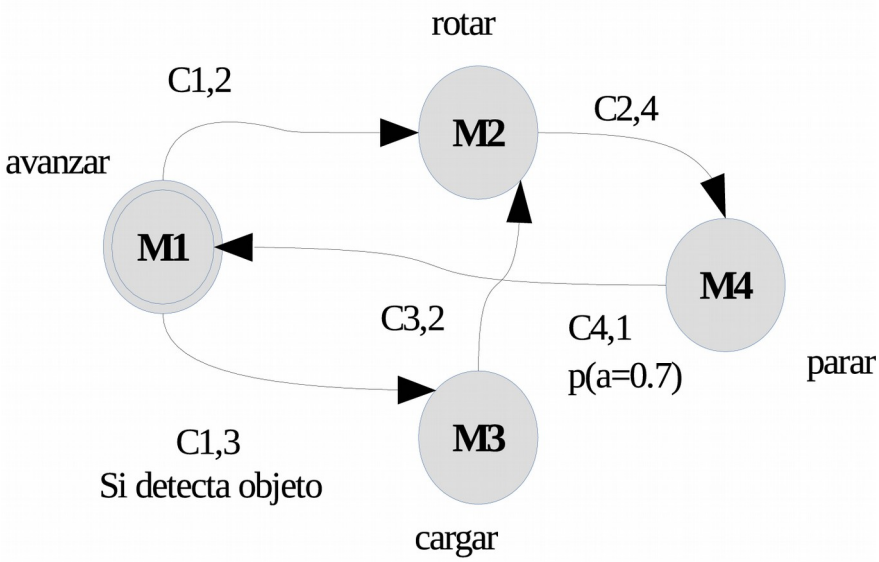


Figura 1.3. Representación de comportamientos con máquinas de estados.

Cuando se utilizan máquinas de estados para representar comportamientos el genotipo de cada individuo de la población codifica la estructura de la máquina de estados así como los valores de transición entre los estados. De este modo el resultado del algoritmo evolutivo es una combinación adecuada de estados con valores ajustados de transición entre dichos estados que representan el comportamiento para de cada robot. La transición entre estados son típicamente umbrales de variables o percepciones del entorno.

En la mayoría de trabajos de investigación se evolucionan comportamientos para un grupo homogéneo de robots, por lo que el resultado del algoritmo evolutivo es un único comportamiento el cual es asignado a todos los robots.

1.5. Simulación de enjambres de robots

Uno de los aspectos más importantes que interviene en el desarrollo de mecanismos de coordinación para enjambres de robots es la necesidad de disponer de un gran número de robots. Los comportamientos individuales, por un lado, son diseñados para que al ser ejecutados, por una colectividad de robots, se produzca el comportamiento de alto nivel planeado, y por el otro, pensando en su escalabilidad, la cual sólo puede ser verificada con un número grande de robots. Si bien, se han desarrollado algunas plataformas

robóticas para la investigación en robótica de enjambres [7], [26], [52], los costos, los requerimientos de espacio (escenarios de prueba) y el despliegue de una gran cantidad de robots son aún una limitante.

Con el fin de afrontar estas limitaciones se han desarrollado simuladores basados en física que reproducen, con cierto grado de similitud, las condiciones encontradas en escenarios reales. Estos simuladores permiten reproducir las características de plataformas robóticas típicas utilizadas en investigaciones, así como la definición de plataformas completamente nuevas. Los primeros simuladores utilizados en la investigación de robótica de enjambres estaban enfocados al diseño de robótica general y no escalaban de forma adecuada con el número de robots.

Se han desarrollado varios simuladores enfocados a simular conjuntos grades de robots, dentro de lo más destacados se encuentran: Stage [58], Vrep [57], Gazebo [59], Webots [56] y Argos [60]. Estos simuladores proveen un entorno virtual para realizar las validaciones, muchos de ellos proporcionan características adicionales tales como un API para desarrollo e interfaces con frameworks de robótica como ROS.

ROS [67] (Robot Operating System) es un conjunto de librerías, herramientas y convenciones para el desarrollo de software para robots. Este framework proporciona un mecanismo para que programas en un entorno distribuido intercambien información, además de otro amplio rango de herramientas y programas, documentación y amplio soporte por parte de la comunidad en el área de la robótica. Este framework utiliza el concepto de grafo (ROS Graph) para estructurar el software de robots, ROS Graph es un conjunto de nodos (procesos de computador) que utilizan el modelo Publicador/suscriptor para el intercambio de información. Cada nodo publica y se suscribe a canales de comunicación, denominados ROS Topics, por los que se envía un tipo específico de información denominado ROS Message. Adicionalmente, posee un conjunto amplio de herramientas de depuración e introspección. ROS es auspiciado por la OSRF [68] (Open Source Robotics Foundation).

Stage: es un simulador de entornos y robots en dos dimensiones, las simulaciones se limitan sólo a la cinemática de los elementos del entorno y no es posible simular ruido en los sensores y otros elementos. Puede simular poblaciones de hasta mil robots.

Vrep: es un simulador de entornos y robots en tres dimensiones. Es un simulador versátil que incluye motores de física tales como Bullet y ODE (Open Dynamics Engine). Permite simular la dinámica, interacción de diversos elementos físicos y sensores, posee soporte para extensiones y programas en diversos lenguajes (java, C, Python), así como la interacción directa con hardware específico. Posee un conjunto predefinido de modelos de robots, escenarios y sensores listos para usar.

Gazebo: es un simulador de código abierto impulsado por la OSRF. Al igual aque Vrep, es un simulador de entornos en tres dimensiones y posee motores de física (ODE). Es extensible por medio de módulos en el lenguaje C++. Posee interfaces con frameworks

de robótica tales como ROS (Robot Operating System) y cuenta con una amplia base de datos de modelos (robots, sensores, escenarios).

Webots: es un simulador multiplataforma de entornos y robots en tres dimensiones, utiliza el motor de física ODE, cuenta con modelos predefinidos y un API (Application Programming Interface) en seis lenguajes de programación. Permite simular mecanisos de comunicación entre robots. Posee editores de código y de escenas. Permite conexiones con frameworks y aplicaciones tales como Matlab y ROS.

Argos: simulador de entornos y robots en tres dimensiones especialmente diseñado para enjambres heterogéneos de robots. Puede ejecutar de forma simultánea varios motores de física. Cuenta con extensiones para ROS. Los desarrolladores han mostrado simulaciones de diez mil robots de forma simultánea.

1.6 Programación genética y robótica de enjambres

Programación genética (PG) es un tipo de algoritmo genético que evoluciona poblaciones de programas, es una aproximación al diseño automático de programas para problemas complejos [44]. En general, los comportamientos para robots, producidos por los mecanismos de diseño, son programas que se ejecutan en los computadores a bordo de los robots. El potencial de la programación genética para construir programas, para resolver problemas complejos, fue vislumbrado por la comunidad de la robótica de enjambres como una posible solución al problema de coordinación y PG empezó a ser utilizado como una aproximación a la inducción de comportamientos de auto-organización para enjambres de robots.

En varios trabajos de investigación se ha reportado el uso de programación genética para coordinar enjambres de robots. Uno de los primeros reportes fue realizado por Luke et al [61] en su trabajo con fútbol de robots, utilizando programación genética, sintetizaron comportamientos de cooperación (ataque y patear la pelota) para jugar fútbol en la competencia virtual de RoboCup Soccer. Por su parte, Messom y Walker [36] diseñaron (para el dominio de fútbol de robots) comportamientos cooperativos para dos robots que evaden obstáculos. Otro problema en el que se ha empleado programación genética, dentro del campo de la robótica de enjambres, es la planeación de trayectorias. Kala [39] utilizó programación genética y coevolución para generar estrategias de cooperación para un grupo de robots que se mueven dentro de un laberinto, los robots cooperan para generar y seguir trayectorias óptimas. Macedo et al [62] aplicaron programación genética al problema particular de generar controladores para detectar fuentes de olor, en este escenario un grupo de robots explora el entorno en busca de olores (agentes químicos) una vez perciben un potencial agente deben trazar la ruta hasta la fuente y determinar su posición exacta; los investigadores evolucionaron

programas para explorar el entorno, detectar el olor y rastrear su fuente, y declarar su localización.

En programación genética no se construye de forma directa la solución al problema que se trata, sino que se construyen programas que resuelven el problema. Para esto, se requiere representar el conjunto de instrucciones que definen al programa. Dentro de las representaciones más comunes en programación genética se encuentran los árboles de expresión, la representación lineal (programación genética lineal), y las representaciones basadas en grafos. En programación genética aplicada a enjambres de robots las representaciones más utilizadas son los árboles de expresión.

Los árboles de expresión o árboles de sintaxis son estructuras jerárquicas de operaciones, variables y constantes. En la terminología de programación genética, las operaciones se denominan funciones, mientras que las variables y constantes se denominan terminales. Al conjunto de funciones y terminales se denomina como el conjunto primitivo, y es el conjunto de todas los elementos posibles para construir un programa determinado. En la figura 1.4 se ilustran estos conceptos. Los árboles se construyen tomando operaciones del conjunto de funciones, colocando como hojas del árbol elementos del conjunto de terminales.

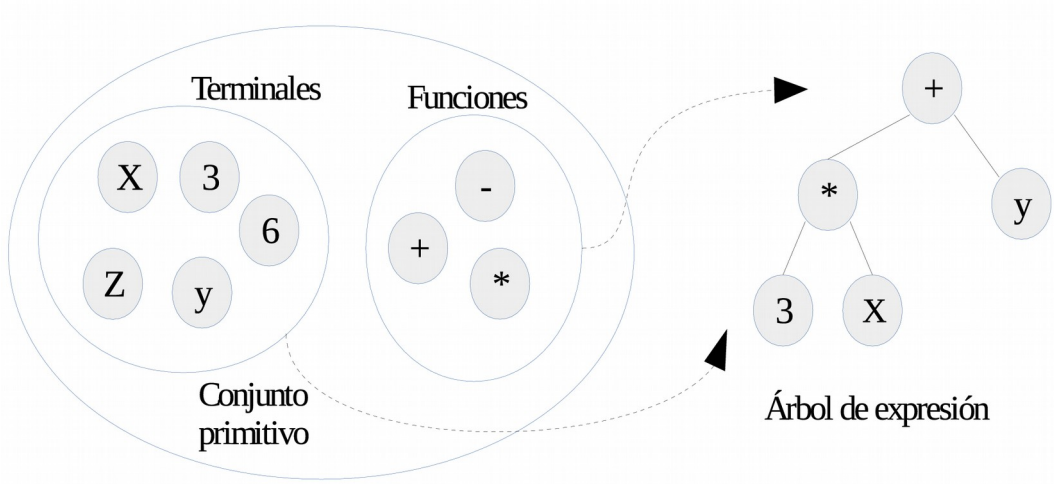


Figura 1.4. Árbol de expresión y su conjunto primitivo correspondiente.

El conjunto primitivo depende del dominio del problema al que se aplique programación genética. El algoritmo de programación genética explora el espacio de todos los programas que se pueden construir con todas las posibles combinaciones de los elementos del conjunto primitivo.

En robótica de enjambres, el conjunto primitivo puede estar compuesto de operaciones en diferentes niveles de complejidad. Por ejemplo, las funciones de movimiento tales

como avanzar una determinada cantidad de pasos o rotar en una dirección establecida, o funciones más complejas tales como una rutina de movimiento con evasión de obstáculos pueden constituir el conjunto de funciones, mientras que procedimientos como obtener el valor de un sensor pueden hacer parte del conjunto de terminales. En la figura 1.5 se ilustra un árbol de expresión en robótica de enjambres.

Las aplicaciones de PG más comunes imponen restricciones de tipo en sus funciones, esto implica que los valores de retorno de las funciones, sus parámetros y otras variables tienen un tipo de dato específico, esta aproximación se conoce como programación genética fuertemente tipada [44]. En programación genética fuertemente tipada los operadores genéticos deben preservar la consistencia del árbol, esto es, los tipos de un nodo padre y un nodo hijo son compatibles (poseen el mismo tipo). Estas restricciones aseguran que una función reciba y opere sólo la información que esta puede procesar, por ejemplo, las funciones que procesan información únicamente de un cierto tipo de sensor deben estar asociadas sólo a funciones que retornan datos de ese sensor.

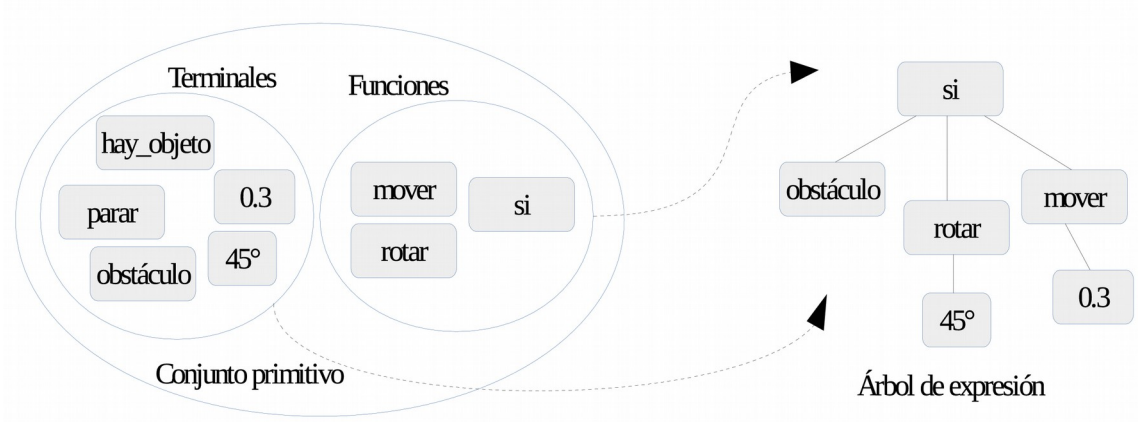


Figura 1.5. Ejemplo del conjunto primitivo y árbol de expresión para representar comportamientos para enjambres de robots.

Otra de los representaciones utilizadas para evolucionar comportamientos en enjambres de robots con PG es la representación lineal [44]. En la representación lineal los programas son secuencias lineales de instrucciones, tal como se ilustra en la figura 1.6. Uno de los dominios en el que se ha aplicado la representación lineal es en la planeación cooperativa de trayectorias [39], como se mencionó al inicio de esta sección.

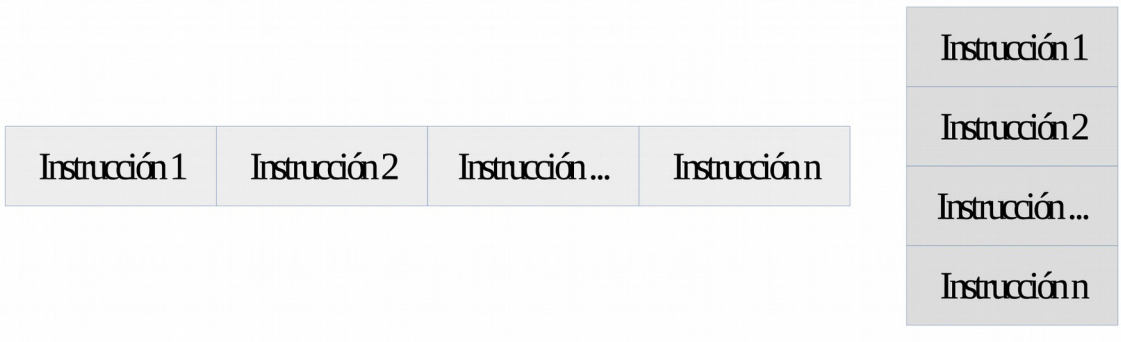


Figura 1.6. Representación lineal de programas en PG.

Gran parte del trabajo realizado con PG para producir comportamientos cooperativos se ha enfocado en enjambres homogéneos de robots, los enjambres están constituidos por varias instancias del mismo robot móvil. De esta forma, el conjunto primitivo es único para todo el enjambre, PG evoluciona poblaciones de potenciales comportamientos a partir de este conjunto, para el cálculo del valor de desempeño cada miembro de la población es asignado a todos los robots del enjambre, es decir, un individuo se evalúa sobre todo el enjambre.

Son muy pocos los trabajos de investigación en los que se ha utilizado PG para el diseño de comportamientos para enjambres heterogéneos, siendo fútbol de robots el dominio en el que más se ha aplicado. En estos casos, el conjunto primitivo es el mismo para todos los robots y se busca generar especialización realizando agrupaciones (grupos de árboles para cada especialidad) y aplicando operadores genéticos con restricciones sobre dichos grupos [38].

CAPÍTULO 2

Plataforma para la coordinación de robots con estrategias evolutivas

2.1. Introducción

Coordinar un conjunto de robots en el ámbito de la robótica de enjambres, consiste en encontrar el conjunto de reglas que llevan a la auto-organización del sistema. Con el fin de encontrar el conjunto de comportamientos a nivel individual que conducen al surgimiento de un patrón deseado a nivel de sistema, en este trabajo, se adoptan las ideas introducidas por Trianni et al [51] en las cuales, se plantea el uso de evolución artificial como herramienta de diseño automático de sistemas que exhiben auto-organización.

La tarea de diseñar los comportamientos para cada uno de los miembros que conduzcan a un resultado emergente deseado es difícil (problema de diseño), entre otras cosas, por la complejidad inherente del sistema. En general, se trata de descomponer el patrón global del conjunto en simples interacciones entre los miembros del conjunto de robots teniendo en cuenta la dinámica del entorno y luego, codificar tales interacciones en controladores para los robots.

Para abordar estas dificultades, en algunos trabajos de investigación se ha propuesto el uso de algoritmos de optimización que, junto con representaciones de los controladores en máquinas de estado y redes neuronales, construyen y sintonizan los comportamientos para un conjunto de robots [9]. Sin embargo, la carencia de metodologías de diseño, entornos y herramientas de desarrollo integrados, dificulta el desarrollo automático de comportamientos para la auto-organización de enjambres de robots. Este capítulo presenta una plataforma computacional, para el diseño automático de comportamientos para un grupo de robots heterogéneos, en al ámbito de la robótica de enjambres evolutiva. Esta plataforma permite describir robots, tareas y comportamientos; simular robots y construir el software de control para dichos robots de forma automática utilizando evolución artificial.

El capítulo está organizado como sigue: en la sección II se presenta el diseño conceptual de la plataforma computacional, en la sección III se presenta el formato de descripción para la especificación de los robots, las tareas y los comportamientos primitivos en la plataforma computacional. En la sección IV se presenta el sistema de simulación utilizado para la validación de los comportamientos construidos durante el ciclo

evolutivo, simular el entorno objetivo y el enjambre de robots. En la sección V se presenta la implementación de la plataforma computacional.

2.2. Plataforma computacional

Buscando un entorno unificado para construir comportamientos de auto-organización para un conjunto de robots con propiedades y capacidades diversas (heterogéneos), se ha planteado el desarrollo de un entorno computacional que administre la definición del enjambre y las tareas que realiza el enjambre, construcción automática del software de control (comportamientos) para cada miembro del enjambre, y el despliegue de los comportamientos en el conjunto de robots.

Con el fin de describir los modelos de los robots de un enjambre y las tareas o problema que resuelve el enjambre, se adjunta al entorno computacional un formato de descripción. El formato de descripción permite definir los aspectos más relevantes de los individuos del enjambre relacionados con sus propiedades mecánicas, elementos de percepción del entorno y efectores. De igual manera, se hace necesario la especificación del problema que se desea que el conjunto de robots resuelva de forma auto-organizada, para lo cual, el formato de descripción permite definir una tarea en términos de objetivos y restricciones predeterminadas. El formato de descripción es una lenguaje basado en etiquetas XML.

Por otro lado, la plataforma computacional, siguiendo la aproximación del área de la robótica evolutiva al problema de diseño [51], integra un mecanismo basado en evolución artificial para la construcción automática de comportamientos de alto nivel a partir de comportamientos primitivos. Esta es una aproximación Bottom-UP en la que programas de control complejos, en la forma de árboles de expresiones, son inducidos a partir de programas simples utilizando programación genética. Cada árbol de expresión representa un programa que es ejecutado por un intérprete (controlador) en el computador abordo de los robots. Los programas simples o comportamientos primitivos son definidos en la plataforma computacional, de la misma forma que el enjambre y las tareas, a través del formato de descripción.

En cada uno de las ciclos del proceso evolutivo los comportamientos generados son evaluados en un entorno virtual. El entorno virtual contiene el modelo del espacio físico donde se despliegan los robots, así como los modelos de cada uno de los miembros del enjambre. En el sistema de simulación se evalúa la dinámica global del enjambre que surge a partir de los comportamientos representados en los árboles de expresión, que a su vez llevan a las interacciones robot-robot y robot-entorno que producen el patrón de auto-organización deseado, tal como lo describe Trianni [51]. La función de aptitud se

define como parte de la tarea a realizar en el formato de descripción. Esta función cuantifica el desempeño del enjambre en cada ciclo evolutivo.

Finalmente, en la plataforma computacional se han incluido módulos de comunicación para asignar o desplegar a los robots los comportamientos construidos al final del proceso evolutivo, esto con el fin de facilitar la asignación de los comportamientos y validar su efecto en robots reales. El esquema conceptual planteado para la plataforma computacional se ilustra en la figura 2.1.

