DISEÑO DE ALGORITMO DE AUTOCONFIGURACIÓN CONSIDERANDO DOS ARQUITECTURAS (1D Y 2D) PARA SISTEMA ROBÓTICO MODULAR

Ing. Geiber Aquileo Rodríguez Lombana

Universidad Militar Nueva Granada

Facultad de Ingeniería

Maestría de Ingeniería en Mecatrónica

Bogotá D.C.

2019

DISEÑO DE ALGORITMO DE AUTOCONFIGURACIÓN CONSIDERANDO DOS ARQUITECTURAS (1D Y 2D) PARA SISTEMA ROBÓTICO MODULAR

Ing. Geiber Aquileo Rodríguez Lombana

Tutor:

Ing. Ricardo Andrés Castillo Estepa, PhD

Trabajo de Grado para Optar por el Título de Magíster en Ingeniería Mecatrónica

Universidad Militar Nueva Granada

Facultad de Ingeniería

Maestría de Ingeniería en Mecatrónica

Bogotá D.C.

2019

ACEPTACIÓN

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

Firma Director

_____

Firma Jurado 1

_____

Firma Jurado 2

Bogotá, _____

Fecha de sustentación

DEDICATORIA

Este trabajo se lo quiero dedicar a Dios, a mi princesita Mónica, a mis padres, a mis hermanas y aquellas personas que siempre han creído en mí.

A Dios por las bendiciones constantes, por la vida tan maravillosa que me ha dado, por la buena salud, por permitirme vivir mi vida de la mejor manera posible, porque sé que aún queda camino que con valor y sacrificio me guiara al éxito para poder llegar cada vez más lejos.

A mi princesita Mónica por la magia que imprime en nuestro camino, por llenar mi vida de colores en un mundo agitado y lleno de grises, por su cariño sincero, por la motivación constante para que se pudiera alcanzar este logro, este nuevo objetivo; por ella seguiré trabajando con constancia, para alcanzar nuevas metas que nos permitan poder tener un futuro mejor y poder labrar juntos el gran fruto que brinda el amor.

A mis padres que con su sacrificio, entrega, apoyo y buenos consejos, me dieron más que la vida misma. Que bajo su buen ejemplo y carácter hicieron lo mejor para formar un ciudadano de buenas costumbres, pues bien sé que sin ellos esto no hubiera podido ser posible.

A mis hermanas por ser parte esencial en mi vida, porque siempre han estado presentes y por ser de quienes he podido compartir los valores enseñados por nuestros padres, sin duda son un buen ejemplo a seguir.

Se lo dedico también a todas las personas que han creído en mí y que de cierta manera me ayudaron a crecer en la parte personal y profesional, a esas personas que eventualmente estuvieron presentes y me brindaron la mano para poder seguir adelante.

AGRADECIMIENTOS

Agradezco a la UNIVERSIDAD MILITAR NUEVA GRANADA por ser un excelente espacio de formación y estudio, por darnos a sus estudiantes el privilegio de tener clases con profesores idóneos para desempeñar su labor que tienen la vocación de educar y una buena disposición para enseñar.

Es gracias a la Universidad que he podido desarrollarme como profesional de manera integral, el gran claustro que me permitió obtener anteriormente las competencias como ingeniero en mecatrónica, especialista en gerencia integral de proyectos y ahora como magíster en ingeniería mecatrónica.

Al profesor Ricardo Andrés Castillo Estepa por darme el privilegio de ser mi tutor de tesis, por quien siento un profundo respeto, una gran admiración, porque hizo posible la conclusión del trabajo y porque con sus constantes aportes llevo a la finalización del mismo.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE GRÁFICOS

LISTA DE CÓDIGOS

LISTA DE ANEXOS

RESUMEN

En este proyecto se realizará el diseño y simulación en software de un algoritmo de ensamble y coordinación de arquitecturas para un sistema compuesto por un conjunto de agentes robots que permitan a un sistema colaborativo multi-agente lograr dos diferentes arquitecturas (1D y 2D) para alcanzar el objetivo. Este algoritmo estará enfocado al desarrollo de tres actividades principales: primero el análisis y planificación de la trayectoria a ejecutar con las arquitecturas más eficientes para lograrlo, segundo el desplazamiento de cada módulo (en una posición inicial aleatoria) y su integración con los demás para formar una configuración, y por último (posteriormente) la configuración integrada se desplazará hasta un punto final definido por el usuario atravesando un entorno no estructurado (evasión de obstáculos), durante este recorrido el robot podrá re-armarse en una segunda configuración cuando sea necesario para evadir los obstáculos presentes. Cada configuración obtenida será considerada como un agente compuesto de la unión de múltiples módulos robóticos MECABOT, los cuales fueron diseñados y construidos previamente por el grupo de investigación DAVINCI del Programa de Ingeniería en Mecatrónica de la Universidad Militar Nueva Granada.

Con el propósito de aumentar la robustez del sistema robótico modular ante eventos no esperados en el ambiente de operación y/o posibles fallas en componentes de los agentes, se propone desarrollar una lógica de comando distribuida bajo la cual cada unidad contará con un control independiente, siempre liderados por el módulo principal, a modo que se pueda reconocer tales eventos y adaptar su plan de operación de forma autónoma. La validación del funcionamiento del sistema robótico se realizará mediante dos etapas: la simulación de cada unidad y luego del sistema colectivo utilizando un software diseñado.

Se desarrollará una estrategia de armado y desarmado eficiente para coordinar los módulos utilizando técnicas de inteligencia artificial la cual otorgará al sistema la capacidad de tomar decisiones automáticamente a fin de optimizar el rendimiento del mismo, llegando a reducir el tiempo de operación y consumo energético.

ABSTRACT

In this project we will design and simulate in software an algorithm for the assembly and coordination of architectures for a system composed of a set of robot agents that allow a multi-agent collaborative system to achieve two different architectures (1D and 2D) to reach the objective. This algorithm will be focused on the development of three main activities: first the analysis and planning of the trajectory to be executed with the most optimal architectures to achieve it, second the displacement of each module (in a random initial position) and its integration with the others to form a configuration, and finally (later) the integrated configuration will move to a final point defined by the user going through an unstructured environment (avoidance of obstacles), during this journey the robot can be re-armed in a second configuration when necessary to evade the present obstacles. Each configuration obtained will be considered as an agent composed of the union of multiple MECABOT robotic modules, which were previously designed and constructed by the DAVINCI research group of the Mechatronics Engineering Program of the New Granada Military University.

In order to increase the robustness of the modular robotic system in the event of unexpected events in the operating environment and / or possible failures in components of the agents, it is proposed to develop a distributed command logic under which each unit will have independent control, always led by the main module, so that you can recognize such events and adapt your operation plan autonomously. The validation of the operation of the robotic system will be done through two stages: the simulation of each unit and then the collective system using a software designed.

An efficient arming and disarming strategy will be developed to coordinate the modules using artificial intelligence techniques which will give the system the ability to automatically make decisions in order to optimize the performance of the same, reducing operating time and energy consumption.

CAPÍTULO I.

INTRODUCCIÓN

Las limitaciones que se han visto reflejadas a lo largo de la historia en el ser humano, han llevado a ser atendidas a partir de una idea particular, la cual posteriormente se desarrolla y se materializa hasta llegar a ser aceptada por las personas y adoptada en la vida cotidiana con la intención de mejorar el bienestar, la eficiencia o facilidad en el hacer de las cosas, generando nuevos puntos de referencia en el desarrollo de las diferentes ramas de la ciencia aplicada con el transcurrir del tiempo.

En el campo de la ingeniería aplicada, el desarrollo tecnológico permite entornos de trabajo en las industrias de las cuales se han fortalecido para optimizar sus procesos y ser más eficientes, llegando a implementar equipos robóticos a sus procesos productivos para lograrlo. El término robot se popularizó con el éxito de la obra R.U.R. (Robots Universales Rossum), escrita por Karel Čapek en 1920. En la traducción al inglés de dicha obra la palabra checa robota, que significa trabajos forzados o trabajador, fue traducida al inglés como robot, estos son diseñados con el fin de interactuar con el entorno de trabajo de forma autómata.

Existen varios tipos de robots, algunas de las más conocidas son según su aplicación, locomoción, arquitectura y posicionamiento, entre ellos están los fijos, que consisten en un brazo articulado sujetado a una base fija y los móviles que tienen la capacidad de moverse en su entorno y no se fijan a un solo lugar. Los robots móviles son un foco importante de la investigación actual puesto que sus aplicaciones se enfocan en la industria, el entretenimiento y el hogar, viéndose éstos en aspiradoras, overboard, vehículos guiados automáticamente AGV, etc.

En razón al entorno de desplazamiento del robot móvil, es necesario implementar algún tipo de mecanismo al dispositivo para que se desenvuelva para tal fin, siendo estas ruedas, patas, hélices, ventiladores, electroimanes, entre otros. Cada una de estas extensiones mecánicas dispone de ventajas sobre las demás, en características como la estabilidad, capacidad de adaptación a terrenos, tracción, tamaño y redundancia, sin obviar métricas como la velocidad de dichos sistemas robóticos que pueden variar dependiendo de sus extensiones a utilizar. Sistemas que a pesar que brindan buenas prestaciones en el campo de movimiento, presentan limitaciones físicas para cambiar y adaptar un nuevo tipo de forma, por lo que su estructura rígida presenta inconvenientes de desplazamiento por espacios estrechos o con obstáculos aleatorios.

Un diseño óptimo que puede adaptar su arquitectura para beneficiarse de las ventajas de los sistemas móviles, son los sistemas robóticos modulares, enfocados y diseñados para que se le permita al robot desplazarse por cualquier terreno que presente variedad de obstáculos, como diferentes entornos en los que el robot se desplazaría para así mismo diseñar una morfología apta para cada entorno; los robots modulares son conformados por múltiples bloques que permiten variar la estructura conformada con su unión para adaptarse a nuevas condiciones del lugar de trabajo y poder llegar incluso a realizar diferentes tareas específicas a diferencia de los robots fijos.

Los sistemas robóticos modulares autónomos reconfigurables son máquinas autónomas cinemáticas con morfología variable. Más allá del accionamiento convencional, detección y control que se encuentran típicamente en los robots de morfología fija, los robots modulares también son capaces de cambiar deliberadamente su propia forma reordenando la conectividad de sus partes, con el fin de adaptarse a las nuevas circunstancias, realizar nuevas tareas, o incluso recuperarse de eventuales daños.

Cada módulo está diseñado de forma elemental, lo que lo hace por si solo con un campo de acción limitado, de forma tal que al unir varios de ellos el resultado sea superior a la suma de los resultados de cada elemento o de cada parte actuando de forma aislada. La sinergia como principio que rige la robótica modular, enfatiza que cada vez que un eslabón se une a la cadena el robot se vuelve cada vez más robusto, con más grados de libertad, aunque buscando conservar el equilibrio de su peso para no afectar su versatilidad y hacer que el robot a implementar brinde la mejor solución.

Este trabajo está enfocado a realizar un algoritmo de autoconfiguración teniendo en cuenta el sistema robótico modular MECABOT 4.0 diseñado e implementado en la Universidad Militar Nueva Granada, el cual presenta la capacidad de coordinación de movimientos de los módulos permitiendo realizar la locomoción deseada, con el objetivo de realizar la configuración de adaptación morfológica con respecto al medio en que se desplaza, implementando algoritmos de inteligencia artificial en los que el robot modular se configura para cambiar su forma de manera automática,

considerando el medio de desplazamiento y el punto de llegada deseado, sin necesidad de la operación asistida durante el proceso. Este sistema permite incorporar la configuración de cada módulo por separado, con su controlador independiente, mediante el procesamiento en paralelo para cada uno y optimizando así el cambio a una nueva forma en el menor tiempo posible.

1.1. Planteamiento del Tema

Actualmente la robótica tradicional se encuentra en el mundo industrial llegando a ser indispensable para la producción en serie de diferentes productos, pero en diversas aplicaciones se requieren robots más versátiles los cuales puedan llegar a lugares que ni el hombre ni un robot tradicional podrían alcanzar. La misión de los sistemas modulares es que la funcionalidad en conjunto sea mayor que la suma de los componentes, además de las múltiples aplicaciones y ventajas que pueden llegar a brindar un conjunto de morfología variable.

Los robots modulares aparecen como una alternativa a los robots que están especializados en la realización de una tarea concreta. Un robot modular se conforma por un conjunto de elementos que pueden combinarse para la realización de diferentes tipos de tareas. Sin embargo, el proceso de auto configuración suele ser complejo y dispendioso cuando se realiza mediante un operador humano, por eso se desea automatizar este proceso mediante algoritmos de inteligencia artificial que de la mejor solución a la morfología para desplazarse en un entorno diferente. Por este motivo y viendo que hay poco desarrollo en el tema se pretende aportar un poco de investigación al campo de robótica modular, que tiene aplicación en Colombia en temas de rescate, exploración, prevención de desastres.

La arquitectura de diseño de muchos de los robots actuales limita la cantidad de problemas que se pueden resolver, sobre todo para aplicaciones como la exploración de lugares donde el acceso a humanos es imposible o riesgoso, por lo tanto el robot debe ser capaz de elegir la mejor forma a adoptar para alcanzar el objetivo. Puesto que los robots modulares no solo pueden entrar a terrenos irregulares y espacios reducidos, sino también presentan otra gran ventaja y es que ayudan a disminuir costos tanto de mantenimiento como adquisición de equipos sofisticados para estas tareas.

Con el fin de generar un aporte a los trabajos sobre robots modulares de configuración automática de arquitecturas, se ha realizado comparación con el desarrollo que existe como precedente en este campo de aplicación, siendo algunos de éstos:

- Modular Self-Reconfigurable Robots [1]

- Distributed replication algorithms for self-reconfiguring modular robots [2]
- Algorithms for self-reconfiguring molecule motion planning [3]
- Generic distributed assembly and repair algorithms for self-reconfiguring robots [4]
- Self-reconfiguring robots: designs, algorithms, and applications [5]
- Scalable parallel algorithm for configuration planning for self-reconfiguring robots [6]
- An efficient algorithm for self-reconfiguration planning in a modular robot [7]
- Distributed locomotion algorithms for self-reconfigurable robots operating on rough terrain [8]

Con base en la información disponible acerca del tema planteado, se exponen y se muestran en la Figura 1 una serie de limitaciones encontradas las cuales son utilizadas como foco de atención sobre la presente investigación, con el ánimo de solventar los problemas plasmados en procura de mejorar los sistemas robóticos modulares de auto configuración considerando obstáculos para la elección de las arquitecturas.

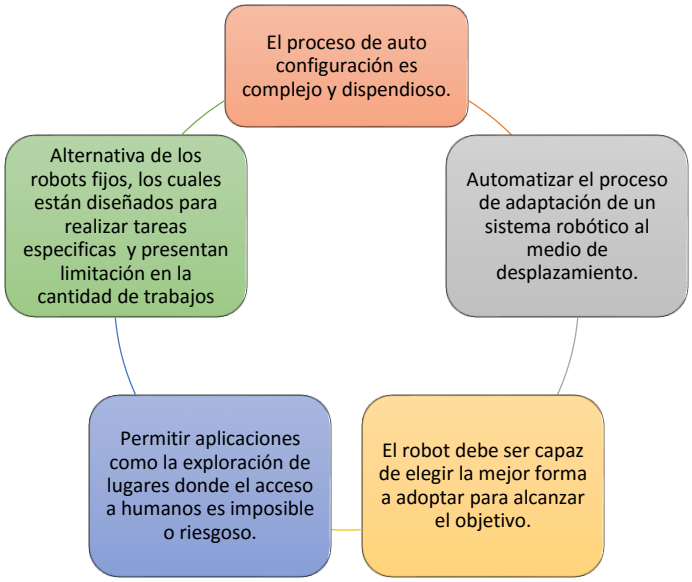


Figura 1. Limitaciones actuales de la robótica modular para la autoconfiguración de arquitecturas. Elaboración propia.

El proyecto consiste en el desarrollo de un algoritmo utilizando un sistema robótico modular que podrá reconfigurarse para formar dos arquitecturas (1D y 2D), el cual permita la conexión y desconexión entre los módulos robóticos, realizando su respectivo diseño y simulación con el fin de obtener un movimiento de translación

en diferentes direcciones aprovechando la versatilidad de los módulos para sortear terrenos difíciles a los que la robótica tradicional no podría acceder, se busca también que el robot sea capaz de elegir la mejor forma (entre las 2 propuestas) a adoptar para alcanzar el objetivo de manera automática.

De acuerdo a esto, se busca la adaptabilidad del robot al medio en el que se desplaza mediante la configuración automática de sus unidades que lo componen para poder así realizar actividades de desplazamiento que son de gran dificultad, o de difícil acceso debido a situaciones en las que no existan las condiciones físicas o donde se pueden presentar una serie de limitaciones espaciales para los seres humanos como son las actividades de exploración, búsqueda y rescate.

¿Cómo se puede lograr que un conjunto de módulos robóticos se ensamble automáticamente para obtener la arquitectura deseada 1D o 2D y pueda cambiar su configuración para la evasión de obstáculos considerando el entorno de trabajo?

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un algoritmo que permita la reconfiguración entre los módulos robóticos del MECABOT para el ensamble automático de dos arquitecturas (1D y 2D), que permita que el sistema integrado pueda realizar actividades de planificación de trayectorias y evasión de obstáculos.

1.2.2. Objetivos Específicos

1. Elaborar una base de datos con las diferentes configuraciones arquitectónicas para superar los obstáculos (2 arquitecturas), que permita llegar al punto final de manera automática.

2. Diseñar un algoritmo que permita implementar las funciones de desplazamiento, conexión y desconexión de los módulos o semi-módulos que se desean integrar en el sistema robótico.

3. Desarrollar un algoritmo de sistema lógico computacional que permita a los módulos configurarse de la forma más efectiva y eficiente teniendo en cuenta el entorno de desplazamiento, así como ejecutar la trayectoria que el conjunto de módulos desee superar.

4. Desarrollar una simulación 3D (utilizando el entorno Webots) que permita evaluar la efectividad del algoritmo a implementar para el armado automático de módulos según la arquitectura escogida y superar los obstáculos analizados.

5. Realizar pruebas de funcionamiento en ambiente de simulación 3D (utilizando software), y calcular indicadores de rendimiento del sistema.

1.3. Justificación

Con la realización de este trabajo se obtendrá un sistema robótico modular que tenga las ventajas técnicas, económicas y de eficiencia de un robot móvil con la capacidad de variar la estructura a una forma nueva a implementar, siendo la ventaja funcional la capacidad de brindar más robustez y mayor nivel adaptativo que los sistemas convencionales mediante el uso de algoritmos de inteligencia artificial, lo que se traduce en reconfiguración automática de ensamble y desensamble para formación de nuevas morfologías. Como ventaja económica la disminución del costo total realizando dispositivos con el mismo esquema de producción para cada módulo, lo que se materializa en economías a escala por la fabricación en serie. Además, la ventaja con respecto a eficiencia al realizar estructuras complejas de manera automática sin necesidad de que un operador pierda tiempo en su configuración, así como elegir la forma que se desempeña adecuadamente en el medio con los módulos que se arman en paralelo.

Con este trabajo se plantea servir como punto de referencia para el desarrollo de sistemas robóticos modulares en Colombia y para la investigación en la UMNG, puesto que es un campo con amplios temas por desarrollar y descubrir, por lo que esta tecnología ofrece en proyección a resolver problemas que requieren de diferentes equipos robóticos para realizar una misma tarea, así como también lo es en casos de desplazamiento debido a que estos sistemas modulares se adaptan al medio en que se desplaza. En este caso se utilizará para la autoconfiguración los módulos robóticos existentes en el grupo Davinci de la Universidad Militar Nueva Granada con el objetivo de adaptarse automáticamente a diferentes entornos.

1.4. Metodología

1. **Investigación sobre robótica modular, inteligencia artificial y de los diferentes tipos de conexión del robot para predefinir las posibles configuraciones morfológicas:**

a. En esta etapa se procedió a investigar acerca de la forma de aplicar la conexión y desconexión de los módulos del sistema robótico modular MECABOT y clasificación mediante base de datos de las diferentes arquitecturas morfológicas que el sistema robótico puede integrar, que de manera predeterminada se ejecutarán en razón a la trayectoria a realizar.

    i. Investigación, selección y aplicación de la forma de realizar la conexión y desconexión de los módulos del sistema robótico.
    ii. Realización de las bases de datos acerca de las configuraciones que el sistema robótico podrá plantear para superar los obstáculos (2 arquitecturas 1D y 2D).
    iii. Búsqueda y evaluación del método de inteligencia artificial adecuado para que el robot decida la forma a adoptar y la trayectoria a ejecutar para evadir los obstáculos.

2. **Diseño de un algoritmo que permita a los módulos robóticos configurarse de la forma más apropiada y ejecutar la trayectoria con obstáculos a superar:**

    a. En esta etapa se inició el proceso de diseño del algoritmo que según la revisión bibliográfica sea el más adecuado para desarrollar durante este proceso de auto configuración morfológica:

        i. Diseño de un algoritmo que pueda determinar la posición y rotación de cada uno de los módulos.
        ii. Diseño de un algoritmo que indique la rutina de ensamble de los módulos hasta formar la arquitectura deseada (de las dos planteadas).
        iii. Implementación de algoritmo de inteligencia artificial seleccionado para ejecutar la trayectoria y evasión de obstáculos que el robot desea superar.

3. **Desarrollo mediante simulación de la implementación del algoritmo de armado según la forma escogida y realizar la trayectoria para superar el obstáculo analizado:**

    a. Una vez implementado el algoritmo se evaluó mediante una función de idoneidad la adaptabilidad del robot al entorno, luego de evaluar la función se procedió a simular la auto-configuración del robot.

        i. Definición función de idoneidad.
        ii. Evaluación mediante la función de idoneidad como se adapta el robot al entorno.

iii. Según el resultado se realizó la iteración cuantas veces sea necesario hasta que se ajustara a un resultado de desplazamiento y ejecución de trayectoria adecuada.

4. **Diseño y parametrización de simulación 3D para visualización de resultados**

    a. Una vez realizados los algoritmos de ensamble, re-ensamble y planificación de trayectorias, se procedió a diseñar y parametrizar un ambiente virtual que permita la visualización y parametrización de los resultados a evaluar.

5. **Comprobación mediante indicadores de rendimiento la efectividad de las acciones realizadas en el ambiente virtual:**

    a. Medición de la eficacia a través de diferentes parámetros como los son el tiempo de decisión, la convergencia a la función ideal y las variaciones presentadas en al ambiente virtual.

CAPÍTULO II.

MARCO REFERENCIAL

2.1. Marco teórico

2.1.1. Robótica

La robótica es la rama de la ingeniería mecatrónica que se ocupa el diseño, construcción, operación, disposición estructural, manufactura y aplicación de los robots. [9] La robótica combina diversas disciplinas como lo son: la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física. [10] Otras áreas importantes en robótica son el álgebra, los autómatas programables, la animatrónica y las máquinas de estados. El término robot se popularizó con el éxito de la obra R.U.R. (Robots Universales Rossum), escrita por Karel Čapek en 1920. En la traducción al inglés de dicha obra la palabra checa robota, que significa trabajos forzados o trabajador, fue traducida al inglés como robot. [11]

2.1.2. Robótica Móvil

Existen robots fijos y móviles. Este documento se centra en los móviles debido a que es el objetivo de estudio. Un robot móvil es una máquina automática que es capaz de locomoción. Los robots móviles tienen la capacidad de moverse en su

entorno y no están fijos a una ubicación física, éstos son usados sólo para navegar o moverse sobre su espacio individual [12]. Los robots móviles pueden ser "autónomos" (AMR - robot móvil autónomo), lo que significa que son capaces de navegar en un entorno incontrolado sin necesidad de dispositivos de guía físicos o electromecánicos. Alternativamente, los robots móviles pueden confiar en dispositivos de guía que les permiten recorrer una ruta de navegación predefinida en un espacio relativamente controlado (AGV - vehículo guiado autónomo). Por el contrario, los robots industriales suelen ser más o menos estacionarios, consisten en un brazo articulado (manipulador multi-enlazado) y un conjunto de pinzas (o un efector final), unidos a una superficie fija.

Los robots móviles se han vuelto más comunes en entornos comerciales e industriales. Los hospitales han estado utilizando robots móviles autónomos para mover materiales durante muchos años. Los almacenes han instalado sistemas robóticos móviles para mover eficientemente los materiales de las estanterías a las zonas del cumplimiento de la orden. Los robots móviles son también un foco importante de la investigación actual y casi todas las principales universidades tienen uno o más laboratorios que se centran en la investigación de robots móviles. [13]

Los robots móviles también se encuentran en la industria, la configuración militar y de seguridad. Los robots domésticos son productos de consumo, incluyendo robots de entretenimiento y aquellos que realizan ciertas tareas domésticas, tales como aspirar o cultivar un huerto.

Los componentes de un robot móvil son un controlador, software de control, sensores y actuadores. El controlador es generalmente un microprocesador, micro controlador incorporado o un ordenador personal (PC). El software de control móvil puede ser lenguaje de nivel de ensamblaje o lenguajes de alto nivel como C, C ++, Pascal, Fortran o software especial en tiempo real. Los sensores utilizados dependen de los requisitos del robot. Los requisitos podrían ser la detección táctil y de proximidad, la triangulación, la evasión de colisiones, la ubicación de posición y otras aplicaciones específicas. [14]

2.1.3. Robótica Modular

Los robots modulares auto-reconfigurable son máquinas autónomas cinemáticas con morfología variable, los cuales están formados por un conjunto de módulos robóticos, cuya principal característica es la habilidad de cambiar de forma de los múltiples módulos robóticos que pueden ser fácilmente reconfigurables en diferentes ambientes de trabajo [15]. Más allá de accionamiento convencional, detección y control que se encuentran típicamente en los robots de morfología fija, los robots de auto-reconfiguración también son capaces de cambiar deliberadamente su propia forma reordenando la conectividad de sus partes, con el

fin de adaptarse a las nuevas circunstancias, realizar nuevas tareas, o recuperarse incluso de los daños.

Por ejemplo, un robot hecho de dichos componentes podría asumir una forma de gusano a moverse a través de un tubo estrecho, volver a montar en algo con las piernas en forma de araña para cruzar un terreno desigual, a continuación, formar un tercer objeto arbitrario para moverse rápidamente sobre un terreno bastante plano; sino que también puede ser utilizado para la fabricación de objetos "fijos", tales como paredes, refugios o edificios. Un sistema MSR – *Modular Self-Reconfigurable* puede cambiar su forma para adaptarse a la tarea, ya sea que esté subiendo por un agujero, rodando como un aro o ensamblando una estructura compleja con muchos brazos. [16]

En algunos casos esto implica que cada módulo tenga 2 o más conectores para la conexión de varios juntos. Pueden contener la electrónica, sensores, procesadores, memoria y fuentes de alimentación; también pueden contener actuadores que se utilizan para la manipulación de su ubicación en el medio ambiente y en relación unos con otros. Una gran cantidad de caras de conexión en cada módulo aumentará la cantidad de configuraciones que estos módulos pueden generar pero, por otro lado, el diseño mecánico será más complejo. Por lo tanto, hay una compensación entre la complejidad del módulo y la re-configurabilidad permitida. [17]

Al decir "auto-reconfiguración" o "auto-reconfigurable", significa que el mecanismo o dispositivo es capaz de utilizar su propio sistema de control, tales como actuadores o con medios estocásticos para cambiar su forma estructural global. Que tiene la cualidad de ser "modular" en "robótica modulares auto-reconfiguración" quiere decir que el mismo módulo o conjunto de módulos pueden ser añadidos o eliminados del sistema, en lugar de ser genéricamente "modular" en el sentido más amplio. La intención subyacente es tener un número indefinido de módulos idénticos, o un conjunto finito y relativamente pequeño de módulos idénticos, en una estructura de malla o matriz de módulos auto-reconfigurable.

La auto-reconfiguración también es diferente del concepto de auto-replicación, y la auto replicación no es necesariamente una cualidad que un módulo de auto-reconfiguración o una colección de tales módulos pueden o deben poseer. Una matriz de N-número de módulos no tiene que ser capaz de aumentar la cantidad de módulos para mayor que N sea considerado auto-reconfigurable. Los módulos de auto-reconfiguración son dispositivos que se producen en una fábrica convencional, donde en cadena de montaje, se ensamblan los componentes para construir cada módulo.

Una característica que se encuentra en algunos casos es la capacidad de los módulos para conectarse automáticamente y desconectarse a sí mismos hacia y desde uno al otro, y para formar en muchos objetos o realizar muchas tareas de

mover o manipular el medio ambiente. Hay dos tipos básicos de métodos de articulación que los mecanismos de auto-reconfiguración pueden utilizar para remodelar sus estructuras, la reconfiguración de la cadena y reconfiguración de celosía. [18]

La robótica modular se ha convertido recientemente en un gran interés para la comunidad robótica por varias razones, incluida la producción masiva económica de unidades, la degradación elegante de la función cuando está dañada y la capacidad de transición a topologías adecuadas para la tarea en cuestión. Dos beneficios adicionales son la capacidad de auto ensamblarse, que consiste en organizar un sistema de varias unidades a partir de una colección de unidades independientes, y la autoconfiguración, que es la capacidad de transición de una topología a otra mediante una serie de archivos adjuntos y separaciones . Además, la auto-reparación, que es un tipo especial de autoconfiguración, permite que un robot reemplace unidades dañadas con unidades nuevas, o que se reconfigure en una forma diferente para mantener la disponibilidad y permitir que el robot continúe con la tarea actual sin interrupciones. [19]

Según Mark Yim, el diseño, aplicación e implementación de los sistemas robóticos modulares se deben regir con el mandato de cumplir las tres promesas de la robótica modular [20], las cuales son la versatilidad, la fiabilidad y el bajo coste. La versatilidad puesto que estos sistemas deben ofrecer la posibilidad de cambiar su forma y desplazarse por entornos muy diversos. La fiabilidad es debido a que el sistema debe tener la capacidad de auto-reparación. Si uno de los módulos falla se elimina o se sustituye por otro. Finalmente, el bajo coste se consigue aplicando la economía de gran escala a la fabricación de los módulos. Si se fabrican masivamente, el precio se reducirá.

2.1.4. Clasificación de los Robots Modulares

Como lo considera Gonzales y Yim, se muestra en la Figura 2, la agrupación en diferentes clases, de acuerdo a las propiedades de locomoción de los sistemas robóticos modular, las configuraciones que se pueden adoptar basado en la estructura y el conexionado entre los módulos.

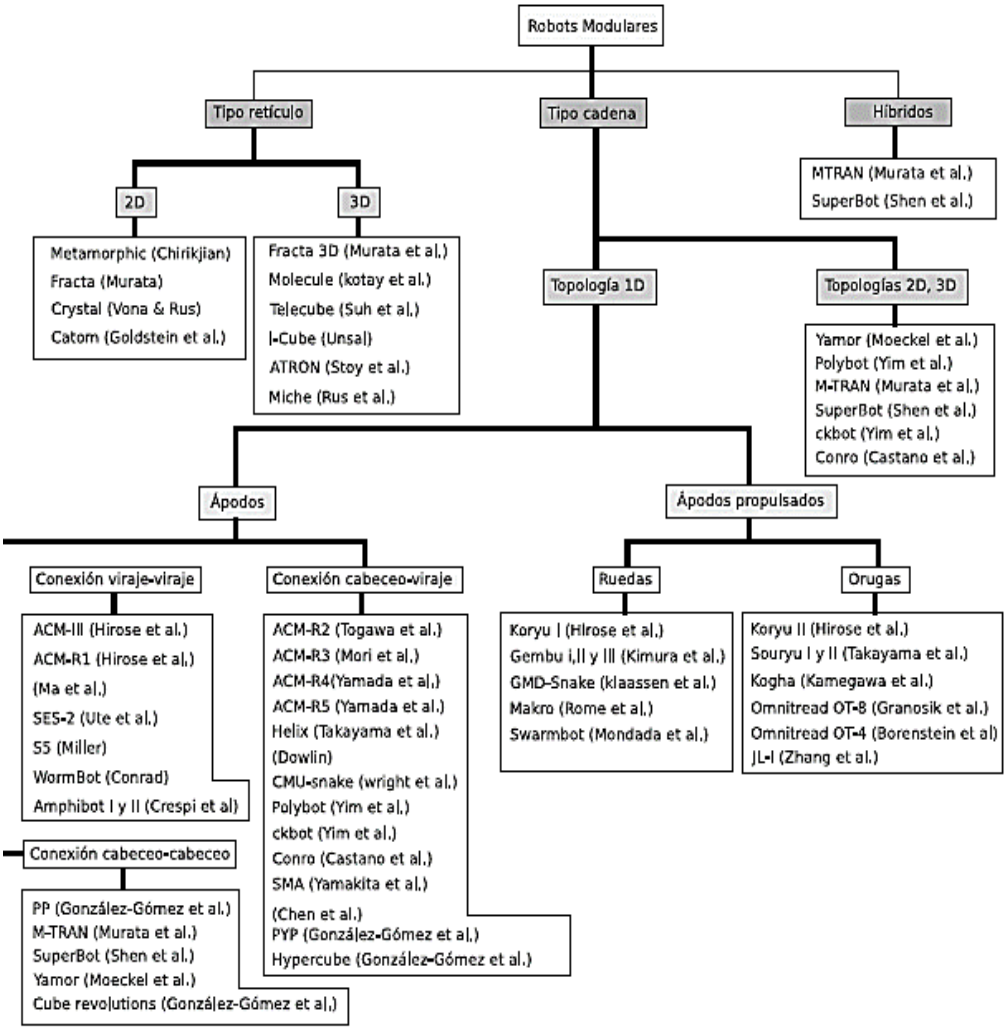


Figura 2. Clasificación de los robots modulares [21]

Según la clasificación propuesta en la Figura 2, el primer nivel de los robots modulares se encuentra el tipo retículo, tipo cadena e híbrido. Los robots modulares tipo retículo son aquellos que están formados por módulos que se comportan como átomos, los cuales se pueden unir o separar para formar estructuras, y pueden cambiar de configuración o funcionalidad, aunque no se pueden desplazar. Según el tipo de estructuras que puedan formar, se clasifican en 2D y 3D. [22]

La otra clase son los tipo cadena, que son lo que están formados por la unión de módulos en serie [23], y los robots híbridos son aquellos que tienen características de los robots tipo retículo y tipo cadena. Los robots tipo cadena se clasifican en topologías 1D, 2D y 3D. Las topologías 1D más usadas pueden ser gusanos, serpientes, brazos, piernas, entre otras, y su nombre se debe a que su estructura

es en una sola dimensión, con aplicaciones conocidas como para crear una cadena de módulos y escalar obstáculos. [23]

De acuerdo a su forma de desplazarse, estos robots tipo cadena se pueden clasificar en robots ápodos y robots ápodos auto-propulsados. Si el robot se desplaza mediante movimientos sincronizados de sus módulos, como se muestra en la Figura 3, el robot se clasifica como ápodo, pero si el desplazamiento se realiza por medio de elementos auxiliares como ruedas, orugas u otro medio, entonces el robot se clasifica como robot ápodo auto-propulsado.

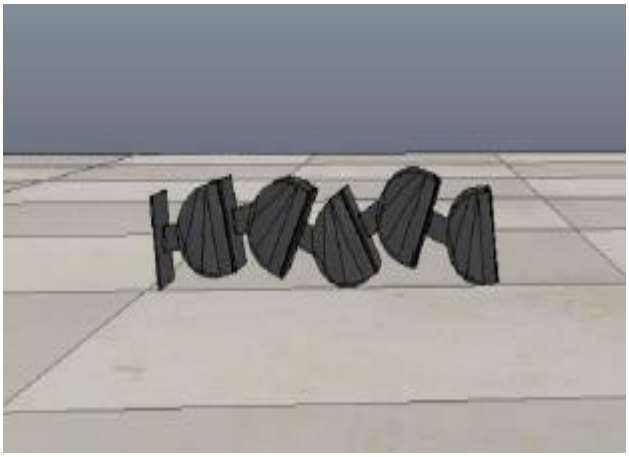


Figura 3. Robot tipo cadena, Topología 1D. Robot Apodo [24]

Los robots ápodos se pueden clasificar también de acuerdo al tipo de conexión entre sus módulos, lo que define el movimiento sobre la orientación del eje de rotación de cada módulo a saber: cabeceo-cabeceo, viraje-viraje y cabeceo-viraje.

2.1.5. Conexiones y Movimientos de los Robots Modulares

Como lo menciona Lozada [25] , los movimientos básicos que se pueden identificar y se han generalizado en la diferente gama de robots modulares se enmarcan en cinco: en línea recta (adelante y atrás), describiendo un arco circular, desplazamiento lateral, rotación y rodar.

Se puede llegar a distinguir entre los sistemas modulares los que se acoplan entre si mecánicamente, en donde para conseguir la locomoción, un robot modular ha de tener como mínimo dos módulos. Existen varias configuraciones básicas mínimas que se obtienen utilizando módulos de un grado de libertad.

Destacan los trabajos conocidos como Polybot, M-TRAN y CONRO. [25]

2.1.5.1. Configuración Pitch-Pitch

La configuración Pitch – Pitch, está compuesta por dos módulos que pueden moverse en línea recta, y se pueden desplazar hacia adelante o hacia detrás. Como se puede apreciar en la Figura 4.

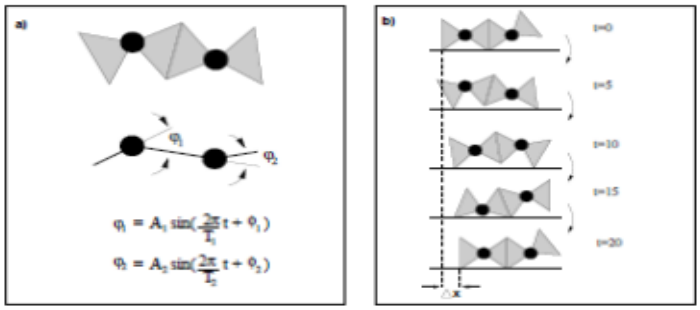


Figura 4. Representación gráfica de la configuración PP. [25]

La unión también llamada cabeceo-cabeceo, describe un movimiento en una dimensión, siendo este alrededor de un eje transversal en el mismo plano [24], como se ilustra en la Figura 5.

Figura 5. Configuración de pitch-pitch y módulo con ángulo de vástago. [24]

2.1.5.2. Configuración Yaw-Yaw

La configuración Yaw-Yaw, está compuesta por dos módulos que describe un movimiento alrededor de un eje vertical [24], como se aprecia en la Figura 6 es una unión conocida como tipo viraje-viraje que permite moverse como serpientes (dos dimensiones, adelante-atrás e izquierda-derecha).

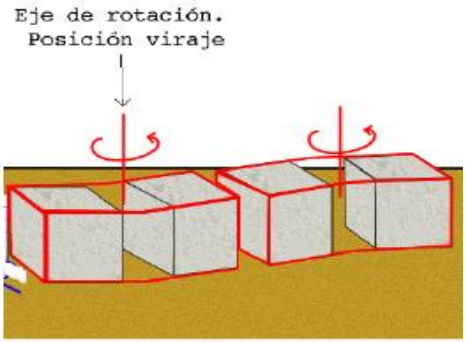


Figura 6. Diagrama de Conexión Viraje -Viraje

2.1.5.3. Configuración Pitch-Yaw

La configuración Pitch-Yaw, está compuesta por dos módulos, con la combinación de movimientos descritos anteriormente, los movimientos se obtienen dependiendo de la posición del módulo en el robot y la fase [24]. Como se aprecia en la Figura 7, mediante una unión tipo cabeceo-viraje es posible lograr movimientos en tres dimensiones.

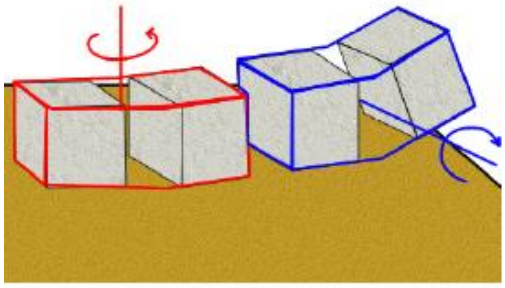


Figura 7. Diagrama de Conexión Cabeceo -Viraje

2.1.5.4. Configuración Pitch-Yaw- Pitch

Esta configuración que describe un movimiento en 3D, está compuesta por tres módulos: los dos de los extremos se mueven en el eje vertical y el central lo hace horizontalmente (como se aprecia en la Figura 8). Los movimientos que consiguen son:

En línea recta, hacia adelante o hacia atrás.

- Giros circulares.
- Movimiento lateral.
- Rotación paralela al suelo.
- Rodar sobre sí mismo.

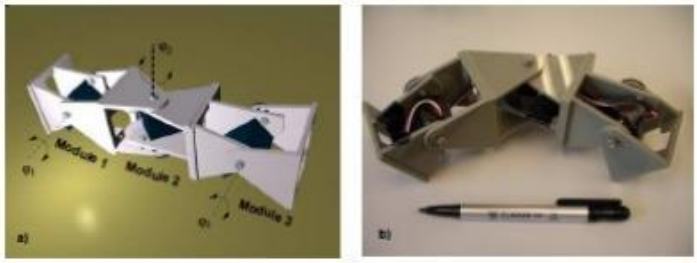


Figura 8. Representación gráfica de la configuración PYP. [25]

2.1.5.5. Configuración en Estrella

Está compuesta por tres módulos puestos en forma de estrella y que puede moverse línea recta en tres direcciones y rotar en paralelo al suelo, como se aprecia en la Figura 9.

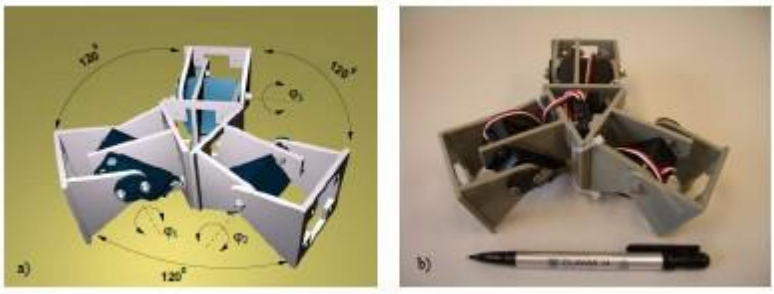


Figura 9. Representación gráfica de la configuración en estrella. [25]

2.1.6. Tipos de Mecanismos de Acople en los Sistemas Robóticos Modulares

El mecanismo de acople debe ser simple, en razón a que los módulos no se deben retraer mientras que los mecanismos de conexión estén activados, a menos que ese sea el propósito. Es sumamente importante el peso que la unión debe soportar para que sea seguro, puesto que no se tiene en cuenta, es posible que no se puede ejecutar cualquier arquitectura.

Hay dos clases de mecanismo de unión los cuales son:

- **Conexión Magnética:** Este tipo de conexión es simple y se puede controlar fácilmente de manera electrónica. La conexión magnética es ideal ya que auto alinea los imanes mediante la fuerza magnética y la conexión entre los módulos es rígida. Cuando se utilizan dos imanes permanentes para el acoplamiento, es difícil desconectar los dos módulos. Por lo tanto, para una fácil desconexión, se utiliza un imán permanente para un lado de un conector y un electroimán para el otro lado. De esta forma, los módulos se pueden conectar o desconectar. En la Figura 10 EP-Face connector es un conector novedoso para robots modulares híbridos tipo cadena de alta resistencia, compacto, rápido, eficiente en el consumo de energía y resistente a los errores de posición. El conector consiste en una matriz de imanes incrustados en una cara plana. Los imanes son de estado sólido, pueden activarse y apagarse, requieren alimentación solo cuando cambian de estado. [26]

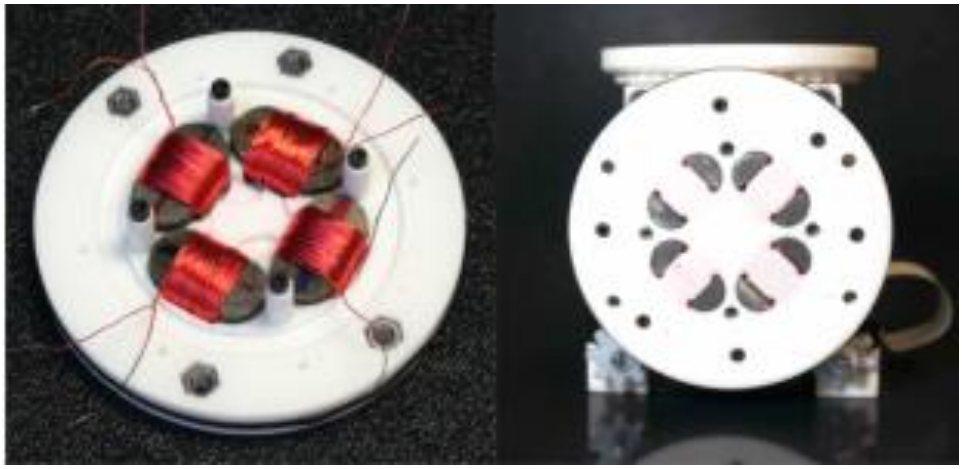


Figura 10. Foto EP-Face Connector [26]

- **Conexión mecánica:** Los mecanismos de conexión de este tipo no son muy utilizados debido a la alta complejidad de unión, ya que se debe disponer de una perfecta alineación del módulo con el que se desea conectar. Igualmente la composición física de estos conectores requiere de mayor complejidad

- **Conexión auto-soldable:** Es un método de conexión fuerte, liviano y de estado sólido basado en el calentamiento de una aleación de bajo punto de fusión para formar conexiones soldadas reversibles. No se requiere manipulación externa para formar o romper conexiones entre conectores adyacentes, lo que hace que este método sea adecuado para sistemas reconfigurables, como robots modulares de autoconfiguración. En la Figura 11 se aprecia un ejemplo del dispositivo. [27]

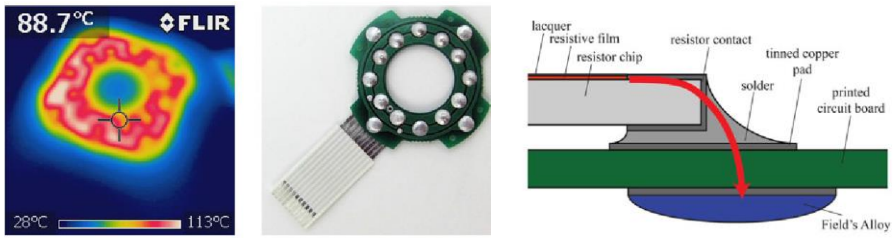


Figura 11. Conectores auto soldables para robots modulares [27]

2.1.7. Categorías de las Arquitecturas

Hay varias formas de categorizar sistemas robóticos MSR. Uno se basa en la regularidad de las ubicaciones para adjuntar; celosía vs cadena vs móvil, y otra se basa en los métodos de moverse entre esas ubicaciones; estocástico vs determinista. [16]

- **Arquitectura celosía** como se muestra en la Figura 12, tienen sus unidades de conexión para sus interfaces en los puntos en células virtuales de alguna cuadrícula regular. Esta red de puntos de conexión se puede comparar con los átomos en un cristal y la red de ese cristal. Por lo tanto, las características cinemáticas de los robots de celosía se pueden caracterizar por sus grupos de desplazamiento cristalográficos correspondientes. Por lo general pocas unidades son suficientes para llevar a cabo una etapa de reconfiguración. La arquitectura de celosía permite un diseño mecánico más simple y una planificación representación computacional y reconfiguración más simple que se puede escalar más fácilmente a sistemas complejos.

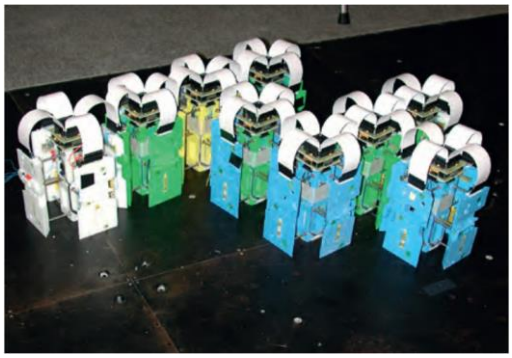


Figura 12. Representación gráfica celosía [16]

- **Arquitectura de la cadena:** Como se muestra en la Figura 13, un sistema MSR basado en cadena consiste en módulos dispuestos en grupos de cadenas en serie conectadas, formando estructuras de árbol y bucle. Dado que estos módulos están típicamente dispuestos en un punto arbitrario en el espacio, la coordinación de una reconfiguración es compleja. En particular, la cinemática directa e inversa, la planificación del movimiento y la detección

de colisiones son problemas que no se adaptan bien a medida que aumenta el número de módulos. [16]

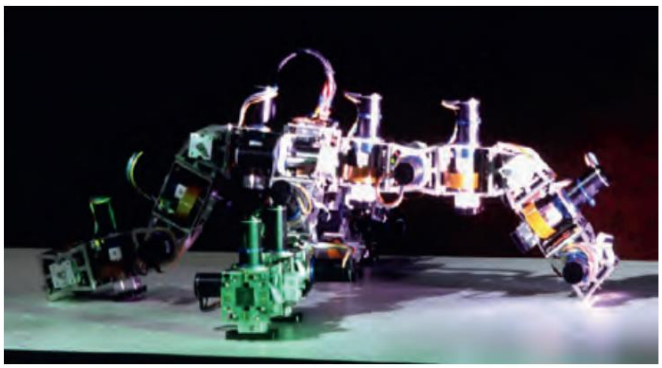


Figura 13. Representación gráfica cadena [16]

- **Móvil**: la clase de reconfiguración móvil se produce con módulos que se mueven en el entorno desconectado de otros módulos. Cuando se unen, pueden terminar en cadenas o en un enrejado. Entre los ejemplos de dispositivos de reconfiguración móvil se incluyen varios robots con ruedas que circulan y se unen para formar trenes, módulos que flotan en un espacio líquido o exterior y se acoplan con otros módulos. [16]

- **Determinista:** En sistemas MSR deterministas, los módulos se mueven o son manipulados directamente de una posición a otra en el enrejado o cadena Las posiciones de cada módulo en el sistema son conocidas en todo momento. Se determina la cantidad de tiempo que le toma a un sistema cambiar de una configuración a otra. El mecanismo de reconfiguración de un módulo requiere una estructura de control que le permita coordinar y realizar secuencias de reconfiguración con sus vecinos. [16]

- **Estocástica** En un sistema estocástico, los módulos se mueven aleatoriamente en un entorno 2D o 3D y forman estructuras uniéndose a un sustrato y / u otros módulos. Los módulos se mueven en el entorno en un estado pasivo. Una vez que un módulo entra en contacto con el sustrato u otro módulo, toma una decisión sobre si se unirá a la estructura o rechazará un vínculo. El tiempo que tarda el sistema en alcanzar una configuración deseada es limitado de forma probabilística. La dependencia de las fuerzas ambientales permite que la actuación mecánica se simplifique, ya que solo se requiere la activación interna del módulo. Un ejemplo se muestra en la Figura 14.

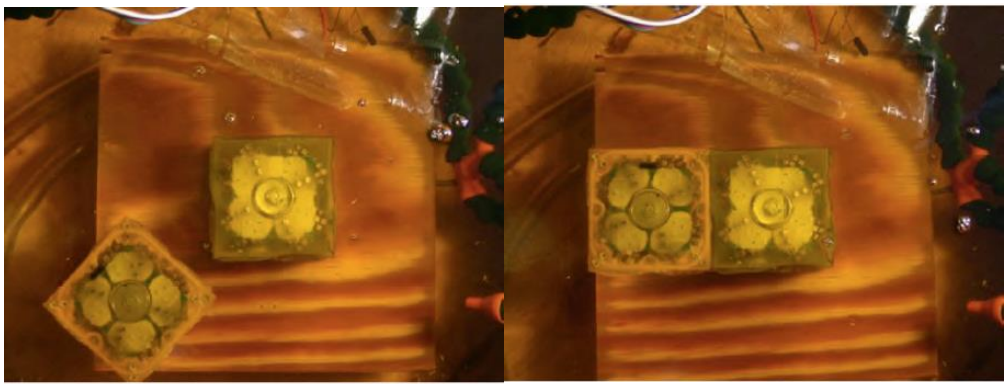


Figura 14. Representación gráfica estocástica [16]

- **Los sistemas de robots modulares homogéneos** tienen muchos módulos del mismo diseño que forman una estructura adecuada para llevar a cabo la tarea requerida. Una ventaja sobre otros sistemas es que son fáciles de escalar en tamaño (y, posiblemente, función), mediante la adición de más unidades. Una desventaja comúnmente descrita es en razón a límites de la funcionalidad, estos sistemas a menudo requieren más módulos para lograr una función dada, que los sistemas heterogéneos.

- **Los sistemas de robots modulares heterogéneos** tienen diferentes módulos, cada uno de los cuales hacen funciones especializadas, formando una estructura adecuada para realizar una tarea. Una ventaja es la compacidad y la versatilidad para diseñar y poner unidades para realizar cualquier tarea. Una desventaja comúnmente descrito es un aumento de la complejidad de los métodos de diseño, fabricación, y de simulación.

  Existen otros sistemas robóticos modulares que no son auto-reconfigurable, y por lo tanto no pertenecen formalmente a esta familia de robots aunque pueden tener apariencia similar. Por ejemplo, los sistemas de auto-montaje pueden estar compuestos de varios módulos, pero no pueden controlar dinámicamente su forma de destino. Del mismo modo, la robótica tensegríticas pueden estar compuestas por varios módulos intercambiables, pero no puede auto-reconfigure. [28]

2.1.8. Campos de Aplicación para Robots Modulares Autoconfigurables

Los campos de aplicación deseados para los robots modulares auto configurables (Figura 15) son aquellos en los que se su ambiente de interacción es estático, sin embargo se proyecta desenvolver el tema en campos donde no se conoce previamente el lugar. Por ejemplo escenarios de exploración, búsqueda y rescate.

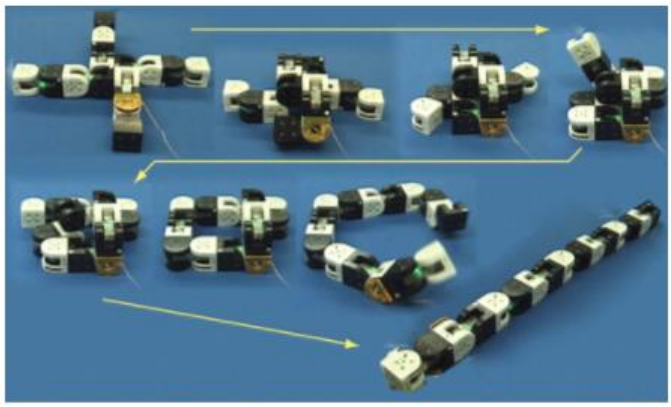


Figura 15. Configuración automática de robot modular [29]

2.1.8.1. Búsqueda y rescate

Los robots modulares autoconfigurables son proyectados para resolver en situaciones en las que el escenario de desplazamiento no es predecible y el robot debería adaptar su morfología para poderse desplazar en el entorno. Por ejemplo, como se aprecia en la Figura 16 en un temblor puede haber muchos escombros a través de los cuales el robot debería moverse, esta tarea normalmente requeriría varios tipos de robots. En estas condiciones los robots modulares serían ideales, pues mediante un solo robot se podrían atravesar todos los obstáculos mediante la adaptación morfológica automática, en obstáculos estrechos se podrían transformar en una serpiente, en áreas planas que requieren velocidad en una rueda y en montones de escombros en un hexápodo. Por lo tanto, el escenario de búsqueda y rescate se vuelve la proyección ideal donde este tipo de robot mejor se desenvolvería dado que el medio es impredecible. [30]

Figura 16. Imagen de Búsqueda y rescate [30]

2.1.8.2. Exploración

Otro campo de proyección para este tipo robots es la exploración. Desde la exploración de planetas (ver Figura 17) hasta el mar profundo, cualquier escenario en el que no se puede predecir los obstáculos, se vuelve atractivo diseñar con la expectativa para los robots modulares autoconfigurables. En esta clase de entornos es donde se destacarían los robots modulares, puesto que podrían mostrar cada una de sus ventajas en la adaptabilidad morfológica, mientras que otros robots no podrían realizar la actividad debido a que se requerirían varios con una morfología diferente para poder desarrollar la actividad.

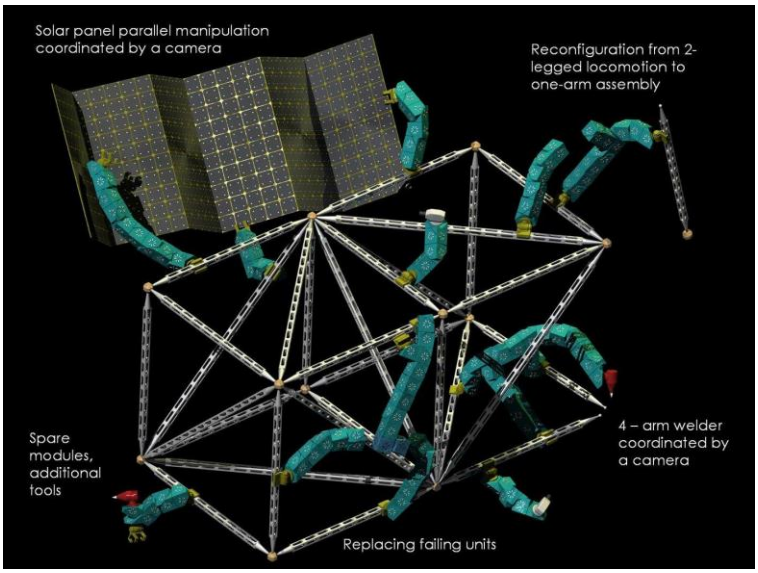


Figura 17. SPACE MOLECUBES: Aplicación Espacial para Robots Modulares Autoconfigurables [31]

2.1.9. Sistemas de Control para Sistemas Robóticos Modulares

Para el control de sistemas modulares con enfoque en considerar los grados de libertad que se generen para unidades entrelazadas hay 2 estrategias de control, el centralizado y el descentralizado.

- **Centralizado:** Mediante esta estrategia de control cada módulo recibe órdenes de un sistema centralizado, el cual puede observar de manera global el comportamiento de cada módulo, tomar la decisión y enviarla a cada robot. Sin embargo, en aplicaciones en que la comunicación del sistema central con cada módulo se dificulta no se puede utilizar. En este sistema se limita la autonomía de los módulos, así como se presentan limitaciones para

determinar de manera exacta en que lugar de la estructura queda un módulo luego de una reconfiguración. [32]

- **Descentralizado:** En esta estrategia de control los módulos son completamente autónomos en la capacidad de procesamiento ya que cada módulo tiene su propio algoritmo, requiere de comunicación entre todos los módulos para realizar las órdenes dadas. Este tipo de controladores funciona haciendo las tareas de las que se encargaba el controlador central de manera distribuida en cada uno de los módulos que forman la estructura del robot. Una forma de lograr esto es introduciendo la idea de un coordinador que coordine los demás módulos mientras que cada uno ejecuta tareas locales. [33]

2.1.10. Planificación de Trayectorias en la Robótica Móvil

En el campo de la Robótica, la intersección de las referencias espaciales temporales que debe seguir un vehículo hasta la referencia definitiva se denomina trayectoria, término que se emplea para referirse al camino a recorrer el robot durante la ejecución de una rutina de desplazamiento.

2.1.10.1. Métodos de Planificación de Trayectorias en Ambientes Estáticos

Los métodos de planificación se fundamentan en una fase de construcción de algún tipo de grafo o conjunto de rectas sobre el espacio libre de obstáculos, desde la posición inicial hasta una posición final, en razón a la información que poseen del entorno, para posteriormente hallar la mejor trayectoria empleando un algoritmo de búsqueda eficiente de acuerdo a la aplicación específica. Existen tres métodos de planificación de trayectorias en ambientes los cuales sus obstáculos de mantienen fijos o estáticos durante el desplazamiento del robot móvil, a saber: Grafos de Visibilidad, Descomposición de Celdas y Campos Potenciales, los cuales se explican a continuación:

- Planificación Basada en Grafos de Visibilidad

Los grafos de visibilidad, como el mostrado en Figura 18 proporcionan un enfoque geométrico útil para resolver el problema de la planificación. Supone un entorno bidimensional en el cual los obstáculos están modelados mediante polígonos. [34] Para la generación del grafo este método introduce el concepto de visibilidad, según el cual define dos puntos del entorno como visibles si y solo si se pueden unir mediante un segmento rectilíneo que no intercepte ningún obstáculo (si dicho

segmento resulta tangencial a algún obstáculo se consideran los puntos afectados como visibles)

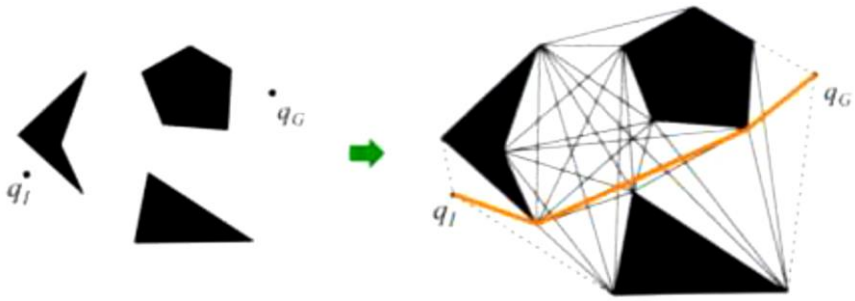


Figura 18. Ejemplo aplicación metodología de planificación basada en grafos de visibilidad [34]

- Planificación Basada en Descomposición de Celdas

Este tipo de métodos se fundamenta en una descomposición en celdas del espacio libre. [34] Así, la búsqueda de una ruta desde una postura inicial qa hasta otra final qf, consiste en encontrar una sucesión de celdas que no presente discontinuidades, tal que la primera de ellas contenga a qa y la última a qf.

Al contrario que los métodos anteriormente mencionados, no encuentra una serie de segmentos que modele la ruta, sino una sucesión de celdas; por ello, se hace necesario un segundo paso de construcción de un grafo de conectividad, encargado de definir la ruta.

Para la planificación según el método de descomposición en celdas, se precisa la resolución de dos problemas: la descomposición del espacio libre en celdas y la construcción de un grafo de conectividad como se muestra en Figura *19*.

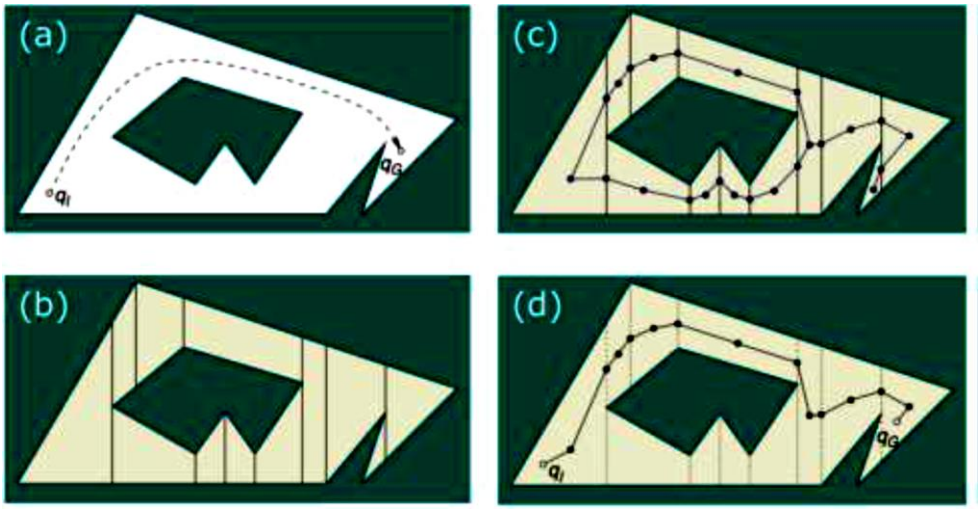


Figura 19. Ejemplo aplicación metodología de planificación basada en descomposición de celdas [34]

- Planificación Basada en Campos Potenciales

Los métodos basados en campos potenciales poseen una concepción totalmente distinta al estar basados en técnicas reactivas de navegación. El ámbito de uso de esta técnica se centra en la planificación local en entornos desconocidos, como puede ser el sorteo en tiempo real de obstáculos o de los que no se tiene constancia. La teoría de campos potenciales considera al robot como una partícula bajo la influencia de un campo potencial artificial, cuyas variaciones modelan el espacio libre, un ejemplo de estos campos se muestra a continuación en Figura *20*.

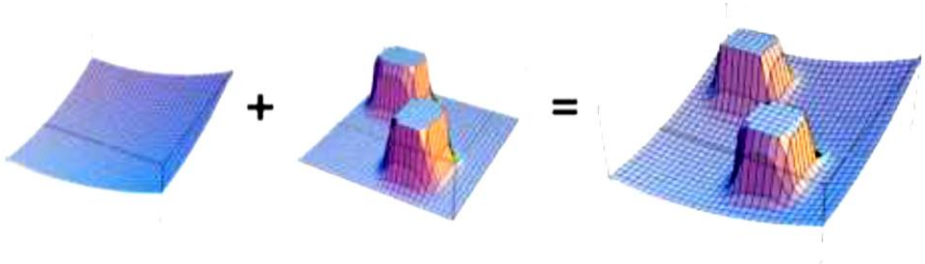


Figura 20. Ejemplo aplicación metodología de planificación basada en campos potenciales [34]

2.1.10.2. Algoritmo de Búsqueda A* (A-Star)

El algoritmo A* pertenece a los algoritmos de búsqueda en grafos. Se parte de la base de dividir el espacio de trabajo en una rejilla rectangular para obtener celdas que permitan trazar una trayectoria a medida que no se encuentren obstáculos, donde bajo unas determinadas condiciones se encuentra el camino de menor coste o el más corto entre un nodo origen y un nodo objetivo.

Se emplea para encontrar un camino de celdas libres desde las condiciones actuales $Ninicial$ hasta el objetivo indicado $Nobjetivo$. Este algoritmo explora iterativamente las celdas o nodos, abriendo diferentes caminos desde el nodo inicial. Para cada nodo visitado N, el algoritmo produce uno o varios caminos conectándolo con $Ninicial$ pero solo memoriza la representación del camino cuyo coste asociado es mínimo. En cada iteración de este algoritmo, el conjunto de nodos expandidos forma un árbol de expansión con origen en $Ninicia$. Cada nodo visitado nace de su predecesor, por lo que para obtener la trayectoria desde un nodo N hasta el inicial, basta con recorrer el árbol hacia su origen. [35]

El algoritmo A* asigna a cada nodo explorado una función de costo. Los nodos explorados se insertan en una lista denominada *Abiertos*, en la que se ordenan de menor a mayor costo. En cada iteración se analiza uno de los nodos de *Abiertos*, el de menor costo hacia el nodo objetivo. [35]

Mediante el operador de expansión, se van abriendo sus casillas vecinas, para cada celda abierta se comprueba que está libre de colisión, que es alcanzable en ese nivel temporal y se calcula el coste asociado. Si está libre de colisión la celda aparecerá marcada en la malla como vacía, en cuyo caso se estima el coste asociado mediante una función de coste.

Básicamente, se trata de una estimación heurística del coste asociado. La función consta de dos términos, donde g(N) representa el coste asociado al camino desde el nodo inicial $Ninicial$, hasta el nodo actual N, $h(N)$, es una estimación heurística del coste del camino de menor coste entre N y $Nobjetivo$ que incluye el coste asociado a la seguridad, la ecuación es la siguiente:

Ecuación 1:

$$f(N) = g(N) + h(N)$$

Como se puede apreciar en la Figura 21, la ejecución del algoritmo de búsqueda A*; donde inicialmente se exploran los nodos adyacentes al nodo inicial (en verde), con el menor costo respecto al objetivo (en amarillo) y se marcan como candidatos de trayectoria (en rojo). Los nodos por explorar (en azul) serán posteriormente evaluados mientras se mantienen en memoria los costos asociados para llegar al objetivo.

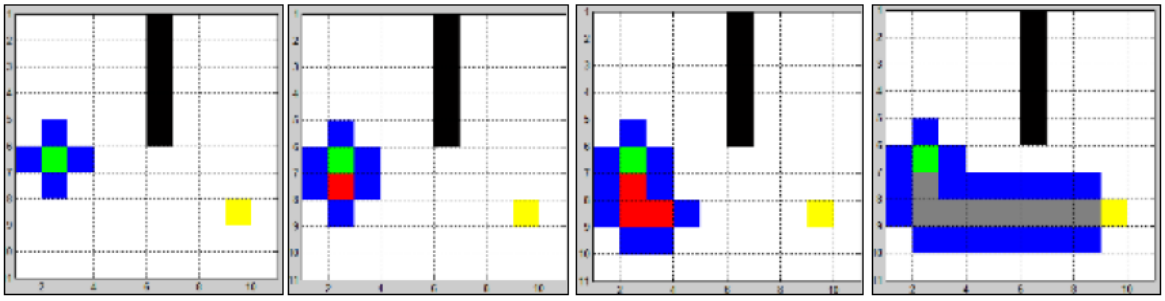


Figura 21. Exploración y Generación de Trayectoria mediante el Algoritmo A* [36]

2.1.11. Simulación en la Robótica

La simulación es la imitación de la operación de un proceso o sistema del mundo real en el tiempo. Para simular algo se requiere que se desarrolle un modelo; este modelo representa las características, comportamientos y funciones clave del sistema o proceso físico seleccionado. Los experimentos en computadora se usan para estudiar modelos de simulación, el modelo representa el sistema en sí, mientras que la simulación representa el funcionamiento del sistema a lo largo del tiempo. [37]

La simulación también se usa con modelos científicos de sistemas naturales o sistemas humanos para obtener información sobre su funcionamiento [38], como también en economía. La simulación se puede usar para mostrar los efectos reales eventuales de condiciones y cursos de acción alternativos, también se usa cuando el sistema real no puede ser activado, porque puede no ser accesible, o puede ser peligroso o inaceptable, o está siendo diseñado, pero aún no construido. [39]

2.1.12. Software de Simulación Webots

Webots es un software de simulación profesional que permite programar y simular en un entorno 3D robots móviles y manipuladores robóticos, su principal herramienta es ODE (Open Dynamics Engine) para detectar colisiones y simular dinámicas rígidas de los cuerpos, la biblioteca ODE permite simular con precisión las propiedades físicas de los objetos y del entorno tales como la gravedad, la velocidad, la inercia y la fricción, teniendo en cuenta propiedades intrínsecas que se pueden establecer como la masa del objeto, textura, entre otras, que buscan darle realismo a los movimientos y condiciones del sistema.

Dentro de sus principales características, Webots permite importar figuras, piezas o robots completos en formato VRML a los cuales se les puede aginar todo tipo de sensores y actuadores como lo son motores, sensores de posicionamiento como GPS, giroscopios, como también dispositivos de comunicación para la recepción y envío de datos, entre otros; Elementos que Webots permite programar en los lenguajes C, C++, Java, Python y Matlab, cuyos controladores se pueden trasladar a robots reales, para el comportamiento deseado visto en el entorno de simulación. [36]

Como se puede apreciar en la Figura 22, Webots trabaja con una jerarquía de nodos [40], para el mundo virtual y la programación típica de los mundos virtuales VRML, además de otros propios para ir dándole propiedades y capacidades para crear sistemas más complejos entre los que están:

- Nodo Worlinfo: En este se establece las propiedades del ambiente 3D junto con la información de éste, como ¿quién lo creó? y la fecha en que se hizo, además de definir propiedades físicas como la gravedad en los ejes de coordenadas, el paso del tiempo y la velocidad de simulación.
- Nodo Background: Para establecer el color de fondo en el subnodo skyColor.
- Nodo Viewpoint: Permite orientar la cámara que interactúa con el mundo permitiendo rotar, hacer zoom y trasladar la cámara en sus respectivos subnodos.

- Nodo Pointlight: Dentro de sus subnodos permite cambiar la luminosidad, el color, intensidad, la posición y número de fuentes de luz, así como manipular las sombras generadas.
- Nodo RectangleArena: Este nodo permite cambiar las propiedades de la superficie donde actuará al robot, el tipo de piso, las texturas y materiales, así como la orientación de este. Por defecto el mundo de Webots trae un piso cuadriculado al que se le puede modificar el tamaño de cuadro y el espacio que ocupa en el mundo.
- Nodo World: Son nodos a los que se le asigna una figura o robot y un subnodo de transformación en el que se le asigna la apariencia y orientación del robot. Este nodo se utiliza como base para creación del robot ya que en este se define el tipo pieza, asignándole propiedades mediante nodos más específicos como los son: el nodo Robot, generalmente asignado a todo un ensamble importado, al que se le pueden asignar todas las propiedades de un robot, como lo son el controlador que usa, si posee batería y cómo sería el consumo, además de los subnodos de las articulaciones y especificar las colisiones entre estas.
- Nodo Robot: Es usado como base para construir un robot, un brazo robótico articulado, un robot humanoide, un robot proporcionado con ruedas. [41] El cual se le encarga las funciones a realizar mediante un controlador, maneja diferentes lenguajes de programación, como C, C++, Phyton o Matlab.
- El Nodo Solid hace referencia a un objeto y sus propiedades físicas como dimensiones, masa y material.