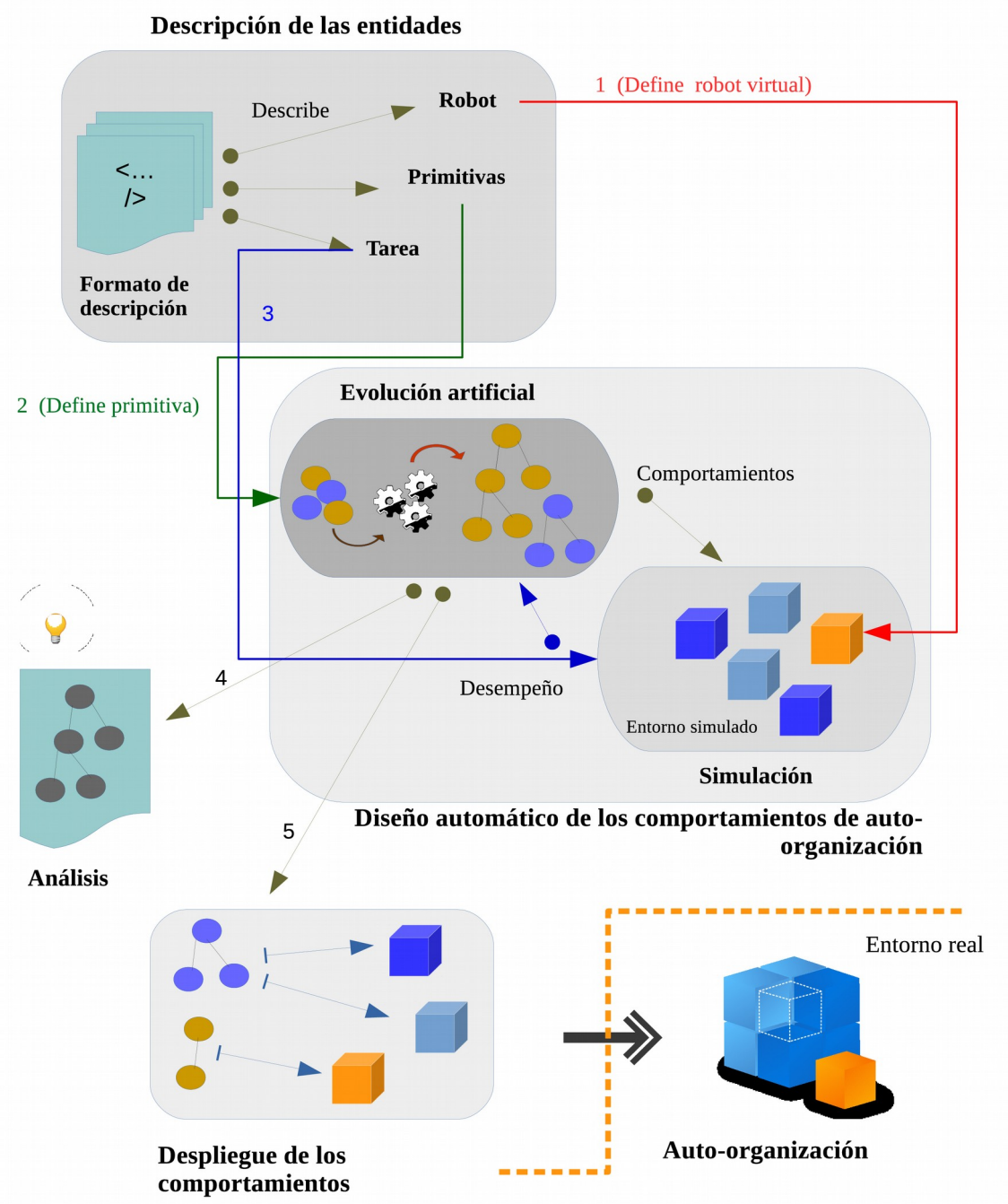


Figura 2.1. Esquema conceptual de la plataforma computacional.

En el formato de descripción (parte superior de la figura) se definen (por medio de un conjunto de marcas) los robots, las primitivas y la tarea que el enjambre debe realizar. A partir de la descripción de los robots se crea una representación virtual de los mismos en el simulador (flecha 1), así como una representación virtual del entorno físico en el que los robots interactúan. De forma similar, a partir de la descripción de los comportamientos primitivos (flecha 2), se definen las primitivas a partir de los cuales se construyen los comportamientos de auto-organización para los robots.

En la parte central del esquema se ilustra la interacción entre el mecanismo de construcción automática de comportamientos de auto-organización y el entorno virtual. Una implementación de programación genética construye y envía programas de alto nivel (comportamientos) al simulador en el que los robots virtuales los ejecutan y se evalúa el desempeño del enjambre. La descripción de la tarea define cómo se evalúa el desempeño (flecha 3), en este trabajo se adopta el llamado de funciones definidas en el formato de descripción para evaluar el desempeño de los individuos tal como es presentado en la sección 2.3.

En la parte inferior del esquema se ilustran dos posibles acciones a realizar una vez el ciclo de generación de comportamientos ha finalizado. Los comportamientos pueden ser asignados directamente a los robots físicos o quedar disponibles para analizar las reglas de auto-organización generadas durante el proceso. En la plataforma computacional se ha incluido un sistema para la comunicación, asignación y ejecución de los programas generados a robots reales.

2.3. Formato de descripción

Con el fin de proveer un mecanismo para la descripción de los elementos requeridos en el proceso de construcción de comportamientos de auto-organización, se ha desarrollado un formato de descripción para la plataforma computacional que permite especificar los siguientes tres elementos: miembros del enjambre, tareas que realiza el enjambre y un conjunto de comportamientos primitivos (primitivas) que posee cada robot del enjambre. Este formato de descripción es un lenguaje de marcas compuesto de un conjunto de elementos XML para especificar las propiedades más relevantes de los robots, las tareas y el conjunto primitivo. El formato de descripción planteado es nombrado en este trabajo como lenguaje de descripción de enjambres (Swarm Specification Language).

SSL se define con un conjunto de marcas para la descripción jerárquica de entidades como se describe a continuación. SSL contiene marcas para definir tres entidades atómicas y dos entidades compuestas. Las entidades atómicas son descripciones de objetos físicos (robots) o abstractos (tareas y primitivas). Las entidades atómicas en SSL

son: la descripción de un robot, la descripción de una tarea y la descripción de un grupo de primitivas. Una entidad compuesta es una agrupación lógica de entidades atómicas.

Por otro lado, las entidades compuestas en SSL son: la descripción de un enjambre, que asocia un conjunto de robots con un conjunto de primitivas, y la descripción de un proceso auto-organizado que asocia un enjambre con una tarea que se quiere que resuelva o ejecute dicho enjambre. En la figura 2.2 se muestran las relaciones que pueden establecerse entre las entidades en SSL. Una entidad robot posee un conjunto de primitivas (indican las habilidades básicas del robot), un enjambre está compuesto por una agrupación de robots y la entidad proceso relaciona un enjambre con una tarea.

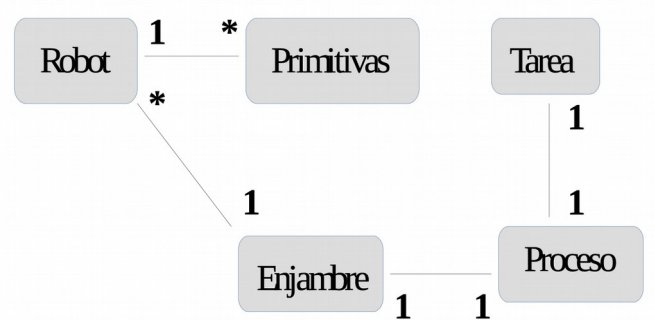


Figura 2.2. Entidades SSL.

Figura 2.3. Formato de descripción para la plataforma computacional.

En la figura 2.3 se ilustra la estructura del lenguaje SSL. Cada entidad en SSL se define a través de un conjunto de marcas XML. SSL contiene al formato de descripción SDF (Simulation Description Format) [63], como formato para la descripción de los robots del enjambre.

2.3.1. Descripción de los robots

Según lo expresa Brooks [64], un robot se desenvuelve en un entorno (situatedness) y actúa en el entorno (embodiment), y esto lo que define el comportamiento de un robot. En esta aproximación de la robótica basada en comportamientos, es necesario que un robot sea una entidad física capaz de percibir e interactuar con el entorno.

El formato de descripción propuesto para la plataforma computacional permite definir un robot en tres aspectos: propiedades físicas del robot, sistema de percepción del robot y los efectores del robot. El primer aspecto, propiedades físicas, permite modelar al robot como una entidad física en un entorno (embodiment) virtual, mientras que los aspectos de percepción y efectores permiten definir los elementos que el robot virtual posee para percibir e interactuar con el entorno (situatedness), es decir, simular los sensores y efectores que posee el robot.

En este trabajo se utiliza el formato de descripción de simulaciones (SDF, por sus siglas en inglés) [63], como un formato embebido dentro del formato de descripción SSL de la plataforma computacional, para realizar las descripciones de los robots y entorno de simulación que son instanciados dentro del simulador para realizar las validaciones de los comportamientos construidos durante el proceso evolutivo.

El formato de descripción SSL permite definir las siguientes propiedades físicas de un robot utilizando el formato SDF:

- Conjunto de eslabones (sus dimensiones y posiciones) y juntas que posee el robot.
- Propiedades inerciales de los eslabones: masa y momentos de inercia de los eslabones.
- Propiedades visuales de los eslabones: geometrías, colores y texturas.

El formato de descripción permite definir los siguientes sistemas de recolección de información del entorno para un robot:

- Sistemas LIDAR (Ligth Detection and Ranging) en 2D y 3D.
- Cámaras RGB y RGB-D.
- Sensores inerciales: IMU (Inertial Measurement Unit) y giroscopios.

- Sensores de contacto.

El formato de descripción permite definir los siguientes efectores:

- Motores (por medio de juntas).
- Cualquier composición de eslabones, juntas y motores que representen un elemento que interactúe con el entorno.

Adicional a la descripción física, de sensores y efectores, se han definido los siguientes elementos para especificar componentes de comunicación y asignación implementados en la plataforma computacional:

- Manejador (manager): elemento para especificar el administrador encargado de asignar el comportamiento al robot.
- Canal (channel): elemento para especificar el nombre del canal de comunicaciones con los robots reales.

Tanto SDF como el formato de descripción de la plataforma computacional están definidos en el lenguaje XML. En la figura 2.4 se ilustra la descripción de un robot en el formato SSL.

La definición de un robot se realiza en dos archivos de descripción. En el primer archivo, se describen las propiedades mecánicas, sensores y efectores que posee el robot, utilizando el conjunto de elementos del formato de descripción SDF. En el segundo archivo se referencia el archivo de descripción SDF y se definen los canales de comunicación para el despliegue de los comportamientos en los robots.

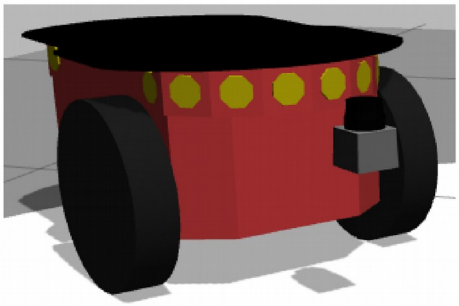
El elemento *robot* marca el inicio de la descripción de un robot en SSL, dentro de este elemento se coloca el elemento *dbfDefinition* o *fileDefinition* para referenciar la ubicación de la descripción SDF del robot, *dbfDefinition* se utiliza para hacer referencia un robot localizado en la base de datos del simulador (que incluye archivos CAD externos), mientras que el elemento *fileDefinition* referencia un archivo de texto plano que describe al robot en el formato SDF. El elemento *manager* especifica el administrador de comunicaciones para la asignación del comportamiento construido al robot físico. Finalmente, el elemento *channel* indica el nombre del canal de comunicación por el cual se asigna el comportamiento. Los dos últimos elementos, *manager* y *channel*, son específicos de la implementación de la plataforma computacional y representan módulos que administran la comunicación con los robots físicos.

Descripción SDF

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="pioneer2dxa">
    <link name="chassis">
      <pose>0 0 0.16 0 0 0</pose>
      <inertial>
        <mass>5.67</mass>
        <inertia>
          <ixx>0.07</ixx>
          <iyy>0.08</iyy>
          <izz>0.10</izz>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyz>0</iyz>
        </inertia>
      </inertial>
      ...
<plugin filename="libpcerobotcontroller.so" name="diff_drive"/>
      <include>
        <uri>model://hokuyo</uri>
        <pose>0.35 0 0.2 0 0 0</pose>
      </include>
  </model>
</sdf>
```

Cabecera

Información inercial



Robot virtual definido en SSL

Información del controlador

Información de los sensores

Descripción SSL

```
<?xml version="1.0" encoding="UTF-8"?>
<ssl:robot
   xmlns:ssl='http://ssl/entities/robot'
   name="pioneer2dxa">
   <ssl:dbDefinition>model://pioneer2dxa</ssl:dbDefinition>
   <ssl:manager>ROSManager</ssl:manager>
   <ssl:channel>/pioneer2dxa/controller</ssl:channel>
</ssl:robot>
```

Referencia a la descripción SDF

Canales de comunicación

Figura 2.4. Definición de un robot en el formato de descripción de la plataforma computacional. Descripción física en SDF (parte superior), Descripción completa en SSL (parte inferior).

2.3.2. Descripción de las tareas

Dos de los aspectos más importantes en el proceso de construcción automática de comportamientos, es la habilidad para evaluar el desempeño de las soluciones candidatas y la capacidad para guiar el proceso de construcción de los comportamientos

para los robots, con el fin de que éstos alcancen uno o varios objetivos deseados (tarea) [51]. Uno de los métodos más utilizados para evaluar el desempeño en evolución artificial son las funciones de aptitud. En este trabajo se adopta el uso de funciones para guiar el proceso de construcción de los comportamientos. Una tarea se define, en la plataforma computacional, como un conjunto de funciones (objetivos) que son llamadas durante la fase de evaluación de un comportamiento construido. Estas funciones son ejecutadas en un orden concreto, este orden en la ejecución de las funciones se denomina aquí como las restricciones de la tarea.

Esta aproximación permite modelar tareas en términos de objetivos y restricciones. Se han definido dos tipos de restricciones para los objetivos: secuencia y ventanas de tiempo. La restricción de secuencia impone un orden secuencial en el llamado de las funciones: un objetivo es ejecutado justo después de otro en el orden establecido. La restricción de ventana de tiempo define un retardo en el llamado de cada objetivo. Estas restricciones se ilustran en la figura 2.5.

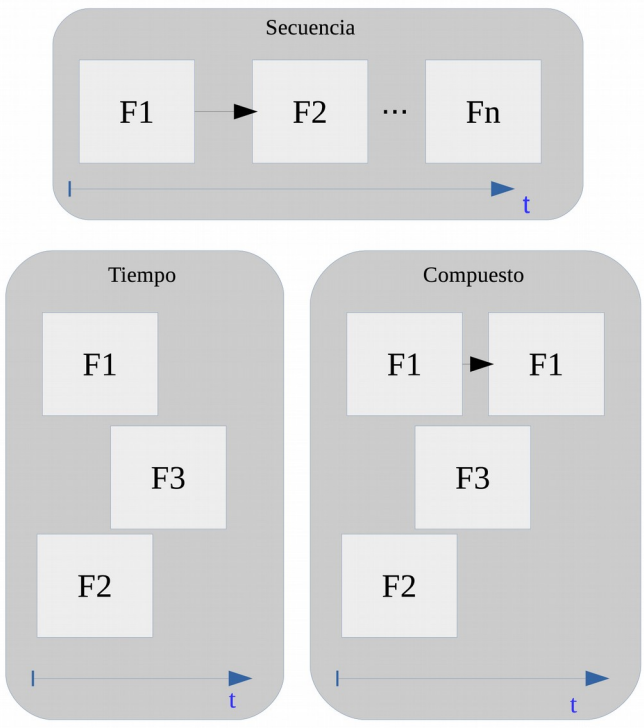


Figura 2.5. Modelos de las restricciones de las tareas soportadas en el formato de descripción. Secuencia (superior), ventana de tiempo (izquierda), ventana y secuencia (derecha).

Con el fin de definir funciones para cuantificar el desempeño y tareas para un conjunto de robots, en el formato de descripción de la plataforma computacional se ha definido

un conjunto de marcas que permiten describir un conjunto de funciones, sus parámetros y las restricciones de tiempo y secuencia. En la figura 2.6 se ilustran los elementos que definen una tarea en el formato SSL.

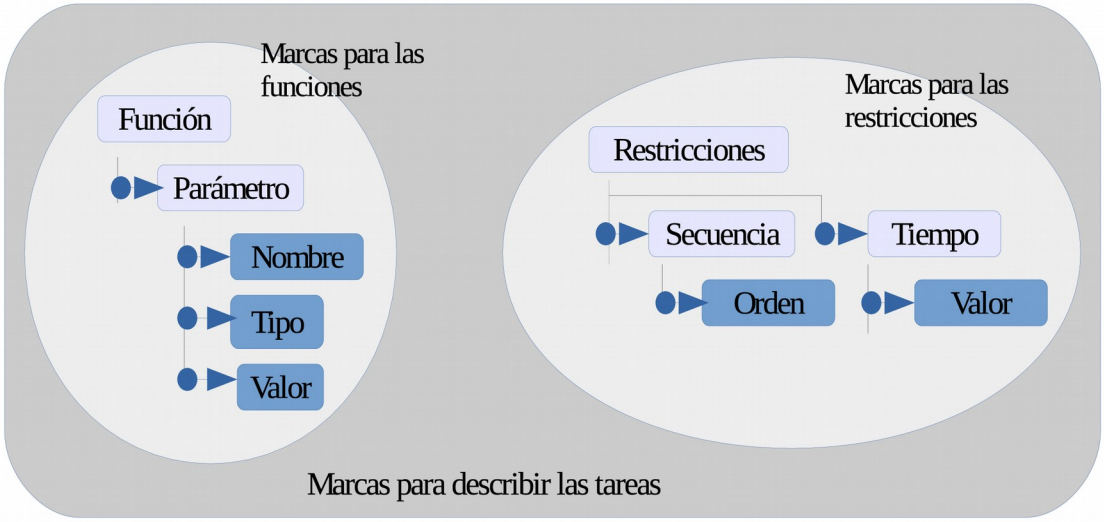


Figura 2.6. Marcas para describir tareas en el formato SSL.

Cada tarea se describe como una composición de funciones y restricciones. Cada función se define con un nombre y un conjunto de parámetros con un nombre, un tipo de dato y un valor. Las restricciones pueden ser de secuencia, tiempo o una combinación de las dos. Las restricciones de secuencia contienen una lista ordenada (referencias a las funciones), que define el orden de ejecución o llamado de las funciones. Por su parte, las restricciones de tiempo contienen una lista con un valor que indica el tiempo en el que debe ejecutarse cada función. En la figura 2.7 se presenta la definición de una tarea en el formato de descripción SSL.

La definición de una tarea se realiza con el elemento tarea (*task)*, dentro del cual se colocan los conjuntos de objetivos y restricciones. El conjunto de objetivos se encierra con el elemento objetivos (*targets)*. Cada uno de los objetivos de la tarea se define con el elemento objetivo (*goal)*, dentro del cual se colocan elementos (*arg)* según el número de argumentos que posea el objetivo (cada función representa un objetivo). Para cada argumento se debe especificar un nombre, un valor y un tipo. El conjunto de restricciones se define por medio del elemento restricciones (*restrictions)*, dentro de este elemento se colocan las restricciones según el tipo. Las restricciones de tiempo se definen por medio del elemento ventana de tiempo (*timeWindow)* seguido de un conjunto de elementos tiempo (*time)* para los cuales se debe referenciar el objetivo para el cual imponen la restricción y el tiempo de activación. Las restricciones de secuencia se definen por medio del elemento secuencia (*sequence)*, dentro del cual se colocan

elementos paso (*step)*. Para cada elemento paso (*step)* se debe especificar el objetivo para el cual se impone la restricción. El orden en la ejecución de los objetivos está determinado por el orden de aparición de cada elemento paso (*step)*. Las funciones declaradas con el formato SSL deben estar definidas en el simulador, de forma tal que la plataforma pueda encontrarlas y ejecutarlas según se especifica en su declaración SSL.

Tarea declarada en SSL

```
<?xml version="1.0" encoding="UTF-8"?>
<task xmlns="http://ssl/entities/task"
      xmlns:r="http://ssl/entities/task/restrictions"
      name="aggregation">

  <targets name="goals">
    <goal name="goal0">
      <arg name="x" value="-8.01723" type="double"/>
      <arg name="y" value="-3.64815" type="double"/>
      <arg name="rad" value="1.0" type="double"/>
    </goal>
    <goal name="goal1">
      <arg name="param1" value="0" type="boolean"/>
      <arg name="param1" value="0" type="float"/>
    </goal>
  </targets>
  <restrictions>
    <r:timeWindow>
      <r:time for="goal0" value="60" units="s"/>
    </r:timeWindow>
    <r:sequence>
      <r:step for=”goal1”/>
      <r:step for=”goal0”/>
    </r:sequence>
  </restrictions>
  <description />

</task>
```

Tarea definida en el simulador

```
void goal0( double x, double y, double rad ){ ... }
void goal1( boolean param1, float param2 ){ ... }
```

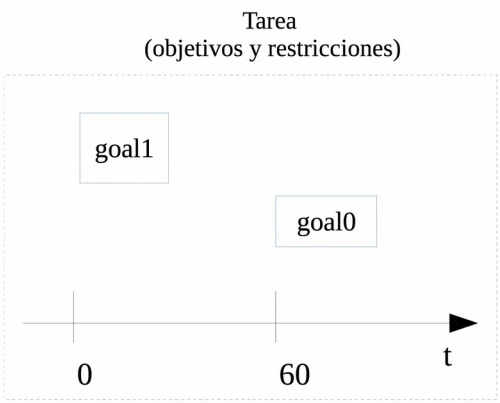


Figura 2.7. Definición de una tarea en el formato de descripción de la plataforma computacional.

2.3.3. Descripción de los comportamientos primitivos

En la robótica basada en comportamientos se define un comportamiento como el resultado de una red de conexiones que determinan cómo módulos utilizan el resultado de otros módulos [51]. En este trabajo, los comportamientos de auto-organización son

construidos conectando módulos predeterminados. Estos módulos predeterminados son nombrados aquí: primitivas. Las primitivas son módulos de software disponibles en los robots, que estos pueden ejecutar para obtener información interna o del entorno o ejecutar una acción.

Con el fin de describir primitivas que puede usar la plataforma computacional para la construcción automática de comportamientos para un grupo de robots, se ha definido un conjunto de marcas en el formato de descripción para especificar primitivas en forma de funciones. Cada primitiva es modelada como una función con parámetros y valor de retorno. Los parámetros y valores de retorno tienen un tipo de dato específico. En la figura 2.8 se ilustran los elementos de SSL para definir las primitivas.

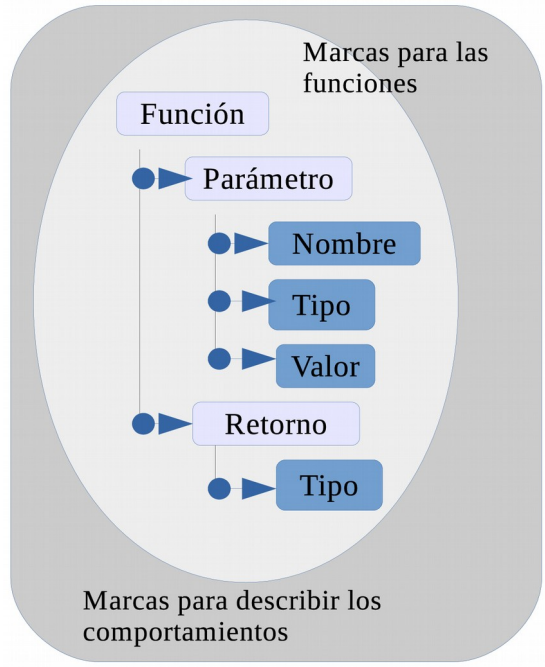


Figura 2.8. Marcas para definir primitivas en el formato SSL.

Cada robot puede tener una o varias funciones que representan su conjunto de primitivas. Como se muestra en el capítulo 3 de este trabajo, las primitivas descritas en este formato conforman, junto con sentencias de control, el conjunto primitivo de la implementación de programación genética para la inducción de programas de control para los robots. En la figura 2.9 se presenta la descripción de un conjunto de primitivas para un robot móvil en el formato de descripción propuesto.

La definición de un conjunto de comportamientos inicia con el elemento comportamiento (*behavior)*, dentro de este elemento se describen todas las funciones del conjunto. La definición de una función inicia con el elemento función (*function)* seguido de los elementos nombre *(name)*, retorno (*return)* y parámetro (*param)*, para

indicar el nombre de la función, el tipo de retorno y los parámetros de la función, respectivamente. Finalmente, el elemento descripción (*description)* encierra un texto opcional que describe al conjunto de primitivas. Las funciones o comportamientos primitivos deben estar definidos en el controlador de cada robot.

Conjunto primitivo declarado en SSL

```
<?xml version="1.0" encoding="UTF-8"?>
<behavior xmlns="http://ssl/entities/behavior"
   xmlns:p="http://ssl/control/primitive"
   name="behavior"
   type="simple">

   <function>
     <name>forward</name>
     <return>none</return>
     <param name="speedf" type="double" value="[3.0 4.5]"/>
   </function>
   <function>
     <name>rotate</name>
     <return>none</return>
     <param name="speedr" type="double" value="0.003"/>
   </function>
   <function>
     <name>stop</name>
     <return>none</return>
   </function>

        ...

   <description>
        Lista de funciones a usar como bloques constructivos.
   </description>

</behavior>
```

Conjunto primitivo definido en el controlador del robot

```
void forward( double speedf ){ ... }
void rotate( double speedr ){ ... }
void stop(){ ... }
```

Figura 2.9. Descripción de un conjunto de comportamientos primitivos en el formato de descripción de la plataforma computacional.

Además de los elementos del formato de descripción presentados hasta aquí: robot, tarea y conjunto de primitivas (definidos como entidades atómicas en la sección 2.3), se

han definido marcas para describir dos entidades compuestas: enjambre y proceso. La entidad enjambre establece una relación entre la descripción de un robot y la descripción de un conjunto de comportamientos, determina qué primitivas están asociadas con un robot en particular cuando se construyen los comportamientos de auto-organización. De forma similar, la entidad proceso permite relacionar un enjambre con la tarea que éste debe realizar. En la figura 2.10 se presentan la descripción de un enjambre y un proceso en el formato de descripción SSL.

La descripción de un enjambre se realiza con el elemento enjambre (*swarm)*, dentro de este elemento se define un conjunto de miembros. El conjunto de miembros se especifica con el elemento miembros (*members)*. Cada uno de los miembros del conjunto se define con el elemento miembro (*member)* que encierra un elemento *robot* y un elemento comportamiento (*behavior)* que asocia al robot declarado con el conjunto de comportamientos declarado. Finalmente, la descripción de un proceso se realiza con el elemento proceso (*process)*, el cual asocia un enjambre declarado con el elemento enjambre (*swarm)* con una tarea declarada con el elemento tareas (*tasks)*.

```
<?xml version="1.0" encoding="UTF-8"?>
<swarm xmlns="http://ssl/entities/swarm"
  name="HM">

  <members>
    <member>
      <robot>robot1.xml</robot>
      <behavior>behavior5.xml</behavior>
    </member>
    <member>
      <robot>robot2.xml</robot>
      <behavior>behavior5.xml</behavior>
    </member>
    <member>
      <robot>robot3.xml</robot>
      <behavior>behavior5.xml</behavior>
    </member>
  </members>

</swarm>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<process xmlns="http://ssl/process"
  name="my_first_process">

  <swarm>/swarm.xml</swarm>
  <tasks>/task.xml</tasks>

</process>
```

Figura 2.10. Descripción de un enjambre de robots (superior) y un proceso (inferior) en el formato de descripción de la plataforma computacional.

2.4. Simulación del enjambre de robots

Con el fin de recrear las condiciones encontradas en entornos físicos en los que se desenvolvería un enjambre en aplicaciones prácticas del mundo real y validar los comportamientos construidos durante el proceso de diseño automático, se ha implementado un simulador basado en física para la plataforma computacional. A partir de las descripciones realizadas en el formato de descripción presentado en la sección 2.3.1, se crean instancias virtuales de los robots y se despliegan en un entorno simulado.

Este sistema de simulación esta compuesto por varios elementos, tal como se ilustra en la figura 2.11. Con las definiciones de los robots y el entorno realizadas con el formato de simulación se crea el escenario virtual, escenario conformado por los robots virtuales y el entorno en el que los robots interactúan. Cada robot virtual posee un controlador que implementa las funciones de las primitivas declaradas con el formato de descripción. Cada controlador interpreta los árboles de expresiones, generados por una implementación de programación genética, y envía comandos a los robots virtuales. Los robots ejecutan los comandos y se generan las interacciones de forma similar a como se espera que suceda en el entorno real.

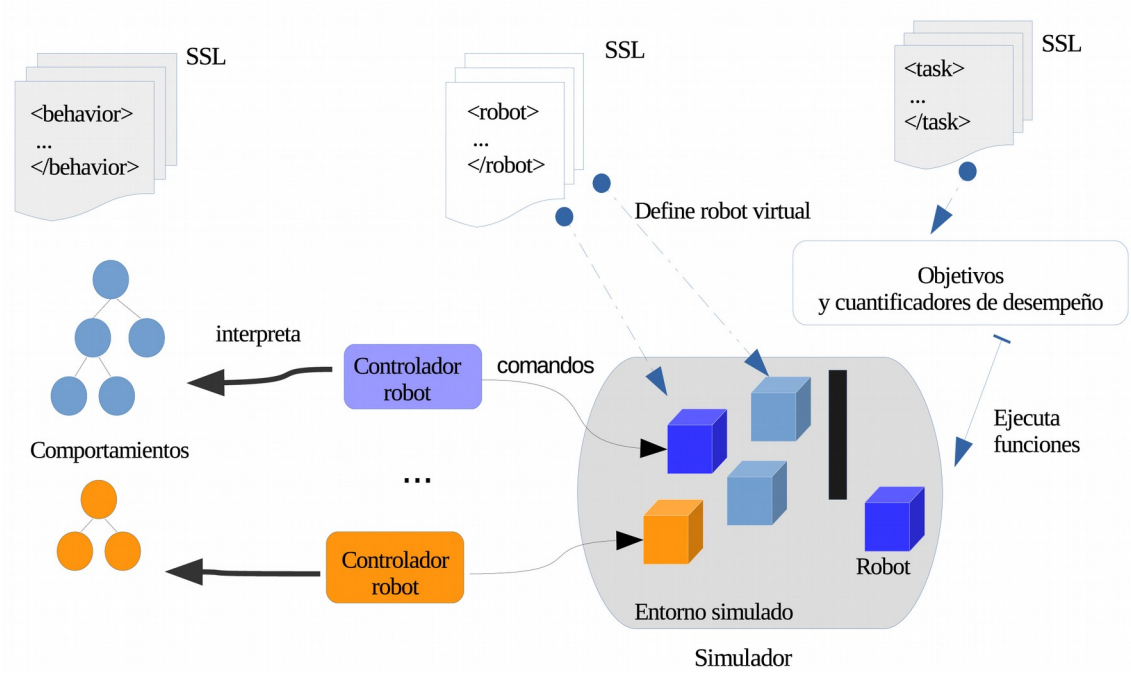


Figura 2.11. Sistema de simulación de la plataforma computacional.

El simulador contiene una implementación de las funciones que representan los objetivos de la tarea definida con el formato de descripción. Estas funciones son

llamadas periódicamente durante el proceso de inducción de los comportamientos y pueden acceder al estado de la simulación y generar información que influye en el estado de los robots. El proceso que cuantifica el desempeño del enjambre se implementa en una de las funciones u objetivos de la tarea.

Dentro de los simuladores disponibles para la investigación en el campo de la robótica se encuentra Gazebo [59]. Gazebo es un simulador flexible basado en física, que permite simular una amplia variedad de robots, sensores, efectores, escenarios y facilidad de adaptación para su integración con otros sistemas. Gazebo es utilizado como entorno de simulación en este trabajo y es extendido para soportar los controladores de los robots, las funciones que representan tareas, carga automática de las descripciones SSL, y los módulos de intercambio de información con la implementación de programación genética para cuantificar el desempeño de los comportamientos generados. En la figura 2.12 se presenta una captura de pantalla del simulador Gazebo.

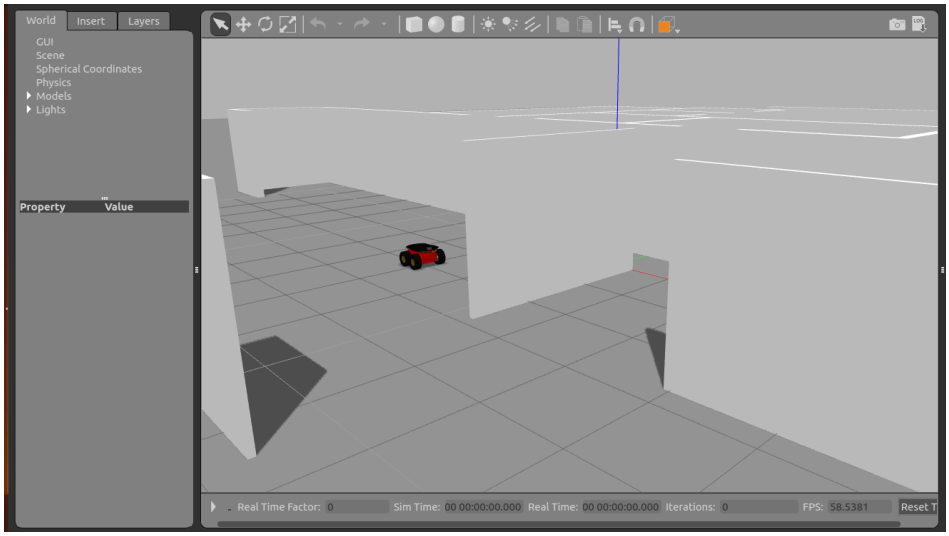


Figura 2.12. Simulador Gazebo utilizado en la plataforma computacional.

2.5. Implementación de la plataforma computacional

La plataforma computacional se implementa de la siguiente manera. Se desarrolla un conjunto de módulos acoplados mediante servicios web tipo REST, esto con el fin de facilitar el desarrollo e integración de nuevos módulos a la plataforma computacional. Se desarrollaran los siguientes módulos.

1. Módulo para la validación de sintaxis de los archivos de descripción en el formato SSL: el módulo valida las descripciones SSL y crea los artefactos para los módulos de simulación y programación genética.
2. Módulos contenedores: actúan como almacenes para los artefactos que utilizan los módulos de la plataforma. Los artefactos son los modelos de las tareas y las primitivas para cada robot.
3. Módulo de programación genética: implementación de programación genética encargada de sintetizar los comportamientos para los robots. Durante su operación toma los modelos de las primitivas para cada robot de los módulos contenedores, construye comportamientos y los envía al módulo de simulación para su ejecución y validación.
4. Módulo de simulación: sistema de simulación que contiene al simulador Gazebo, los controladores de los robots virtuales, la implementación de las funciones que representan la tarea y los cuantificadores de desempeño. Se encarga de recibir los comportamientos y ejecutarlos en cada robot virtual dentro del simulador, llamar a las funciones de la tarea y cuantificadores de desempeño.
5. Módulo de asignación: contiene la implementación de los mecanismos de comunicación con cada uno de los robots físicos. Gestiona todo el proceso de envío de los comportamientos a los robots.

Los módulos 1, 2, 3 y 5 son implementados en el lenguaje Java, se usa como backend para los servicios REST el framework WS2O Microservices Framework for Java [43], se utiliza el framework ECJ [41] como implementación de programación genética. El módulo 4 se implementa en el lenguaje C++ como un conjunto de plugins para el simulador Gazebo. Se definen los siguientes plugins para Gazebo:

- Un plugin para definir las funciones que representan a las tareas y funciones de aptitud. En este plugin se definen todas las funciones que son llamadas por la implementación durante el proceso de construcción de los comportamientos para los robots. Estas funciones se ejecutan según el modelo presentado en la sección 2.3.2.
- Un plugin para controlar el estado del simulador. Es un envolvente para el API de Gazebo que expone la funciones del simulador a través de servicios tipo REST. Estos servicios son utilizados por los módulos de programación genética e interfaz web que conforman la plataforma computacional.

Con el fin de proporcionar una interfaz para interactuar con la plataforma computacional se desarrolla una interfaz web con las siguientes funciones.

- Editor de Texto para el formato de descripción: permite crear y editar archivos de descripción de robots, tareas, primitivas y el enjambre en el formato SSL.

- Panel de herramientas: contiene un conjunto de controles para compilar las archivos de descripción (validación sintáctica y generación de artefactos), activar o desactivar el simulador, iniciar el proceso de construcción automático de comportamientos, activar o desactivar los administradores de comunicación para el intercambio de información con los robots reales, guardar o cargar un proyecto SSL. En la figura 2.13 se presenta la interfaz web de la plataforma computacional. La interfaz web es desarrollada con la tecnología Java Server Faces.

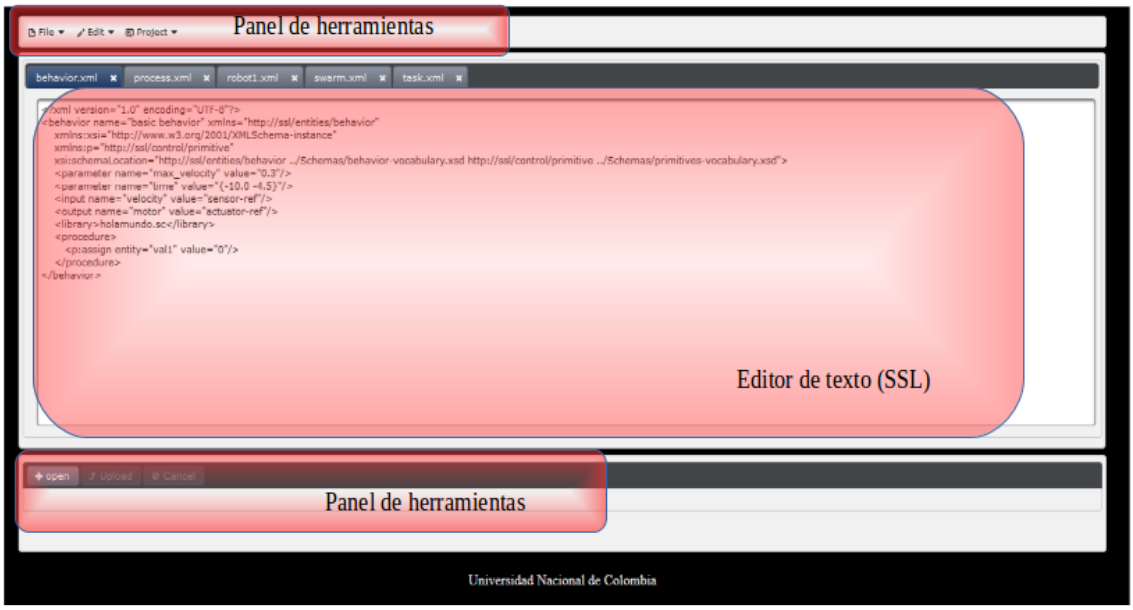


Figura 2.13. Interfaz web de la plataforma computacional.

CAPÍTULO 3

Construcción de comportamientos para robots con programación genética

3.1. Introducción

Como ya se ha mencionado en este trabajo, la robótica de enjambres evolutiva es una aproximación al diseño automático de comportamientos de auto-organización para conjuntos grandes de robots. Esta aproximación utiliza técnicas de evolución artificial para sintetizar las reglas de las interacciones robot-robot y robot-entorno que producen un patrón global deseado. Entre las técnicas de evolución artificial más utilizadas se encuentran los algoritmos genéticos y la programación genética [34], [35], [36], [37]. La programación genética es un tipo de algoritmo genético que evoluciona poblaciones de programas.

La programación genética ha sido utilizada en el diseño de programas (que representan comportamientos) para grupos de robots. Fútbol de robots [36], [37], [38], autoensamble de robots [40], planeación de trayectorias [39] y recientemente búsqueda de olores [62], han sido escenarios en los que se ha aplicado programación genética para la inducción de comportamientos colaborativos. En dichos trabajos, se explora el hecho de que un comportamiento es un programa, de forma y tamaño desconocidos, que típicamente relaciona percepciones del entorno con acciones del robot. El programa para un comportamiento es una composición jerárquica de operaciones que se realizan en los datos de entrada tales como el flujo de información proporcionado por un sensor, variables de estado y otros elementos, que tienen como resultado acciones que el robot ejecuta en el entorno.

Dada su capacidad natural para construir estructuras funcionales complejas, la programación genética (y como ha sido demostrado en los trabajos mencionados) es una opción viable para la construcción de programas para un enjambre heterogéneo de robots. En este trabajo se utiliza programación genética para la inducción de comportamientos de auto-organización para un conjunto homogéneo o heterogéneo de robots.

Este capítulo presenta la implementación de programación genética como algoritmo para la síntesis de programas de control para un conjunto de robots, se explora la representación de comportamientos en el espacio de búsqueda, así como los operadores

genéticos que se aplican sobre los individuos. De la misma forma, se examinan funciones de aptitud para tareas de agregación.

El capítulo está organizado de la siguiente forma: en la sección II se introduce la técnica de programación genética como método para el diseño automático de comportamientos, en la sección III se presenta la representación de los individuos en programación genética, en la sección IV se describen los operadores genéticos de cruce y mutación, mientras que las funciones de aptitud se presentan en la sección V.

3.2. Programación genética

La programación genética (PG) es un algoritmo evolutivo que genera programas de computador por medio de un proceso que simula la evolución biológica. Es una herramienta para la búsqueda automática de soluciones a problemas en los que no se conoce la forma o estructura de la solución, solución que es generada o aproximada durante un proceso cíclico en el que se aplican operadores genéticos a programas candidatos. La programación genética construye estructuras jerárquicas a partir de funciones primitivas y realiza una búsqueda sobre el espacio de todas las posibles composiciones de estas primitivas. Las primitivas dependen del tipo de problema abordado, en particular pueden incluir: los operadores aritméticos y lógicos, constantes, funciones matemáticas y, en general, cualquier función de dominio específico. Las variables y constantes se conocen como terminales, mientras que los demás elementos se denominan funciones, el conjunto de funciones y terminales se conoce como el conjunto primitivo. Una implementación típica de PG sigue la estructura del algoritmo 1.1, con algunas diferencias, tal como se indica en el algoritmo 3.1.

Programación genética

1: *Crear un conjunto de individuos, de forma aleatoria, a partir del conjunto de primitivas.*
2: **Hacer**
3: *Ejecutar cada individuo y evaluar su desempeño.*
4: *Seleccionar los individuos para aplicar los operadores genéticos.*
5: *Aplicar operadores genéticos y generar nuevos individuos.*
6: **Hasta**: *encontrar una solución aceptable o cumplir otro criterio.*
7: *Devolver mejor individuo.*

Algoritmo 3.1. Algoritmo general de Programación Genética.

La PG mantiene una población de individuos creada inicialmente usando uno de varios mecanismos que incorporan el concepto de profundidad de nodo. La profundidad de un nodo es el número de aristas que hay entre el nodo y el nodo raíz del árbol al cual pertenece el nodo. La profundidad de un árbol es la profundidad de su nodo más extremo. Los mecanismos más utilizados son: *grow*, *full* y *ramped half-and-half*.

Los métodos de generación *grow* y *full* crean árboles con una profundidad especificada. El método *full* genera árboles seleccionando de forma aleatoria nodos tipo función del conjunto primitivo hasta la profundidad especificada menos uno, a partir de este punto sólo se escogen terminales hasta alcanzar la profundidad máxima, este método genera árboles completos. Por su parte, el método *grow* construye árboles seleccionando funciones y terminales hasta alcanzar la profundidad especificada, de esta forma, se generan árboles de diversas formas y tamaños. El método *ramped half-and-half* es una combinación de los métodos *grow* y *full*; una parte de la población es generada utilizando *grow* y la otra parte *full,* ambas, con profundidades en un rango de valores. Este último método es utilizado en este trabajo con el fin de generar una población de árboles con una amplia variedad de formas y tamaños.

PG es un proceso iterativo, hasta que se cumple un criterio establecido, modifica los individuos de la población. PG aplica un mecanismo de selección para escoger individuos de una generación que producen nuevos individuos de la población de la siguiente generación. En general, el método de selección escoge un individuo con una probabilidad dependiente de la aptitud del individuo. De forma que, los mejores individuos tienen mayor probabilidad de pasar o aportar a soluciones de la siguiente generación. Los métodos de selección más utilizados son: selección por torneo, selección proporcional y muestreo universal estocástico.

En el método de selección por torneo, se escoge de forma aleatoria un subconjunto de la población, del cual se toma al mejor de acuerdo a la función de aptitud, individuo al cual se aplican los operadores genéticos. En la implementación de PG realizada en este trabajo, se utiliza el método de selección por torneo para la selección de los individuos para las operaciones de cruce y mutación, este método mantiene constante la presión de selección en la población previniendo la pérdida prematura de diversidad.

Los individuos escogidos en el proceso de selección son modificados por medio de los operadores genéticos. Los operadores genéticos más importantes en PG son el cruce y la mutación. El cruce crea un nuevo individuo combinando partes tomadas de forma aleatoria de dos individuos seleccionados. Por su parte, la mutación genera un nuevo individuo alterando de forma aleatoria una parte de un individuo seleccionado. Los nuevos individuos son asignados a la nueva población. Este proceso (llamado iteración) continúa hasta cumplirse un criterio de parada. Los criterios de parada típicos en PG son: encontrar un individuo aceptable o número determinado de iteraciones. En este trabajo se utiliza una combinación de los dos criterios, si se encuentra un individuo con un valor de la medida de aptitud preestablecido se termina el proceso evolutivo de lo

contrario, continúa el proceso hasta alcanzar el número de iteraciones establecido. El criterio utilizado en este trabajo para determinar que un individuo es aceptable es el valor de aptitud estandarizado (*e*), el valor utilizado en la selección de individuos es el valor ajustado propuesto por Koza [65], que se indica en la ecuación 3.1.

$$f_{ajustado} = \frac{1}{1+e} \qquad (3.1)$$

$$e \in [0, \infty); 0 \rightarrow valor\ de\ aptitud\ ideal$$

La implementación de programación genética realizada en este trabajo se presenta en el algoritmo 3.2.

Algoritmo de programación genética de la plataforma computacional