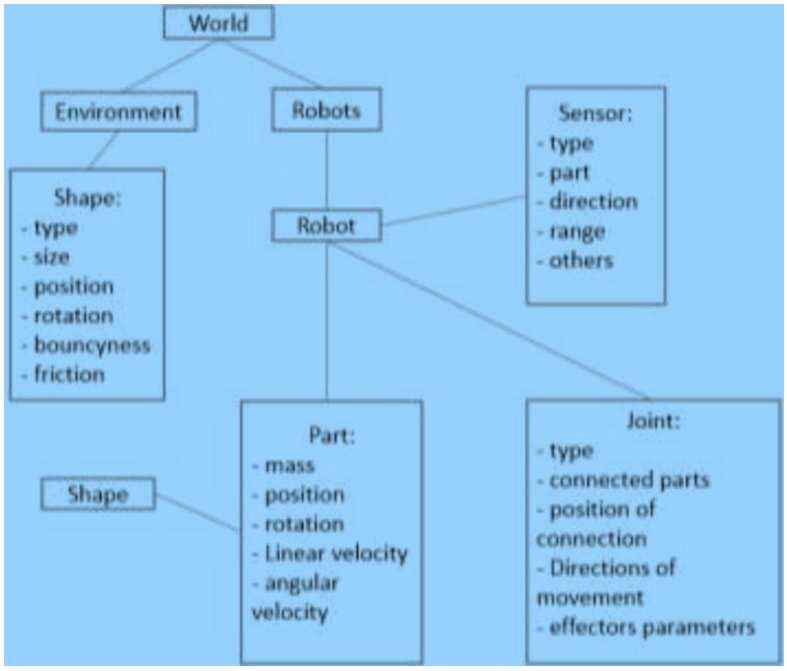


Figura 22. Jerarquía Nodos en Software Webots [40]

2.2. Antecedentes del Proyecto

Las raíces del concepto de robots modulares auto-reconfigurable se remontan a los que cambiaban del efector final y automático de herramientas "cambio rápido" en control numérico y centros de mecanizado de un ordenador en la década de 1970. Aquí, los módulos especiales, cada uno con un mecanismo de conexión común podrían ser intercambiados de forma automática en el extremo de un brazo robótico. Sin embargo, tomando el concepto básico del mecanismo de conexión común y su aplicación a todo el robot fue presentado por Toshio Fukuda con el CEBOT (abreviatura de robot móvil) a finales de 1980. [11]

La década de 1990 vio un mayor desarrollo de Greg Chirikjian, Mark Yim, Joseph Michael, y Satoshi Murata. Chirikjian, Michael, y Murata desarrollando sistemas de reconfiguración de celosía y Yim desarrollando un sistema basado en la cadena. Aunque estos investigadores comenzaron con un énfasis de la ingeniería mecánica, diseño y construcción de módulos para luego desarrollar código para programarlos, el trabajo de Daniela Rus y Wei-min Shen desarrollado el hardware, pero tuvo un mayor impacto en los aspectos de la programación. Comenzaron una tendencia hacia algoritmos demostrables o distribuidos verificables para el control de un gran número de módulos. [8]

Una de las plataformas de hardware más interesantes recientemente ha sido la MTRAN II y III los sistemas desarrollados por Satoshi Murata et al. Este sistema es una cadena híbrido y sistema de celosía. Tiene la ventaja de ser capaz de realizar tareas más fácilmente como los sistemas de cadena, sin embargo, volver a configurar como un sistema de celosía. [2]

Más recientemente nuevos esfuerzos en el auto ensamblaje estocástico se hayan interpuesto por Hod Lipson y Eric Klavins. Un gran esfuerzo en la Universidad Carnegie Mellon dirigido por Seth Goldstein y Todd Mowry ha comenzado a mirar los problemas en el desarrollo de millones de módulos. [42]

Muchas de las tareas se han demostrado para ser alcanzable, especialmente con los módulos de reconfiguración en cadena. Esto demuestra la versatilidad de estos sistemas, sin embargo, las otros dos ventajas, robustez y bajo costo no se han demostrado. En general, los sistemas de prototipo desarrollado en los laboratorios han sido frágil y costosa como sería de esperar durante cualquier desarrollo inicial. [14]

Hay un número creciente de grupos de investigación que participan activamente en la investigación robótica modular. Hasta la fecha, cerca de 30 [43], como se ve en la Tabla 1.

Tabla 1 Sistemas físicos creados desde 1988. Elaboración Propia

Sistema	Clase, DOF	Autor	Año
CEBOT	Mobile	Fukuda et al. (Tsukuba)	1988
Polypod	chain, 2, 3D	Yim (Stanford)	1993
Metamorphic	lattice, 6, 2D	Chirikjian (Caltech)	1993
Fracta	lattice, 3 2D	Murata (MEL)	1994
Fractal Robots	lattice, 3D	Michael(UK)	1995
Tetrobot	chain, 1 3D	Hamline et al. (RPI)	1996
3D Fracta	lattice, 6 3D	Murata et al. (MEL)	1998
Molecule	lattice, 4 3D	Kotay & Rus (Dartmouth)	1998
CONRO	chain, 2 3D	Will & Shen (USC/ISI)	1998
PolyBot	chain, 1 3D	Yim et al. (PARC)	1998
TeleCube	lattice, 6 3D	Suh et al., (PARC)	1998
Vertical	lattice, 2D	Hosakawa et al., (Riken)	1998
Crystalline	lattice, 4 2D	Vona & Rus, (Dartmouth)	1999
I-Cube	lattice, 3D	Unsal, (CMU)	1999
Micro Unit	lattice, 2 2D	Murata et al.(AIST)	1999
M-TRAN I	hybrid, 2 3D	Murata et al.(AIST)	1999
Pneumatic	lattice, 2D	Inou et al., (TiTech)	2002
Uni Rover	mobile, 2 2D	Hirose et al., (TiTech)	2002
M-TRAN II	hybrid, 2 3D	Murata et al., (AIST)	2002
S-bot	mobile, 3 2D	Mondada et al., (EPFL)	2003
Stochastic	lattice, 0 3D	White, Kopanski, Lipson (Cornell)	2004
Superbot	hybrid, 3 3D	Shen et al., (USC/ISI)	2004
Y1 Modules	chain, 1 3D	Gonzalez-Gomez et al., (UAM)	2004
M-TRAN III	hybrid, 2 3D	Kurokawa et al., (AIST)	2005
AMOEBA-I	Mobile, 7 3D	Liu JG et al., (SIA)	2005
Catom	lattice, 0 2D	Goldstein et al., (CMU)	2005
Stochastic-3D	lattice, 0 3D	White, Zykov, Lipson (Cornell)	2005
Molecubes	hybrid, 1 3D	Zykov, Mytilinaios, Lipson (Cornell)	2005
Prog. parts	lattice, 0 2D	Klavins, (U. Washington)	2005
Atron	lattice, 1 3D	Jonrgensen et al., (U.S Denmark)	2006
Miche	lattice, 0 3D	Rus et al., (MIT)	2006
YAMOR	chain, 0 1D	Moeckel et al. (EPFL)	2006
GZ-I Modules	chain, 1 3D	Zhang & Gonzalez-Gomez (U. Hamburg, UAM)	2006

The Distributed Flight Array	lattice, 6 3D	Oung & D'Andrea (ETH Zurich)	2008
Evolve	chain, 2 3D	Chang Fanxi, Francis (NUS)	2008
EM-Cube	Lattice, 2 2D	An, (Dran Computer Science Lab)	2008
Roombots	Hybrid, 3 3D	Sproewitz, Moeckel, Ijspeert, Biorobotics Laboratory, (EPFL)	2009
Programmable Matter by Folding	Sheet, 3D	Wood, Rus, Demaine et al., (Harvard & MIT)	2010
Sambot	Hybrid, 3D	HY Li, HX Wei, TM Wang et al., (Beihang University)	2010
Moteins	Chain, 1 3D	Center for Bits and Atoms, (MIT)	2011
ModRED	Chain, 4 3D	C-MANTIC Lab, (UNO/UNL)	2011
Programmable Smart Sheet	Sheet, 3D	An & Rus, (MIT)	2011
SMORES	Hybrid, 4, 3D	Davey, Kwok, Yim (UNSW, UPenn)	2012
Symbrion	Hybrid, 3D	EU Projects Symbrion and Replicator	2013
Cubelets	Hybrid, 3D	Correll & Wailes	2013
ReBiS - Re-configurable Bipedal Snake	Chain, 1, 3D	Rohan, Ajinkya, Sachin, S. Chiddarwar, K. Bhurchandi (VNIT, Nagpur)	2014
UBOT	Chain, 2, 3D	Wang, Jin, Zhu	2015
Modred II	Chain, 4 3D	C-MANTIC Lab, (UNO/UNL)	2015

2.2.1. Sistemas Robóticos Modulares Autoconfigurables Representativos

2.2.1.1. Polypod (1995)

Polypod es el nombre del sistema de robot modular reconfigurable diseñado e implementado por Mark Yim, el cual se encuentra explicado en su tesis doctoral en 1994 [22]. Como se aprecia en la Figura 23, consta de dos módulos, por lo que, por definición, no es exactamente modular por unidad, siendo la primer aproximación de reconfiguración a pesar que esta fuera asistida, sin embargo, virtualmente todos los beneficios son los mismos. Además, desde la mayor parte de la funcionalidad de Polypod se encuentra en uno de los dos módulos, tratando a Polypod como si fuera una unidad modular.

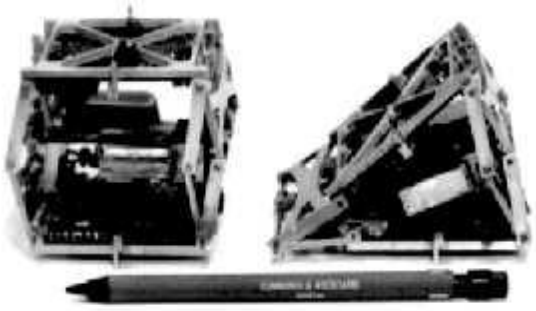


Figura 23. Segmento y nodo del Polypod [21]

2.2.1.2. Polybot G3 (2002)

Como se muestra en Figura 24, es un robot desarrollado en el centro de investigación "Xerox Palo Alto", grupo liderado por Mark Yim para el desarrollo del conocido Polybot [44]. Implementa una arquitectura software que interpreta una estructura multimaestro/multiesclavo en un entorno mulithilo con tres capas de protocolo de comunicaciones. Todo se implementa utilizado un protocolo de comunicación basado en CANbus que implica un bajo precio, múltiples fuentes, alta robustez, amplia aceptación, etc. Esta estructura multihilo permite un manejo eficiente de múltiples peticiones de hardware y computación en tiempo real. [25]

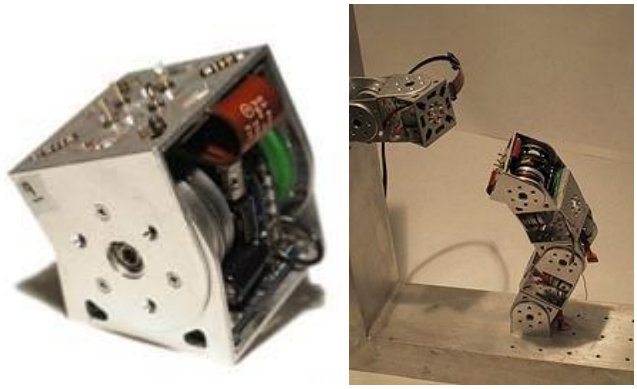


Figura 24. Polybot G3 Modular self-reconfigurable robot

Como se muestra en la Figura 25, las últimas actualizaciones de estos módulos son los **G1v5,** los cuales a pesar de no ser reconfigurables, están diseñados a partir de las lecciones aprendidas con todos los módulos Polybot de las generaciones anteriores. Para su manejo y programación han desarrollado el entorno Polykinectis. [45]

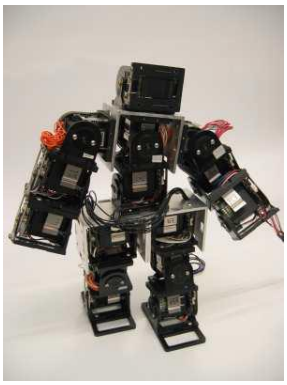


Figura 25. Polybot G1v5 Humanoide [21]

2.2.1.3. Ckbot (2006)

Como inspiración de la versión G1V5 (Figura 25), desde la Universidad de Pensilvania, el Modular Robotic Laboratory – ModLab desarrolló el módulo Ckbot, como se aprecia en la Figura 26, los cuales no son dinámicamente reconfigurables, pero permite crear diferentes tipos de configuraciones para explorar las capacidades de locomoción implementadas. [46]

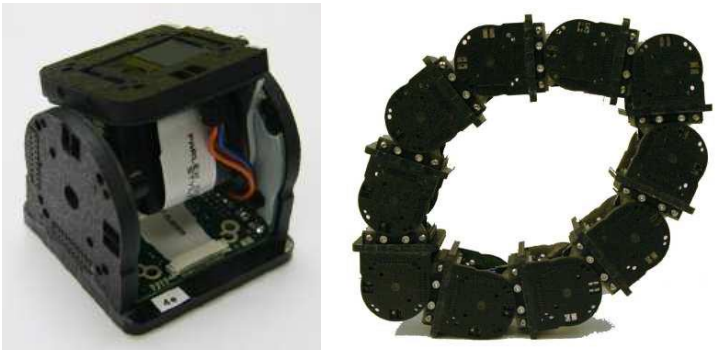


Figura 26. Ckbot – Configuración Tipo Rueda [21]

2.2.1.4. M-tran III (2005)

Por sus siglas *Modular Transformer*, desarrollado por el Instituto Nacional de Ciencias y Tecnologías Industriales Avanzadas de Japón (AIST) [47], la evolución tecnológica del módulo y sus desarrollos en aplicaciones de diferentes arquitecturas, lo convierten en uno de los más avanzados en la actualidad, en la Figura 27 se muestran los módulos desarrollados desde 1998 hasta la fecha.

Figura 27. M-Tran I, II y III

En su versión más actualizada como se muestra en la Figura 28, es un sistema de auto - reconfigurable tipo híbrido. Cada módulo es de dos tamaños del cubo (65 mm de lado), y tiene 2 DOF rotacional y 6 superficies planas para la conexión. Es el 3ª prototipo M - TRAN. En comparación con el anterior (M - TRAN II), la velocidad y la fiabilidad de la conexión se mejora en gran medida. Como un sistema de tipo de cadena, locomoción CPG (Pattern Generador Central) controlador en diversas formas ha sido demostrado por M- TRAN II. Como un sistema de tipo de celosía, que puede cambiar su configuración, por ejemplo, entre uno de 4 patas a una oruga. [48]

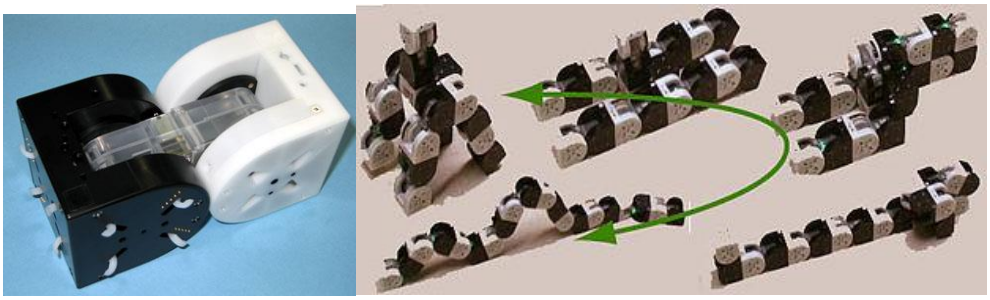


Figura 28. Metamorphosis by a self-reconfigurable robot, M-TRAN III [49]

2.2.1.5. CONRO (2000)

Es un robot desarrollado en la “University of Southern California” por ISI (Information Science Institute) [50], como se aprecia en la Figura 29, presenta un mecanismo de control distribuido inspirado en el concepto de hormonas de los sistemas biológicos. Se entiende por hormona un mensaje especial que provoca acciones diferentes en cada módulo, dependiendo del estado en el que se encuentre dicho módulo y el momento en el que se recibe el mensaje. Basado en la idea de “hormonas” se utilizan para coordinar movimientos y para reconfigurar el robot. Cada módulo del

robot dispone de un identificador ID y un conjunto de enlaces o canales de comunicación con el resto de módulos. [25]

Figura 29. CONRO [25]

2.2.1.6. Superbot (2005)

Es un robot modular de autoconfiguración elaborado por el Information Science Institute ISI, siendo uno de los más modernos al servir de referencia en su elaboración de los modelos anteriores como Conro, Polybot, MTRAN entre otros. Se basó su uso principalmente para aplicaciones espaciales [51], contó con la motivación y financiación de la NASA. La mecánica de los módulos de Superbot está inspirada en el MTRAN, con la implementación de un grado más de libertad, como se puede apreciar en la Figura 30 se constituye entre 8 a 10 módulos para su reconfiguración en la plataforma necesaria, tipo cadena o estructuras solidas de forma hibrida.

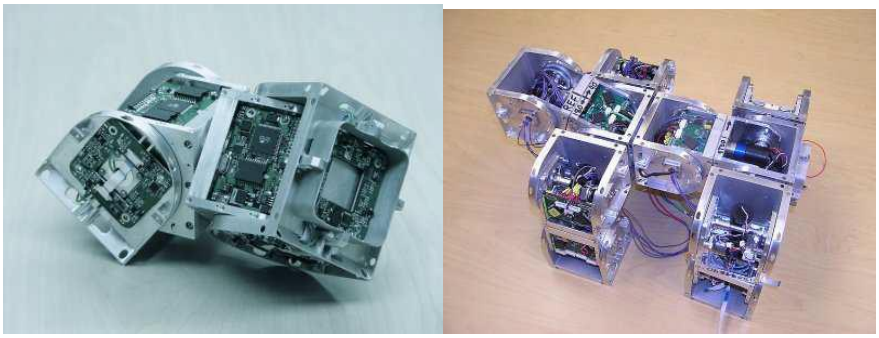


Figura 30. Módulo Superbot con configuración hibrida [51]

2.2.1.7. Amoeba-I (2005)

AMOEBA-I, uno de tres módulos reconfigurable robot móvil se desarrolló en Shenyang Instituto de Automatización (SIA), la Academia China de Ciencias (CAS) por Liu JG et al. [52] [53] AMOEBA-I tiene nueve tipos de configuraciones y alta movilidad en virtud del medio. 4 no estructurados de su plataforma se han

desarrollado y una serie de investigaciones se han llevado a cabo en su mecanismo de reconfiguración, configuraciones que no son isomorfas, la estabilidad de vuelco, y la planificación de reconfiguración. Los experimentos han demostrado que dicho tipo de estructura permite una buena movilidad y una alta flexibilidad a terreno desigual. Al ser hiper-redundante, modular y reconfigurable, AMEBA-I tiene muchas aplicaciones posibles, tales como Búsqueda y Rescate Urbano (USAR) y la exploración espacial.

2.2.1.8. La matriz de vuelo distribuido (2009)

Como se ve en la Figura 31, la matriz de vuelo distribuida es un robot modular que consiste en unidades de un solo rotor en forma hexagonal, que pueden tomar casi cualquier forma o modalidad. A pesar de que cada unidad es capaz de generar suficiente impulso para elevarse del suelo, por sí solo es incapaz de vuelo al igual que un helicóptero no puede volar sin su rotor de cola. Sin embargo, cuando se unen, estas unidades evolucionan en un sistema multi-rotor sofisticado capaz de vuelo coordinado y mucho más. [54]

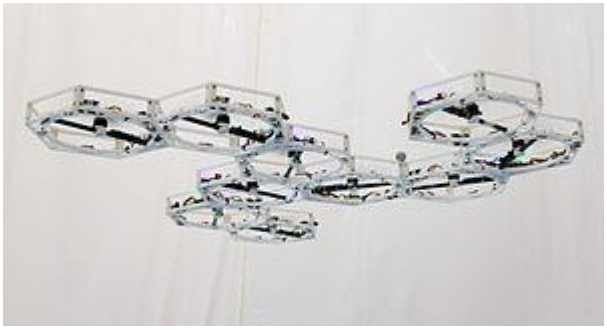


Figura 31. A 10-module configuration of the Distributed Flight Array in flight. [54]

2.2.1.9. M-blocks

Varios investigadores del Laboratorio de Informática e Inteligencia Artificial (CSAIL) del MIT han logrado desarrollar pequeños robots modulares llamados M-Blocks (Figura 32) que se auto ensamblan dependiendo de la tarea.

Figura 32. M-blocks

El concepto es especialmente prometedor ya que aunque los pequeños cubos que componen estas estructuras no disponen de partes externas móviles, sí son capaces de encaramarse los unos a los otros**,** rodar e incluso moverse cuando están pegados boca abajo a superficies metálicas. Para ello cada M-Block dispone de un volante que puede alcanzar velocidades de hasta 20.000 revoluciones por minuto**.** Al frenarse confiere un momento angular al cubo, y los imanes situados estratégicamente en las caras del cubo permiten que dos cubos se acoplen el uno al otro. [55]

2.2.1.10. YAMOR (2006)

El módulo YAMOR cuenta con control digital por FPGA y conexión por Bluetooth inalámbrica, lo que permite un control a distancia por medio de interfaces gráficas por computador, brindando un mayor grado de versatilidad en la aplicación. Como se observa en la Figura 33 físicamente fue diseñado con un único grado de libertad actuado por un servo y la comunicación entre módulos, con el fin de generar mayor simpleza y estudiar la locomoción adaptativa a partir de este. [56]

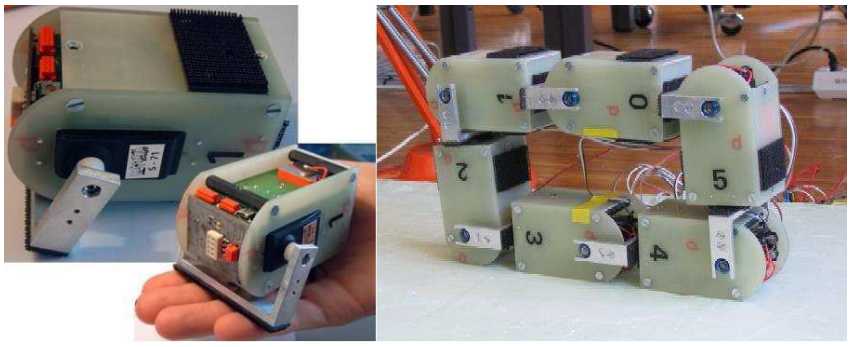


Figura 33. Módulo YAMOR – Configuración tipo rueda [56]

2.2.1.11. ATRON (2006)

Como se aprecia en la Figura 34, el robot Atron, basa su diseño e implementación a partir de las ideas de los robots modulares CONRO y M-TRAN, con un diseño de módulo esférico que puede girar alrededor de su eje medio, dividiendo el módulo en dos semiesferas para lograr la rotación de una con respecto a la otra. Los módulos se pueden acoplar entre ellos de forma que la rotación se haga en cualquiera de los tres ejes: x, y, z.

Figura 34. Robot tipo reticular – ATRON [57]

2.2.1.12. CUBELETS (2013)

Es un kit de construcción robótica modular que permite construir sistemas robóticos autónomos. Los Cubelets consisten en pequeños bloques cúbicos que se conectan a través de imanes y un conector unisex. Este conector permite que la información y la energía fluyan entre los módulos. Cada módulo Cubelets implementa una función específica; como detección, actuación o lógica, como se aprecia en Figura 35.

Figura 35. Módulos Cubelets [58]

2.2.1.13. ModRED II (2014)

De sus siglas en inglés (Modular Robot for Exploration and Discovery), es la segunda generación de robots modulares auto-configurable, clasificado como un sistema homogéneo con módulos de cuatro grados de libertad, como se aprecia en la Figura 36, cada módulo ha sido rediseñado con la inclusión de nuevas tecnologías

que mejoran las capacidades de procesamiento y de autonomía al sistema, cada módulo contiene un circuito integrado, un módulo XBEE que permite la comunicación inalámbrica entre los módulos. Adicionalmente le agregaron dos caras más de conexión para un total de cuatro mecanismos de conexión, para lograr conexiones complejas, junto con cuatro pares de transmisores y receptores infrarrojos en cada cara de conexión para como sensor de proximidad entre las caras, estrategias de localización y de comunicación local. [59]

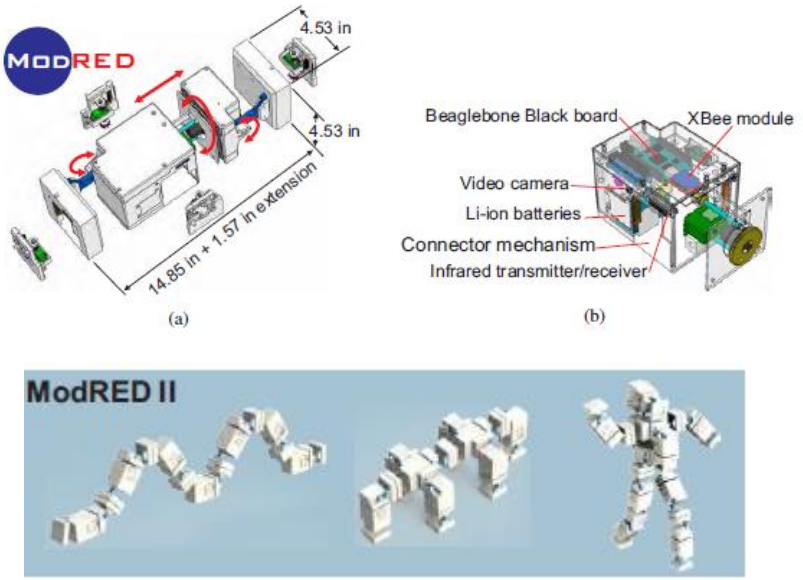


Figura 36. Esquema Módulo ModRED II – Posibles Configuraciones [59]

2.2.1.14. UBOT (2015)

Los módulos del Ubot son de los más tecnológicos debido a los recientes dispositivos mecánicos y electrónicos implementados para su desarrollo, con materiales que los hacen ser ultraligeros y cada uno posee una identificación independiente para cumplir la función que requiere. Este sistema robótico moderno cuenta con la capacidad dinámica de ajuste de geometría dependiendo de la tarea a realizar en labores de transporte de carga, para ajustar el centro de gravedad del objeto, por lo que se encuentra diseñado para que en se estructura se modifiquen los ángulos (como se aprecia en la Figura 37), con el uso de un modelo que está basado en un complejo modelo matemático adaptativo, la capacidad de compresión de la estructura se logra por al análisis continuo y discreto de los serpentoides donde por medio de una función conocida se calculan los ángulos que estáticamente dan más ventajas a toda la estructura. [60]

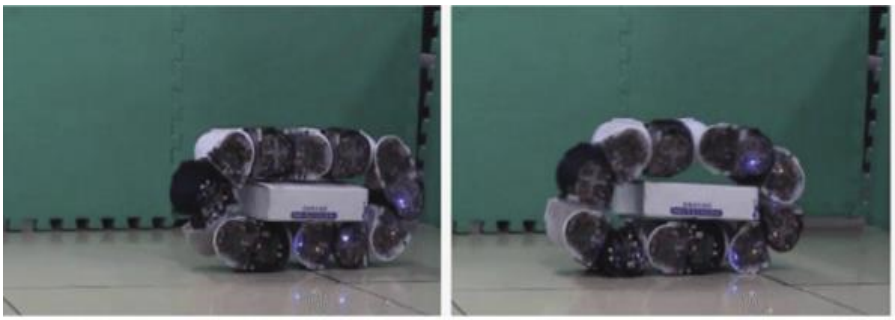


Figura 37. UBOT, plataforma transportando una carga [60]

2.2.2. Sistemas Robóticos Modulares realizados en la Universidad Militar "Nueva Granada" MECABOT Versión 1 a 4

A través de los diferentes trabajos de investigación liderados por el grupo DAVINCI - UMNG, se puede mostrar aquellos que han generado aporte en los sistemas planteados, los cuales permiten dar un paso y servir como punto de referencia para la autoconfiguración sobre los módulos robóticos existentes en la Universidad Militar Nueva Granada, como se indican a continuación:

2.2.2.1. DISEÑO Y SIMULACIÓN DE UN ROBOT MODULAR RECONFIGURABLE (2013)

El trabajo realizado por los Ingenieros Oscar Rubiano y Camilo Hurtado [61], muestra el diseño y la simulación de un robot modular reconfigurable, analizándolo desde diferentes enfoques, es decir, su arquitectura mecánica, su sistema de control, el diseño electrónico, la sensórica usada, el material empleado para su futura construcción y el ambiente virtual recreado para su simulación en WEBOTS. En la Figura 38 se muestra el diseño final planteado en este trabajo de investigación, resaltando que los factores peso y costo, los cuales determinan en gran medida el éxito que puede tener el proyecto y la posibilidad que existe de llevarlo a cabo en el futuro.

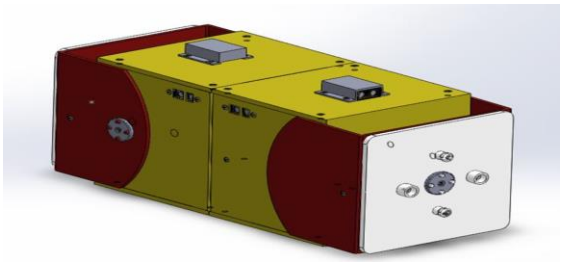


Figura 38. Diseño final del módulo propuesto por los Ingenieros Hurtado y Rubiano [61]

2.2.2.2. SIMULACIÓN E IMPLEMENTACIÓN DE MOVIMIENTOS PARA SISTEMA ROBÓTICO MODULAR CONSIDERANDO DIFERENTES CONFIGURACIONES (2016)

El trabajo realizado por las Ingenieras Paola Lancheros y Laura Sanabria [62], se diseñó teniendo en cuenta los primeros modelos implementados del módulo MECABOT, el cual se realizó un rediseño considerando todas las falencias de estos, como su gran tamaño y la falta de fuerza en los motores. El módulo MECABOT 3.0 presenta un diseño compacto y más estilizado que consta de 6 servomotores para cada semimódulo, 2 para pivote y 4 para las caras rotacionales, además de usar una tarjeta Teensy 3.2 para su comando y sistema de comunicación mediante Xbee Pro Serie 1 que envía las instrucciones a los módulos. Como se aprecia en Figura 39, el rediseño de los módulos permitió implementar las configuraciones oruga y serpiente en las que se acoplaba una serie de módulos mediante dos imanes en sus caras, la fuerza de esta conexión permitió generar movimientos ondulatorios típicos de las topologías 1D, pero para las configuraciones rueda y oruga es necesario que los módulos levanten más peso y su acople sea más fuerte.

Figura 39. Configuración oruga implementada con los módulos MECABOT 3.0, implementado por las Ingenieras Lancheras y Sanabria [62]

2.2.2.3. SIMULACIÓN E IMPLEMENTACION DE SISTEMA ROBÓTICO EN ARQUITECTURA TIPO RUEDA UTILIZANDO ROBOTICA MODULAR (2016)

El trabajo realizado por el Ingeniero Wilhelm Andrewy Miño [63], consistió en la configuración tipo rueda con robots modulares utilizando el sistema MECABOT 4.0, el cual fue completamente desarrollado en la Universidad Militar Nueva Granada, específicamente enfocando la locomoción en esta configuración. El principal propósito de este trabajo como lo muestra la Figura 40, fue simular e implementar la configuración tipo rueda haciendo uso de robots modulares, la cual es capaz de ofrecer un desplazamiento de un lugar a otro, y resaltar las ventajas que traen estos

sistemas como lo son: versatilidad en su mantenimiento, transporte y uso en las diversas aplicaciones en que la requieran.

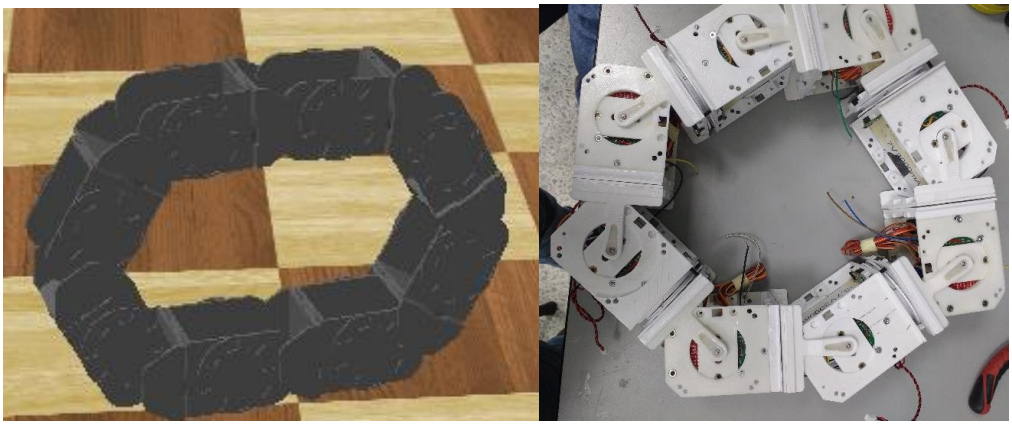


Figura 40. Diseño configuración tipo rueda MECABOT 4, por el Ingeniero Miño [63]

2.2.2.4. SIMULACIÓN E IMPLEMENTACIÓN DE UNA CONFIGURACIÓN DE ROBOT HEXÁPODO UTILIZANDO SISTEMAS DE ROBÓTICA MODULAR PARA EVALUAR SU LOCOMOCIÓN (2017)

El trabajo realizado por el Ingeniero Mateo Cotera [64], se centró en la configuración de tipo hexápodo, modelo bio-inspirado en los insectos muy utilizado en la robótica debido a su gran estabilidad y movimientos con independencia del terreno, realizando su simulación e implementación con los módulos MECABOT 4.0. Como se aprecia en la Figura 41, se buscó evaluar las capacidades motoras de esta configuración a partir de módulos, para esto se utilizó el software de simulación Webots que permitió simular las características físicas del modelo y su comportamiento ante el programa implementado en este.

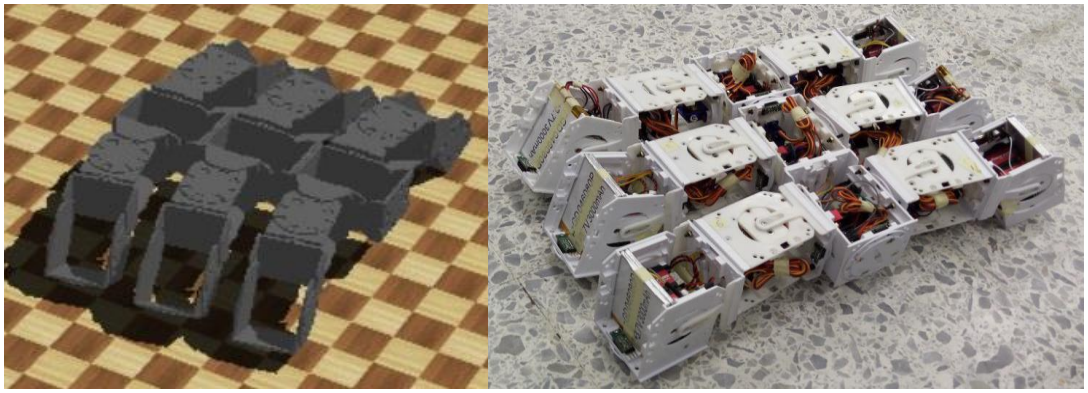


Figura 41.Diseño configuración modular tipo hexápodo MECABOT 4.0, por el Ingeniero Cotera [64]

2.2.3. Descripción Módulo MECABOT 4 en Software Webots

Como se puede apreciar en la Figura 42, el módulo dispone inicialmente de un modelo con cada uno de los motores del sistema, dado la simplicidad del software para efectos de simulación se importa el mismo modelo, sin embargo se excluyen los motores, para dejar únicamente los accionamientos mecánicos que tienen precedido el movimiento de los actuadores omitidos para efectos de la programación a realizar en cada uno de los controladores. El MECABOT tiene 4 grados de libertad (DOF) correspondientes a los que le brindan los motores de las ruedas de atrás-adelante, izquierda derecha y el pivote. Se puede dividir en cinco nodos sólidos: el cuerpo, las tres ruedas y el pivote

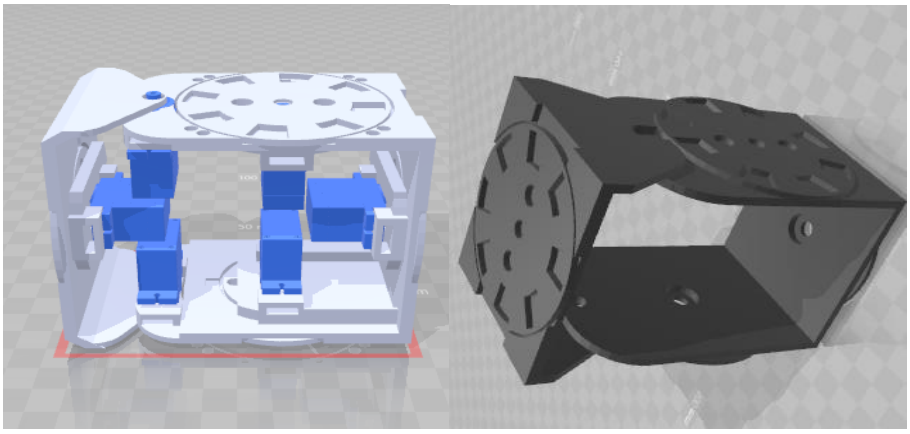


Figura 42. Ensamblaje completo y simplificado del Módulo MECABOT 4

2.2.4. Conclusiones del Capítulo

En este capítulo se presentó el tipo de robot modular de referencia sobre el cual se realizó el trabajo de autoconfiguración de arquitecturas, las diferentes herramientas físicas como tipos de conexión y modos de configuración entre módulos, la aplicación para la simulación, junto con los algoritmos propuestos para la búsqueda y de planificación de trayectorias, haciendo un repaso de los modelos ya existentes a nivel mundial, como en los propios elaborados en la universidad.

Por otra parte, se ahonda en los trabajos anteriormente realizados, para verificar el estado del arte en el que se encuentra el avance de la robótica modular, especialmente con aquellos que sean o puedan contar con herramientas para la autoconfiguración de diferentes arquitecturas, junto con la gestión propuesta por la

Universidad Militar Nueva Granada para aportar mayor conocimiento sobre este campo de investigación.

Se propone el modelamiento de robots modulares en WEBOTS con un enfoque más sencillo, dividiendo al robot en las partes que lo componen, luego se selecciona el nodo común a todas las partes para ubicarlas de manera jerárquica, con el fin de optimizar la velocidad de simulación se propone reducir algunas partes del robot como motores, uniones y demás que no son tenidas en cuenta al momento de simular y verificar la funcionalidad del sistema propuesto, que para el caso del MECABOT la forma más simple que lo puede representar es un paralelepípedo de 6 lados.

Tomando como base los trabajos elaborados en el grupo DAVINCI de la Universidad Militar Nueva Granada, se propuso realizar un trabajo que confluyera en realizar el diseño del control de los módulos MECABOT 4, que cuente con la capacidad de acoplarse con los módulos entre sí, para que de esta manera a través de protocolos de comunicación, se pueda lograr el armado de varias arquitecturas, anteriormente presentadas por tesis de pregrado anteriores en el programa, como lo son las arquitecturas oruga y hexápodo, con la característica de autoconfiguración de arquitecturas para ejecutar una trayectoria previamente planificada por el algoritmo de búsqueda A*.

CAPÍTULO III.

DISEÑO Y DESARROLLO DE HERRAMIENTAS Y ALGORITMOS PARA EL CONTROL INDIVIDUAL DE LOS MÓDULOS MECABOT 4 JUNTO CON LOS DISPOSITIVOS ASOCIADOS A CADA UNO

En este capítulo se presenta el diseño de control individual de **15 módulos** MECABOT 4.0, que abarca desde la integración de dispositivos y elementos necesarios para el funcionamiento de cada agente modular, hasta la generación de algoritmos, para el funcionamiento integrado de los mismos en conjunto, individualmente se asignó un controlador a cada módulo-robot, el cual es programado con código fuente en *lenguaje de programación* C.

El control parte desde el desplazamiento individual de cada módulo, teniendo en cuenta los actuadores, sensores y accesorios integrados a cada módulo para lograrlo, la integración de protocolos de comunicación junto con los dispositivos utilizados para la transmisión y recepción de datos, la capacidad de acoplamiento de los módulos entre sí, para posteriormente establecer estrategias de armado y desarmado que permitan lograr la configuración de las arquitecturas oruga y hexápodo.

En el **Anexo *1*** y en el **Anexo *2*** se muestran las variables declaradas para los módulos 1 (maestro) y del 2 hasta el 15 (esclavos) respectivamente, que se encuentran definidas y explicadas a lo largo del documento.

3.1. Diseño de Control para el Desplazamiento de cada Módulo Individual

3.1.1. Definición de cero de coordenadas absoluto módulo MECABOT

Para el inicio del trabajo se verificó el eje coordenado obtenido a partir de la importación de piezas de archivos VRML (.wrl) [1], como se puede apreciar en la Figura 43, el módulo asocia un sistema coordenado compuesto por tres ejes, el eje "*x*", el eje "*y*", finalmente el eje "*z*"; marcados con líneas de color "rojo", "verde" y "azul" respectivamente.

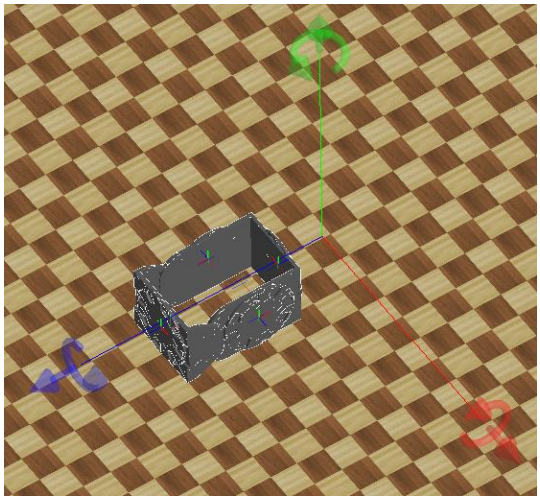


Figura 43. Eje coordenado módulo MECABOT 4. Tomado del Software WEBOTS. Elaboración propia

Es importante precisar que el origen del eje coordenado, se encuentra situado fuera del módulo, lo que presenta dificultades al momento de determinar su posición[2] sobre un plano y por lo tanto se hace necesario un ajuste al cero de coordenadas absoluto, al momento de establecer el control, el cual debe corresponder al centro de giro de las dos ruedas laterales, símil a un sistema de robot móvil con configuración diferencial (Figura 44) solo que sin el apoyo de una tercera rueda o "rueda loca" puesto que el módulo de por sí mantiene el equilibrio.

[1] Para conocer el detalle de importación de geometrías, ver el trabajo realizado por las Ingenieras Paola Lancheros y Laura Sanabria [62].

[2] La posición se obtiene a través del dispositivo GPS integrado al módulo, para conocer el detalles ver sección 3.1.4., página 76.

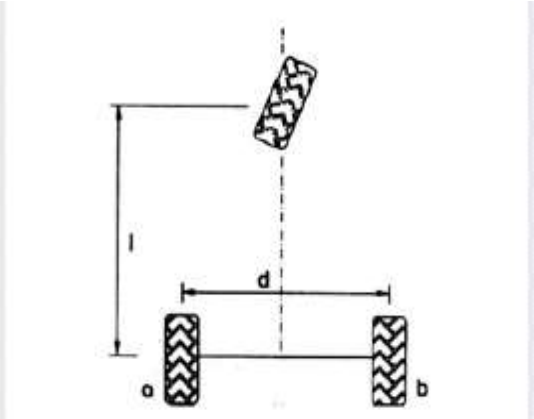


Figura 44. Configuración Robot Móvil Diferencial [65]

Como se puede apreciar en la Figura 45, el ajuste al cero de coordenadas absoluto, al momento de establecer el control, se hace principalmente a lo largo del eje *z* marcado con la línea de color "azul" del sistema de coordenadas asignado al módulo, con la intersección de la recta prolongada entre los centros de giro de las dos ruedas laterales, el cual se encuentra señalado con una marca de color "naranja" en la Figura 45, lo que representa un "*offset"* en el código del programa de una distancia de 0.087m a lo largo del eje "*z*" desde el origen del sistema coordenado.

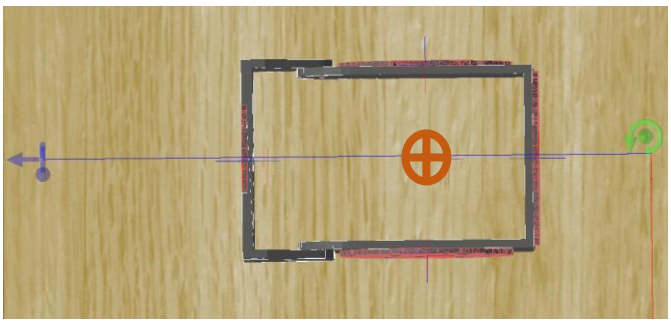


Figura 45. Eje coordenado módulo MECABOT 4 – Vista desde arriba. Tomado del Software WEBOTS. Elaboración propia

3.1.2. Actuadores y conectores definidos para controlar cada módulo MECABOT

Para establecer el control de desplazamiento sobre cada módulo por individual, se definen y se señalan en la Figura 46, cada uno de los actuadores que componen el módulo junto con los elementos de acople, los cuales son descritos en la Tabla 2; así como también se mencionan en la Tabla 3 los accesorios de posicionamiento y comunicación del sistema.

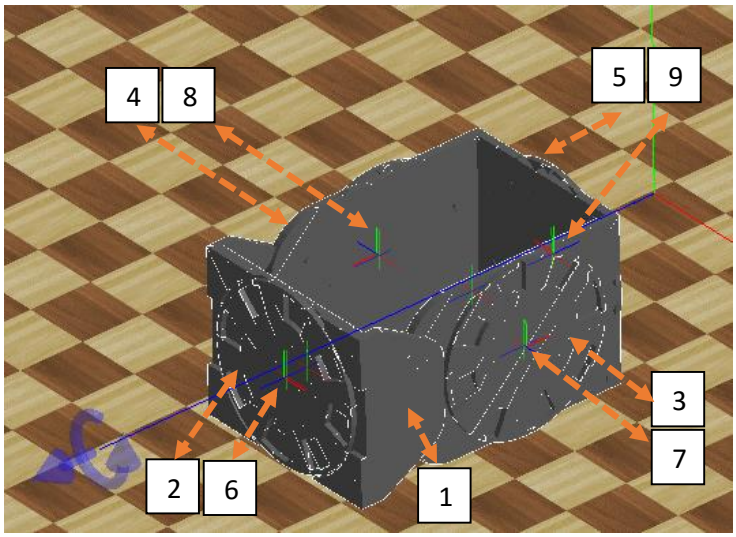


Figura 46. Vista isométrica módulo MECABOT, distinción de actuadores y accesorios. Tomado del Software WEBOTS. Elaboración propia

Como se señala en la Figura 46, de los componentes descritos en la Tabla 2, se evidencia que hay cinco (5) motores[3] definidos, uno (1) para cada una de las cuatro (4) ruedas, enfocados para el desplazamiento individual del módulo y para voltear los demás módulos que se acoplen a dichas ruedas mientras se arman las arquitecturas preestablecidas, el quinto motor se encuentra definido para el accionamiento del pivote el cual sirve para realizar, mediante movimientos integrados con los demás módulos, el desplazamiento en conjunto durante la implementación de alguna arquitectura.

Tabla 2. Lista de Actuadores y Conectores establecidos para cada Módulo. Elaboración propia

Motor	Conector
1. Motor Pivote	No aplica
2. Motor Rueda Pivote	6. Conector Pivote
3. Motor Rueda Izquierda	7. Conector Izquierdo
4. Motor Rueda Derecha	8. Conector Derecho
5. Motor Rueda Posterior	9. Conector Posterior

Entre los elementos definidos para cada módulo, se puede apreciar en la Figura 46 que se señalan los conectores[4] (marcados con números del 6 al 9) igualmente descritos en la Tabla 2, referenciados uno (1) para cada rueda, éstos son una herramienta que brinda el software Webots, los cuales son usados para simular sistemas de acoplamiento mecánico u otro tipo de dispositivo, que puede

[3] Para conocer el detalle acerca del control del motor, ver sección 3.1.3., página 74.
[4] Para conocer el detalle acerca del control del conector, ver sección 3.2.1., página 86.

dinámicamente crear una conexión física con otro dispositivo del mismo tipo (por ejemplo, con otro módulo). [66]

Tabla 3. Lista de dispositivos incorporados para cada módulo en el Software WEBOTS. Elaboración propia

Dispositivos Incorporados en WEBOTS
10. Compass
11. GPS
12. Emitter
13. Receiver

Finalmente, como se aprecia en la Tabla 3 se incorporaron a cada módulo 4 dispositivos adicionales, de los cuales dos (2), los números 10. COMPASS y 11. GPS[5], se utilizaron para permitir al controlador de cada módulo durante la ejecución del programa, de manera autónoma, conocer su ubicación y orientación con respecto al eje coordenado establecido para el mapa de trabajo. Así mismo, se agregan a cada módulo MECABOT los otros dos (2) dispositivos, los números 12. EMITTER y 13. RECEIVER[6], utilizados para la comunicación entre módulos, de forma que se pueda conocer el momento en que cada módulo ejecuta y culmina la rutina encomendada, al momento de recibir (por el receptor) el comando esperado para ejecutarla, así como también enviar el comando específico (por el emisor) que permita al sistema integrado continuar con la secuencia programada.

3.1.3. Dispositivo ROTATIONAL MOTOR para controlar desplazamiento de cada módulo MECABOT

Un motor o ROTATIONAL MOTOR en Webots, puede ser utilizado para alimentar una articulación (HINGEJOINT, ver Figura 47) con el fin de producir un movimiento de rotación alrededor del eje elegido. [67]

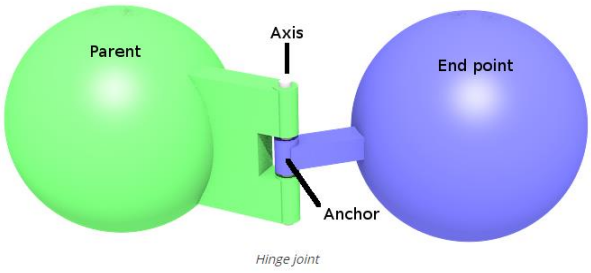


Figura 47. Diagrama HINGEJOINT para Software WEBOTS. [68]

[5] Para conocer el detalle acerca del control de los dispositivos GPS y COMPASS, ver sección 3.1.4., página 76.
[6] Para conocer el detalle acerca del control de los dispositivos EMITTER y RECEIVER, ver sección 3.3.1., página 96.

La asignación de los motores para cada módulo en el software Webots, como se aprecia resaltado en la Figura 48 con recuadro de color “rojo”, se realiza en el segundo nivel de jerarquía de cada módulo, en la carpeta *hijos* o *children*, después de asociar y definir cada rueda como tipo *Hingejoint* (resaltado con círculo de color “rojo”), para posteriormente indicar los dispositivos asociados al movimiento de ésta última, catalogados como *RotationalMotor*. Para el ejemplo en la Figura 48, se hace con referencia a la rueda posterior y su motor, señalados con el número cinco (5) en la Figura 46 y en la Tabla 2, que para efectos de control en la programación ha sido denominado como “*motor_rueda_posterior*”.

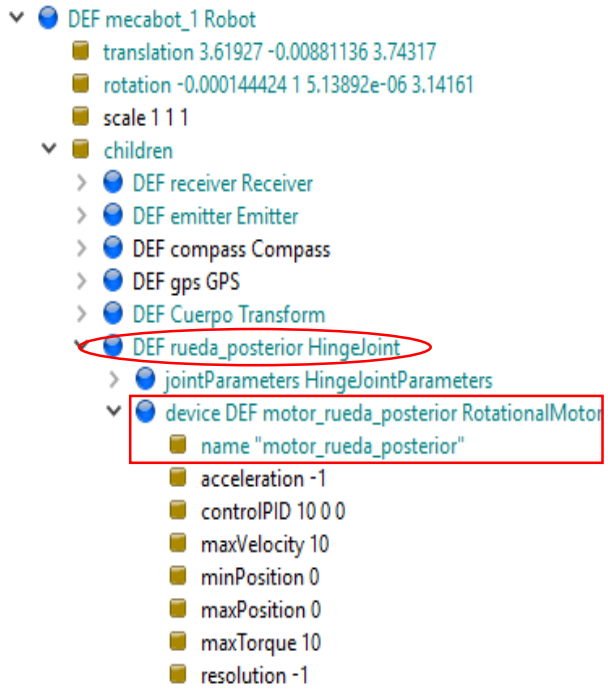


Figura 48. Jerarquía Nodo Robot MECABOT, identificación de motor – Software WEBOTS. Elaboración propia

La definición de variables para controlar los motores, procede a la asignación en la interfaz de programación de aplicaciones (API) de los motores para cada módulo en el software Webots, como se aprecia resaltado en el Código 1 con recuadro de color “rojo” se resalta el comando de código específico para la definición de la variable “*motor_rueda_posterior*” (variable declarada y mostrada en **Anexo *1***) de lo mencionado anteriormente en la Figura 48, de forma que se enlace la variable física con la de control.

```
//variables motores
motor_pivote=wb_robot_get_device("motor_pivote");
motor_rueda_pivote=wb_robot_get_device("motor_rueda_pivote");
motor_rueda_izquierda=wb_robot_get_device("motor_rueda_izquierda");
motor_rueda_derecha=wb_robot_get_device("motor_rueda_derecha");
motor_rueda_posterior=wb_robot_get_device("motor_rueda_posterior");
```

Código 1. Definición de variables para controlar los motores - Software WEBOTS. Elaboración propia

3.1.4. Dispositivos GPS y COMPASS para controlar posicionamiento de cada módulo MECABOT

El dispositivo GPS en el software WEBOTS, es utilizado para modelar un sensor de posicionamiento global (GPS, por sus siglas en ingles), el cual permite obtener información sobre la posición absoluta (del sistema coordenado del módulo mostrado en la Figura 45, con respecto al espacio de trabajo) desde el programa del controlador. [69]

Por otra parte, el dispositivo COMPASS en el software WEBOTS, es utilizado para modelar una brújula digital (sensor magnético) de uno, dos o tres ejes. El nodo *Compass* devuelve un vector que indica la dirección del *norte virtual*. El *norte virtual* está especificado por el campo *northDirection* del nodo *WorldInfo*. [70]

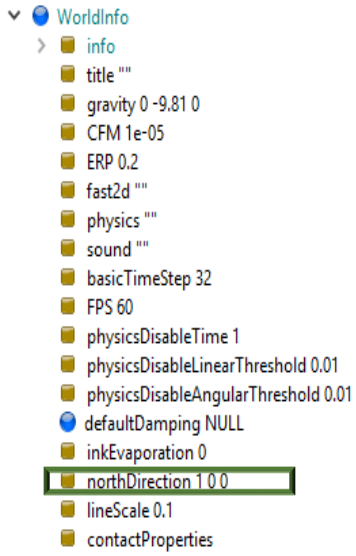


Figura 49. Jerarquía nodo WorldInfo, identificación campo "northDirection" – Software WEBOTS. Elaboración propia

Como se puede apreciar en la Figura 49, el campo *northDirection* del nodo *WorldInfo*, corresponde a "*x*"=1, "*y*"=0 y "*z*"=0, lo que especifica que el norte se encuentra a lo largo del eje "*x*" del sistema cartesiano (Figura 50), marcado con línea de color "rojo", significa que la orientación del módulo es relativa al norte del sistema coordenado del espacio de trabajo.

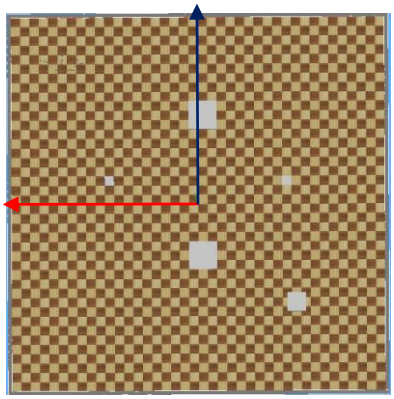


Figura 50. Espacio de trabajo definido en el ambiente virtual del Software WEBOTS, identificación de sistema cartesiano. Elaboración propia

La integración del dispositivo COMPASS al módulo, superpone los ejes del dispositivo en el mismo robot, mostrados en la Figura 43, lo que resulta que el sistema coordenado es el mismo para ambos nodos (robot y dispositivo COMPASS) y que la lectura asociada a la orientación del dispositivo se muestra asociada igualmente a la del módulo.

En la Figura 51, se muestran resaltados con el recuadro de color "purpura", los dispositivos GPS y COMPASS incluidos para cada módulo en el software WEBOTS, los cuales para efectos de control en la programación han sido denominados como "*gps*" y "*compass*" respectivamente.

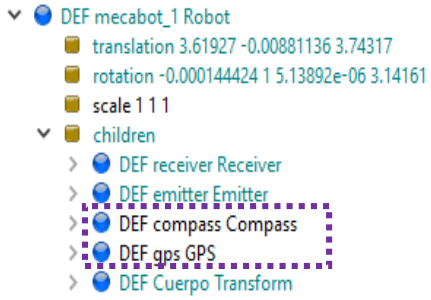


Figura 51. Jerarquía Nodo Robot MECABOT, identificación de dispositivos GPS y COMPASS – Software WEBOTS. Elaboración propia

Se prosiguió a la asignación de los dispositivos señalados en la Figura 51 en la ventana del controlador de los dispositivos para cada módulo en el software Webots, como se aprecia resaltado en el Código 2, se muestra el comando de código específico para la definición de las variables "*globalpos*" para el dispositivo "*gps*" y "*globalrot*" para el dispositivo "*compass*" (declaradas en **Anexo *1***), de forma que se enlace las variables físicas con las de control. Adicionalmente, se aprecia que los dispositivos son habilitados para la lectura de datos durante la ejecución del programa con las funciones definidas por el Software WEBOTS "*wb_gps_enable*" y "*wb_compass_enable*" respectivamente para las variables previamente declaradas.

```
  // get gps tag and enable
globalpos= wb_robot_get_device("gps");
wb_gps_enable(globalpos, TIME_STEP);
wb_robot_step(TIME_STEP);    // execute simulation step
// get compass tag and enable
globalrot = wb_robot_get_device("compass");
wb_compass_enable(globalrot, TIME_STEP);
wb_robot_step(TIME_STEP);    // execute simulation step
```

Código 2. Definición de variables para controlar los dispositivos GPS y COMPASS - Software WEBOTS. Elaboración propia

A diferencia del dispositivo GPS, al cual para su operación únicamente se le realiza el *Offset* a la lectura de datos (previamente explicado y mostrado en la Figura 45); para la operación del dispositivo COMPASS es necesario registrar los datos arrojados por el sensor mediante alguna expresión matemática, la caracterización del sensor *Compass* integrado a cada módulo, se realizó para proyectar dicha expresión que permita obtener la orientación en grados a partir de la lectura arrojada por cada eje del dispositivo, los cuales miden el grado de alineación de cada eje del módulo con respecto a la dirección norte marcada por el plano (el eje "*x*" del sistema cartesiano, Figura 50).

- Caracterización del dispositivo COMPASS Software WEBOTS para su operación durante la ejecución del programa:

Para la caracterización del sensor COMPASS, se realizaron una serie de pruebas en la lectura de datos, en las que se midió la orientación del módulo de su eje "z" (eje marcado con línea de color "azul", Figura 43), con respecto al *northDirection* del plano de trabajo, para analizar las condiciones en las cuales es posible establecer una expresión matemática que permita obtener la orientación en grados.

Las pruebas se realizaron girando el módulo cada 90 grados, como se aprecia en la Figura 52, para tomar los datos arrojados por el dispositivo como se muestran en la Tabla 4, lo que permite concluir que la orientación del módulo en el eje "*z*" se encuentra relacionado con la función *coseno* y el eje "x" con la función *seno*.

Tabla 4. Caracterización Dispositivo Sensor COMPASS - Software WEBOTS. Elaboración propia

Caso	Sentido de Orientación del módulo a lo largo del	Orientación del módulo respecto a *northDirection*	Lectura en X COMPASS	Lectura en Z COMPASS
1	Eje Z	**π/2**	1	0
2	Eje X	**0**	0	1
3	Eje –Z	**3π/2**	-1	0
4	Eje –X	π	0	-1

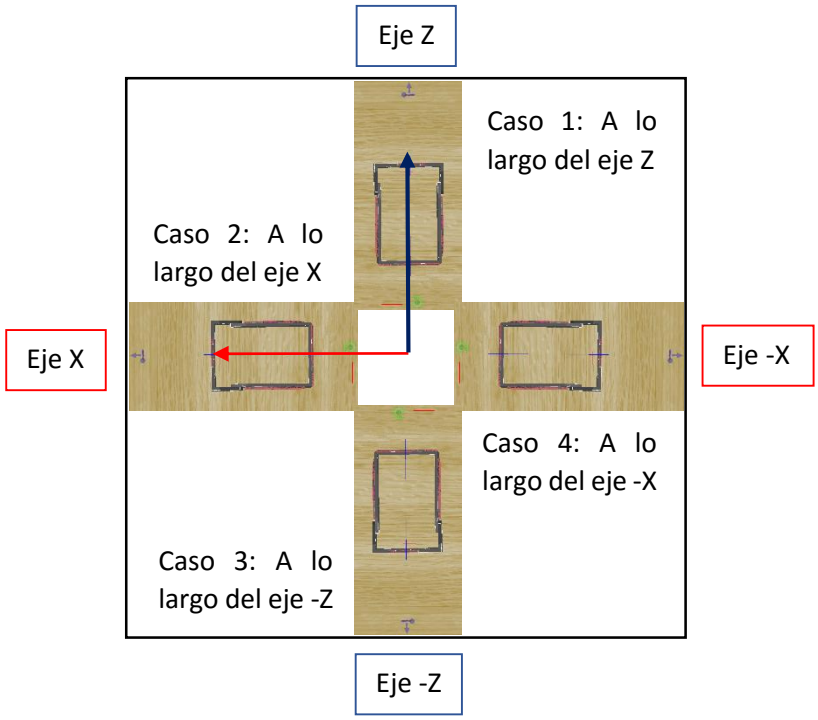


Figura 52. Pruebas de orientación del módulo para caracterización de dispositivo COMPASS – Software WEBOTS. Elaboración propia

Establecida la relación trigonométrica de cada eje del módulo y el dispositivo COMPASS, se procedió a obtener la expresión que permitiera la obtención del ángulo de orientación en grados, realizando pruebas de giro del módulo a cada grado de orientación y tomando lectura de los ejes "*x*" y "*z*" del dispositivo COMPASS, bajo las expresiones algorítmicas arco*seno* y *arcocoseno* respectivamente, siendo éstas las que gobiernan la lectura de cada eje de acuerdo al patrón presentado en la Tabla 4, el detalle se puede apreciar en el **Anexo *4***.

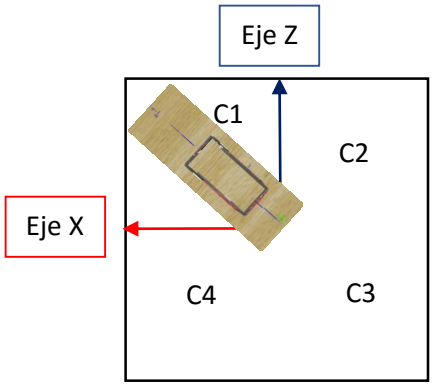


Figura 53. Pruebas de orientación del módulo a partir de expresiones algebraicas obtenidas, para caracterización de dispositivo COMPASS y obtención de orientación del módulo en grados – Software WEBOTS. Elaboración propia

A partir de las pruebas con el uso de expresiones algebraicas, se concluye con los datos reflejados en el **Anexo *4***, como se muestra en la Tabla 5, según la partición de cuadrantes de la Figura 53, que el uso de las relaciones trigonométricas, permite establecer una orientación del módulo en grados con respecto a la lectura del eje "*z*" el cual describe un comportamiento creciente entre los cuadrantes *C1* - *C2* y decreciente entre los cuadrantes *C3* – *C4*, el cual se complementa para la obtención en grados para todo el plano cartesiando, con el signo de la lectura del eje "*x*", que es positivo para los dos primeros cuadrantes y negativo para los dos últimos.

Tabla 5. Obtención de orientación en grados con las expresiones trigonométricas obtenidas, para caracterización dispositivo Sensor COMPASS - Software WEBOTS. Elaboración propia

Cuadrante	Lectura Eje "X" asin(Comp [x])	Lectura Eje "Z" acos(Comp [z])
C1	Signo +	**0 – 180°**
C2		
C3	Signo -	**180 – 0°**
C4		

Con la caracterización del dispositivo COMPASS mostrado en la Tabla 4 y los resultados obtenidos mediante el uso de expresiones trigonométricas en la Tabla 5, procede a la definición de variables referidas en el Código 2, definir la expresión matemática a ser incluida en la secuencia de programación del controlador de cada módulo, con el fin de obtener las coordenadas de posición y orientación de cada uno en el momento deseado durante la ejecución del programa.

Como se aprecia en el Código 3, se define la función *coordenadas_modulo*, que inicia con la lectura de datos de los dispositivos incluidos mediante el uso de las funciones definidas por el Software WEBOTS "*wb_gps_get_values*" y "*wb_compass_get_values*" (en el recuadro de color "azul") para las variables definidas "*pos3D*" y "*north3D*" respectivamente, se continuó con la obtención de orientación en grados del dispositivo COMPASS haciendo uso de las expresiones trigonométricas mencionadas en la Tabla 5, registrándose en la variable "*angxz*" para ambos ejes (en el recuadro de color "amarillo").

Posteriormente, se obtuvo mediante expresiones lógicas el ángulo de orientación en grados, tomando como patrón la lectura del eje "*z*", que a pesar de que registra únicamente valores entre 0 y 180°, se puede considerar que si el signo de la lectura del eje "*x*" es positivo el módulo se encuentra orientado entre los cuadrantes *C1* - *C2* y sí en cambio es negativo entre los cuadrantes *C3* – *C4*, por lo tanto se realiza el ajuste para éste último caso de restarle 360°, tal y como se expresa en el código de programación (Código 3, recuadro de color "amarillo"), con el fin de obtener la

orientación del módulo en grados en el rango de 0 a 359°, es decir en lo circundante de todo el plano cartesiano de trabajo.

```
void coordenadas_modulo()
{
  //para hallar coordenadas en mm y orientación del módulo
  pos3D = wb_gps_get_values(globalpos);
  north3D = wb_compass_get_values(globalrot);
  angxz[0]=asin(north3D[0])*180/PI;
  angxz[1]=acos(north3D[2])*180/PI;
  if(angxz[0]>=0){angulo=round(angxz[1]);}
  else {angulo=round(360-angxz[1]);}
  if(angulo==360){angulo=0;}
  //calculo de posicion modulo
  if(final_morfologia!=2){
    posxz[0]=round((pos3D[0]+0.087*cos(angulo*PI/180))*1000);
    posxz[1]=round((pos3D[2]+0.087*sin(angulo*PI/180))*1000);
  }
  //calculo de posicion HEXAPODO
  else{
    posxz[0]=round((pos3D[0]-0.028*cos(angulo*PI/180))*1000);
    posxz[1]=round((pos3D[2]-0.028*sin(angulo*PI/180))*1000);
  }
  dist=sqrt(pow(zf-posxz[1],2)+pow(xf-posxz[0],2));
  //printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
}
```

Código 3. Secuencia de programación para controlar los dispositivos GPS y COMPASS - Software WEBOTS. Elaboración propia

Finalmente, se obtuvo la expresión matemática que registra la posición del módulo (Código 3, recuadro de color “rojo”), entorno a la variable “*posxz*” para cada eje del plano cartesiano “*x*” y “*z*”, considerando el “*offset”* en el código del programa de una distancia de 0.087m a lo largo del eje “*z*” desde el origen del sistema coordenado del módulo, según lo mostrado en la Figura 45, ajuste que se ejecutó considerando las expresiones trigonométricas *seno* y *coseno* explicadas anteriormente para cada eje del sistema coordenado.

3.1.5. Algoritmo y Pseudocódigo para el control de desplazamiento del módulo MECABOT 1

El desplazamiento del módulo individual se logra a través de los motores de las ruedas de la izquierda y de la derecha, señalados en la Tabla 2 con los números tres (3) y cuatro (4) respectivamente, el motor de ambas ruedas precede el movimiento en coordenadas absolutas, lo cual implica el almacenamiento de la posición actual en una variable para ejecutar una nueva cantidad de movimiento. Como se puede evidenciar en el Código 4, se aprecia en la secuencia de programación del controlador de cada módulo, cuatro (4) funciones definidas para desplazar el módulo hacia el frente y atrás, como también rotar hacia la izquierda y derecha; lo anterior se logra almacenando la posición actual de cada rueda en las variables “*pos_ri*” y “*pos_rd*” respectivamente, para sumarle o restarle a dicha posición *(pi/180) radianes* en cada rueda, dependiendo sea el caso deseado.

```
static void modulo_defrente()
{
  pos_ri=pos_ri+PI/180;
  pos_rd=pos_rd+PI/180;
  mover_motor_iz_der();
  coordenadas_modulo();
}
```

```
static void modulo_atras()
{
  pos_ri=pos_ri-PI/180;
  pos_rd=pos_rd-PI/180;
  mover_motor_iz_der();
  coordenadas_modulo();
}
```

```
static void modulo_izquierda()
{
  pos_ri=pos_ri-PI/180;
  pos_rd=pos_rd+PI/180;
  mover_motor_iz_der();
  coordenadas_modulo();
}
```

```
static void modulo_derecha()
{
  pos_ri=pos_ri+PI/180;
  pos_rd=pos_rd-PI/180;
  mover_motor_iz_der();
  coordenadas_modulo();
}
```

Código 4. Secuencia de programación para desplazar el módulo mediante el control de la posición de los motores de las ruedas izquierda y derecha - Software WEBOTS. Elaboración propia

Como se puede evidenciar en el Código 4, en cada función una vez definida la nueva posición de cada rueda en las variables "*pos_ri*" y "*pos_rd*", se hace el llamado de la función "*mover_motor_iz_der*", que de manera recursiva se definió para ejecutar en la simulación del programa el movimiento en los motores de la rueda izquierda y derecha a su nueva posición, como se puede apreciar en la Figura 56 en el recuadro de color "azul".

```
void mover_motor_iz_der()
{
  outputi=div(round(pos_ri*180/PI), 360);p_rig=outputi.rem;if(p_rig<0){p_rig=360+p_rig;}
  outputd=div(round(pos_rd*180/PI), 360);p_rdg=outputd.rem;if(p_rdg<0){p_rdg=360+p_rdg;}
  //printf("pos_ri= %lf, %d gr , pos_rd = %lf, %d gr\n", pos_ri ,p_rig , pos_rd, p_rdg );
  wb_motor_set_position(motor_rueda_izquierda,pos_ri);
  wb_motor_set_velocity(motor_rueda_izquierda,1);
  wb_motor_set_position(motor_rueda_derecha,pos_rd);
  wb_motor_set_velocity(motor_rueda_derecha,1);
  wb_robot_step(25);
}
```

Código 5. Secuencia de programación para ejecutar el desplazamiento del módulo mediante el control de la posición de los motores de las ruedas izquierda y derecha - Software WEBOTS. Elaboración propia

Con el objetivo de lograr el desplazamiento individual del módulo de manera controlada, para que autónomamente ejecute el movimiento desde una posición inicial hasta una posición deseada, se inició por definir la forma de obtener la posición y la orientación inicial del módulo con respecto al plano de trabajo, mediante el uso de la función *coordenadas_modulo* (previamente mostrada en el Código 3).

Posteriormente, se procedió a hallar la distancia a recorrer por el módulo y el sentido de orientación hacia donde desplazarse para poder llegar hasta el objetivo, como se puede apreciar en la Figura 54, se definieron las coordenadas iniciales (X_i, Z_i) registradas como las coordenadas actuales del módulo, obtenidas en la variable

"*posxz*" de la función *coordenadas_modulo*, como también las coordenadas finales (X_f, Z_f), las cuales son definidas por el usuario en el programa.

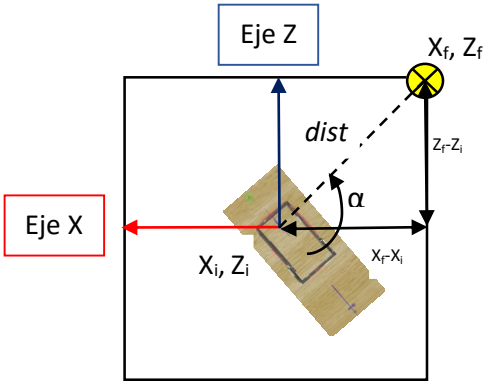


Figura 54. Representación esquemática para hallar la distancia y orientación desde la posición del módulo hasta una posición deseada – Elaboración propia

La distancia (marcada como "*dist*" en la Figura 54), se obtuvo a partir de la expresión matemática (Teorema de Pitágoras[7]) mostrada en la Ecuación 2, la cual se puede evidenciar incluida en el algoritmo de programación del controlador para cada módulo con el registro de la variable "*dist*" en el Código 3 (recuadro de color "negro" – línea continua).

Ecuación 2:

$$dist = \sqrt{(Z_f - Z_i)^2 + (X_f - X_i)^2}$$

$$\text{dist} = \sqrt{(\text{Z}_\text{f} - \text{posxz}[1])^2 + (\text{X}_\text{f} - \text{posxz}[0])^2}$$

El ángulo de orientación ("*α*", Figura 54), se obtuvo a partir de la expresión trigonométrica *arcoseno* y el ajuste en grados dependiendo del cuadrante (definido en la Figura 53) en el que se halla la posición final con respecto a la inicial para cada uno de los ejes, de forma que se pueda establecer hacia cual cuadrante de los cuatro definidos se encuentra dirigida la posición final con respecto a la del módulo, en la Tabla 6, se muestran las expresiones lógicas planteadas para definir el cuadrante en el que se encuentre la posición deseada con respecto a la del módulo, en el momento de ejecutar.