```
1:   funcion PG( max_profundidad,generaciones,n_individuos )
2:       P = poblacionInicial( max_profundidad )
3:       p_cruce = 0.1
4:       p_mutacion = 0.9
5:       mientras (generacion_actual < generaciones) hacer
6:           evaluar_aptitud( P )
7:           P_cruce = por_cruce( P, torneo, p_cruce, n_individuos )
8:           P_mutacion = por_mutacion( P, torneo, p_mutacion, n_individuos )
9:           P = P_crue U P_mutacion
10:      fin mientras
11:      Devolver mejor individuo
12:  fin funcion
```

Algoritmo 3.2. Algoritmo de programación genética implementado en la plataforma computacional.

El primer paso del algoritmo es generar la población inicial, utilizando el método *ramped half-and-half* se crean *n_individuos*. De forma iterativa, hasta alcanzar el número de generaciones especificado, se aplican los pasos seis (6) al nueve (9) de la siguiente forma. En el paso seis (6) se toma cada individuo y se asignan los programas a cada uno de los robots, se ejecutan los programas en el simulador y se calcula el valor

de aptitud para cada programa. Al final de este paso a cada individuo es asignado un valor de aptitud que depende del desempeño del enjambre ejecutando la tarea asignada.

La población de la siguiente generación es creada aplicando cruce y mutación de forma excluyente, es decir, un individuo es creado y asignado a la siguiente generación sólo utilizando uno de estos dos operadores. La siguiente generación es producida en un 10% aplicando el operador de cruce mientras que el 90% restante es generado aplicando mutación. En el paso siete (7) se generan los individuos, de la porción de la siguiente generación, por medio del operador de cruce (ver sección 3.4 Operadores genéticos), este operador genera dos nuevos individuos. En el paso ocho (8) se genera la porción restante de la nueva generación con el operador de mutación. En ambos casos la selección de los individuos que participan en las operaciones genéticas es realizada por medio del método de selección por torneo con un número indicado (parametrizado) de individuos.

En el paso nueve (9) se realiza la unión de los conjuntos de individuos generados por cruce y los generados por mutación, el nuevo conjunto constituye la población de la siguiente iteración. Finalmente, cuando se ha alcanzado el número de generaciones especificado el mejor individuo es retornado.

A continuación se describe cómo se representan los individuos y cómo se aplican los operadores de cruce y mutación, en la implementación de programación genética de la plataforma computacional.

3.3. Representación de los comportamientos primitivos

Los árboles de expresión definen uno de los mecanismo de representación más utilizado en programación genética. En implementaciones típicas se utiliza un árbol por individuo con un único conjunto primitivo para todos los individuos. Sin embargo, cundo se trabaja con grupos heterogéneos de robots surge la necesidad de representar robots que poseen propiedades y capacidades diversas (móviles, manipuladores seriales y/o paralelos, etc). Esta diversidad dificulta la existencia de un único conjunto primitivo a partir del cual se construyen los árboles. Para abordar estas dificultades, en este trabajo, se adopta la siguiente aproximación para un conjunto heterogéneo de robots.

Para cada uno de los tipos de robots se establece un conjunto primitivo independiente, este conjunto primitivo es construido a partir de la descripción de primitivas para cada robot realizadas en el formato de descripción SSL. El comportamiento de un robot individual es representado por medio de un árbol de expresión asociado a su conjunto primitivo. Cada árbol está limitado a tener una profundidad máxima especificada y se mantiene tras la aplicación de los operadores genéticos.

Un individuo de la población posee un conjunto de árboles independientes (uno por cada tipo de robot), cada individuo representa a un enjambre completo. En la figura 3.1 se ilustra cómo se representa un individuo en la implementación de programación genética de la plataforma computacional (caso conjunto heterogéneo).

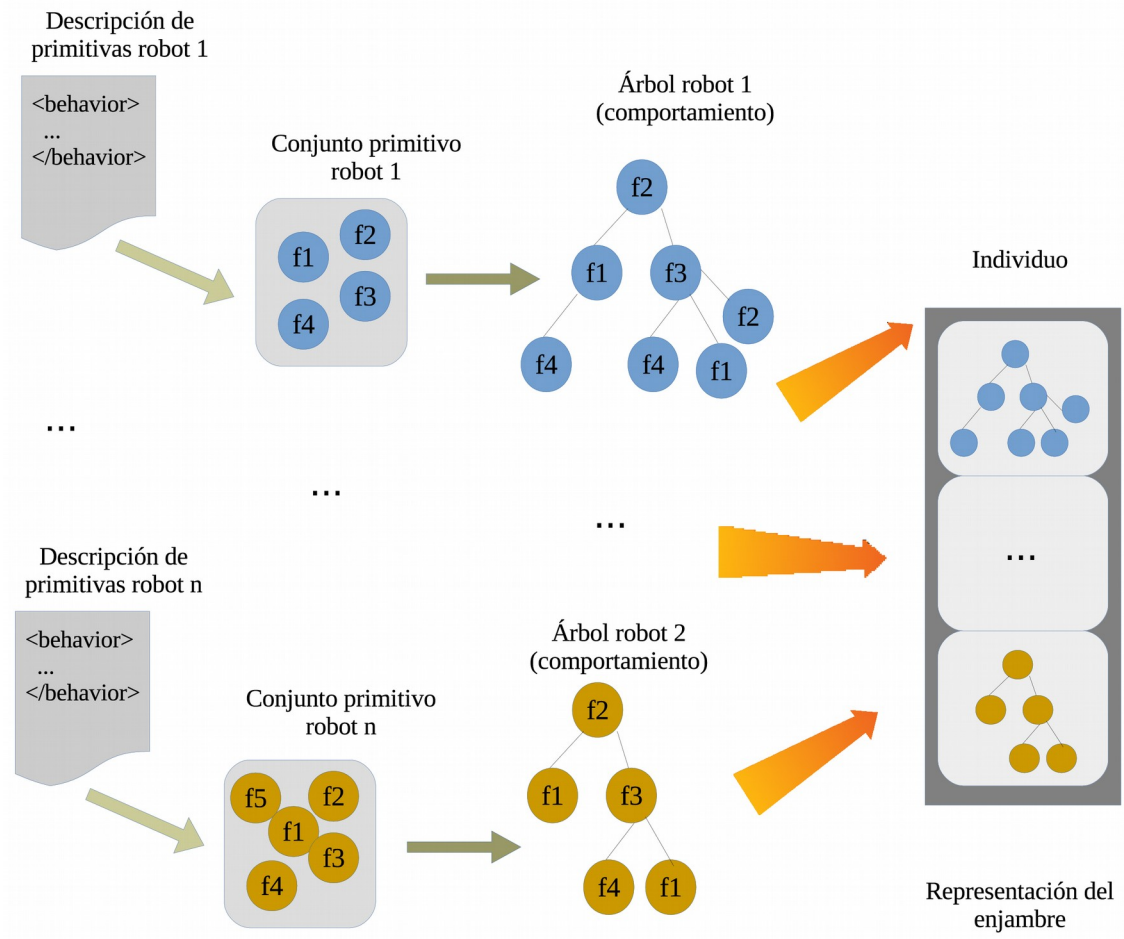


Figura 3.1. Estructura de un individuo.

Un individuo puede ser visto como un árbol cuyas ramas son los árboles del comportamiento para cada robot. De esta forma, se codifica todo el enjambre en un individuo y se evoluciona una población de enjambres donde cada miembro evoluciona su propio programa. Una ventaja adicional derivada del uso de esta representación es que se evalúa el desempeño del enjambre completo donde el resultado emergente está determinado por las interacciones de los robots.

La representación expuesta no es exclusiva para grupos heterogéneos. Para el caso de un conjunto homogéneo de robots, es posible usar la aproximación tradicional (utilizar un único árbol por individuo). Cada individuo de la población contiene un único árbol, creado a partir del mismo conjunto primitivo (todos los robots tienen las mismas

primitivas), que es asignado a todos los robots durante el paso de evaluación del desempeño del individuo.

3.4. Operadores genéticos

En este trabajo se implementan los dos operadores típicos de una aplicación de programación genética: cruce y mutación. Se definen los operadores genéticos de la siguiente forma:

3.4.1. Operador de cruce

El operador genético de cruce o recombinación construye un nuevo individuo combinando partes de dos individuos seleccionados. La operación se aplica para cada uno de los árboles que componen al individuo. En la figura 3.2 se ilustra la operación de cruce entre dos individuos. Se selecciona cada par de árboles en la misma posición de los individuos (que corresponden al mismo robot) posteriormente, se selecciona de forma aleatoria un nodo (punto de cruce) del árbol del primer individuo, el árbol cuya raíz es dicho nodo es reemplazado por un árbol cuya raíz es un nodo seleccionado del árbol del segundo individuo. Esta operación típicamente produce un sólo individuo, sin embargo, la implementación soporta la generación de dos individuos en cuyo caso las porciones retiradas de un árbol del primer individuo son reemplazadas por las partes tomadas del árbol del segundo individuo. Como lo propone Koza [65], el tipo de nodo (función o terminal) seleccionado como punto de cruce es escogido aplicando una probabilidad de selección: 90% para nodos tipo función y 10% para nodos tipo terminal, esto ayuda a prevenir el intercambio de sólo pequeñas porciones de los árboles, propio del método de selección con probabilidad uniforme.

Cuando el conjunto de robots es homogéneo, el operador de cruce se aplica de forma tradicional (al único árbol que posee el individuo). Se seleccionan de forma aleatoria los nodos de dos árboles individuos y se intercambian los árboles con las raíces seleccionadas. La probabilidad de selección para el tipo de nodo es la misma que para el caso del grupo heterogéneo de robots: 90% para función y 10% para terminal.

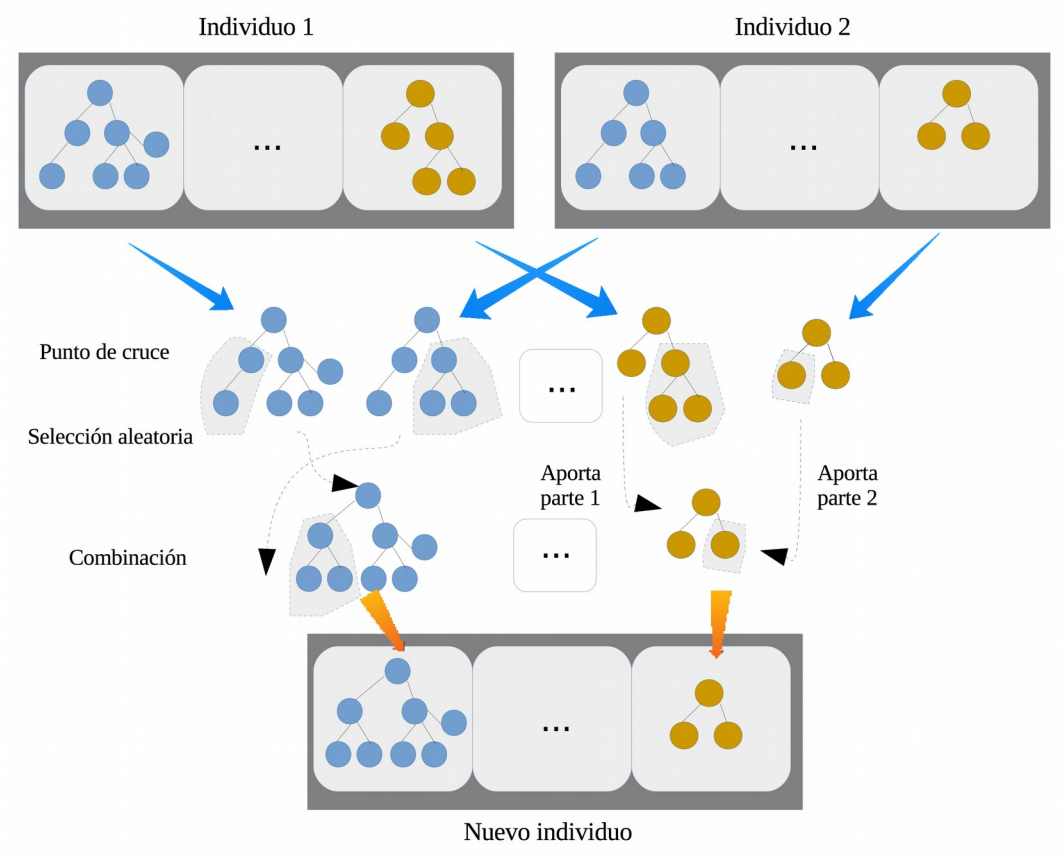


Figura 3.2. Operación de cruce.

3.4.2 Operador de Mutación

La operación de mutación produce un nuevo individuo aplicando, posiblemente, leves alteraciones a un individuo seleccionado. En este trabajo, se implementa el operador de cruce y se aplica a un individuo que representa un enjambre de la siguiente manera. Para cada uno de los árboles que constituyen al individuo, se selecciona de forma aleatoria un nodo (punto de mutación), el árbol cuya raíz es el nodo seleccionado es reemplazado por un nuevo árbol generado a partir del conjunto primitivo correspondiente, el árbol generado es calculado de forma que el árbol del individuo no exceda la profundidad establecida. La operación de cruce se ilustra en la figura 3.3.

Para el caso del conjunto homogéneo de robots, el operador de cruce se aplica (al único árbol que posee el individuo) de la misma forma en que es aplicado a uno de los árboles de la representación anteriormente descrita.

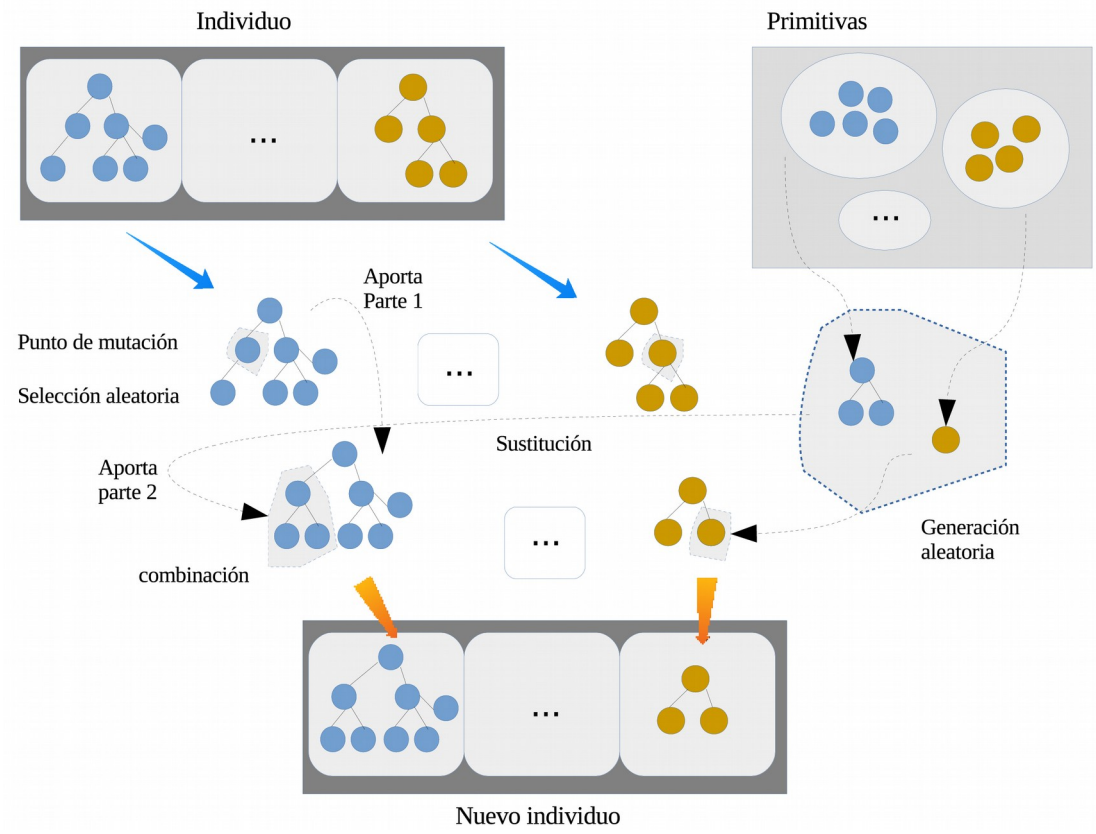


Figura 3.3. Operación de mutación.

3.5. Funciones de aptitud

La función de aptitud es el mecanismo utilizado en programación genética para especificar el/los requerimiento/s de alto nivel del problema que se desea resolver. En aplicaciones de programación genética al área de robótica de enjambres las funciones dependen del dominio de aplicación del enjambre de robots. En muchas investigaciones se han utilizado enjambres de robots para resolver tareas encontradas en los enjambres naturales, si bien no es una norma, estas tareas sirven como mecanismo de validación y comparación de resultados. Entre las tareas más abordadas se encuentran: agregación, segregación, forrajeo colectivo, transporte colectivo y movimiento coordinado. Cada una de estas tareas requiere de una función de aptitud y en la mayoría de los casos su forma es definida por el investigador basándose en su experiencia. Por ejemplo, dos funciones utilizadas para tareas de agregación se presentan en las ecuaciones 3.2 y 3.3.

$$f_{agg} = \left(\sum_{t=i}^{tmax} \sum_{i,j} d_{ij}(t)\right)^{-1} \qquad (3.2)$$

$$f = \frac{max(a, b)}{N} \qquad (3.3)$$

La ecuación 3.2, definida por Nishikawa et al [42], expresa la medida de aptitud del enjambre como la suma en el tiempo de la distancia entre los robots *i* y *j*, el objetivo es maximizar *f*. Por su parte, la ecuación 3.3, definida por Francesca et al [9], mide el desempeño realizando un conteo del número de robots dentro de las áreas de agregación respecto al número total de robots. De igual forma, se han utilizado funciones de aptitud para tareas de forrajeo. El forrajeo en robótica móvil es una abstracción del proceso de recolección de alimentos observado en enjambres naturales, los robots deben agrupar objetos, inicialmente dispersos en el entorno (fuente), en un área de acopio. En este caso, se ha utilizado como medida de aptitud el número de objetos recolectados o (en representaciones más abstractas de la tarea) el número de veces que los robots pasan por un lugar A (fuente) y luego por otro lugar B (acopio) dentro del entorno.

En capítulos anteriores se definió una tarea (en el ámbito de la plataforma computacional) en términos de funciones que son llamadas por la implementación durante la fase de evaluación del desempeño de un individuo. Con esta aproximación, pueden representarse los tipos de tareas típicas abordadas en las investigaciones de la robótica de enjambres y que pueden ser expresadas en términos de funciones de optimización (como las presentadas en las ecuaciones 3.2 y 3.3).

En la plataforma computacional, las funciones de aptitud son declaradas utilizando el formato SSL (utilizando las marcas de descripción de tareas). Adicionalmente, estas funciones deben estar implementadas en el simulador para que puedan ser llamadas durante la fase de evaluación del desempeño.

CAPÍTULO 4

Aproximación al diseño de comportamientos para robots móviles

4.1. Introducción

Con el fin de validar la plataforma computacional como una herramienta para la generación automática de comportamientos de auto-organización para un enjambre de robots en el contexto de la robótica de enjambres evolutiva, se generan comportamientos de auto-organización para un conjunto de robots simulados que realizan tareas de agregación. Los robots virtuales utilizados en los experimentos no son una representación fiel de sus contrapartes físicas, estos sólo poseen los módulos que les permiten realizar sus comportamientos primitivos.

Poli et al [44] formularon cuatro pasos preparatorios para aplicar programación genética para resolver un problema determinado. El primer paso es establecer el conjunto primitivo, en este paso se definen las funciones y terminales que constituyen los elementos primarios para la construcción de las posibles soluciones. En el segundo paso se define la función de aptitud que representa el problema que se quiere resolver. En el tercer paso se definen los valores de los parámetros de ejecución de programación genética, que incluye especificar, entre otros, el tamaño de la población, las probabilidades de aplicación de las operaciones genéticas y el tamaño máximo de los programas a generar. El paso final consiste en definir el criterio de terminación del ciclo evolutivo. A continuación se definen los escenarios experimentales y se aplican los pasos descritos anteriormente para una de los problemas más abordados en la robótica de enjambres: la agregación.

4.2. Descripción del escenario experimental

4.2.1. Descripción de la tarea

En su forma más simple, el problema (tarea) de agregación se define de la siguiente manera: para un conjunto de robots que se encuentran inicialmente dispersos en un entorno y una región marcada en el entorno como destino, el objetivo es que los robots se agrupen dentro de dicha región. La agregación es un comportamiento de enjambres observado en sociedades de insectos y grupos de bacterias. Este tarea es abordada ampliamente en la investigación de enjambres de robots ya que es considerada como uno de los comportamientos básicos o bloque constructivo para comportamientos más complejos permitiendo que los robots estén lo suficientemente cerca para interactuar [2].

Las funciones utilizadas para la tarea de agregación tienen formas diversas (ver sección 3.5). En este trabajo se representa la tarea de agregación con una función que calcula la distancia de los robots al centro de la región de agregación, según se indica en la ecuación 4.1.

$$f_{agg} = \sum_{i=1}^{N} \sqrt{(x_{agg} - x_i)^2 + (y_{agg} - y_i)^2} \qquad (4.1)$$

x_{agg} = *coordenada x del centro del área de agregación*

y_{agg} = *coordenada y del centro del área de agregación*

x_i = *coordenada x del robot i*

y_i = *coordenada y del robot i*

N = *número total de robots*

En la figura 4.1 se presenta la descripción de la tarea de agregación en el formato de descripción SSL. La tarea es declarada en el formato de descripción SSL según las reglas presentadas en el capítulo 2 de este trabajo. Adicionalmente, la tarea es definida en el módulo de gestión del simulador en el lenguaje C++. La definición SSL le indica a la implementación qué función llamar y las restricciones impuestas (ver capítulo 2) para su ejecución, en este caso se ha impuesto una restricción de tiempo, por lo que la función que representa la tarea de agregación es ejecutada transcurridos 60 segundos después de la asignación de los comportamientos construidos por el sistema a los robots virtuales en el simulador. La función recibe como argumentos las coordenadas del centro del área de agregación, con los cuales calcula el valor de desempeño de acuerdo

con la ecuación 4.1. Este proceso es ejecutado en cada ciclo evolutivo durante la construcción de los comportamientos para la tarea de agregación. Nótese que la tarea de agregación se ha definido en términos de una función que mide el desempeño (función de aptitud) de los comportamientos construidos por el sistema para agrupar a los robots en el área especificada.

Declaración SSL

```
<?xml version="1.0" encoding="UTF-8"?>
<task xmlns="http://ssl/entities/task"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="aggregation">

  <targets name="goals">
    <goal name="goal0">
      <arg name="x" value="-8.01723" type="double"/>
      <arg name="y" value="-3.64815" type="double"/>
      <arg name="rad" value="1.0" type="double"/>
    </goal>
  </targets>
  <restrictions>
    <r:timeWindow>
      <r:time for="goal0" value="60" units="s"/>
    </r:timeWindow>
  </restrictions>
  <description />

</task>
```

Definición en el simulador (c++)

```
void fitness0( std::vector<std::string>* args ){
    // Extrae las coordenadas de la región de agregación
    double gx = std::stod( args->at( 0 ) );
    double gy = std::stod( args->at( 1 ) );
    double grd = std::stod( args->at( 2 ) );
    // Calcula y suma las distancias de cada robot al centro de la región de agregación
    math::Pose ps;
    double sum = 0;
    for( int i = 0;i<robotn.size();i++ ){
      ps = _world->GetModel( robotn.at( i ) )->GetWorldPose();
      sum += euclid( gx,gy,ps.pos.x,ps.pos.y );
    }
    // Publica el valor de desempeño
    fitness_pub→Publish( sum );
}
```

Figura 4.1. Declaración (en SSL) y definición (en el simulador) de la tarea de agregación.

La tarea de agregación puede ser representada con una función más simple, tal como realizar el conteo de los robots dentro del área de agregación al final de la restricción de tiempo. Sin embargo, una función de este estilo restringe la construcción progresiva de los comportamientos, ya que, en general, se asignarán valores aceptables de desempeño

sólo a los individuos que, dentro del tiempo de ejecución establecido, agrupan robots dentro del área de agregación, mientras que al resto de individuos se asignará el mismo valor (no aceptable) de desempeño. Por su parte, la función utilizada aquí permite asignar diferentes valores de desempeño a los individuos dependiendo de la cercanía al área de agregación, esto permite que programas que acercan robots al área de agregación (no necesariamente agrupándolos dentro del área de agregación) no sean repentinamente desechados (al asignar valores no aceptables de desempeño). Estos comportamientos podrían requerir sólo pequeñas modificaciones para resolver la tarea, las cuales podrían realizarse en la siguiente iteración.

4.2.2. Descripción del entorno

El escenario utilizado para desplegar los robots para la tarea de agregación es una representación virtual de un área cuadrada de ocho metros de longitud, sin obstáculos (excepto los límites). Se asigna al escenario un sistema coordenado en la parte superior derecha para referencia. Respecto a este sistema de referencia se define un área de agregación en la posición (-8, 3.6) y de radio un metro tal como se ilustra en la figura 4.2.

La descripción del escenario se realiza utilizando el formato de descripción SDF y se integra como un modelo en la base de datos del simulador Gazebo [59]. En la figura 4.3 se muestra una porción de la descripción del escenario en el formato SDF. La descripción contiene todos los elementos del escenario (límites, piso, iluminación), entre los cuales se encuentran el área de agregación y su posición dentro del escenario. De forma similar, se debe especificar el módulo encargado de gestionar la escena (comunicaciones) y la definición de la tarea como se presenta en la figura 4.1.

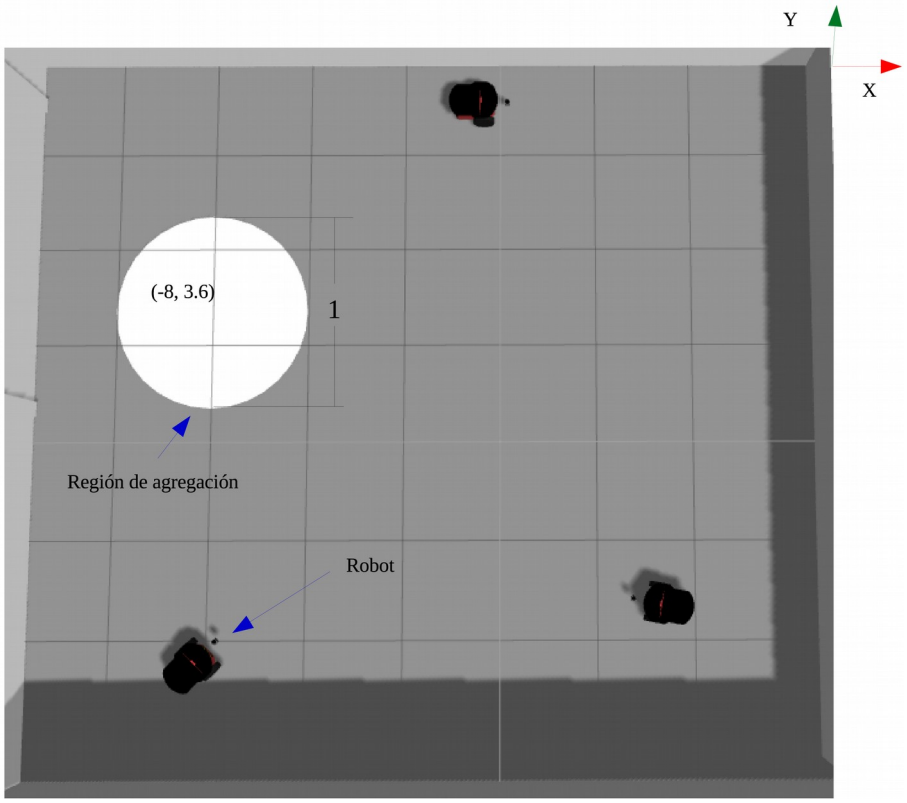


Figura 4.2. Escenario experimental para la tarea de agregación.

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="agg_world">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://lab_un</uri>
    </include>
    <model name='area1_space'>
    <static>true</static>
      <pose>-8.01723 -3.64815 -0.05 0 -0 0</pose>

        ...

    <plugin name="eval" filename="libPCEEvaluatorsImplLib.so"/>
  </world>
</sdf>
```

Nombre del escenario

Elementos del escenario

Área de agregación

Posición del área agregación

Plugin con la implementación de la tarea y otros módulos

Figura 4.3. Definición parcial del escenario para la tarea de agregación en el formato SDF.

4.2.3. Descripción de los robots

Los robots para los que se construyen los comportamientos para resolver la tarea de agregación son instancias del robot pioneer2dx de Omron Adept MobileRobots LLC [66] (disponible en el simulador Gazebo). Este robot es un robot móvil diferencial con tracción en dos ruedas y una rueda de apoyo, al cual se ha agregado un sensor láser que le permite realizar mediciones de la geometría del entorno y detección de obstáculos. El modelo disponible en la base datos de Gazebo es una versión simplificada que sólo contiene la cinemática del robot. Para este modelo se implementa una controlador que recibe y ejecuta un árbol generado por la implementación del programación genética de la plataforma computacional. En la tabla 4.1 se presentan algunos de los elementos que posee el modelo virtual del robot pioneer2dx, así como los elementos agregados para complementar las funciones del robot para ejecutar los comportamientos construidos. Al modelo geométrico del robot se agrega un sensor láser 2D para detectar obstáculos, este sensor mide la distancia a la que se encuentran los objetos en un área definida en frente del robot. Se implementa y agrega el controlador de movimiento del robot virtual, este módulo recibe e interpreta el árbol que representa el comportamiento construido automáticamente. Este controlador se implementa en ROS [67]. El robot tiene las dimensiones presentadas y puede moverse a velocidad lineal de *1.2m/s*. En la figura 4.4 se presenta el robot pioneer2dx junto a su modelo virtual disponible en el simulador Gazebo.

Sensores	**Variables**	**Funciones**
Láser Hokuyo	Distancia	Detección de obstáculos
		Procesamiento / Software
Módulo controlador	ROS	
		Propiedades mecánicas
Dimensiones	H237xW455xD381 (mm)	
Velocidad máxima	1.2 m/s	
Velocidad de rotación	300 °/s	
Capacidad de carga	17 Kg	

Tabla 4.1. Elementos del modelo virtual del robot pioneer2dx.

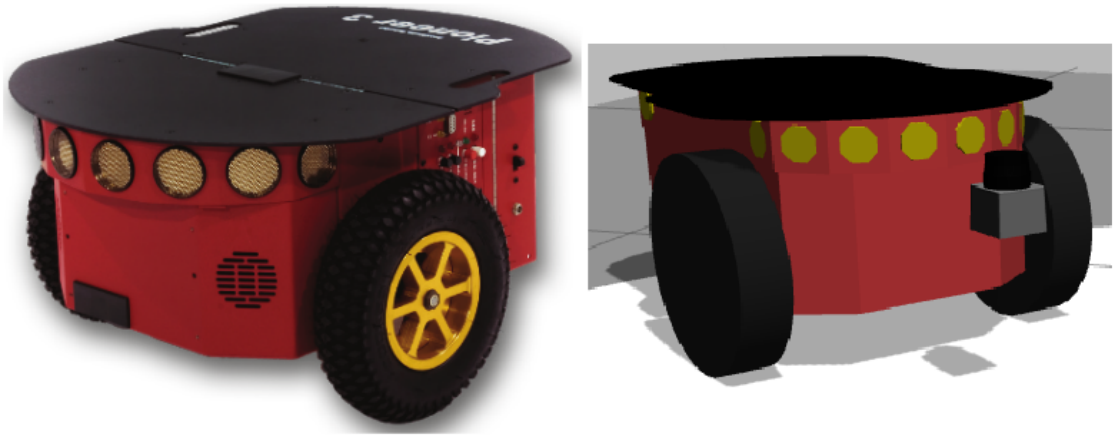


Figura 4.4. Robot pioneer 2DX.

En la figura 4.5 se muestra la descripción parcial del robot móvil pioneer2dx, de la figura 4.4, en el formato de descripción SSL. La descripción está compuesta por dos declaraciones: la declaración de las propiedades físicas del robot y sensores, descritas en el formato SDF (parte superior de la figura 4.5), y la descripción de los canales de comunicaciones en el formato SSL (parte inferior de la figura 4.5). La descripción en SSL tiene una referencia a la descripción SDF y es el punto de entrada para la plataforma computacional. Las dos descripciones conforman la definición completa de un robot en la plataforma computacional.

4.2.4. Descripción del conjunto primitivo

La definición del conjunto de primitivas necesario para construir las soluciones a la tarea (suficiencia del conjunto primitivo) abordada, en general, es un problema aún abierto [44]. La aproximación adoptada en este trabajo para construir el conjunto primitivo suficiente es netamente experimental. La suficiencia del conjunto es comprobada por un experto humano quien puede construir, de forma manual, un comportamiento de agregación a partir del conjunto definido (ver sección 4.3).

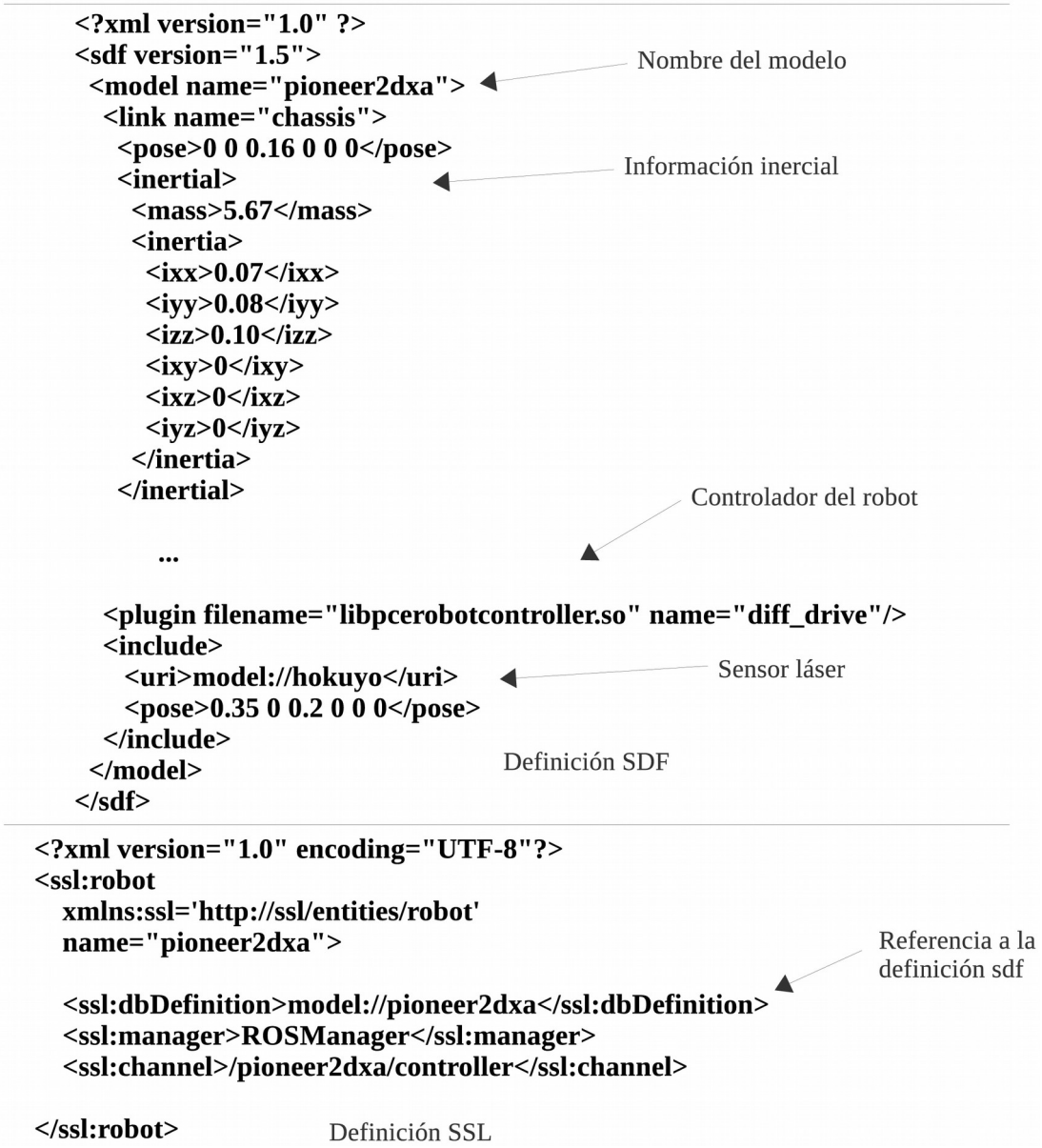


Figura 4.5. Descripción parcial del robot pioneer2dx en el formato SSL.

Se definen las primitivas que se presentan en la tabla 4.2 que constituyen un conjunto suficiente para resolver la tarea da agregación. Todas las funciones tienen un tipo de parámetro y tipo de valor de retorno de forma que, cuando la implementación de programación genética construye los árboles, sólo es posible conectar nodos del mismo tipo. Es decir, una función con un parámetro (contenedor de nodo) de tipo *none* sólo puede albergar una función (nodo) con un tipo de retorno *none*, y de la misma forma para todos los tipos.

Primitiva	**Tipo**	**Aridad**	**Parámetro/tipo**	**Tipo de retorno**
				Movimiento
forward	Función	1	Velocidad de avance/real	none
rotate_to_goal	Terminal	0		none
rotate	Función	1	Incremento en la orientación actual/real	none
stop	Terminal	0		none
				Percepción del entorno
in_agg_region	Terminal	0		Valor de verdad
obstacle	Terminal	0		Valor de verdad
				Estructuras de control
sequence	Función	2	Función sentencia 1/none Función sentencia 2/none	none
if	Función	3	Sentencia condicional/valor de verdad Sentencias bifurcación 1/none Sentencias bifurcación 2/none	none
				Operadores lógicos
not	Función	1	FA/Valor de verdad	Valor de verdad
or	Función	2	FA, FB/valor de verdad, valor de verdad	Valor de verdad
and	Función	2	FA, FB/valor de verdad, valor de verdad	Valor de verdad
				Constante temporales
lineal_speed	Terminal			real
rot_increment	Terminal			real

Tabla 4.2. Conjunto primitivo para la tarea de agregación.

Se definen cuatro primitivas de movimiento: avanzar en línea recta a velocidad específica (*forward*) con detección de obstáculos, detener el movimiento del robot (*stop*), rotar en dirección a la región de agregación, rotar en una dirección específica. El valor de la velocidad de avance para la función *forward* y el valor del ángulo de rotación para la función *rotate* son ajustados a partir de un rango de velocidades establecido. Se definen dos funciones para percibir el entorno: una función que detecta objetos a una distancia de veinte centímetros frente al robot (*obstacle*), y otra que detecta si el robot se encuentra dentro de la región de agregación (*in_agg_region*).

El conjunto primitivo contiene funciones que actúan como estructuras de control. La estructura de control si-entonces (*if*) es una función de tres parámetros, su primer parámetro es la condición, el segundo parámetro es la función a ejecutar si la condición es verdadera y el tercer parámetro es la función a ejecutar si la condición es falsa. La condición es una función que debe retornar un valor de verdad. Las funciones que van en las bifurcaciones son del tipo *none* que es un tipo especial de retorno que permite construir estructuras de funciones que ejecutan acciones pero no retornan un valor. La sentencia de control secuencia (*sequence*) permite agrupar dos funciones que son ejecutadas una después de la otra, es una función de dos parámetros cada uno de los cuales recibe una función con valor de retorno *none*, la función en el segundo parámetro es ejecutada justo después de la función en el primer parámetro.

El conjunto primitivo contiene los operadores lógicos *and*, *or* y *not*. Estos operadores son funciones con parámetros que tienen apuntadores a funciones que retornan valores de verdad, el valor de retorno de cada operador es un valor de verdad. El operador *and* implementa el operador lógico conjunción, tiene dos parámetros cuyos tipos son valores de verdad. De igual forma, el operador *or* tiene dos parámetros cuyos tipos son valores de verdad e implementa el operador lógico de disyunción. El operador *not* tiene un parámetro cuyo tipo es un valor de verdad e implementa el operador lógico de negación.

Finalmente, se agregan dos constantes temporales que representan los valores de velocidad lineal y velocidad de rotación para las funciones *forward* y *rotate*. El valor de estos terminales es generado cuando se construye el árbol o se aplica un operador genético que afecta a dicha terminal y se mantienen contantes durante la ejecución del árbol en el simulador. De esta forma, la implementación de programación genética ajusta las velocidades de movimiento de los robots.

Cuando el conjunto de robots es homogéneo el conjunto primitivo presentado en esta sección es único y es compartido por todos los robots, en este caso la representación presentada en la sección 3.3 contiene un sólo árbol, cada individuo de programación genética posee un único árbol que, al ser evaluado su desempeño en el simulador, es asignado a todos los robots virtuales. Por su parte, cuando el conjunto es heterogéneo, un individuo de PG posee un árbol por cada tipo de robot que es asignado a las instancias respectivas en el simulador.

En este experimento se construyen comportamientos de agregación para un conjunto homogéneo de robots y para un conjunto heterogéneo aproximado. Con conjunto heterogéneo aproximado se hace referencia a que, en realidad, el conjunto no posee robots con diferentes habilidades, sino que se realiza una adaptación del conjunto homogéneo y se evolucionan soluciones tratándolo como un conjunto heterogéneo. Cuando se trabaja con un conjunto heterogéneo, cada tipo de robot posee su propio conjunto primitivo dado que cada tipo posee habilidades diferentes. La adaptación consiste en mantener un único tipo de robots en el grupo (robots para el caso

homogéneo) y utilizar el mismo conjunto primitivo para crear los diferentes árboles del individuo de PG. Para cada uno de los tres robots se crea un árbol a partir del mismo conjunto primitivo, si bien, los árboles comparten el conjunto primitivo, estos son evolucionados de forma independiente (agrupados en un mismo individuo de PG) generando estructuras diferentes que, como se presenta más adelante, producen el mismo comportamiento de agregación.

```
<?xml version="1.0" encoding="UTF-8"?>
<behavior xmlns="http://ssl/entities/behavior"
   xmlns:p="http://ssl/control/primitive"
   name="conjunto_primitivo"
   type="simple">

   <function>
     <name>forward</name>
     <return>none</return>
     <param name="speedf" type="double" value="[3.0 4.5]"/>
   </function>
   <function>
     <name>rotate</name>
     <return>none</return>
     <param name="speedr" type="double" value="0.003"/>
   </function>
  <function>
     <name>obstacle</name>
     <return>boolean</return>
  </function>
  <function>
    <name>not</name>
    <return>boolean</return>
     <param name="ntpred" type="boolean" value="NAT"/>
  </function>
  ...
  <function>
   <name>if</name>
   <return>none</return>
   <param name="vvf" type="boolean" value="NAT"/>
   <param name="body1" type="none" value="0"/>
   <param name="body2" type="none" value="0"/>
 </function>
</behavior>
```

Figura 4.6. Conjunto primitivo para los robots virtuales definido en el formato SSL.

4.3. Evolución de los comportamientos

Como se presenta en la parte introductoria de este capítulo, los pasos preparatorios tres (3) y cuatro (4) para aplicar programación genética consisten en definir los parámetros de ejecución y el criterio de terminación. Los parámetros, para la tarea de agregación, se ilustran en la tabla 4.3.

Se define una población de diez (10) individuos. Los individuos de la población inicial son construidos, seleccionando aleatoriamente elementos del conjunto primitivo, utilizando el método ramped half-and-half con una profundidad máxima de diecisiete (17). Durante el ciclo evolutivo, los individuos seleccionados para participar en las operaciones genéticas son escogidos utilizando el método de selección por torneo con cuatro (4) individuos. A los individuos seleccionados se les aplican los operadores de cruce y mutación, el 80% de los individuos para la siguiente generación son producidos por mutación de los individuos seleccionados, mientras que el 20% restante es generado por cruce de individuos seleccionados. Se establece como criterio de parada un número máximo de cincuenta (50) generaciones o si es encontrado un individuo con un valor de desempeño ideal (sección 3.1).

Población	10 individuos
Generación inicial	Ramped half and half
Profundidad del árbol	17
Selección	Torneo con 4 individuos
Criterio de terminación	Generaciones: 50, aptitud ideal (igual a 1)
Aplicación op. de cruce	10%
Aplicación op. de mutación	80%

Tabla 4.3. Parámetros de ejecución para la tarea de agregación.

En cada ciclo evolutivo la nueva generación es evaluada en el simulador, para ello, el estado del escenario es restablecido a sus valores por defecto, posteriormente cada uno de los diez (10) individuos es asignado a todos los robos (un individuo es asignado a todos los robots y se evalúa su desempeño), los árboles son ejecutados por un período de sesenta (60) segundos, luego del cual es evaluado el desempeño según es presentado en la sección 4.1.1. El tiempo aproximado para realizar una ejecución completa de programación genética con el número de individuos y número de generaciones presentado en la tabla 4.3 es de ocho horas (se tiene en cuenta sólo el tiempo de evaluación de cada árbol).