Tabla 6. Definición de cuadrantes a partir de la diferencia de posición final – inicial. Elaboración propia

Cuadrante	Diferencia Eje Z	Diferencia Eje X
C1	$Z_f - Z_i >= 0$	$X_f - X_i > 0$
C2	$Z_f - Z_i > 0$	$X_f - X_i <= 0$
C3	$Z_f - Z_i <= 0$	$X_f - X_i < 0$
C4	$Z_f - Z_i < 0$	$X_f - X_i >= 0$

[7] Teorema de Pitágoras: En un triángulo rectángulo, el cuadrado de la hipotenusa es igual a la suma de los cuadrados de los catetos. [73]

Con el fin de poder obtener el ángulo de orientación "*α*" en magnitud angular acorde a cada cuadrante, siendo estos, entre 0° a 89° los ángulos a obtener para el primer cuadrante, 90° a 179° para el segundo cuadrante, 180° a 269° para el tercer cuadrante y 270° hasta 359° para el cuarto cuadrante, se realiza un ajuste haciendo una adición al ángulo "*α*", dependiendo del cuadrante hallado. Tal y como se puede apreciar en el Código 6, para la secuencia de programación del controlador de cada módulo, se definió la función "*orientación_módulo*" la cual se asignó para determinar el ángulo de orientación del módulo hacia el punto deseado y registrarlo en la variable "*alfa*", acorde a los parámetros definidos en la Tabla 6.

```
void orientacion_modulo()
{
  if((zf-posxz[1])==0 && (xf-posxz[0])==0){alfa=0;}
  else if((zf-posxz[1])>=0 && (xf-posxz[0])>0){alfa=round(asin((zf-posxz[1])/dist)*180/PI);}
  else if((zf-posxz[1])>0 && (xf-posxz[0])<=0){alfa=round(90+asin((fabs(xf-posxz[0]))/dist)*180/PI);}
  else if((zf-posxz[1])<=0 && (xf-posxz[0])<0){alfa=round(270-asin((fabs(xf-posxz[0]))/dist)*180/PI);}
  else if((zf-posxz[1])<0 && (xf-posxz[0])>=0){alfa=round(360-asin(fabs((zf-posxz[1]))/dist)*180/PI);}
  if(alfa==360){alfa=0;}
  //printf("Distancia:%lf, alfa:%d,\n", dist, alfa);
}
```

Código 6. Secuencia de programación para hallar la orientación del módulo hacia el punto deseado - Software WEBOTS. Elaboración propia

Para ejecutar el desplazamiento del módulo 1 (maestro) hasta el punto deseado, se definió la estructura secuencial como se muestra en la Figura 55, el estado de cada paso se representa mediante una marca registrada en la variable "*flag*", estructura que simboliza lo definido en la secuencia de programación del controlador del módulo en la función "*desplazamiento_modulo*", la cual se encuentra detallada en el **Anexo *5*** y se explica la lógica en la programación a continuación:

- El desplazamiento individual del módulo inicia en la marca "*flag=0*", con la lectura de coordenadas del módulo, de la cual se obtiene la distancia desde las coordenadas actuales del módulo hasta el punto deseado, junto con la lectura de la orientación del módulo, de la cual se obtiene el ángulo para dirigirse hacia las coordenadas mencionadas (Figura 54).
- Con la marca "*flag=1*", junto con los datos de coordenadas actuales, la distancia a recorrer y la orientación, se prosigue a realizar el giro del módulo sobre su centro de referencia (Figura 45), desde su posición angular actual, hasta la final, al ejecutar el movimiento en las ruedas, la una hacia adelante y la otra hacia atrás, o viceversa, dependiendo cual giro se encuentra más cercano en un radio de 180°, para lograr el giro del módulo deseado.
- Con la marca "*flag=2*", se ejecuta el movimiento del módulo haciendo girar ambas ruedas hacia adelante al mismo tiempo, una cantidad determinada de movimiento, que permita tomar la lectura a cada paso dado, para verificar si

el módulo ha realizado el corte con la recta del punto en "*z*" en el plano cartesiano del área de trabajo.

- Con la marca "*flag=3*", verificado el corte del módulo con la prolongación de la recta del punto en "*z*", se verifica que la distancia sea menor a 1mm, en caso afirmativo se concluye con el proceso de desplazamiento, la marca "*flag*" se iguala a 6 (seis), de no ser así se prosigue a realizar el corte ahora con el punto en "*x*".
- Con la marca "*flag=4*", se gira el módulo hasta las coordenadas finales que guie el camino hacia el corte en el punto en "*x*".
- Con la marca "*flag=5*", se ejecuta el movimiento del módulo haciendo girar ambas ruedas hacia adelante al mismo tiempo, una cantidad determinada de movimiento, que permita tomar la lectura a cada paso dado, para verificar si el módulo ha realizado el corte con la recta del punto en "*x*" en el plano cartesiano del área de trabajo, la marca "*flag*" se iguala a 0 (cero) para validar nuevamente el proceso en el punto en "*z*" y concluir con el desplazamiento.
- Con la marca "*flag=6*", llegado el módulo a las coordenadas finales, se procede a girar el módulo hasta la orientación final, acorde con el cálculo del controlador durante la ejecución del programa.

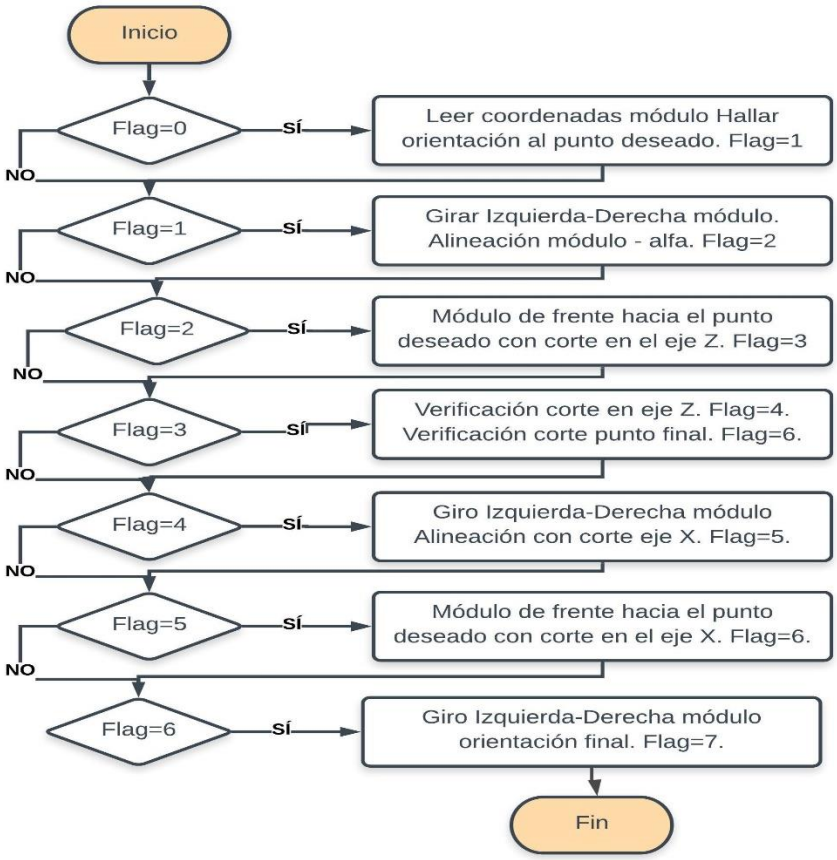


Figura 55. Estructura secuencial para ejecutar el desplazamiento del módulo 1 hasta el punto deseado. Elaboración propia

3.2. Diseño de Control para la Conexión de Módulos entre sus Ruedas de Acople

3.2.1. Dispositivo CONNECTOR para controlar los puntos de acople de cada módulo MECABOT

Como se definió anteriormente en la Figura 46 y en la Tabla 2, se señalan los conectores marcados con números del 6 al 9, para emular la conexión en cada rueda de cada módulo, los cuales fueron empleados como sistema de acoplamiento mecánico, bajo el principio de funcionamiento de un electroimán[8], para lograr la conexión y desconexión manipulada desde el controlador asociada a cada módulo.

En la Figura 56, a manera de ejemplo se muestra resaltado para la rueda posterior del módulo del tipo *Hingejoint* (con círculo de color "verde") en la jerarquía *children* con recuadro de color "verde", catalogado como *Connector*, el elemento con denominación *conector_posterior*, el cual permite emular el control de la conexión física entre módulos en la simulación del Software WEBOTS.

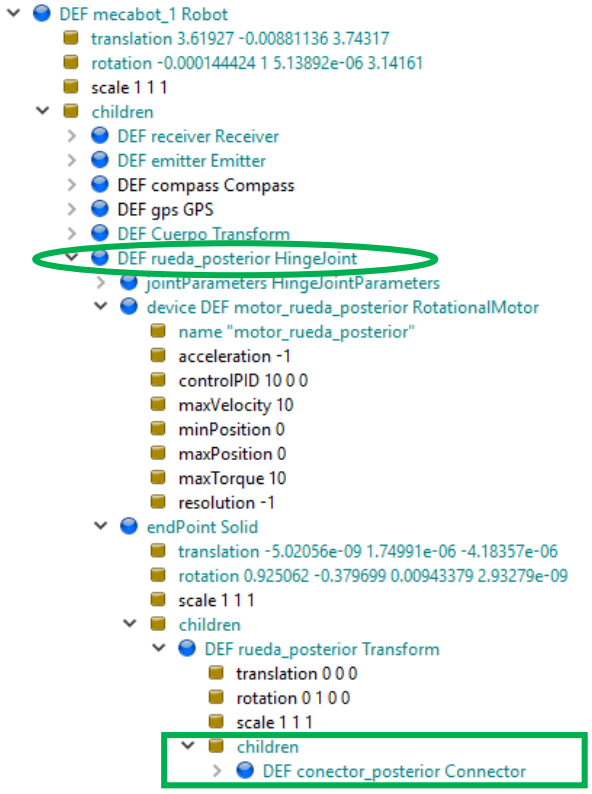


Figura 56. Jerarquía Nodo Robot MECABOT, identificación de conector – Software WEBOTS. Elaboración propia

[8] Electroimán: Elemento que adquiere las propiedades magnéticas de un imán natural cuando se hace circular una corriente eléctrica a través de un inductor que a su vez está enrollado en un núcleo de hierro. [74]

Para el manejo del conector en el Software WEBOTS, se presenta en la Figura 57, la configuración disponible para el elemento *Connector*, previamente definida por el Software, para tener en cuenta al momento de realizar los ajustes pertinentes, puesto que hay disponible una serie de campos que limitan la capacidad del elemento que se ven reflejados en el comportamiento del sistema durante la simulación.

```
Connector {
  SFString type              "symmetric"   # {"symmetric", "active", "passive"}
  SFBool   isLocked          FALSE         # {TRUE, FALSE}
  SFBool   autoLock          FALSE         # {TRUE, FALSE}
  SFBool   unilateralLock    TRUE          # {TRUE, FALSE}
  SFBool   unilateralUnlock  TRUE          # {TRUE, FALSE}
  SFFloat  distanceTolerance 0.01          # [0, inf)
  SFFloat  axisTolerance     0.2           # [0, pi]
  SFFloat  rotationTolerance 0.2           # [0, pi]
  SFInt32  numberOfRotations 4             # [0, inf)
  SFBool   snap              TRUE          # {TRUE, FALSE}
  SFFloat  tensileStrength   -1            # {-1, [0, inf)}
  SFFloat  shearStrength     -1            # {-1, [0, inf)}
}
```

Figura 57. Configuración disponible del elemento Connector para el control de acoplamiento. [66]

Las características que describen el comportamiento de los campos disponibles son [66]:

- *Type*: “symmetric”, “active”, o “passive”, Un conector “simétrico” solo puede bloquear (y desbloquear) otro conector “simétrico”. Un conector “activo” solo puede bloquear (y desbloquear) un conector “pasivo”. Un conector “pasivo” no se puede bloquear o desbloquear.
- *isLocked*: representa el estado de bloqueo del conector, el estado de bloqueo se puede cambiar a través de las funciones de *wb_connector_lock* y *wb_connector_unlock* en el interfaz de programación de aplicaciones.
- *autoLock:* especifica si el bloqueo automático está habilitado o deshabilitado. El bloqueo automático permite que un conector se bloquee automáticamente cuando un par compatible se hace presente.
- *unilateralLock*: indica que bloquear solo un par es suficiente para crear un enlace físico.
- *unilateralUnlock*: indica que desbloquear solo un par es suficiente para romper el enlace físico.
- *distanceTolerance*: La distancia máxima (en metros) entre dos conectores que se les permite bloquear con éxito. La distancia se mide entre los orígenes de los sistemas de coordenadas de los conectores.
- *axisTolerance*: el ángulo máximo (en radianes) entre los ejes “z” de dos conectores en los que se pueden bloquear con éxito.

- *rotationTolerance*: la diferencia de ángulo tolerada con respecto a cada una de las rotaciones de acoplamiento permitidas. (Figura 58)
- *numberOfRotations*: especifica cuántas rotaciones de acoplamiento diferentes se permiten en una rotación completa de 360 grados alrededor del eje “z” del conector. (Figura 58)
- *snap*: cuando es “True”, los dos conectores se ajustan automáticamente (alinean, ajustan, etc.) cuando se acoplan.

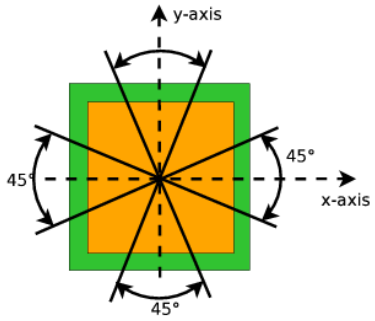


Figura 58. Ejemplo de alineación rotacional (número de rotaciones=4 y tolerancia rotacional 22.5°). [66]

En la Figura 59, se aprecia que para un exitoso acople durante la simulación, deben estar alineados los ejes de coordenadas de ambos conectores a unir, igualmente, se debe tener en cuenta los demás parámetros especificados anteriormente, como la distancia de tolerancia máxima permitida entre ejes, la tolerancia rotacional y el tipo de conector definido, acorde al objetivo esperado para el acople y desacople de módulos.

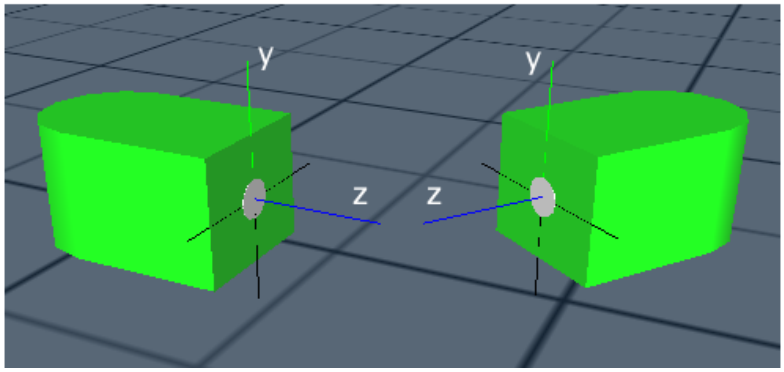


Figura 59. Ejemplo de alineación de ejes entre módulos para el control de acoplamiento [66]

Con respecto a los parámetros disponibles por el Software WEBOTS para el manejo del elemento *Connector* (Figura 57), las condiciones específicas para el funcionamiento de conexión y desconexión entre módulos (Figura 58 y Figura 59), se establecieron las características propias de cada conector de cada rueda conforme a la configuración mostrada en la Figura 60, el cual a manera de ejemplo muestra la

configuración del elemento con denominación *conector_posterior* (previamente definido en la Figura 56).

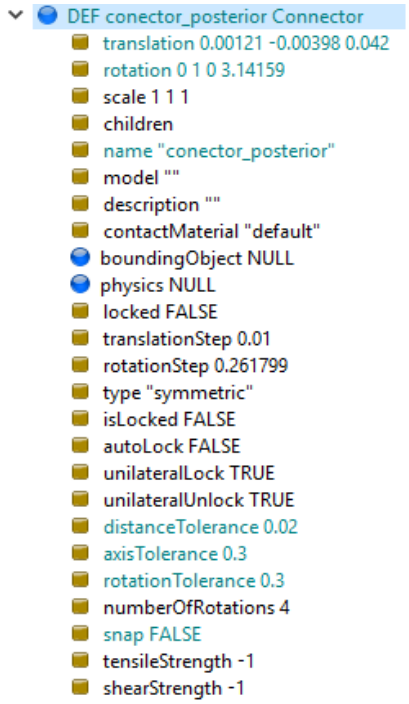


Figura 60. Configuración del elemento Connector para el control de acoplamiento – Software WEBOTS. Elaboración propia

Se prosiguió a la asignación de los elementos *Connector* (como el señalado en la Figura 56), en la interfaz de programación para cada módulo en el software Webots, como se muestra en el Código 7, con el comando de código específico de la definición de las variables "*conector*" para los elementos *Connector* que recibieron igualmente la denominación enunciada (variables declaradas y mostradas en **Anexo *1***), de forma que se enlacen las variables físicas con las de control.

```
// variables connector
conector_pivote=wb_robot_get_device("conector_pivote");
conector_izquierdo=wb_robot_get_device("conector_izquierdo");
conector_derecho=wb_robot_get_device("conector_derecho");
conector_posterior=wb_robot_get_device("conector_posterior");
```

Código 7. Definición de variables para controlar el elemento Connector – Software WEBOTS. Elaboración propia

La secuencia de programación para controlar el acoplamiento entre módulos a través de la conexión de sus ruedas, cumpliendo las características previamente definidas de alineación y separación entre ejes de conexión Figura 59 y Figura 60, se ejecuta mediante el uso de las funciones definidas por el Software WEBOTS *wb_connector_lock* y *wb_connector_unlock*, para conectar y desconectar respectivamente.

Como se aprecia en el Código 8, el conector de la rueda del pivote de cada módulo, señalado con el número seis (6) en la Figura 46 y en la Tabla 2, denominado "*conector_pivote*", es el utilizado para ejecutar la conexión entre módulos, debido a que se encuentra en la parte del frente de cada módulo, lo que significa que cada módulo desde el 2 hasta el módulo 15, una vez ejecuta su trayectoria de desplazamiento hasta la rueda de acople deseada, ejecutará las líneas de código dispuestas para su conexión y desconexión a través del "*conector_pivote*".

```
wb_connector_lock(conector_pivote);

wb_connector_unlock(conector_pivote);
```

Código 8. Secuencia de programación para controlar el elemento Connector en la conexión y desconexión de módulos – Software WEBOTS. Elaboración propia

3.2.2. Definición puntos de bordeo y de acercamiento a cada rueda de acople por módulo MECABOT

Para la conexión de módulos, se definió el módulo 1 como el módulo maestro, por lo tanto no se conecta a ningún otro módulo, mientras que los módulos del 2 al 15 como los esclavos, puesto que reciben instrucciones del módulo principal para realizar las conexiones para ejecutar las arquitecturas planteadas. Como se puede apreciar en la

Figura 61, se tomaron tres posibilidades: las ruedas posterior, izquierda y derecha del módulo a ser acoplado (señalado en el recuadro de color "negro" – línea continua) por el módulo de acople (señalado en el recuadro de color "rojo" – línea punteada); mientras que se definió la rueda del pivote del módulo, como la que debía realizar la acción de acople (Código 8).

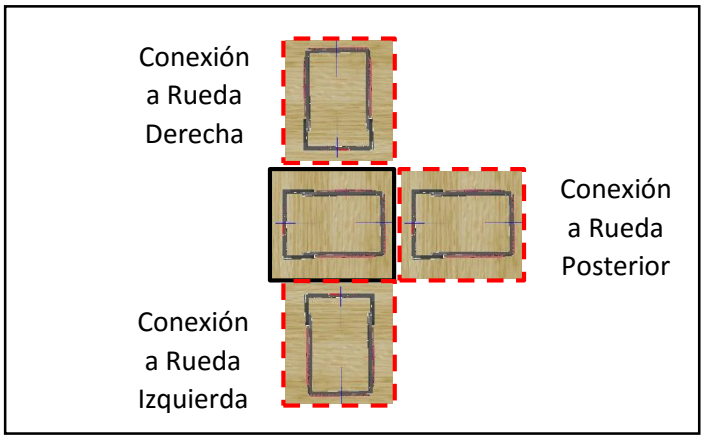


Figura 61. Definición de ruedas de conexión entre módulos – Software WEBOTS. Elaboración propia

Para lograr la conexión entre módulos, sin que el módulo de acople colisionase con el módulo a acoplar, como criterio de seguridad se diseñó una zona rectangular perimetral que impidiera cualquier contacto no deseado, para lo cual se establecieron, como se ilustra en la Tabla 7, una serie de cuatro (4) puntos alrededor del módulo a acoplar que sirvieran para que el módulo de acople pudiera "bordear" al otro módulo mientras se llegaba a la rueda a acoplar deseada.

Tabla 7. Definición de puntos de bordeo de cada módulo. Elaboración propia

Puntos de Bordeo	P_1, P_2, P_3 y P_4

Igualmente, como se ilustra en la Tabla 8, se definieron los puntos de acercamiento para permitirle al módulo de acople, acercarse de manera controlada a la rueda deseada del módulo a acoplar, hasta llegar a situarse justo al frente de la rueda, para que de esta manera los ejes de ambas ruedas se encuentren y estos puedan acoplarse mediante la acción de bloqueo del conector, previamente explicado en el Código 8.

Tabla 8. Definición de puntos de acercamiento para cada rueda de los módulos. Elaboración propia

Rueda a acoplar	Puntos de acercamiento
Pivote	No Aplica
Posterior	P_5, P_6, P_7
Izquierda	P_8, P_9, P_{10}
Derecha	P_{11}, P_{12}, P_{13}

En la Figura 62, se muestra la ubicación de los puntos de bordeo definidos en la Tabla 7 (señalados con puntos de color negro en el marco rectangular con línea punteada), los cuales conforman un rectángulo alrededor del módulo a acoplar, el objetivo de dichos puntos situados a los extremos de la zona perimetral, es permitirle al módulo de acople desplazarse hasta al módulo a acoplar de forma segura y controlada mediante la llegada al punto de bordeo más próximo, para continuar su trayectoria entre éstos hasta llegar a la altura de la rueda deseada del módulo a acoplar, puntos P_5, P_8 o P_{11}, para las ruedas posterior, izquierda o derecha respectivamente. A su vez, se muestra la ubicación de los puntos de acercamiento definidos en la Tabla 8 (señalados con puntos de color verde para los de distancia media y rojo para los más cercanos), el objetivo de éstos últimos es permitirle al módulo de acople acercarse de manera precisa y controlada a la rueda deseada del módulo a acoplar, con el fin de alinear los ejes de ambas ruedas y estos puedan acoplarse mediante

la acción de bloqueo del conector durante la simulación del programa en el Software WEBOTS.

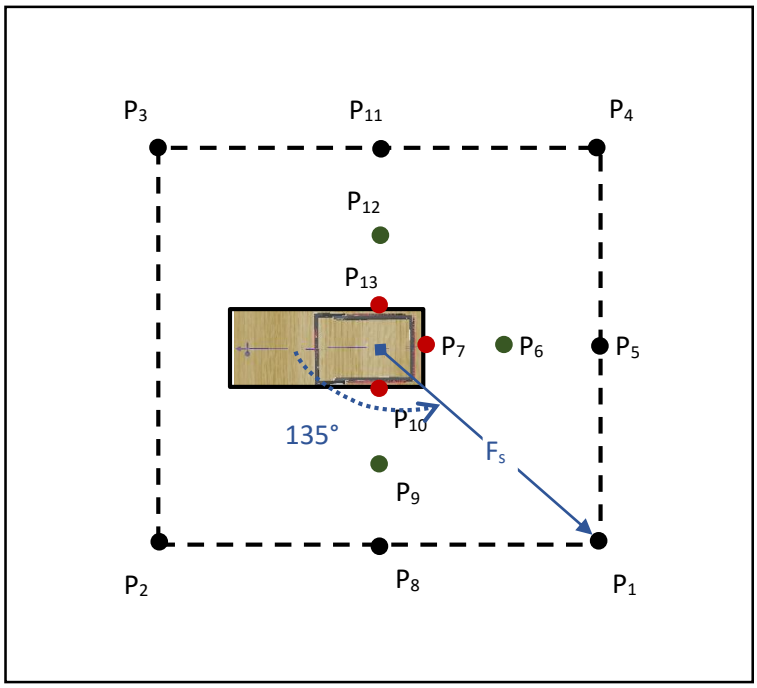


Figura 62. Definición y ubicación puntos de bordeo y puntos de acercamiento – Software WEBOTS. Elaboración propia

Para el cálculo de los puntos de bordeo del módulo a acoplar, en la Ecuación 3, se muestra a manera de ejemplo la obtención de las coordenadas del punto de bordeo *P1*, que se proyecta esquemáticamente en la Figura 62, tomándose como referencia los datos de la orientación y la posición del centro de giro (Figura 45) del módulo ("*Ang"* , "*posx*" y "*posz*" respectivamente), junto una distancia F_S tomada como factor de seguridad para definir la zona perimetral, así:

Ecuación 3:

$$P_{1X} = F_S * Cos(Ang - 135°) + posx$$

$$P_{1Z} = F_S * Sen(Ang - 135°) + posz$$

$$F_S = 0.2m$$

Con los parámetros establecidos en la Ecuación 3, para el desarrollo del control de desplazamiento del módulo de acople hasta la rueda deseada del módulo a acoplar, a través de los puntos de bordeo, en la interfaz de programación de los módulos 2 al 15 en el software Webots, se definieron las variables "*p*(*n*)*xz*", siendo *n* el número

de puntos de bordeo desde 1 hasta 4, como también los puntos de acercamiento desde 5 hasta 13 (variables declaradas y mostradas en el **Anexo *2***).

Como se muestra en el Código 9, en continuación del ejemplo de la Ecuación 3, para la definición de las variables "*P1x*" y "*P1z*", tomando F_s = 200, como también la variables de comunicación[9] "*datosderecibo"* entre módulos, siendo *datosderecibo*[3]= "*Ang"*, *datosderecibo*[1]= "*posx*" y *datosderecibo*[3]= "*posz*" del módulo a acoplar, se proyectan las líneas de programación para el cálculo de los puntos mencionados en el aplicativo del controlador, que hacen parte de la función "*puntos_bordeo*" (puesta en detalle en el **Anexo *6***), que detalla la obtención de los demás puntos de bordeo y de acercamiento, junto con la distancia desde el módulo de acople hasta los puntos de bordeo del módulo a acoplar, además ejecuta el análisis del punto más cercano para dar inicio al desplazamiento individual del mismo.

```
//puntos de bordeo
p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]);
p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]);
```

Código 9. Definición de variables de los puntos de bordeo – Software WEBOTS. Elaboración propia

3.2.3. Algoritmo y Pseudocódigo para el control de desplazamiento de los módulos MECABOT 2 hasta el 15 por los puntos de bordeo y de acercamiento hasta las ruedas deseadas del módulo a acoplar

Para ejecutar el desplazamiento de los módulos "esclavos" 2 hasta el 15, que como se mencionó anteriormente se conectan entre sí para conformar las arquitecturas planteadas, se definió la estructura secuencial como se muestra en la Figura 63, la cual parte de la ya establecida para el módulo maestro No. 1 (Figura 55), adicionando 5 estados y modificando 1, registrados en la variable "*flag*", estructura que simboliza lo definido en la secuencia de programación del controlador de los módulos esclavos en la función "*desplazamiento_modulo*", la cual se encuentra detallada en el **Anexo *6*** y se explican los cambios presentados con respecto al control de desplazamiento del módulo principal[10] en la lógica de programación a continuación:

- El desplazamiento individual del módulo de acople (esclavo) inicia en la marca "*flag=-2*", con el cálculo y obtención de los puntos de bordeo y de acercamiento del módulo a acoplar (Figura 62), seleccionando el punto con menor distancia como primer objetivo a llegar, mediante la ejecución de la función "*puntos_bordeo*" (Código 9), puesta en detalle en el ***Anexo 6***.
- Con la marca "*flag=-1*", se asigna a las variables "$X_{f,}$" y "Z_f" (Figura 54), las coordenadas obtenidas del punto de bordeo y de acercamiento al que se

[9] Para conocer el detalle acerca del control del protocolo de comunicación, ver sección 3.3.2., página 97.
[10] Presentados en la sección 3.1.5., página 81.

debe llegar, esta operación se hace para cada punto siguiente al que se valla llegando, mediante la ejecución de la función "*coord_ptobordeo*", puesta en detalle en el ***Anexo 6***.

- Desde la marca "*flag=0*" hasta "*flag=5*" no presentan cambios al control de desplazamiento planteado para el módulo principal (Figura 55).
- Con la marca "*flag=6*", llegado el módulo a las coordenadas finales, se calcula la orientación hacia el próximo punto de bordeo o acercamiento, según la secuencia mostrada en la Tabla *9*, que guía desde el punto actual la orientación del próximo punto de llegada hasta la rueda deseada; mediante la ejecución de la función "*orientacion_ptobordeo*", puesta en detalle en el ***Anexo 6***.

Tabla 9. Secuencia de desplazamiento entre puntos de bordeo y acercamiento para acoplarse a la rueda deseada por parte del módulo de acople. Elaboración propia

	Punto Mínimo (actual)	Rueda Posterior	Rueda Izquierda	Rueda Derecha
Puntos de Bordeo	1	5	8	4
	2	1	8	1
	3	4	4	11
	4	5	1	11
Puntos Rueda posterior	5	6	1	4
	6	7	No aplica	No aplica
	7	Llegada	No aplica	No aplica
Puntos Rueda Izquierda	8	1	9	1
	9	No aplica	10	No aplica
	10	No aplica	Llegada	No aplica
Puntos Rueda Derecha	11	4	4	12
	12	No aplica	No aplica	13
	13	No aplica	No aplica	Llegada

- Con la marca "*flag=7*", se procede a girar el módulo con el ángulo especifico en alineación al siguiente punto (de bordeo o de acercamiento), posteriormente se ejecuta el análisis para escoger el próximo punto, según la secuencia mostrada en la Tabla *9*, que guía desde el punto de desplazamiento actual hasta el punto de llegada final a cada rueda deseada; la marca "*flag*" se iguala a -1 (uno negativo), para continuar con el desplazamiento del módulo de acople a las nuevas coordenadas finales asignadas, que permita llegar hasta el último punto de aproximación de la rueda deseada del módulo a acoplar, para lo cual la marca "*flag*" se iguala a

8 (ocho); algoritmo que se ejecuta mediante la ejecución de la función "*proximo_ptobordeo*", puesta en detalle en el ***Anexo 6***.

- Con la marca "*flag=8*", llegado el módulo a las coordenadas finales de acercamiento a la rueda deseada, se procede a acercar el módulo hasta que los dos ejes de las ruedas a ser acopladas queden alineadas y en contacto (definido por el desplazamiento del módulo de acople de 16 pasos al frente).
- Con la marca "*flag=9*", se ejecuta la comunicación[11] con el módulo maestro para avisar que ya llegó a la rueda deseada del módulo a acoplar, para finalmente realizar la conexión a la rueda mencionada mediante la función bloqueo del conector (Código 8).

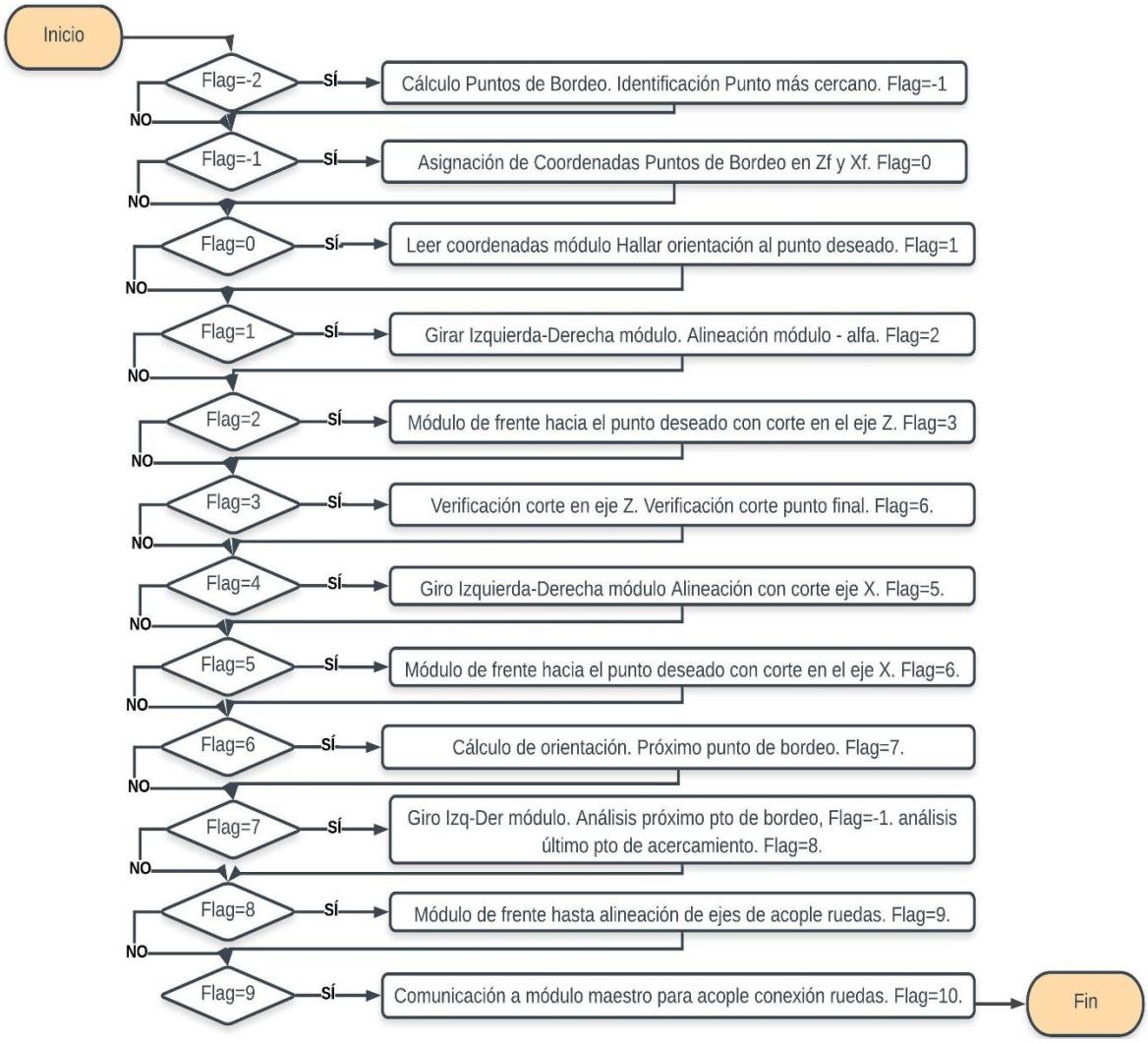


Figura 63. Estructura secuencial para ejecutar el desplazamiento del módulo de acople hasta la rueda deseada del módulo a acoplar. Elaboración propia

[11] Para conocer el detalle acerca del control del protocolo de comunicación, ver sección 3.3.2., página 97.

3.3. Diseño de Control y Protocolo de Comunicación entre Módulos

3.3.1. Dispositivos EMITTER y RECEIVER para controlar comunicación de cada módulo MECABOT

Como se definió en la Tabla 3, se muestran los dispositivos adicionales a los utilizados para el desplazamiento de los módulos, pero que complementan el uso del mismo para lograr el control integrado del sistema, entre los cuales se señalan con números 12 y 13, los elementos Emitter y Receiver respectivamente, empleados para emular la comunicación entre los módulos, con el fin de efectuar sincrónicamente la conexión y desconexión entre módulos en la ejecución de las rutinas de armado de las arquitecturas planteadas para el presente proyecto.

El dispositivo EMITTER en el software WEBOTS, es utilizado para modelar emisores de señal de radio, serial o infra-roja. Un nodo EMITTER debe ser agregado en el folder *children* de un robot o un supervisor. El emisor puede enviar datos pero éste no puede ser utilizado para recibirlos. Para simular una comunicación unidireccional entre dos robots, un robot debe contar con un dispositivo EMITTER mientras que el otro robot debe tener un dispositivo RECEIVER. Tener en cuenta que los mensajes nunca deben ser transmitidos desde un robot a sí mismo. [71]

Por otra parte, el dispositivo RECEIVER en el software WEBOTS, es utilizado para modelar receptores de señal de radio, serial o infra-roja. Un nodo RECEIVER debe ser agregado en el folder *children* de un robot o un supervisor. El receptor puede recibir datos pero éste no puede ser utilizado para enviarlos. Para lograr una comunicación bidireccional, un robot necesita contar con ambos dispositivos EMITTER y RECEIVER en el folder de cada módulo. [72]

En la Figura 64, se muestran resaltados con el recuadro de color “verde”, los dispositivos EMITTER y RECEIVER incluidos para cada módulo en el software WEBOTS, los cuales para efectos de control en la programación han sido denominados como “*emitter*” y “*receiver*” respectivamente.

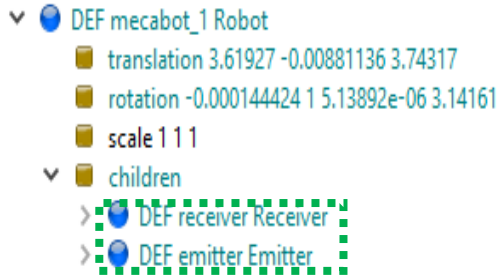


Figura 64. Jerarquía Nodo Robot MECABOT, identificación de dispositivos RECEIVER y EMITTER – Software WEBOTS. Elaboración propia

Para el manejo del emisor en el Software WEBOTS, se presenta en la Figura 65, la configuración disponible para el dispositivo *EMITTER*, previamente definida por el Software, para tener en cuenta al momento de realizar los ajustes pertinentes, puesto que hay disponible una serie de campos que limitan la capacidad del dispositivo y que se ven reflejados en el comportamiento del sistema durante la acción de la comunicación en el momento de la simulación.

```
Emitter {
  SFString type       "radio"   # {"radio", "serial", "infra-red"}
  SFFloat  range      -1        # {-1, [0, inf)}
  SFFloat  maxRange   -1        # {-1, [0, inf)}
  SFFloat  aperture   -1        # {-1 ,[0, 2*pi]}
  SFInt32  channel    0         # [0, inf)
  SFInt32  baudRate   -1        # {-1, [0, inf)}
  SFInt32  byteSize   8         # [8, inf)
  SFInt32  bufferSize -1        # {-1, [0, inf)}
}
```

Figura 65. Configuración disponible del dispositivo EMITTER para el control de comunicación entre módulos. [71]

Las características que describen el comportamiento de los campos disponibles en el dispositivo EMITTER son [71]:

- *Type*: tipo de señales "radio", "serial" o "infra-rojo", las señales de “radio” (definida por defecto) y “serial” son transmitidas sin tener en cuenta los obstáculos. Señales de tipo “infra-rojo”, sin embargo toman el potencial de obstáculos entre el emisor y receptor en cuenta. Cualquier objeto (solido, robots, etc) con un límite definido es un potencial obstáculo para una comunicación “infra-roja”.
- *Range*: radio de la esfera de emisión (en metros). Un receptor solo puede recibir un mensaje si éste está localizado dentro de la esfera de emisión. El valor de -1 (por defecto) es considerado como un rango con valor infinito.
- *maxRange*: define el máximo valor permitido por *Range*. Este campo define el máximo valor que puede ser ajustado usando la función definida por el Software Webots *wb_emitter_set_range*. El valor de -1 (por defecto) es considerado como un rango con valor infinito.
- *Aperture*: Angulo de apertura del cono de emisión (en radianes); para señales de “infra-rojo” únicamente. Un emisor “infra-rojo” solo puede enviar información al receptor si se encuentra ubicado dentro de su cono de emisión (Figura 66).

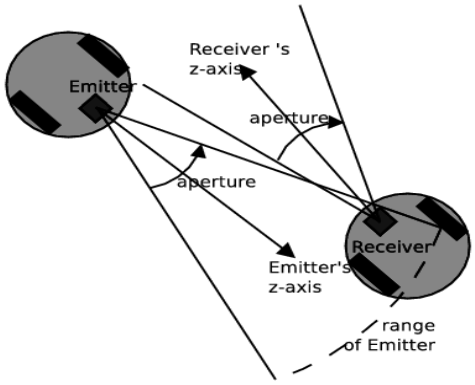


Figura 66. Ejemplo de apertura y rango para un Emisor/Receptor "infra-rojo". [71]

- *Channel*: Canal de transmisión. Éste es un número de identificación para un emisor "infra-rojo" o una frecuencia para un emisor de "radio". Normalmente un receptor debe usar el mismo canal como el de un emisor para recibir los datos emitidos. Sin embargo, el canal especial -1 permite difundir mensajes a todos los canales. Canal 0 (definido por defecto) está reservado para la comunicación con una entrada física. Para comunicación entre robots, se recomienda utilizar números positivos.
- *baudRate*: es la velocidad de transmisión en la comunicación expresada en número de bits por segundo. Una velocidad de transmisión de valor -1 (definido por defecto) se considera infinito y hace que los datos se transmitan inmediatamente (dentro de un paso de tiempo básico) desde el emisor al receptor.
- *byteSize*: el tamaño del byte es el número de bits requerido para transmitir un byte de información. Éste es usualmente de 8 (por defecto), pero puede ser más si se requiere.
- *bufferSize*: especifica el tamaño (en bytes) del buffer de transmisión. El número total de bytes en los paquetes en cola en el emisor no puede exceder este número. Un valor -1 (definido por defecto) es considerado como ilimitado el tamaño de buffer.

Con respecto a los parámetros disponibles por el Software WEBOTS para el manejo del dispositivo EMITTER (Figura 65), las condiciones específicas para el funcionamiento del dispositivo para la comunicación entre módulos (Figura 66), se establecieron las características propias de los emisores de cada módulo (el maestro y los esclavos) conforme a la configuración mostrada en la Figura 67, el cual a manera de ejemplo muestra la configuración de los dispositivos EMITTER para el módulo 1 (recuadro color "negro") con el parámetro "*channel" o* canal -1 (uno negativo) para comunicarse a todos los módulos esclavos, mientras que para los módulos 2 al 15 (recuadro color "rojo") con el parámetro "*channel"* o canal 1 (uno) para que se comuniquen todos los módulos con el módulo maestro.

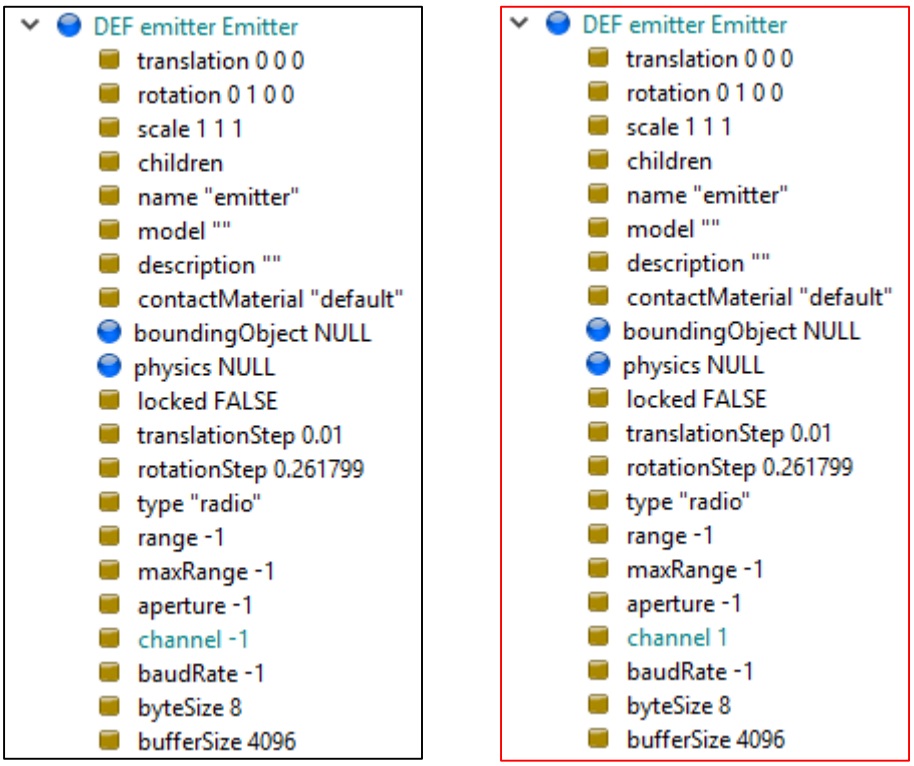


Figura 67. Configuración del dispositivo EMITTER para el control de comunicación entre módulos, módulo maestro (izquierda) y esclavos (derecha) – Software WEBOTS. Elaboración propia

Por otra parte, para el manejo del receptor en el Software WEBOTS, se presenta en la Figura 68, la configuración disponible para el dispositivo *RECEIVER*, previamente definida por el Software, para tener en cuenta al momento de realizar los ajustes pertinentes, puesto que hay disponible una serie de campos que limitan la capacidad del dispositivo y que se ven reflejados en el comportamiento del sistema durante la acción de la comunicación en el momento de la simulación.

```
Receiver {
  SFString type                "radio"   # {"radio", "serial", "infra-red"}
  SFFloat  aperture            -1        # {-1, [0, 2*pi]}
  SFInt32  channel             0         # [-1, inf)
  SFInt32  baudRate            -1        # {-1, [0, inf)}
  SFInt32  byteSize            8         # [8, inf)
  SFInt32  bufferSize          -1        # {-1, [0, inf)}
  SFFloat  signalStrengthNoise 0         # [0, inf)
  SFFloat  directionNoise      0         # [0, inf)
}
```

Figura 68. Configuración disponible del dispositivo RECEIVER para el control de comunicación entre módulos. [72]

Las características que describen el comportamiento de los campos disponibles en el dispositivo RECEIVER son [72]:

- *Type*: tipo de señales "radio", "serial" o "infra-rojo", las señales de “radio” (definida por defecto) y “serial” son transmitidas sin tener en cuenta los obstáculos. Señales de tipo “infra-rojo”, sin embargo toman el potencial de obstáculos entre el emisor y receptor en cuenta. Cualquier objeto (solido, robots, etc) con un límite definido es un potencial obstáculo entre una comunicación “infra-roja”. La estructura del propio robot emisor o receptor no bloqueará una transmisión "infrarroja". Actualmente, no hay diferencia de implementación entre los tipos "radio" y "serie".
- *Aperture*: Angulo de apertura del cono de recepción (en radianes); para señales de “infra-rojo” únicamente. Un emisor “infra-rojo” solo puede recibir información desde emisores si se encuentra ubicado dentro de su cono de recepción.
- *Channel*: Canal de recepción. El valor es un número de identificación para un receptor “infra-rojo” o una frecuencia para un receptor de “radio”. Normalmente un receptor debe usar el mismo canal como el de un emisor para recibir los datos emitidos. Sin embargo, el canal especial -1 permite al receptor escuchar todos los canales.
- *baudRate*: es la velocidad de transmisión en la comunicación expresada en número de bits por segundo. Éste debería ser la misma velocidad del emisor.
- *byteSize*: el tamaño del byte es el número de bits utilizados para representar un byte de los datos transmitidos (generalmente 8, pero puede ser más si se usan bits de control). Debe ser del mismo tamaño que el tamaño del byte del emisor.
- *bufferSize*: especifica el tamaño (en bytes) del buffer de recepción. El tamaño de los datos recibidos no debe exceder el tamaño del búfer en ningún momento, de lo contrario los datos pueden perderse. Un valor -1 (definido por defecto) es considerado como ilimitado el tamaño de buffer.

Con respecto a los parámetros disponibles por el Software WEBOTS para el manejo del dispositivo RECEIVER (Figura 68), las condiciones específicas para el funcionamiento del dispositivo para la comunicación entre módulos, se establecieron las características propias de los receptores de cada módulo (el maestro y los esclavos) conforme a la configuración mostrada en la Figura 69, el cual a manera de ejemplo muestra la configuración de los dispositivos RECEIVER para el módulo 1 (recuadro color “negro”) con el parámetro “*channel” o* canal 1 (uno) para recibir comunicación de los módulos esclavos que se comuniquen con el maestro, igualmente que para los módulos 2 al 15 (recuadro color “rojo”) con el parámetro “*channel”* 2 al 15 sucesivamente para cada uno, a fin de recibir la comunicación del módulo maestro individualmente o en conjunto si el módulo principal así lo ordena.

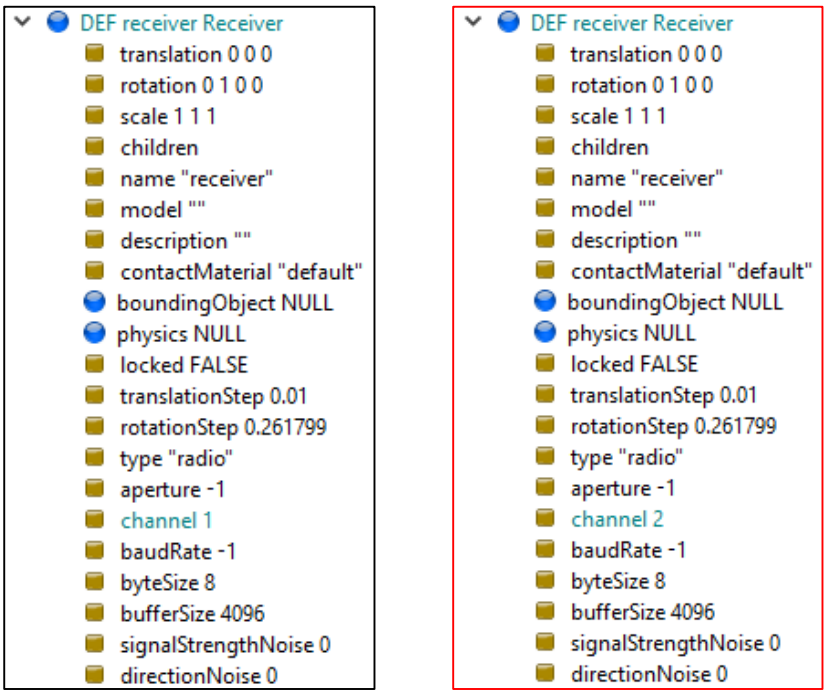


Figura 69. Configuración del dispositivo RECEIVER para el control de comunicación entre módulos, módulo maestro (izquierda) y esclavos (derecha) – Software WEBOTS. Elaboración propia

Se prosiguió a la asignación de los dispositivos señalados en la Figura 67 y Figura 69 en la interfaz del aplicativo para la programación del controlador, para cada módulo en el software Webots, como se aprecia resaltado en el Código 10, se muestra el comando de código específico para la definición de las variables "*emisor*" para el dispositivo "emitter", como también "*receptor*" para el dispositivo "receiver" (variables declaradas en **Anexo *1***), de forma que se enlacen las variables físicas con las de control. Adicionalmente, se aprecia que el dispositivo receptor es habilitado para lectura de datos durante la ejecución del programa con la función definida por el Software WEBOTS "*wb_receiver_enable*" para la variable "*receptor*" previamente declarada.

```
// obtener emisor
emisor=wb_robot_get_device("emitter");
/* if the channel is not the right one, change it */
channel = wb_emitter_get_channel(emisor);
if (channel != COMMUNICATION_CHANNEL_EMITTER) {
  wb_emitter_set_channel(emisor, COMMUNICATION_CHANNEL_EMITTER);
}
// obtener receptor
receptor=wb_robot_get_device("receiver");
/* if the channel is not the right one, change it */
channel = wb_receiver_get_channel(receptor);
if (channel != COMMUNICATION_CHANNEL_RECEIVER) {
  wb_receiver_set_channel(receptor, COMMUNICATION_CHANNEL_RECEIVER);
}
wb_receiver_enable(receptor, TIME_STEP);
```

Código 10. Definición de variables para controlar los dispositivos EMITTER y RECEIVER – Software WEBOTS. Elaboración propia

3.3.2. Protocolo de comunicación entre módulos MECABOT con base a sistema maestro – esclavo

Se definió el módulo 1 como el maestro puesto que es el que establece la comunicación para ejecutar la secuencia de armado de las arquitecturas definidas en el presente proyecto, registra la posición y el almacenamiento de datos de los demás los módulos; como también se definieron los módulos 2 al 15 como los esclavos, puesto que es a través de alguna orden particular impartida por el módulo principal de manera individual a alguno de ellos, que realizan alguna acción, también por una orden general para realizar una acción en conjunto para los movimientos sincronizados del sistema en la ejecución de alguna arquitectura.

Para la comunicación entre módulos a través de los dispositivos integrados, se realizó una sincronización de un sistema de comunicación asíncrono, de forma que se organizara el envío de datos entre el módulo 1, definido como el módulo maestro y los demás módulos del 2 al 15, definidos como esclavos; estableciendo el módulo maestro con el canal -1 (uno negativo) en el emisor (ver parámetro en Figura 65) de forma que pueda enviar información a todos los módulos esclavos al mismo tiempo, los cuales se les asigno el número *n* de canal en el receptor (ver parámetro en Figura 68), siendo *n* el asignado como mismo número para cada módulo del 2 al 15, por otra parte, los módulos esclavos solo pueden enviar información al módulo maestro, mediante el establecimiento del canal 1 (uno) en el emisor de éstos, como se muestra en la Figura 70.

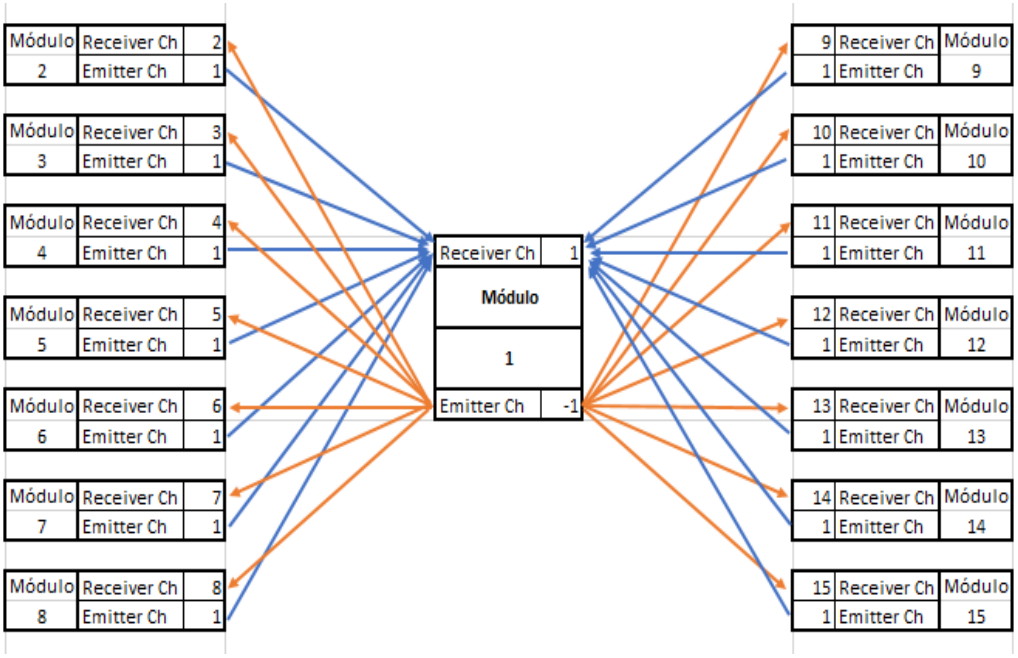


Figura 70. Esquema de sincronización en la comunicación entre módulos maestro (el 1) y esclavos (del 2 al 15) – Software WEBOTS. Elaboración propia

Para controlar la comunicación entre módulos a través de los dispositivos integrados (Figura 70), la secuencia de programación para emitir los datos deseados, se realizó mediante el uso de la función definida por el Software WEBOTS *wb_emitter_send* para enviar la trama de datos desde el módulo principal a los módulos esclavos y viceversa, junto con la función definida por el Software WEBOTS *wb_receiver_get_data* para recibir la trama de datos enviada desde el módulo principal a los módulos esclavos y viceversa.

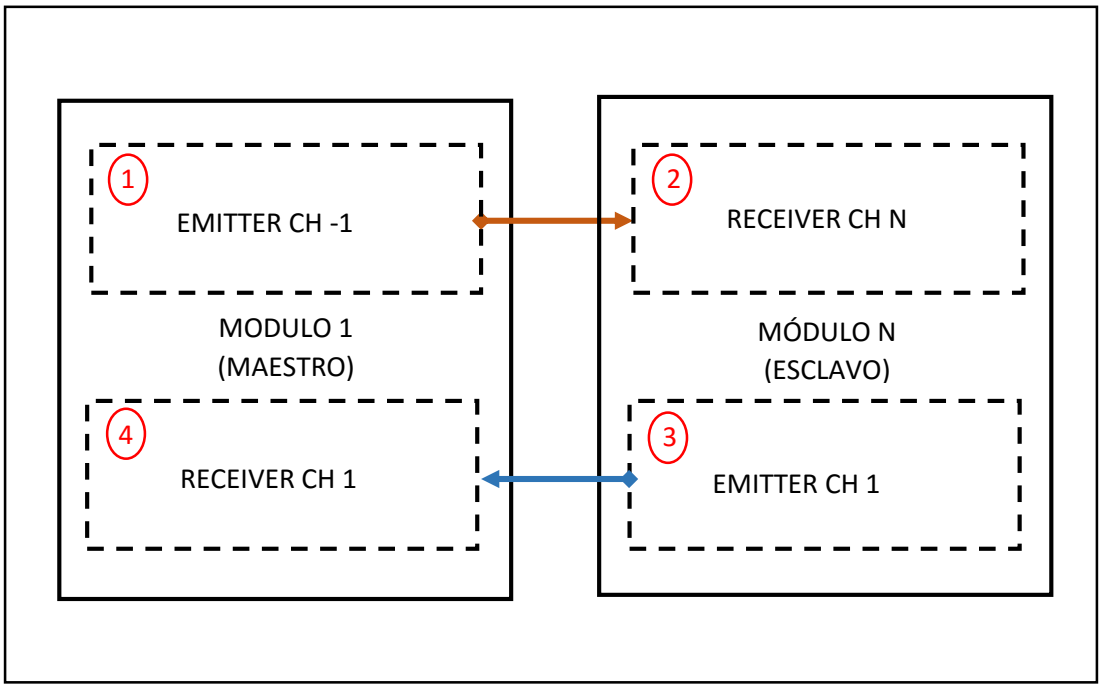


Figura 71. Secuencia de comunicación entre módulos maestro (el 1) y esclavos (del 2 al 15) para la sincronización del sistema – Software WEBOTS. Elaboración propia

La secuencia de programación para controlar la comunicación entre módulos, se realiza en cuatro momentos diferentes, como se señalan en la Figura 71, el primero es enviar la información desde el módulo maestro hacia los esclavos, el segundo para recibir la información en los módulos esclavos enviada por el módulo maestro, el tercero para enviar la información desde el módulo esclavo hacia el maestro y cuarto para recibir la información en el módulo maestro enviada desde los esclavos, como se detalla a continuación:

3.3.2.1. Envío de información desde el módulo 1 maestro a los módulos esclavos del 2 al 15

Para el envío de información desde el módulo 1 maestro hasta los demás módulos, se generó una función de nombre "*comunicacion_modulos*", la cual se encuentra

detallada en el **Anexo 7**, en donde se definió un conjunto de 6 datos a ser enviados de tipo entero y registrados en la variable tipo *int array* "*datosdenvio*" que representan los siguientes datos mostrados en la Tabla 10:

Tabla 10. Información registrada en la variable "datosdenvio" para el envío de información desde el módulo maestro a los módulos esclavos. Elaboración propia

No. Dato	Variable *"datosdenvio[P]"*	Información a ser enviada
1	P= modulo_accion	Módulo a comunicar para que ejecute la acción.
2	P= datosmodulos[modulo_conectar][1]	Coordenadas en "*x*" del módulo al que debe acoplarse
3	P= datosmodulos[modulo_conectar][2]	Coordenadas en "*y*" del módulo al que debe acoplarse
4	P= datosmodulos[modulo_conectar][3]	Posición angular "Ang" del módulo al que debe acoplarse
5	P= com_forma	Ver Tabla 11
6	P= arquitectura	Tipo de arquitectura a ejecutar, siendo 1 para Oruga y 2 para Hexápodo

Conforme los datos mostrados en la Tabla 10, se puede apreciar que el comando principal que encomienda el módulo maestro al esclavo que acción realizar, es la impuesta por el dato de envío No. 5, registrado en la variable "*com_forma*", siendo esta variable, como se puede apreciar en la Tabla 11, la que envía la instrucción particular al módulo esclavo (definido por la variable "*modulo_accion*" en dato de envío No. 1 - Tabla 10) por ejemplo, requerir la referencia de las coordenadas de éste, o a que rueda acoplarse, subir o bajar el módulo, entre otras; como también, el envío de alguna instrucción general para el conjunto de módulos que conforman el sistema, en ejecución de alguna arquitectura (dependiendo de la variable "*arquitectura"* en dato de envío No. 6 - Tabla 10), así como para el desplazamiento coordinado de los módulos, en el momento de estar configurada una de éstas morfologías.

Tabla 11. Información registrada en la variable "com_forma" para el envío de información de "datosdenvio", desde el módulo maestro a los módulos esclavos. Elaboración propia

Valor de "com_forma"	Información que describe la instrucción a ser enviada desde el módulo maestro 1 hacia los módulos esclavos 2 al 15 para ejecutar la acción
0	Pedido de coordenadas al módulo esclavo
1	Conectarse a la rueda posterior del módulo a acoplar
2	Conectarse a la rueda izquierda del módulo a acoplar
3	Conectarse a la rueda derecha del módulo a acoplar

4	Ejecutar movimiento oruga de frente
5	Levantar módulo
6	Bajar módulo
7	Alinear ejes de ruedas izquierda-derecha para ensamble / desensamble
8	Voltear módulos acoplados a ruedas izquierda y derecha
9	Despegar y mover a atrás el módulo
10	Colocar el Hexápodo de pie
11	Hexápodo adelante: 2 patas izquierda adelante y 1 derecha adelante, centro 1 pata izquierda y 2 patas derecha
12	Hexápodo adelante: 1 pata izquierda adelante y 2 derecha adelante, centro 2 patas izquierda y 1 pata derecha
13	Hexápodo giro hacia la derecha: 2 patas izquierda adelante y 1 derecha atrás, centro 1 pata izquierda y 2 patas derecha
14	Hexápodo giro hacia la derecha: 1 pata izquierda adelante y 2 derecha atrás, centro 2 patas izquierda y 1 pata derecha
15	Hexápodo giro hacia la izquierda: 2 patas izquierda atrás y 1 derecha adelante, centro 1 pata izquierda y 2 patas derecha
16	Hexápodo giro hacia la izquierda: 1 pata izquierda adelante y 2 derecha atrás, centro 2 patas izquierda y 1 pata derecha
17	Hexápodo en reposo: 1 pata izquierda y 2 derecha
18	Hexápodo en reposo: 2 patas izquierda y 1 derecha
19	Hexápodo abajo (acostar – posición de reposo)
20	Oruga abajo (acostar – posición de reposo)
21	Desacoplar módulo de la arquitectura Hexápodo
22	Desacoplar módulo de la arquitectura Oruga

Como se aprecia en el Código 11, el dispositivo EMITTER de cada módulo, señalado con el número doce (12) en la Tabla 3, denominado "*emisor*" (Código 10), es el utilizado para transmitir información en la comunicación entre módulos, ejecutándose las líneas de código dispuestas para el envío de datos mediante la función *wb_emitter_send*, el cual para efectos de la simulación en el Software WEBOTS, ejecuta la acción de enviar el mensaje registrado en el apuntador de la variable "*message*", que define la dirección de memoria de la variable "*datosdenvio*" definida anteriormente en la Tabla 10. El algoritmo mencionado se encuentra mostrado en detalle en la función "*comunicacion_modulos*" mostrada en el **Anexo 7**.

```
const int *message = &datosdenvio[0];
wb_emitter_send(emisor, message, 6 * sizeof(int));
```

Código 11. Secuencia de programación para controlar el elemento EMITTER en la comunicación entre módulos – Software WEBOTS. Elaboración propia

3.3.2.2. Recepción de la información enviada por el módulo maestro 1 a los módulos esclavos del 2 al 15

Para la recepción de información enviada por el módulo maestro 1 hacia los demás módulos esclavos, se generó una función de nombre "*recepcion_informacion_modulo*", detallada en el **Anexo 8**, en la cual se definió un conjunto de 6 datos a ser recibidos, de la misma forma que éstos fueron definidos para el envío desde el módulo 1, datos de tipo entero que son enviados previamente por el módulo maestro y registrados en la variable tipo *int array* "*datosderecibo*", que representan los siguientes datos mostrados en la Tabla 12:

Tabla 12. Información registrada en la variable "datosderecibo" para la recepción de información enviada desde el módulo maestro a los módulos esclavos. Elaboración propia

No. Dato	**Envío desde módulo Maestro** Variable *"datosderecibo[P]"*	**Recepción desde módulos esclavo** Variable *"datosderecibo[P]"*	Información enviada - recibida
1	P= modulo_accion	P=0	Módulo a comunicar para que ejecute la acción.
2	P= datosmodulos[modulo_conectar][1]	P=1	Coordenadas en "*x*" del módulo al que debe acoplarse
3	P= datosmodulos[modulo_conectar][2]	P=2	Coordenadas en "*y*" del módulo al que debe acoplarse
4	P= datosmodulos[modulo_conectar][3]	P=3	Posición angular "Ang" del módulo al que debe acoplarse
5	P= com_forma	P=4	Ver Tabla 11
6	P= arquitectura	P= 5	Tipo de arquitectura a ejecutar, siendo 1 para Oruga y 2 para Hexápodo

Como se aprecia en el Código 12, el dispositivo RECEIVER de cada módulo, señalado con el número 13 (trece) en la Tabla 3, denominado "*receptor*" (Código 10), es el utilizado para recibir información en la comunicación entre módulos, ejecutándose las líneas de código dispuestas para la recepción de datos mediante la función *wb_receiver_get_data*, el cual para efectos de la simulación en el Software WEBOTS, ejecuta la acción de recibir el mensaje registrado en el apuntador de la variable "buffer", para registrar la información recibida en la variable

"*datosderecibo*" definida anteriormente en la Tabla 12. El algoritmo mencionado se encuentra mostrado en detalle en la función "*comunicacion_modulos*" mostrada en el **Anexo 8**.

```
const int *buffer = wb receiver get data(receptor);

            for (i=0;i<6;i++){
              datosderecibo[i]=*(buffer+i);
            }
```

Código 12. Secuencia de programación para controlar el elemento RECEIVER en la comunicación entre módulos – Software WEBOTS. Elaboración propia

Nótese que la información enviada desde el módulo principal, según lo mostrado en la Tabla 10, se recibe en la misma cantidad y modo en los módulos esclavos la información enviada por el módulo maestro Tabla 12; para esto, el módulo 1 envía la información por el canal -1 (uno negativo) para que le llegue la información a todos los módulos y sea a través de la lectura del módulo a ejecutar la acción (variable "*modulo_accion*" en dato de envío No. 1 - Tabla 10), que entre los módulos esclavos del 2 al 15, se determine cual debe realizar la acción a ejecutar de acuerdo a la enumeración dada a cada módulo; si por otra parte es una instrucción que desde el módulo principal se desea enviar al mismo tiempo a todos los módulos para ejecutar desplazamientos en conjunto, se envía en la variable "*modulo_accion*" el número 1 (uno) para indicar instrucciones a todos al mismo tiempo.

3.3.2.3. Envío de información desde los módulos esclavos del 2 al 15 hacia el módulo maestro 1

Una vez recibida la instrucción desde el módulo maestro y realizada la acción encomendada al módulo esclavo, se envía la información hacia al módulo 1 para que éste pueda continuar con la secuencia de ejecución de las arquitecturas con los demás módulos. Para el envío de información desde los módulos esclavos del 2 al 15 hacia el módulo maestro 1, se generó una función de nombre "*comunicacion_modulos*", la cual se encuentra detallada en el **Anexo 8**, en donde se definió un conjunto de 6 datos a ser enviados al módulo principal de tipo entero y registrados en la variable tipo *int array* "*datosdenvio*" (Código 13), siendo éstos en su orden, el número de identificación del módulo esclavo, su posición actual en "*x*" y en "*y*", su ángulo de orientación actual, la instrucción recibida por el módulo 1 en la variable "*com_forma*" y la arquitectura definida.

```
int datosdenvio[6] = {id,posxz[0], posxz[1],angulo,datosderecibo[4],datosderecibo[5]};
```

Código 13. Variable transmitida por el elemento EMITTER de los módulos esclavos en la comunicación entre módulos – Software WEBOTS. Elaboración propia

3.3.2.4. Recepción de la información enviada por los módulos esclavos del 2 al 15 hacia el módulo maestro 1

Para la recepción de información enviada por los módulos esclavos del 2 al 15 hacia el módulo maestro, se generó una función de nombre "*recepcion_informacion_modulo*", detallada en el **Anexo 7**, en la cual se definió un conjunto de 6 datos a ser recibidos, de la misma forma que éstos fueron definidos para el envío desde los módulo esclavos (Código 14); datos de tipo entero que son enviados previamente por éstos módulos y registrados en la variable tipo *int array* "*datosderecibo*", variable cuyos datos se utilizan para almacenarse en la variable "*datosmodulos*", con el fin de registrar en un arreglo bidireccional la información actualizada de todos los módulos (esclavos), como parámetros de referencia que son utilizados para nuevamente reenviar a los otros, para propósitos de control en la ejecución acoples para la conformación de las arquitecturas y sus desplazamientos.

```
for (info_paq=0;info_paq<6;info_paq++){
  datosderecibo[info_paq]=*(buffer+info_paq);
  datosmodulos[*(buffer+0)][info_paq]=datosderecibo[info_paq];
}
```

Código 14. Variable registrada para recibir datos del elemento RECEIVER en el módulo maestro en la comunicación entre módulos – Software WEBOTS. Elaboración propia

3.3.3. Integración código de programación para la ejecución del protocolo de comunicación entre módulos MECABOT con los otros dispositivos integrados

La sincronía en la comunicación de cada módulo, se logra ejecutando la recepción y el envío de información en dos momentos específicos, el primero se hace mediante el llamado de la función "*recepcion_informacion_modulo*", en la instrucción "*while*" dentro de la función principal (como ejemplo ver la función *main* del módulo 1 en el **Anexo *3***), la cual repetitivamente va ejecutando las líneas de comandos en todos los módulos para verificar la entrada de algún paquete de información dirigida a algún módulo en específico desde el módulo maestro a los esclavos o viceversa.

Por otra parte, el segundo momento, enfocado para el envío de información desde el módulo maestro a los esclavos o viceversa, se realiza mediante el llamado de la función "*comunicacion_modulos*", empleada para enviar la información desde el módulo principal a los esclavos para la ejecución de las instrucciones específicas por cada módulo (Tabla 10); como también desde algún módulo esclavo al maestro,

principalmente utilizado para la conexión entre módulos, como se indica en el algoritmo mostrado en la Figura 63, en la marca "*flag=9*" de la función "*desplazamiento_modulo*" (detallada en el ***Anexo 6***), que ejecuta la acción de comunicarse con el módulo principal para dar la señal que el módulo de acople ya llegó a la rueda deseada del módulo a acoplar y ejecutar la acción de bloqueo del conector (Código 15), para posteriormente permitirle al módulo principal continuar con la secuencia de armado de las arquitecturas con los demás módulos.

```
//para comunicarse con el otro modulo (modulo 1)
else if(flag==9){comunicacion_modulos();wb_connector_lock(conector_pivote);flag=10;printf("listo 10: envio de datos al modulo 1\n");}
```

Código 15. Secuencia de programación para enviar la información desde el módulo esclavo al maestro en la comunicación entre módulos – Software WEBOTS. Elaboración propia

3.4. Conclusiones del Capítulo

En este capítulo se presentó el diseño de las herramientas necesarias para el control de los dispositivos asociados a cada módulo, junto con el desarrollo del algoritmo de control de los elementos que lo componen, mediante el uso del software WEBOTS para su simulación, teniendo en cuenta los trabajos previamente elaborados en el grupo DAVINCI de la Universidad Militar Nueva Granada.

Se desarrollaron las diferentes herramientas para el uso óptimo de cada módulo, con el empleo y el control de los elementos asociados, siendo éstos, los motores que accionan los mecanismos de movimiento individual, los elementos de conexión que son utilizados para el acoplamiento entre módulos, como también los dispositivos de envío y recepción de información, implementando varias metodologías para la integración de los mismos, de forma que se permitiera proporcionar a los módulos la capacidad de desplazamiento, de conexión y de comunicación con los demás, de forma autónoma, integrada y sincronizada para proceder a ejecutar las arquitecturas propuestas.

CAPÍTULO IV.

DISEÑO Y DESARROLLO DE ALGORITMO PARA EL ARMADO Y DESARMADO DE ARQUITECTURAS ORUGA Y HEXÁPODO PARA SISTEMA ROBÓTICO MODULAR MECABOT 4

En este capítulo se presentan las diferentes estrategias de armado planteadas para la obtención de las arquitecturas oruga y hexápodo, junto con las secuencias de movimiento individual para el desplazamiento en conjunto de cada arquitectura, como el procedimiento de desarmado de cada una y reubicación individual de cada módulo para quedar en disposición de ejecutar cualquiera de las dos arquitecturas planteadas para el presente proyecto.

4.1. Estrategia de Armado Arquitectura ORUGA

Conforme a las herramientas brindadas para el desplazamiento individual de los módulos, la conexión y la comunicación entre módulos, mediante el control de los dispositivos incorporados a cada uno (Tabla 2 y Tabla 3), establecidos los protocolos de envío y recepción de información, se crearon las estrategias de armado automático de la primer arquitectura planteada para el presente proyecto, esta es la oruga.

4.1.1. Secuencia de armado arquitectura ORUGA

La arquitectura oruga[12], como su nombre lo indica consiste en ubicar y conectar los módulos en línea recta, uno tras el otro, siendo acoplados éstos por los conectores de la rueda pivote al de la rueda posterior del otro (señalados con los números 6 y 9 en Tabla 2 respectivamente), la secuencia de armado es liderada por el módulo 1, que además de ser considerado el módulo maestro, se ubica delante de los demás módulos en la conformación de la arquitectura oruga; en la Figura 72 se muestra la organización de los módulos para su configuración, de acuerdo al número que le fue asignado a cada módulo para su identificación, la línea roja representa el motor de la rueda pivote del módulo, parte frente del mismo (número 2 en Tabla 2).