4.3.1. Resultados

Con el fin de comparar los comportamientos generados por la plataforma computacional y a su vez verificar la suficiencia del conjunto primitivo para la tarea de agregación, se construye de forma manual (a partir del mismo conjunto primitivo utilizado en la plataforma computacional) un árbol que representa comportamientos de agregación. El árbol construido (árbol de referencia) es presentado en la figura 4.7.

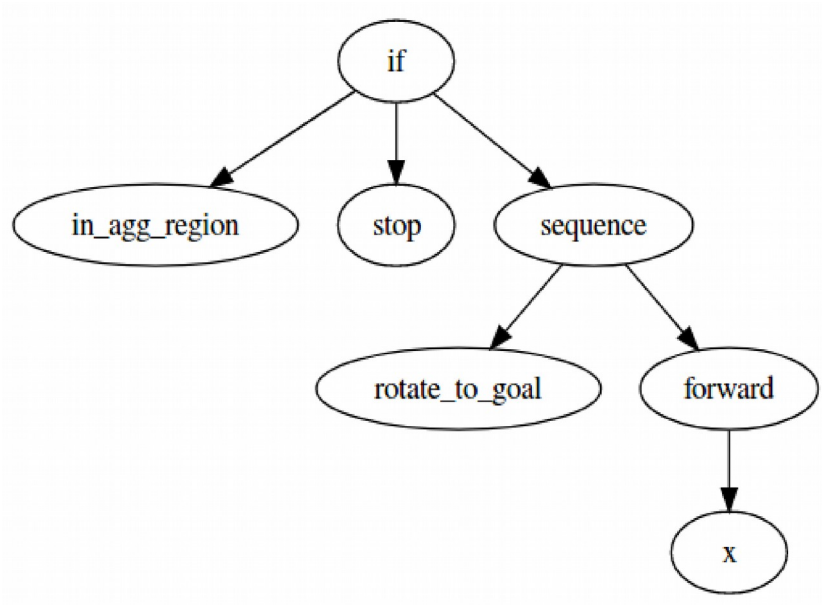


Figura 4.7. Árbol construido manualmente para la tarea de agregación.

El árbol contiene dos estructuras de control, una función de percepción del entorno y tres funciones de movimiento. La raíz del árbol está compuesta por una estructura condicional cuya condición verifica si el robot se encuentra dentro de la región de agregación. La función *in_agg_region* retorna el valor de verdad verdadero si el robot se encuentra dentro de la región de agregación y el valor falso en otro caso. Si la condición de la función *if* se evalúa a verdadero (el robot se encuentra dentro de la región de agregación) se ejecuta la función especificada en el segundo parámetro, en este caso, la función *stop* que detiene el movimiento del robot. La tercera rama del árbol está conformada por una estructura secuencial, que ejecuta la función *rotate_to_goal* y posteriormente la función *forward*. La función *rotate_to_goal* orienta al robot hacia la región de agregación, por su parte la función *forward* mueve el robot a la velocidad lineal especificada en su parámetro *x*. De esta forma, si el robot no se encuentra dentro de la región de agregación este rota hacia la región de agregación y se mueve en dicha dirección a la velocidad establecida, el robot se detiene cuando se encuentra dentro de la

región de agregación. Todo árbol es interpretado y ejecutado por el robot virtual en un ciclo de control a una frecuencia de 50Hz.

Conjunto homogéneo

La implementación de PG de la plataforma computacional es capaz de evolucionar comportamientos, para un conjunto homogéneo de robots, que en algunos casos (realizando algunas simplificaciones), son equivalentes al comportamiento construido de forma manual. Múltiples ejecuciones de la configuración experimental en la plataforma computacional construyen árboles con estructuras diferentes que producen comportamientos de agregación en un grupo de robots virtuales. En la figura 4.8 se presenta un árbol obtenido en una corrida del escenario experimental presentado.

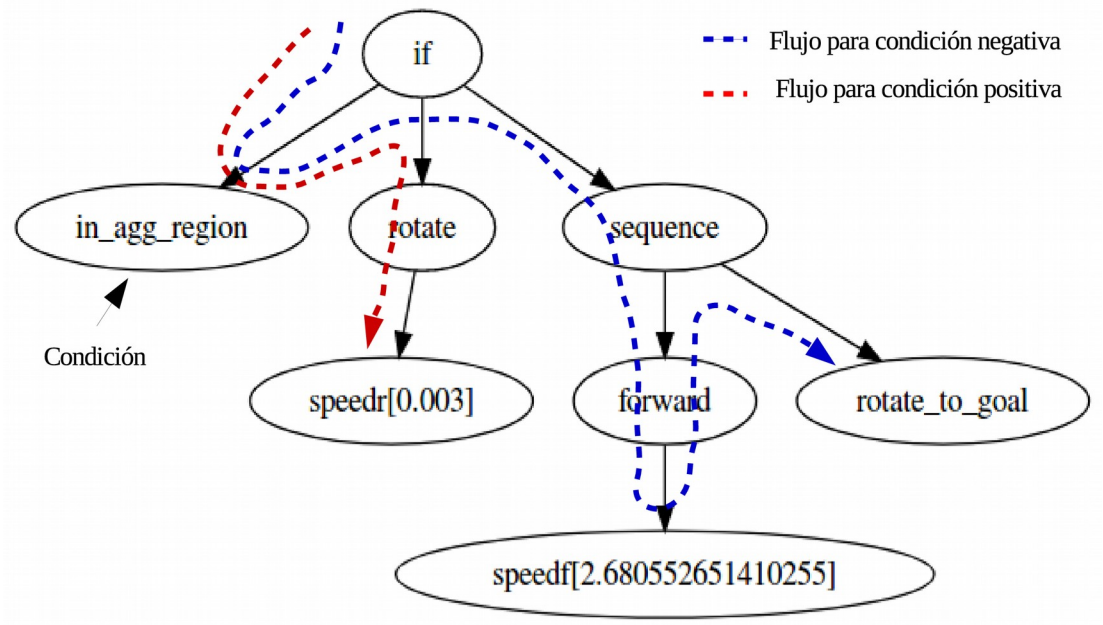


Figura 4.8. Árbol construido por programación genética para la tarea de agregación.

De forma similar al árbol construido manualmente, el árbol construido por programación genética posee dos estructuras de control. La raíz del árbol está constituida por la estructura de control condicional, en su nodo de condición se encuentra el nodo terminal *in_agg_region* que retorna el valor verdadero si el robot se encuentra dentro del área de agregación, de esta forma el robot siempre está verificando si se encuentra en el área de agregación. En la rama de bifurcación para el valor verdadero se encuentra la función de movimiento *rotate* con un valor de referencia de

0.003 radianes, lo que indica que una vez el robot ha detectado que se encuentra dentro de la región de agregación comenzará a rotar con incrementos de *0.003* radianes. Finalmente, la rama para la bifurcación negativa contiene la estructura de control de secuencia con dos funciones de movimiento. La primera función de movimiento es la función *forward* que produce un movimiento de avance lineal del robot a la velocidad especificada por el nodo terminal *speedf* (*2.68m/s*), la segunda función permite que el robot busque la región de agregación: *rotate_to_goal*. El árbol descrito produce el siguiente comportamiento en cada robot: a partir de su posición inicial el robot realiza un movimiento de avance (producido por la función *forward*) hacia la región de agregación (producido por la función *rotate_to_goal*), cuando el robot llega a la región de agregación y el estado es detectado (*in_agg_region*) el robot rota con incrementos de *0.003* radianes, el robot nunca se detiene sino que está en constante movimiento. Desde el punto de vista funcional, el árbol construido, cuando es asignado a todo el conjunto de robots, hace que los robots se agrupen dentro del área de agregación. Sin embargo, desde el punto de vista energético, el árbol no es eficiente, ya que los robots continúan en movimiento aún después de alcanzar su objetivo. Este resultado puede explicarse considerando el hecho de que la función de evaluación del desempeño no considera ninguna componente adicional al cálculo de la distancia al centro de la región de agregación, como restricciones o penalizaciones por movimiento indeseado de los robots. A pesar de que la función de desempeño no contiene ningún tipo de restricción, la implementación de PG es capaz de producir comportamientos que resuelven la tarea (como el construido de forma manual), como es ilustrado más adelante.

En la figura 4.9 se muestra otro árbol, producido por la plataforma, que es capaz de resolver el problema de agregación. Sin embargo, en este caso, el árbol tiene una estructura más compleja y con código que no agrega funcionalidad (región no codificante). Este árbol representa un comportamiento de agregación. Sin embargo, posee estructuras de control condicionales que no aportan a la solución y otras funciones que no son ejecutadas. Como es indicado por la línea punteada de color azul, si el robot se encuentra dentro de la región de agregación este detendrá su movimiento, el flujo va hasta el último condicional que lleva a la ejecución de la función que detiene el movimiento del robot (*stop*). Por su parte y como lo indica la línea punteada de color rojo, si el robot no se encuentra dentro del área de agregación este mantiene su movimiento lineal hasta encontrar los límites del escenario, y debido a que esta función contiene una rutina de detección de obstáculos el robot cambiará de dirección para continuar con su movimiento lineal, de esta forma, el robot explorará aleatoriamente el escenario hasta encontrar el área de agregación en cuyo caso detiene su movimiento.

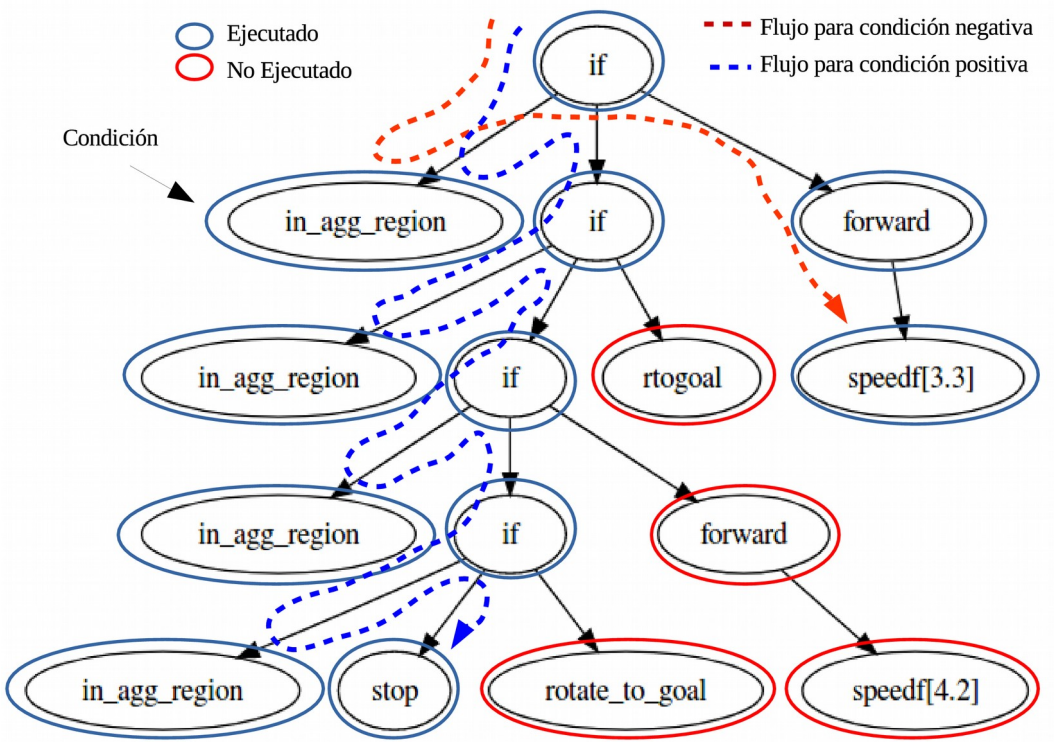


Figura 4.9. Árbol con región no codificante.

El flujo de ejecución está condicionado a ser alguno de los indicados por las líneas punteadas, por lo que las funciones resaltadas en óvalos de color rojo nunca son ejecutadas. Nótese que, a diferencia del árbol presentado en la figura 4.8, este árbol nunca ejecuta la función que orienta al robot hacia el área de agregación (*rotate_to_goal*). Este árbol hace que el robot realice una exploración aleatoria y en algún momento el área de agregación es encontrada produciendo el mismo resultado, posiblemente, empleando mayor tiempo de ejecución. La solución cumple el objetivo definido en la función de evaluación del desempeño, ya que los robots se agrupan en la región de agregación. En la figura 4.10 se presenta una versión simplificada equivalente del árbol presentado en la figura 4.9.

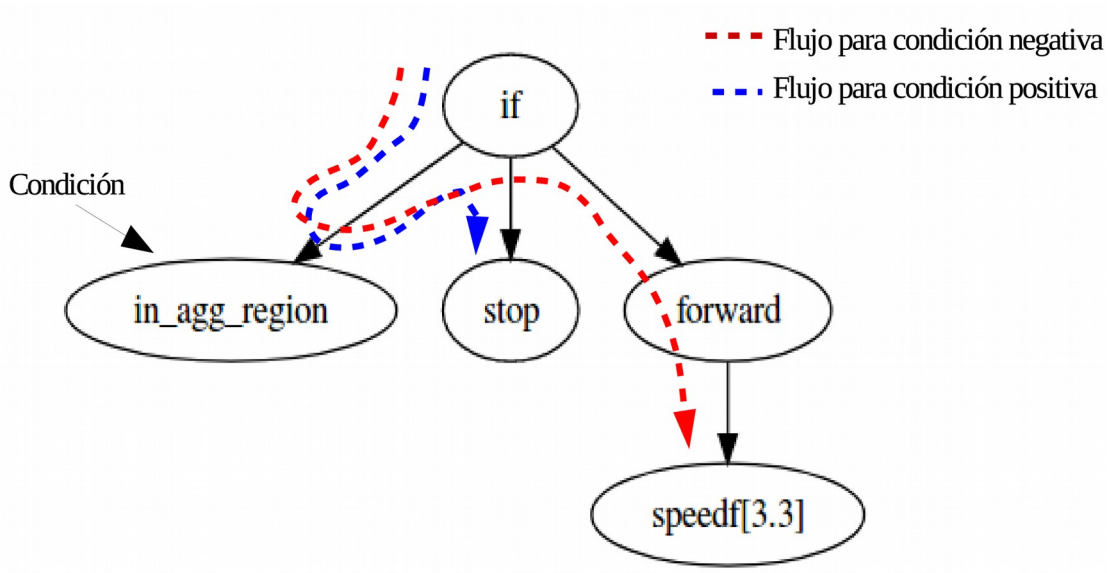


Figura 4.10. Árbol simplificado equivalente.

Como se mencionó anteriormente, la plataforma computacional sintetiza árboles similares al árbol de referencia (figura 4.7). En la figura 4.11 se presenta un árbol, estructural y funcionalmente equivalente al árbol de referencia, construido por programación genética. El árbol de la figura 4.11 comparte con el árbol de referencia las estructuras de control condicional y secuencia, las funciones de movimiento *forward* y *stop*, así como también la función de rotación hacia la región de agregación. A diferencia del árbol de referencia, incluye el operador lógico de negación de forma repetitiva. Si se observa la rama que evalúa la condición del nodo *if*, el operador *not* se encuentra tres veces consecutivas lo que hace posible realizar una eliminación de dos de estas funciones sin que el funcionamiento del árbol se vea alterado. La existencia del nodo *not* (posterior a la eliminación) hace que los flujos de ejecución para las condiciones positiva y negativa se encuentren intercambiadas respecto al árbol de referencia. De esta forma, el resultado de evaluar el árbol construido por programación genética es completamente equivalente al resultado producido por el árbol de referencia.

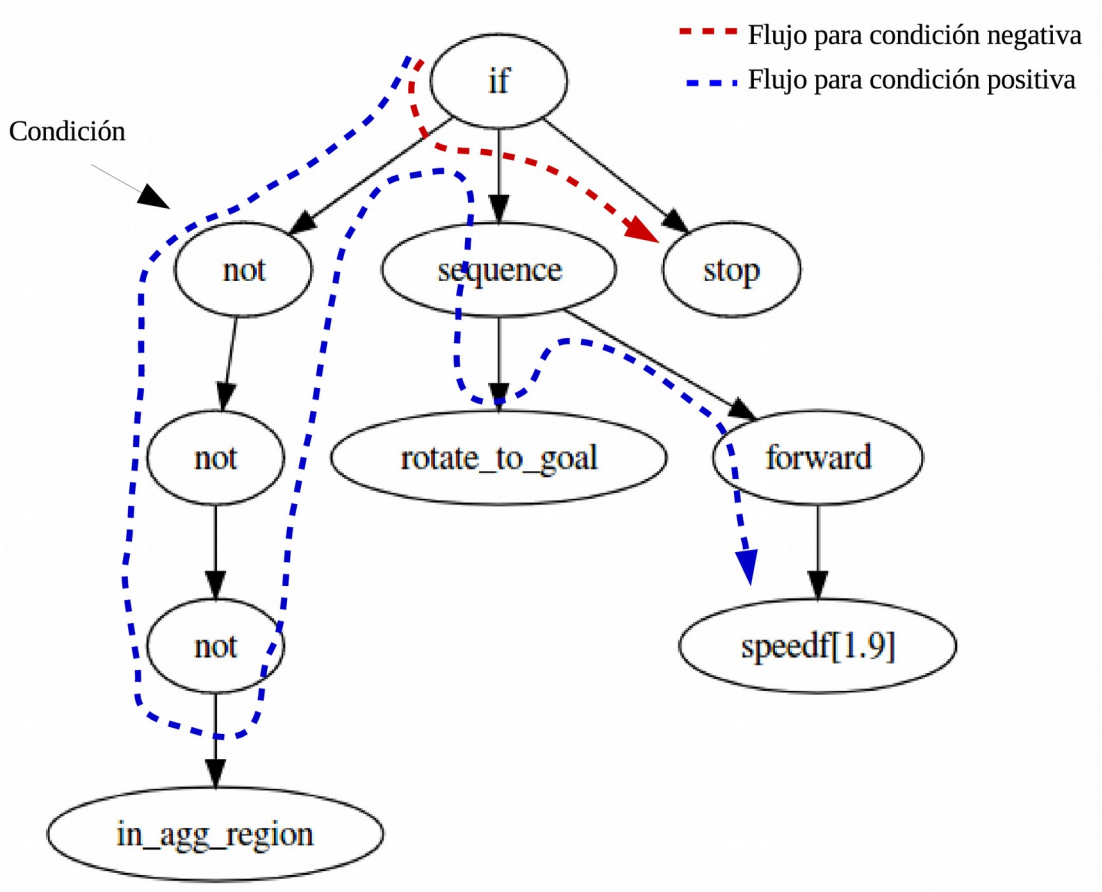


Figura 4.11. Árbol construido por programación genética equivalente al árbol de referencia.

Conjunto Heterogéneo

Los resultados obtenidos (como se esperaba) muestran que el mejor individuo es un conjunto de tres árboles que difieren en su estructura pero que hacen que cada robot se agrupe en la región de agregación. En la figura 4.12 se presentan los árboles obtenidos para la tarea de agregación utilizando el conjunto heterogéneo de robots aproximado.

Los árboles, de igual forma que para el caso del conjunto homogéneo, poseen una porción no codificante, o código redundante. El árbol uno (1) incluye en su estructura funciones que no se ejecutan, así como estructuras condicionales anidadas que no aportan al desempeño del comportamiento representado. El árbol dos (2) tiene código redundante, al incluir dos llamados consecutivos a la función de movimiento *forward*, de igual forma, el árbol tres (3) posee dos llamados al operador *not*, que pueden ser eliminadas sin afectar la funcionalidad de los árboles.

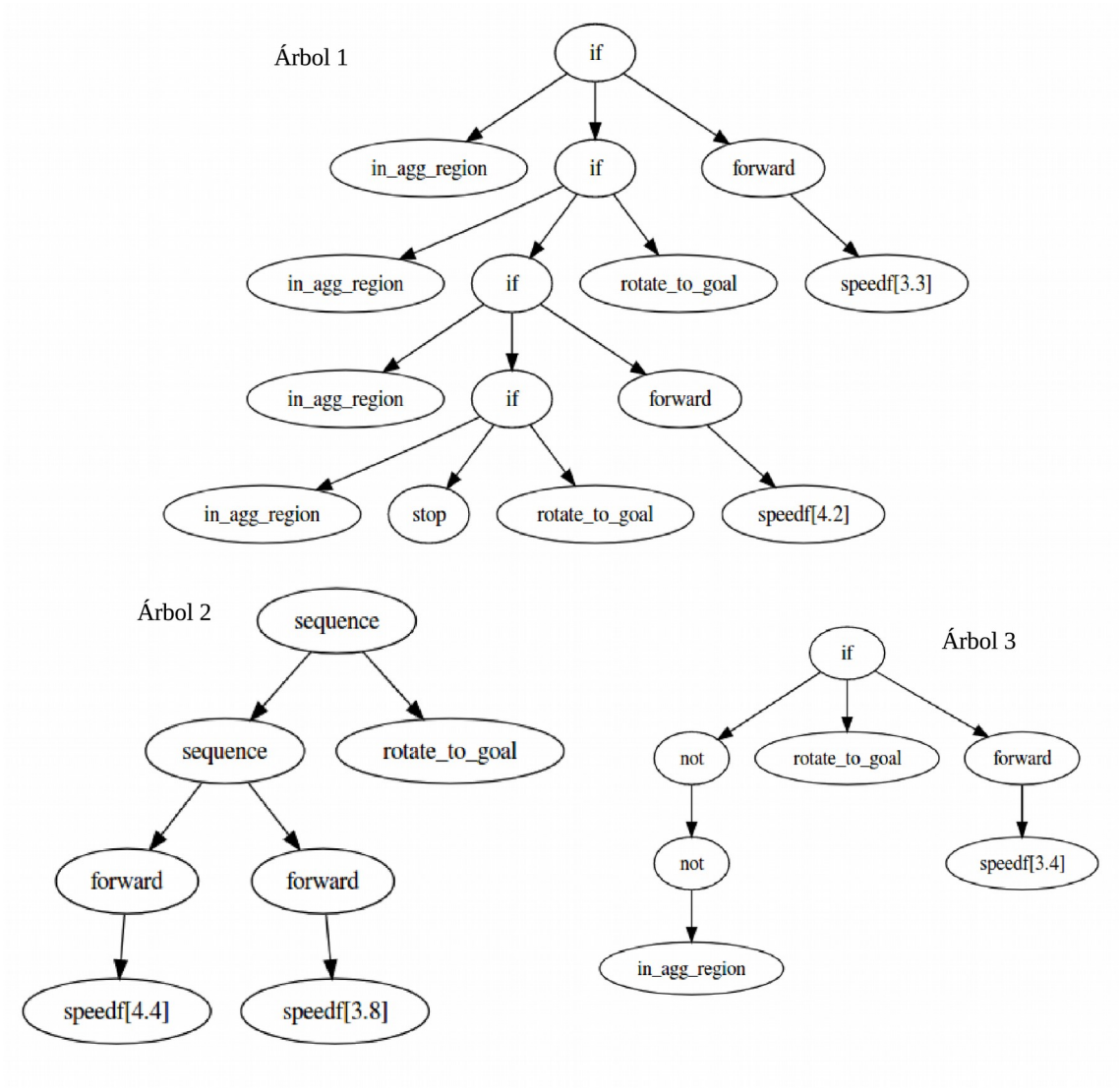


Figura 4.12. Árboles obtenidos para la tarea de agregación (grupo heterogéneo de robots).

El árbol uno (1) realiza exploración aleatoria del entorno hasta encontrar la región de agregación, aunque este árbol posee la función *rotate_to_goal*, que orienta al robot en dirección a la región de agregación, esta nunca se ejecuta. Por su parte, los árboles dos (2) y tres (3) incluyen la función que orienta al robot hacia la región de agregación, lo cual hace que los robots no realicen exploración aleatoria, sino que en cada ciclo de control estos se orientan y avanzan hacia la región de agregación. Finalmente, los árboles uno (1) y dos (3) detienen el movimiento del robot una vez este alcanza la región de agregación, ejecutando continuamente las funciones *stop* y *rotate_to_goal*. Sin embargo, el árbol dos (2) no detiene el movimiento del robot, este se mantiene oscilando alrededor del punto central de la región de agregación, este comportamiento se explica de la siguiente manera: el árbol sólo contiene las funciones de movimiento *forward* y *rotate_to_goal* las cuales son ejecutadas de forma secuencial, esto hace que el robot se oriente y avance hacia la región de agregación; cuando el robot llega a la región

este continúa su movimiento (pasando por el centro de la región de agregación) hasta que un cambio en la orientación entre el robot y el área de agregación es detectado por la función *rotate_to_goal*, la cual lo llevará nuevamente a orientarse en dicha dirección. Sin embargo, y como se explicó anteriormente, el árbol cumple con el objetivo, determinado por la función de aptitud, de llevar el robot a la región de agregación.

En la figura 4.13 se presentan algunas capturas de pantalla de la ejecución de algunos árboles obtenidos en las diferentes ejecuciones del escenario experimental presentado en este capítulo. La figura incluye la posición inicial de los robots antes de la ejecución de los árboles y tres resultados diferentes tras la ejecución de diferentes comportamientos obtenidos con la implementación de programación genética integrada a la plataforma computacional. Videos demostrativos de la plataforma computacional y de los experimentos realizados en este trabajo se encuentran disponibles en [69].

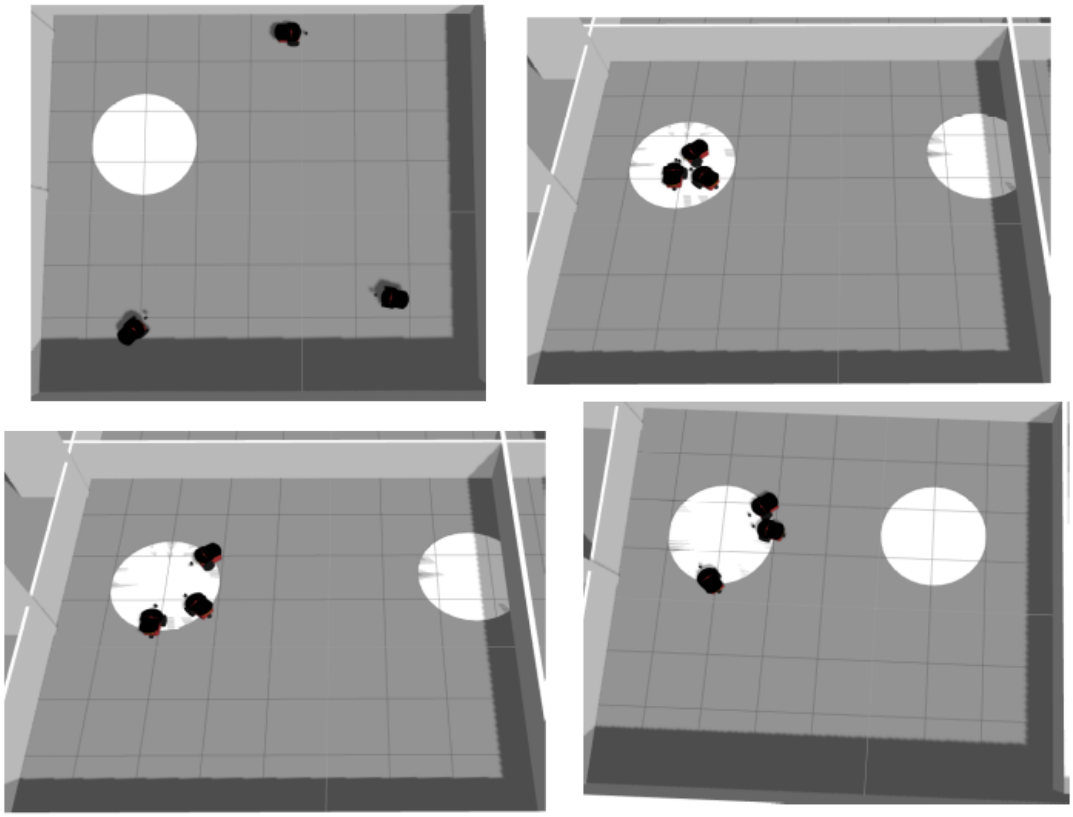


Figura 4.13. Resultado de diferentes comportamientos de agregación obtenidos por programación genética (superior izq – posición inicial).

4.4. Despliegue de los comportamientos en robots reales

Con el fin de verificar el desempeño de los comportamientos generados por la plataforma en robots reales, se despliegan tales comportamientos en tres robots del laboratorio fábrica experimental de la Universidad Nacional de Colombia sede Bogotá (LabFabEx). El beneficio de desplegar los comportamientos en robots reales se da en dos sentidos. Por un lado, se verifica la consistencia funcional de los comportamientos generados por la plataforma, propuesta en este trabajo, tanto en robots virtuales como en robots reales. Por otro lado, se realiza todo el desarrollo y despliegue de la infraestructura necesaria para la asignación y ejecución de los comportamientos en robots reales, permitiendo que la plataforma computacional pueda ser utilizada como una herramienta para la investigación y aplicación de técnicas de robótica de enjambres en los robots del Laboratorio Fábrica experimental. En la figura 4.14 se ilustra el flujo de trabajo e integración de la plataforma computacional en el laboratorio Fábrica Experimental.

La plataforma computacional se integra al Laboratorio Fábrica Experimental como una herramienta que facilita el diseño de los comportamientos del grupo de robots para resolver tareas específicas utilizando evolución artificial. El usuario accede a la plataforma a través de un portal web que le brinda las herramientas necesarias para describir las propiedades de los robots y sus primitivas, así como las tarea que se quiere resolver con el grupo de robots (por medio del formato de descripción SSL). Basado en estas descripciones, el sistema construye programas (para cada robot) que podrían resolver la tarea especificada. Si bien, la plataforma no resolverá cualquier tarea encontrada en un entorno complejo de manufactura como el LabFabEx, si puede construir de forma recursiva módulos que, al ser conectados, pueden resolver una tarea compleja. Tal como fue presentado en este capítulo, la plataforma construye soluciones al problema de la agregación, que a su vez es un bloque funcional de otras tareas más complejas.

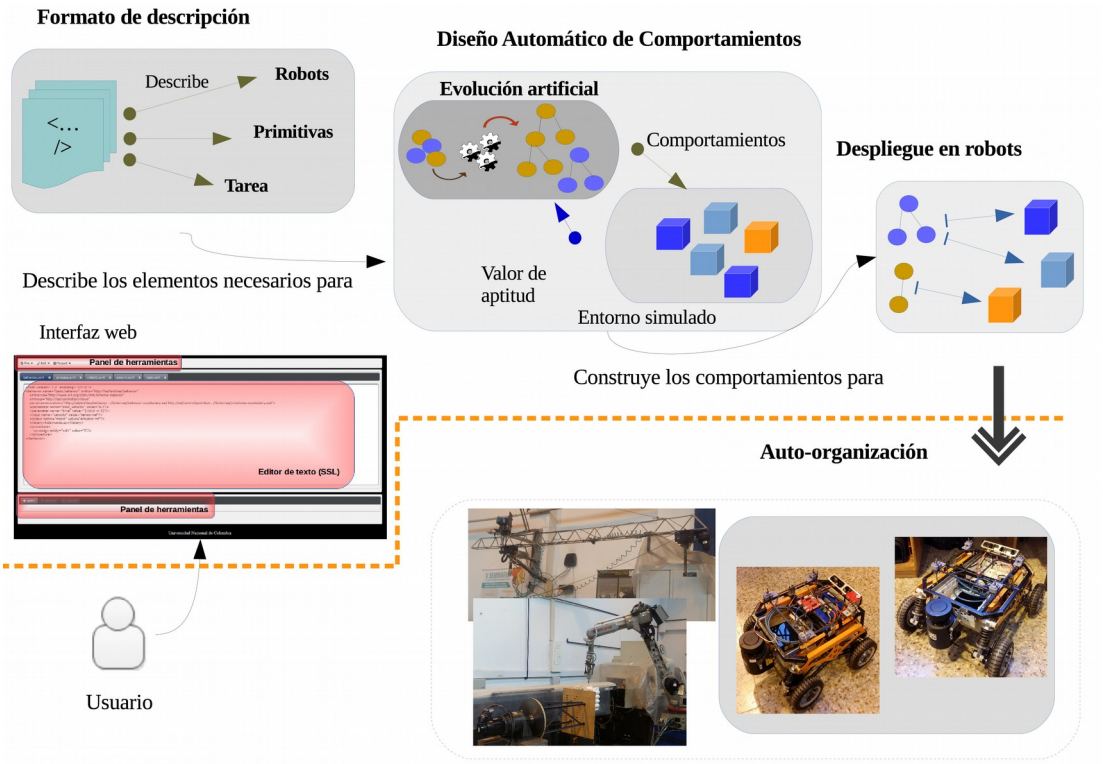


Figura 4.14. Flujo del proceso de construcción automática de comportamientos en la plataforma computacional para los robots del LabFabEx.

4.4.1 Descripción del Laboratorio Fábrica Experimental

El entorno de aplicación de los robots es el Laboratorio Fábrica Experimental de la Universidad Nacional de Colombia. El Laboratorio es una minifábrica experimental compuesta por tres zonas de manufactura: la zona de máquinas industriales, la zona de máquinas experimentales y la zona de prototipado rápido.

La zona de manufactura industrial está compuesta por dos equipos de maquinado multiejes (Torno tipo suizo, centro de mecanizado multiejes) y dos robots manipuladores seriales (Motoman y SCARA). En estas máquinas es posible realizar piezas de diversas geometrías y materiales (aceros, aluminios, polímeros), así como, el ensamble de piezas.

La zona de máquinas experimentales está compuesta por máquinas diseñadas y desarrolladas por estudiantes e Ingenieros en diversos proyectos. Estas máquinas incluyen: máquinas para el mecanizado multiejes (máquina multiejes), máquina de medición por coordenadas (CMM), máquina tipo Gantry para el posicionamiento de piezas y un robot manipulador híbrido (Maya).

La zona de prototipado rápido está compuesta por máquinas industriales y experimentales para la fabricación de prototipos por adición de material con tecnología FDM (Fused Deposit Modelling).

En la figura 4.15 se muestra la virtualización del LabFaEx en el simulador de la plataforma computacional. La zona de agregación, utilizada en este experimento, se ilustra en la región circular de la celda experimental.

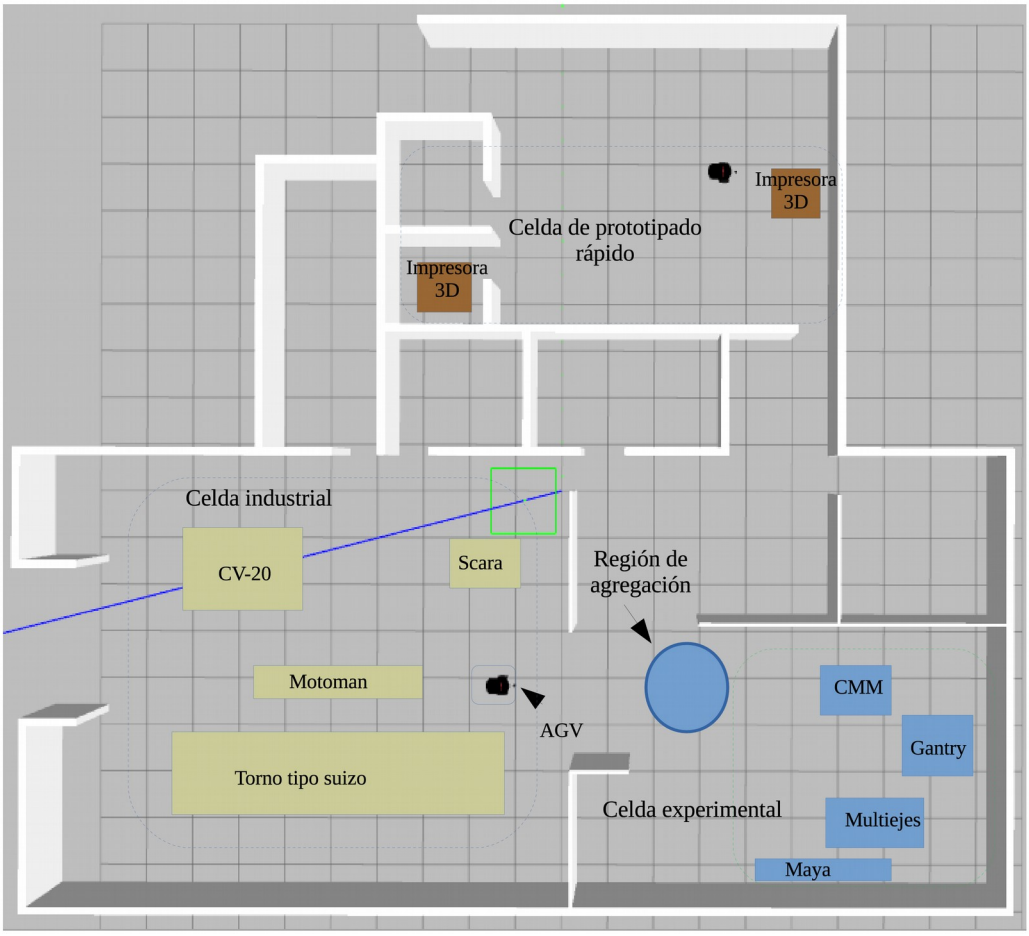


Figura 4.15. Modelo virtual del LabFabEx en el simulador Gazebo.

4.4.2. Descripción de los robots móviles del Laboratorio Fábrica Experimental

Los robots del LabFabEx son tres robots móviles de arquitectura diferencial de cuatro ruedas diseñados y desarrollados por estudiantes e Ingenieros de la Universidad Nacional. Los robots cuentan con diferentes tipos de sensores, tal como se indica en la tabla 4.4. Estos sensores y sus sistemas de procesamiento abordo le permiten a los

robots percibir y moverse en el entorno de manufactura evadiendo obstáculos. En la figura 4.16 se ilustran los robots móviles del LabFabEx.

Todos los módulos de software de los robots están implementados en el framework ROS, con nodos para el acceso a sensores, interfaces de control de movimiento, elementos de control para la navegación y sistema de comunicación inalámbrica.

Sensores	**Variables**	**Funciones**
Láser sick LMS 102	Distancia	Mapeo/Localización
Unidad de medición inercial (IMU)	Aceleración y orientación en tres ejes	Localización
Unidad RFID		Rastreo de material
Sensores Ultrasónicos	Distancia	Detección de obstáculos
Galgas Extensiométricas	Fuerza	Peso de la carga
		Procesamiento / Software
Computador abordo	Intel NUC	
Software	ROS	
		Propiedades mecánicas
Dimensiones	H40xW49xD60	
Velocidad máxima	0.6 m/s	
Capacidad de carga	10 Kg	

Tabla 4.4. Características generales de los robots móviles del LabFabEx.

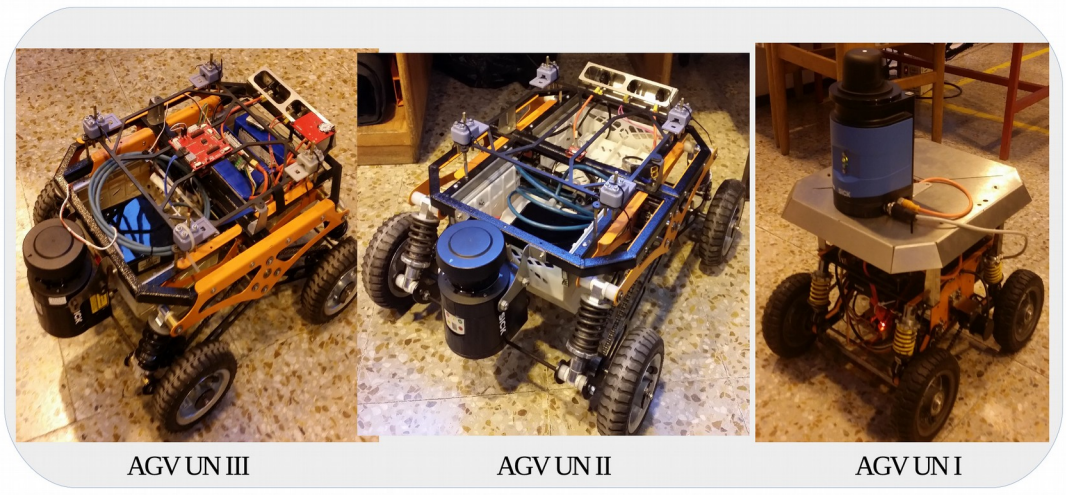


Figura 4.16. Robots móviles del LabFabEx.

4.4.3. Asignación y ejecución de los comportamientos

Se desarrolla e integra, a la plataforma computacional, un módulo de comunicaciones que utiliza el modelo de comunicaciones publicador/suscriptor de ROS utilizado por los robots, para asignar los comportamientos a los robots. Bajo este esquema, la plataforma publica un canal de comunicaciones (ROS Topic) para cada robot, por el cual envía un mensaje (ROS Message) que contiene el árbol construido por PG para la tarea de agregación, un intérprete en el controlador del robot se encarga de ejecutar el árbol asignado. En la figura 4.17 se muestra el grafo simplificado de los módulos ROS involucrados en el proceso de asignación de los comportamientos para la tarea de agregación (este esquema se mantiene para la asignación de comportamientos para cualquier otra tarea).

Cuatro módulos intervienen en el proceso de asignación. Los controladores de los robots, indicados en la figura 4.17 como controladores uno (1), dos (2) y tres (3), reciben e interpretan los árboles. Los controladores son nodos ROS que corren en los computadores de los robots móviles del LabFabEx. El módulo de asignación (es un nodo ROS que corre en el servidor de la plataforma computacional) es el encargado de asignar a cada robot el árbol correspondiente. La comunicación entre el módulo de asignación y los controladores de los robots es establecida a través de los topics ROS, indicados en la figura como topic uno (1), topic dos (2) y topic tres (3).

Los árboles son asignados secuencialmente por la plataforma computacional. La ejecución de cada árbol, por parte de los robots, se inicia con la recepción del árbol y no se usa alguna señal de sincronización para el inicio de la ejecución de los árboles.

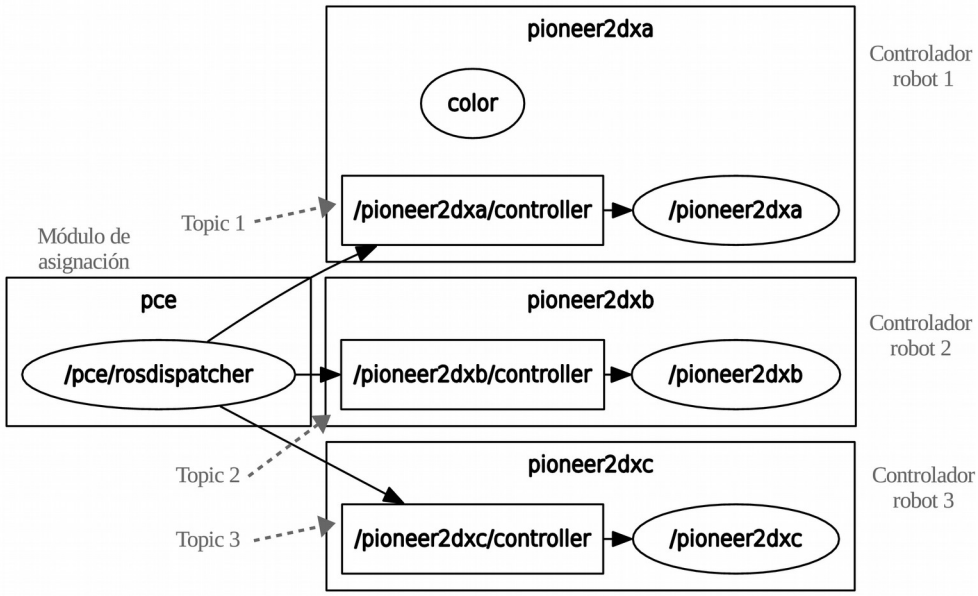


Figura 4.17. Grafo ROS indicando las relaciones entre nodos para la asignación de comportamientos a los robots del LabFabEx.

En la figura 4.18 se presentan las posiciones finales de los robots (para varias ejecuciones) en la región de agregación (ver figura 4.15), el árbol asignado a cada robot es el mejor árbol construido por PG y corresponde al árbol presentado en la figura 4.11. Los robots inician en las posiciones indicadas en la tabla 4.5, las distancias se miden respecto al punto central del área de agregación. En esta tabla también se indican los tiempos que emplean los robots en llegar a la región de agregación.

	Distancia a la región de agregación (m)	**Tiempo (s)**
1	Robot 1: 1.3 Robot 2: 1.7 Robot 3: 2.0	21 23 24
2	Robot 1: 2.4 Robot 2: 2.8 Robot 3: 3.0	26 27 29
3	Robot 1: 3.4 Robot 2: 3.9 Robot 3: 4.2	31 33 35

Tabla 4.5. Configuración inicial de los robots y tiempos de llegada a la región de agregación.