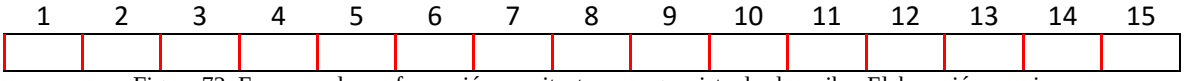


Figura 72. Esquema de conformación arquitectura oruga, vista desde arriba. Elaboración propia

4.1.2. Algoritmo y Pseudocódigo para controlar el armado arquitectura ORUGA

Para la ejecución del algoritmo para el armado de la arquitectura oruga, se acude al uso de las herramientas formadas anteriormente, tales como el control de posicionamiento y de orientación de cada módulo (Código 3 y Figura 55), el control de conexión de las ruedas entre módulos (Código 8), el control de desplazamiento del módulo de acople a través de los puntos de bordeo del módulo a acoplar (Figura 63) y la ejecución del protocolo de comunicación entre el módulo maestro 1 y los módulos esclavos del 2 al 15 (Figura 71).

- Algoritmo de control para armado de arquitectura oruga en el módulo 1

Para ejecutar el armado de la arquitectura oruga, para el módulo maestro 1 se definió la estructura secuencial como se muestra en la Figura 73, registrándose los estados en la variable "*flag*", con la cadencia en la ejecución de la siguiente instrucción dependiendo de la variable "*datosderecibo[0]*", que indica cuando se recibe la señal en el módulo maestro de que el módulo esclavo en acción ha terminado la orden y devuelto la información de acuerdo al protocolo de comunicación (Código 14). La estructura que simboliza lo definido en la secuencia de programación del controlador del módulo 1, se generó en la función "*arquitectura_oruga*", la cual se encuentra detallada en el **Anexo 9**.

[12] Para conocer el detalle de desarrollo de la arquitectura oruga, ver el trabajo realizado por las Ingenieras Paola Lancheros y Laura Sanabria [62].

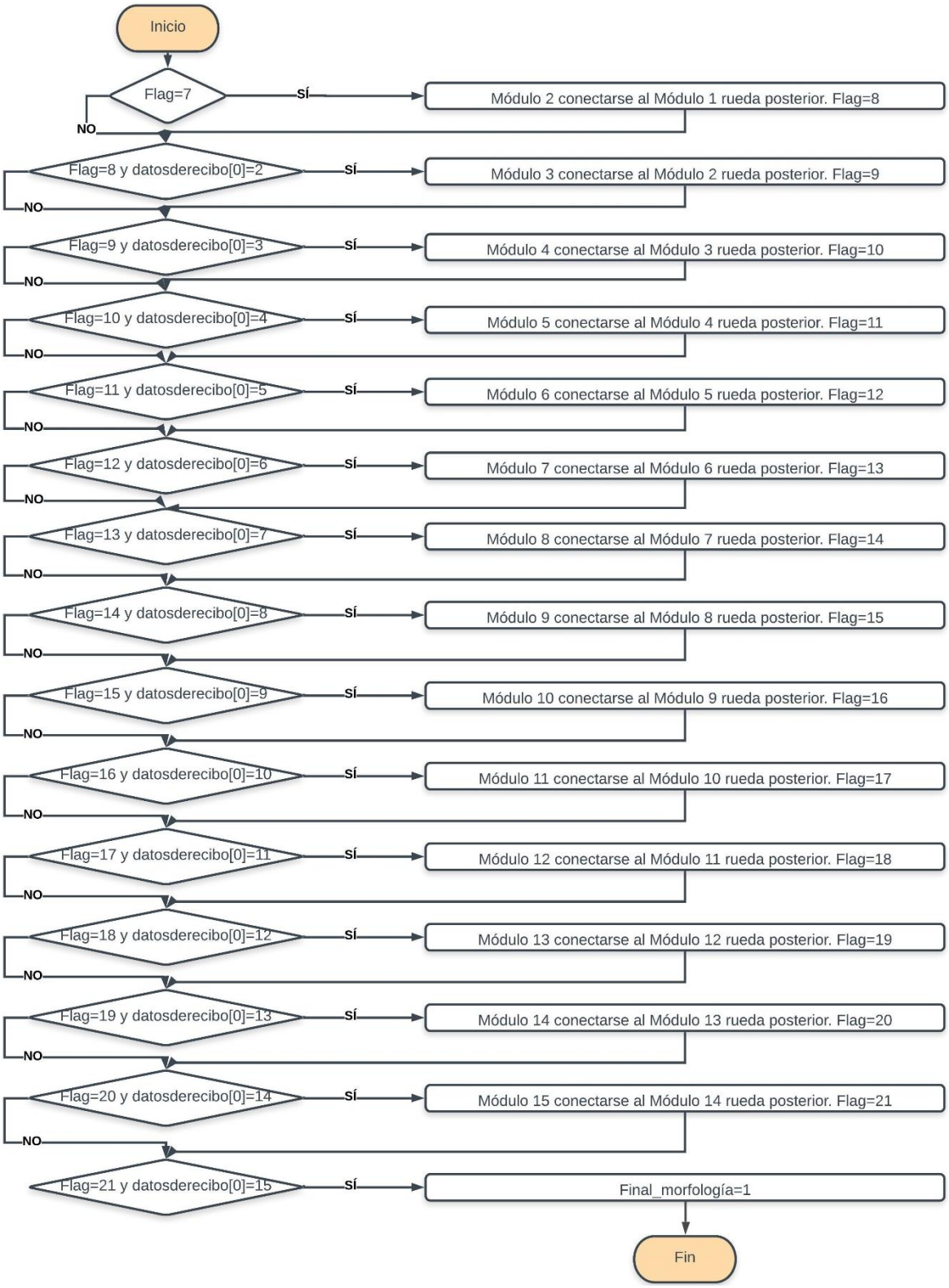


Figura 73. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) de la estrategia de armado de la arquitectura oruga con los demás módulos del 2 al 15 (esclavos). Elaboración propia

Como se puede apreciar en la Figura 73, el armado de la arquitectura oruga, empieza en el estado representado por la marca "*flag=7*", en continuidad con la Figura 55, lo que indica que una vez terminado el movimiento individual del módulo 1, prosigue a comunicarse con los demás módulos sucesivamente desde el 2 hasta el 15, para que se acoplen a su predecesor en la rueda posterior, para lo cual se define la variable "*com_forma=1*" (ver Tabla 11) al inicio de la función "*arquitectura_oruga*", (**Anexo 9**).

El final del armado de la arquitectura oruga, está representado por la marca "*flag=21*" y la comunicación recibida de acople del módulo 15 en la variable "*datosderecibo[0]*" (Figura 73), para proceder entonces a definir la variable "*final_morfologia=1*" (variable declarada y mostrada en **Anexo *1***), tomada como otra marca que lleva el registro de culminación de armado de la arquitectura oruga.

- Algoritmo de control para armado de arquitectura oruga en los módulos esclavos 2 al 15

Por otra parte, para los módulos esclavos del 2 al 15, una vez recibida la instrucción desde el módulo maestro 1 al módulo esclavo correspondiente, mediante la validación de la identificación asignada a cada uno (variable "*id*" con "*datosderecibo[0]*" ver Tabla 12 y Código 12), junto con la validación de la instrucción de la rueda a ser acoplada del módulo deseado, que para el caso es la posterior según la instrucción del módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 16, incluida en la función principal del controlador de cada módulo esclavo, la cual señala la acción del desplazamiento y posterior conexión a la rueda deseada del módulo a acoplar, mediante la ejecución del algoritmo de desplazamiento (mostrado en la Figura 63) y concluye con la respuesta de comunicación al módulo principal (señalada en el Código 15).

```
//Para realizar el movimiento del modulo individual para llevarlo a la posicion y orientacion deseada
if(datosderecibo[0]==id && datosderecibo[4]>0 && datosderecibo[4]<4){desplazamiento_modulo();}
```

Código 16. Secuencia de programación para ejecutar el desplazamiento y posterior conexión a la rueda del módulo a acoplar en los módulos esclavos del 2 al 15– Software WEBOTS. Elaboración propia

4.1.3. Coordinación de ORUGA para desplazamiento hacia adelante

Para la ejecución del desplazamiento hacia adelante de la arquitectura oruga, como se menciona en el trabajo realizado por las Ingenieras Paola Lancheros y Laura Sanabria [62], se utiliza la secuencia mostrada en el Código 17, que indica la función "*modulo_serpiente*" creada en la secuencia de programación para cada controlador de cada módulo; dicha función, como se resalta en el recuadro de color "verde" de el Código 17, actualiza la posición del motor del pivote de cada módulo (señalado con el número 1 en la Figura 46 y en la Tabla 2), según la identificación asignada a

cada uno (variable "*id*"), conforme al ajuste incremental del parámetro de la medida de tiempo, registrado en la variable "*t*", para la actualización de la fase en la representación sinusoidal que describe el movimiento oruga.

```
static void modulo_serpiente()
{
  double shift=((2*PI*k/M)*((id-1)+d));
  double phase =2.0*PI*F*t;
  //wb_motor_set_position(motor_pivote, -A*cos(phase-shift));
  wb_motor_set_position(motor_pivote, A*sin(phase+shift));
  t += TIME_STEP / 1000.0;
}
```

Código 17. Secuencia de programación en cada módulo para hacer desplazar hacia adelante la conformación de la arquitectura oruga en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Para la ejecución del desplazamiento de cada módulo, se ejecuta el llamado de la función a través de la estructura de nombre "*states*", como se muestra en el Código 18, que para el caso específico del módulo 1 (recuadro de color "azul"), la función "*modulo_serpiente*" (Código 17), que describe el movimiento de la serpiente, se encuentra en la posición 5 (cinco); como también para el caso del módulo 2 (recuadro de color "rojo"), se encuentra en la posición 6 (seis).

```
struct
{
  void (*func) ();
}

states[] =
{
  {modulo_defrente},
  {modulo_atras},
  {modulo_izquierda},
  {modulo_derecha},
  {modulo_arriba},
  {modulo_serpiente},
  {modulo_oruga_abajo},
  { NULL}
};
```

```
struct
{
  void (*func) ();
}

states[] =
{
  {modulo_defrente},
  {modulo_atras},
  {modulo_izquierda},
  {modulo_derecha},
  {modulo_arriba},
  {modulo_centro},
  {modulo_serpiente},
  {modulo_oruga_abajo},
  { NULL}
};
```

Código 18. Definición estructura "states" izquierda controlador módulo 1 y derecha controlador módulo 2 - Software WEBOTS. Elaboración propia

Ejecutado el proceso de armado de la arquitectura oruga (Figura 73), se generó la posibilidad de desplazamiento de todo el conjunto hacia adelante, con el concurso de todos módulos al mismo tiempo, para lo cual se definió en el controlador del módulo maestro 1 la función "*oruga_defrente*", como se muestra en el Código 19, la cual a través de la marca "*flag=22*", comunica la instrucción a todos los demás módulos mediante la variable "*modulo_accion*=1" (Código 12), la orden de ejecutar el movimiento oruga, según la variable "*com_forma=4*" (ver Tabla 11); para posteriormente ejecutar por parte del módulo 1 el llamado de la función

"*modulo_serpiente*", con el uso de la estructura "*states[5]*" (Código 18), representado por la marca "*flag=23*".

```
void oruga_defrente()
{
  if(flag==22){modulo_accion=1;modulo_conectar=0;com_forma=4;comunicacion_modulos();flag=23;
  if(flag==23){(*states[5].func) ();contador_movimiento++;}
}
```

Código 19. Secuencia de programación en el módulo 1 (maestro) para coordinar el desplazamiento hacia adelante de la arquitectura oruga en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos esclavos del 2 al 15, una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de desplazamiento de la arquitectura oruga, que para el caso es "*datosderecibo[4]=4*" (ver Tabla 11) según la instrucción del módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 20, incluida en la función principal del controlador de cada módulo esclavo, la cual para el caso de ejecutar el movimiento oruga por parte del módulo 2, señala la acción del llamado de la función "*modulo_serpiente*", con el uso de la estructura "*states[6]*", en continuidad con el ejemplo del Código 18 (recuadro de color "rojo").

```
//para moverse como ORUGA
if(datosderecibo[0]==1 && datosderecibo[4]==4 && datosderecibo[5]==1){if(flag==10){(*states[6].func) ();}}
```

Código 20. Secuencia de programación en el módulo 2 (esclavo) para ejecutar el desplazamiento hacia adelante de la arquitectura oruga en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

4.1.4. Coordinación de ORUGA para posición de reposo

Para la ejecución de la posición de reposo de la arquitectura oruga, se utiliza la secuencia mostrada en el Código 21, que indica la función "*modulo_oruga_abajo*" creada en la secuencia de programación para cada controlador de cada módulo; dicha función, como se resalta en el recuadro de color "azul" de el Código 21, actualiza la posición del motor del pivote de cada módulo (señalado con el número 1 en la Figura 46 y en la Tabla 2), llevándolo a su posición inicial, que es 0 grados.

```
static void modulo_oruga_abajo()
{
  t=0.0;
  wb_motor_set_position(motor_pivote,0*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(100);
}
```

Código 21. Secuencia de programación en cada módulo para ejecutar la posición de reposo en la conformación de la arquitectura oruga en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Ejecutado el desplazamiento hacia adelante de la arquitectura oruga (Figura 73), se generó la posibilidad de volver a la posición de reposo de todo el conjunto, con el concurso de todos módulos al mismo tiempo, para lo cual se definió en el controlador del módulo maestro 1 la función "*oruga_abajo*", como se muestra en el Código 22, la cual a través de la marca "*flag=24*", comunica la instrucción a todos los demás módulos mediante la variable "*modulo_accion*=1" (Código 12), la orden de ejecutar la posición de reposo en la oruga, según la variable "*com_forma=20*" (ver Tabla 11); para posteriormente ejecutar por parte del módulo 1 el llamado de la función "*modulo_oruga_abajo*", con el uso de la estructura "*states[6]*" (Código 18), representado por la marca "*flag=25*".

```
void oruga_abajo()
{
  //para colocar la oruga abajo
  if(flag==24){modulo_accion=1;modulo_conectar=0;com_forma=20;comunicacion_modulos();flag=25;}
  //para colocar la oruga abajo
  if(flag==25){(*states[6].func) ();flag=53;contador_movimiento++;}
}
```

Código 22. Secuencia de programación en el módulo 1 (maestro) para coordinar posición de reposo de la arquitectura oruga en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos esclavos del 2 al 15, una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de reposo de la arquitectura oruga, que para el caso es "*datosderecibo[4]=20*" (ver Tabla 11) según la instrucción del módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 23, incluida en la función principal del controlador de cada módulo esclavo, la cual para el caso de ejecutar la posición de reposo de la oruga por parte del módulo 2, señala la acción del llamado de la función "*modulo_oruga_abajo*", con el uso de la estructura "*states[7]*", en continuidad con el ejemplo del Código 18 (recuadro de color "rojo").

```
//para colocar la ORUGA abajo
if(datosderecibo[0]==1 && datosderecibo[4]==20 && datosderecibo[5]==1 && message_printed == 1){(*states[7].func) ();}
```

Código 23. Secuencia de programación en el módulo 2 (esclavo) para ejecutar la posición de reposo de la arquitectura oruga en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

4.2. Estrategia de Armado Arquitectura HEXÁPODO

Conforme a las herramientas brindadas para el desplazamiento individual de los módulos, la conexión y la comunicación entre módulos, mediante el control de los dispositivos incorporados a cada uno (Tabla 2 y Tabla 3), establecidos los protocolos de envío y recepción de información y la base de la conformación morfológica de la oruga, se crearon las estrategias de armado automático de la segunda arquitectura planteada para el presente proyecto, esta es la hexápodo.

4.2.1. Secuencia de armado arquitectura HEXÁPODO

La arquitectura hexápodo[13], como su nombre lo indica consiste en ubicar y conectar los módulos de forma semejante a un insecto de seis patas, siendo acoplados éstos por los conectores de la rueda pivote al de la rueda izquierda, derecha o posterior del otro (señalados con los números 6, 7, 8 y 9 en Tabla 2 respectivamente), la secuencia de armado es liderada por el módulo 1, que además de ser considerado el módulo maestro, se ubica delante de los demás módulos en la conformación de la arquitectura hexápodo; en la Figura 74 se muestra la organización de los módulos para su configuración, de acuerdo al número que le fue asignado a cada módulo para su identificación, la línea roja representa el motor de la rueda pivote del módulo, parte frente del mismo (número 2 en Tabla 2).

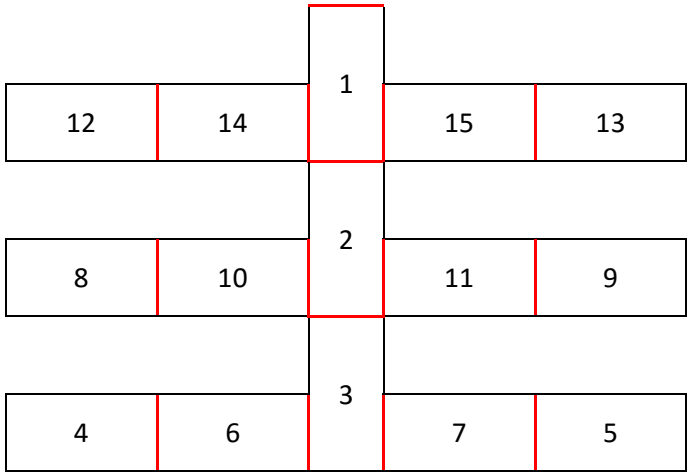


Figura 74. Esquema de conformación arquitectura hexápodo, vista desde arriba. Elaboración propia

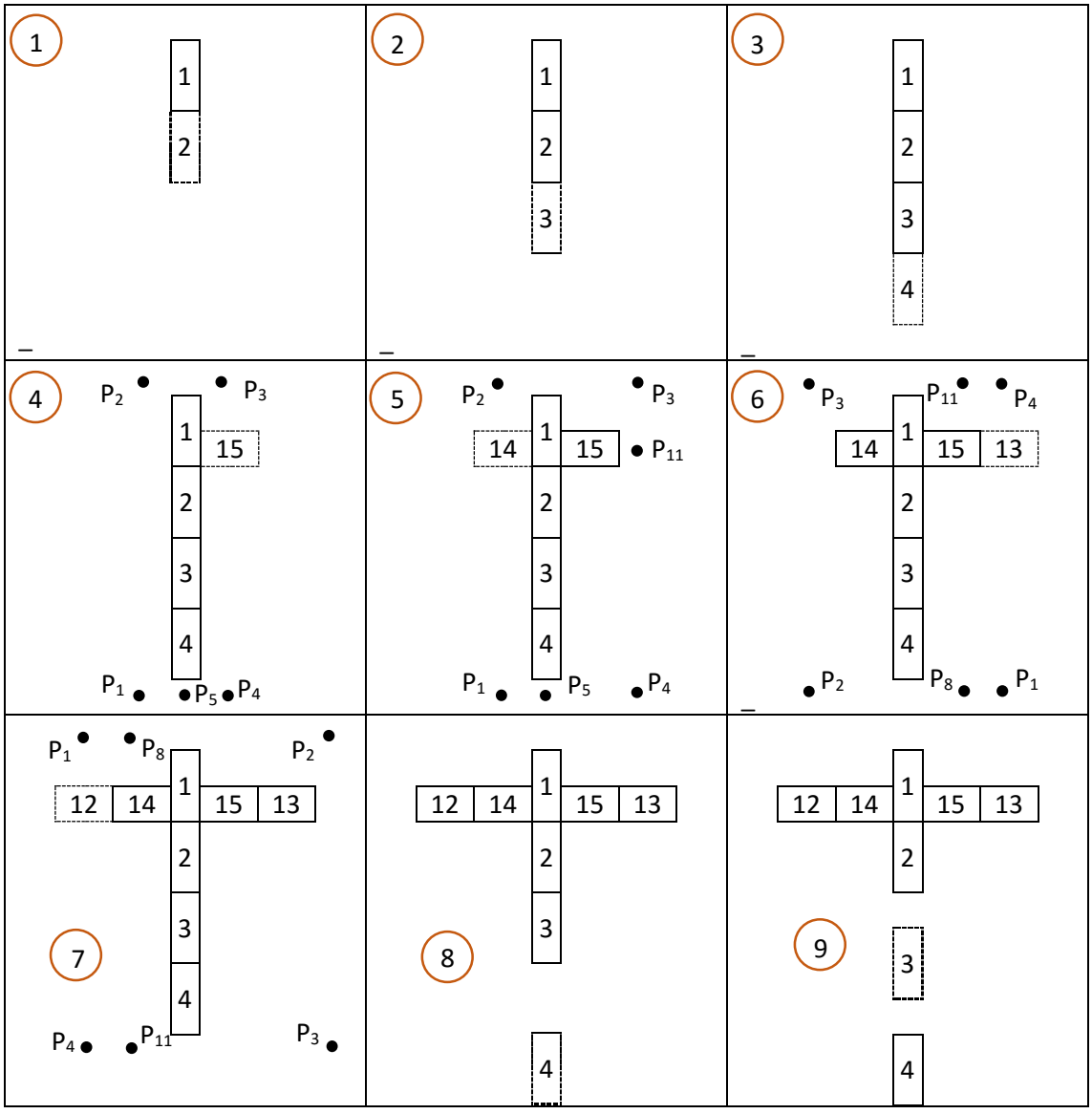
La secuencia de armado de la arquitectura hexápodo, se diseñó de forma que el sistema se pudiera desarmar y reorganizar para ejecutar el armado de la arquitectura oruga, también propuesta para el presente proyecto, de manera eficiente y efectiva; es en este sentido como se muestra en la Figura 74, que los módulos más próximos al módulo 3 (tres), son el 4 (cuatro) hasta el 7 (siete), pues son los que primero se desarmarán, para generar el espacio suficiente para los siguientes y de la misma manera permitirle a los últimos módulos, desde el 12 (doce) hasta el 15 (quince), desacoplarse y quedar disponibles para conformar la arquitectura oruga.

En la Tabla 13, de forma enumerada en el círculo de color “naranja”, se muestra la secuencia de conexión de los módulos, para conformar la arquitectura hexápodo,

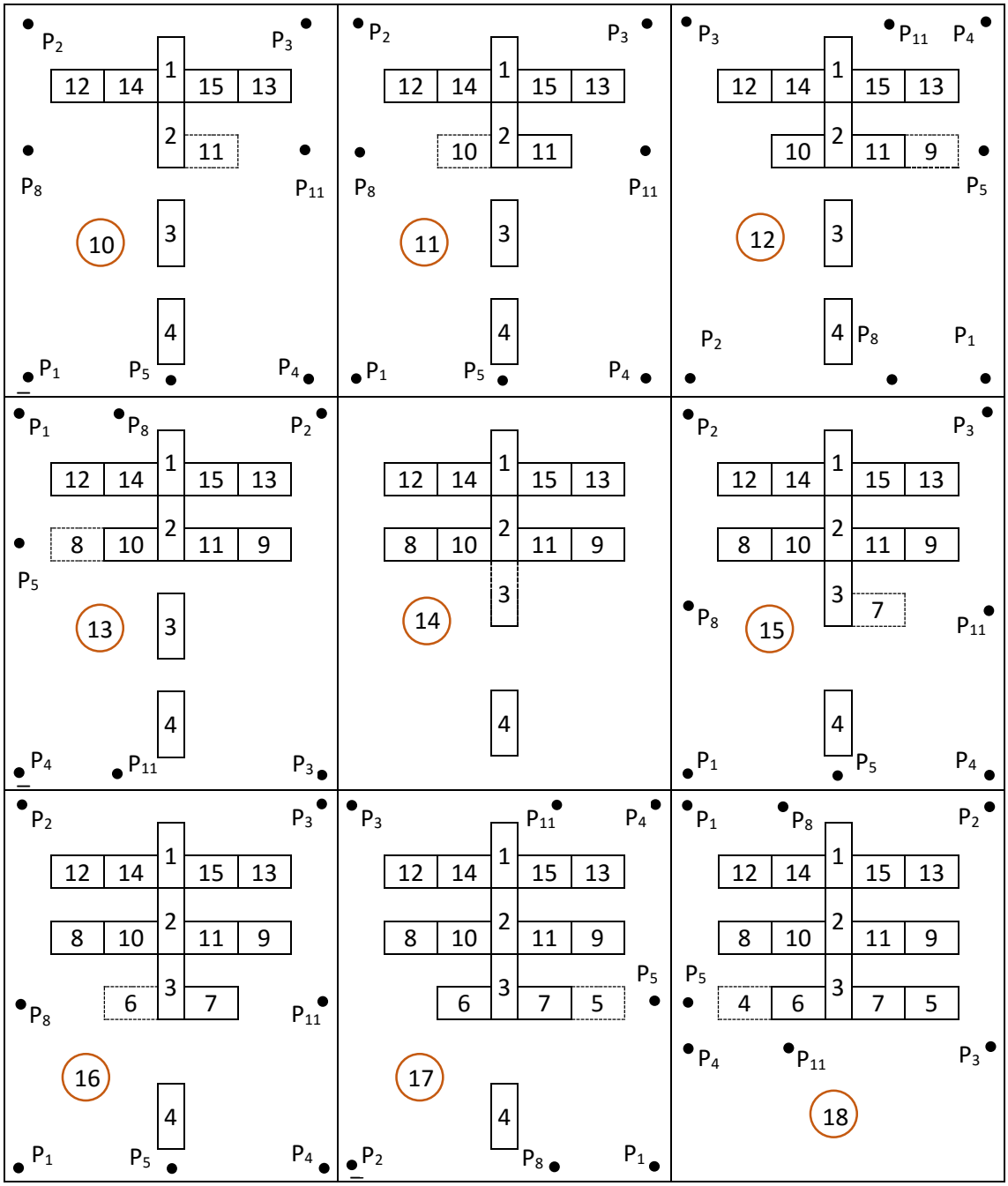
[13] Para conocer el detalle de desarrollo de la arquitectura hexápodo, ver el trabajo realizado por Ingeniero Mateo Cotera [64].

cambiando desde la conexión del módulo 4 al sistema, considerando que los para los módulos anteriores (2 y 3), se realiza la conexión de la misma manera que para la arquitectura oruga; las imágenes de la secuencia de armado mostrados en la Tabla 13, además, resaltan el ajuste de los puntos de bordeo[14] a tener en cuenta el módulo de acople (línea punteada), para hacer la conexión a la rueda deseada del módulo a acoplar, considerando como obstáculos los demás módulos ya acoplados al sistema de la morfología en proceso.

Tabla 13. Secuencia de armado para la conformación de la arquitectura hexápodo, colocación de nueva ubicación de puntos de bordeo. Elaboración propia



[14] Mencionado anteriormente en la sección 3.2.3, página 93.



Para la ubicación de los puntos de bordeo mostrados en la Tabla 13, se tomó como base la metodología de ubicación de los puntos planteada anteriormente y mostrada en la Figura 62 para la conexión entre módulos, con el objetivo de situarlos a los extremos de la zona perimetral del sistema en proceso de armado, para permitirle al módulo de acople desplazarse (según el algoritmo mostrado en Figura 63) hasta al módulo a acoplar de forma segura y controlada mediante la llegada al punto de

bordeo más próximo, de forma que pueda continuar su trayectoria entre éstos hasta llegar a la altura de la rueda deseada del módulo a acoplar, puntos P_5, P_8 o P_{11}, para las ruedas posterior, izquierda o derecha respectivamente.

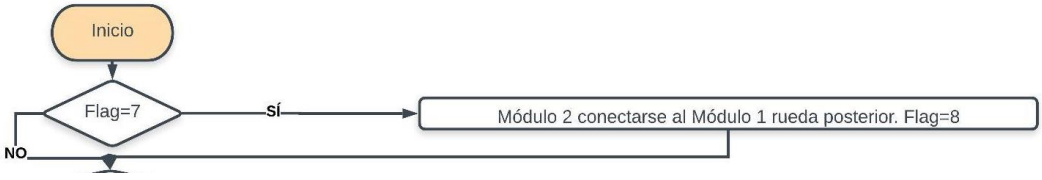
Para integrar a la secuencia de programación de los controladores de cada módulo, la ubicación de los nuevos puntos de bordeo utilizados para el armado de la arquitectura hexápodo, pero al mismo tiempo, lograrlos diferenciar con los puntos expuestos en la Figura 62, para el armado de la arquitectura oruga, se incluye desde el módulo maestro ordenar cuál de éstas dos morfologías ejecutar (definido por la variable "*arquitectura*" en dato de envío No. 6 - Tabla 10); instrucción que en los módulos esclavos se efectúa con la adición de los nuevos puntos en la función "*puntos_bordeo*" para el módulo 4 hasta el 15 (Tabla 13), los cuales se presenta el detalle en el **Anexo 10** de las sentencias incluidas para cada controlador.

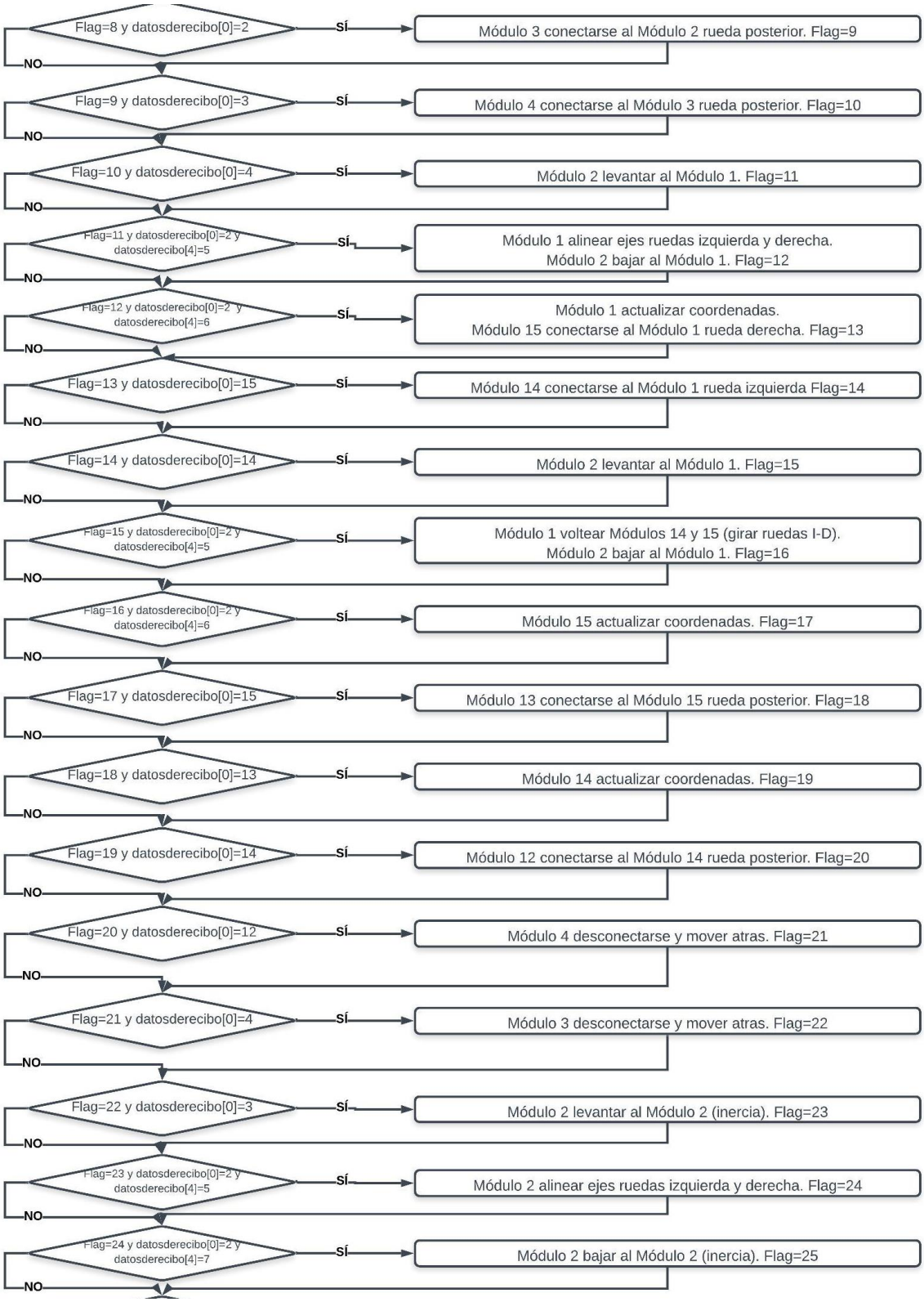
4.2.2. Algoritmo y Pseudocódigo para controlar el armado arquitectura HEXÁPODO

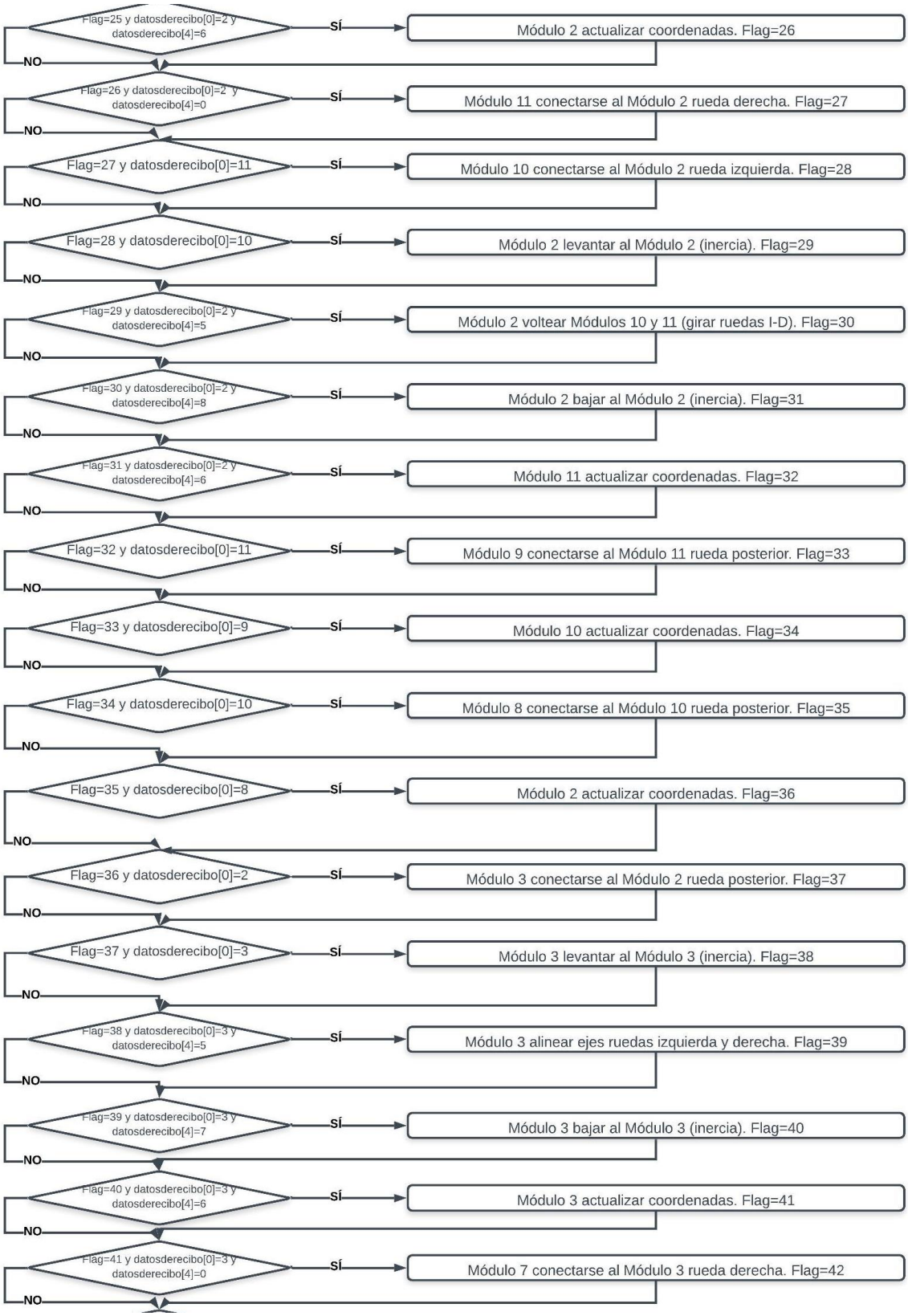
Para la ejecución del algoritmo para el armado de la arquitectura hexápodo, se acude al uso de las herramientas formadas anteriormente, tales como el control de posicionamiento y de orientación de cada módulo (Código 3 y Figura 55), el control de conexión de las ruedas entre módulos (Código 8), el control de desplazamiento del módulo de acople a través de los puntos de bordeo del módulo a acoplar (Figura 63 y Tabla 13) y la ejecución del protocolo de comunicación entre el módulo maestro 1 y los módulos esclavos del 2 al 15 (Figura 71).

- Algoritmo de control para armado de arquitectura oruga en el módulo 1

Para ejecutar el armado de la arquitectura hexápodo, para el módulo maestro 1 se definió la estructura secuencial como se muestra en la Figura 75, registrándose los estados en la variable "*flag*", con la cadencia en la ejecución de la siguiente instrucción dependiendo de la variable "*datosderecibo[0]*" y "*datosderecibo[4]*", que indica cuando se recibe la señal en el módulo maestro de que el módulo esclavo en acción ha terminado la orden y devuelto la información de acuerdo al protocolo de comunicación (Código 14). La estructura que simboliza lo definido en la secuencia de programación del controlador del módulo 1, se generó en la función "*arquitectura_hexapodo*", la cual se encuentra detallada en el **Anexo 11**.




Flag=8 y datosderecibo[0]=2
SÍ
Módulo 3 conectarse al Módulo 2 rueda posterior. Flag=9
NO
Flag=9 y datosderecibo[0]=3
SÍ
Módulo 4 conectarse al Módulo 3 rueda posterior. Flag=10
NO
Flag=10 y datosderecibo[0]=4
SÍ
Módulo 2 levantar al Módulo 1. Flag=11
NO
Flag=11 y datosderecibo[0]=2 y datosderecibo[4]=5
SÍ
Módulo 1 alinear ejes ruedas izquierda y derecha.
Módulo 2 bajar al Módulo 1. Flag=12
NO
Flag=12 y datosderecibo[0]=2 y datosderecibo[4]=6
SÍ
Módulo 1 actualizar coordenadas.
Módulo 15 conectarse al Módulo 1 rueda derecha. Flag=13
NO
Flag=13 y datosderecibo[0]=15
SÍ
Módulo 14 conectarse al Módulo 1 rueda izquierda Flag=14
NO
Flag=14 y datosderecibo[0]=14
SÍ
Módulo 2 levantar al Módulo 1. Flag=15
NO
Flag=15 y datosderecibo[0]=2 y datosderecibo[4]=5
SÍ
Módulo 1 voltear Módulos 14 y 15 (girar ruedas I-D).
Módulo 2 bajar al Módulo 1. Flag=16
NO
Flag=16 y datosderecibo[0]=2 y datosderecibo[4]=6
SÍ
Módulo 15 actualizar coordenadas. Flag=17
NO
Flag=17 y datosderecibo[0]=15
SÍ
Módulo 13 conectarse al Módulo 15 rueda posterior. Flag=18
NO
Flag=18 y datosderecibo[0]=13
SÍ
Módulo 14 actualizar coordenadas. Flag=19
NO
Flag=19 y datosderecibo[0]=14
SÍ
Módulo 12 conectarse al Módulo 14 rueda posterior. Flag=20
NO
Flag=20 y datosderecibo[0]=12
SÍ
Módulo 4 desconectarse y mover atras. Flag=21
NO
Flag=21 y datosderecibo[0]=4
SÍ
Módulo 3 desconectarse y mover atras. Flag=22
NO
Flag=22 y datosderecibo[0]=3
SÍ
Módulo 2 levantar al Módulo 2 (inercia). Flag=23
NO
Flag=23 y datosderecibo[0]=2 y datosderecibo[4]=5
SÍ
Módulo 2 alinear ejes ruedas izquierda y derecha. Flag=24
NO
Flag=24 y datosderecibo[0]=2 y datosderecibo[4]=7
SÍ
Módulo 2 bajar al Módulo 2 (inercia). Flag=25
NO


Flag=25 y datosderecibo[0]=2 y datosderecibo[4]=6
SÍ
Módulo 2 actualizar coordenadas. Flag=26
NO
Flag=26 y datosderecibo[0]=2 y datosderecibo[4]=0
SÍ
Módulo 11 conectarse al Módulo 2 rueda derecha. Flag=27
NO
Flag=27 y datosderecibo[0]=11
SÍ
Módulo 10 conectarse al Módulo 2 rueda izquierda. Flag=28
NO
Flag=28 y datosderecibo[0]=10
SÍ
Módulo 2 levantar al Módulo 2 (inercia). Flag=29
NO
Flag=29 y datosderecibo[0]=2 y datosderecibo[4]=5
SÍ
Módulo 2 voltear Módulos 10 y 11 (girar ruedas I-D). Flag=30
NO
Flag=30 y datosderecibo[0]=2 y datosderecibo[4]=8
SÍ
Módulo 2 bajar al Módulo 2 (inercia). Flag=31
NO
Flag=31 y datosderecibo[0]=2 y datosderecibo[4]=6
SÍ
Módulo 11 actualizar coordenadas. Flag=32
NO
Flag=32 y datosderecibo[0]=11
SÍ
Módulo 9 conectarse al Módulo 11 rueda posterior. Flag=33
NO
Flag=33 y datosderecibo[0]=9
SÍ
Módulo 10 actualizar coordenadas. Flag=34
NO
Flag=34 y datosderecibo[0]=10
SÍ
Módulo 8 conectarse al Módulo 10 rueda posterior. Flag=35
NO
Flag=35 y datosderecibo[0]=8
SÍ
Módulo 2 actualizar coordenadas. Flag=36
NO
Flag=36 y datosderecibo[0]=2
SÍ
Módulo 3 conectarse al Módulo 2 rueda posterior. Flag=37
NO
Flag=37 y datosderecibo[0]=3
SÍ
Módulo 3 levantar al Módulo 3 (inercia). Flag=38
NO
Flag=38 y datosderecibo[0]=3 y datosderecibo[4]=5
SÍ
Módulo 3 alinear ejes ruedas izquierda y derecha. Flag=39
NO
Flag=39 y datosderecibo[0]=3 y datosderecibo[4]=7
SÍ
Módulo 3 bajar al Módulo 3 (inercia). Flag=40
NO
Flag=40 y datosderecibo[0]=3 y datosderecibo[4]=6
SÍ
Módulo 3 actualizar coordenadas. Flag=41
NO
Flag=41 y datosderecibo[0]=3 y datosderecibo[4]=0
SÍ
Módulo 7 conectarse al Módulo 3 rueda derecha. Flag=42
NO

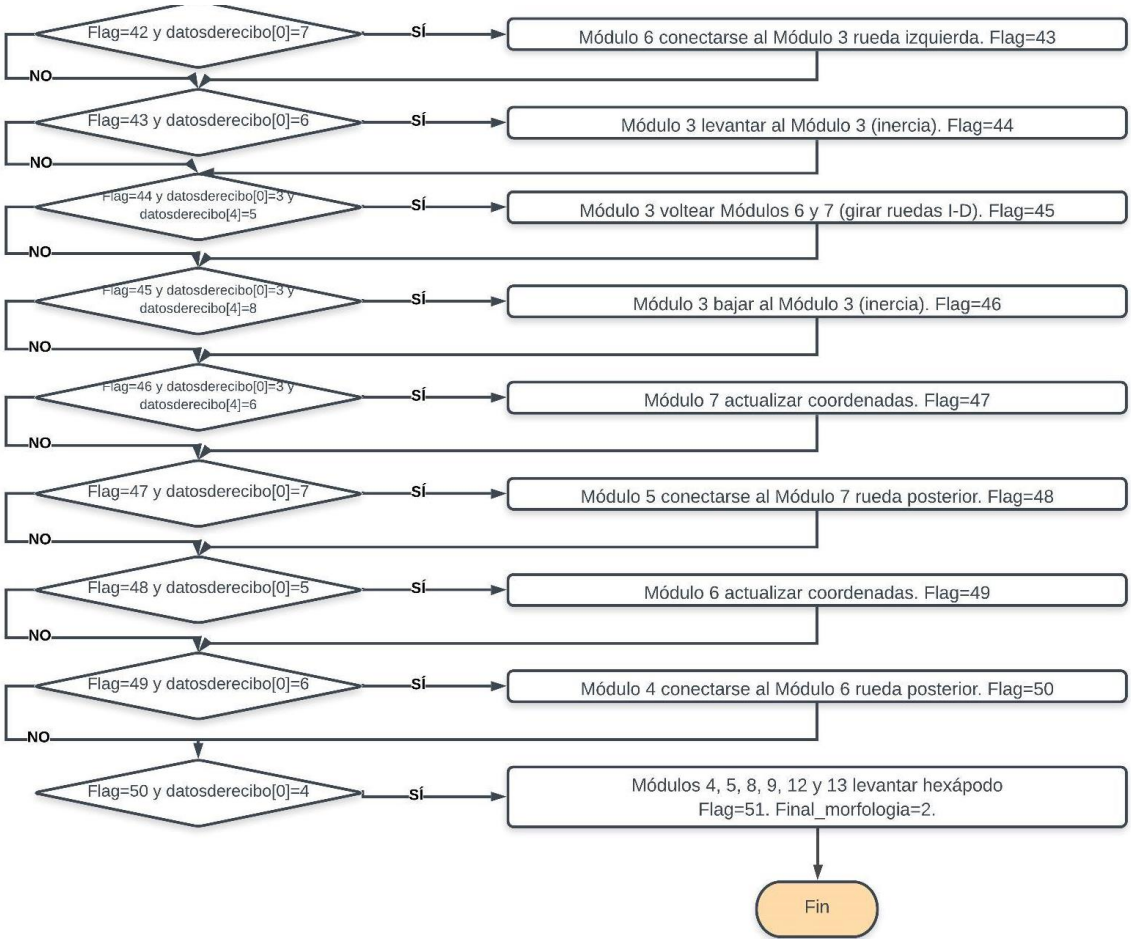


Figura 75. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) de la estrategia de armado de la arquitectura hexápodo con los demás módulos del 2 al 15 (esclavos). Elaboración propia

Como se puede apreciar en la Figura 75, el armado de la arquitectura hexápodo, empieza en el estado representado por la marca "*flag=7*", en continuidad con la Figura 55, lo que indica que una vez terminado el movimiento individual del módulo 1, prosigue a comunicarse con los demás módulos sucesivamente entre el 2 y el 15, para que se acoplen conforme a la secuencia de armado mostrada en la Tabla 13, para lo cual se define la variable "*com_forma*" (parámetros en Tabla 11) acorde a la instrucción a ejecutar en cada estado de la secuencia mostrada en la Figura 75.

El final del armado de la arquitectura hexápodo, está representado por la marca "*flag=50*" y la comunicación recibida de acople del módulo 4 en la variable "*datosderecibo[0]*" (Figura 75), para proceder entonces a definir la variable "*final_morfologia=2*" (variable declarada y mostrada en **Anexo *1***), tomada como otra marca que lleva el registro de culminación de armado de la arquitectura hexápodo.

- Algoritmo de control para armado de arquitectura oruga en los módulos esclavos 2 al 15

Por otra parte, para los módulos esclavos del 2 al 15, una vez recibida la instrucción desde el módulo maestro 1 al módulo esclavo correspondiente, mediante la validación de la identificación asignada a cada uno (variable "*id*" con "*datosderecibo[0]*" ver Tabla 12 y Código 12), junto con la validación de la instrucción ordenada al módulo, se procede por parte del mismo a ejecutar alguna de las siguiente acciones, conforme a lo mostrado en la Figura 75:

- Conexión a la rueda a ser acoplada del módulo deseado, Código 16, ejecución de todo los módulos esclavos.
- Para levantar y bajar el módulo 2, Código 24.

```
//para levantar el modulo 1 girando el pivote
if(datosderecibo[0]==id && datosderecibo[4]==5 && message_printed == 1){(*states[4].func) ();comunicacion_modulos();
//para bajar el modulo 1 girando el pivote
if(datosderecibo[0]==id && datosderecibo[4]==6 && message_printed == 1){(*states[5].func) ();comunicacion_modulos();
```

Código 24. Secuencia de programación para levantar y bajar el módulo 2 – Software WEBOTS. Elaboración propia

- Para girar las ruedas izquierda y derecha con el fin de alinear los ejes de acople y para voltear los módulos acoplados, Código 25, ejecución módulos 2 y 3.

```
//para girar las ruedas del modulo 2 y alinear ejes despues para bajar el modulo 2
if(datosderecibo[0]==id && datosderecibo[4]==7 && message_printed == 1){condicion_giro=0;alineacion_ejes_ruedasizqder();comunicacion_modulos();
//para girar las ruedas del modulo 2 y alinear ejes despues para bajar el modulo 2
if(datosderecibo[0]==id && datosderecibo[4]==8 && message_printed == 1){condicion_giro=0;volteo_ruedasizqder();comunicacion_modulos();
```

Código 25. Secuencia de programación para alinear los ejes de acople del módulo y voltear los módulos acoplados – Software WEBOTS. Elaboración propia

- Para actualizar coordenadas, Código 26, ejecución módulos 2, 3, 6, 7, 10, 11, 14 y 15.

```
//para enviar coordenadas al modulo 1
if(datosderecibo[0]==id && datosderecibo[4]==0 && message_printed == 1){coordenadas_modulo();comunicacion_modulos();
```

Código 26. Secuencia de programación para actualizar coordenadas del módulo esclavo y enviarlas al módulo maestro – Software WEBOTS. Elaboración propia

- Para desacoplar el módulo esclavo y moverlo atrás, Código 27, ejecución módulos 3 y 4.

```
if(datosderecibo[0]==id && datosderecibo[4]==9 && message_printed == 1)
{wb_connector_unlock(conector_pivote);for(i=0;i<90;i++){(*states[1].func) ();}comunicacion_modulos();
```

Código 27. Secuencia de programación para desacoplar el módulo esclavo y moverlo atrás– Software WEBOTS. Elaboración propia

- Para hacer poner de pie (acción de levantarse) la arquitectura hexápodo: Se ejecuta con la secuencia de programación creada en la función "*modulo_hexapodo_pie*", mostrada en el Código 28, la cual acciona a 60 grados el motor del pivote de los módulos 4, 5, 8, 9, 12 y 13.

```
static void modulo_hexapodo_pie()
{
  wb_motor_set_position(motor_pivote,60*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(1000);
}
```

Código 28. Secuencia de programación en los módulos 4, 5, 8, 9, 12 y 13 para hacer poner de pie al hexápodo mediante el accionamiento del motor del pivote- Software WEBOTS. Elaboración propia

Para la ejecución del desplazamiento de cada módulo, considerado como la base de la pata, se ejecuta el llamado de la función a través de la estructura de nombre "*states*", como se muestra en el Código 29, que para el caso de los módulos 4, 5, 8, 9, 12 y 13, la función "*modulo_hexapodo_pie*" (Código 28) que describe el movimiento individual de cada módulo para levantar el hexápodo, se encuentra en la posición 6 (seis).

```
states[] =
{
  {modulo_defrente},
  {modulo_atras},
  {modulo_izquierda},
  {modulo_derecha},
  {modulo_arriba},
  {modulo_serpiente},
  {modulo_hexapodo_pie},
  {modulo_hexapodo_levantapie},
  {modulo_hexapodo_abajo},
  {modulo_oruga_abajo},
  { NULL}
};
```

Código 29. Definición estructura "states" módulos 4, 5, 8, 9, 12 y 13 - Software WEBOTS. Elaboración propia

Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de hacer poner de pie la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=10*" (ver Tabla 11) según la instrucción del módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 30, incluida en la función principal del controlador de los módulos 4, 5, 8, 9, 12 y 13, la cual señala la acción del llamado de la función "*modulo_hexapodo_pie*", con el uso de la estructura "*states[6]*".

```
//para colocar el Hexapodo de Pie
if(datosderecibo[0]==1 && datosderecibo[4]==10 && datosderecibo[5]==2 && message_printed == 1){(*states[6].func) ();}
```

Código 30. Secuencia de programación en los módulos 4, 5, 8, 9, 12 y 13 para ejecutar la instrucción de hacer poner de pie el hexápodo - Software WEBOTS. Elaboración propia

4.2.3. Coordinación de HEXÁPODO para desplazamiento hacia adelante

Para la ejecución del desplazamiento hacia adelante de la arquitectura hexápodo, se utiliza la secuencia de movimiento de los módulos mostrada en la Figura 76, que como permite ver realiza el movimiento de los mismos de manera sincronizada para mover las patas de la izquierda y de la derecha hacia adelante con el fin de generar el desplazamiento del sistema en conjunto, para efectos de estabilidad se ejecuta el movimiento coordinado con 3 (tres) puntos de apoyo en cada acción.

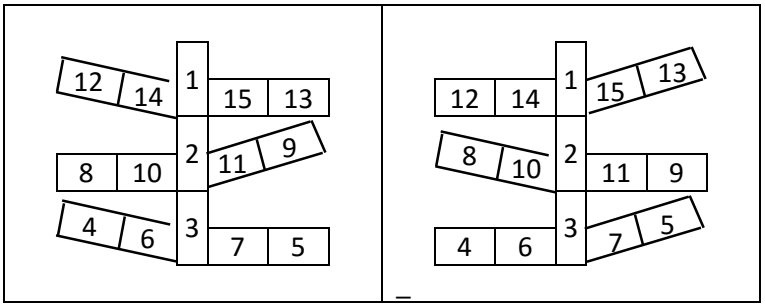


Figura 76. Secuencia de movimiento módulos para desplazamiento hacia adelante arquitectura hexápodo. Elaboración propia

La secuencia de movimiento de los módulos para el desplazamiento hacia adelante del sistema hexápodo, se ejecuta estableciendo desde el módulo maestro las sentencias de programación mostradas en el Código 31**¡Error! No se encuentra el origen de la referencia.**, que indica la función "*hexapodo_defrente*" creada en la secuencia de programación para el controlador del módulo 1, la cual coordina las siguientes 2 (dos) acciones:

- El primer ciclo (Figura 76 – cuadro de la izquierda): Con la marca "*flag=51*", comunica la instrucción mediante la variable "*com_forma =11*" a los módulos 4, 9 y 12 para levantar el pie junto con la orden al 14, 11 y 6 para desplazar hacia adelante las patas del hexápodo (2 izquierda y 1 derecha). A su vez envía la orden a los módulos 15, 10 y 7 de retornar a la posición 0 mientras se encuentran suspendidas en el aire las patas de los módulos 4, 9 y 12 (lo que genera la acción de desplazamiento del sistema en conjunto).

- El segundo ciclo (Figura 76 – cuadro de la derecha): Con la marca "*flag=52*", comunica la instrucción mediante la variable "*com_forma =12*" a los módulos 5, 8 y 13 para levantar el pie junto con la orden al 15, 10 y 7 para desplazar hacia adelante las patas del hexápodo (1 izquierda y 2 derecha). A su vez envía la orden a los módulos 14, 11 y 6 de retornar a la posición 0 mientras se encuentran suspendidas en el aire las patas de los módulos 4, 9 y 12 (lo que genera la acción de desplazamiento del sistema en conjunto).

```
void hexapodo_defrente()
{
  //para comunicarse con los modulos 4, 9 y 12 para levantar pie junto con 14, 11 y 6 para caminar hacia adelante el hexapodo
  if(flag==51 && datosderecibo[0]==13){modulo_accion=1;modulo_conectar=0;com_forma=11;comunicacion_modulos();flag=52;}
  //para comunicarse con los modulos 5, 8 y 13 para levantar pie junto con 15, 10 y 7 para caminar hacia adelante el hexapodo
  if(flag==52 && datosderecibo[0]==12){modulo_accion=1;modulo_conectar=0;com_forma=12;comunicacion_modulos();flag=51;}

}
```

Código 31. Secuencia de programación en el módulo 1 (maestro) para coordinar el desplazamiento hacia delante de la arquitectura hexápodo en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos esclavos, una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con las instrucciones de validación de desplazamiento hacia adelante de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=11*" y "*datosderecibo[4]=12*" (ver Tabla 11), según la orden del módulo maestro; se prosigue a ejecutar las líneas de código incluidas en la función principal del controlador de cada módulo esclavo, como se muestra a continuación:

- El primer ciclo (Figura 76 – cuadro de la izquierda) inicia con la actividad de levantar el pie de los módulos 4, 9 y 12, mediante accionamiento de motor del pivote con la función "*modulo_hexapodo_levantapie*" mostrada en el Código 32, para accionar el motor del pivote en la posición de 30 (treinta) grados, luego ejecutar un retardo (mientras los módulos 14, 11 y 6 ejecutan el movimiento de las patas hacia adelante) y volver a su posición de 60 (sesenta) grados.

```
static void modulo_hexapodo_levantapie()
{
  wb_motor_set_position(motor_pivote,30*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(600);
  wb_motor_set_position(motor_pivote,60*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(500);
}
```

Código 32. Secuencia de programación en los módulos 4, 9 y 12, como también 5, 8 y 13 para hacer levantar el pie del hexápodo mediante el accionamiento del motor del pivote- Software WEBOTS. Elaboración propia

Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de levantar el pie en ejecución de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=11*" (ver Tabla 11) según la orden impartida por el módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 33, incluida en la función principal del controlador de los módulos 4, 9

y 12 la cual señala el llamado de la función "*modulo_hexapodo_levantapie*", con el uso de la estructura "*states[7]*" (Código 29).

```
//para levantar el Pie del Hexapodo
if(datosderecibo[0]==1 && (datosderecibo[4]==11 || datosderecibo[4]==13 || datosderecibo[4]==15
...|| datosderecibo[4]==18) && datosderecibo[5]==2 && message_printed == 1){(*states[7].func) ();}
```

Código 33. Secuencia de programación en los módulos 4, 9 y 12 para ejecutar la instrucción levantar el pie del hexápodo - Software WEBOTS. Elaboración propia

- Consecuentemente, mientras las patas de los módulos 4, 9 y 12 se encuentran suspendidas en el aire, se ejecuta el movimiento de los módulos 14, 11 y 6, considerados como el soporte de las patas o pierna, hacia adelante, mediante la función creada para ejecutar el movimiento "*modulo_hexapodo_pieadelante*", mostrada en el Código 34, la cual inicia con un retardo (mientras se levantan las patas de los módulos 4, 9 y 12), luego se acciona el motor del pivote en la posición de -10 (diez negativo) grados.

```
static void modulo_hexapodo_pieadelante()
{
  wb_robot_step(300);
  wb_motor_set_position(motor_pivote,-10*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(500);
}
```

Código 34. Secuencia de programación en los módulos 6, 11 y 14 para hacer accionar hacia adelante las patas del hexápodo suspendidas en el aire mediante el accionamiento del motor del pivote- Software WEBOTS. Elaboración propia

Para la ejecución del desplazamiento de cada módulo, considerado como la pierna o soporte de la pata, se ejecuta el llamado de la función a través de la estructura de nombre "*states*", como se muestra en el Código 35, que para el caso de los módulos 6, 7, 10, 11, 14 y 15, la función "*modulo_hexapodo_pieadelante*" (Código 34), que describe el movimiento individual de cada módulo para mover hacia adelante la pierna o soporte de la pata del hexápodo, se encuentra en la posición 6 (seis).

```
states[] =
{
  {modulo_defrente},
  {modulo_atras},
  {modulo_izquierda},
  {modulo_derecha},
  {modulo_arriba},
  {modulo_serpiente},
  {modulo_hexapodo_pieadelante},
  {modulo_hexapodo_piecentro},
  {modulo_hexapodo_pieatras},
  {modulo_oruga_abajo},
  { NULL}
};
```

Código 35. Definición estructura "states" módulos 6, 11 y 14 como también 7, 10 y 15 - Software WEBOTS. Elaboración propia

Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de levantar el pie en ejecución de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=11*" (ver Tabla 11) según la orden impartida por el módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 36, incluida en la función principal del controlador de los módulos 6, 11 y 14 la cual señala el llamado de la función "*modulo_hexapodo_pieadelante*", con el uso de la estructura "*states[6]*" (Código 35).

```
//para mover el Pie del Hexapodo hacia adelante
if(datosderecibo[0]==1 && (datosderecibo[4]==11 || datosderecibo[4]==13)...

...&& datosderecibo[5]==2  && message_printed == 1){(*states[6].func) ();}
```

Código 36. Secuencia de programación en los módulos 6, 11 y 14 para ejecutar la instrucción accionamiento hacia adelante de las patas del hexápodo suspendidas en el aire - Software WEBOTS. Elaboración propia

- Para generar la acción de movimiento hacia adelante de la arquitectura hexápodo, mientras las patas de los módulos 4, 9 y 12 se encuentran suspendidas en el aire y la pierna o base de la misma pata, es decir los módulos 6, 11 y 14 se mueven hacia adelante, se dispone a ejecutar la posición al centro los módulos 7, 10 y 15, considerados como la pierna o base de las patas apoyadas en el suelo, acción que es la que genera el desplazamiento dado el movimiento precedido que éste lleva, mediante la función creada para ejecutar el movimiento "*modulo_hexapodo_piecentro*", mostrada en el Código 37, la cual inicia con un retardo (mientras se levantan las patas de los módulo 4, 9 y 12), luego se acciona el motor del pivote en la posición de -10 (diez negativo) grados.

```
static void modulo_hexapodo_piecentro()
{
  wb_robot_step(300);
  wb_motor_set_position(motor_pivote,0*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(500);
}
```

Código 37. Secuencia de programación en los módulos 7, 10 y 15 para hacer accionar hacia el centro las patas del hexápodo apoyadas en el suelo mediante el accionamiento del motor del pivote- Software WEBOTS. Elaboración propia

Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de levantar el pie en ejecución de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=11*" (ver Tabla 11) según la orden impartida por el

módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 38, incluida en la función principal del controlador de los módulos 7, 10 y 15 la cual señala el llamado de la función "*modulo_hexapodo_piecentro*", con el uso de la estructura "*states[7]*" (Código 35).

```
//para mover el Pie del Hexapodo hacia el centro
if(datosderecibo[0]==1 && (datosderecibo[4]==11 || datosderecibo[4]==13 || datosderecibo[4]==15
...|| datosderecibo[4]==17)  && datosderecibo[5]==2 && message_printed == 1){(*states[7].func) ();}
```

Código 38. Secuencia de programación en los módulos 7, 10 y 15 para hacer accionar hacia el centro las patas del hexápodo apoyadas en el suelo - Software WEBOTS. Elaboración propia

- Culminado el primer ciclo de desplazamiento hacia adelante del hexápodo mostrado en la Figura 76 (cuadro de la izquierda) y explicado en las tres viñetas (•) anteriores, se prosigue de la misma manera a ejecutar el segundo ciclo (Figura 76 – cuadro de la derecha), comandado por la instrucción "*datosderecibo[4]=12*" tomada desde el módulo maestro 1, con la ejecución de las líneas de programación para levantar las patas del hexápodo, con el accionamiento de los módulos 5, 8 y 13 (Código 32 y Código 33), para luego ejecutar el movimiento hacia adelante de los módulos 7, 10 y 15 mientras estas patas se encuentran suspendidas en el aire (Código 34 y Código 36), junto con la acción de desplazamiento del hexápodo, llevando la posición al centro las patas apoyadas en el suelo, mediante el movimiento de los módulos 6, 11 y 14 (Código 37 y Código 38) mientras el otro conjunto de patas se encuentran suspendidas en el aire (módulos 5, 8 y 13).

4.2.4. Coordinación de HEXÁPODO para giro a la derecha

Para la ejecución del giro hacia la derecha de la arquitectura hexápodo, se utiliza la secuencia de movimiento de los módulos mostrada en la Figura 77, que como permite ver realiza el movimiento de los mismos de manera sincronizada para mover las patas de la izquierda hacia adelante y las de la derecha hacia atrás con el fin de generar el giro del sistema en conjunto, para efectos de estabilidad se ejecuta el movimiento coordinado con 3 (tres) puntos de apoyo en cada acción.

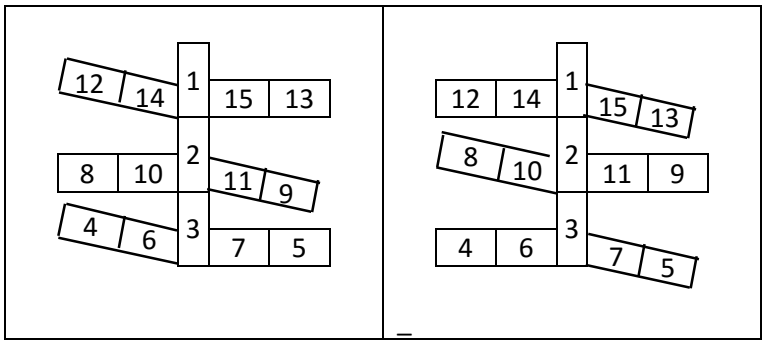


Figura 77. Secuencia de movimiento módulos para giro a la derecha de la arquitectura hexápodo. Elaboración propia

La secuencia de movimiento de los módulos para el giro a la derecha del sistema hexápodo, se ejecuta estableciendo desde el módulo maestro las sentencias de programación mostradas en el Código 39, que indica la función "*hexapodo_derecha*" creada en la secuencia de programación para el controlador del módulo 1, la cual coordina las siguientes 2 (dos) acciones:

- El primer ciclo (Figura 77 – cuadro de la izquierda): Con la marca "*flag=51*", comunica la instrucción mediante la variable "*com_forma =13*" a los módulos 4, 9 y 12 para levantar el pie junto con la orden al 6 y 14 hacia adelante, como el 11 hacia atrás, para giro del hexápodo (2 izquierda y 1 derecha). A su vez envía la orden a los módulos 15, 10 y 7 de retornar a la posición 0 mientras se encuentran suspendidas en el aire las patas de los módulos 4, 9 y 12 (lo que genera la acción de giro del sistema en conjunto).

- El segundo ciclo (Figura 77 – cuadro de la derecha): Con la marca "*flag=52*", comunica la instrucción mediante la variable "*com_forma =14*" a los módulos 5, 8 y 13 para levantar el pie junto con la orden al 10 hacia adelante, como el 15, y 7 hacia atrás, para giro del hexápodo (1 izquierda y 2 derecha). A su vez envía la orden a los módulos 14, 11 y 6 de retornar a la posición 0 mientras se encuentran suspendidas en el aire las patas de los módulos 4, 9 y 12 (lo que genera la acción de desplazamiento del sistema en conjunto).

```
void hexapodo_derecha()
{
  //para comunicarse con los modulos 4, 9 y 12 para levantar pie junto con 14 y 6 para caminar hacia adelante y 11 hacia atras
  if(flag==51 && datosderecibo[0]==13){modulo_accion=1;modulo_conectar=0;com_forma=13;comunicacion_modulos();flag=52;}
  //para comunicarse con los modulos 5, 8 y 13 para levantar pie junto con 15 y 7 para caminar hacia atras y 10 hacia adelante
  if(flag==52 && datosderecibo[0]==12){modulo_accion=1;modulo_conectar=0;com_forma=14;comunicacion_modulos();flag=51;}
}
```

Código 39. Secuencia de programación en el módulo 1 (maestro) para coordinar el giro hacia la derecha de la arquitectura hexápodo en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos esclavos, una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con las instrucciones de validación de giro a la derecha de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=13*" y "*datosderecibo[4]=14*" (ver Tabla 11), según la orden del módulo maestro; se prosigue a ejecutar las líneas de código incluidas en la función principal del controlador de cada módulo esclavo, como se muestra a continuación:

- El primer ciclo (Figura 77 – cuadro de la izquierda) inicia con la actividad de levantar el pie de los módulos 4, 9 y 12, (Código 32 y Código 33).
- Consecuentemente, mientras las patas de los módulos 4, 9 y 12 se encuentran suspendidas en el aire, se ejecuta el movimiento de los módulos 6 y 14, hacia adelante (Código 34), a su vez el módulo 11 hacia atrás,

mediante la función creada para ejecutar el movimiento "*modulo_hexapodo_pieatras*", mostrada en el Código 40, la cual inicia con un retardo (mientras se levantan las patas de los módulos 4, 9 y 12), luego se acciona el motor del pivote en la posición de 10 (diez) grados.

```
static void modulo_hexapodo_pieatras()
{
  wb_robot_step(300);
  wb_motor_set_position(motor_pivote,10*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(500);
}
```

Código 40. Secuencia de programación en el módulo 11 para hacer accionar hacia atrás la pata del hexápodo suspendida en el aire mediante el accionamiento del motor del pivote- Software WEBOTS. Elaboración propia

Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de levantar el pie en ejecución de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=13*" (ver Tabla 11) según la orden impartida por el módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 41, incluida en la función principal del controlador del módulo 11 la cual señala el llamado de la función "*modulo_hexapodo_pieatras*", con el uso de la estructura "*states[8]*" (Código 35), a su vez los módulos 6 y 14 ejecutan la línea de código mostrada en el Código 36, para mover el pie adelante.

```
//para mover el Pie del Hexapodo hacia atras
if(datosderecibo[0]==1 && datosderecibo[4]==13 && datosderecibo[5]==2 && message_printed == 1){(*states[8].func) ();}
```

Código 41. Secuencia de programación en el módulo 11 para ejecutar la instrucción accionamiento hacia atrás la pata del hexápodo suspendida en el aire - Software WEBOTS. Elaboración propia

- Para generar la acción de movimiento de giro a la derecha de la arquitectura hexápodo, mientras las patas de los módulos 4, 9 y 12 se encuentran suspendidas en el aire y las piernas tomada como los módulos 6 y 14 se mueven hacia adelante, como 11 hacia atrás, se dispone a ejecutar la posición al centro los módulos 7, 10 y 15, conforme a la función mostrada en el Código 37, la cual se ejecuta una vez recibida la instrucción del módulo maestro, mediante la instrucción "*datosderecibo[4]=13*", para proceder a ejecutar la línea de programación mostrada en el Código 38.
- Culminado el primer ciclo de giro a la derecha del hexápodo mostrado en la Figura 77 (cuadro de la izquierda) y explicado en las tres viñetas (•) anteriores, se prosigue de la misma manera a ejecutar el segundo ciclo (Figura 77 – cuadro de la derecha), comandado por la instrucción "*datosderecibo[4]=14*" tomada desde el módulo maestro 1, con la ejecución de las líneas de programación para levantar las patas del hexápodo, con el accionamiento de

los módulos 5, 8 y 13 (Código 32 y Código 33), para luego ejecutar el movimiento hacia atrás de los módulos 7 y 15 (Código 40 y Código 41), como hacia adelante el 10, mientras estas patas se encuentran suspendidas en el aire, junto con la acción de giro del hexápodo, llevando la posición al centro las patas apoyadas en el suelo, mediante el movimiento de los módulos 6, 11 y 14 (Código 37 y Código 38) mientras el otro conjunto de patas se encuentran suspendidas en el aire (módulos 5, 8 y 13).

4.2.5. Coordinación de HEXÁPODO para giro a la izquierda

Se utiliza la secuencia de movimiento de los módulos mostrada en la Figura 78, que realiza el movimiento de los módulos de manera sincronizada para mover las patas de la izquierda hacia atrás y las de la derecha hacia adelante con el fin de generar el giro del sistema, con 3 (tres) puntos de apoyo en cada acción para la estabilidad.

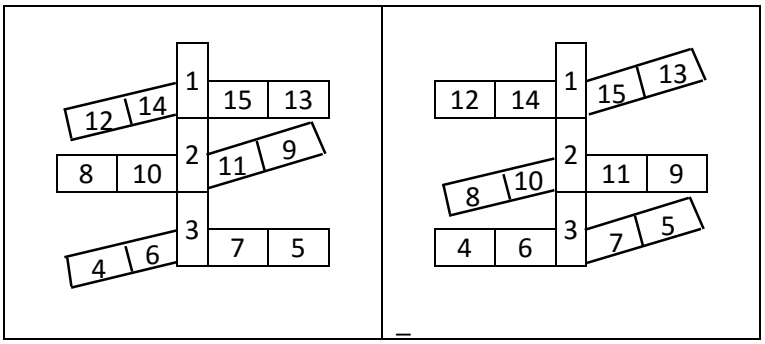


Figura 78. Secuencia de movimiento módulos para giro a la izquierda de la arquitectura hexápodo. Elaboración propia

La secuencia de movimiento de los módulos para el giro a la izquierda del sistema hexápodo, se ejecuta estableciendo desde el módulo maestro las sentencias de programación mostradas en el Código 42, que indica la función "*hexapodo_izquierda*" creada en la secuencia de programación para el controlador del módulo 1, la cual coordina las siguientes 2 (dos) acciones:

- El primer ciclo (Figura 78 – cuadro de la izquierda): Con la marca "*flag=51*", comunica la instrucción mediante la variable "*com_forma =15*" a los módulos 4, 9 y 12 para levantar el pie junto con la orden al 6 y 14 hacia atrás, como el 11 hacia adelante, para giro del hexápodo (2 izquierda y 1 derecha). A su vez envía la orden a los módulos 15, 10 y 7 de retornar a la posición 0 mientras se encuentran suspendidas en el aire las patas de los módulos 4, 9 y 12 (lo que genera la acción de giro del sistema en conjunto).