Figura 4.18. Comportamientos de agregación de los robots móviles del LabFabEx.

Conclusiones

En este trabajo se abordó el diseño automático de software para un conjunto de robots utilizando programación genética. Se planteó un esquema para representar tanto grupos homogéneos de robots como grupos heterogéneos en un individuo de programación genética (PG). Cuando los robots comparten las mismas habilidades dicha representación ofrece la posibilidad de inducir un sólo programa para todos los robots. Por su parte, cuando el grupo posee diversas habilidades, la representación ofrece la posibilidad de inducir, de forma simultánea, varios programas para los robots. De igual forma, se plantearon operadores genéticos de cruce y mutación para aplicar a los individuos cuando representan grupos homogéneos y heterogéneos durante el proceso de evolución artificial. Con tal representación y operadores fue posible inducir programas para que el comportamiento de agregación espacial emergiera en un grupo de robots virtuales y reales.

Con el fin de facilitar el proceso de diseño automático de comportamientos para un grupo de robots y desplegar tales comportamientos en robots reales se diseñó e implementó una plataforma computacional. La plataforma integra herramientas para que el diseñador pueda describir todos los elementos necesarios en el proceso de diseño, tales como: el grupo de robots y sus propiedades diversas, un conjunto de primitivas para cada robot, la tarea para el grupo de robots, y el escenario en el que estos realizan la tarea. Esta capacidad de descripción es proporcionada por medio de un formato basado en marcas XML. La capacidad de diseño automático es proporcionada por una implementación de programación genética que sintetiza árboles que son validados en un simulador basado en física. La capacidad de despliegue de los comportamientos en robots reales es proporcionada por un módulo de comunicaciones con el que es posible asignar los comportamientos evolucionados a robots del Laboratorio Fábrica Experimental de la Universidad Nacional de Colombia.

Por medio de la plataforma computacional fue posible construir programas para cada miembro de un conjunto de robots para comportamientos emergentes específicos (utilizando programación genética, modelos virtuales de los robots y del entorno objetivo, modelos de tareas y primitivas bien definidas). La obtención de estos comportamientos específicos indican que la representación de PG para grupos homogéneos y heterogéneos, y los respectivos operadores genéticos de cruce y mutación propuestos, representan y exploran el espacio de todos los posibles programas construidos a partir de los conjuntos primitivos especificados. Esto permite (aunque posiblemente no para todo el universo de tareas) obtener soluciones para grupos homogéneos y heterogéneos de robots. Como se presentó, se indujo el comportamiento de agregación para un grupo propiamente homogéneo, y los resultados obtenidos al aproximar el conjunto heterogéneo de robots, sugieren que es posible inducir programas para un conjunto propiamente heterogéneo. Inducir programas, (estructuralmente

diferentes) para un grupo homogéneo representándolo en PG como si fuese un grupo heterogéneo (replicar el mismo conjunto primitivo para cada árbol del individuo), que producen comportamientos de agregación, abre la posibilidad a que programas, construidos a partir de diferentes conjuntos primitivos, puedan ser inducidos para que un conjunto heterogéneo ejecute una tarea específica.

Otro aspecto importante es que la implementación de modelos virtuales de los robots y del entorno, en el simulador basado en física adjunto a la plataforma computacional, proporcionó la información necesaria para evaluar el desempeño de los programas generados (por programación genética), mitigando la necesidad de robots reales, en la fase de validación del proceso de construcción automática de los comportamientos. Si bien, los modelos virtuales de los robots y el entorno no son una representación fiel de sus contrapartes reales, el nivel de abstracción (proporcionado por el formato SSL y el simulador) es suficiente para producir soluciones a tareas específicas que pueden ser desplegadas en robots reales.

La convergencia del proceso de evolución artificial a una solución para la tarea abordada depende, entre otras cosas, de la complejidad de dicha tarea, la función de optimización utilizada para definir la tarea y sus restricciones, el tiempo empleado para probar los comportamientos en el simulador y la suficiencia del conjunto primitivo definido para cada robot. Un aspecto importante, observado en la etapa experimental de este trabajo, es el tiempo de validación de una solución candidata. Si el tiempo especificado para evaluar los comportamientos en el simulador no es suficiente, los robots podrían no ejecutar completamente el programa asignado y/o no emerger el comportamiento global. En el caso de la tarea de agregación, se observó que para períodos cortos de evaluación, los robots no tenían el tiempo suficiente (dada una velocidad de movimiento) para llegar a la región de agregación, lo que afectaba el valor de desempeño de soluciones potenciales haciendo que PG no encontrará una solución durante el período de evolución especificado.

Por su parte, el conjunto primitivo desempeña (como en cualquier otra aplicación de programación genética) un papel crucial para la inducción de los programas para las tareas abordadas. Dado que PG genera la estructura de una solución, la suficiencia del conjunto primitivo le permite construir estructuras que tienen los elementos necesarios para solucionar una tarea. Adicionalmente, la funcionalidad codificada en cada primitiva influye en la convergencia de PG y las soluciones encontradas. Con primitivas muy simples PG puede requerir un número muy grande de iteraciones (en algunos casos inaceptable para ciertas aplicaciones) para estructurar una solución. En los experimentos se observó que, con primitivas básicas de movimiento, PG sólo construía comportamientos de exploración aleatoria del entorno en un número determinado de iteraciones. Al encapsular este comportamiento de exploración aleatoria como una primitiva, en una nueva ejecución PG construyó el comportamiento de agregación en un número de iteraciones similar. Este resultado también indica que es posible construir soluciones a problemas más complejos de forma recursiva, es decir, estructurando

soluciones en un nivel de complejidad que se convierten en primitivas para estructurar soluciones en otro nivel de complejidad. El comportamiento de agregación obtenido en este trabajo puede ser utilizado como bloque constructivo para inducir comportamientos más complejos tales como el forrajeo y el transporte colectivos.

Los comportamientos obtenidos, por la plataforma computacional a partir de las funciones de optimización definidas, poseen regiones no codificantes, elementos redundantes y estructuras diversas. La función utilizada para la tarea de agregación da cuenta de este efecto, dicha función calcula la cercanía de los robots a la región de agregación sin incluir restricciones que influyan en la estructura de las soluciones. Definir restricciones en la estructura de las soluciones o mecanismos de procesamiento posterior debería mitigar o eliminar tales efectos.

Finalmente, al integrar elementos importantes, (tales como: el uso de un formato de descripción para especificar robots, primitivas y tareas, herramientas de simulación, un algoritmo evolutivo y módulos de asignación) a la plataforma computacional, facilita la investigación de comportamientos para la coordinación de robots en el campo de la robótica de enjambres.

Trabajo Futuro

La plataforma computacional propuesta en este trabajo es una herramienta que puede ser utilizada para construir, de forma automática, programas para un grupo de robots en en el ámbito de la robótica de enjambres. Aunque los resultados obtenidos son prometedores, algunos aspectos pueden ser complementados y/o mejorados.

En este trabajo se construyeron comportamientos para tareas de agregación espacial, si bien, este resultado es importante, ya que demuestra la capacidad de la plataforma para construir soluciones a tareas específicas relativamente sencillas, su aplicación para encontrar soluciones a tareas más completas (forrajeo colectivo, formación de patrones, transporte colaborativo), y en últimas, soluciones a problemas prácticos del mundo real (transporte de material en celdas de manufactura), aún plantea varios retos.

Uno de estos retos tiene que ver con la capacidad de la plataforma para representar las tareas y sus restricciones. Como fue presentado, el formato de descripción propuesto (SSL) permite declarar funciones (que representan los objetivos de las tareas) y la forma en que tales funciones son ejecutadas (restricciones). Las restricciones soportadas son de secuencia (orden en el que se ejecutan las funciones) y tiempos de activación, esto limita la posibilidad de describir otro tipo de relaciones entre los objetivos de tareas más complejas. Por ejemplo, ciertas tareas requieren que se ejecuten algunos de sus objetivos sólo si otros pudieron ser ejecutados. Para soportar esta y otras restricciones el formato SSL debe ser extendido.

Otro aspecto a tratar está relacionado con la construcción recursiva de soluciones para las tareas abordadas. La implementación actual de la plataforma construye comportamientos (utilizando técnicas de evolución artificial) para una tarea en una sola fase, es decir, trata de resolver la tarea en una única ejecución de programación genética con el conjunto primitivo inicialmente especificado. Esta aproximación puede generar soluciones para cierto tipo de tareas; sin embargo, encontrar soluciones para problemas del mundo real podría requerir una aproximación incremental en la que, soluciones encontradas en una ejecución de PG (fase) constituyen el conjunto primitivo para una segunda ejecución de PG. Para que la plataforma computacional soporte esta aproximación, deben establecerse criterios para determinar cuándo finaliza una fase y mecanismos para que las soluciones encontradas en una fase sean encapsuladas como primitivas para la fase siguiente (todo esto de forma automática).

En términos de las soluciones obtenidas por PG, se observó que los árboles poseían regiones no codificantes y otros elementos que incrementan la complejidad estructural del programa sin aportar a la solución. Esto podría ser mitigado estableciendo restricciones de desempeño en el proceso de construcción o realizando procesos de limpieza a las soluciones finales.

Otros aspectos funcionales, para facilitar el uso de la plataforma computacional como herramienta de investigación, que pueden ser tratados en trabajos futuros son la integración de las interfaces Web y del simulador en una única interfaz para el acceso unificado a la plataforma computacional y el desarrollo de módulos de monitoreo en tiempo real de la ejecución de los comportamientos en los robots reales.

Bibliografía

[1] Banik, S. C., Watanabe, K., & Izumi, K. (2007). Task allocation with a cooperative plan for an emotionally intelligent system of multi-robots. *SICE Annual Conference 2007*. doi:10.1109/sice.2007.4421131.

[2] BrBrambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence, 7*(1), 1-41. doi:10.1007/s11721-012-0075-2.

[3] Brutschy, A. (2009). Task allocation in swarm robotics. towards a method for self-organized allocation to complex tasks. Master's thesis, IRIDIA, Universit´e Libre de Bruxelles, Brussels, Belgium, Brussels, Belgium. Rapport d'avancement de recherche.

[4] Dambrosio, D. B., Lehman, J., Risi, S., & Stanley, K. O. (2011). Task switching in multirobot learning through indirect encoding. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. doi:10.1109/iros.2011.6094509.

[5] Dias, M. B., Zlot, R., Kalra, N., & Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. Proceedings of the IEEE, 94(7), 1257-1270.

[6] Doncieux, S., & Mouret, J. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence, 7*(2), 71-93. doi:10.1007/s12065-014-0110-x.

[7] Dorigo, M. Floreano, D. Gambardella, L.M. Mondada, F. Nolfi, S. Baaboura, T. Birattari, M. Bonani, M. Brambilla, M. Brutschy, A. Burnier, D. Campo, A. Christensen, A.L. Decugniere, A. Di Caro, G. Ducatelle, F. Ferrante, E. Forster, A. Martinez Gonzales, J. Guzzi, J. Longchamp, V. Magnenat, S. Mathews, N. Montes de Oca, M. O'Grady, R. Pinciroli, C. Pini, G. Retornaz, P. Roberts, J. Sperati, V. Stirling, T. Stranieri, A. Stutzle, T. Trianni, V. Tuci, E. Turgut, A.E. Vaussard, F. 2013. Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms. *Robotics & Automation Magazine, IEEE*, *20*(4), 60-71.

[8] Ehsaei, M. S., Heydarzadeh, Y., Aslani, S., & Haghighat, A. T. (2008). Pattern-Based Planning System (PBPS): A novel approach for uncertain dynamic multi-agent environments. 2008 3rd International Symposium on Wireless Pervasive Computing. doi:10.1109/iswpc.2008.4556263.

[9] Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., & Birattari, M. (2014). AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence, 8*(2), 89-112. doi:10.1007/s11721-014-0092-4.

[10] García, P., Caamaño, P., Duro, R. J., & Bellas, F. (2013). Scalable Task Assignment for Heterogeneous Multi-Robot Teams. *International Journal of Advanced Robotic Systems, 10*(2), 105. doi:10.5772/55489.

[11] Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. The International Journal of Robotics Research, 23(9), 939-954.

[12] Goldingay, H., & Mourik, J. V. (2013). Distributed Sequential Task Allocation in Foraging Swarms. *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*. doi:10.1109/saso.2013.14.

[13] Hecker, J. P., & Moses, M. E. (2015). Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms. *Swarm Intelligence*. doi:10.1007/s11721-015-0104-z.

[14] Liu, W. (2008). Design and modelling of adaptive foraging in swarm robotic systems (Doctoral dissertation, Faculty of Environment and Technology, University of the West of England, Bristol).

[15] Lopes, A. L., & Botelho, L. M. (2007, April). Task decomposition and delegation algorithms for coordinating unstructured multi agent systems. In Complex, Intelligent and Software Intensive Systems, 2007. CISIS 2007. First International Conference on (pp. 209-214). IEEE.

[16] Meng, Y., & Gan, J. (2008, June). Self-adaptive distributed multi-task allocation in a multi-robot system. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on (pp. 398-404). IEEE.

[17] Nouyan, S., Campo, A., & Dorigo, M. (2007). Path formation in a robot swarm. *Swarm Intelligence, 2*(1), 1-23. doi:10.1007/s11721-007-0009-6.

[18] Ohkura, K., Yasuda, T., & Matsumura, Y. (2013, October). Coordinating the Collective Behavior of Swarm Robotics Systems Based on Incremental Evolution. In Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on (pp. 4024-4029). IEEE.

[19] Parker, L. E. (2008). Distributed intelligence: overview of the field and its application in multi-robot systems. *Journal of Physical Agents (JoPha), 2*(1), 5-14. doi:10.14198/jopha.2008.2.1.02.

[20] Parker, L. E. (1996). L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. Advanced Robotics, 11(4), 305-322.

[21] Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., & Birattari, M. (2011). Task partitioning in swarms of robots: an adaptive method for strategy selection. *Swarm Intelligence, 5*(3-4), 283-304. doi:10.1007/s11721-011-0060-1.

[22] Pugh, J., & Martinoli, A. (n.d.). Distributed Adaptation in Multi-robot Search Using Particle Swarm Optimization. *Lecture Notes in Computer Science From Animals to Animats 10,* 393-402. doi:10.1007/978-3-540-69134-1_3.

[23] Quiñonez, Y., Tostado, I., & Sánchez, O. (2013, November). Coordination Model for Multi-robot Systems Based on Cooperative Behaviors. In Artificial Intelligence (MICAI), 2013 12th Mexican International Conference on (pp. 33-37). IEEE.

[24] Riedmiller, M., Gabel, T., Hafner, R., & Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots, 27*(1), 55-73. doi:10.1007/s10514-009-9120-4.

[25] Santos, V. G., Pimenta, L. C., & Chaimowicz, L. (2014, May). Segregation of multiple heterogeneous units in a robotic swarm. In Robotics and Automation (ICRA), 2014 IEEE International Conference on (pp. 1112-1117). IEEE.

[26] Sariel-Talay, S., Balch, T. R., & Erdogan, N. (2011). A Generic Framework for Distributed Multirobot Cooperation. *Journal of Intelligent & Robotic Systems, 63*(2), 323-358. doi:10.1007/s10846-011-9558-4.

[27] Shi, Z., Tu, J., Li, Y., & Wei, J. (2014). Modeling of Task Planning for Multirobot System Using Reputation Mechanism. *The Scientific World Journal, 2014*, 1-12. doi:10.1155/2014/818701.

[28] Shucker, B., & Bennett, J. K. (n.d.). Scalable Control of Distributed Robotic Macrosensors. *Distributed Autonomous Robotic Systems 6,* 379-388. doi:10.1007/978-4-431-35873-2_37.

[29] Soysal, O., & Sahin, E. (2005, June). Probabilistic aggregation strategies in swarm robotic systems. In Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE (pp. 325-332). IEEE.

[30] Spears, D., Kerr, W., & Spears, W. (2006). Physics-based robot swarms for coverage problems. International Journal on Intelligent Control and Systems, 11(3), 11-23.

[31] Szwaykowska, K., Romero, L. M., & Schwartz, I. B. (2015). Collective Motions of Heterogeneous Swarms. *IEEE Transactions on Automation Science and Engineering, 12*(3), 810-818. doi:10.1109/tase.2015.2403253.

[32] Torreño, A., Onaindia, E., & Sapena, Ó. (2012). A flexible coupling approach to multi-agent planning under incomplete information. *Knowledge and Information Systems, 38*(1), 141-178. doi:10.1007/s10115-012-0569-7.

[33] Waibel, M., Keller, L., & Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. Evolutionary Computation, IEEE Transactions on, 13(3), 648-660.

[34] Alitappeh, R. J., Jeddisaravi, K., & Guimarães, F. G. (2016). Multi-objective multi-robot deployment in a dynamic environment. *Soft Computing, 21*(21), 6481-6497. doi:10.1007/s00500-016-2207-x.

[35] Suthambut, K., & Polvichai, J. (2011). Evolutionary Artificial Neural Networks on simulated soccer robots. *2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*. doi:10.1109/jcsse.2011.5930105.

[36] Messom, C., & Walker, M. (2002). Evolving cooperative robotic behaviour using distributed genetic programming. *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.* doi:10.1109/icarcv.2002.1234823.

[37] Ramya, R. (2010). Evolving Bio-Inspire Robots For Keep Away Soccer Through Genetic Programming. *NTERACT-2010*, Chennai, 2010, pp. 329-333.

[38] Larik, A., Haider, S. (2016). On Using Evolutionary Computation Approach for Strategy Optimization in Robot Soccer. *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI), Rawalpindi, 2016, pp. 11-16*

[39] Kala, R. (2012). Multi-robot path planning using co-evolutionary genetic programming. *Expert Systems with Applications, 39*(3), 3817-3831. doi:10.1016/j.eswa.2011.09.090.

[40] Lee, J., Ahn, C. (2012). Evolutionary Self-Assembling Swarm Robots using Genetic Programming. 2012 Proceedings of SICE Annual Conference (SICE), Akita, 2012, pp. 807-811.

[41] ECJ,A Java-based Evolutionary Computation Research System, www.cs.umd.edu/projects/plus/ec/ecj/.

[42] Nishikawa, N., Suzuki, R., & Arita, T. (2016). Coordination control design of heterogeneous swarm robots by means of task-oriented optimization. *Artificial Life and Robotics, 21*(1), 57-68. doi:10.1007/s10015-015-0255-4.

[43] WSO2 Microservices Framework for Java https://github.com/wso2/msf4j.

[44] Poli, R., Langdon, W., McPhee, N. (2008). A Field Guide to Genetic Programming. Computer Science Faculty, 2008, [online] Available: http://digitalcommons.morris.umn.edu/cs_facpubs/l.

[45] Şahin, E. (2005). Swarm Robotics: From Sources of Inspiration to Domains of Application. *Swarm Robotics Lecture Notes in Computer Science,* 10-20. doi:10.1007/978-3-540-30552-1_2.

[46] Pires, A. G., Macharet, D. G., & Chaimowicz, L. (2015). Exploring heterogeneity for cooperative localization in Swarm Robotics. *2015 International Conference on Advanced Robotics (ICAR)*. doi:10.1109/icar.2015.7251488.

[47] M. Dorigo and E. Şahin. (2004) Swarm robotics. Au-tonomous Robots, 17(2–3):111–113.

[48] Gazi, V., & Fidan, B. (n.d.). Coordination and Control of Multi-agent Dynamic Systems: Models and Approaches. *Swarm Robotics Lecture Notes in Computer Science,* 71-102. doi:10.1007/978-3-540-71541-2_6.

[49] Bayindir, L., & Şahin, E. (2007). A review of studies in swarm robotics. Turkish Journal of Electrical Engi-neering, 15(2), 115–147.

[50] Byington, M. D., & Bishop, B. E. (2008). Cooperative Robot Swarm Locomotion Using Genetic Algorithms. *2008 40th Southeastern Symposium on System Theory (SSST)*. doi:10.1109/ssst.2008.4480232.

[51] Trianni, V. (2008). Evolutionary swarm robotics. Evolving Self-Organizing Behaviors in Groups of Autonomous Robots. Studies in Computational Intelligence, Volume 108.

[52] Rubenstein, M., Ahler, C., & Nagpal, R. (2012). Kilobot: A low cost scalable robot system for collective behaviors. *2012 IEEE International Conference on Robotics and Automation*. doi:10.1109/icra.2012.6224638.

[53] Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science, 345*(6198), 795-799. doi:10.1126/science.1254295.

[54] Werner, M., Dyer, M. (1993). Evolution of herding behavior in artificial animals. In J.-A. Meyer, H. Roitblat, and S. W. Wilson, editors, From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB 92), pages 393–399. MIT Press, Cambridge, MA

[55] Reynolds, C. (1993). An evolved, vision-based behavioral model of coordinated group motion. In J.-A. Meyer, H. Roitblat, and S. W. Wilson, editors, From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB 92), pages 384–392. MIT Press, Cambridge, MA.

[56] Michel, O. (2004). Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems, 1*(1), 5. doi:10.5772/5618.

[57] VREP Simulator, http://www.coppeliarobotics.com/.

[58] Vaughan, R. (2008). Massively multi-robot simulation in stage. *Swarm Intelligence, 2*(2-4), 189-208. doi:10.1007/s11721-008-0014-4.

[59] Koenig, N., & Howard, A. (2004.). Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. doi:10.1109/iros.2004.1389727.

[60] Pinciroli, C., Trianni, V., Ogrady, R., Pini, G., Brutschy, A., Brambilla, M., . . . Dorigo, M. (2011). ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. doi:10.1109/iros.2011.6048500.

[61] Luke, S., Hohn, C., Farris, J., Jackson, G., & Hendler, J. (1998). Co-evolving Soccer Softbot team coordination with genetic programming. *RoboCup-97: Robot Soccer World Cup I Lecture Notes in Computer Science,* 398-411. doi:10.1007/3-540-64473-3_76.

[62] Macedo, J., Marques, L., & Costa, E. (2017). Robotic odour search: Evolving a robots brain with Genetic Programming. *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. doi:10.1109/icarsc.2017.7964058.

[63] Simulation Definition Format. http://sdformat.org/spec.

[64] Brooks, R. (1991). Intelligence without reason. In R. Myopoulos and J. Reiter, editors, Proceedings of the 12 th International Joint Conference on Artificial Intelligence, pages 569–595. Morgan Kaufmann Publishers, San Francisco, CA.

[65] Koza, J. R. (1990). *Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems*.. Stanford University, CA.

[66] Robot pioneer 2DX. http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx

[67] Robot Operatig System. http://www.ros.org/

[68] Open Source Robotics Foundation. https://www.osrfoundation.org/

[69] Videos demostrativos de la plataforma computacional. https://github.com/labfabexun/cp-sdl.