- El segundo ciclo (Figura 78 – cuadro de la derecha): Con la marca "*flag=52*", comunica la instrucción mediante la variable "*com_forma =16*" a los módulos 5, 8 y 13 para levantar el pie junto con la orden al 10 hacia atrás, como el 15, y 7 hacia adelante, para giro del hexápodo (1 izquierda y 2 derecha). A su vez

envía la orden a los módulos 14, 11 y 6 de retornar a la posición 0 mientras se encuentran suspendidas en el aire las patas de los módulos 4, 9 y 12 (lo que genera la acción de desplazamiento del sistema en conjunto).

```
void hexapodo_izquierda()
{
  //para comunicarse con los modulos 4, 9 y 12 para levantar pie junto con 14 y 6 para caminar hacia atras y 11 hacia adelante
  if(flag==51 && datosderecibo[0]==13){modulo_accion=1;modulo_conectar=0;com_forma=15;comunicacion_modulos();flag=52;}
  //para comunicarse con los modulos 5, 8 y 13 para levantar pie junto con 15 y 7 para caminar hacia adelante y 10 hacia atras
  if(flag==52 && datosderecibo[0]==12){modulo_accion=1;modulo_conectar=0;com_forma=16;comunicacion_modulos();flag=51;}
}
```

Código 42. Secuencia de programación en el módulo 1 (maestro) para coordinar el giro hacia la izquierda de la arquitectura hexápodo en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos esclavos, una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con las instrucciones de validación de giro a la izquierda de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=15*" y "*datosderecibo[4]=16*" (ver Tabla 11), según la orden del módulo maestro; se prosigue a ejecutar las líneas de código incluidas en la función principal del controlador de cada módulo esclavo, como se muestra a continuación:

- El primer ciclo (Figura 78 – cuadro de la izquierda) inicia con la actividad de levantar el pie de los módulos 4, 9 y 12, (Código 32 y Código 33).
- Consecuentemente, mientras las patas de los módulos 4, 9 y 12 se encuentran suspendidas en el aire, se ejecuta el movimiento de los módulos 6 y 14 hacia atrás (Código 40), a su vez el módulo 11 hacia adelante (Código 34).

  Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con la validación de la instrucción de levantar el pie en ejecución de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=15*" (ver Tabla 11) según la orden impartida por el módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 43, incluida en la función principal del controlador del módulo 11 la cual señala el llamado de la función "*modulo_hexapodo_pieadelante*", con el uso de la estructura "*states[6]*" (Código 35), a su vez los módulos 6 y 14 ejecutan la línea de código mostrada en el Código 44, para mover el pie atrás.

```
//para mover el Pie del Hexapodo hacia adelante
if(datosderecibo[0]==1 && (datosderecibo[4]==11 || datosderecibo[4]==15) ...
... && datosderecibo[5]==2 && message_printed == 1){(*states[6].func) ();}
```

Código 43. Secuencia de programación en el módulo 11 para ejecutar la instrucción de accionamiento hacia adelante la pata del hexápodo suspendida en el aire - Software WEBOTS. Elaboración propia

```
//para mover el Pie del Hexapodo hacia atras
if(datosderecibo[0]==1 && datosderecibo[4]==15 && datosderecibo[5]==2 && message_printed == 1){(*states[8].func) ();}
```

Código 44. Secuencia de programación en los módulos 6 y 14 para ejecutar la instrucción de accionamiento hacia atrás las patas del hexápodo suspendidas en el aire - Software WEBOTS. Elaboración propia

- Para generar la acción de movimiento de giro a la izquierda de la arquitectura hexápodo, mientras las patas de los módulos 4, 9 y 12 se encuentran suspendidas en el aire y las piernas tomada como los módulos 6 y 14 se mueven hacia atrás, como 11 hacia adelante, se dispone a ejecutar la posición al centro los módulos 7, 10 y 15, conforme a la función mostrada en el Código 37, la cual se ejecuta una vez recibida la instrucción del módulo maestro, mediante la instrucción "*datosderecibo[4]=15*", para proceder a ejecutar la línea de programación mostrada en el Código 38.
- Culminado el primer ciclo de giro a la izquierda del hexápodo mostrado en la Figura 78 (cuadro de la izquierda) y explicado en las tres viñetas (•) anteriores, se prosigue de la misma manera a ejecutar el segundo ciclo (Figura 78 – cuadro de la derecha), comandado por la instrucción "*datosderecibo[4]=16*" tomada desde el módulo maestro 1, con la ejecución de las líneas de programación para levantar las patas del hexápodo, con el accionamiento de los módulos 5, 8 y 13 (Código 32 y Código 33), para luego ejecutar el movimiento hacia adelante de los módulos 7 y 15 (Código 34 y Código 36), como hacia atrás el 10 (Código 40 y Código 44), mientras estas patas se encuentran suspendidas en el aire, junto con la acción de giro del hexápodo, llevando la posición al centro las patas apoyadas en el suelo, mediante el movimiento de los módulos 6, 11 y 14 (Código 37 y Código 38) mientras el otro conjunto de patas se encuentran suspendidas en el aire (módulos 5, 8 y 13).

4.2.6. Coordinación de HEXÁPODO para posición de reposo

La acción de reposo, es la que hace llevar a la posición de centro todas las patas que no se encuentren en dicha posición (Figura 74), por ejecución de los movimientos del hexápodo, como el desplazamiento hacia adelante, también los giros hacia la derecha o izquierda de la arquitectura, lo que implica eventualmente que los módulos que componen las piernas del hexápodo (6, 7, 10, 11, 14 y 15) se encuentren en posición de adelante o atrás, para lo cual ante la instrucción del módulo maestro se hacen retornar a su posición de centro.

La secuencia de movimiento de los módulos para la posición de reposo del sistema hexápodo, se ejecuta estableciendo desde el módulo maestro las sentencias de programación mostradas en el Código 45, que indica la función "*hexapodo_reposo*"

creada en la secuencia de programación para el controlador del módulo 1, la cual coordina las siguientes 2 (dos) acciones:

- El primer ciclo con la marca "*flag=51*", comunica la instrucción mediante la variable "*com_forma =17*" a los módulos 5, 8 y 13 para levantar el pie junto con la orden al 7, 10 y 15 hacia el centro (1 izquierda y 2 derecha).

- El segundo ciclo con la marca "*flag=52*", comunica la instrucción mediante la variable "*com_forma =18*" a los módulos 4, 9 y 12 para levantar el pie junto con la orden al 6, 11 y 14 hacia el centro (2 izquierda y 1 derecha).

```
void hexapodo_reposo()
{
  //para comunicarse con los modulos 5, 8 y 13 para levantar pie junto con con los modulos 15 y 7 para caminar hacia el centro y 10 hacia centr
  if(flag==51 && datosderecibo[0]==13 && message_printed == 1){modulo_accion=1;modulo_conectar=0;com_forma=17;comunicacion_modulos();flag=51;}
  //para comunicarse con los modulos 4, 9 y 12 para levantar pie junto con los modulos 14 y 6 para caminar hacia el centro y 11 hacia el centro
  if(flag==52 && datosderecibo[0]==12 && message_printed == 1){modulo_accion=1;modulo_conectar=0;com_forma=18;comunicacion_modulos();flag=52;}
}
```

Código 45. Secuencia de programación en el módulo 1 (maestro) para coordinar la posición de reposo de la arquitectura hexápodo en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos esclavos, una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con las instrucciones de validación de posición de reposo de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=17*" y "*datosderecibo[4]=18*" (ver Tabla 11), según la orden del módulo maestro; se prosigue a ejecutar las líneas de código incluidas en la función principal del controlador de cada módulo esclavo para levantar las patas (Código 33) y llevarlas a la posición de centro (Código 37).

4.2.7. Coordinación de HEXÁPODO para posición inicial

Una vez llevada la posición de reposo la arquitectura hexápodo, para volver a la posición de inicial mostrada en la Figura 74, la acción de acostar el hexápodo, consiste en llevar a la posición de centro todos los módulos que componen las patas de la arquitectura estableciendo desde el módulo maestro las sentencias de programación mostradas en el Código 46, que indica la función "*hexapodo_abajo*" para dar la instrucción a los módulos 4, 5, 8, 9, 12 y 13 de posición de inicio de la arquitectura.

```
void hexapodo_abajo()
{
  //para comunicarse con los modulos 4, 5, 8, 9, 12 y 13 para bajar el hexapodo
  if(((flag==51 && datosderecibo[0]==13) || (flag==52 && datosderecibo[0]==12)) && message_printed == 1){
    modulo_accion=1;modulo_conectar=0;com_forma=19;comunicacion_modulos();flag=53;}
}
```

Código 46. Secuencia de programación en el módulo 1 (maestro) para coordinar la posición inicial de la arquitectura hexápodo en conjunto con los demás módulos- Software WEBOTS. Elaboración propia

Por otra parte, para los módulos que se comportan como las patas del hexápodo, se ejecuta con la secuencia de programación creada en la función "*modulo_hexapodo_abajo*", mostrada en el Código 47, la cual retorna a 0 grados el motor del pivote de los módulos 4, 5, 8, 9, 12 y 13.

```
static void modulo_hexapodo_abajo()
{
  wb_motor_set_position(motor_pivote,0*PI/180);
  wb_motor_set_velocity(motor_pivote,2);
  wb_robot_step(1000);
}
```

Código 47. Secuencia de programación en los módulos 4, 5, 8, 9, 12 y 13 para llevar a la posición de inicio al hexápodo (acción de acostarse) mediante el accionamiento del motor del pivote- Software WEBOTS. Elaboración propia

Una vez recibida la instrucción desde el módulo maestro 1, mediante la validación de la identificación asignada a todos (variable "*datosderecibo[0]=1*" ver Código 12), junto con las instrucción de validación de posición de inicio de la arquitectura hexápodo, que para el caso es "*datosderecibo[4]=19*" (ver Tabla 11), según la orden del módulo maestro; se prosigue a ejecutar la línea de código mostrada en el Código 48, incluida en la función principal del controlador de los módulos en mención, la cual señala el llamado de la función "*modulo_hexapodo_abajo*", con el uso de la estructura "*states[8]*" (Código 29).

```
//para colocar el Hexapodo abajo
if(datosderecibo[0]==1 && datosderecibo[4]==19 && datosderecibo[5]==2 && message_printed == 1){(*states[8].func) ();}
```

Código 48. Secuencia de programación en los módulos 4, 5, 8, 9, 12 y 13 para ejecutar la instrucción de posición de inicio de las patas que conforman el hexápodo - Software WEBOTS. Elaboración propia

4.3. Estrategia de Desarmado Arquitectura ORUGA

Conforme a las herramientas planteadas y explicadas anteriormente para la comunicación y desconexión entre módulos, el desplazamiento individual de los mismos y la secuencia de armado de la arquitectura oruga planteada, se crearon las estrategias de desarmado automático de ésta, lo que para el proyecto se toma como una tercera arquitectura para efectos de la programación en cada módulo.

4.3.1. Secuencia de desarmado arquitectura ORUGA

La secuencia de desarmado de la arquitectura oruga, consiste en desconectar los módulos en orden contrario al de la conformación de la primer arquitectura (Figura 72), es decir desde el módulo 15 en sentido decreciente y ubicarlos de forma que queden disponibles para ejecutar el armado de la arquitectura hexápodo (Figura 74) en caso de requerirse, sin estropearse ni obstaculizarse entre sí, siendo desacoplados éstos por los conectores de la rueda pivote del de la rueda posterior del otro (señalados con los números 6 y 9 en Tabla 2 respectivamente); la secuencia

de desarmado que es liderada por el módulo 1, ejecuta la organización de los módulos mostrada en la Figura 79 para su configuración, de acuerdo al número asignado a cada módulo para su identificación, para lo cual la línea roja representa el motor de la rueda pivote del módulo, parte frente del mismo (número 2, Tabla 2) en el sentido de orientación.

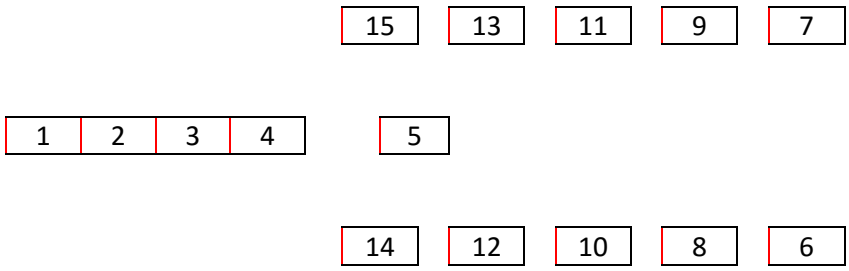
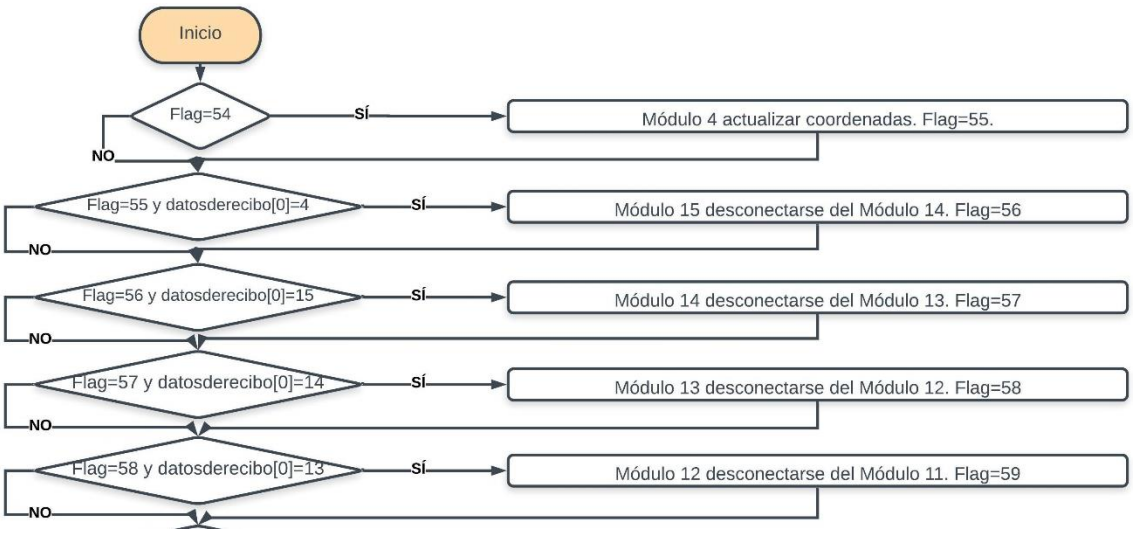


Figura 79. Esquema de conformación desarmado arquitectura oruga, vista desde arriba. Elaboración propia

4.3.2. Algoritmo y Pseudocódigo para controlar el desarmado arquitectura ORUGA

Para ejecutar el desarmado de la arquitectura oruga, para el módulo maestro 1 se definió la estructura secuencial como se muestra en la Figura 80, registrándose los estados en la variable "*flag*", con la cadencia en la ejecución de la siguiente instrucción dependiendo de la variable "*datosderecibo[0]*", que indica cuando se recibe la señal en el módulo maestro de que el módulo esclavo en acción ha terminado la orden y devuelto la información de acuerdo al protocolo de comunicación (Código 14). La estructura que simboliza lo definido en la secuencia de programación del controlador del módulo 1, se generó en la función "*desacople_oruga*", la cual se encuentra detallada en el **Anexo 12**.



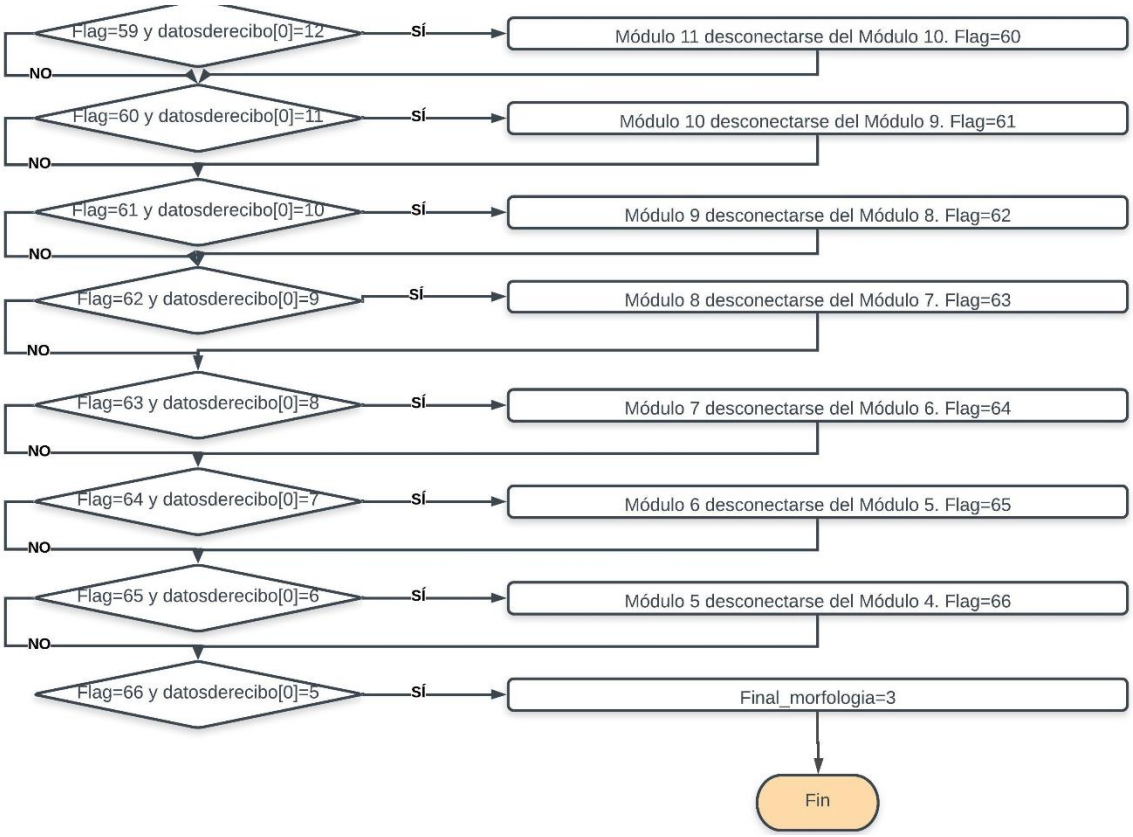


Figura 80. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) de la estrategia de desarmado de la arquitectura oruga con los demás módulos del 4 al 15 (esclavos). Elaboración propia

Como se puede apreciar en la Figura 80, el desarmado de la arquitectura oruga, empieza en el estado representado por la marca "*flag=54*", lo que indica que una vez registrado este estado, procede a comunicarse con los demás módulos sucesivamente desde el 15 hasta el 5 (sentido decreciente), para que se desacoplen de la rueda posterior de su predecesor, para lo cual se define la variable "*com_forma=22*" (ver Tabla 11) como instrucción a impartir para la desconexión de cada módulo en la función "*desacople_oruga*", (**Anexo 12**).

El final del desarmado de la arquitectura oruga, está representado por la marca "*flag=66*" y la comunicación recibida de acople del módulo 5 en la variable "*datosderecibo[0]*" (Figura 73), para proceder entonces a definir la variable "*final_morfologia=3*" (variable declarada y mostrada en **Anexo *1***), tomada como otra marca que lleva el registro de culminación del desarmado de la arquitectura oruga.

- Algoritmo de control para el desarmado de la arquitectura oruga en los módulos esclavos 5 al 15

Por otra parte, para los módulos esclavos del 5 al 15, una vez recibida la instrucción desde el módulo maestro 1 al módulo esclavo correspondiente, mediante la

validación de la identificación asignada a cada uno (variable "*id*" con "*datosderecibo[0]*" ver Tabla 12 y Código 12), junto con la validación de la instrucción desacople del módulo conectado, que para el caso es "*datosderecibo[4]*"=22 (ver Tabla 10); se prosigue a ejecutar la línea de código mostrada en el Código 49, incluida en la función principal del controlador de cada módulo esclavo (5 al 15), la cual señala la acción de desconexión de la rueda del módulo acoplado y posterior desplazamiento del módulo hasta el punto final de ubicación (Figura 79).

```
//para desacoplar el modulo de la arquitectura ORUGA
if(datosderecibo[0]==id && datosderecibo[4]==22 && datosderecibo[5]==3){
  if(message_printed == 1){wb_connector_unlock(conector_pivote);for(i=0;i<60;i++){(*states[1].func) ();}}
  else {desplazamiento_modulo();}
}
```

Código 49. Secuencia de programación para ejecutar el desarmado de la arquitectura oruga, mediante la desconexión del módulo acoplado y desplazamiento de los módulos esclavos del 5 al 15– Software WEBOTS. Elaboración propia

Ejecutado el código para el desacople del módulo (Código 49), el módulo en acción se mueve hasta el punto final de ubicación, mediante la ejecución del algoritmo de desplazamiento (mostrado en la Figura 63), integrando la ubicación dos puntos nuevos de bordeo (P_1 y P_2), los cuales a manera de ejemplo se pueden apreciar en la Figura 81 para el módulo 15 (azul) y 14 (naranja), utilizando como referencia las coordenadas del módulo 4 (punto negro) para su ubicación; a su vez, para lograrlos diferenciar con los puntos expuestos en la Figura 62 y Tabla 13, se incluye como condición el parámetro "*datosderecibo[4]*"=22 (ver Tabla 10) en la función "*puntos_bordeo*" para el módulo 5 hasta el 15, los cuales se presenta el detalle en el **Anexo 13** de las sentencias incluidas para cada controlador.

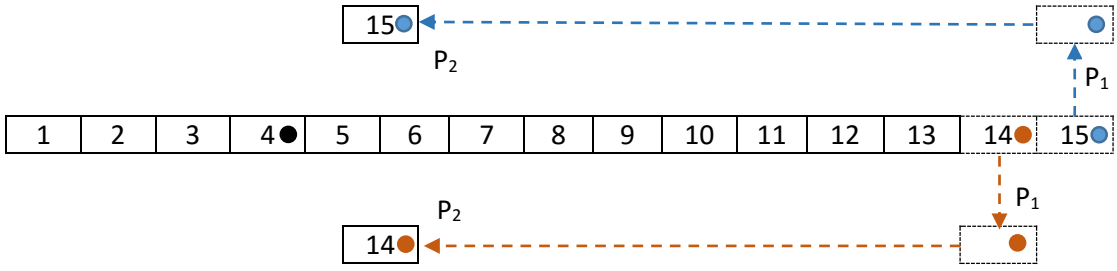


Figura 81. Esquema de conformación desarmado arquitectura oruga, vista desde arriba. Elaboración propia

Con el cálculo de los nuevos puntos de bordeo para el desarmado de la arquitectura oruga, igualmente se incluyeron las líneas mostradas en el Código 50, para definir la orientación y la selección del próximo punto de bordeo, dentro de las funciones "*orientacion_ptobordeo*" (recuadro amarillo) y "*proximo_ptobordeo*" (recuadro verde) respectivamente, explicadas en la Figura 63 y mostradas al detalle en el ***Anexo 6*** para el desplazamiento de los módulos.

```
  //orientacion de puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
  if(datosderecibo[4]==22){
    if(puntomin==2){orientacion_final=datosderecibo[3];printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==1){orientacion_final=datosderecibo[3];printf("listo 6.1: calculo de orientacion final\n");}
  }
```

```
//proximo punto de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  if(puntomin==2){flag=20;printf("listo 8: giro hacia la orientacion final\n");}
  else if(puntomin==1){puntomin=2;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
}
```

Código 50. Secuencia de programación para seleccionar orientación y próximo punto de bordeo en el desplazamiento de los módulos esclavos del 5 al 15– Software WEBOTS. Elaboración propia

La desconexión (Código 49) y desplazamiento de cada módulo al punto de posición final (Figura 79), conforme los nuevos puntos de bordeo (Figura 81), concluye con la respuesta de comunicación al módulo principal; para lo cual, se ingresó en la función "*desplazamiento_modulo*" de cada módulo (***Anexo 6***), las líneas de programación mostradas en el Código 51, que de acuerdo al estado registrado en la variable "*flag=20*", indica la llegada a la posición final del módulo en acción (Código 50– recuadro verde), para proceder a comunicarse con el módulo maestro y así continuar con la ejecución de desarmado con los demás módulos.

```
//para comunicarse con el otro modulo (modulo 1) para avisar que ya desacoplo y queda listo para ejecutar la otra arquitectura
else if(flag==20){comunicacion_modulos();flag=21;printf("listo 12: despegado de modulo y envio de datos al modulo 1\n");}
```

Código 51. Secuencia de programación para enviar la información al módulo maestro en la comunicación entre módulos durante la ejecución del desarmado de la arquitectura oruga – Software WEBOTS. Elaboración propia

4.4. Estrategia de Desarmado Arquitectura HEXÁPODO

Conforme a las herramientas planteadas y explicadas anteriormente para la comunicación entre módulos, el desplazamiento individual de los mismos y la secuencia de armado de la arquitectura hexápodo planteada, se crearon las estrategias de desarmado automático de ésta, lo que para el proyecto se toma como una cuarta arquitectura para efectos de la programación en cada módulo.

4.4.1. Secuencia de desarmado arquitectura HEXÁPODO

La secuencia de desarmado de la arquitectura hexápodo, consiste en desconectar los módulos en orden contrario al de la conformación de la segunda arquitectura (Figura 74), es decir desde el módulo 4 en sentido creciente y ubicarlos de forma que queden disponibles para ejecutar el armado de la arquitectura oruga (Figura 72) en caso de requerirse, sin estropearse ni obstaculizarse entre sí; la secuencia de desarmado que es liderada por el módulo 1, ejecuta la organización de los módulos mostrada en la Figura 82 para su configuración, de acuerdo a la enumeración en el círculo de color "naranja".

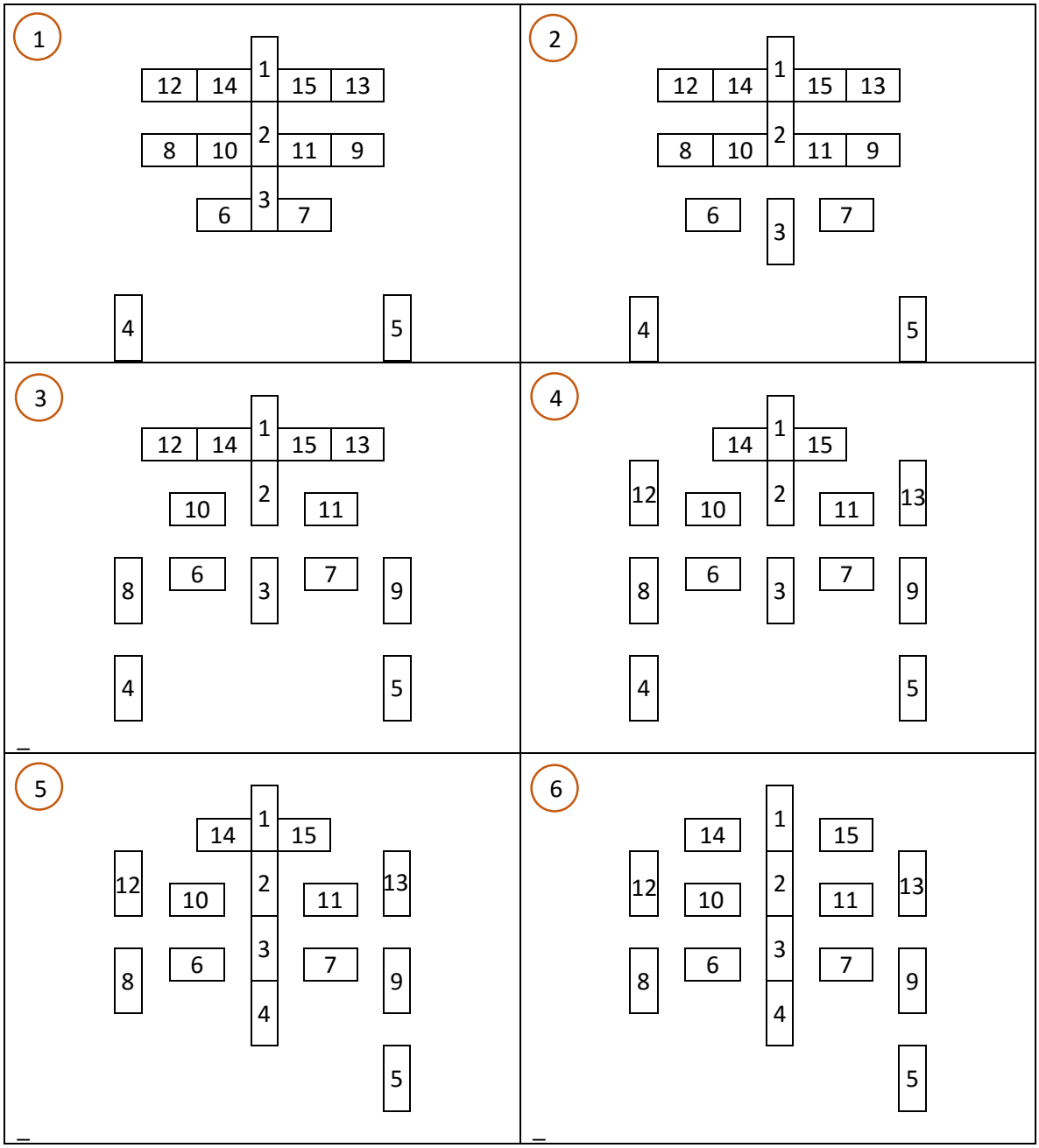
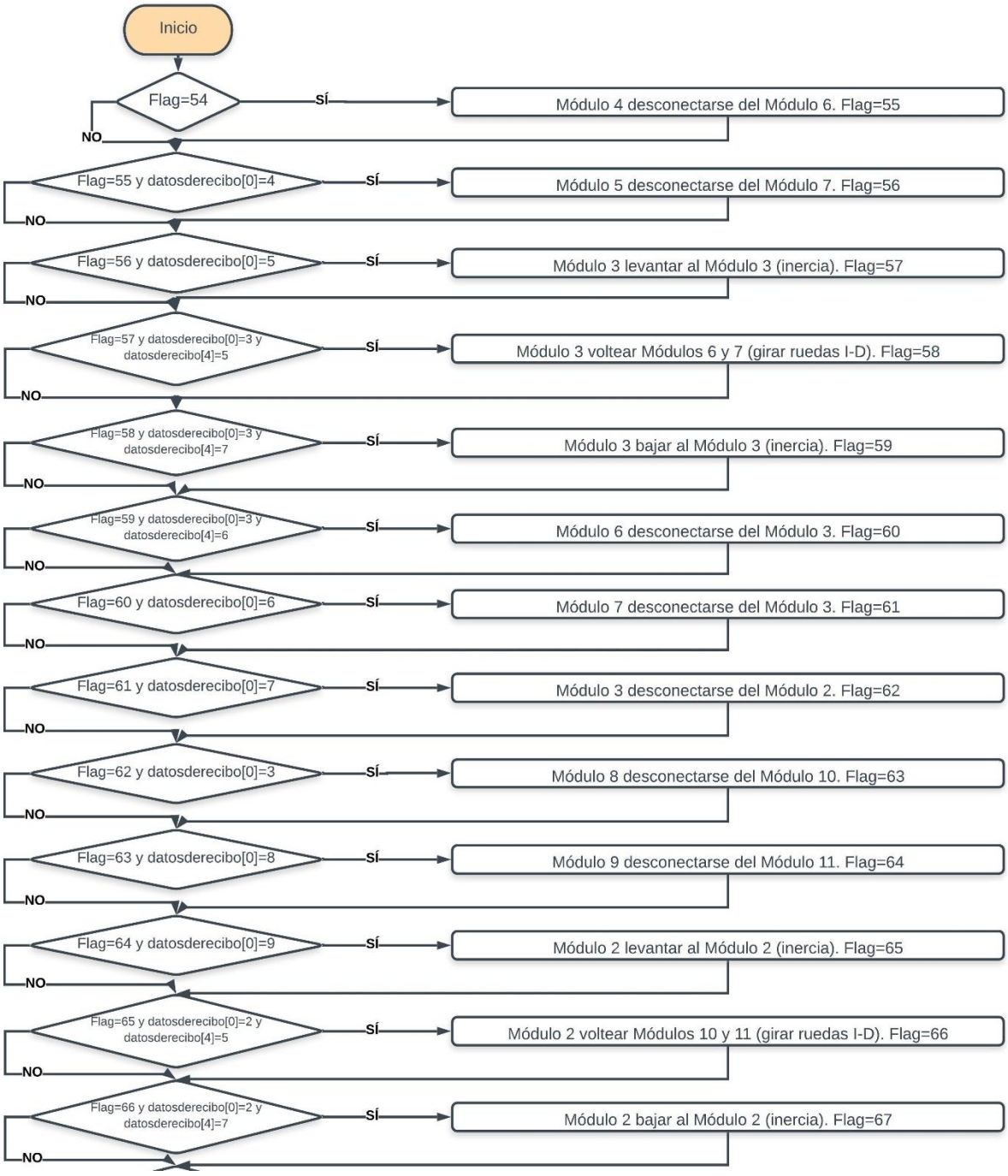


Figura 82. Esquema de desarmado de la arquitectura hexápodo, vista desde arriba. Elaboración propia

4.4.2. Algoritmo y Pseudocódigo para controlar el desarmado arquitectura HEXÁPODO

Para ejecutar el desarmado de la arquitectura hexápodo, para el módulo maestro 1 se definió la estructura secuencial como se muestra en la Figura 83, registrándose los estados en la variable "*flag*", con la cadencia en la ejecución de la siguiente instrucción dependiendo de la variable "*datosderecibo[0]*" y "*datosderecibo[4]*", que

indica cuando se recibe la señal en el módulo maestro de que el módulo esclavo en acción ha terminado la orden y devuelto la información de acuerdo al protocolo de comunicación (Código 14). La estructura que simboliza lo definido en la secuencia de programación del controlador del módulo 1, se generó en la función "*desacople_hexapodo*", la cual se encuentra detallada en el **Anexo 14**.



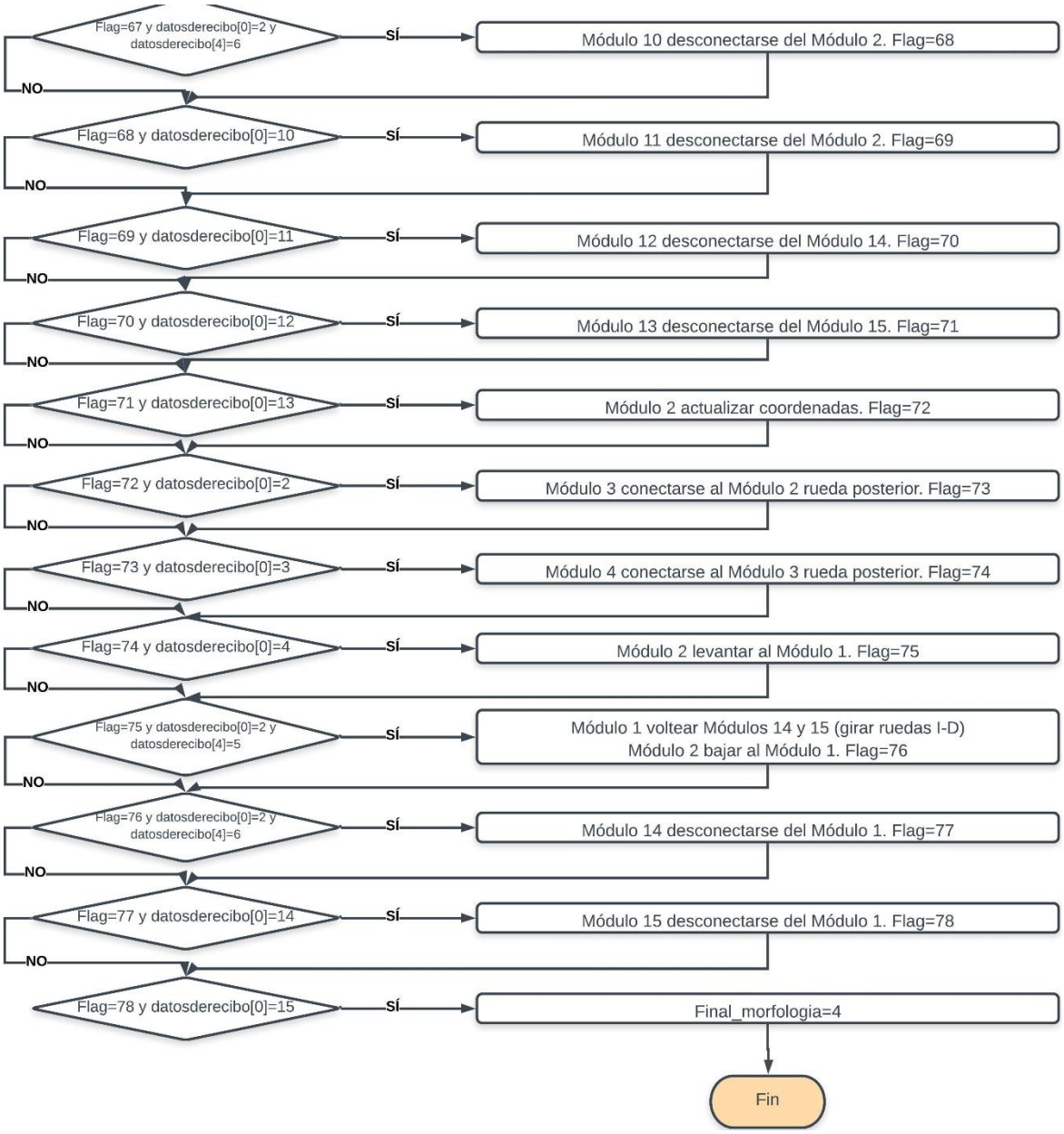


Figura 83. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) de la estrategia de desarmado de la arquitectura hexápodo con los demás módulos del 2 al 15 (esclavos). Elaboración propia

- Algoritmo de control para el desarmado de la arquitectura hexápodo en los módulos esclavos 4 al 15

Por otra parte, para los módulos esclavos del 4 al 15, una vez recibida la instrucción desde el módulo maestro 1 al módulo esclavo correspondiente, mediante la validación de la identificación asignada a cada uno (variable "*id*" con "*datosderecibo[0]*" ver Tabla 12 y Código 12), junto con la validación de la instrucción desacople del módulo conectado, que para el caso es "*datosderecibo[4]*"=21 (ver

Tabla 10); se prosigue a ejecutar la línea de código mostrada en el Código 52, incluida en la función principal del controlador de cada módulo esclavo (4 al 15), la cual señala la acción de desconexión de la rueda del módulo acoplado, posterior desplazamiento del módulo hacia atrás, giro a la derecha y desplazamiento hacia adelante cierta cantidad de pasos dependiendo sea el caso de cada módulo, según lo mostrado en la Figura 82; finalmente la acción concluye con la respuesta de comunicación al módulo principal, para continuar con la ejecución de desarmado con los demás módulos.

```
//para desacoplar el modulo de la arquitectura HEXAPODO
if(datosderecibo[0]==id && datosderecibo[4]==21 && datosderecibo[5]==4 && message_printed == 1){
  wb_connector_unlock(conector_pivote);for(i=0;i<80;i++){(*states[1].func) ();}
  for(i=0;i<111;i++){(*states[3].func) ();}for(i=0;i<200;i++){(*states[0].func) ();}
  comunicacion_modulos();printf("listo 12: despegado de modulo y envio de datos al modulo 1\n");}
```

Código 52. Secuencia de programación para ejecutar el desarmado de la arquitectura hexápodo, mediante la desconexión de la rueda del módulo acoplado y posterior desplazamiento de los módulos esclavos del 5 al 15– Software WEBOTS. Elaboración propia

4.5. Conclusiones del Capítulo

En este capítulo se presentó el diseño y desarrollo del algoritmo de armado y desarmado para las arquitecturas oruga y hexápodo, con el uso del software WEBOTS para su simulación, teniendo en cuenta los trabajos previamente elaborados en el grupo DAVINCI de la Universidad Militar Nueva Granada, sin la asistencia de la herramienta *Supervisor* que brinda el Software dispuesto para este proyecto.

Con las herramientas formadas para brindar la capacidad de desplazamiento, de conexión y de comunicación, de forma autónoma, integrada y sincronizada para ejecutar las arquitecturas propuestas a cada módulo, se estableció a partir del empoderamiento del No. 1 como el maestro de la operación, las estrategias de armado de las arquitecturas oruga y hexápodo, junto con las secuencias de movimiento individual para el desplazamiento en conjunto de cada arquitectura, como el procedimiento de desarmado de cada una y reubicación individual de cada módulo para quedar en disposición de ejecutar cualquiera de las dos arquitecturas planteadas para el presente proyecto, con el fin de ejecutar una trayectoria previamente planificada bajo los parámetros establecidos en el siguiente capítulo.

CAPÍTULO V.

DISEÑO Y DESARROLLO DE ALGORITMO DE BÚSQUEDA PARA PLANIFICACIÓN DE TRAYECTORIAS Y DE AUTOCONFIGURACIÓN DE LAS ARQUITECTURAS ORUGA Y HEXAPODO PARA RECORRER LAS RECTAS OBTENIDAS

Tomando como base los trabajos elaborados en el grupo DAVINCI de la Universidad Militar Nueva Granada, se propuso realizar un trabajo que confluyera en realizar el diseño del control de los módulos MECABOT 4, con la capacidad de acoplarse con los módulos entre sí, para que de esta manera a través de protocolos de comunicación, se pueda lograr el armado sincronizado de varias arquitecturas, anteriormente presentadas por tesis anteriores en el programa de Ingeniería Mecatrónica, como lo son las arquitecturas oruga y hexápodo, con la característica de desarmado autónomo y la autoconfiguración de arquitecturas para ejecutar una trayectoria previamente planificada por el algoritmo de búsqueda A*.

Los desarrollos mostrados en el capítulo anterior, para la adaptación morfológica de las arquitecturas de oruga y hexápodo, su secuencia de armado y desarmado, requiere tener en cuenta unos parámetros mínimos al momento de buscar y planificar la mejor trayectoria para el óptimo funcionamiento, como distancias

mínimas para su acoplamiento entre el conjunto de módulos ya acoplados y los diferentes obstáculos puestos en el escenario de trabajo, incluyendo las paredes del mismo; con el fin de dar robustez al sistema y la capacidad de generar rutas acordes a la complejidad morfológica de las dos arquitecturas implementadas.

5.1. Caracterización de escenario en Software WEBOTS como espacio de trabajo para desarrollo de algoritmo de búsqueda A*

Como se mencionó en el marco teórico, el algoritmo de búsqueda A* (A-Star)[15], parte de la base de dividir el espacio de trabajo en una rejilla rectangular para obtener celdas que permitan trazar una trayectoria a medida que no se encuentren obstáculos, en la búsqueda de la mejor ruta; concepto aplicado en el escenario definido en el Software WEBOTS, para que éste fuera utilizado como una malla de nodos, en la que se definiera la cantidad de filas y columnas del mismo, que permitiera la ubicación nodal del sistema robótico en ejecución de cada una de las arquitecturas planteadas para el presente proyecto, junto con los obstáculos ubicados en el escenario.

Teniendo en cuenta las dimensiones de cada módulo, considerando que el mismo no supere las dimensiones de cada nodo, se generó sobre el escenario WEBOTS, una rejilla como espacio de trabajo de 40 filas por 40 columnas, con dimensiones de 0,25 metros para la longitud de cada cara que conforma el cuadrado nodal, lo que representa un espacio total de trabajo de 10 metros cuadrados; como se puede apreciar en la Figura 84, se muestra la ubicación del origen del plano cartesiano con respecto a la malla del mapa de trabajo, el cual se estableció en el centro del mapa, lo que indica que la coordenada 0 en los ejes “*x*” y “*z*”, se encuentra en el nodo de la fila 20 y columna 20. Por otra parte, El nodo de la fila 1 y columna 1, corresponde a las coordenadas de 5 metros en los ejes “*x*” y “*z*”, por lo que a partir de ese punto, se distancia cada nodo 0,25 metros en ambos ejes hasta llegar al nodo de la fila 40 y columna 40, correspondiente a las coordenadas de -5 metros en los ejes “*x*” y “*z*”.

[15] Numeral 2.1.10.2, página 50.

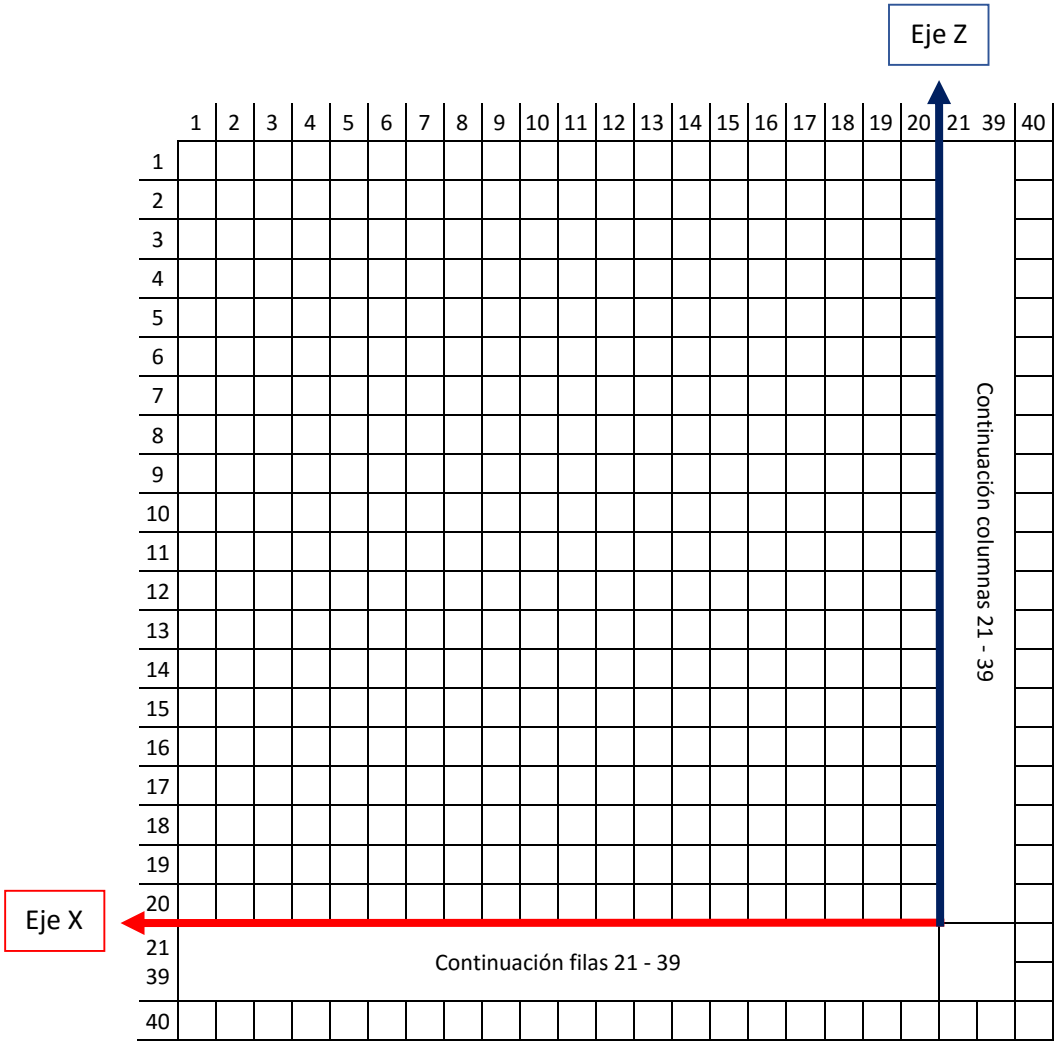


Figura 84. Caracterización escenario de Software WEBOTS para malla de trabajo, vista desde arriba. Elaboración propia

Acorde a los parámetros establecidos para crear la malla de trabajo sobre el escenario del Software WEBOTS, se procedió a definir la malla sobre el objeto "*floor*" (mostrado en la Figura 50), de forma que la propiedad "estilo ajedrez" del suelo, permitiera representar cada cuadro como un nodo considerando las dimensiones descritas, con el fin de fijar las medidas correspondientes (mostradas en Figura 84) en el nodo del Software WEBOTS "*RectangleArena*" tal y como se indica en la Figura 85, donde se definió a su vez la percepción de muro para delimitar el perímetro del espacio de trabajo, con los campos "*wallThickness*" y "*wallHeight*".

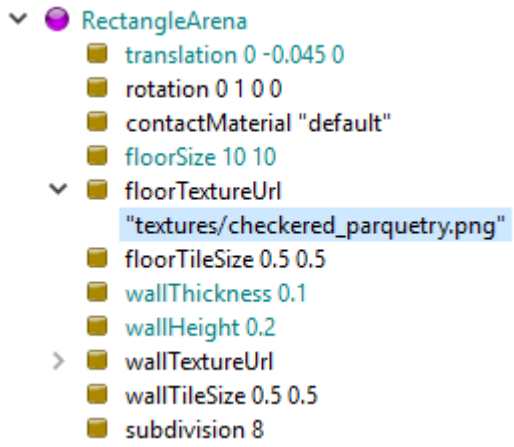


Figura 85. Jerarquía Nodo "RectangleArena", parametrización espacio de trabajo – Software WEBOTS. Elaboración propia

Se prosigue entonces con el objetivo de identificar las restricciones del escenario y las condiciones para superar los eventuales obstáculos planteados, con el fin de definirlos en el algoritmo desarrollado, de forma que éstos no afecten el sistema durante la ejecución de las dos arquitecturas, para que de esa manera sea el algoritmo de búsqueda de trayectorias el que se adapte y encuentre la mejor ruta acorde a los parámetros establecidos, en lugar del caso contrario a éste propósito, en donde se termine en la imposibilidad de ajustar la configuración morfológica, por no haber considerado propiamente el sistema, dichas condiciones previsibles en el terreno.

5.2. Condiciones y restricciones del escenario y puesta de obstáculos

En la Figura 86, se hace una representación de las condiciones del escenario que influyen en la autoconfiguración y el desplazamiento de las dos arquitecturas planteadas en el presente proyecto, como también las restricciones que presentan las mismas arquitecturas para permitir la adaptación de una morfología oruga a hexápodo o viceversa, las mismas se presentan a continuación:

- Como condición, se trabajó bajo el supuesto que para llevar el sistema robótico modular con capacidad de autoconfiguración de las 2 (dos) arquitecturas, representado en color "verde" en la imagen, hasta el nodo objetivo ("amarillo"), éste debía detectar como áreas de seguridad un ancho de 2 (dos) cuadros adicionales de más a las paredes y a los obstáculos (representados con color "rojo" y "azul" en la imagen respectivamente), en atención que para la morfología hexápodo (Figura 74), el centro de las coordenadas, se encuentra en el módulo 2 y se debe contemplar el ancho del sistema, es decir la extensión de las patas en el movimiento, de modo

que el desplazamiento al frente o giros en dicha arquitectura, no genere eventuales colisiones con los obstáculos puestos en el terreno.

- Como restricción, se tiene que para ejecutar la arquitectura oruga, debe haber un espacio libre de obstáculos (incluyendo áreas de seguridad) de 7 cuadros detrás del módulo 1, en las rectas de la trayectoria encontrada con más de 7 nodos de desplazamiento.

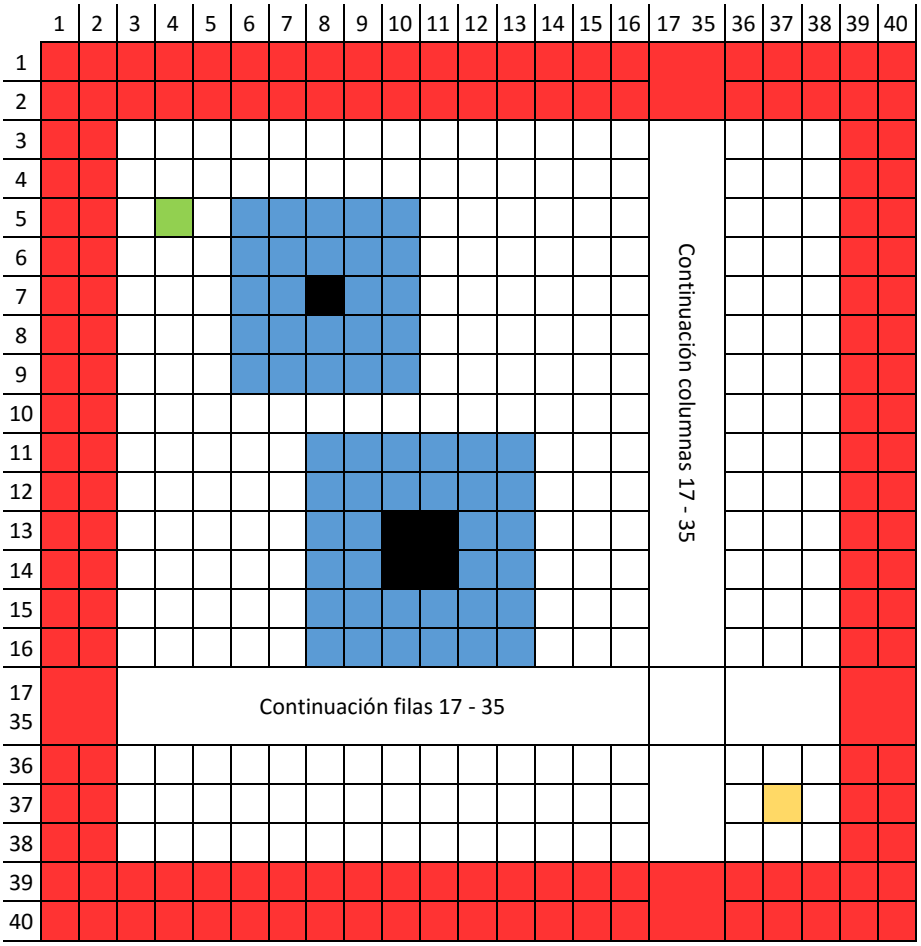


Figura 86. Esquema de restricciones del escenario y condiciones para puesta de obstáculos, vista desde arriba. Elaboración propia

5.3. Algoritmo y Pseudocódigo para ejecutar el método de búsqueda y planificación de trayectorias A*

Para ejecutar el algoritmo de búsqueda y elección de trayectorias, mediante metodología A* (A-Star), para el módulo maestro 1 se definió la estructura secuencial como se muestra en la Figura 87, mediante la ejecución secuencial de expresiones lógicas y matemáticas, para determinar la ubicación de los obstáculos, la búsqueda de trayectorias, la selección de la mejor ruta, la partición de rectas de

la trayectoria escogida y selección de arquitecturas a ejecutar para recorrer cada recta. La estructura que simboliza lo definido en la secuencia de programación del controlador del módulo 1, se generó en la función "*algoritmo_a_star*", la cual se encuentra detallada en el **Anexo 15**.

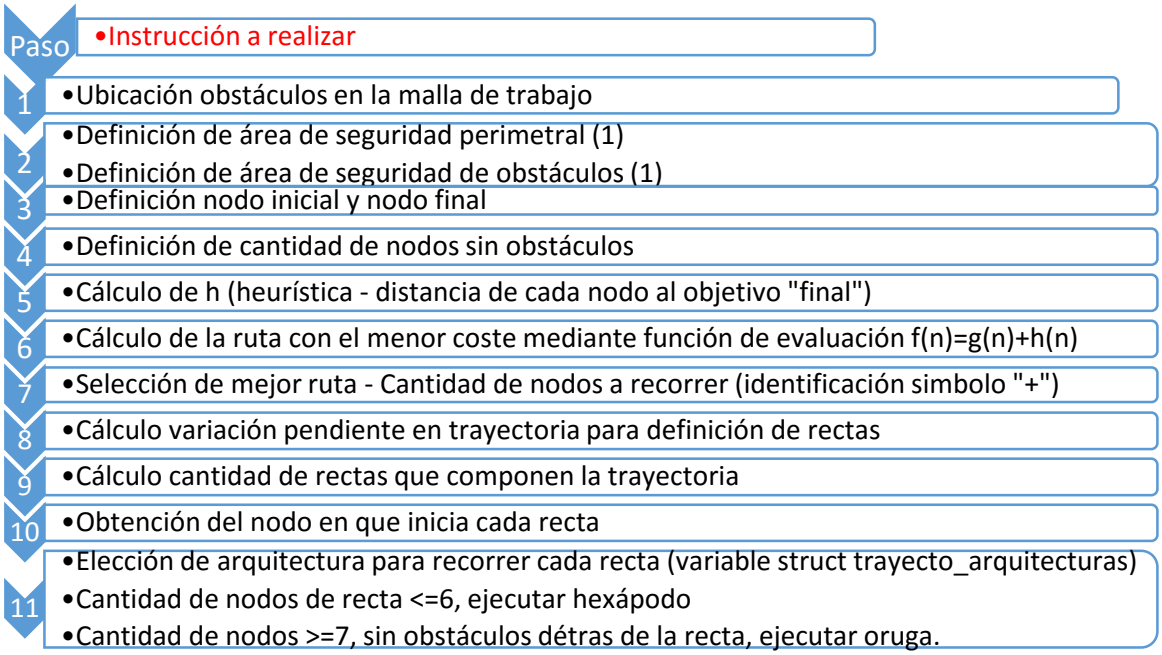


Figura 87. Estructura secuencial para ejecutar por el módulo 1 (maestro) la búsqueda, selección de trayectorias, partición de rectas y selección de arquitecturas para el desplazamiento. Elaboración propia

Como se puede apreciar en la Figura 87, la secuencia para la búsqueda y selección de trayectorias, inicia con la ubicación y puesta de obstáculos, la cual como condición inicial, se establecieron 5 (cinco) obstáculos con forma rectangular, destacando que éstos pueden variar si el usuario así lo desea, conforme se muestra en el Código 53, a pesar que el programa no toma la cantidad ni la ubicación de los objetos puestos en el escenario, es posible recrear y ajustar la cantidad mediante la variable "*cantidad_cubos*", como también la estructura obstáculo, que se ajusta dinámicamente conforme el redimensionamiento de la misma mediante la función de la librería C "*realloc*" (tipo arreglo), se puede ver el detalle en el **Anexo 15**.

```
cantidad_cubos=5;
obstaculos = (struct obstaculo *)realloc(obstaculos, 1 * sizeof(struct obstaculo));
obstaculos[0].obst_coord_n = 10;
obstaculos[0].obst_coord_m = 20;
obstaculos[0].obst_ancho = 3;
```

Código 53. Secuencia de programación para variar la cantidad de obstáculos en el código de programación, conforme a los puestos en el escenario – Software WEBOTS. Elaboración propia

El resultado se registra en la variable tipo estructura "*trayecto_arquitecturas*" (variable declarada y mostrada en **Anexo *1***), la cual dinámicamente ajusta su

tamaño, en razón a las rectas que componen la trayectoria obtenida, para llevar el registro del número de línea a recorrer, la cantidad de nodos que componen la línea, la arquitectura a ejecutar para hacer el recorrido y el nodo en el que inicia la recta.

5.4. Segmentación de rectas sobre la trayectoria planificada

Con respecto a la trayectoria obtenida, resultante de la ejecución del algoritmo de búsqueda A* (Figura 87), se procede entonces a obtener la cantidad de rectas que conforman la trayectoria, utilizando como método la variación de pendientes en el ángulo de cada punto nodal con respecto al anterior, ángulo medido igualmente con respecto a la horizontal del eje coordenado, para lo cual se estableció como nuevo origen del sistema coordenado el nodo de la fila 0 y columna 0, mostrado en la Figura 88, lo cual representa como coordenadas -5 metros tanto en el eje "*x*" como en el eje "*z*" del sistema coordenado en el escenario de trabajo del Software WEBOTS (Figura 84**¡Error! No se encuentra el origen de la referencia.**).

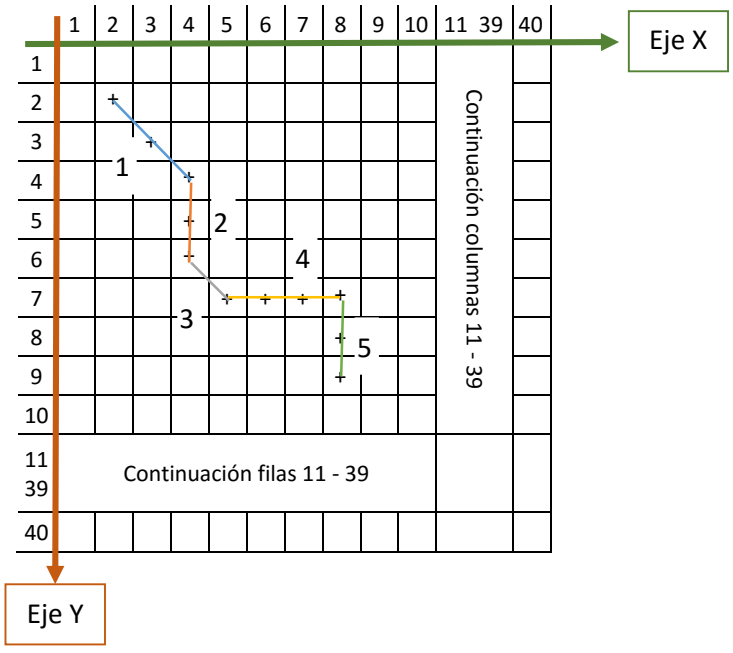


Figura 88. Ubicación nuevo sistema coordinado para malla de trabajo a base de nodos, vista desde arriba. Elaboración propia

Para la obtención del ángulo, de un punto con respecto al anterior, como se muestra en la Figura 89, se manejó 8 diferentes posibilidades, mismas opciones de desplazamiento entre un nodo al otro en la conformación de la trayectoria como a manera de ejemplo se representa en la Figura 88, la cual muestra un recorrido de 11 (once) nodos, conformado por 5 (cinco) rectas y presenta las características mostradas en la Tabla 14.

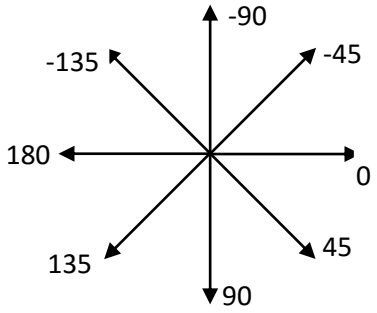


Figura 89. Posibilidad de ángulos para obtener la pendiente entre un punto y otro en la conformación de rectas de la misma trayectoria, vista desde arriba. Elaboración propia

Conforme a la trayectoria ilustrada en la Figura 88 y la forma de hallar las pendientes mostrada en la Figura 89, para la segmentación de rectas, se muestra en la Tabla 14, a manera de ejemplo la metodología utilizada para obtener la cantidad de rectas que componen la trayectoria obtenida por el método de búsqueda A*; secuencia de programación mostrada en detalle en el **Anexo 15**.

Tabla 14. Ejemplo de segmentación de rectas de la trayectoria obtenida. Elaboración propia

Recta	Pendiente	Cantidad de Nodos	Nodo que Inicia	Fila de Nodo (Eje y)	Columna de Nodo (Eje x)
1	45°	2	1	2	2
2	90°	2	3	4	4
3	45°	1	5	6	4
4	0°	3	6	7	5
5	90°	2	9	7	8

Para la segmentación de las rectas, es importante hacer notar, el evento que se puede presentar en la aplicación y simulación del algoritmo, como resultante de dos rectas sobre una sola, cualidad denominada doble segmentación de una misma recta, si llegado el caso para el armado de una arquitectura oruga se hace necesario para su ejecución, generar el espacio suficiente, libre de obstáculos, a través del desplazamiento en hexápodo nodos adelante del inicio de la recta, hasta dejar el espacio suficiente para ensamble de la arquitectura oruga (7 nodos atrás), bajo la condición de seguir contando con al menos 7 nodos adelante en la misma recta, lo que resulta en el desplazamiento corto en arquitectura hexápodo hasta el nuevo punto de inicio de la arquitectura oruga para una misma recta.

Desarrollado y obtenido el algoritmo para brindar a los módulos MECABOT, la capacidad de acoplarse entre sí, para el armado y desarmado sincronizado de las arquitecturas oruga y hexápodo, mediante la ejecución de protocolos de comunicación, junto con la generación de las líneas de programación para simular las condiciones físicas del terreno y la ejecución del algoritmo A* para la búsqueda

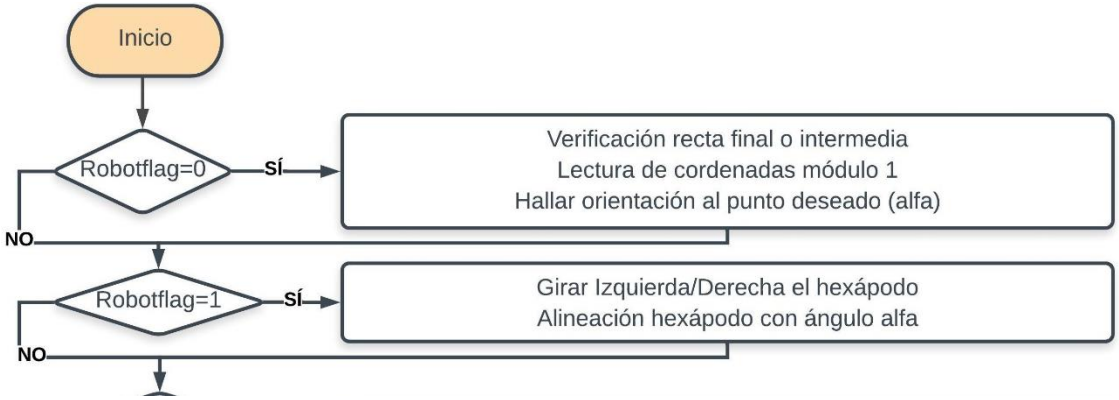
y planificación de la mejor trayectoria, se prosigue entonces a unificar ambos frentes para obtener un sistema robótico integral que sea capaz de encontrar la mejor trayectoria al punto deseado y proceda entonces a recorrerla haciendo uso eficiente en la autoconfiguración de las arquitecturas disponibles para lograrlo.

5.5. Algoritmo y Pseudocódigo de Autoconfiguración de Arquitecturas para ejecutar el Recorrido de la Trayectoria Obtenida por el Método de Búsqueda A*

Para dotar al sistema la capacidad de autoconfiguración de las arquitecturas oruga y hexápodo, que permita ejecutar el recorrido de la trayectoria obtenida, mediante la metodología A* (A-Star), se definió la ejecución de las morfologías planteadas en razón a la recta a recorrer resultado del algoritmo de búsqueda (Figura 87), a través de la lectura de la variable tipo estructura "*trayecto_arquitecturas",* junto con la cadencia en la ejecución de los estados del sistema en conjunto, en la variable "*robotflag*", como también la posibilidad de ejecución de los desplazamientos de las arquitecturas de acuerdo al uso de la variable "*final_morfologia*", que permite conocer al sistema la culminación de armado o desarmado de alguna arquitectura. La estructura que simboliza lo definido en la secuencia de programación del controlador del módulo 1, se generó en la función "*ejecutar_trayectoria*", la cual se encuentra detallada en el **Anexo 16**, ésta se compone de cuatro partes para la ejecución del armado y desarmado de las arquitecturas oruga y hexápodo, conforme a lo dispuesto para cada recta, como se detallan a continuación:

5.5.1. Cuando la arquitectura armada para recorrer la recta es hexápodo

Condición *"trayecto_arquitecturas[recta_ejecucion].forma_ejecutar=2"* (o 1) y *"final_morfologia=2",* determina que ejecuta la secuencia mostrada en la Figura 90 y la Figura 91, si la arquitectura de la recta a ejecutar es hexápodo (identificación 2) y si el sistema ha finalizado su acoplamiento.



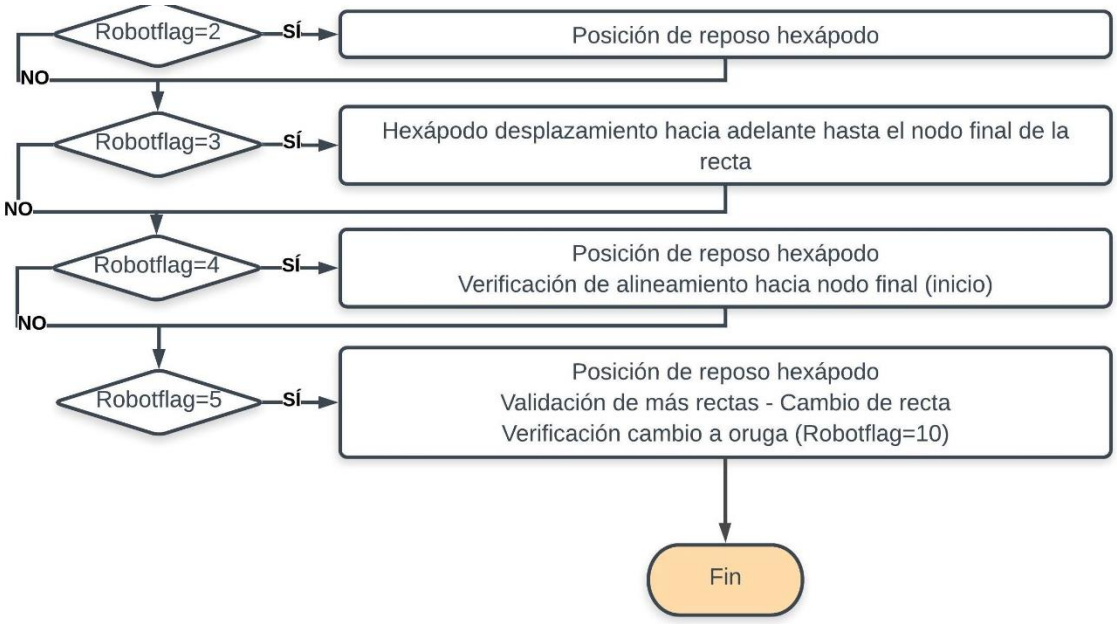


Figura 90. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) para el recorrido de la recta en arquitectura hexápodo, en coordinación con los módulos 2 al 15 (esclavos). Elaboración propia

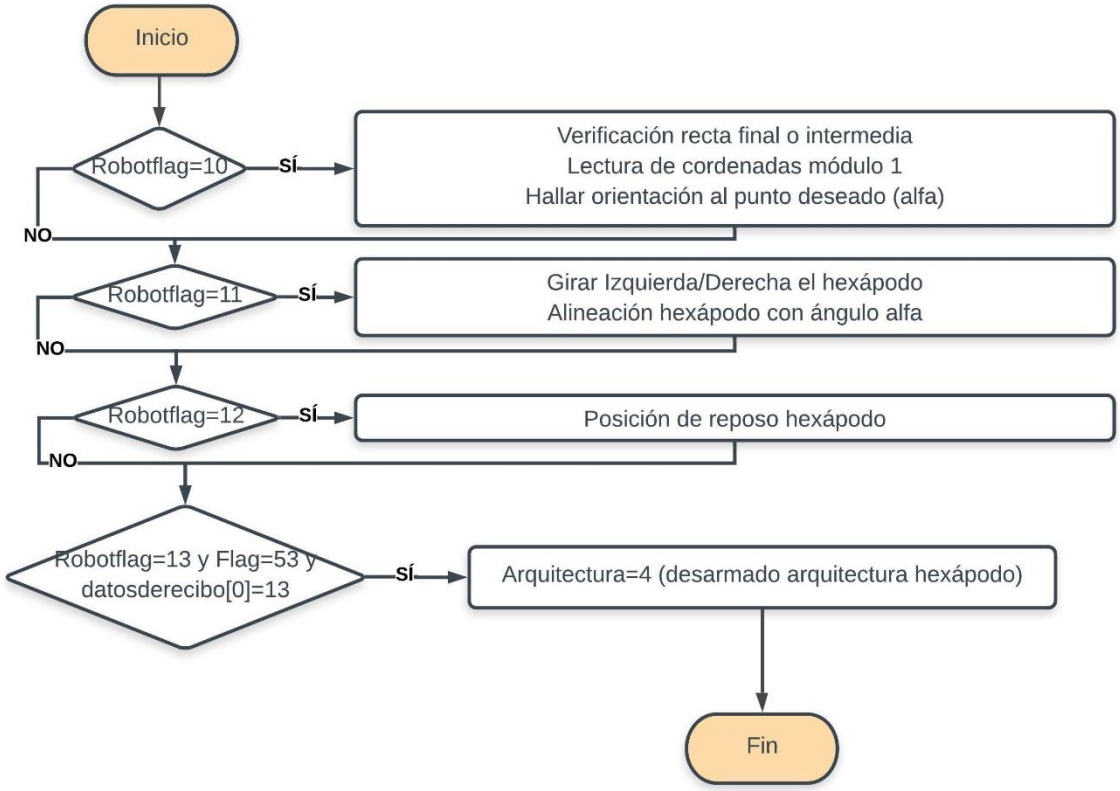


Figura 91. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) para el giro en arquitectura hexápodo, para posteriormente cambiar a arquitectura oruga, en coordinación con los módulos 2 al 15 (esclavos). Elaboración propia

- Cuando está desarmada la arquitectura hexápodo: condición "*final_morfologia=4*", determina que ejecuta la secuencia mostrada en la Figura 92, si el sistema ha culminado el desarmado de la arquitectura hexápodo.

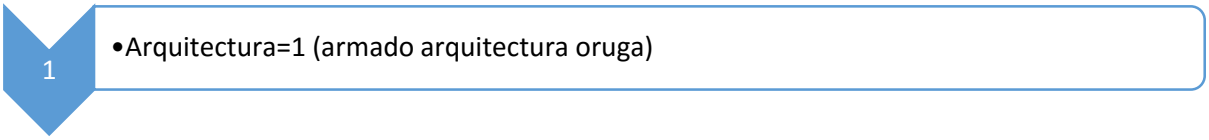
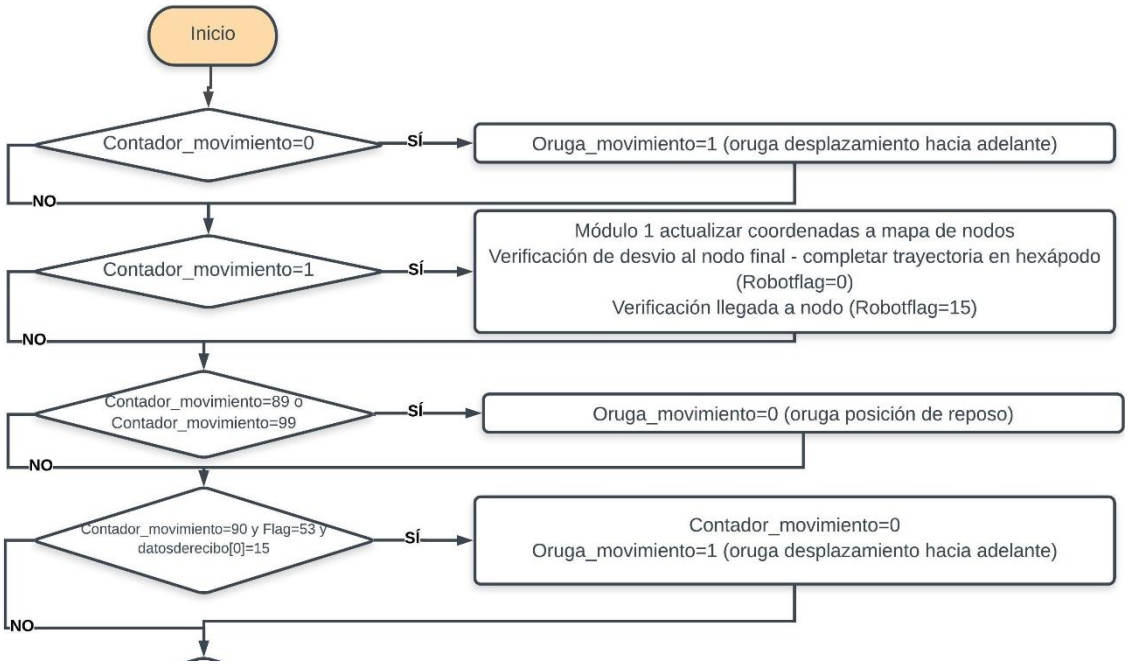


Figura 92. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) para el cambio de arquitectura a oruga, en coordinación con los módulos 2 al 15 (esclavos). Elaboración propia

5.5.2. Cuando la arquitectura armada para recorrer la recta es oruga

Condición *"trayecto_arquitecturas[recta_ejecucion].forma_ejecutar=1"* y *"final_morfologia=1"*, determina que ejecuta la secuencia mostrada en Figura 93, si la arquitectura de la recta a ejecutar es oruga (identificación 1) y si el sistema ha finalizado su acoplamiento.



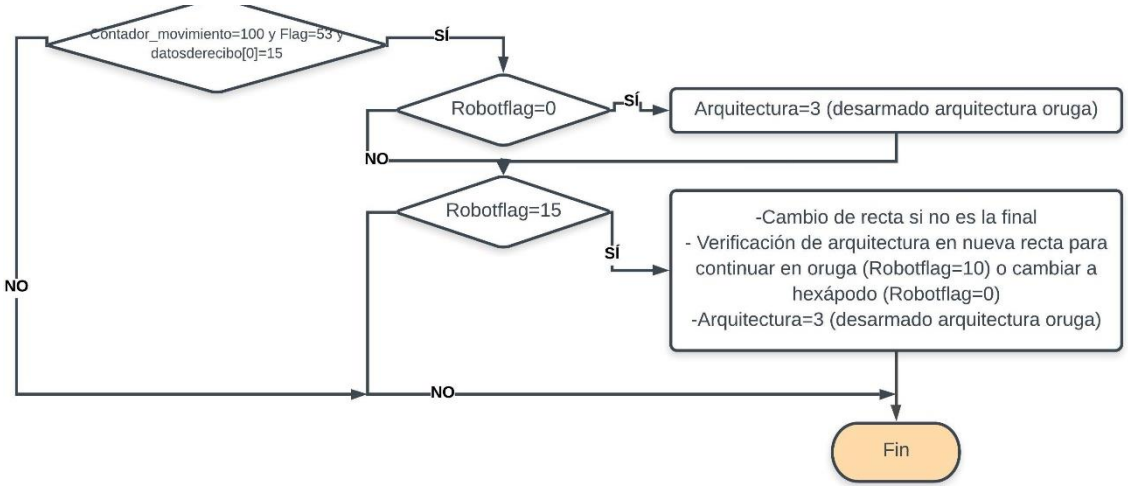


Figura 93. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) para el recorrido de la recta en arquitectura oruga, en coordinación con los módulos 2 al 15 (esclavos). Elaboración propia

- Cuando está desarmada la arquitectura oruga: condición "*final_morfologia=3*", determina que ejecuta la secuencia mostrada en Figura 94, si el sistema ha culminado el desarmado de la arquitectura oruga.

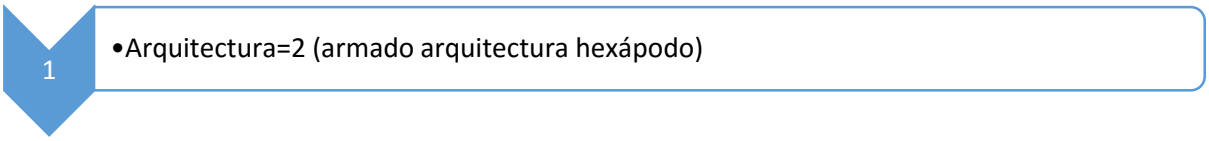


Figura 94. Estructura secuencial para ejecutar la coordinación del módulo 1 (maestro) para el cambio de arquitectura a hexápodo, en coordinación con los módulos 2 al 15 (esclavos). Elaboración propia

5.5.3. Ejecución de arquitecturas y movimientos disponibles para cada morfología

Conjuntamente a la autoconfiguración de arquitecturas para el recorrido de la trayectoria, se generó la función "*ejecutar_morfologia*", mostrada en el Código 54, la cual efectúa el llamado de las funciones[16] para el armado y desarmado de las arquitecturas, como también el desplazamiento de las mismas una vez éstas se encuentran acopladas, conforme la ejecución de los estados presentados en la función "*ejecutar_trayectoria*" (secuencia mostrada desde la Figura 90 hasta la Figura 94 - **Anexo 16**). Cada uno de los movimientos coordinados para el movimiento de las arquitecturas, dependen del cumplimiento condicional de las variables "*arquitectura*", que determina la arquitectura a ejecutar y "*final_morfologia*", que permite conocer el estado de armado del sistema en conjunto; como también las

[16] Previamente explicadas en el capítulo cuarto del presente documento, página 110.

variables “*oruga_movimiento*” y “*hexapodo_movimiento*” que comandan la acción a efectuar para el movimiento coordinado de todos los módulos y el desplazamiento integrado de las arquitecturas.

```
void ejecutar_morfologia(){
  if(arquitectura==1 && final_morfologia!=1){arquitectura_oruga();}
  else if(arquitectura==1 && final_morfologia==1 && oruga_movimiento==0){oruga_abajo();}
  else if(arquitectura==1 && final_morfologia==1 && oruga_movimiento==1){oruga_defrente();}
  if(arquitectura==2 && final_morfologia!=2){arquitectura_hexapodo();}
  else if(arquitectura==2 && final_morfologia==2 && hexapodo_movimiento==0){hexapodo_abajo();}
  else if(arquitectura==2 && final_morfologia==2 && hexapodo_movimiento==1){hexapodo_defrente();}
  else if(arquitectura==2 && final_morfologia==2 && hexapodo_movimiento==2){hexapodo_izquierda();}
  else if(arquitectura==2 && final_morfologia==2 && hexapodo_movimiento==3){hexapodo_derecha();}
  else if(arquitectura==2 && final_morfologia==2 && hexapodo_movimiento==4){hexapodo_reposo();}
  if(arquitectura==3 && final_morfologia!=3){desacople_oruga();}
  if(arquitectura==4 && final_morfologia!=4){desacople_hexapodo();}
}
```

Código 54. Secuencia de programación para coordinar desde el módulo 1 el llamado de las funciones que accionan el armado de las arquitecturas y los posibles desplazamientos a ejecutar- Software WEBOTS. Elaboración propia

5.6. Interfaz usuario – máquina para ejecución de trayectoria con las arquitecturas escogidas

Culminado el diseño y desarrollo del algoritmo de autoconfiguración de las arquitecturas oruga y hexápodo, para recorrer las rectas de la trayectoria obtenida, mediante la metodología A*, se procedió a generar las líneas de código para la interacción con el usuario, en la interfaz de la simulación del programa en el Software WEBOTS, con el fin de brindarle al usuario la capacidad de elegir el punto final deseado, como el ejemplo de la Figura 95, mediante el uso del teclado del computador para ingresar el número de la fila y columna del nodo objetivo, para ejecutar el algoritmo de búsqueda; con lo cual se prosiguió a generar las funciones “*leer_teclado*” y “*menu_programa*”, mostradas en detalle en el **Anexo 17**.

```
[PRUEBA1] PARA EMPEZAR PRESIONE TECLA (e)
[PRUEBA1] Empezar
[PRUEBA1] BIENVENIDO AL PROGRAMA
[PRUEBA1] Ingrese el primer digito de la fila objetivo (2-38) luego la tecla enter
[PRUEBA1] 3
[PRUEBA1] Valor ingresado: 3
[PRUEBA1] Ingrese el segundo digito de la fila objetivo (2-38) luego la tecla enter
[PRUEBA1] 5
[PRUEBA1] Valor ingresado: 5
[PRUEBA1] La fila objetivo es: 35
[PRUEBA1] Ingrese el primer digito de la columna objetivo (2-38) luego la tecla enter
[PRUEBA1] 2
[PRUEBA1] Valor ingresado: 2
[PRUEBA1] Ingrese el segundo digito de la columna objetivo (2-38) luego la tecla enter
[PRUEBA1] 0
[PRUEBA1] Valor ingresado: 0
[PRUEBA1] La columna objetivo es: 20
```

Figura 95. Ejemplo de ejecución del menú elaborado para la interacción con el usuario, mediante el uso del teclado del computador- Software WEBOTS. Elaboración propia

5.7. Integración en función principal del módulo 1 los Algoritmos de Autoconfiguración de Arquitecturas para el Recorrido de la Trayectoria obtenida con la interfaz del menú de usuario

Generadas todas las funciones para la elaboración del proyecto planteado, se procedió a integrarlas en la función principal del módulo maestro 1, como se muestra en el Código 55, la cual inicia con la inicialización de las variables y dispositivos asociados al módulo 1, para cíclicamente (función "*while*") continuar con la lectura del teclado en el menú del programa, que permita dar inicio a la ejecución del algoritmo A*, para coordinar con los demás módulos la autoconfiguración de las arquitecturas oruga y hexápodo, en virtud de recorrer las rectas de la trayectoria obtenida para llegar al punto final ingresado por el usuario.

```
int main()
{
  inicializacion_variables_dispositivos();
  printf("PARA EMPEZAR PRESIONE TECLA (e)\n");
  while (wb_robot_step(TIME_STEP) != -1) {
    leer_teclado();
    menu_programa();
    if(bandera==11){
      algoritmo_a_star();//funcion para hallar la trayectoria ejecutando el algoritmo A*
    }
    if(bandera==13){
      //Se verifica si se recibe informacion de algun modulo
      recepcion_informacion_modulo();
      //Para realizar el movimiento del modulo individual para llevarlo a la posicion y
      if(flag<=6){desplazamiento_modulo();}
      //para ejecutar la trayectoria planificada utilizando las dos arquitecturas
      ejecutar_trayectoria();
      //para ejecutar alguna arquitectura y sus movimientos
      ejecutar_morfologia();
    }
  };
  wb_robot_cleanup();
  return 0;
}
```

Código 55. Secuencia de programación en la función principal del módulo 1 para la autoconfiguración de arquitecturas para recorrer la trayectoria obtenida hasta llegar al punto deseado - Software WEBOTS. Elaboración propia

5.8. Conclusiones del Capítulo

En este capítulo se presentó el diseño y desarrollo del algoritmo de búsqueda y planificación de trayectorias, considerando las restricciones del escenario y las limitaciones puestas en el terreno como obstáculos, para la obtención de la mejor ruta de desplazamiento para llegar al punto objetivo, con el fin de definir la arquitectura más apropiada con la se va ejecutar cada tramo o recta de la trayectoria obtenida.

Se tuvo en cuenta las condiciones de espacio para la elección de arquitecturas, bien fuera por el evento en que el tramo es una recta de más de 7 nodos de recorrido con el espacio suficiente detrás del módulo 1 para el armado de la arquitectura oruga, incluso en caso de ser necesario, generándolo nodos adelante en rectas con suficiente espacio para su ejecución, generando dos tramos de recta en una sola para ejecutar ambas arquitecturas, como también atender el evento en el que se

presenten rectas cortas o se necesite girar el sistema para dar inicio al recorrido de la siguiente recta mediante la arquitectura hexápodo.

Además, se presentó la unificación de los algoritmos para el armado, desarmado y desplazamiento de las arquitecturas oruga y hexápodo, junto con el de búsqueda y planificación trayectorias, a través del método A*, para lograr que el sistema robótico pueda autoconfigurarse de manera eficaz y eficiente para recorrer la trayectoria que permita llegar al nodo objetivo.

Para la visualización integral de resultados, se desarrolló el código que posibilita la interacción del usuario, a través de un menú en la interfaz de simulación del programa en el Software WEBOTS, para elegir el punto final deseado, funciones que fueron integradas recursivamente en la función principal del módulo maestro 1 para lograr la ejecución efectiva del presente proyecto.

CAPÍTULO VI.

PRUEBAS Y RESULTADOS

6.1. Pruebas de funcionamiento del algoritmo desarrollado

6.1.1. Adaptación de los dispositivos asociados

A continuación se muestra el registro de las pruebas obtenidas para la verificación de funcionamiento de cada uno de los elementos asociados y los dispositivos integrados a cada uno de los módulos que conforman el sistema, previamente explicados en el presente documento.

6.1.1.1. Funcionamiento dispositivo GPS

En la Figura 96, se aprecian las coordenadas del GPS mostradas en milímetros, como también la orientación tomada del COMPASS mostrada en grados.

Figura 96. Prueba de funcionamiento y visualización de resultados dispositivo GPS y COMPASS - Software WEBOTS. Elaboración propia

6.1.1.2. Caracterización y funcionamiento dispositivo COMPASS para orientación en grados

En la Figura 97, se muestra la prueba de funcionamiento y caracterización del dispositivo COMPASS, para la obtención de la orientación en grados de cada módulo, acorde a los datos registrados en el **Anexo *4***.

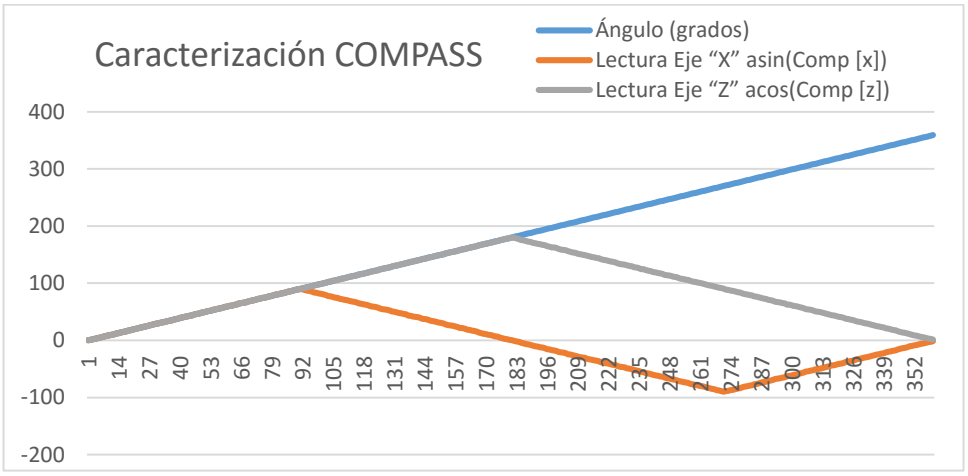


Figura 97. Prueba de funcionamiento dispositivo COMPASS y caracterización para orientación del módulo en grados -. Elaboración propia

6.1.1.3. Funcionamiento elemento CONNECTOR para acople entre módulos

En la Figura 98, se aprecia la prueba de funcionamiento del elemento CONNECTOR, que simula la acción de acople entre módulos, pudiéndose verificar su correcta aplicación, en el armado de la arquitectura hexápodo, al levantarse el módulo 1, con el giro del pivote del módulo 2, sin que el primero se caiga.

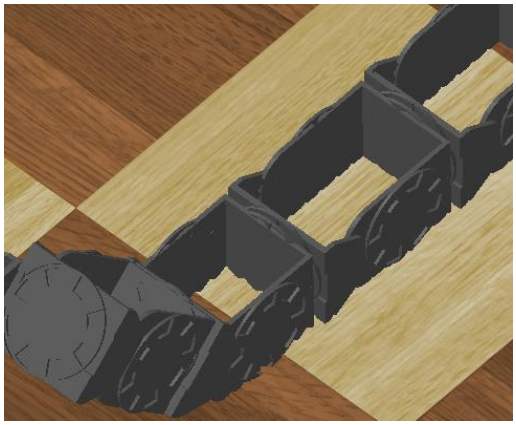


Figura 98. Prueba de funcionamiento del elemento CONNECTOR para acoplamiento entre módulos - Software WEBOTS. Elaboración propia

6.1.1.4. Funcionamiento dispositivos EMITTER – RECEIVER para validación de protocolo de comunicación

En la Figura 99 y Figura 100, se aprecia la validación de funcionamiento de los dispositivos EMITTER y RECEIVER, como también del protocolo de comunicación propuesto, al mostrarse la recepción de datos en el módulo 2 enviados desde el módulo 1 (Figura 99), además del envío de datos del módulo 2 al módulo 1 (Figura 100), una vez realizada la acción encomendada desde el módulo maestro en la anterior acción.

```
[PRUEBA2] Recibido: 2, 3624, 3625, 270, 1, 2
[PRUEBA2] P1X:3483 , P1Z:3766 , Dist:138.852440
[PRUEBA2] P2X:3483 , P2Z:3484 , Dist:338.520310
[PRUEBA2] P3X:3765 , P3Z:3484 , Dist:342.660182
[PRUEBA2] P4X:3765 , P4Z:3766 , Dist:148.660687
[PRUEBA2] P5X:3624 , P5Z:3766 , Dist:28.442925
[PRUEBA2] P8X:3483 , P8Z:3630 , Dist:213.053984
[PRUEBA2] P11X:3765 , P11Z:3627 , Dist:221.822001
[PRUEBA2] El puntomin es:5, Dist:28.442925
```

Figura 99. Prueba de funcionamiento dispositivo RECEIVER, validación protocolo de comunicación para la recepción de datos en el módulo 2 enviados desde el módulo 1 - Software WEBOTS. Elaboración propia

```
[PRUEBA2] listo 10: envio de datos al modulo 1
[PRUEBA1] Recibido: 2, 3624, 3738, 270, 1, 2
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 1, 3624, 3625, 270, 1, 2,
[PRUEBA1] 2, 3624, 3738, 270, 1, 2,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
[PRUEBA1] 0, 0, 0, 0, 0, 0,
```

Figura 100. Prueba de funcionamiento dispositivo EMITTER validación protocolo de comunicación para el envío del módulo 2 y la recepción de datos en el módulo 1 - Software WEBOTS. Elaboración propia

6.1.1.5. Verificación puntos de bordeo y acercamiento a recorrer el módulo de acople para llegar a la rueda deseada del módulo a acoplar

En continuación con el ejemplo mostrado en la Figura 99, de la acción encomendada al módulo 2 desde el módulo maestro para el acoplamiento al módulo 3, acorde a las coordenadas mostradas en la Tabla 15, se muestra entonces en la Tabla 16, la validación de las coordenadas de los puntos de bordeo igualmente obtenidas y mostradas en la Figura 99, junto con las coordenadas de acercamiento para cada una de las ruedas deseadas al módulo a acoplar.

Tabla 15. Ejemplo de coordenadas del módulo a acoplar como referencia para el módulo de acople

Coordenada X	Coordenada Z	Orientación
3624	3625	270

Las coordenadas de los puntos obtenidos en la Tabla 16, tanto de bordeo (en color "azul"), como los de acercamiento (colores "verde", "naranja" y "amarillo") se grafican y se señalan en la Figura 101, con la misma convención de colores mencionados, siendo los puntos de bordeo indicados con color "azul", junto con los de acercamiento a cada rueda, posterior de color "verde", izquierda de color "naranja" y derecha de color "amarillo", destacando que el eje positivo "x" del mapa en el escenario del Software WEBOTS se encuentra invertido al que en la Figura 101 se indica, para efectos de ubicación de las ruedas izquierda y derecha.

Tabla 16. Prueba de coordenadas obtenidas de los puntos de bordeo y acercamiento a recorrer el módulo de acople para llegar a la rueda deseada del módulo a acoplar

Bordeo / Acercamiento	Rueda	Número de Punto	Coordenadas en X (mm)	Coordenadas en Z (mm)
Bordeo	Ninguna	1	3483	3766
Bordeo	Ninguna	2	3483	3484
Bordeo	Ninguna	3	3765	3484
Bordeo	Ninguna	4	3765	3766
Bordeo	Posterior	5	3624	3766
Acercamiento	Posterior	6	3624	3756
Acercamiento	Posterior	7	3624	3746
Bordeo	Izquierda	8	3483	3630
Acercamiento	Izquierda	9	3493	3630
Acercamiento	Izquierda	10	3503	3630
Bordeo	Derecha	11	3765	3627
Acercamiento	Derecha	12	3755	3627
Acercamiento	Derecha	13	3745	3627

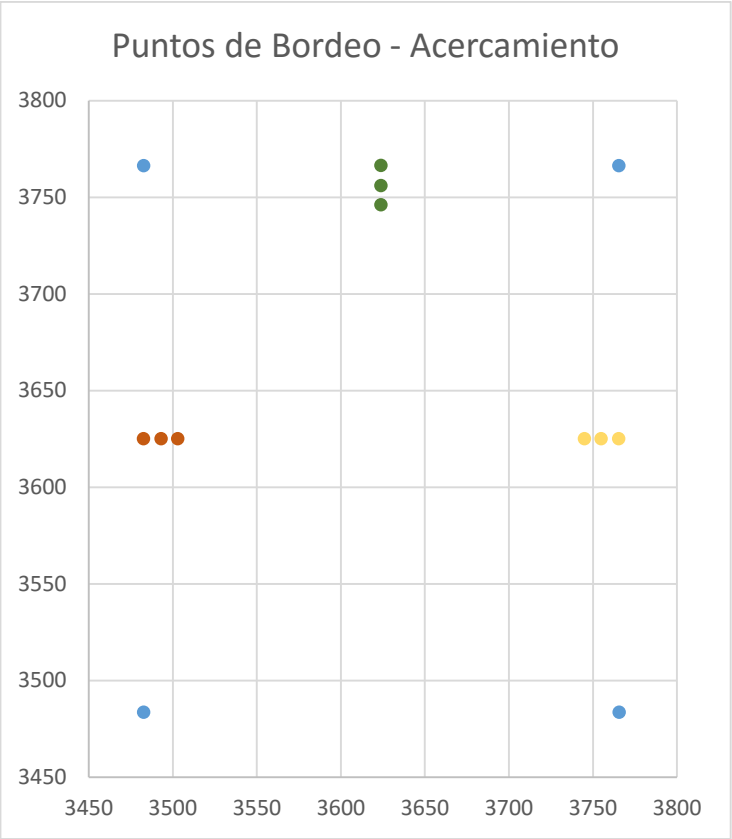


Figura 101. Ubicación de los puntos de bordeo y acercamiento del módulo a acoplar y cada una de sus ruedas obtenidos, a partir del ejemplo de la Tabla 16 -. Elaboración propia

6.1.2. Condiciones iniciales de simulación – ubicación de módulos y obstáculos

Para efectos de la simulación, en el entorno virtual del Software WEBOTS, se fija la posición del sistema robótico modular y los obstáculos a evitar, conforme se muestra en la Figura 102, que corresponden a las siguientes características:

- Posición inicial sistema robótico: fila 6, columna 6.
- Obstáculo 1: ancho 3 nodos cuadrados, fila 11, columna 21.
- Obstáculo 2: ancho 2 nodos cuadrados, fila 31, columna 31.
- Obstáculo 3: ancho 1 nodo cuadrado, fila 21, columna 11.
- Obstáculo 4: ancho 1 nodo cuadrado, fila 18, columna 30.
- Obstáculo 5: ancho 3 nodos cuadrados, fila 26, columna 21.

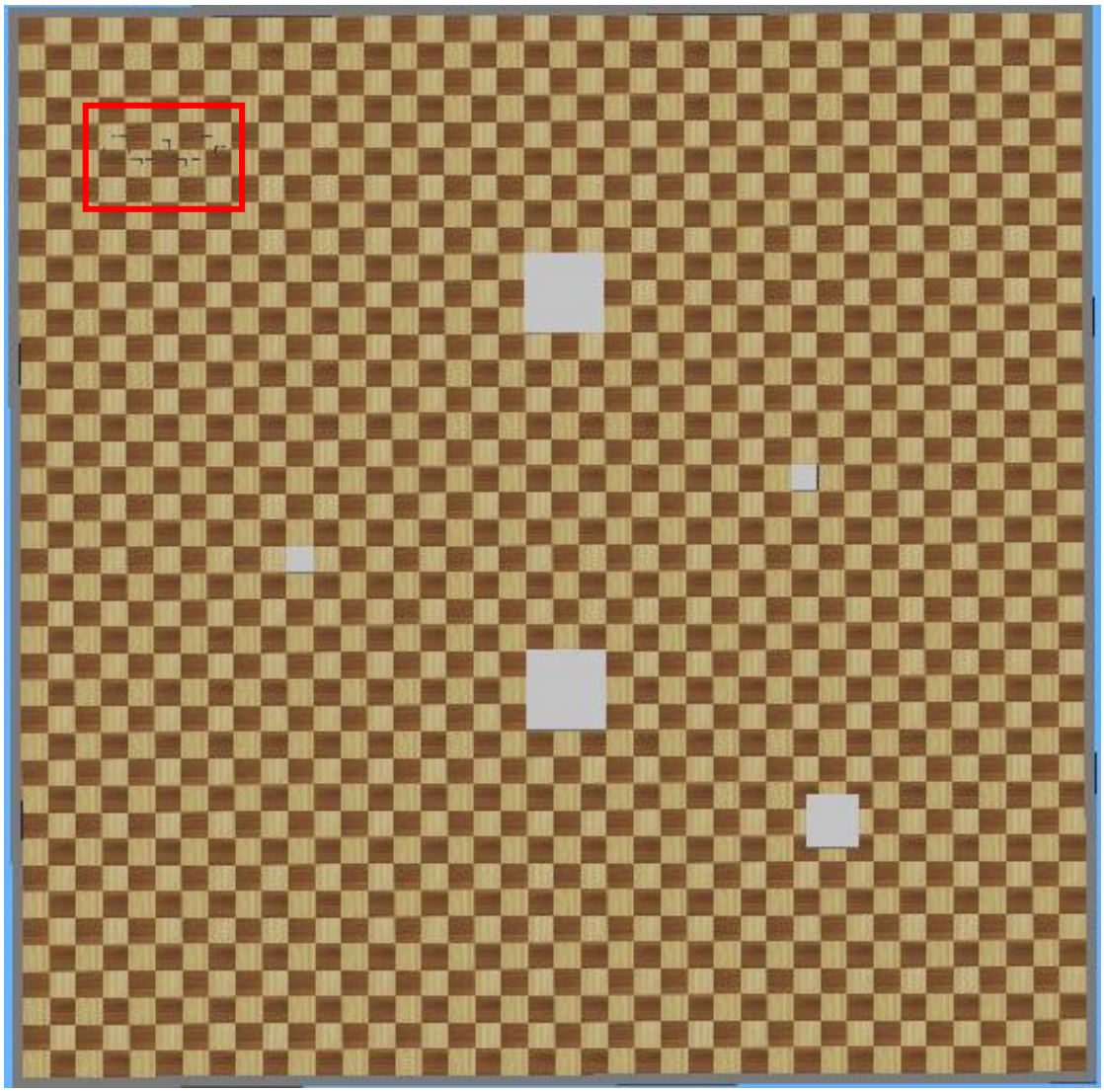


Figura 102. Ubicación de obstáculos y posicionamiento sistema robótico, entorno virtual - Software WEBOTS. Elaboración propia

Por otra parte, se fijó como posición inicial del sistema robótico modular, la fila 6 y columna 6 (recuadro color “rojo” Figura 102), para que de esta manera, conocidas las restricciones del escenario y conforme la ubicación de los módulos como se muestra en la Figura 103, se inicie la simulación con el armado de la arquitectura hexápodo.

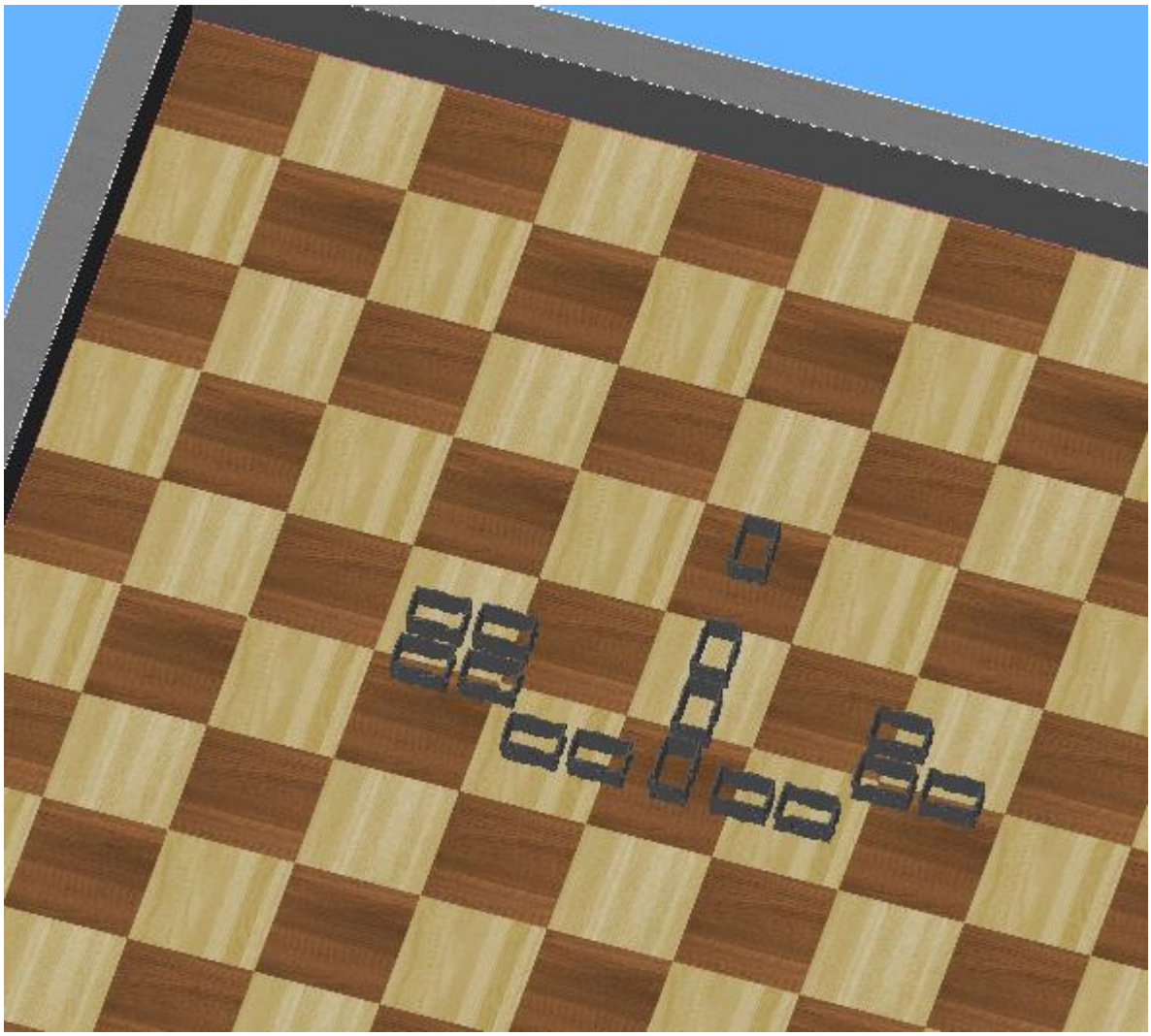


Figura 103. Ubicación de módulos para autoconfiguración arquitectura hexápodo, condición inicial en el entorno virtual - Software WEBOTS. Elaboración propia

6.1.3. Búsqueda y planificación de trayectorias – metodología A* (A-Star) elección de arquitecturas para recorrido de la ruta obtenida

En continuación con el ejemplo mostrado en la Figura 95**¡Error! No se encuentra el rigen de la referencia.**, cuyo nodo objetivo ingresado por el usuario corresponde a la fila 35 y columna 20, junto con las condiciones iniciales fijadas para la ubicación de obstáculos (Figura 102) y el sistema robótico modular (Figura 103), se muestra en la Figura 104, la continuación de ejecución de la misma simulación, los resultados obtenidos del algoritmo de búsqueda A*, en donde se escoge la mejor trayectoria a recorrer el sistema robótico, señalada visualmente con el símbolo "+", de forma que pueda llegar al punto final y pueda evitar los obstáculos planteados (con el símbolo "x").

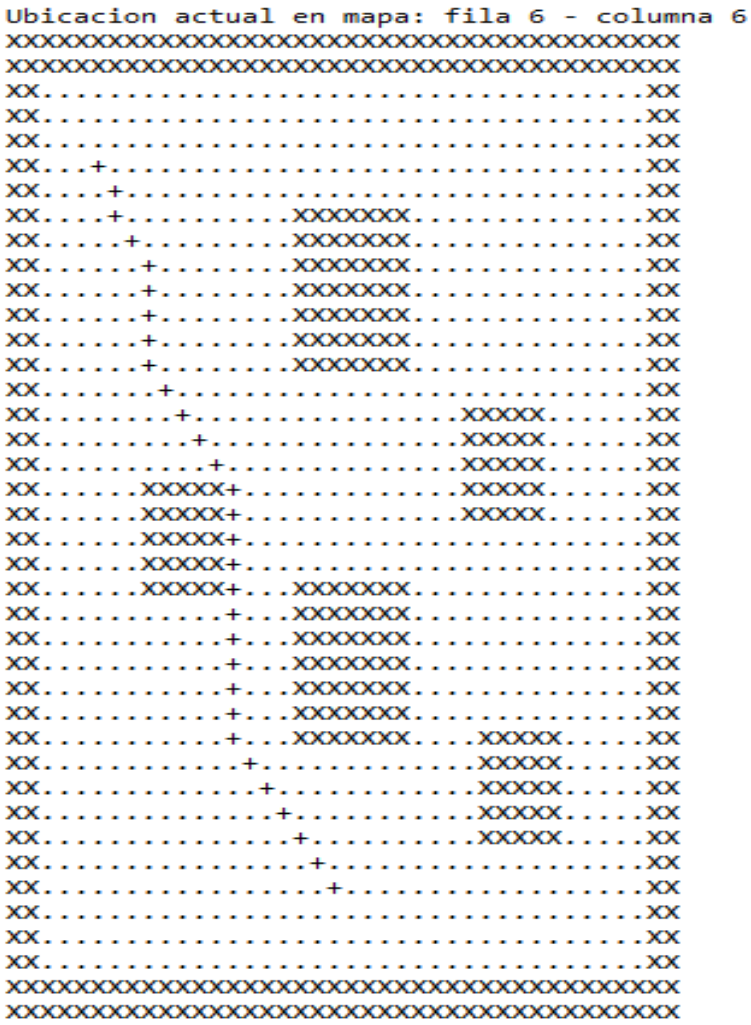


Figura 104. Ejemplo de ejecución del algoritmo A* implementado para la autoconfiguración de arquitecturas oruga y hexápodo - Ventana de simulación en Software WEBOTS. Elaboración propia

Para el recorrido de la trayectoria obtenida (Figura 104), se muestra en la Figura 105, la segmentación de rectas, la cantidad de nodos que las componen, junto con las arquitecturas elegidas, donde el número 1, corresponde para la arquitectura oruga y 2 para la arquitectura hexápodo.

```
RESULTADO ELECCION DE ARQUITECTURA(S) SOBRE LA TRAYECTORIA PLANTEADA:
Recta 1 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:1 - fila/columna: 6, 6
Recta 2 - Pendiente:90, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:2 - fila/columna: 7, 7
Recta 3 - Pendiente:45, Cantidad de Nodos: 2, arquitectura:2, nodo que inicia:3 - fila/columna: 8, 7
Recta 4 - Pendiente:90, Cantidad de Nodos: 4, arquitectura:2, nodo que inicia:5 - fila/columna: 10, 9
Recta 5 - Pendiente:45, Cantidad de Nodos: 5, arquitectura:2, nodo que inicia:9 - fila/columna: 14, 9
Recta 6 - Pendiente:90, Cantidad de Nodos: 10, arquitectura:1, nodo que inicia:14 - fila/columna: 19, 14
Recta 7 - Pendiente:45, Cantidad de Nodos: 5, arquitectura:2, nodo que inicia:24 - fila/columna: 29, 14
Recta 8 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:29 - fila/columna: 34, 19
```

Figura 105. Resultado de las arquitecturas elegidas por el algoritmo elaborado para el recorrido de las rectas que componen la trayectoria obtenida – Ventana de simulación en Software WEBOTS. Elaboración propia

6.1.4. Armado de la arquitectura hexápodo

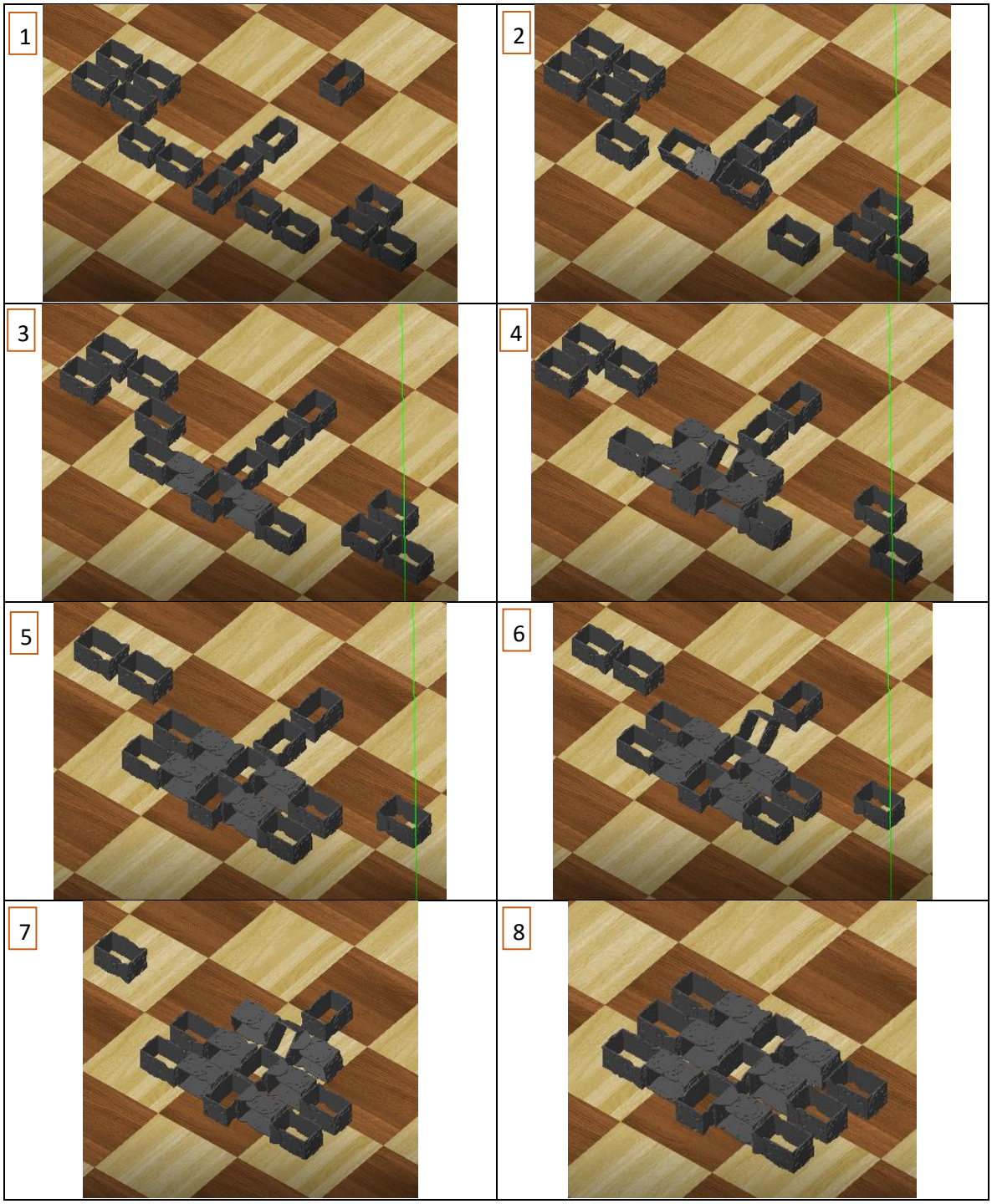


Figura 106. Esquema de armado de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.4.1. Desplazamiento hacia adelante arquitectura hexápodo

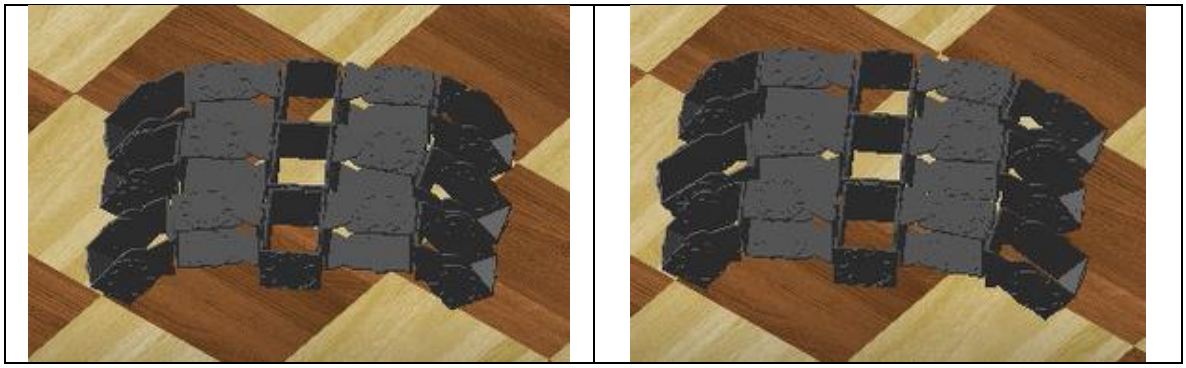


Figura 107. Prueba de desplazamiento hacia delante de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.4.2. Giro a la derecha arquitectura hexápodo

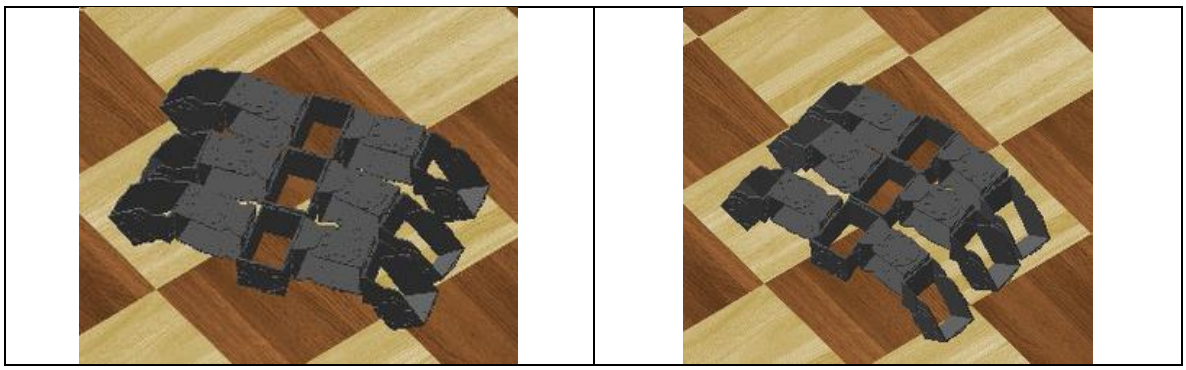


Figura 108. Prueba de giro a la derecha de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.4.3. Giro a la izquierda arquitectura hexápodo

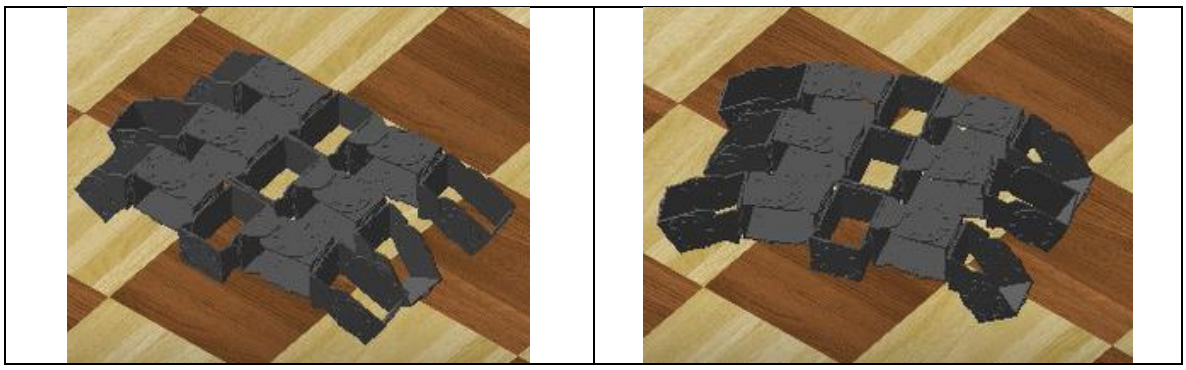


Figura 109. Prueba de giro a la izquierda de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.4.4. Posición de reposo arquitectura hexápodo

Figura 110. Prueba posición de reposo de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.4.5. Posición inicial arquitectura hexápodo

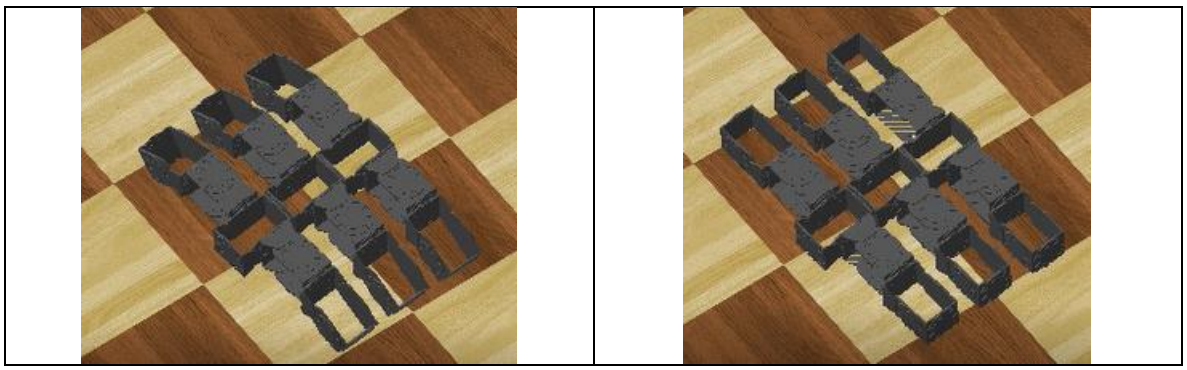
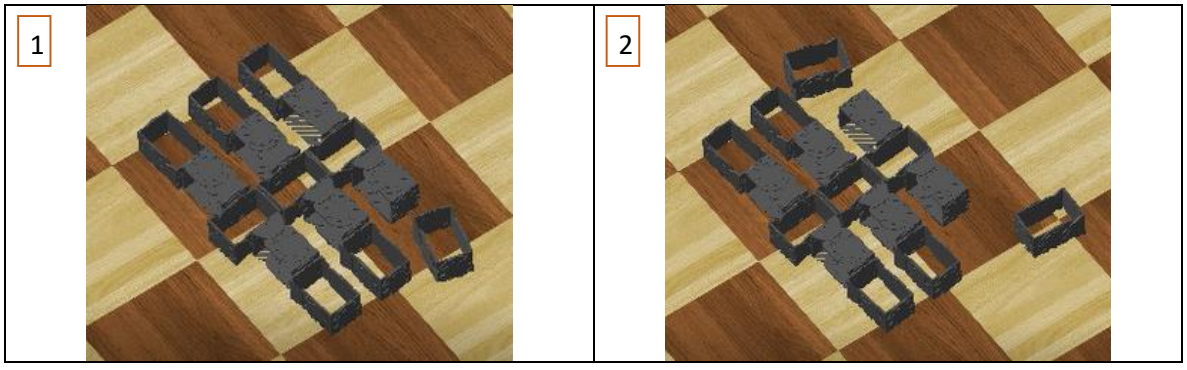


Figura 111. Prueba posición inicial de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.5. Desarmado de la arquitectura hexápodo



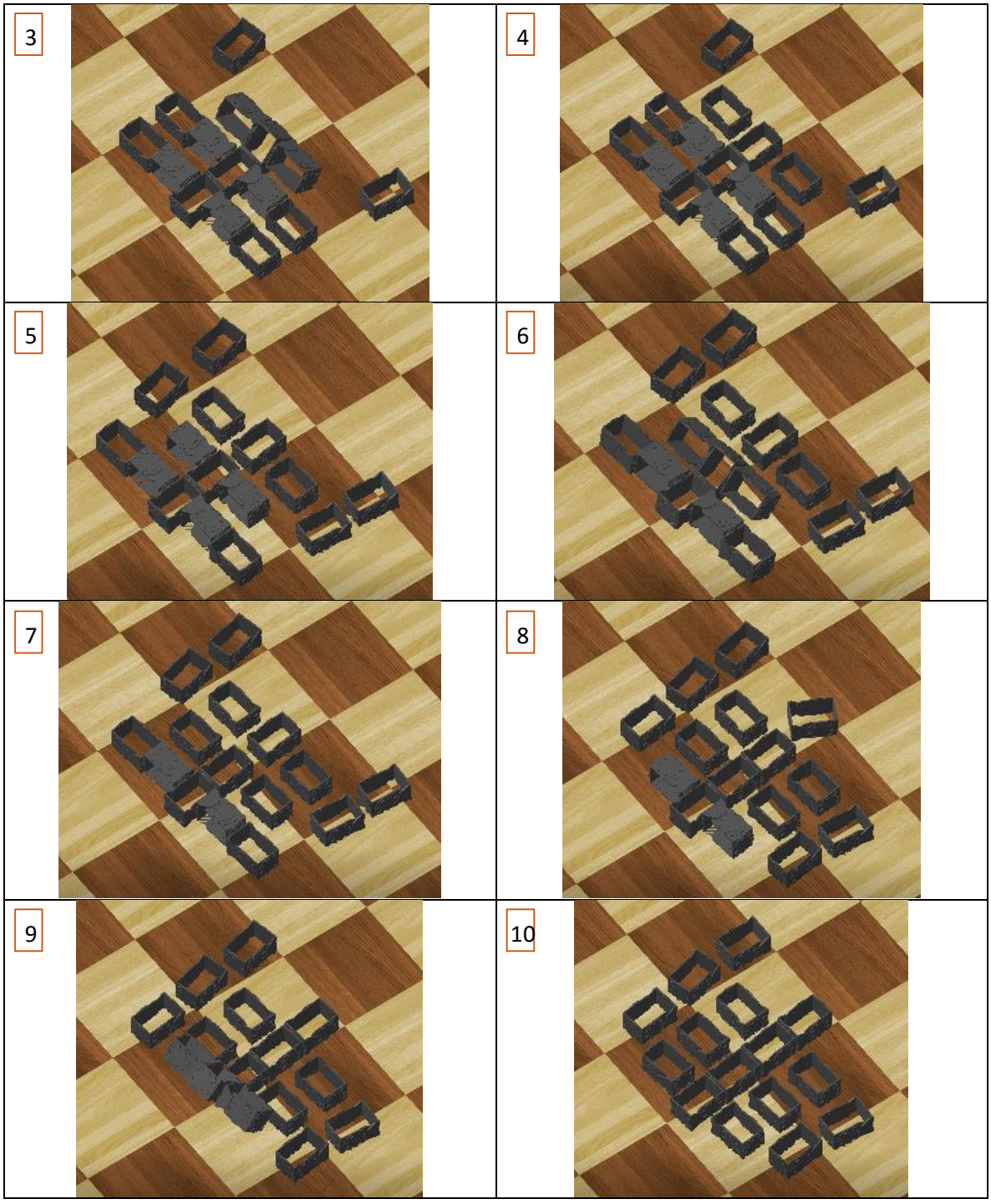


Figura 112. Esquema de desarmado de la arquitectura hexápodo en entorno virtual - Software WEBOTS. Elaboración propia

6.1.6. Armado de la arquitectura oruga

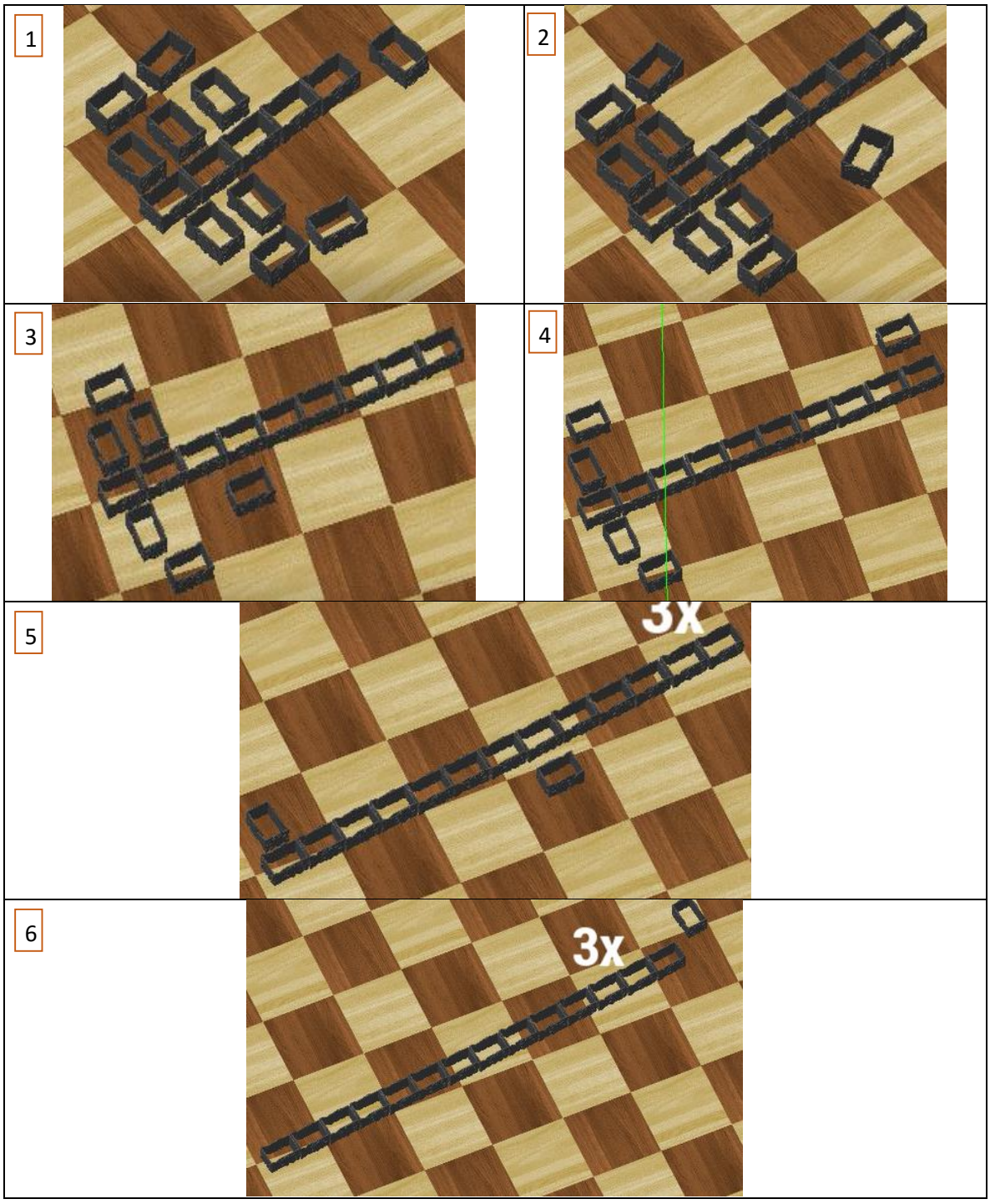


Figura 113. Esquema de armado de la arquitectura oruga en entorno virtual - Software WEBOTS. Elaboración propia

6.1.6.1. Desplazamiento hacia adelante arquitectura oruga

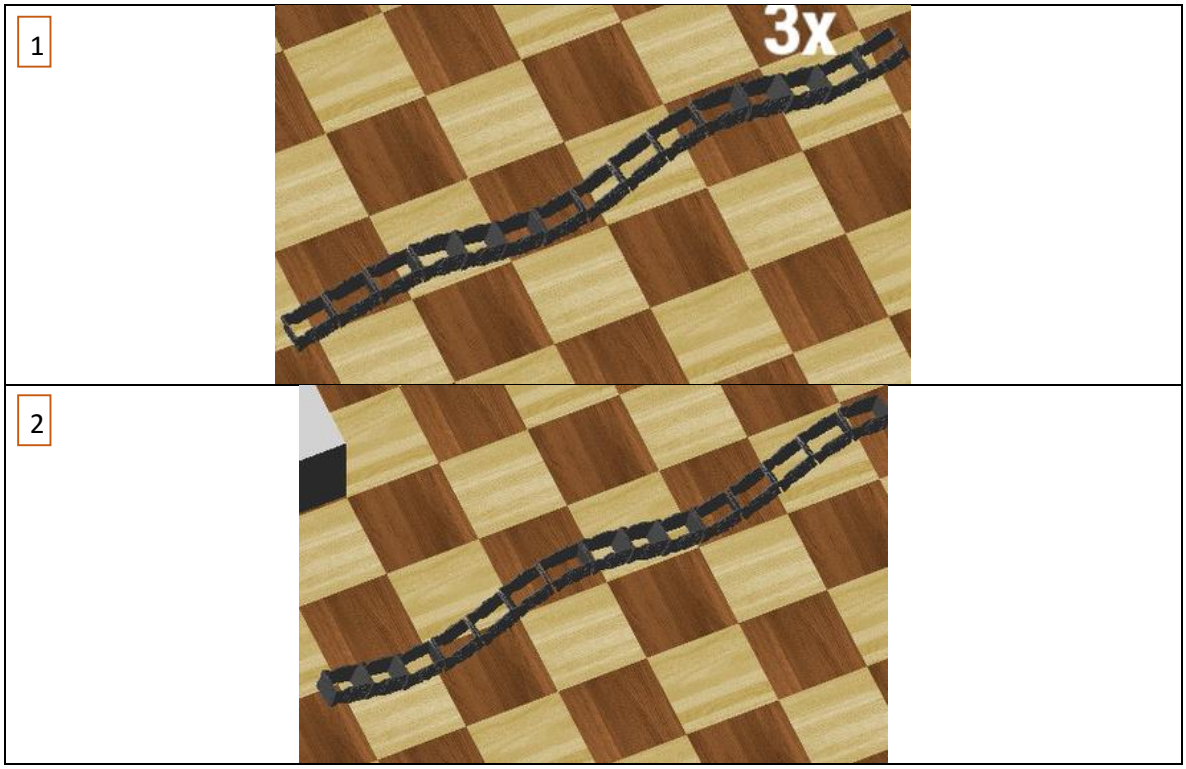
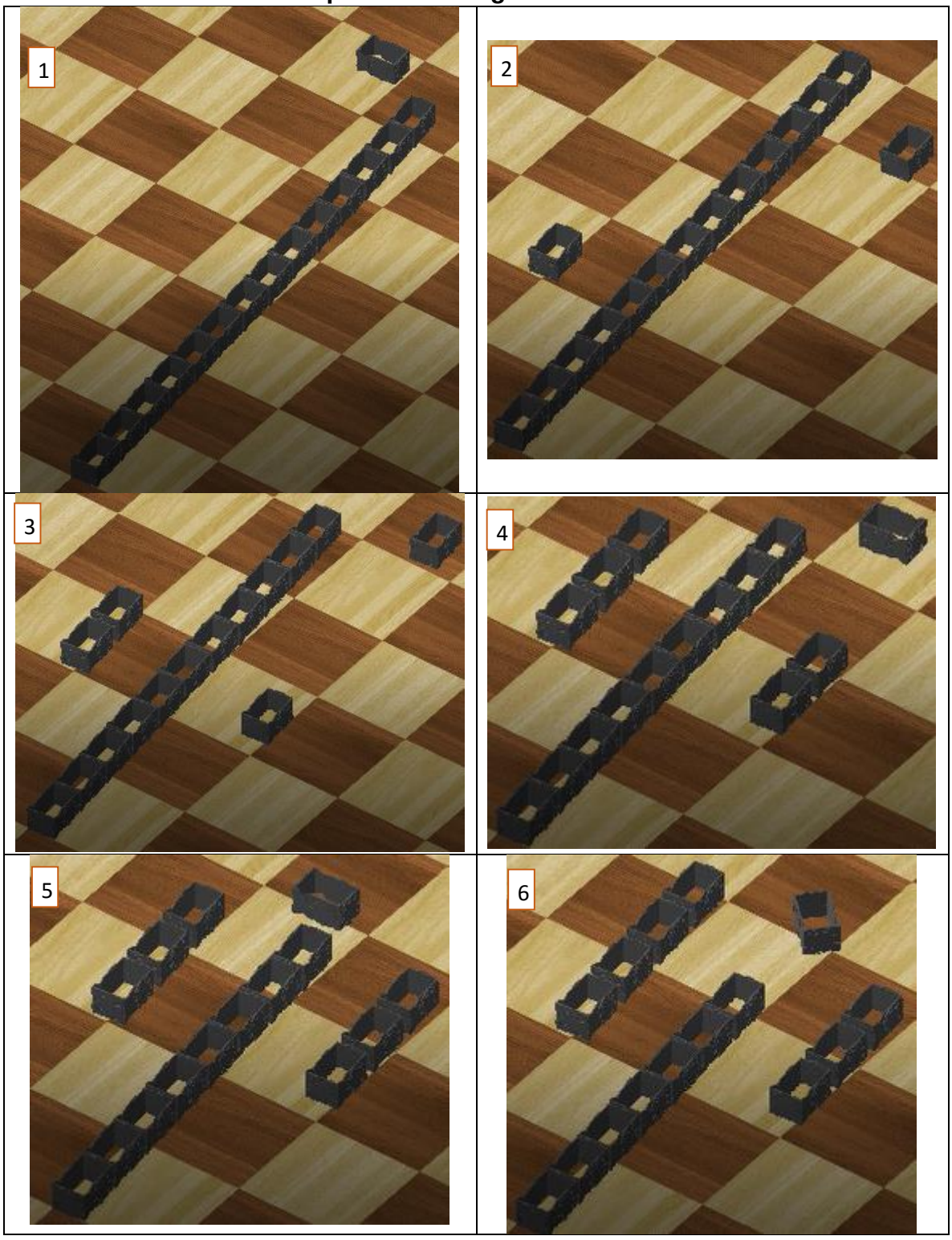


Figura 114. Prueba de desplazamiento hacia delante de la arquitectura oruga en entorno virtual - Software WEBOTS. Elaboración propia

6.1.6.2. Posición de reposo arquitectura oruga

Figura 115. Prueba posición de reposo de la arquitectura oruga en entorno virtual - Software WEBOTS. Elaboración propia

6.1.7. Desarmado de la arquitectura oruga



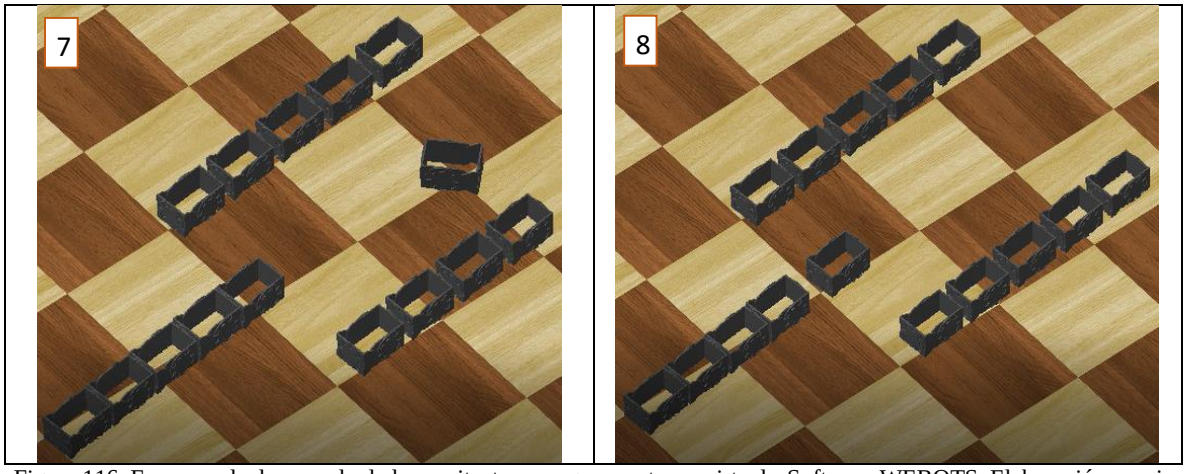


Figura 116. Esquema de desarmado de la arquitectura oruga en entorno virtual - Software WEBOTS. Elaboración propia

6.2. Resultados obtenidos para diferentes puntos finales ingresados por el usuario

Comprobada la efectividad del algoritmo de autoconfiguración de las arquitecturas hexápodo y oruga para recorrer la trayectoria obtenida mediante la metodología A*, se prosigue entonces a mostrar en la Tabla 17, los resultados obtenidos de cinco (5) registros realizados en la simulación variando el punto final de llegada del sistema robótico y con variación de obstáculos, se registra el resumen fotográfico de los resultados obtenidos a partir del **Anexo 18** hasta el **Anexo 22**.

Tabla 17. Resultados obtenidos para diferentes nodos objetivo. Elaboración propia

Resultado	Fila Objetivo	Columna Objetivo	N° nodos trayectoria	Número de rectas	Cambios de arquitectura	Tiempo total Minutos -3X
1	19	7	14	3	2	33:15
2	19	7	14	3	2	34:03
3	15	35	30	4	2	34:44
4	35	20	30	8	2	36:10
5	37	25	32	6	4	47:09

A continuación se muestra el detalle de los resultados obtenidos, en los cuales se registra el tiempo tomado para realizar cada una de las acciones de armado, desplazamiento y desarmado de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de las trayectorias obtenidas, como también la cantidad de nodos recorridos por cada arquitectura en cada tramo o recta, además del consolidado general de estos eventos registrados para cada arquitectura en el recorrido de las trayectorias, a fin de presentar un análisis final a los datos conseguidos, que permitan identificar el rendimiento del sistema robótico en la ejecución del algoritmo desarrollado.

6.2.1. Resultado 1 – Trayectoria de 14 Nodos con 2 Cambios de Arquitectura

Como complemento, en el **Anexo 18**, se encuentra el resumen fotográfico del resultado 1 obtenido.

Tabla 18. Resultado 1 - Detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de la trayectoria obtenida. Elaboración propia

Recta	Arquitectura	Acción	N° nodos recta	Nodo Inicia	Nodo Final	Duración mm:ss
1	Hexápodo	Armado	-	-	-	3:27
1	Hexápodo	Desplazamiento	3	1	4	0:16
1	Hexápodo	Desarmado	-	-	-	1:10
2	Oruga	Armado	-	-	-	11:22
2	Oruga	Desplazamiento	9	4	13	0:09
2	Oruga	Desarmado	-	-	-	6:12
3	Hexápodo	Armado	-	-	-	10:25
3	Hexápodo	Desplazamiento	1	13	14	0:14
Total						33:15

Tabla 19. Resultado 1 - Resumen ejecución de arquitecturas en trayectoria recorrida. Elaboración propia

Arquitectura	Acción	Cuenta de Arquitectura	Suma de N° nodos recta	Suma de Duración Minutos
Hexápodo	Armado	2	0	13:52
	Desarmado	1	0	01:10
	Desplazamiento	2	4	00:30
Total Hexápodo		**5**	**4**	**15:32**
Oruga	Armado	1	0	11:22
	Desarmado	1	0	06:12
	Desplazamiento	1	9	00:09
Total Oruga		**3**	**9**	**17:43**
Total general		**8**	**13**	**33:15**

6.2.2. Resultado 2 – Trayectoria de 14 Nodos con 2 Cambios de Arquitectura

Como complemento, en el **Anexo 19**, se encuentra el resumen fotográfico del resultado 2 obtenido.

Tabla 20. Resultado 2 - Detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de la trayectoria obtenida. Elaboración propia

Recta	Arquitectura	Acción	N° nodos recta	Nodo Inicia	Nodo Final	Duración mm:ss
1	Hexápodo	Armado	-	-	-	03:30
1	Hexápodo	Desplazamiento	3	1	4	00:20
1	Hexápodo	Desarmado	-	-	-	01:09
2	Oruga	Armado	-	-	-	11:22
2	Oruga	Desplazamiento	9	4	13	00:13
2	Oruga	Desarmado	-	-	-	06:47
3	Hexápodo	Armado	-	-	-	10:19
3	Hexápodo	Desplazamiento	1	13	14	00:24
Total						34:03

Tabla 21. Resultado 2 - Resumen ejecución de arquitecturas en trayectoria recorrida. Elaboración propia

Arquitectura	Acción	Cuenta de Arquitectura	Suma de N° nodos recta	Suma de Duración Minutos
Hexápodo	Armado	2	0	13:49
	Desarmado	1	0	01:09
	Desplazamiento	2	4	00:44
Total Hexápodo		**5**	**4**	**15:42**
Oruga	Armado	1	0	11:22
	Desarmado	1	0	06:47
	Desplazamiento	1	9	00:13
Total Oruga		**3**	**9**	**18:22**
Total general		**8**	**13**	**34:04**

6.2.3. Resultado 3 – Trayectoria de 30 Nodos con 2 Cambios de Arquitectura

Como complemento, en el **Anexo 20**, se encuentra el resumen fotográfico del resultado 3 obtenido.

Tabla 22. Resultado 3 - Detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de la trayectoria obtenida. Elaboración propia

Recta	Arquitectura	Acción	N° nodos recta	Nodo Inicia	Nodo Final	Duración mm:ss
1	Hexápodo	Armado	-	-	-	03:27
1	Hexápodo	Desplazamiento	3	1	4	00:20
1	Hexápodo	Desarmado	-	-	-	01:07
2	Oruga	Armado	-	-	-	12:19

2	Oruga	Desplazamiento	17	4	21	00:25
2	Oruga	Desarmado	-	-	-	06:18
3	Hexápodo	Armado	-	-	-	09:07
3	Hexápodo	Desplazamiento	9	21	30	01:42
Total						34:44

Tabla 23. Resultado 3 - Resumen ejecución de arquitecturas en trayectoria recorrida. Elaboración propia

Arquitectura	Acción	Cuenta de Arquitectura	Suma de N° nodos recta	Suma de Duración Minutos
Hexápodo	Armado	2	0	12:34
	Desarmado	1	0	01:07
	Desplazamiento	2	12	02:02
Total Hexápodo		**5**	**12**	**15:43**
Oruga	Armado	1	0	12:19
	Desarmado	1	0	06:18
	Desplazamiento	1	17	00:25
Total Oruga		**3**	**17**	**19:02**
Total general		**8**	**29**	**34:45**

6.2.4. Resultado 4 – Trayectoria de 30 Nodos con 2 Cambios de Arquitectura

Como complemento, en el **Anexo 21**, se encuentra el resumen fotográfico del resultado 4 obtenido.

Tabla 24. Resultado 4 - Detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de la trayectoria obtenida. Elaboración propia

Recta	Arquitectura	Acción	N° nodos recta	Nodo Inicia	Nodo Final	Duración mm:ss
1	Hexápodo	Armado	-	-	-	03:30
1	Hexápodo	Desplazamiento	1	1	2	00:10
2	Hexápodo	Desplazamiento	1	2	3	00:07
3	Hexápodo	Desplazamiento	2	3	5	00:17
4	Hexápodo	Desplazamiento	4	5	9	00:20
5	Hexápodo	Desplazamiento	5	9	14	00:48
5	Hexápodo	Desarmado	-	-	-	01:13
6	Oruga	Armado	-	-	-	12:09
6	Oruga	Desplazamiento	10	14	24	00:10
6	Oruga	Desarmado	-	-	-	06:46
7	Hexápodo	Armado	-	-	-	09:31
7	Hexápodo	Desplazamiento	6	24	30	01:09
Total						36:10

Tabla 25. Resultado 4 - Resumen ejecución de arquitecturas en trayectoria recorrida. Elaboración propia

Arquitectura	Acción	Cuenta de Arquitectura	Suma de N° nodos recta	Suma de Duración Minutos
Hexápodo	Armado	2	0	13:01
	Desarmado	1	0	01:13
	Desplazamiento	6	19	02:51
Total Hexápodo		**9**	**19**	**17:05**
Oruga	Armado	1	0	12:09
	Desarmado	1	0	06:46
	Desplazamiento	1	10	00:10
Total Oruga		**3**	**10**	**19:05**
Total general		**12**	**29**	**36:10**

6.2.5. Resultado 5 – Trayectoria de 32 Nodos con 4 Cambios de Arquitectura

Como complemento, en el **Anexo 22**, se encuentra el resumen fotográfico del resultado 5 obtenido.

Tabla 26. Resultado 5 - Detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de la trayectoria obtenida. Elaboración propia

Recta	Arquitectura	Acción	N° nodos recta	Nodo Inicia	Nodo Final	Duración mm:ss
1	Hexápodo	Armado	-	-	-	03:25
1	Hexápodo	Desplazamiento	1	1	2	00:10
2	Hexápodo	Desplazamiento	1	2	3	00:07
3	Hexápodo	Desplazamiento	1	3	4	00:11
3	Hexápodo	Desarmado	-	-	-	01:11
4	Oruga	Armado	-	-	-	11:37
4	Oruga	Desplazamiento	6	4	10	00:08
4	Oruga	Desarmado	-	-	-	06:38
5	Hexápodo	Armado	-	-	-	09:54
5	Hexápodo	Desplazamiento	5	10	15	00:24
6	Hexápodo	Desplazamiento	1	15	16	00:10
6	Hexápodo	Desarmado	-	-	-	01:28
7	Oruga	Armado	-	-	-	11:16
7	Oruga	Desplazamiento	16	16	32	00:32
Total						47:09

Tabla 27. Resultado 5 - Resumen ejecución de arquitecturas en trayectoria recorrida. Elaboración propia

Arquitectura	Acción	Cuenta de Arquitectura	Suma de N° nodos recta	Suma de Duración Minutos
Hexápodo	Armado	2	0	13:19
	Desarmado	2	0	02:39
	Desplazamiento	5	9	01:02
Total Hexápodo		**9**	**9**	**17:00**
Oruga	Armado	2	0	22:53
	Desarmado	1	0	06:38
	Desplazamiento	2	22	00:40
Total Oruga		**5**	**22**	**30:11**
Total general		**14**	**31**	**47:11**

6.2.6. Análisis a los resultados obtenidos

Con base a los resultados arrojados en los cinco (5) registros realizados, se muestra en la Tabla 28, el consolidado de datos acerca del detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de las trayectorias planificadas por el algoritmo desarrollado, con el fin de obtener el tiempo promedio (columna F) utilizado en cada acción de las dos arquitecturas, resultante de dividir la duración de cada acción (armado, desarmado y desplazamiento) por la cantidad de veces que cada acción fue incurrida por cada arquitectura en ejecución de los recorridos de cada tramo de las trayectorias.

Tabla 28. Resultados totales - Detalle de las arquitecturas ejecutadas para el recorrido de cada una de las rectas de la trayectorias obtenidas. Elaboración propia

A	B	C	D	E	F=E/C	G=F/D
Arquitectura	Acción	Cuenta de Arquitectura	Suma de N° nodos recta	Suma de Duración Minutos	Promedio	Tiempo Prom. Nodo
Hexápodo	Armado	2	0	13:52	06:56	0,00
Hexápodo	Desarmado	1	0	01:10	01:10	0,00
Hexápodo	Desplazamiento	2	4	00:30	00:15	3,75
Oruga	Armado	1	0	11:22	11:22	0,00
Oruga	Desarmado	1	0	06:12	06:12	0,00
Oruga	Desplazamiento	1	9	00:09	00:09	1,00
Hexápodo	Armado	2	0	13:49	06:54	0,00
Hexápodo	Desarmado	1	0	01:09	01:09	0,00
Hexápodo	Desplazamiento	2	4	00:44	00:22	5,50
Oruga	Armado	1	0	11:22	11:22	0,00

Oruga	Desarmado	1	0	06:47	06:47	0,00
Oruga	Desplazamiento	1	9	00:13	00:13	1,44
Hexápodo	Armado	2	0	12:34	06:17	0,00
Hexápodo	Desarmado	1	0	01:07	01:07	0,00
Hexápodo	Desplazamiento	2	12	02:02	01:01	5,08
Oruga	Armado	1	0	12:19	12:19	0,00
Oruga	Desarmado	1	0	06:18	06:18	0,00
Oruga	Desplazamiento	1	17	00:25	00:25	1,47
Hexápodo	Armado	2	0	13:01	06:30	0,00
Hexápodo	Desarmado	1	0	01:13	01:13	0,00
Hexápodo	Desplazamiento	6	19	02:51	00:29	1,53
Oruga	Armado	1	0	12:09	12:09	0,00
Oruga	Desarmado	1	0	06:46	06:46	0,00
Oruga	Desplazamiento	1	10	00:10	00:10	1,00
Hexápodo	Armado	2	0	13:19	06:39	0,00
Hexápodo	Desarmado	2	0	02:39	01:20	0,00
Hexápodo	Desplazamiento	5	9	01:02	00:12	1,33
Oruga	Armado	2	0	22:53	11:27	0,00
Oruga	Desarmado	1	0	06:38	06:38	0,00
Oruga	Desplazamiento	2	22	00:40	00:20	0,91

En la Tabla 29, se muestra como resultado, el tiempo promedio dispuesto para la ejecución de cada acción, las arquitecturas hexápodo y oruga, considerando los resultados totales de las trayectorias recorridas, datos que se encuentran graficados en la Figura 117, de los cuales se puede deducir que el tiempo para el armado y desarmado de la arquitectura oruga es considerablemente mayor al tiempo de las mismas acciones en la arquitectura hexápodo; por otra parte, el tiempo del desplazamiento de la arquitectura oruga es menor que el de la arquitectura hexápodo, factor importante considerando que fue utilizado para el desplazamiento lineal en rectas de más de 7 nodos, mientras que para el hexápodo fue usado para desplazamientos de rectas cortas y giros sobre su eje para conseguir la orientación *deseada.*

Tabla 29. Resultados totales – Tiempo promedio dispuesto para la ejecución de cada acción por las dos arquitecturas en las trayectorias recorridas. Elaboración propia

Arquitectura	Armado	Desarmado	Desplazamiento	Total general
Hexápodo	06:39	01:12	00:28	08:19
Oruga	11:44	06:32	00:15	18:31
Total general	**18:23**	**07:44**	**00:43**	**26:50**

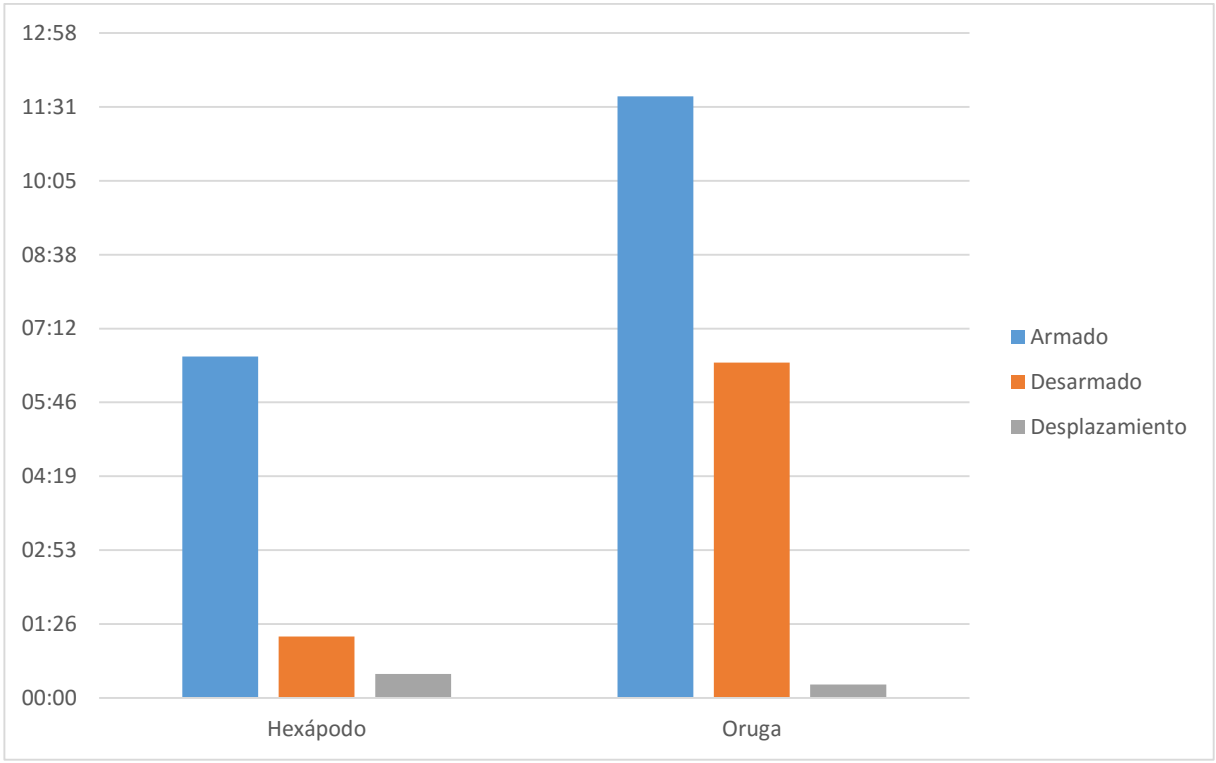


Figura 117. Tiempo promedio dispuesto para la ejecución de cada arquitectura en las trayectorias recorridas. Elaboración propia

En la Tabla 30, se muestra como resultado final, el tiempo promedio de desplazamiento de un (1) nodo para las arquitecturas hexápodo y oruga, calculado a partir del dato obtenido en la columna G de la Tabla 28, que es la división entre el tiempo promedio en la acción de desplazamiento por la cantidad de nodos recorridos, junto con la media final para cada arquitectura, con lo cual se deduce que la arquitectura oruga se desplaza en promedio tres (3) veces más rápido que la arquitectura hexápodo.

Tabla 30. Tiempo promedio de desplazamiento de un (1) nodo en cada arquitectura. Elaboración propia

Arquitectura	Media de Tiempo Promedio Nodo
Hexápodo	3,438596491
Oruga	1,164824718

En la Tabla 31, finalmente se muestra la cantidad de veces ejecutadas las acciones de armado, desarmado y desplazamiento, para ambas arquitecturas, datos que se encuentran graficados en la Figura 118, los cuales muestran la eficiencia y notable rendimiento del sistema, como también del algoritmo implementado, puesto que la arquitectura hexápodo se utiliza en total dos (2) veces más que la arquitectura oruga y casi tres (3) veces más para el desplazamiento de nodos, notando que se optimiza

el uso de los recursos disponibles entre las dos arquitecturas para el recorrido de las trayectorias, al recorrer la arquitectura hexápodo líneas cortas y rotaciones deseadas sobre su eje para conseguir la orientación deseada del sistema (situación más frecuente), dada la velocidad lenta del sistema y la cadencia rápida para su armado y desarmado; en contraste con la arquitectura oruga, la cual recorre líneas con rectas de más de 7 nodos (situación menos frecuente), dada la rapidez del sistema y la cadencia lenta para su armado y desarmado.

Tabla 31. Cantidad de veces ejecutada cada acción para ambas arquitecturas en las trayectorias recorridas. Elaboración propia

Etiquetas de fila	Armado	Desarmado	Desplazamiento	Total general
Hexápodo	10	6	17	33
Oruga	6	5	6	17
Total general	**16**	**11**	**23**	**50**

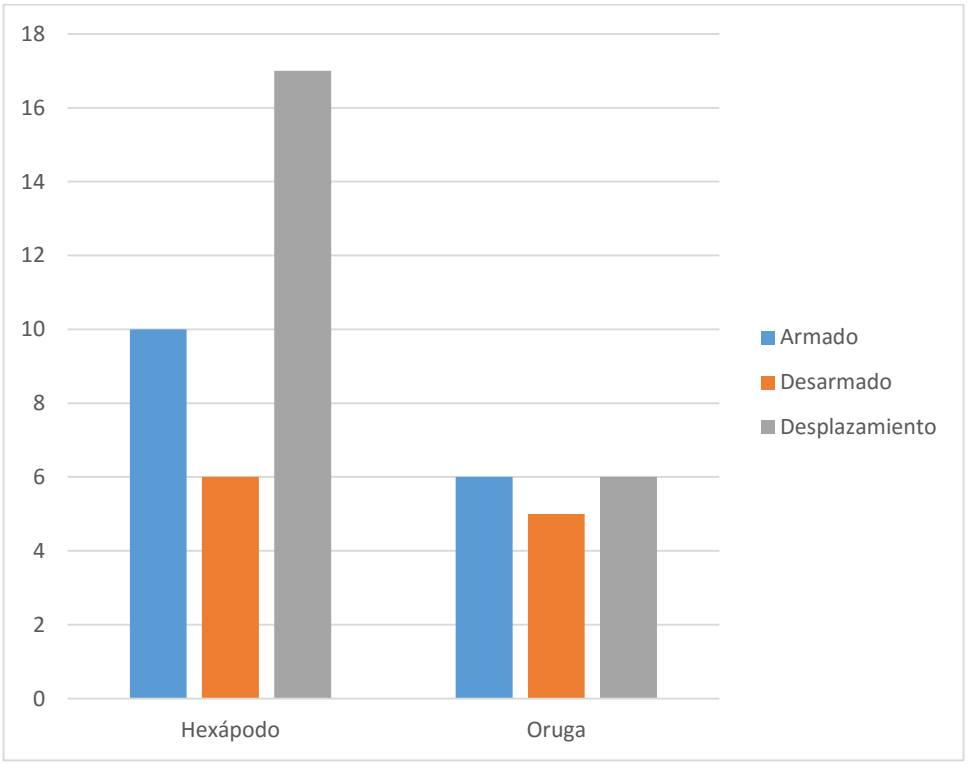


Figura 118. Registro de cantidad de veces ejecutada cada acción para ambas arquitecturas en las trayectorias recorridas. Elaboración propia

CAPÍTULO VII.

CONCLUSIONES

El presente documento presentó el tipo de robot modular de referencia sobre el cual se realizó el trabajo de autoconfiguración de arquitecturas, las diferentes herramientas físicas como tipos de conexión y modos de configuración entre módulos, la aplicación para la simulación, junto con los algoritmos propuestos para la búsqueda y de planificación de trayectorias, verificándose el estado del arte en el que se encuentra el avance de la robótica modular y la autoconfiguración de diferentes arquitecturas, junto con lo desarrollado por la Universidad Militar Nueva Granada como aporte al conocimiento sobre este campo de investigación.

Se propuso el modelamiento de robots modulares MECABOT 4.0 en el Software WEBOTS, con un enfoque sencillo para su accionamiento, tomando como base los trabajos elaborados en el grupo DAVINCI de la Universidad Militar Nueva Granada, con el fin de realizar un trabajo que confluyera en realizar el diseño del control de los módulos, que cuenten con la capacidad de acoplarse entre sí, para que de esta manera a través de protocolos de comunicación, se pueda lograr el armado de las arquitecturas oruga y hexápodo, con la característica de autoconfiguración de arquitecturas para ejecutar una trayectoria previamente planificada por el algoritmo de búsqueda A*.

Los objetivos propuestos del presente proyecto, se plantearon y se lograron sin acudir al uso de la herramienta de supervisor que proporciona el Software

WEBOTS, puesto que el objetivo fue el de simular todas las variables de los módulos y recrear las limitaciones del entorno que asemejen un ambiente real, tendientes a generar y desarrollar un algoritmo idóneo que permitiera servir de base para una futura implementación física en otros trabajos a realizar.

Las fases para la realización del presente proyecto, comprendieron el diseño y desarrollo del algoritmo de armado y desarmado para las arquitecturas oruga y hexápodo, a través de la generación de diferentes herramientas para el uso óptimo de cada módulo, con el empleo y el control eficaz de los elementos disponibles e integrados, para proporcionar a los módulos la capacidad de desplazamiento, de conexión y de comunicación entre ellos, de forma autónoma, integrada y sincronizada para ejecutar las arquitecturas propuestas. El módulo 1 se estableció como el maestro de la operación para coordinar las secuencias de movimiento individual para el desplazamiento en conjunto de cada arquitectura, como también para ubicarlos en disposición de ejecutar cualquiera de las dos arquitecturas planteadas para el presente proyecto.

Otra fase, consistió en el diseño y desarrollo del algoritmo de búsqueda y planificación de trayectorias, para la obtención de la mejor ruta de desplazamiento que permita llegar al punto objetivo, considerando las restricciones del escenario y las limitaciones puestas en el terreno como obstáculos, con el fin de definir la mejor arquitectura con la se va ejecutar cada tramo o recta de la trayectoria obtenida, teniendo en cuenta las condiciones de espacio para una apropiada elección.

Como última fase, se integraron los algoritmos para el armado, desarmado y desplazamiento de las arquitecturas oruga y hexápodo, junto con el de búsqueda y planificación trayectorias, a través del método A*, para lograr que el sistema robótico pueda autoconfigurarse de manera efectiva para recorrer la trayectoria que permita llegar al nodo objetivo. Por último, se diseñó el código que posibilita la interacción del usuario, para la visualización de resultados en la simulación, a través de un menú en la interfaz de simulación del programa en el Software WEBOTS, para elegir el punto final deseado.

Las pruebas y los resultados obtenidos permiten concluir:

- La eficacia en el desarrollo y control de los diferentes elementos, dispositivos y herramientas integradas para el desplazamiento, conexión y comunicación de los módulos, como también los parámetros de seguridad elegidos en la identificación de obstáculos sobre el mapa de nodos, para la planeación de la mejor trayectoria, que permitiera ejecutar sin obstrucción alguna el armado de las arquitecturas planteadas.

- La eficiencia del sistema, acorde a los criterios propuestos para la elección y ejecución de las arquitecturas del sistema, al hacer uso de la arquitectura hexápodo en la situaciones más frecuentes dado su rápido armado y lento desplazamiento, los cuales son para giros y rectas con distancias cortas

menores a siete nodos; en comparación con la arquitectura oruga, la cual se usa en situaciones con menor frecuencia dado su lento armado y rápido desplazamiento, los cuales son para rectas con distancias mayores de 7 nodos, lo que nota un óptimo funcionamiento y notable rendimiento del sistema para el uso de ambas arquitecturas.

- La efectividad del algoritmo desarrollado conforme la metodología propuesta y plasmada en el presente documento, de acuerdo a la secuencia lógica utilizada para realización del proyecto, al conseguirse llevar el sistema robótico desde un punto inicial, hasta un punto final, considerando los diferentes obstáculos puestos en el terreno para obtener la mejor trayectoria y ejecutar la arquitectura más apropiada en cada tramo de la misma, posibilitando la reconfiguración automática de arquitecturas, de manera que se le permitiera al sistema solventar cualquier eventualidad para llegar al punto deseado.

CAPÍTULO VIII.

RECOMENDACIONES Y TRABAJOS FUTUROS

- Continuar con lo desarrollado en el presente proyecto de tesis, para brindar al algoritmo desarrollado la capacidad de permitir mayores alternativas de autoconfiguración al sistema robótico, incorporando diferentes arquitecturas con sus estrategias de armado y desarmado a las planteadas en este documento.

- Aportar en la optimización de diferentes aspectos del sistema robótico desarrollado, tendientes a ejecutar el desplazamiento de las trayectorias en el menor tiempo posible, como por ejemplo, en la ejecución de las estrategias de armado y desarmado para las arquitecturas oruga y hexápodo, el cual puede contar con el desplazamiento individual de módulos de manera simultánea, para ejecutar las estrategias de una manera más rápida, de forma que resulte en mejoras de eficiencia del sistema.

- Brindar al sistema la capacidad autónoma para la identificación de obstáculos, mediante la incorporación de sistemas de visión artificial, como también el desarrollo de herramientas de procesamiento de imágenes e identificación de colores, para a evadir de manera automática los diferentes obstáculos que pueden variar de posición de manera aleatoria.

- Llevar el presente proyecto a su siguiente fase de implementación, considerando las diferentes herramientas planteadas y plasmadas en el documento, como por ejemplo, el control de desplazamiento individual, los protocolos de comunicación entre módulos (maestro – esclavo) y el acoplamiento entre los mismos, las cuales fueron generadas sin acudir al uso del robot supervisor que proporciona el Software WEBOTS, para poder recrear y generar un algoritmo lo más parecido al que se implementaría en un entorno físico real.

CAPÍTULO IX.

BIBLIOGRAFÍA

[1] M. S. W. M. S. B. R. D. M. M. L. H. .. &. C. G. S. Yim, «Modular self-reconfigurable robot systems [grand challenges of robotics.,» *IEEE Robotics & Automation Magazine,* 2007.

[2] Z. M. S. &. R. D. Butler, «Distributed replication algorithms for self-reconfiguring modular robots. Distributed Autonomous Robotic Systems,» pp. 37-48, 2002.

[3] K. D. &. R. D. L. Kotay, «Algorithms for self-reconfiguring molecule motion planning. In Intelligent Robots and Systems,» 2000.

[4] K. &. R. D. Kotay, «Generic distributed assembly and repair algorithms for self-reconfiguring robots. In Intelligent Robots and Systems,» 2004.

[5] K. Kotay, «Self-reconfiguring robots: designs, algorithms, and applications.».

[6] D. &. K. K. Rus, «Scalable parallel algorithm for configuration planning for self-reconfiguring robots. In Proc. of the Conf. on Sensor Fusion & Decentralized Control in Robotic Systems III».

[7] T. &. R. S. Larkworthy, «An efficient algorithm for self-reconfiguration planning in a modular robot. In Robotics and Automation,» de *IEEE International Conference*, 2010.

[8] Z. &. R. D. Butler, «Distributed locomotion algorithms for self-reconfigurable robots operating on rough terrain. In Computational Intelligence in Robotics and Automation,» de *IEEE International Symposium*, 2003.

[9] D. d. r. -. roboticspot.com, 2 Diciembre 2008. [En línea].

[10] I. Spotlight, « Robotics from Monster Career Advice,» 26 Agosto 2007.

[11] S. Bermejo, «Desarrollo de robots basados en el comportamiento,» *Ediciones UPC,* Vols. %1 de %2ISBN 84-8301-712-1, pp. Págs. 26-27, 2003.

[12] G. BERMÚDEZ, «ROBOTS MÓVILES. TEORIA, APLICACIONES Y EXPERIENCIAS,» *T E C N U R A,* p. 13, 2002.

[13] P. B.-T. P. Moubarak, «Adaptive Manipulation of a Hybrid Mechanism Mobile Robot,» *IEEE International Symposium on Robotic and Sensors Environments (ROSE),* pp. 113 - 118, 2011.

[14] B. Gopalakrishnan, S. Tirunellayi y R. Todkar, «Design and development of an autonomous mobile smart vehicle: a mechatronics application,» *Mechatronics,* p. 14: 491–514, (2014).

[15] M.-C. Shiu, H.-T. Lee, F.-L. Lian y L.-C. Fu, «Design of the Octabot Self-Reconfigurable Robot,» *Proceedings of the 17th World Congress The International Federation of Automatic Control,* p. 6, 2008.

[16] M. YIM, P. WHITE, M. PARK and J. SASTRA, "Modular Self-Reconfigurable Robots," *ENCYCLOPEDIA OF COMPLEXITY AND SYSTEMS SCIENCE,* vol. 14, p. 14, 2009.

[17] C. Liu, J. Liu, R. Moreno, F. Veenstra y A. Faina, «The impact of module morphologies on modular robots,» *International Conference on Advanced Robotics (ICAR),* vol. Proceedings of the 2017 18th, nº 1, pp. 237-243, 2017.

[18] P. Moubarak, « Modular and Reconfigurable Mobile Robotics,» *Journal of Robotics and Autonomous Systems,* p. 1648 – 1663, 2012.

[19] R. Alattas, «Hybrid Evolutionary Designer of Modular Robots,» *Department of Computer Science & Engineering - University of Bridgeport,* vol. IEEE, nº 978-1-5090-0799-8/16, pp. 1-4, 2016.

[20] M. Yim, D. Duff y Y. Zhang, «Modular robots,» *IEEE Spectrum,* vol. 39, nº 2, p. 30–34, Febrero 2002.

[21] J. Gonzáles Gómez, «Robótica modular y locomoción: Aplicación a robots ápodos,» de *XI Semana de la investigación y cultura en la facultad de informática*, Madrid, 2007.

[22] M. YIM, LOCOMOTION WITH A UNIT-MODULAR RECONFIGURABLE ROBOT, Stanford University, 1994.

[23] S. H. Turlapati, A. Srivastava, K. M. Krishna y S. V. Shah, «Detachable Modular Robot capable of Cooperative Climbing and MultiAgent Exploration,» *IEEE International Conference on Robotics and Automation (ICRA),* vol. Singapore, p. 6, 2017.

[24] P. Guimarães, M. Nunes, T. Galembeck, L. Bamidele, R. Tenório, D. Magalhães y F. de Barros Vidal, «A bio-inspired apodal and modular robot,» *Latin American Robotics Symposium and IV Brazilian Robotics Symposium,* vol. 13, p. 6, 2016.

[25] M. Losada Rodríguez, R. Guerrero Gómez-Olmedo, V. Bartolomé, P. Pérez Hernández, D. Borja Pérez y A. Sánchez, Robots Modulares - Introducción al diseño de Microrrobots Móviles, 2006.

[26] T. Tosun, J. Davey, C. Liu y M. Yim, «Design and characterization of the EP-Face connector,» *International Conference on Intelligent Robots and Systems (IROS), ,* vol. IEEE/RSJ, pp. 45-51., 2016.

[27] N. Rost, «Self-Soldering Connectors for Modular Robots,» [En línea]. Available: Disponible en: /paper/Self-Soldering-Connectors-for-Modular-Robots-Neubert-Rost/2e92ddcf2e7a9d6c27875ec442637e13753f21a2.. [Último acceso: 20 12 2018].

[28] F. B. A. a. P. B. N. Brener, «Characterization of Lattice Modular Robots by Discrete Displacement Groups,» *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS),* 2010.

[29] R. Pfeifer, M. Lungarella y F. Iida, «Self-Organization, Embodiment, and Biologically Inspired Robotics,» *Science,* vol. 318, nº 5853, pp. 1088-1093, 2007.

[30] I. IPCED, «Búsqueda y Rescate,» IPCED., [En línea]. Available: http://www.ipced.mx/cursos/busqueda-rescate/. [Último acceso: 19 02 2019].

[31] V. Kee, «Modular Self-Configurable Robots / Angles,» *MIT Media Studies,* p. 1, 2013.

[32] F. Hou y W.-M. Shen, «Distributed, dynamic, and autonomous reconfiguration,» *IEEE International Conference on Robotics and Automation,* vol. ICRA 2008, pp. 3135 –3140, 19-23, 2008.

[33] K. Stoy, W. Shen y P. Will, «Using role-based control to produce locomotion,» *IEEE/ASME transactions on mechatronics,* vol. 7(4), p. 410–417, 2002.

[34] D. Martinez, «Tesis Doctoral. Planificación de Trayectorias para Robots Móviles,» *Universidad,* 1995.

[35] J.-C. Latombe, «Robot Motion Planning.,» 1990.

[36] J. D. Preciado Aguilar, «PLANIFICACIÓN DE TRAYECTORIAS DE ROBOTS MÓVILES DE DIFERENTES ARQUITECTURAS EN ENTORNOS DINÁMICOS,» 2018. [En línea]. Available: https://repository.unimilitar.edu.co/bitstream/handle/10654/18006/PreciadoAguilarJuanDavid2018.pdf?sequence=1&isAllowed=y.

[37] J. Banks, J. Carson, B. Nelson y D. Nicol, Discrete-Event System Simulation, Prentice Hall, 2010.

[38] E. D. Reilly, Concise Encyclopedia of Computer Science, John Wiley & Sons, 2004.

[39] Wiley.com., «Principles of Modeling and Simulation: A Multidisciplinary Approach,» [En línea]. Available: https://www.wiley.com/en-us/Principles+of+Modeling+and+Simulation%3A+A+Multidisciplinary+Approach-p-9780470289433. . [Último acceso: 2019 febrero 23].

[40] J. Weng, «Creación del Modelo del Robot Humanoide Teo para el Simulador WEBOTS,» *IEEE,* 2012.

[41] Cyberbotics Ltd., «Webots Reference Manual R2019a - Robot,» Cyberbotics Ltd., 01 01 2019. [En línea]. Available: https://www.cyberbotics.com/doc/reference/robot?tab=c. [Último acceso: 2019 02 28].

[42] M. F. R. A. y. M. A. S.-U. Juan A. Escalera, «TELEMANIPULACIÓN Y LOCOMOCIÓN MEDIANTE ROBOTS MODULARES,» *RIAI,* 2008.

[43] P. Levi, E. Meister, A. van Rossum, T. Krajnik, V. Vonasek, P. Stepan y W. Liu, «A cognitive architecture for modular and self-reconfigurable robots,» p. 465–472, 31 Marzo 2014.

[44] M. Yim, D. G. Duff y K. D. Roufas, «PolyBot: a Modular Reconfigurable Robot,» *Proceedings of the IEEE International Conference on Robotics and,* p. 514–520, Abril 2000.

[45] M. Yim, A. Golovinsky, Z. Ying y C. Eldershaw, «PolyBot and PolyKinetic System: a modular,» *Proceedings of the IEEE International Conference on,* vol. 2, p. 1381– 1386, 2004.

[46] M. Park, S. Chitta, A. Teichman y M. Yim, «Automatic Configuration Recognition Methods,» *The International Journal of Robotics Research,* vol. 27, nº 3-4, p. 403–421, 2008.

[47] S. Murata y H. Kurokawa, «Self-Reconfigurable Robot,» *IEEE Robotics and Automation,* vol. 14, nº 1, p. 71–78, 2007.

[48] «http://unit.aist.go.jp/is/frrg/dsysd/mtran3/,» [En línea].

[49] «M-TRAN(Modular Transformer) MTRAN III,» [En línea]. Available: Disponible en: https://unit.aist.go.jp/is/frrg/dsysd/mtran3/. . [Último acceso: 18 Febrero 2018].

[50] A. Castano, W. Shen y P. Will, «CONRO: Towards deployable robots with inter-robots,» *Autonomous Robots,* vol. 8, nº 3, p. 309–324, 2000.

[51] W.-M. Shen, J. Bogdanowicz, W. Chun, M. Yim, P. Will, M. SIms, S. Colombano, D. Kortenkamp, S. Vanderzyl, E. Baumgartner y J. Taylor, «SuperBots: Modular, multifunctional,,» *Lunar Exploration Analysis Group,* 2005.

[52] J. Liu, S. Ma, Z. Lu, Y. Wang, B. Li y J. Wang, «Design and experiment of a novel link-type shape shifting modular robot series,» *IEEE,* 2005.

[53] W. Y. L. B. M. S. T. D. Liu JinGuo, «Center-configuration selection technique for the reconfigurable modular robot,» *Springer link,* 2007.

[54] «http://www.idsc.ethz.ch/research-dandrea/research-projects/distributed-flight-array.html,» [En línea].

[55] «http://robohub.org/robots-m-blocks/,» [En línea].

[56] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui y A. Jan Ijspeert, «Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface,» *Industrial Robot,,* vol. 33, nº 4, p. 285–290, 2006.

[57] M. W. Jorgensen, E. H. Ostergaard y H. H. Lund, «Modular ATRON: modules for a selfreconfigurable,» *Proceedings of the IEEE/RSJ International Conference on Intelligent,* vol. 2, nº ISBN 0-7803-8463-6, p. 2068– 2073, 2005.

[58] N. Correll, C. Wailes y S. Slaby, «A One-hour Curriculum to Engage Middle School Students in Robotics and Computer Science using Cubelets,» *Distributed Autonomous Robotic Systems,* vol. Springer Berlin Heidelberg, pp. 165-176, 2014.

[59] j. Baca, B. Woosley, P. Dasgupta y C. Nelson, «Real-Time Distributed Configuration Discovery of Modular Self-reconfigurable Robots,» *IEEE International Conference on Robotics and Automation (ICRA),* vol. Washington State Convention Center, p. 6, 2015.

[60] X. Wang, J. Hongzhe, Y. Zhu, B. Chen, D. Bie, Y. Zhang y J. Zhao, «Serpenoid polygonal rolling for chain-type modular robots: A study of modeling, pattern switching and application,» *Robotics and Computer-Integrated Manufacturing - Elsevier,* vol. 39, pp. 56-67, 2015.

[61] O. G. Rubiano Montaña y C. A. Hurtado Erasso, «Diseño y simulación de un robot modular reconfigurable,» *Repositorio Institucional - Universidad Militar Nueva Granada,* vol. http://hdl.handle.net/10654/10150, p. 205, 2013.

[62] P. N. Lancheros Guzmán y L. B. Sanabria Galvis, «Simulación e Implementación de Movimientos para Sistema Robótico Modular Considerando Diferentes Configuraciones,»

Repositorio Institucional - Universidad Militar Nueva Granada, vol. http://hdl.handle.net/10654/15382, p. 157, 2016.

[63] W. A. Miño Rubiano, «Simulación e implementación de sistema robotico en arquitectura tipo rueda utilizando robotica modular,» *Repositorio institucional - Universidad Militar Nueva Granada,* vol. http://hdl.handle.net/10654/16081, p. 73, 2016.

[64] B. M. Cotera, «Simulación e Implementación de una Configuración de Robot Hexápodo Utilizando Sistemas de Robótica Modular para Evaluar su Locomoción,» *Repositorio Documental - Universidad Militar Nueva Granada,* vol. http://hdl.handle.net/10654/16120, p. 135, 2017.

[65] C. Queretaro, «Diseño Mecatrónico de un Robot Móvil (Configuración Diferencial),» *8º,* 2009.

[66] Cyberbotics Ltd., «Webots Reference Manual R2019a - Connector,» Cyberbotics Ltd., 01 01 2019. [En línea]. Available: https://cyberbotics.com/doc/reference/connector. [Último acceso: 2019 02 26].

[67] Cyberbotics Ltd., «Webots Reference Manual R2019a - RotationalMotor,» 01 01 2019. [En línea]. Available: https://cyberbotics.com/doc/reference/rotationalmotor. [Último acceso: 2019 02 26].

[68] Cyberbotics Ltd., «Webots Reference Manual R2019a - HingeJoint,» Cyberbotics Ltd., 01 01 2019. [En línea]. Available: https://cyberbotics.com/doc/reference/hingejoint. [Último acceso: 26 02 2019].

[69] Cyberbotics Ltd., «Webots Reference Manual R2019a - GPS,» Cyberbotics Ltd., 01 01 2019. [En línea]. Available: https://www.cyberbotics.com/doc/reference/gps?tab=c. [Último acceso: 01 01 2019].

[70] Cyberbotics Ltd., «Webots Reference Manual R2019a - Compass,» Cyberbotics Ltd., 01 01 2019. [En línea]. Available: https://www.cyberbotics.com/doc/reference/compass. [Último acceso: 02 03 2019].

[71] Cyberbotics Ltd., «Webots Reference Manual R2019a - Emitter,» 2019. [En línea]. Available: https://www.cyberbotics.com/doc/reference/emitter. [Último acceso: 6 Marzo 2019].

[72] Cyberbotics Ltd., «Webots Reference Manual R2019a - Receiver,» 2019. [En línea]. Available: https://www.cyberbotics.com/doc/reference/receiver. [Último acceso: 6 Marzo 2019].

[73] Querelle y Cia Ltda., «Diseño, programación y desarrollo: Profesor en Línea.,» Querelle y Cia Ltda. , 01 01 2019. [En línea]. Available:

http://www.profesorenlinea.cl/geometria/PitagorasTeorema.htm. [Último acceso: 12 03 2019].

[74] A. F. Mendieta, «ACADEMIA,» 2017. [En línea]. Available: https://www.academia.edu/6879670/Electroim%C3%A1n. [Último acceso: 18 Marzo 2019].

CAPÍTULO X.

ANEXOS

Anexo 1. Declaración de variables utilizadas para el desarrollo del programa de control del módulo 1

```
static WbDeviceTag motor_pivote;
static WbDeviceTag motor_rueda_pivote;
static WbDeviceTag motor_rueda_izquierda;
static WbDeviceTag motor_rueda_derecha;
static WbDeviceTag motor_rueda_posterior;
static WbDeviceTag conector_pivote;
static WbDeviceTag conector_izquierdo;
static WbDeviceTag conector_derecho;
static WbDeviceTag conector_posterior;
static WbDeviceTag globalrot;
static WbDeviceTag globalpos;
static WbDeviceTag emisor;
static WbDeviceTag receptor;
```

```
//variables A*
int cont_kl,k_star,l_star,current,t_recta,e,s,obstac_fila,obstac_columna,obstaculo_encontrado;
char map[map_size_rows][map_size_cols];
int ind[map_size_rows][map_size_cols];
int i, j, b, found,cont_nodo,con_recta,t_nodo,contador_nodos_recta,contador_nodos_recta_anterior;
int fin,cont_final,end,fila_final,columna_final,inicio,start,fila_inicial,columna_inicial;
int p_len = 0;
int * path = NULL;
int c_len = 0;
int * closed = NULL;
int o_len = 1;
double mediancho,min, tempg;
int cub_obs,cantidad_cubos;
int s_len = 0;
struct stop * stops = NULL;
int r_len = 0;
struct route * routes = NULL;
struct obstaculo * obstaculos = NULL;
struct info_trayect * info_trayectos = NULL;
struct trayecto_arquitectura * trayecto_arquitecturas = NULL;
int mapa_coord_fil[map_size_rows+1],mapa_coord_col[map_size_cols+1];
int ancho_nodo=250;//Longitud de cuadro o nodo
int dimension_en_x,dimension_en_z,posactual_x,posactual_z,nodo_fila_actual,nodo_columna_actual,posfinal_x,posfinal_z,nodo_fila_final,nodo_columna_final;

//Variables robots
static int columna_nodofinal,fila_nodofinal,bandera=0,num_teclado,num_teclado_ant,key,key_anterior=0,p_rig,p_rdg,arquitectura,modulo_conectar,com_forma;
static int modulo_accion,angulo=0,alfa,flag=0,zf=3625,xf=3625,orientacion_final=270,robotflag=0,recta_ejecucion=0;
static int condicion_giro=0,info_paq,fila,columna,final_morfologia=0,oruga_movimiento=10,contador_movimiento=0,hexapodo_movimiento=10,message_printed = 0;
static double pos_ri=0,pos_rd=0,dist=0,dist_anterior=0;
const double *pos3D,*north3D;
static double angxz[2]={0,0};
static int posxz[2]={0,0};
static int datosderecibo[6]={0,0,0,0,0,0};
static int datosmodulos[16][6];
static div_t outputi,outputd;
static double t=0.0; //
static int id;//
static const double A=0.5;//
static const double F=1;//
static int k=3;//
static int M=15;//
static double d= 1/5;//

//Estructura A*
/* description of graph node */
struct stop {
  double col, row;
  /* array of indexes of routes from this stop to neighbours in array of all routes */
  int * n;
  int n_len;
  double f, g, h;
  int from;
};
/* description of route between two nodes */
struct route {
  /* route has only one direction! */
  int x; /* index of stop in array of all stops of src of this route */
  int y; /* intex of stop in array of all stops od dst of this route */
  double d_star;
};
/* informacion obstaculos*/
struct obstaculo{
  int obst_coord_n; //coordenada en X
  int obst_coord_m; // coordenada en Y
  double obst_ancho;
};
/* informacion trayectoria*/
struct info_trayect{
  double fila_nodo;
  double columna_nodo;
  int pendiente;
  int recta;
};
```

```
/* informacion arquitectura sobre la trayectoria*/
struct trayecto_arquitectura{
  int numero_linea;
  int cantidad_nodos;
  int forma_ejecutar;
  int nodo_inicia_recta;
};
```

Anexo 2. Declaración de variables utilizadas para el desarrollo del programa de control del módulo 2 al 15

```
static WbDeviceTag motor_pivote;
static WbDeviceTag motor_rueda_pivote;
static WbDeviceTag motor_rueda_izquierda;
static WbDeviceTag motor_rueda_derecha;
static WbDeviceTag motor_rueda_posterior;
static WbDeviceTag conector_pivote;
static WbDeviceTag conector_izquierdo;
static WbDeviceTag conector_derecho;
static WbDeviceTag conector_posterior;
static WbDeviceTag globalrot;
static WbDeviceTag globalpos;
static WbDeviceTag emisor;
static WbDeviceTag receptor;

static int p_rig,p_rdg,pasos_acercamiento=0,condicion_giro=0,angulo=0,alfa,i,flag=-2,zf,xf,orientacion_final,puntomin;
static int message_printed = 0; /* used to avoid printing continuously the communication state */
static double pos_ri=0,pos_rd=0,dist=0,dist5p[7],menor;
const double *pos3D,*north3D;
static double angxz[2]={0,0};
static int posxz[2]={0,0};
static int p1xz[2],p2xz[2],p3xz[2],p4xz[2],p5xz[2],p6xz[2],p7xz[2],p8xz[2],p9xz[2],p10xz[2],p11xz[2],p12xz[2],p13xz[2];
static int datosderecibo[6]={0,0,0,0,0,0};
static div_t outputi,outputd;
static double t=0.0; //
static int id;//
static const double A=0.5;//
static const double F=1;//
static int k=3;//
static int M=15;//
static double d= 1/5;//
```

Anexo 3. Secuencia de programación en el controlador de la función principal del módulo 1 (módulo maestro)

```
int main()
{
  inicializacion_variables_dispositivos();
  printf("PARA EMPEZAR PRESIONE TECLA (e)\n");
  while (wb_robot_step(TIME_STEP) != -1) {
    leer_teclado();
    menu_programa();
    if(bandera==11){
      algoritmo_a_star();//funcion para hallar la trayectoria ejecutando el algoritmo A*
    }
    if(bandera==13){
      //Se verifica si se recibe informacion de algun modulo
      recepcion_informacion_modulo();
      //Para realizar el movimiento del modulo individual para llevarlo a la posicion y orientacion deseada
      if(flag<=6){desplazamiento_modulo();}
      //para ejecutar la trayectoria planificada utilizando las dos arquitecturas
      ejecutar_trayectoria();
      //para ejecutar alguna arquitectura y sus movimientos
      ejecutar_morfologia();
    }
  };
  wb_robot_cleanup();
  return 0;
}

void inicializacion_variables_dispositivos(){
  int channel;
  wb_robot_init();
  const char *name = wb_robot_get_name();
  id =atoi(name+8);
  //variables motores
  motor_pivote=wb_robot_get_device("motor_pivote");
  motor_rueda_pivote=wb_robot_get_device("motor_rueda_pivote");
  motor_rueda_izquierda=wb_robot_get_device("motor_rueda_izquierda");
  motor_rueda_derecha=wb_robot_get_device("motor_rueda_derecha");
  motor_rueda_posterior=wb_robot_get_device("motor_rueda_posterior");
  // variables connector
  conector_pivote=wb_robot_get_device("conector_pivote");
  conector_izquierdo=wb_robot_get_device("conector_izquierdo");
  conector_derecho=wb_robot_get_device("conector_derecho");
  conector_posterior=wb_robot_get_device("conector_posterior");
    // get gps tag and enable
  globalpos= wb_robot_get_device("gps");
  wb_gps_enable(globalpos, TIME_STEP);
  wb_robot_step(TIME_STEP);    // execute simulation step
  // get compass tag and enable
  globalrot = wb_robot_get_device("compass");
  wb_compass_enable(globalrot, TIME_STEP);
  wb_robot_step(TIME_STEP);    // execute simulation step
  // obtener emisor
  emisor=wb_robot_get_device("emitter");
  /* if the channel is not the right one, change it */
  channel = wb_emitter_get_channel(emisor);
  if (channel != COMMUNICATION_CHANNEL_EMITTER) {
    wb_emitter_set_channel(emisor, COMMUNICATION_CHANNEL_EMITTER);
  }
```

```
  // obtener receptor
  receptor=wb_robot_get_device("receiver");
  /* if the channel is not the right one, change it */
  channel = wb_receiver_get_channel(receptor);
  if (channel != COMMUNICATION_CHANNEL_RECEIVER) {
    wb_receiver_set_channel(receptor, COMMUNICATION_CHANNEL_RECEIVER);
  }
  wb_receiver_enable(receptor, TIME_STEP);
  /* get key presses from keyboard */
  wb_robot_keyboard_enable(TIME_STEP);
  arquitectura=2;//se inicia arquitectura hexapodo
  //se inicia arreglo de datos de todos los modulos en 0
  for(fila=0;fila<16;fila++){
    for(columna=0;columna<6;columna++){
      datosmodulos[fila][columna]=0;
    }
  }
  dimension_en_x=5000;dimension_en_z=5000; //en milimetros
  //se inicia el arreglo que contiene las coordenadas del mapa
  for(fila=0;fila<=map_size_rows;fila++){
    mapa_coord_fil[fila]=dimension_en_z-ancho_nodo*fila;
  }
  for(columna=0;columna<=map_size_cols;columna++){
    mapa_coord_col[columna]=dimension_en_x-ancho_nodo*columna;
  }
}
```

Anexo 4. Caracterización dispositivo COMPASS, resultados arrojados en los ejes del dispositivo a cada grado de orientación del módulo.

Ángulo (grados)	*Lectura Eje "X" asin(Comp [x])*	*Lectura Eje "Z" acos(Comp [z])*
0	0,139739	0,139856
1	0,950058	0,950101
2	1,810317	1,81038
3	2,693846	2,693944
4	3,700543	3,700633
5	4,538819	4,538902
6	6,039038	6,039074
7	6,817947	6,817965
8	7,892184	7,892194
9	8,696342	8,696353
10	9,53226	9,532263
11	10,592123	10,592127
12	12,131443	12,131451
13	12,839552	12,839556
14	13,719242	13,719244
15	14,592302	14,592305
16	15,564192	15,564195
17	16,886313	16,886319
18	17,625073	17,625074
19	19,285723	19,285727
20	20,130582	20,130589
21	21,01338	21,013386
22	21,908388	21,908391
23	22,572071	22,572071
24	24,229352	24,229356
25	25,108278	25,108284
26	26,116061	26,116065
27	26,960763	26,960765
28	28,301983	28,301984
29	29,114286	29,114287
30	29,802649	29,802653
31	30,975121	30,975121
32	31,813934	31,813936
33	32,688759	32,688761
34	33,68892	33,688922
35	34,657413	34,657413
36	35,527295	35,527296
37	36,524254	36,524256
38	37,738155	37,738155
39	39,346957	39,346958
40	40,175098	40,1751
41	41,046143	41,046147
42	41,937041	41,937045
43	42,828447	42,828448
44	43,530061	43,530066
45	45,25813	45,25813
46	46,148177	46,14818
47	46,547676	46,547684
48	48,256655	48,256655
49	49,254744	49,254744
50	50,091176	50,091181
51	50,959955	50,959963
52	51,552076	51,552084
53	53,284234	53,284234
54	54,114424	54,114424
55	54,985955	54,985957
56	55,816279	55,816286
57	56,574778	56,574786
58	58,315555	58,315555
59	59,21101	59,211012
60	59,985717	59,985721
61	60,820226	60,820234
62	61,690755	61,690756
63	62,569518	62,569521
64	64,448471	64,448471
65	65,28769	65,287692
66	65,77363	65,77364
67	66,530611	66,530612
68	68,15546	68,155461
69	69,021684	69,021684
70	69,908973	69,908977
71	70,671371	70,671384
72	72,306	72,306018
73	73,166911	73,166921
74	74,046685	74,046689
75	74,938331	74,938332
76	75,951419	75,951419
77	76,787535	76,787536
78	77,623661	77,623661
79	79,119151	79,119154
80	80,127574	80,127577
81	80,964651	80,964661
82	81,837015	81,837037
83	82,726615	82,726653
84	83,586306	83,586329
85	85,308549	85,308569
86	86,180194	86,180197
87	87,068924	87,068928
88	87,965632	87,965693
89	89,187369	89,187398

90	89,872603	89,875031
91	89,236067	90,763737
92	88,339485	91,660498
93	87,325112	92,674883
94	86,242149	93,757815
95	85,418548	94,581434
96	83,810839	96,189129
97	83,016327	96,983667
98	82,15266	97,847339
99	81,331991	98,667993
100	80,092395	99,907581
101	79,335197	100,664777
102	77,672715	102,327272
103	76,778139	103,221856
104	75,766782	104,233218
105	74,951794	105,048206
106	74,340372	105,659597
107	72,699713	107,300286
108	71,698111	108,301889
109	70,856035	109,143964
110	69,982994	110,017003
111	69,093964	110,906033
112	68,19989	111,800107
113	67,229108	112,770888
114	66,407888	113,59211
115	64,64357	115,356427
116	64,323609	115,676389
117	63,159958	116,840026
118	62,339076	117,660912
119	61,478036	118,521959
120	59,739498	120,2605
121	59,164326	120,835674
122	58,310655	121,689339
123	57,437697	122,562291
124	55,580062	124,419935
125	54,745169	125,254831
126	54,07499	125,925008
127	53,397154	126,602841
128	52,435723	127,564271
129	50,755825	129,244173
130	49,868703	130,131297
131	48,985922	131,014078
132	47,990728	132,009269
133	47,159258	132,84073
134	46,317591	133,682393
135	45,445868	134,554119
136	43,538826	136,461174
137	42,78868	137,211319
138	41,932017	138,067981
139	41,291933	138,708058
140	40,418447	139,581544
141	39,4104	140,589594
142	37,697493	142,302507
143	37,352474	142,647526
144	36,468848	143,531148
145	34,784702	145,215289
146	33,926578	146,073408
147	33,03923	146,960757
148	32,143272	147,85672
149	31,133727	148,866272
150	30,294766	149,70523
151	29,421723	150,57827
152	27,644027	152,355965
153	26,631498	153,368501
154	25,954601	154,045399
155	25,248158	154,751837
156	24,372397	155,627594
157	23,372249	156,627743
158	21,664024	158,335975
159	21,286078	158,713922
160	20,436541	159,563457
161	18,930761	161,069228
162	18,065597	161,934391
163	17,179953	162,820038
164	16,282795	163,717203
165	15,178447	164,821548
166	13,62237	166,377608
167	12,731637	167,268334
168	11,721436	168,278529
169	10,88327	169,116698
170	10,009263	169,990723
171	9,123124	170,876862
172	8,227398	171,772594
173	7,219655	172,78032
174	6,382084	173,617874
175	4,62315	175,376775
176	3,726791	176,273173
177	2,714346	177,285632
178	1,890707	178,109256
179	1,018472	178,98139
180	0,129553	179,869192
181	-0,766874	179,232824
182	-1,780064	178,219812
183	-2,619593	177,380338
184	-4,382939	175,617037
185	-5,281057	174,718911
186	-6,258029	173,741951
187	-7,097783	172,902203
188	-7,906791	172,093201
189	-9,334609	170,665385

190	-10,117963	169,882026
191	-10,97018	169,029809
192	-11,768897	168,231093
193	-12,749307	167,25069
194	-13,532272	166,467722
195	-15,188418	164,811556
196	-16,077118	163,922861
197	-17,089047	162,910945
198	-17,586443	162,413553
199	-18,537801	161,462199
200	-20,364863	159,635134
201	-21,192982	158,807012
202	-21,565567	158,434423
203	-22,529116	157,470884
204	-24,298513	155,701477
205	-25,161059	154,838927
206	-26,049761	153,950226
207	-26,951701	153,048293
208	-28,28481	151,71519
209	-29,135547	150,864451
210	-30,016153	149,983843
211	-30,637211	149,362783
212	-32,13847	147,86153
213	-33,120251	146,879747
214	-33,950757	146,049238
215	-34,827081	145,172908
216	-35,705801	144,29419
217	-36,510111	143,489883
218	-38,209999	141,79
219	-39,067112	140,932886
220	-39,946491	140,053505
221	-40,766537	139,233462
222	-41,865202	138,134798
223	-42,75066	137,249338
224	-43,650244	136,349746
225	-45,221065	134,778934
226	-46,203655	133,796345
227	-47,030197	132,969801
228	-47,576488	132,423506
229	-48,872306	131,127694
230	-49,723361	130,276638
231	-50,60123	129,398767
232	-52,066559	127,933438
233	-52,717078	127,282922
234	-53,57727	126,42273
235	-55,399255	124,600742
236	-55,715679	124,284312
237	-57,19578	122,80422
238	-58,023376	121,976624
239	-58,874615	121,125382
240	-60,281083	119,718917
241	-61,113997	118,886001
242	-61,976131	118,023868
243	-62,865163	117,134837
244	-63,758931	116,241066
245	-65,237463	114,762534
246	-66,243147	113,756853
247	-66,718722	113,281274
248	-67,857018	112,142981
249	-68,68751	111,312483
250	-69,554347	110,44564
251	-71,336039	108,663937
252	-72,02754	107,972456
253	-72,84731	107,152687
254	-73,728769	106,271231
255	-74,621749	105,378248
256	-76,178811	103,821188
257	-77,186823	102,813165
258	-78,02302	101,976957
259	-78,894451	101,105513
260	-79,704384	100,295562
261	-80,538786	99,461209
262	-81,733243	98,266754
263	-82,603952	97,396048
264	-83,743831	96,256106
265	-84,629186	95,37079
266	-85,523788	94,476211
267	-86,829393	93,170606
268	-87,715932	92,283994
269	-88,610271	91,38955
270	-89,62274	90,376942
271	-88,922317	88,922355
272	-88,044581	88,044622
273	-87,079258	87,079403
274	-86,238946	86,239034
275	-85,364995	85,365042
276	-84,475361	84,475378
277	-83,456653	83,456677
278	-81,943688	81,943747
279	-81,117966	81,118013
280	-80,27708	80,277112
281	-79,268674	79,268689
282	-78,430325	78,430328
283	-77,129849	77,129867
284	-76,331087	76,331091
285	-75,427834	75,42785
286	-73,895476	73,895481
287	-73,027477	73,027477
288	-72,357676	72,357678
289	-70,906283	70,9063

290	-70,047312	70,047318
291	-69,059487	69,059487
292	-67,731275	67,731281
293	-67,082514	67,082547
294	-66,115442	66,115472
295	-65,292296	65,292321
296	-64,420542	64,420557
297	-62,630708	62,630708
298	-62,265982	62,265991
299	-61,166726	61,166739
300	-60,302262	60,302267
301	-59,415472	59,415472
302	-58,300523	58,300542
303	-57,442689	57,4427
304	-56,450827	56,450829
305	-54,75278	54,752781
306	-54,074871	54,074871
307	-53,191965	53,191967
308	-52,35453	52,354535
309	-50,67342	50,673421
310	-49,848395	49,848395
311	-49,285989	49,28599
312	-48,176253	48,176265
313	-47,30475	47,304757
314	-46,415258	46,41526
315	-44,922157	44,922157
316	-44,060742	44,060743
317	-43,181334	43,181341
318	-42,309429	42,309437
319	-41,302174	41,302177
320	-40,46616	40,466161
321	-39,480486	39,48049
322	-37,841369	37,841376
323	-36,972407	36,972412
324	-36,086645	36,086646
325	-35,394968	35,394968
326	-33,739531	33,739545
327	-32,882574	32,882601
328	-32,001949	32,001987
329	-31,111865	31,111903
330	-30,099715	30,099739
331	-29,26538	29,265389
332	-28,404201	28,404203
333	-26,971714	26,971726
334	-25,971356	25,971369
335	-25,133676	25,133693
336	-24,262441	24,262455
337	-23,371548	23,371553
338	-22,485615	22,485616
339	-21,018444	21,018455
340	-20,153936	20,153947
341	-19,26844	19,268447
342	-18,370924	18,370925
343	-17,383842	17,383847
344	-15,880649	15,880665
345	-14,990412	14,990429
346	-13,978111	13,978118
347	-13,231099	13,231102
348	-12,226992	12,226999
349	-10,639435	10,639453
350	-9,802447	9,802466
351	-8,929284	8,929296
352	-8,103557	8,103561
353	-6,835954	6,835971
354	-6,001738	6,001808
355	-5,131561	5,131702
356	-4,243862	4,244061
357	-3,347598	3,347801
358	-2,333106	2,333209
359	-1,49346	1,493508

Anexo 5. Secuencia de programación para realizar el desplazamiento individual del módulo 1 (módulo maestro) para llevarlo al punto deseado.

```
void desplazamiento_modulo(){
  // para hallar el angulo de orientacion al punto deseado
  if(flag==0){
    coordenadas_modulo();
    orientacion_modulo();
    //una vez se halla el punto deseado se hace ir al punto deseado con las demas banderas
    if(alfa==180 || alfa==0){flag=3;printf("listo 1a: calculo giro corte en x\n");}
    else {flag=1;printf("listo 1: calculo del angulo de orientacion hacia el punto deseado\n");}
  }
  //para girar hacia al punto deseado
  else if(flag==1){
    coordenadas_modulo();
    if(dist<=1){flag=3;}
    else if((alfa-angulo)==0){
      flag=2;printf("listo 2: giro hacia el punto deseado\n");
    }
    else if((alfa-angulo)>0){
      if(abs(alfa-angulo)<=180){(*states[3].func) ();}
      else {(*states[2].func) ();}
    }
    else{
      if(abs(alfa-angulo)<=180){(*states[2].func) ();}
      else {(*states[3].func) ();}
    }
  }

  //para ir de frente hacia el punto deseado con corte en el eje z
  else if(flag==2){
    coordenadas_modulo();
    if ((alfa-angulo)==0){
      if(zf!=posxz[1] && dist>1){(*states[0].func) ();}
      else {flag=3;printf("listo 3: llegada punto de corte eje z\n");}
    }
    else {
      flag=0;
    }
  }
  else if(flag==3){
    coordenadas_modulo();
    if (dist<=1){printf("listo 6: llegada a punto final\n");flag=6;}
    else if(xf>posxz[0]){alfa=0;flag=4;}
    else {alfa=180;flag=4;}
  }

  //para girar hacia al punto deseado
  else if(flag==4){
    coordenadas_modulo();
    if(dist<=1){flag=3;}
    else if((alfa-angulo)==0){
      flag=5;printf("listo 4: giro para corte en eje x\n");
    }
    else if((alfa-angulo)>0){
      if(abs(alfa-angulo)<=180){(*states[3].func) ();}
      else {(*states[2].func) ();}
    }
    else{
      if(abs(alfa-angulo)<=180){(*states[2].func) ();}
      else {(*states[3].func) ();}
    }
  }
```

```
  //para ir de frente hacia el punto deseado con corte en el eje x
  else if(flag==5){
    coordenadas_modulo();
    if ((alfa-angulo)==0){
      if(xf!=posxz[0] && dist>1){(*states[0].func) ();}
      else {flag=0;printf("listo 5: llegada a punto de corte eje x\n");}
    }
    else {
      flag=0;
    }
  }

  //para girar hacia la orientacion final
  else if(flag==6){
    coordenadas_modulo();
    if(orientacion_final==360){orientacion_final=0;}
    if(orientacion_final<0){orientacion_final=360+orientacion_final;}
    else if(orientacion_final>360){orientacion_final=orientacion_final-360;}
    if(dist>5){flag=0;}
    else if((orientacion_final-angulo)==0){
      com_forma=1;
      registrar_datos_modulo1();
      flag=7;printf("listo 7: giro hacia la orientacion final\n");
    }
    else if((orientacion_final-angulo)>0){
      if(abs(orientacion_final-angulo)<=180){(*states[3].func) ();}
      else {(*states[2].func) ();}
    }
    else{
      if(abs(orientacion_final-angulo)<=180){(*states[2].func) ();}
      else {(*states[3].func) ();}
    }
  }
}
```

Anexo 6. Secuencia de programación para realizar el desplazamiento individual del módulo de acople (módulos esclavo del 2 hasta el 15) para llevarlos a la rueda deseada del módulo a acoplar.

```
void desplazamiento_modulo()
{
  // Se hallan los cinco puntos de bordeo y distancia con respecto al modulo 1
  if(flag==-2){
    coordenadas_modulo();
    puntos_bordeo();
    flag=-1;
  }
  //se halla el (proximo) punto de bordeo a llegar
  else if(flag==-1){
    coord_ptobordeo();
    flag=0;
  }

 // para hallar el angulo de orientacion al punto deseado
 if(flag==0){
   coordenadas_modulo();
   orientacion_modulo();
   //una vez se halla el punto deseado se hace ir al punto deseado con las demas banderas
   if(alfa==180 || alfa==0){flag=3;printf("listo 1a: calculo giro corte en x\n");}
   else {flag=1;printf("listo 1: calculo del angulo de orientacion hacia el punto deseado\n");}
 }
 //para girar hacia al punto deseado
 else if(flag==1){
   coordenadas_modulo();
   if(dist<=1){flag=3;}
   else if((alfa-angulo)==0){
     flag=2;printf("listo 2: giro hacia el punto deseado\n");
   }
   else if((alfa-angulo)>0){
     if(abs(alfa-angulo)<=180){(*states[3].func) ();}
     else {(*states[2].func) ();}
   }
   else{
     if(abs(alfa-angulo)<=180){(*states[2].func) ();}
     else {(*states[3].func) ();}
   }
 }

 //para ir de frente hacia el punto deseado con corte en el eje z
 else if(flag==2){
   coordenadas_modulo();
   if ((alfa-angulo)==0){
     if(zf!=posxz[1] && dist>1){(*states[0].func) ();}
     else {flag=3;printf("listo 3: llegada punto de corte eje z\n");}
   }
   else {
     flag=0;
   }
 }
 else if(flag==3){
   coordenadas_modulo();
   if (dist<=1){printf("listo 6: llegada a punto final\n");flag=6;}
   else if(xf>posxz[0]){alfa=0;flag=4;}
   else {alfa=180;flag=4;}
 }
```

```
  //para girar hacia al punto deseado
  else if(flag==4){
    coordenadas_modulo();
    if(dist<=1){flag=3;}
    else if((alfa-angulo)==0){
      flag=5;printf("listo 4: giro para corte en eje x\n");
    }
    else if((alfa-angulo)>0){
      if(abs(alfa-angulo)<=180){(*states[3].func) ();}
      else {(*states[2].func) ();}
    }
    else{
      if(abs(alfa-angulo)<=180){(*states[2].func) ();}
      else {(*states[3].func) ();}
    }
  }

  //para ir de frente hacia el punto deseado con corte en el eje x
  else if(flag==5){
    coordenadas_modulo();
    if ((alfa-angulo)==0){
      if(xf!=posxz[0] && dist>1){(*states[0].func) ();}
      else {flag=0;printf("listo 5: llegada a punto de corte eje x\n");}
    }
    else {
      flag=0;
    }
  }

//se escoge y se calcula la orientacion hacia el proximo punto
else if(flag==6){
  orientacion_ptobordeo();
  flag=7;
}
//para girar hacia la orientacion final y escoger el proximo punto de bordeo a llegar hasta llegar al ultimo
else if(flag==7){
  coordenadas_modulo();
  if(dist>5){flag=0;}
  else if((orientacion_final-angulo)==0){
    proximo_ptobordeo();////// funcion que escoge el proximo punto de bordeo
  }
  else if((orientacion_final-angulo)>0){
    if(abs(orientacion_final-angulo)<=180){(*states[3].func) ();}
    else {(*states[2].func) ();}
  }
  else{
    if(abs(orientacion_final-angulo)<=180){(*states[2].func) ();}
    else {(*states[3].func) ();}
  }
}

  //para acercarse al otro modulo
  else if(flag==8){
    if(pasos_acercamiento<16){(*states[0].func) ();pasos_acercamiento++;}
    else{pasos_acercamiento=0;printf("listo 9: acercamiento al modulo para conexion\n");flag=9;}
  }
  //para comunicarse con el otro modulo (modulo 1)
  else if(flag==9){comunicacion_modulos();wb_connector_lock(conector_pivote);flag=10;printf("listo 10: envio de datos al modulo 1\n");}
}
```

```
void puntos_bordeo()
{
  //puntos de bordeo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]);
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]);
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]);
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]);
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]);
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]);
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]);
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]);
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]);
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]);
  p6xz[0]=round(129*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]);
  p6xz[1]=round(129*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]);
  p7xz[0]=round(121*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]);
  p7xz[1]=round(121*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]);
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180)));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180)));
  p9xz[0]=round(125*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180)));
  p9xz[1]=round(125*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180)));
  p10xz[0]=round(112*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180)));
  p10xz[1]=round(112*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180)));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180)));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180)));
  p12xz[0]=round(125*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180)));
  p12xz[1]=round(125*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180)));
  p13xz[0]=round(109*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180)));
  p13xz[1]=round(109*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180)));

  //distancias
  dist5p[0]=sqrt(pow(p1xz[1]-posxz[1],2)+pow(p1xz[0]-posxz[0],2));
  dist5p[1]=sqrt(pow(p2xz[1]-posxz[1],2)+pow(p2xz[0]-posxz[0],2));
  dist5p[2]=sqrt(pow(p3xz[1]-posxz[1],2)+pow(p3xz[0]-posxz[0],2));
  dist5p[3]=sqrt(pow(p4xz[1]-posxz[1],2)+pow(p4xz[0]-posxz[0],2));
  dist5p[4]=sqrt(pow(p5xz[1]-posxz[1],2)+pow(p5xz[0]-posxz[0],2));
  dist5p[5]=sqrt(pow(p8xz[1]-posxz[1],2)+pow(p8xz[0]-posxz[0],2));
  dist5p[6]=sqrt(pow(p11xz[1]-posxz[1],2)+pow(p11xz[0]-posxz[0],2));

  printf("P1X:%d  , P1Z:%d  , Dist:%lf\n",p1xz[0], p1xz[1],dist5p[0]);
  printf("P2X:%d  , P2Z:%d  , Dist:%lf\n",p2xz[0], p2xz[1],dist5p[1]);
  printf("P3X:%d  , P3Z:%d  , Dist:%lf\n",p3xz[0], p3xz[1],dist5p[2]);
  printf("P4X:%d  , P4Z:%d  , Dist:%lf\n",p4xz[0], p4xz[1],dist5p[3]);
  printf("P5X:%d  , P5Z:%d  , Dist:%lf\n",p5xz[0], p5xz[1],dist5p[4]);
  printf("P8X:%d  , P8Z:%d  , Dist:%lf\n",p8xz[0], p8xz[1],dist5p[5]);
  printf("P11X:%d , P11Z:%d , Dist:%lf\n",p11xz[0], p11xz[1],dist5p[6]);

  menor=dist5p[0];puntomin=1;
  for(i=0;i<7;i++){
    if(dist5p[i]<menor){
      menor=dist5p[i];
      if(i==5){puntomin=8;}
      else if(i==6){puntomin=11;}
      else {puntomin=i+1;}
    }
  }
  printf("El puntomin es:%d, Dist:%lf\n",puntomin,menor);
}
```

```
void coord_ptobordeo()
{
  switch (puntomin){
    case 1:
      xf=p1xz[0];
      zf=p1xz[1];
      break;
    case 2:
      xf=p2xz[0];
      zf=p2xz[1];
      break;
    case 3:
      xf=p3xz[0];
      zf=p3xz[1];
      break;
    case 4:
      xf=p4xz[0];
      zf=p4xz[1];
      break;
    case 5:
      xf=p5xz[0];
      zf=p5xz[1];
      break;
    case 6:
      xf=p6xz[0];
      zf=p6xz[1];
      break;
    case 7:
      xf=p7xz[0];
      zf=p7xz[1];
      break;
    case 8:
      xf=p8xz[0];
      zf=p8xz[1];
      break;
    case 9:
      xf=p9xz[0];
      zf=p9xz[1];
      break;
    case 10:
      xf=p10xz[0];
      zf=p10xz[1];
      break;
    case 11:
      xf=p11xz[0];
      zf=p11xz[1];
      break;
    case 12:
      xf=p12xz[0];
      zf=p12xz[1];
      break;

    case 13:
      xf=p13xz[0];
      zf=p13xz[1];
      break;
  }
  printf("Puntomin:%d, X: %d, Z:%d, Ori:%d\n",puntomin,xf, zf,orientacion_final);
}

void orientacion_ptobordeo()
{
  if(datosderecibo[4]==1){
    if(puntomin==7){orientacion_final=datosderecibo[3];printf("listo 6.5: calculo de orientacion final\n");}
    else if(puntomin==6){orientacion_final=datosderecibo[3];printf("listo 6.4: calculo de orientacion final\n");}
    else if(puntomin==5){orientacion_final=datosderecibo[3];printf("listo 6.3: calculo de orientacion final\n");}
    else if(puntomin==4){orientacion_final=datosderecibo[3]-90;printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==1){orientacion_final=datosderecibo[3]+90;printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==2 || puntomin==8){orientacion_final=datosderecibo[3]-180;printf("listo 6.1: calculo de orientacion final\n");}
    else if(puntomin==3 || puntomin==11){orientacion_final=datosderecibo[3]-180;printf("listo 6.1: calculo de orientacion final\n");}
  }
  if(datosderecibo[4]==2){
    if(puntomin==10){orientacion_final=datosderecibo[3]+90;printf("listo 6.5: calculo de orientacion final\n");}
    else if(puntomin==9){orientacion_final=datosderecibo[3]+90;printf("listo 6.4: calculo de orientacion final\n");}
    else if(puntomin==8){orientacion_final=datosderecibo[3]+90;printf("listo 6.3: calculo de orientacion final\n");}
    else if(puntomin==2){orientacion_final=datosderecibo[3]-180;printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==1){orientacion_final=datosderecibo[3];printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==4 || puntomin==5){orientacion_final=datosderecibo[3]-90;printf("listo 6.1: calculo de orientacion final\n");}
    else if(puntomin==3 || puntomin==11){orientacion_final=datosderecibo[3]-180;printf("listo 6.1: calculo de orientacion final\n");}
  }
  if(datosderecibo[4]==3){
    if(puntomin==13){orientacion_final=datosderecibo[3]-90;printf("listo 6.5: calculo de orientacion final\n");}
    else if(puntomin==12){orientacion_final=datosderecibo[3]-90;printf("listo 6.4: calculo de orientacion final\n");}
    else if(puntomin==11){orientacion_final=datosderecibo[3]-90;printf("listo 6.3: calculo de orientacion final\n");}
    else if(puntomin==3){orientacion_final=datosderecibo[3]-180;printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==4){orientacion_final=datosderecibo[3];printf("listo 6.2: calculo de orientacion final\n");}
    else if(puntomin==1 || puntomin==5){orientacion_final=datosderecibo[3]+90;printf("listo 6.1: calculo de orientacion final\n");}
    else if(puntomin==2 || puntomin==8){orientacion_final=datosderecibo[3]-180;printf("listo 6.1: calculo de orientacion final\n");}
  }
  //para volver positiva la orientación final
  if(orientacion_final==360){orientacion_final=0;}
  if(orientacion_final<0){orientacion_final=360+orientacion_final;}
  else if(orientacion_final>360){orientacion_final=orientacion_final-360;}
  printf("orientacion_final:%d,\n",orientacion_final);
}
```

```
void proximo_ptobordeo()
{
  if(datosderecibo[4]==1){
    if(puntomin==7){flag=8;printf("listo 8: giro hacia la orientacion final\n");}
    else if(puntomin==6){puntomin=7;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==5){puntomin=6;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==4){puntomin=5;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==1){puntomin=5;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==2 || puntomin==8){puntomin=1;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==3 || puntomin==11){puntomin=4;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
  }
  if(datosderecibo[4]==2){
    if(puntomin==10){flag=8;printf("listo 8: giro hacia la orientacion final\n");}
    else if(puntomin==9){puntomin=10;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==8){puntomin=9;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==2){puntomin=8;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==1){puntomin=8;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==4 || puntomin==5){puntomin=1;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==3 || puntomin==11){puntomin=4;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
  }
  if(datosderecibo[4]==3){
    if(puntomin==13){flag=8;printf("listo 8: giro hacia la orientacion final\n");}
    else if(puntomin==12){puntomin=13;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==11){puntomin=12;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==3){puntomin=11;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==4){puntomin=11;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==1 || puntomin==5){puntomin=4;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
    else if(puntomin==2 || puntomin==8){puntomin=1;flag=-1;printf("listo 7: giro hacia la orientacion final\n");}
  }
}
```

Anexo 7. Secuencia de programación para realizar la lectura y envío de información del módulo 1 (módulo maestro) para la comunicación entre módulos

```
void recepcion_informacion_modulo()
{
  /* is there at least one packet in the receiver's queue ? */
  if (wb_receiver_get_queue_length(receptor) > 0) {
    /* read current packet's data */
    const int *buffer = wb_receiver_get_data(receptor);
    if (message_printed != 1) {
      /* print null-terminated message */
      printf("Recibido: %d, %d, %d, %d, %d, %d\n",*(buffer+0), *(buffer+1),*(buffer+2),*(buffer+3),*(buffer+4),*(buffer+5));
      for (info_paq=0;info_paq<6;info_paq++){
        datosderecibo[info_paq]=*(buffer+info_paq);
        datosmodulos[*(buffer+0)][info_paq]=datosderecibo[info_paq];
      }
      message_printed = 1;
    }
    /* fetch next packet */
    wb_receiver_next_packet(receptor);
    for(fila=0;fila<16;fila++){
      for(columna=0;columna<6;columna++){
        printf("%d, ",datosmodulos[fila][columna]);
      }
      printf("\n");
    }
  }
  else {
    if (message_printed != 2) {
      printf("Communication broken !\n");
      message_printed = 2;
    }
  }
}

void comunicacion_modulos()
{
  int datosdenvio[6] = {modulo_accion,datosmodulos[modulo_conectar][1],datosmodulos[modulo_conectar][2],datosmodulos[modulo_conectar][3],com_forma,arquitectura};
  const int *message = &datosdenvio[0];
  wb_emitter_send(emisor, message, 6 * sizeof(int));
  printf("listo envio de datos al modulo %d\n",modulo_accion);
}
```

Anexo 8. Secuencia de programación para realizar la lectura y envío de información de los módulos 2 al 15 (módulos esclavos) para la comunicación entre módulos

```
void recepcion_informacion_modulo()
{
   /* is there at least one packet in the receiver's queue ? */
  if (wb_receiver_get_queue_length(receptor) > 0) {
    /* read current packet's data */
    const int *buffer = wb_receiver_get_data(receptor);
    if (message_printed != 1) {
      /* print null-terminated message */
      printf("Recibido: %d, %d, %d, %d, %d, %d\n",*(buffer+0), *(buffer+1),*(buffer+2),*(buffer+3),*(buffer+4),*(buffer+5));
      for (i=0;i<6;i++){
        datosderecibo[i]=*(buffer+i);
      }
      message_printed = 1;
    }
    /* fetch next packet */
    wb_receiver_next_packet(receptor);
  }
  else {
    if (message_printed != 2) {
      printf("Communication broken !\n");
      message_printed = 2;
    }
  }
}
void comunicacion_modulos()
{
  int datosdenvio[6] = {id,posxz[0], posxz[1],angulo,datosderecibo[4],datosderecibo[5]};
  const int *message = &datosdenvio[0];
  wb_emitter_send(emisor, message, 6 * sizeof(int));
}
```

Anexo 9. Secuencia en el módulo 1 (maestro) para coordinar la estrategia de armado de la arquitectura oruga con los módulos esclavos del 2 al 15

```
void arquitectura_oruga(){
  com_forma=1;
  //para comunicarse con el otro modulo CON EL 2
  if(flag==7){modulo_accion=2;modulo_conectar=1;comunicacion_modulos();flag=8;}
  //para comunicarse con el modulo 3
  if(flag==8 && datosderecibo[0]==2){modulo_accion=3;modulo_conectar=2;comunicacion_modulos();flag=9;}
  //para comunicarse con el modulo 4
  if(flag==9 && datosderecibo[0]==3){modulo_accion=4;modulo_conectar=3;comunicacion_modulos();flag=10;}
  //para comunicarse con el modulo 5
  if(flag==10 && datosderecibo[0]==4){modulo_accion=5;modulo_conectar=4;comunicacion_modulos();flag=11;}
  //para comunicarse con el modulo 6
  if(flag==11 && datosderecibo[0]==5){modulo_accion=6;modulo_conectar=5;comunicacion_modulos();flag=12;}
  //para comunicarse con el modulo 7
  if(flag==12 && datosderecibo[0]==6){modulo_accion=7;modulo_conectar=6;comunicacion_modulos();flag=13;}
  //para comunicarse con el modulo 8
  if(flag==13 && datosderecibo[0]==7){modulo_accion=8;modulo_conectar=7;comunicacion_modulos();flag=14;}
  //para comunicarse con el modulo 9
  if(flag==14 && datosderecibo[0]==8){modulo_accion=9;modulo_conectar=8;comunicacion_modulos();flag=15;}
  //para comunicarse con el modulo 10
  if(flag==15 && datosderecibo[0]==9){modulo_accion=10;modulo_conectar=9;comunicacion_modulos();flag=16;}
  //para comunicarse con el modulo 11
  if(flag==16 && datosderecibo[0]==10){modulo_accion=11;modulo_conectar=10;comunicacion_modulos();flag=17;}
  //para comunicarse con el modulo 12
  if(flag==17 && datosderecibo[0]==11){modulo_accion=12;modulo_conectar=11;comunicacion_modulos();flag=18;}
  //para comunicarse con el modulo 13
  if(flag==18 && datosderecibo[0]==12){modulo_accion=13;modulo_conectar=12;comunicacion_modulos();flag=19;}
  //para comunicarse con el modulo 14
  if(flag==19 && datosderecibo[0]==13){modulo_accion=14;modulo_conectar=13;comunicacion_modulos();flag=20;}
  //para comunicarse con el modulo 15
  if(flag==20 && datosderecibo[0]==14){modulo_accion=15;modulo_conectar=14;comunicacion_modulos();flag=21;}
  //se espera que el modulo 15 se conecte
  if(flag==21 && datosderecibo[0]==15 && message_printed == 1){final_morfologia=1;}
}
```

Anexo 10. Secuencia de programación adicionada en la función "puntos_bordeo" para los módulos 4 hasta el 15 con el fin de ubicar los nuevos puntos de bordeo en la conformación de la arquitectura hexápodo

- Para el módulo 4:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+310*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+310*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+310*cos(datosderecibo[3]*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+310*sin(datosderecibo[3]*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180))+240*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180))+240*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180)));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180)));
}
```

- Para el módulo 5:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+130*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+130*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+320*cos(datosderecibo[3]*PI/180)+130*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+320*sin(datosderecibo[3]*PI/180)+130*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+320*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+320*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180))+130*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180))+130*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180))+240*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180))+240*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 6:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-150*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-150*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-150*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-150*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-150*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-150*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 7:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-150*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-150*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-150*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-150*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-150*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-150*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-3*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-3*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 8:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+110*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+110*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+310*cos(datosderecibo[3]*PI/180)+110*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+310*sin(datosderecibo[3]*PI/180)+110*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+310*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+310*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180))+110*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180))+110*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180))+240*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180))+240*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 9:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+320*cos(datosderecibo[3]*PI/180)+240*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+320*sin(datosderecibo[3]*PI/180)+240*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+320*cos(datosderecibo[3]*PI/180)+120*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+320*sin(datosderecibo[3]*PI/180)+120*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180)+120*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180)+120*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-110*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-110*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180))+240*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180))+240*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180))+120*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180))+120*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 10:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+120*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+120*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+120*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+120*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-240*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-240*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 11:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+120*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+120*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+120*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+120*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-240*cos(datosderecibo[3]*PI/180)+210*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-240*sin(datosderecibo[3]*PI/180)+210*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-240*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-240*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-1*cos(datosderecibo[3]*PI/180))+210*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-1*sin(datosderecibo[3]*PI/180))+210*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 12:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]);
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]);
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+300*cos(datosderecibo[3]*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+300*sin(datosderecibo[3]*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+300*cos(datosderecibo[3]*PI/180)+320*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+300*sin(datosderecibo[3]*PI/180)+320*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]+320*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]+320*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]);
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]);
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180)));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180)));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180))+320*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180))+320*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 13:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]+320*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]+320*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]+190*cos(datosderecibo[3]*PI/180)+320*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]+190*sin(datosderecibo[3]*PI/180)+320*sin((datosderecibo[3]-90)*PI/180));
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+190*cos(datosderecibo[3]*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+190*sin(datosderecibo[3]*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]);
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]);
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]);
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]);
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180))+320*cos((datosderecibo[3]-90)*PI/180));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180))+320*sin((datosderecibo[3]-90)*PI/180));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180)));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180)));
}
```

- Para el módulo 14:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-340*cos(datosderecibo[3]*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-340*sin(datosderecibo[3]*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]);
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]);
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]+120*cos((datosderecibo[3]+90)*PI/180));
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]+120*sin((datosderecibo[3]+90)*PI/180));
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-340*cos(datosderecibo[3]*PI/180)+120*cos((datosderecibo[3]+90)*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-340*sin(datosderecibo[3]*PI/180)+120*sin((datosderecibo[3]+90)*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-340*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-340*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-7*cos(datosderecibo[3]*PI/180)));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-7*sin(datosderecibo[3]*PI/180)));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-1*cos(datosderecibo[3]*PI/180))+120*cos((datosderecibo[3]+90)*PI/180));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-1*sin(datosderecibo[3]*PI/180))+120*sin((datosderecibo[3]+90)*PI/180));
}
```

- Para el módulo 15:

```
if(datosderecibo[5]==2){//para conectar en Hexapodo
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-340*cos(datosderecibo[3]*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-340*sin(datosderecibo[3]*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-45)*PI/180)+datosderecibo[1]);
  p2xz[1]=round(200*sin((datosderecibo[3]-45)*PI/180)+datosderecibo[2]);
  p3xz[0]=round(200*cos((datosderecibo[3]+45)*PI/180)+datosderecibo[1]);
  p3xz[1]=round(200*sin((datosderecibo[3]+45)*PI/180)+datosderecibo[2]);
  p4xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-340*cos(datosderecibo[3]*PI/180));
  p4xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-340*sin(datosderecibo[3]*PI/180));
  //rueda posterior
  p5xz[0]=round(141.4*cos((datosderecibo[3]-180)*PI/180)+datosderecibo[1]-340*cos(datosderecibo[3]*PI/180));
  p5xz[1]=round(141.4*sin((datosderecibo[3]-180)*PI/180)+datosderecibo[2]-340*sin(datosderecibo[3]*PI/180));
  //rueda izquierda
  p8xz[0]=round(141.4*cos((datosderecibo[3]-90)*PI/180)+(datosderecibo[1]-5*cos(datosderecibo[3]*PI/180)));
  p8xz[1]=round(141.4*sin((datosderecibo[3]-90)*PI/180)+(datosderecibo[2]-5*sin(datosderecibo[3]*PI/180)));
  //rueda derecha
  p11xz[0]=round(141.4*cos((datosderecibo[3]+90)*PI/180)+(datosderecibo[1]-2*cos(datosderecibo[3]*PI/180)));
  p11xz[1]=round(141.4*sin((datosderecibo[3]+90)*PI/180)+(datosderecibo[2]-2*sin(datosderecibo[3]*PI/180)));
}
```

Anexo 11. Secuencia de programación del módulo 1 (maestro) para coordinar la estrategia de armado de la arquitectura hexápodo con los demás módulos del 2 al 15 (esclavos)

```
void arquitectura_hexapodo(){
  //para comunicarse con el otro modulo CON EL 2
  if(flag==7){modulo_accion=2;modulo_conectar=1;com_forma=1;comunicacion_modulos();flag=8;}
  //para comunicarse con el modulo 3
  if(flag==8 && datosderecibo[0]==2){modulo_accion=3;modulo_conectar=2;com_forma=1;comunicacion_modulos();flag=9;}
  //para comunicarse con el modulo 4
  if(flag==9 && datosderecibo[0]==3){modulo_accion=4;modulo_conectar=3;com_forma=1;comunicacion_modulos();flag=10;}
  //para levantar el modulo 1 -girando el pivote del modulo 2
  if(flag==10 && message_printed == 1){
    if(datosderecibo[0]==4){modulo_accion=2;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=11;printf("listo 12: envio de datos al modulo 2 para levantar el modulo 1\n");}
  }
  //para girar las ruedas del modulo 1 y alinear ejes despues para bajar el modulo 1
  if(flag==11 && datosderecibo[0]==2 && datosderecibo[4]==5 && message_printed == 1){
    alineacion_ejes_ruedasizqder();
    //para bajar el modulo 1
    modulo_accion=2;modulo_conectar=0;com_forma=6;comunicacion_modulos();
    flag=12;printf("listo 13: envio de datos al modulo 2 para bajar el modulo 1\n");
  }
  //para comunicarse con el modulo 15
  if(flag==12 && datosderecibo[0]==2 && datosderecibo[4]==6 && message_printed == 1){
    //para saber las nuevas coordenadas del modulo 1
    coordenadas_modulo();registrar_datos_modulo1();
    modulo_accion=15;modulo_conectar=1;com_forma=3;comunicacion_modulos();flag=13;
  }
  //para comunicarse con el modulo 14
  if(flag==13 && datosderecibo[0]==15){modulo_accion=14;modulo_conectar=1;com_forma=2;comunicacion_modulos();flag=14;}
  //para levantar el modulo 1 -girando el pivote del modulo 2
  if(flag==14 && message_printed == 1){
    if(datosderecibo[0]==14){condicion_giro=0;modulo_accion=2;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=15;
      printf("listo 12: envio de datos al modulo 2 para levantar el modulo 1\n");}
  }

 //para girar las ruedas del modulo 1 y voltear los modulos 14 y 15. despues para bajar el modulo 1
 if(flag==15 && datosderecibo[0]==2 && datosderecibo[4]==5 && message_printed == 1){
   volteo_ruedasizqder();
   //para bajar el modulo 1
   modulo_accion=2;modulo_conectar=0;com_forma=6;comunicacion_modulos();
   flag=16;printf("listo 13: envio de datos al modulo 2 para bajar el modulo 1\n");
 }
 //para conocer las nuevas coordenadas del modulo 15
 if(flag==16 && message_printed == 1){
   if(datosderecibo[0]==2 && datosderecibo[4]==6){modulo_accion=15;modulo_conectar=15;com_forma=0;comunicacion_modulos();flag=17;
     printf("listo 14: envio de datos al modulo 15 para conocer nuevas coordenadas\n");}
 }
 //para comunicarse con el modulo 13
 if(flag==17 && datosderecibo[0]==15){modulo_accion=13;modulo_conectar=15;com_forma=1;comunicacion_modulos();flag=18;}
 //para conocer las nuevas coordenadas del modulo 14
 if(flag==18 && datosderecibo[0]==13){
   modulo_accion=14;modulo_conectar=14;com_forma=0;comunicacion_modulos();flag=19;printf("listo 15: envio de datos al modulo 14 para conocer nuevas coordenadas\n");
 }
 //para comunicarse con el modulo 12
 if(flag==19 && datosderecibo[0]==14){modulo_accion=12;modulo_conectar=14;com_forma=1;comunicacion_modulos();flag=20;}
 //para despegar el modulo 4
 if(flag==20 && datosderecibo[0]==12){modulo_accion=4;modulo_conectar=0;com_forma=9;comunicacion_modulos();flag=21;}
```

```
//para despegar el modulo 3
if(flag==21 && datosderecibo[0]==4){modulo_accion=3;modulo_conectar=0;com_forma=9;comunicacion_modulos();flag=22;}
//para levantar el modulo 2 -girando el pivote del modulo 2
if(flag==22 && message_printed == 1){
  if(datosderecibo[0]==3){modulo_accion=2;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=23;printf("listo 16: envio de datos al modulo 2 para levantar el modulo 2\n");}
}
//para girar las ruedas del modulo 2 y alinear ejes despues para bajar el modulo 2
if(flag==23 && datosderecibo[0]==2 && datosderecibo[4]==5){
  modulo_accion=2;modulo_conectar=0;com_forma=7;comunicacion_modulos();flag=24;printf("listo 17: envio de datos al modulo 2 para alinear ejes\n");
}

//para bajar el modulo 2 -girando el pivote del modulo 2
if(flag==24 && datosderecibo[4]==7){
  if(datosderecibo[0]==2){modulo_accion=2;modulo_conectar=0;com_forma=6;comunicacion_modulos();flag=25;printf("listo 18: envio de datos al modulo 2 para bajar el modulo 2\n");}
}
//para conocer las nuevas coordenadas del modulo 2
if(flag==25 && datosderecibo[0]==2 && datosderecibo[4]==6){
  modulo_accion=2;modulo_conectar=0;com_forma=0;comunicacion_modulos();flag=26;printf("listo 19: envio de datos al modulo 2 para conocer nuevas coordenadas\n");
}
//para comunicarse con el modulo 11
if(flag==26 && datosderecibo[0]==2 && datosderecibo[4]==0){modulo_accion=11;modulo_conectar=2;com_forma=3;comunicacion_modulos();flag=27;}
//para comunicarse con el modulo 10
if(flag==27 && datosderecibo[0]==11){modulo_accion=10;modulo_conectar=2;com_forma=2;comunicacion_modulos();flag=28;}
//para levantar el modulo 2 -girando el pivote del modulo 2
if(flag==28 && message_printed == 1){
  if(datosderecibo[0]==10){modulo_accion=2;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=29;printf("listo 20: envio de datos al modulo 2 para levantar el modulo 2\n");}
}
//para girar las ruedas del modulo 2 y voltear los modulos 10 y 11 despues para bajar el modulo 2
if(flag==29 && datosderecibo[0]==2 && datosderecibo[4]==5){
  modulo_accion=2;modulo_conectar=0;com_forma=8;comunicacion_modulos();flag=30;printf("listo 21: envio de datos al modulo 2 para voltear modulos ejes\n");
}
//para bajar el modulo 2 -girando el pivote del modulo 2
if(flag==30 && datosderecibo[4]==8){
  if(datosderecibo[0]==2){modulo_accion=2;modulo_conectar=0;com_forma=6;comunicacion_modulos();flag=31;printf("listo 22: envio de datos al modulo 2 para bajar el modulo 2\n");}
}
//para conocer las nuevas coordenadas del modulo 11
if(flag==31 && datosderecibo[0]==2 && datosderecibo[4]==6){
  modulo_accion=11;modulo_conectar=11;com_forma=0;comunicacion_modulos();flag=32;printf("listo 23: envio de datos al modulo 11 para conocer nuevas coordenadas\n");
}
//para comunicarse con el modulo 9
if(flag==32 && datosderecibo[0]==11){modulo_accion=9;modulo_conectar=11;com_forma=1;comunicacion_modulos();flag=33;}
//para conocer las nuevas coordenadas del modulo 10
if(flag==33 && datosderecibo[0]==9){
  modulo_accion=10;modulo_conectar=10;com_forma=0;comunicacion_modulos();flag=34;printf("listo 24: envio de datos al modulo 10 para conocer nuevas coordenadas\n");
}

//para comunicarse con el modulo 8
if(flag==34 && datosderecibo[0]==10){modulo_accion=8;modulo_conectar=10;com_forma=1;comunicacion_modulos();flag=35;}
//para conocer las nuevas coordenadas del modulo 2
if(flag==35 && datosderecibo[0]==8){
  modulo_accion=2;modulo_conectar=0;com_forma=0;comunicacion_modulos();flag=36;printf("listo 25: envio de datos al modulo 2 para conocer nuevas coordenadas\n");
}
//para comunicarse con el modulo 3
if(flag==36 && datosderecibo[0]==2){modulo_accion=3;modulo_conectar=2;com_forma=1;comunicacion_modulos();flag=37;}
//para levantar el modulo 3 -girando el pivote del modulo 3
if(flag==37 && datosderecibo[0]==3){modulo_accion=3;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=38;
  printf("listo 26: envio de datos al modulo 3 para levantar el modulo 3\n");}
```

```
  //para girar las ruedas del modulo 3 y alinear ejes despues para bajar el modulo 3
  if(flag==38 && datosderecibo[0]==3 && datosderecibo[4]==5 && message_printed == 1){
    modulo_accion=3;modulo_conectar=0;com_forma=7;comunicacion_modulos();flag=39;printf("listo 27: envio de datos al modulo 3 para alinear ejes\n");
  }
  //para bajar el modulo 3 -girando el pivote del modulo 3
  if(flag==39 && datosderecibo[4]==7 && datosderecibo[0]==3){modulo_accion=3;modulo_conectar=0;com_forma=6;comunicacion_modulos();flag=40;
    printf("listo 28: envio de datos al modulo 3 para bajar el modulo 3\n");}
  //para conocer las nuevas coordenadas del modulo 3
  if(flag==40 && datosderecibo[0]==3 && datosderecibo[4]==6){
    modulo_accion=3;modulo_conectar=0;com_forma=0;comunicacion_modulos();flag=41;printf("listo 29: envio de datos al modulo 3 para conocer nuevas coordenadas\n");
  }
  //para comunicarse con el modulo 7
  if(flag==41 && datosderecibo[0]==3 && datosderecibo[4]==0){modulo_accion=7;modulo_conectar=3;com_forma=3;comunicacion_modulos();flag=42;}
  //para comunicarse con el modulo 6
  if(flag==42 && datosderecibo[0]==7){modulo_accion=6;modulo_conectar=3;com_forma=2;comunicacion_modulos();flag=43;}
  //para levantar el modulo 3 -girando el pivote del modulo 3
  if(flag==43 && message_printed == 1){
    if(datosderecibo[0]==6){modulo_accion=3;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=44;printf("listo 30: envio de datos al modulo 3 para levantar el modulo 3\n");}
  }
  //para girar las ruedas del modulo 3 y voltear los modulos 6 y 7 despues para bajar el modulo 3
  if(flag==44 && datosderecibo[0]==3 && datosderecibo[4]==5 && message_printed == 1){
    modulo_accion=3;modulo_conectar=0;com_forma=8;comunicacion_modulos();flag=45;printf("listo 31: envio de datos al modulo 3 para voltear modulos ejes\n");
  }

  //para bajar el modulo 3 -girando el pivote del modulo 3
  if(flag==45 && datosderecibo[4]==8){
    if(datosderecibo[0]==3){modulo_accion=3;modulo_conectar=0;com_forma=6;comunicacion_modulos();flag=46;printf("listo 32: envio de datos al modulo 3 para bajar el modulo 3\n");}
  }
  //para conocer las nuevas coordenadas del modulo 7
  if(flag==46 && datosderecibo[0]==3 && datosderecibo[4]==6){
    modulo_accion=7;modulo_conectar=7;com_forma=0;comunicacion_modulos();flag=47;printf("listo 33: envio de datos al modulo 7 para conocer nuevas coordenadas\n");
  }
  //para comunicarse con el modulo 5
  if(flag==47 && datosderecibo[0]==7){modulo_accion=5;modulo_conectar=7;com_forma=1;comunicacion_modulos();flag=48;}
  //para conocer las nuevas coordenadas del modulo 6
  if(flag==48 && datosderecibo[0]==5){
    modulo_accion=6;modulo_conectar=6;com_forma=0;comunicacion_modulos();flag=49;printf("listo 34: envio de datos al modulo 6 para conocer nuevas coordenadas\n");
  }
  //para comunicarse con el modulo 4
  if(flag==49 && datosderecibo[0]==6){modulo_accion=4;modulo_conectar=6;com_forma=1;comunicacion_modulos();flag=50;}
  //para comunicarse con los modulos 4, 5, 8, 9, 12 y 13 para levantar el hexapodo (posición inicial)
  if(flag==50 && datosderecibo[0]==4){modulo_accion=1;modulo_conectar=0;com_forma=10;comunicacion_modulos();flag=51;final_morfologia=2;}
}
```

Anexo 12. Secuencia de programación del módulo 1 (maestro) para coordinar la estrategia de desarmado de la arquitectura oruga con los demás módulos del 4 al 15 (esclavos)

```
void desacople_oruga()
{
  //para comunicarse con el modulo 4
  if(flag==54){modulo_accion=4;modulo_conectar=0;com_forma=0;comunicacion_modulos();flag=55;}
  //para comunicarse con el modulo 15
  if(flag==55 && datosderecibo[0]==4){modulo_accion=15;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=56;}
  //para comunicarse con el modulo 14
  if(flag==56 && datosderecibo[0]==15){modulo_accion=14;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=57;}
  //para comunicarse con el modulo 13
  if(flag==57 && datosderecibo[0]==14){modulo_accion=13;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=58;}
  //para comunicarse con el modulo 12
  if(flag==58 && datosderecibo[0]==13){modulo_accion=12;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=59;}
  //para comunicarse con el modulo 11
  if(flag==59 && datosderecibo[0]==12){modulo_accion=11;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=60;}
  //para comunicarse con el modulo 10
  if(flag==60 && datosderecibo[0]==11){modulo_accion=10;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=61;}
  //para comunicarse con el modulo 9
  if(flag==61 && datosderecibo[0]==10){modulo_accion=9;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=62;}
  //para comunicarse con el modulo 8
  if(flag==62 && datosderecibo[0]==9){modulo_accion=8;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=63;}
  //para comunicarse con el modulo 7
  if(flag==63 && datosderecibo[0]==8){modulo_accion=7;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=64;}
  //para comunicarse con el modulo 6
  if(flag==64 && datosderecibo[0]==7){modulo_accion=6;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=65;}
  //para comunicarse con el modulo 5
  if(flag==65 && datosderecibo[0]==6){modulo_accion=5;modulo_conectar=4;com_forma=22;comunicacion_modulos();flag=66;}
  //se espera que el modulo 5 se desacople
  if(flag==66 && datosderecibo[0]==5 && message_printed == 1){final_morfologia=3;}
}
```

Anexo 13. Secuencia de programación adicionada en la función "puntos_bordeo" para los módulos 15 hasta el 6 (sentido decreciente) en el desarmado de la arquitectura oruga con el fin de ubicar los nuevos puntos de bordeo para disposición de armado de arquitectura hexápodo

- Para el módulo 15:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-1150*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-1150*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-50*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-50*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 14:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-1020*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-1020*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-50*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-50*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 13:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-910*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-910*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-180*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-180*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 12:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-790*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-790*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-180*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-180*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 11:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-680*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-680*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-310*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-310*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 10:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-570*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-570*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-310*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-310*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 9:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-480*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-480*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-440*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-440*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 8:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-480*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-480*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-440*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-440*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 7:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-600*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-600*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]+135)*PI/180)+datosderecibo[1]-570*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]+90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]+135)*PI/180)+datosderecibo[2]-570*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]+90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

- Para el módulo 6:

```
// puntos de bordeo para hacer el desacople de ORUGA y ubicar el modulo cerca del modulo 4
if(datosderecibo[4]==22){
  p1xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-600*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p1xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-600*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  p2xz[0]=round(200*cos((datosderecibo[3]-135)*PI/180)+datosderecibo[1]-570*cos(datosderecibo[3]*PI/180)+100*cos((datosderecibo[3]-90)*PI/180));
  p2xz[1]=round(200*sin((datosderecibo[3]-135)*PI/180)+datosderecibo[2]-570*sin(datosderecibo[3]*PI/180)+100*sin((datosderecibo[3]-90)*PI/180));
  puntomin=1;
  printf("P1X:%d  , P1Z:%d\n",p1xz[0], p1xz[1]);
  printf("P2X:%d  , P2Z:%d\n",p2xz[0], p2xz[1]);
}
```

Anexo 14. Secuencia de programación del módulo 1 (maestro) para coordinar la estrategia de desarmado de la arquitectura hexápodo con los demás módulos del 2 al 15 (esclavos)

```
void desacople_hexapodo()
{
  //para comunicarse con el modulo 4
  if(flag==54){modulo_accion=4;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=55;}
  //para comunicarse con el modulo 5
  if(flag==55 && datosderecibo[0]==4){modulo_accion=5;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=56;}
  //para levantar el modulo 3 -girando el pivote del modulo 3
  if(flag==56 && datosderecibo[0]==5 && message_printed == 1){modulo_accion=3;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=57;
    printf("listo 30: envio de datos al modulo 3 para levantar el modulo 3\n");}
  //para girar las ruedas del modulo 3 y alinear los modulos 6 y 7 despues para bajar el modulo 3
  if(flag==57 && datosderecibo[0]==3 && datosderecibo[4]==5 && message_printed == 1){modulo_accion=3;modulo_conectar=0;com_forma=7;comunicacion_modulos();flag=58;
    printf("listo 31: envio de datos al modulo 3 para voltear modulos ejes\n");}
  //para bajar el modulo 3 -girando el pivote del modulo 3
  if(flag==58 && datosderecibo[0]==3 && datosderecibo[4]==7 && message_printed == 1){modulo_accion=3;modulo_conectar=0;com_forma=6;comunicacion_modulos();flag=59;
    printf("listo 32: envio de datos al modulo 3 para bajar el modulo 3\n");}
  //para comunicarse con el modulo 6
  if(flag==59 && datosderecibo[0]==3 && datosderecibo[4]==6 && message_printed == 1){modulo_accion=6;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=60;}
  //para comunicarse con el modulo 7
  if(flag==60 && datosderecibo[0]==6){modulo_accion=7;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=61;}
  //para despegar el modulo 3
  if(flag==61 && datosderecibo[0]==7){modulo_accion=3;modulo_conectar=0;com_forma=9;comunicacion_modulos();flag=62;}
  //para comunicarse con el modulo 8
  if(flag==62 && datosderecibo[0]==3){modulo_accion=8;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=63;}
  //para comunicarse con el modulo 9
  if(flag==63 && datosderecibo[0]==8){modulo_accion=9;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=64;}
  //para levantar el modulo 2 -girando el pivote del modulo 2
  if(flag==64 && datosderecibo[0]==9 && message_printed == 1){modulo_accion=2;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=65;
    printf("listo 33: envio de datos al modulo 2 para levantar el modulo 2\n");}
  //para girar las ruedas del modulo 2 y alinear los modulos 10 y 11 despues para bajar el modulo 2
  if(flag==65 && datosderecibo[0]==2 && datosderecibo[4]==5 && message_printed == 1){modulo_accion=2;modulo_conectar=0;com_forma=7;comunicacion_modulos();flag=66;
    printf("listo 34: envio de datos al modulo 2 para voltear modulos ejes\n");}
  //para bajar el modulo 3 -girando el pivote del modulo 3
  if(flag==66 && datosderecibo[0]==2 && datosderecibo[4]==7 && message_printed == 1){modulo_accion=2;modulo_conectar=0;com_forma=6;comunicacion_modulos();flag=67;
    printf("listo 35: envio de datos al modulo 2 para bajar el modulo 2\n");}

  //para comunicarse con el modulo 10
  if(flag==67 && datosderecibo[0]==2 && datosderecibo[4]==6 && message_printed == 1){modulo_accion=10;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=68;}
  //para comunicarse con el modulo 11
  if(flag==68 && datosderecibo[0]==10){modulo_accion=11;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=69;}
  //para comunicarse con el modulo 12
  if(flag==69 && datosderecibo[0]==11){modulo_accion=12;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=70;}
  //para comunicarse con el modulo 13
  if(flag==70 && datosderecibo[0]==12){modulo_accion=13;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=71;}
  //para conocer las nuevas coordenadas del modulo 2
  if(flag==71 && datosderecibo[0]==13){modulo_accion=2;modulo_conectar=0;com_forma=0;comunicacion_modulos();flag=72;
    printf("listo 25: envio de datos al modulo 2 para conocer nuevas coordenadas\n");}
```

```
  //para comunicarse con el modulo 3
  if(flag==72 && datosderecibo[0]==2){modulo_accion=3;modulo_conectar=2;com_forma=1;comunicacion_modulos();flag=73;}
  //para comunicarse con el modulo 4
  if(flag==73 && datosderecibo[0]==3){modulo_accion=4;modulo_conectar=3;com_forma=1;comunicacion_modulos();flag=74;}
  //para levantar el modulo 1 -girando el pivote del modulo 2
  if(flag==74 && datosderecibo[0]==4 && message_printed == 1){condicion_giro=0;modulo_accion=2;modulo_conectar=0;com_forma=5;comunicacion_modulos();flag=75;
    printf("listo 12: envio de datos al modulo 2 para levantar el modulo 1\n");}
  //para girar las ruedas del modulo 1 y alinear ejes despues para bajar el modulo 1
  if(flag==75 && datosderecibo[0]==2 && datosderecibo[4]==5 && message_printed == 1){
    alineacion_ejes_ruedasizqder();
    //para bajar el modulo 1
    modulo_accion=2;modulo_conectar=0;com_forma=6;comunicacion_modulos();
    flag=76;printf("listo 13: envio de datos al modulo 2 para bajar el modulo 1\n");
  }
  //para comunicarse con el modulo 14
  if(flag==76 && datosderecibo[0]==2 && datosderecibo[4]==6 && message_printed == 1){modulo_accion=14;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=77;}
  //para comunicarse con el modulo 15
  if(flag==77 && datosderecibo[0]==14){modulo_accion=15;modulo_conectar=0;com_forma=21;comunicacion_modulos();flag=78;}
  //se espera que el modulo 15 se desacople
  if(flag==78 && datosderecibo[0]==15 && message_printed == 1){final_morfologia=4;}
}
```

Anexo 15. Secuencia de programación del módulo 1 para ejecutar el algoritmo de búsqueda para planificación de trayectorias mediante metodología A* (A-Star)

```
void algoritmo_a_star()
{
  int * open = (int*)calloc(o_len, sizeof(int));

  ////////////OBSTACULOS EN EL PLANO
  cantidad_cubos=5;
  obstaculos = (struct obstaculo *)realloc(obstaculos, 1 * sizeof(struct obstaculo));
  obstaculos[0].obst_coord_n = 10;
  obstaculos[0].obst_coord_m = 20;
  obstaculos[0].obst_ancho = 3;
  obstaculos = (struct obstaculo *)realloc(obstaculos, 2 * sizeof(struct obstaculo));
  obstaculos[1].obst_coord_n = 30;
  obstaculos[1].obst_coord_m = 30;
  obstaculos[1].obst_ancho = 2;
  obstaculos = (struct obstaculo *)realloc(obstaculos, 3 * sizeof(struct obstaculo));
  obstaculos[2].obst_coord_n = 17;
  obstaculos[2].obst_coord_m = 10;
  obstaculos[2].obst_ancho = 1;
  obstaculos = (struct obstaculo *)realloc(obstaculos, 4 * sizeof(struct obstaculo));
  obstaculos[3].obst_coord_n = 17;
  obstaculos[3].obst_coord_m = 29;
  obstaculos[3].obst_ancho = 1;
  obstaculos = (struct obstaculo *)realloc(obstaculos, 5 * sizeof(struct obstaculo));
  obstaculos[4].obst_coord_n = 25;
  obstaculos[4].obst_coord_m = 20;
  obstaculos[4].obst_ancho = 3;


  for (i = 0; i < map_size_rows; i++) {
    for (j = 0; j < map_size_cols; j++) {
      map[i][j]=0;
      for (cub_obs = 0; cub_obs < cantidad_cubos; cub_obs++) {
        mediancho=ceil(obstaculos[cub_obs].obst_ancho/2);
        if(i==0 || j==0 || i==map_size_rows-1 || j==map_size_cols-1 || i==1 || j==1 || i==map_size_rows-2 || j==map_size_cols-2){map[i][j]=1;}
        else if(i>=(obstaculos[cub_obs].obst_coord_n-mediancho-1) && i<=(obstaculos[cub_obs].obst_coord_n+mediancho+1)
        && j>=(obstaculos[cub_obs].obst_coord_m-mediancho-1) && j<=(obstaculos[cub_obs].obst_coord_m+mediancho+1)){map[i][j]=1;}

      }
      //printf("%d, ", map[i][j]);
    }
    //printf("\n");
  }
  // se define el mapa de trabajo
  for (i = 0; i < map_size_rows; i++) {
    for (j = 0; j < map_size_cols; j++) {
      ind[i][j]=-1;
    }
  }
```

```
  //Se definen los nodos de inicio y final
/* index of start stop */
coordenadas_modulo();
coordenadas_a_mapa();
start=0;
fila_inicial=nodo_fila_actual;
columna_inicial=nodo_columna_actual;
inicio=(40*(fila_inicial-1))+(columna_inicial-1);
/* index of finish stop */
cont_final=0;end=0;
fila_final=fila_nodofinal;
columna_final=columna_nodofinal;
fin=(40*(fila_final-1))+(columna_final-1);

//Se definen la cantidad de nodos sin obstaculos
for (i = 0; i < map_size_rows; i++) {
    for (j = 0; j < map_size_cols; j++) {
      ++cont_final;
        if (!map[i][j]) {
          ++s_len;
          stops = (struct stop *)realloc(stops, s_len * sizeof(struct stop));
          int t = s_len - 1;
          stops[t].col = j;
          stops[t].row = i;
          stops[t].from = -1;
          stops[t].g = DBL_MAX;
          stops[t].n_len = 0;
          stops[t].n = NULL;
          ind[i][j] = t;
          end=end+1;
          start=start+1;
        }
        if(cont_final==fin){e=end;}
        if(cont_final==inicio){s=start;}
    }
}

//calculo de h (distancia de cada nodo al nodo objetivo "final")
for (i = 0; i < s_len; i++) {
    stops[i].h = sqrt(pow(stops[e].row - stops[i].row, 2) + pow(stops[e].col - stops[i].col, 2));
}
```

```
for (i = 1; i < map_size_rows - 1; i++) {
    for (j = 1; j < map_size_cols - 1; j++) {
        if (ind[i][j] >= 0) {
            for (k_star = i - 1; k_star <= i + 1; k_star++) {
                for (l_star = j - 1; l_star <= j + 1; l_star++) {
                    if ((k_star == i) && (l_star == j)) {
                        continue;
                    }
                    if (ind[k_star][l_star] >= 0) {
                        ++r_len;
                        routes = (struct route *)realloc(routes, r_len * sizeof(struct route));
                        int t = r_len - 1;
                        routes[t].x = ind[i][j];
                        routes[t].y = ind[k_star][l_star];
                        routes[t].d_star = sqrt(pow(stops[routes[t].y].row - stops[routes[t].x].row, 2) + pow(stops[routes[t].y].col - stops[routes[t].x].col, 2));
                        ++stops[routes[t].x].n_len;
                        stops[routes[t].x].n = (int*)realloc(stops[routes[t].x].n, stops[routes[t].x].n_len * sizeof(int));
                        stops[routes[t].x].n[stops[routes[t].x].n_len - 1] = t;
                    }
                }
            }
        }
    }
}
open[0] = s;
stops[s].g = 0;
stops[s].f = stops[s].g + stops[s].h;
found = 0;

while (o_len && ! found) {
    min = DBL_MAX;

    for (i = 0; i < o_len; i++) {
        if (stops[open[i]].f < min) {
            current = open[i];
            min = stops[open[i]].f;
        }
    }

    if (current == e) {
        found = 1;

        ++p_len;
        path = (int*)realloc(path, p_len * sizeof(int));
        path[p_len - 1] = current;
        while (stops[current].from >= 0) {
            current = stops[current].from;
            ++p_len;
            path = (int*)realloc(path, p_len * sizeof(int));
            path[p_len - 1] = current;
        }

    }
```

```
    for (i = 0; i < o_len; i++) {
        if (open[i] == current) {
            if (i != (o_len - 1)) {
                for (j = i; j < (o_len - 1); j++) {
                    open[j] = open[j + 1];
                }
            }
            --o_len;
            open = (int*)realloc(open, o_len * sizeof(int));
            break;
        }
    }

    ++c_len;
    closed = (int*)realloc(closed, c_len * sizeof(int));
    closed[c_len - 1] = current;

    for (i = 0; i < stops[current].n_len; i++) {
        b = 0;

        for (j = 0; j < c_len; j++) {
            if (routes[stops[current].n[i]].y == closed[j]) {
                b = 1;
            }
        }

        if (b) {
            continue;
        }

        tempg = stops[current].g + routes[stops[current].n[i]].d_star;

        b = 1;
        if (o_len > 0) {
            for (j = 0; j < o_len; j++) {
                if (routes[stops[current].n[i]].y == open[j]) {
                    b = 0;
                }
            }
        }

        if (b || (tempg < stops[routes[stops[current].n[i]].y].g)) {
            stops[routes[stops[current].n[i]].y].from = current;
            stops[routes[stops[current].n[i]].y].g = tempg;
            stops[routes[stops[current].n[i]].y].f = stops[routes[stops[current].n[i]].y].g + stops[routes[stops[current].n[i]].y].h;

            if (b) {
                ++o_len;
                open = (int*)realloc(open, o_len * sizeof(int));
                open[o_len - 1] = routes[stops[current].n[i]].y;
            }
        }
    }
}
```

```
for (i = 0; i < map_size_rows; i++) {
    for (j = 0; j < map_size_cols; j++) {
        if (map[i][j]) {
            putchar('X');
        } else {
            b = 0;
            for (k_star = 0; k_star < p_len; k_star++) {
                if (ind[i][j] == path[k_star]) {
                    ++b;
                }
            }
            if (b) {
                putchar('+');
            } else {
                putchar('.');
            }
        }
    }
    putchar('\n');
}
cont_nodo=0;con_recta=0,contador_nodos_recta=1;t_recta=0;contador_nodos_recta_anterior=0;
if (! found) {
    puts("IMPOSSIBLE");
} else {
    printf("path cost is %d:\n", p_len);
    for (i = p_len - 1; i >= 0; i--) {
      ++cont_nodo;
      info_trayectos = (struct info_trayect *)realloc(info_trayectos, cont_nodo * sizeof(struct info_trayect));
      t_nodo = cont_nodo - 1;
      info_trayectos[t_nodo].fila_nodo=stops[path[i]].row+1;
      info_trayectos[t_nodo].columna_nodo=stops[path[i]].col+1;
      if(t_nodo==0){info_trayectos[t_nodo].pendiente = 360;info_trayectos[t_nodo].recta = 0;}
      else if(t_nodo>0){//calculo de pendiente en angulo
        if(info_trayectos[t_nodo].columna_nodo-info_trayectos[t_nodo-1].columna_nodo==0){
          if((info_trayectos[t_nodo].fila_nodo-info_trayectos[t_nodo-1].fila_nodo)>=0){info_trayectos[t_nodo].pendiente = 90;}
          else{info_trayectos[t_nodo].pendiente = -90;}
        }
        else if(info_trayectos[t_nodo].columna_nodo<info_trayectos[t_nodo-1].columna_nodo){
          if((info_trayectos[t_nodo].fila_nodo-info_trayectos[t_nodo-1].fila_nodo)>0){
            info_trayectos[t_nodo].pendiente = 180+atan((info_trayectos[t_nodo].fila_nodo-info_trayectos[t_nodo-1].fila_nodo)/
            (info_trayectos[t_nodo].columna_nodo-info_trayectos[t_nodo-1].columna_nodo))*180/M_PI;}
          else{info_trayectos[t_nodo].pendiente = -180+atan((info_trayectos[t_nodo].fila_nodo-info_trayectos[t_nodo-1].fila_nodo)/
            (info_trayectos[t_nodo].columna_nodo-info_trayectos[t_nodo-1].columna_nodo))*180/M_PI;}
        }
        else {info_trayectos[t_nodo].pendiente = atan((info_trayectos[t_nodo].fila_nodo-info_trayectos[t_nodo-1].fila_nodo)/
          (info_trayectos[t_nodo].columna_nodo-info_trayectos[t_nodo-1].columna_nodo))*180/M_PI;}
        //calculo de cantidad de rectas que componen la trayectoria
        if(info_trayectos[t_nodo].pendiente!=info_trayectos[t_nodo-1].pendiente || t_nodo==p_len-1){
          info_trayectos[t_nodo].recta=info_trayectos[t_nodo-1].recta+1;
          ++con_recta;
          trayecto_arquitecturas = (struct trayecto_arquitectura *)realloc(trayecto_arquitecturas, con_recta * sizeof(struct trayecto_arquitectura));
          t_recta = con_recta - 1;
          trayecto_arquitecturas[t_recta].numero_linea=t_recta;
          trayecto_arquitecturas[t_recta].cantidad_nodos=contador_nodos_recta;
```

```
//para hallar el nodo en el que inicia la recta
if(t_nodo==1){
  trayecto_arquitecturas[t_recta].nodo_inicia_recta=contador_nodos_recta;
}
else if(t_recta==1){trayecto_arquitecturas[t_recta].nodo_inicia_recta=trayecto_arquitecturas[t_recta-1].nodo_inicia_recta+contador_nodos_recta_anterior-1;}
else{
  trayecto_arquitecturas[t_recta].nodo_inicia_recta=trayecto_arquitecturas[t_recta-1].nodo_inicia_recta+contador_nodos_recta_anterior;
}
//eleccion arquitectura para cada recta 1-oruga 2-hexapodo
if(contador_nodos_recta<=6){trayecto_arquitecturas[t_recta].forma_ejecutar=2;}
else{//busqueda de obstaculos a partir del sentido determinado para evaluar la posibilidad de hacer la oruga
  obstaculo_encontrado=0;
  //se halla la fila y columna del primer nodo de cada recta
  obstac_fila=info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta-1].fila_nodo;
  obstac_columna=info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta-1].columna_nodo;
  printf("fila:%d - col:%d\n",obstac_fila,obstac_columna);
  //se halla el nodo a verificar de acuerdo al sentido de orientacion hacia "atras" de la oruga
  switch(info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta].pendiente){
    case 0: k_star=0;l_star=-7;break;
    case 45: k_star=-7;l_star=-7;break;
    case 90: k_star=-7;l_star=0;break;
    case 135: k_star=-7;l_star=7;break;
    case -180: k_star=0;l_star=7;break;
    case -135: k_star=7;l_star=7;break;
    case -90: k_star=7;l_star=0;break;
    case -45: k_star=7;l_star=-7;break;
  }
  cont_kl=7;
  while(cont_kl>0){
    //se verifica si en el nodo encontrado hay obstaculo que no obstruya con la oruga
    if((obstac_fila+k_star>=0 && obstac_fila+k_star<map_size_rows) && (obstac_columna+l_star>=0 && obstac_columna+l_star<map_size_cols)){
      printf("revisar fila/col: %d-%d, resultado map:%d\n",obstac_fila+k_star,obstac_columna+l_star,map[obstac_fila+k_star][obstac_columna+l_star]);
      if(map[obstac_fila+k_star][obstac_columna+l_star]==1){
        obstaculo_encontrado++;
      }
    }
     else {
       printf("revision fuera de limites\n");obstaculo_encontrado++;
     }
     if(k_star>0){k_star--;}
     else if(k_star<0){k_star++;}
     if(l_star>0){l_star--;}
     else if(l_star<0){l_star++;}
     cont_kl--;
   }
   //en caso de haber cambia la arquitectura posible de una oruga por hexapodo dada la imposibilidad de ejecutar la oruga
   if(obstaculo_encontrado>0){
     printf("obstaculo(s) encontrado(s): %d\n",obstaculo_encontrado);
     if(contador_nodos_recta-obstaculo_encontrado<=6){
       printf("cambio de arquitectura\n");
       trayecto_arquitecturas[t_recta].forma_ejecutar=2;
     }
```

```
          else{
            info_trayectos[t_nodo].recta=info_trayectos[t_nodo-1].recta+2;
            trayecto_arquitecturas[t_recta].cantidad_nodos=obstaculo_encontrado;
            trayecto_arquitecturas[t_recta].forma_ejecutar=2;
            printf("Recta %d - Cantidad de Nodos: %d, arquitectura:%d, nodo que inicia:%d, pendiente:%d \n",
              trayecto_arquitecturas[t_recta].numero_linea,trayecto_arquitecturas[t_recta].cantidad_nodos,
              trayecto_arquitecturas[t_recta].forma_ejecutar,trayecto_arquitecturas[t_recta].nodo_inicia_recta,
              info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta].pendiente);
            ++con_recta;
            trayecto_arquitecturas = (struct trayecto_arquitectura *)realloc(trayecto_arquitecturas, con_recta * sizeof(struct trayecto_arquitectura));
            t_recta = con_recta - 1;
            trayecto_arquitecturas[t_recta].numero_linea=t_recta;
            trayecto_arquitecturas[t_recta].cantidad_nodos=contador_nodos_recta-obstaculo_encontrado;
            trayecto_arquitecturas[t_recta].forma_ejecutar=1;
            trayecto_arquitecturas[t_recta].nodo_inicia_recta=trayecto_arquitecturas[t_recta-1].nodo_inicia_recta+obstaculo_encontrado;
            contador_nodos_recta=contador_nodos_recta-obstaculo_encontrado;//se ajusta la cantidad de nodos de la nueva recta con arquitectura ajustada
          }
        }
        else{trayecto_arquitecturas[t_recta].forma_ejecutar=1;}
      }
      contador_nodos_recta_anterior=contador_nodos_recta;
      contador_nodos_recta=1;
      printf("Recta %d - Cantidad de Nodos: %d, arquitectura:%d, nodo que inicia:%d, pendiente:%d \n",
        trayecto_arquitecturas[t_recta].numero_linea,trayecto_arquitecturas[t_recta].cantidad_nodos,
        trayecto_arquitecturas[t_recta].forma_ejecutar,trayecto_arquitecturas[t_recta].nodo_inicia_recta,
        info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta].pendiente);
    }
    else {info_trayectos[t_nodo].recta=info_trayectos[t_nodo-1].recta;contador_nodos_recta++;}
  }
  printf("(%1.0f, %1.0f, %d, %d)\n",info_trayectos[t_nodo].fila_nodo,info_trayectos[t_nodo].columna_nodo,info_trayectos[t_nodo].pendiente,info_trayectos[t_nodo].recta);
  if(i == 0){
    ++con_recta;
    trayecto_arquitecturas = (struct trayecto_arquitectura *)realloc(trayecto_arquitecturas, con_recta * sizeof(struct trayecto_arquitectura));
    t_recta = con_recta - 1;
    trayecto_arquitecturas[t_recta].numero_linea=t_recta;
    trayecto_arquitecturas[t_recta].nodo_inicia_recta=trayecto_arquitecturas[t_recta-1].nodo_inicia_recta+contador_nodos_recta_anterior;
    trayecto_arquitecturas[t_recta].cantidad_nodos=p_len - trayecto_arquitecturas[t_recta].nodo_inicia_recta;
    if(info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta].pendiente==info_trayectos[trayecto_arquitecturas[t_recta-1].nodo_inicia_recta].pendiente){
      trayecto_arquitecturas[t_recta].forma_ejecutar=trayecto_arquitecturas[t_recta-1].forma_ejecutar;
    }
    else{trayecto_arquitecturas[t_recta].forma_ejecutar=2;}

    printf("Recta %d - Cantidad de Nodos: %d, arquitectura:%d, nodo que inicia:%d, pendiente:%d \n",
      trayecto_arquitecturas[t_recta].numero_linea,trayecto_arquitecturas[t_recta].cantidad_nodos,
      trayecto_arquitecturas[t_recta].forma_ejecutar,trayecto_arquitecturas[t_recta].nodo_inicia_recta,
      info_trayectos[trayecto_arquitecturas[t_recta].nodo_inicia_recta].pendiente);
  }
}
printf("RESULTADO ELECCION DE ARQUITECTURA(S) SOBRE LA TRAYECTORIA PLANTEADA:\n");
```

```
    //eleccion arquitecturas sobre la trayectoria planteada
    for (i = 1; i <=t_recta; i++) {
      printf("Recta %d - Pendiente:%d, Cantidad de Nodos: %d, arquitectura:%d, nodo que inicia:%d - fila/columna: %1.0f, %1.0f\n",
        trayecto_arquitecturas[i].numero_linea,info_trayectos[trayecto_arquitecturas[i].nodo_inicia_recta].pendiente,
        trayecto_arquitecturas[i].cantidad_nodos,trayecto_arquitecturas[i].forma_ejecutar,
        trayecto_arquitecturas[i].nodo_inicia_recta,info_trayectos[trayecto_arquitecturas[i].nodo_inicia_recta-1].fila_nodo,
        info_trayectos[trayecto_arquitecturas[i].nodo_inicia_recta-1].columna_nodo);
    }
 }
 recta_ejecucion=1;
 for (i = 0; i < s_len; ++i) {
     free(stops[i].n);
 }
 free(stops);
 free(routes);
 free(path);
 free(open);
 free(closed);
 free(obstaculos);
 //free(info_trayectos);
 //free(trayecto_arquitecturas);
 bandera=12;printf("PARA EJECUTAR TRAYECTORIA OPRIMA ENTER\n");
}
```

Anexo 16. Secuencia de programación del módulo 1 (maestro) para ejecutar las arquitecturas obtenidas y recorrer la trayectoria obtenida del algoritmo de búsqueda A* (A-Star)

```
void ejecutar_trayectoria(){
  //CUANDO LA ARQUITECTURA ES HEXAPODO
  if(final_morfologia==2 && (trayecto_arquitecturas[recta_ejecucion].forma_ejecutar==2 || trayecto_arquitecturas[recta_ejecucion].forma_ejecutar==1)){
    // para hallar el angulo de orientacion al punto deseado
    if(robotflag==0 && message_printed == 1){
      if(recta_ejecucion<t_recta){
        nodo_fila_final=info_trayectos[trayecto_arquitecturas[recta_ejecucion+1].nodo_inicia_recta-1].fila_nodo;//nodo en el que inicia la siguiente recta (por eso no tiene el -1)
        nodo_columna_final=info_trayectos[trayecto_arquitecturas[recta_ejecucion+1].nodo_inicia_recta-1].columna_nodo;//nodo en el que inicia la siguiente recta (por eso no tiene e
      }
      else {
        nodo_fila_final=info_trayectos[p_len-1].fila_nodo;
        nodo_columna_final=info_trayectos[p_len-1].columna_nodo;
      }
      mapa_a_coordenadas();
      zf=posfinal_z;
      xf=posfinal_x;
      coordenadas_modulo();
      orientacion_modulo();
      coordenadas_a_mapa();
      printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
      printf("Distancia:%lf, alfa:%d,\n", dist, alfa);
      robotflag=1;
    }
    //para girar hacia al punto deseado
    if(robotflag==1 && message_printed == 1){
      coordenadas_modulo();printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
      if((alfa-angulo)>=-3 && (alfa-angulo)<=3){hexapodo_movimiento=4;printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
        robotflag=2;printf("listo 2: giro hacia el punto deseado\n");
      }
      else if((alfa-angulo)>0){
        if(abs(alfa-angulo)<=180){hexapodo_movimiento=3;}
        else {hexapodo_movimiento=2;}
      }
      else{
        if(abs(alfa-angulo)<=180){hexapodo_movimiento=2;}
        else {hexapodo_movimiento=3;}
      }
    }
    //reposo
    else if(robotflag==2){dist_anterior=dist+500;
      printf("entro 1: reposo\n");hexapodo_movimiento=10;robotflag=3;
    }
```

```
//para ir de frente hacia el nodo marcado por la fila y columna final
else if(robotflag==3 && message_printed == 1){
  coordenadas_modulo();printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
  if ((alfa-angulo)>=-10 && (alfa-angulo)<=10){printf("entro 1\n");coordenadas_a_mapa();
    if(nodo_fila_actual!=nodo_fila_final || nodo_columna_actual!=nodo_columna_final){
      if(dist_anterior>=dist){
        dist_anterior=dist+10;printf("entro 2: Va bien, Paso al frente\n");hexapodo_movimiento=1;
      }
      else {hexapodo_movimiento=4;robotflag=4;printf("entro 3: se esta alejando, validar angulo\n");}
    }
    else {hexapodo_movimiento=4;robotflag=5;printf("listo 3: llegada a nodo\n");}
  }
  else {
    hexapodo_movimiento=4;robotflag=4;printf("entro 3: se desalineo, validar angulo\n");
  }
}
//reposo
else if(robotflag==4){
  printf("entro 4: reposo, calculo de angulo\n");hexapodo_movimiento=10;robotflag=0;
}
//reposo
else if(robotflag==5){
  printf("entro 5: validacion de mas rectas\n");hexapodo_movimiento=10;
  //se cambia de recta si no es la final
  if(recta_ejecucion+1<=t_recta){printf("cambio de recta\n");
    recta_ejecucion++;
    //se verifica si continua en la misma arquitectura
    if(trayecto_arquitecturas[recta_ejecucion].forma_ejecutar==2){
      robotflag=0;
    }
    else{//se debe girar y dejarlo en linea recta para cambiar a ORUGA
      robotflag=10;printf("cambio de arquitectura a Oruga\n");
    }
  }
  else{robotflag=6;printf("llegada a nodo final\n");}//hace nada
}
////////////////////////////cuando hay cambio de recta se encuentra en Hexapodo y debe pasar a oruga se hace alineacion en angulo
// para hallar el angulo de orientacion al punto deseado
if(robotflag==10 && message_printed == 1){
  if(recta_ejecucion<t_recta){
    nodo_fila_final=info_trayectos[trayecto_arquitecturas[recta_ejecucion+1].nodo_inicia_recta-1].fila_nodo;//nodo en el que inicia la siguiente recta (por eso no tiene el -1)
    nodo_columna_final=info_trayectos[trayecto_arquitecturas[recta_ejecucion+1].nodo_inicia_recta-1].columna_nodo;//nodo en el que inicia la siguiente recta (por eso no tiene e
  }
  else {
    nodo_fila_final=info_trayectos[p_len-1].fila_nodo;
    nodo_columna_final=info_trayectos[p_len-1].columna_nodo;
  }
```

```
    mapa_a_coordenadas();
    zf=posfinal_z;
    xf=posfinal_x;
    coordenadas_modulo();
    orientacion_modulo();
    coordenadas_a_mapa();
    printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
    printf("Distancia:%lf, alfa:%d,\n", dist, alfa);
    if((alfa-angulo)>=-3 && (alfa-angulo)<=3){hexapodo_movimiento=4;printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
      hexapodo_movimiento=0;robotflag=13;printf("listo 2 - salto: giro hacia el punto deseado\n");
    }
    else {robotflag=11;printf("entro 6\n");}
  }
  //para girar hacia al punto deseado
  if(robotflag==11 && message_printed == 1){
    coordenadas_modulo();
    if((alfa-angulo)>=-3 && (alfa-angulo)<=3){hexapodo_movimiento=4;printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
      robotflag=12;printf("listo 2: giro hacia el punto deseado\n");
    }
    else if((alfa-angulo)>0){printf("giro 1\n");
      if(abs(alfa-angulo)<=180){hexapodo_movimiento=3;}
      else {hexapodo_movimiento=2;}
    }
    else{printf("giro 2\n");
      if(abs(alfa-angulo)<=180){hexapodo_movimiento=2;}
      else {hexapodo_movimiento=3;}
    }
  }
  //reposo
  else if(robotflag==12){dist_anterior=dist+200;
    printf("entro 7\n");hexapodo_movimiento=0;robotflag=13;
  }
  if(robotflag==13 && flag==53 && datosderecibo[0]==13 && message_printed == 1){
    final_morfologia=0;arquitectura=4;flag=54;robotflag=14;//hace nada
  }
}
//CUANDO LA ARQUITECTURA ES DESACOPLE DE HEXAPODO
if(final_morfologia==4){
  final_morfologia=0;arquitectura=1;flag=10;contador_movimiento=0;
  modulo_accion=4;modulo_conectar=4;com_forma=0;comunicacion_modulos();
}
//CUANDO LA ARQUITECTURA ES ORUGA
if(final_morfologia==1 && trayecto_arquitecturas[recta_ejecucion].forma_ejecutar==1){
  if(contador_movimiento==0){printf("entro 6: inicio oruga\n");oruga_movimiento=1;flag=22;}
  else if(contador_movimiento==1){printf("entro 7: validacion de coordenadas con respecto a destino final\n");
    coordenadas_modulo();
    printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
    if ((alfa-angulo)>=-15 && (alfa-angulo)<=15){printf("entro 8\n");coordenadas_a_mapa();
      if(nodo_fila_actual!=nodo_fila_final || nodo_columna_actual!=nodo_columna_final){printf("entro 9\n");
        if(dist_anterior>=dist){
          if(((alfa-angulo)<=-10 || (alfa-angulo)>=10) && dist<450){contador_movimiento=98;robotflag=0;
            printf("entro 10: punto cercano fuera de angulo, completar trayectoria en hexapodo\n");}
          else if(map[nodo_fila_actual][nodo_columna_actual]==1){contador_movimiento=98;robotflag=0;
            printf("entro 11: punto de obstaculo, completar trayectoria en hexapodo\n");}
```

```
        else {dist_anterior=dist;oruga_movimiento=1;printf("VA BIEN, CONTINUA\n");}
       }
       else {contador_movimiento=98;robotflag=0;printf("entro 12: se esta alejando, completar trayectoria en hexapodo\n");}
     }
     else {contador_movimiento=98;printf("listo 4: llegada a nodo\n");robotflag=15;}
   }
   else {
     contador_movimiento=98;robotflag=0;printf("entro 13: se desalineo, completar trayectoria en hexapodo\n");
   }
 }
 else if(contador_movimiento==89 || contador_movimiento==99){flag=24;oruga_movimiento=0;}
 else if(contador_movimiento==90 && flag==53 && datosderecibo[0]==15 && message_printed == 1){contador_movimiento=0;oruga_movimiento=1;flag=22;}
else if(contador_movimiento==100 && flag==53 && datosderecibo[0]==15 && message_printed == 1){
  if(robotflag==0){
    //se debe cambiar a Hexapodo
    trayecto_arquitecturas[recta_ejecucion].forma_ejecutar=2;
    robotflag=0;printf("cambio de arquitectura a Hexapodo\n");
    contador_movimiento=0;final_morfologia=0;arquitectura=3;flag=54;
  }
  else if(robotflag==15){
    //se cambia de recta si no es la final
    if(recta_ejecucion+1<=t_recta){printf("cambio de recta\n");
      recta_ejecucion++;
      //se verifica si continua en la misma arquitectura
      if(trayecto_arquitecturas[recta_ejecucion].forma_ejecutar==1){
        contador_movimiento=0;robotflag=14;//hace nada
        //se hace el ajuste de nuevas coordenadas del nodo de la siguiente recta
        if(recta_ejecucion<t_recta){
          nodo_fila_final=info_trayectos[trayecto_arquitecturas[recta_ejecucion+1].nodo_inicia_recta-1].fila_nodo;//nodo en el que inicia la siguiente recta (por eso no tiene e
          nodo_columna_final=info_trayectos[trayecto_arquitecturas[recta_ejecucion+1].nodo_inicia_recta-1].columna_nodo;//nodo en el que inicia la siguiente recta (por eso no t
        }
        else {
          nodo_fila_final=info_trayectos[p_len-1].fila_nodo;
          nodo_columna_final=info_trayectos[p_len-1].columna_nodo;
        }
        mapa_a_coordenadas();
        zf=posfinal_z;
        xf=posfinal_x;
        coordenadas_modulo();
        coordenadas_a_mapa();
        printf("GPS position:{X:%d, Z:%d} - Angulo:%d - Dist:%lf \n", posxz[0], posxz[1],angulo,dist);
```

```
            if(info_trayectos[trayecto_arquitecturas[recta_ejecucion].nodo_inicia_recta].pendiente!=
              info_trayectos[trayecto_arquitecturas[recta_ejecucion-1].nodo_inicia_recta].pendiente){
              printf("Siguiente recta, cambia pendiente, ajuste de angulo en hexapodo y despues continua en ORUGA\n");
              orientacion_modulo();
              printf("Distancia:%lf, alfa:%d,\n", dist, alfa);
              robotflag=10;printf("cambio de arquitectura a Hexapodo\n");
              contador_movimiento=0;final_morfologia=0;arquitectura=3;flag=54;
            }
            else{printf("Siguiente recta, misma pendiente - continua en ORUGA\n");}
          }
          else{//se debe cambiar a Hexapodo
            robotflag=0;printf("cambio de arquitectura a Hexapodo\n");
            contador_movimiento=0;final_morfologia=0;arquitectura=3;flag=54;
          }
        }
        else{robotflag=14;printf("llegada a nodo final\n");}//hace nada
      }
    }
  }
  //CUANDO LA ARQUITECTURA ES DESACOPLE DE ORUGA
  if(final_morfologia==3){
    final_morfologia=0;arquitectura=2;flag=10;
    modulo_accion=4;modulo_conectar=4;com_forma=0;comunicacion_modulos();
  }
}
```

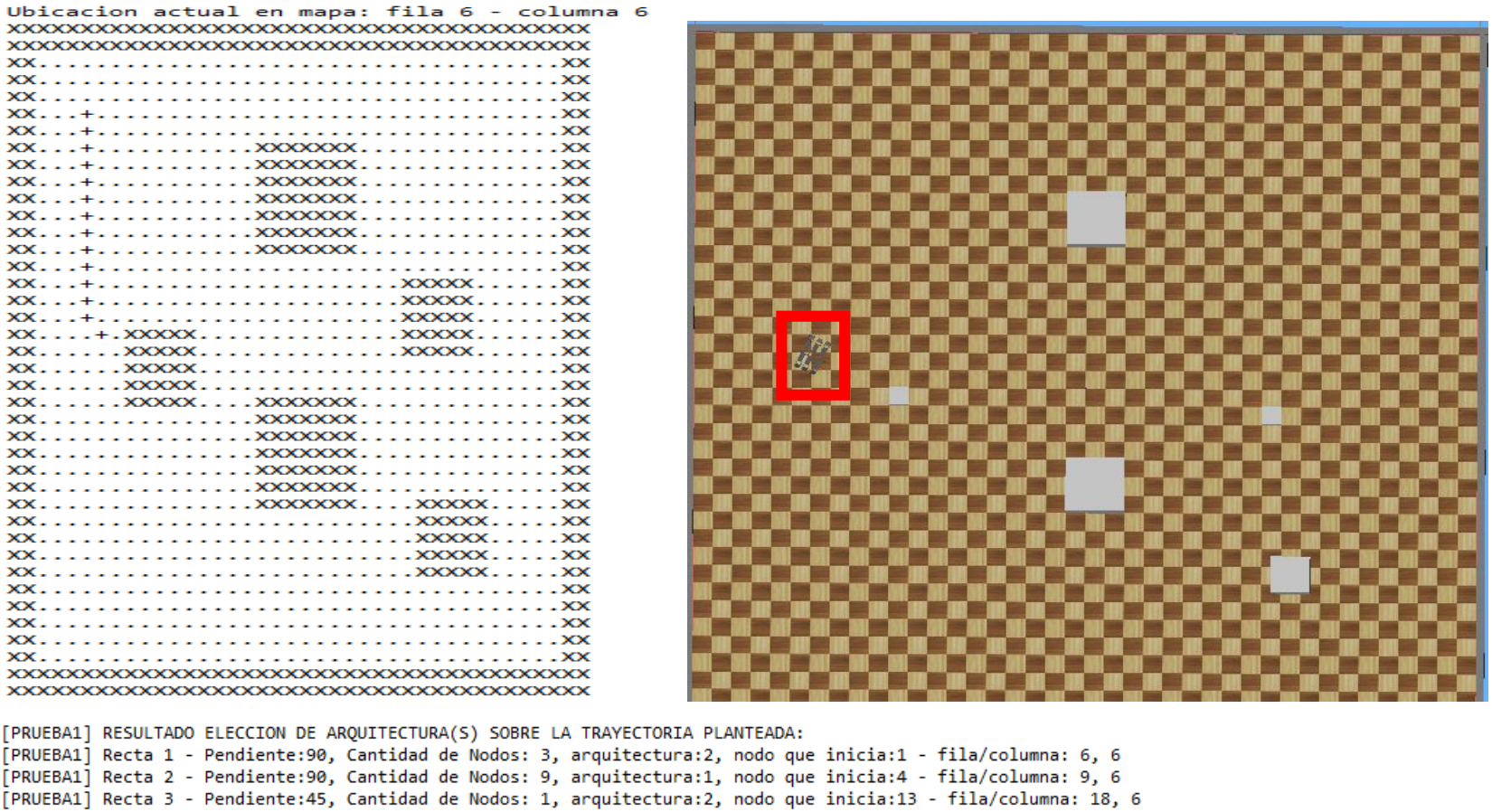
Anexo 17. Secuencia de programación del módulo 1 (maestro) para ejecutar el menú del programa en interacción con el usuario

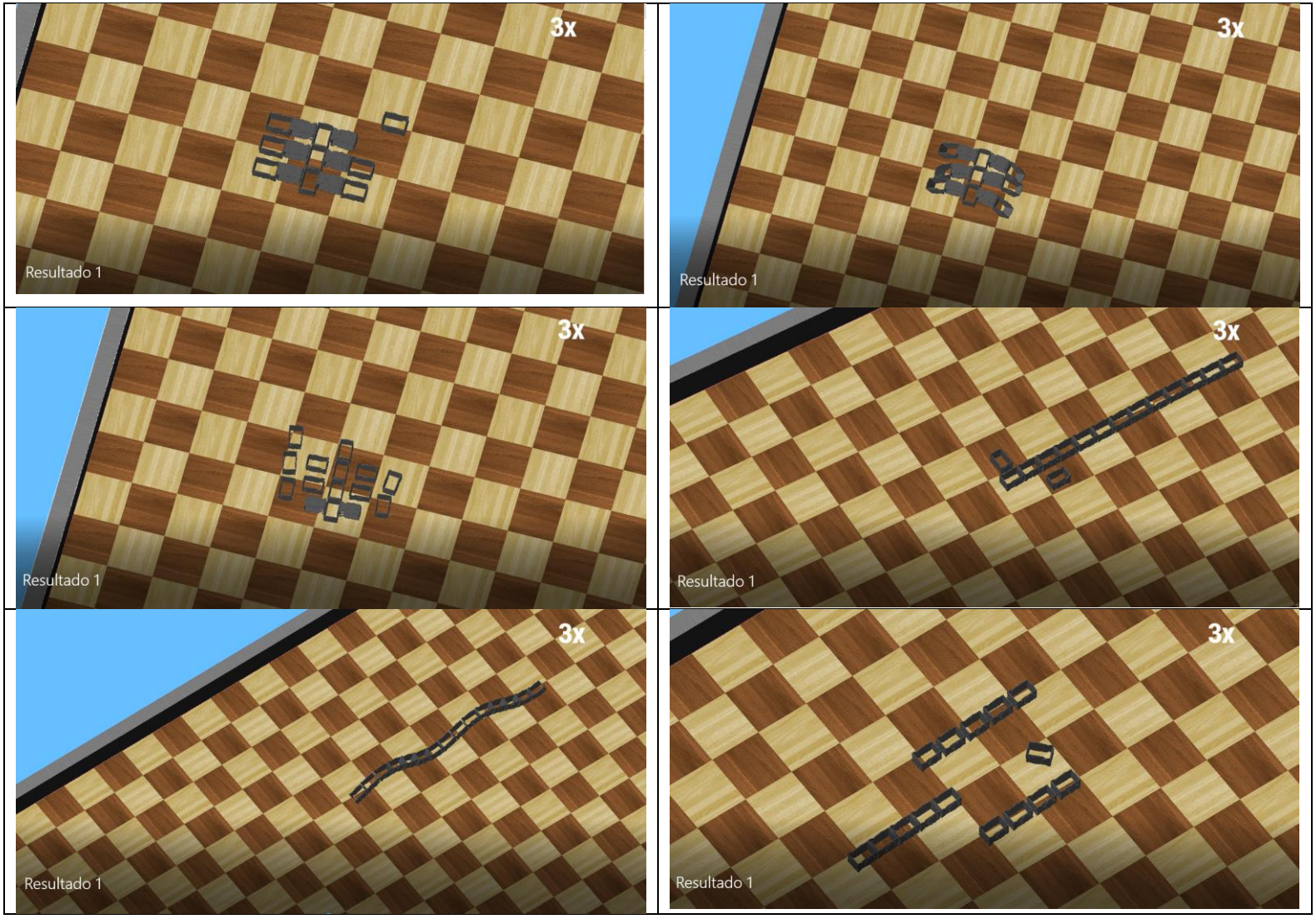
```
void leer_teclado(){
  if(bandera<13){
    key = wb_robot_keyboard_get_key();
    switch(key)
    {
      case 48: num_teclado=0;break;
      case 49: num_teclado=1;break;
      case 50: num_teclado=2;break;
      case 51: num_teclado=3;break;
      case 52: num_teclado=4;break;
      case 53: num_teclado=5;break;
      case 54: num_teclado=6;break;
      case 55: num_teclado=7;break;
      case 56: num_teclado=8;break;
      case 57: num_teclado=9;break;
      case 69: num_teclado=20;break;//(E)mpezar
      case 80: num_teclado=21;break;//(P)arar
      case 82: num_teclado=22;break;//(R)einiciar
      case 4: num_teclado=-1;break;
      case 5: num_teclado=-1;break;
      default: key=0;
    }
    if(key==0){
      key_anterior=0;
    }
    if(key_anterior!=key && key!=0){
      if(num_teclado>=0 && num_teclado<10){num_teclado_ant=num_teclado;printf("%d\n",num_teclado);}
      if(num_teclado==20){printf("Empezar\n");bandera=1;}
      if(num_teclado==21){printf("Parar\n");}
      if(num_teclado==22){printf("Reiniciar\n");}
      key_anterior=key;
    }
  }
}
```

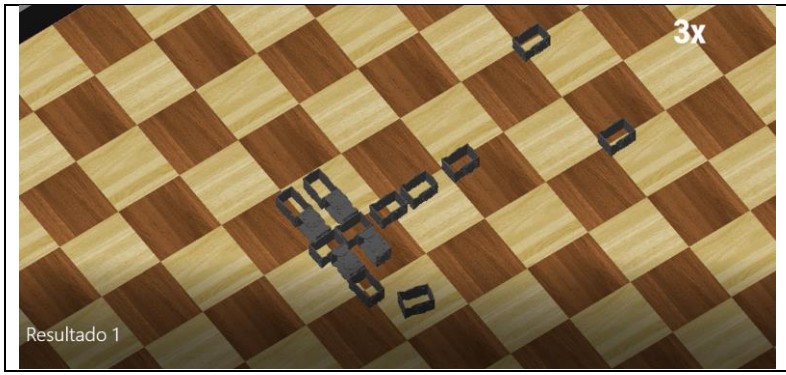
```
void menu_programa(){
  if (bandera==1){printf("BIENVENIDO AL PROGRAMA\n");printf("Ingrese el primer digito de la fila objetivo (2-38) luego la tecla enter\n");bandera=2;}
  else if (bandera==2){if(key==4 || key==5){fila_nodofinal=num_teclado_ant*10;printf("Valor ingresado: %d\n",num_teclado_ant);bandera=3;}}
  else if (bandera==3){if(key==0){printf("Ingrese el segundo digito de la fila objetivo (2-38) luego la tecla enter\n");bandera=4;}}
  else if (bandera==4){if(key==4 ||key==5){fila_nodofinal=fila_nodofinal+num_teclado_ant;printf("Valor ingresado: %d\n",num_teclado_ant);bandera=5;}}
  else if (bandera==5){if(key==0){printf("La fila objetivo es: %d\n",fila_nodofinal);if(fila_nodofinal>1 && fila_nodofinal<=38){bandera=6;}
                       else{bandera=1;printf("error en fila\n");}}}
  else if (bandera==6){printf("Ingrese el primer digito de la columna objetivo (2-38) luego la tecla enter\n");bandera=7;}
  else if (bandera==7){if(key==4 || key==5){columna_nodofinal=num_teclado_ant*10;printf("Valor ingresado: %d\n",num_teclado_ant);bandera=8;}}
  else if (bandera==8){if(key==0){printf("Ingrese el segundo digito de la columna objetivo (2-38) luego la tecla enter\n");bandera=9;}}
  else if (bandera==9){if(key==4 ||key==5){columna_nodofinal=columna_nodofinal+num_teclado_ant;printf("Valor ingresado: %d\n",num_teclado_ant);bandera=10;}}
  else if (bandera==10){if(key==0){printf("La columna objetivo es: %d\n",columna_nodofinal);if(columna_nodofinal>1 && columna_nodofinal<=38){bandera=11;}
                       else{bandera=6;printf("error en columna\n");}}}
  else if (bandera==12){if(key==4 ||key==5){printf("EJECUTANDO TRAYECTORIA\n");bandera=13;}}
}
```

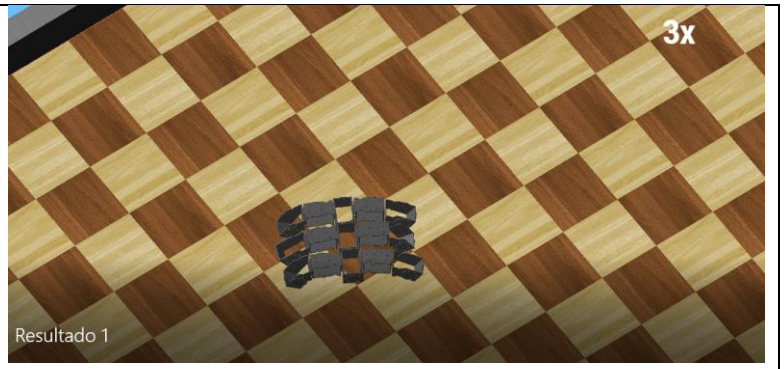
Anexo 18. Resultado 1 obtenido – resumen fotográfico

Resultado	Fila Objetivo	Columna Objetivo	N° nodos trayectoria	Número de rectas	Cambios de arquitectura	Tiempo total Minutos -3X
1	19	7	14	3	2	33:15



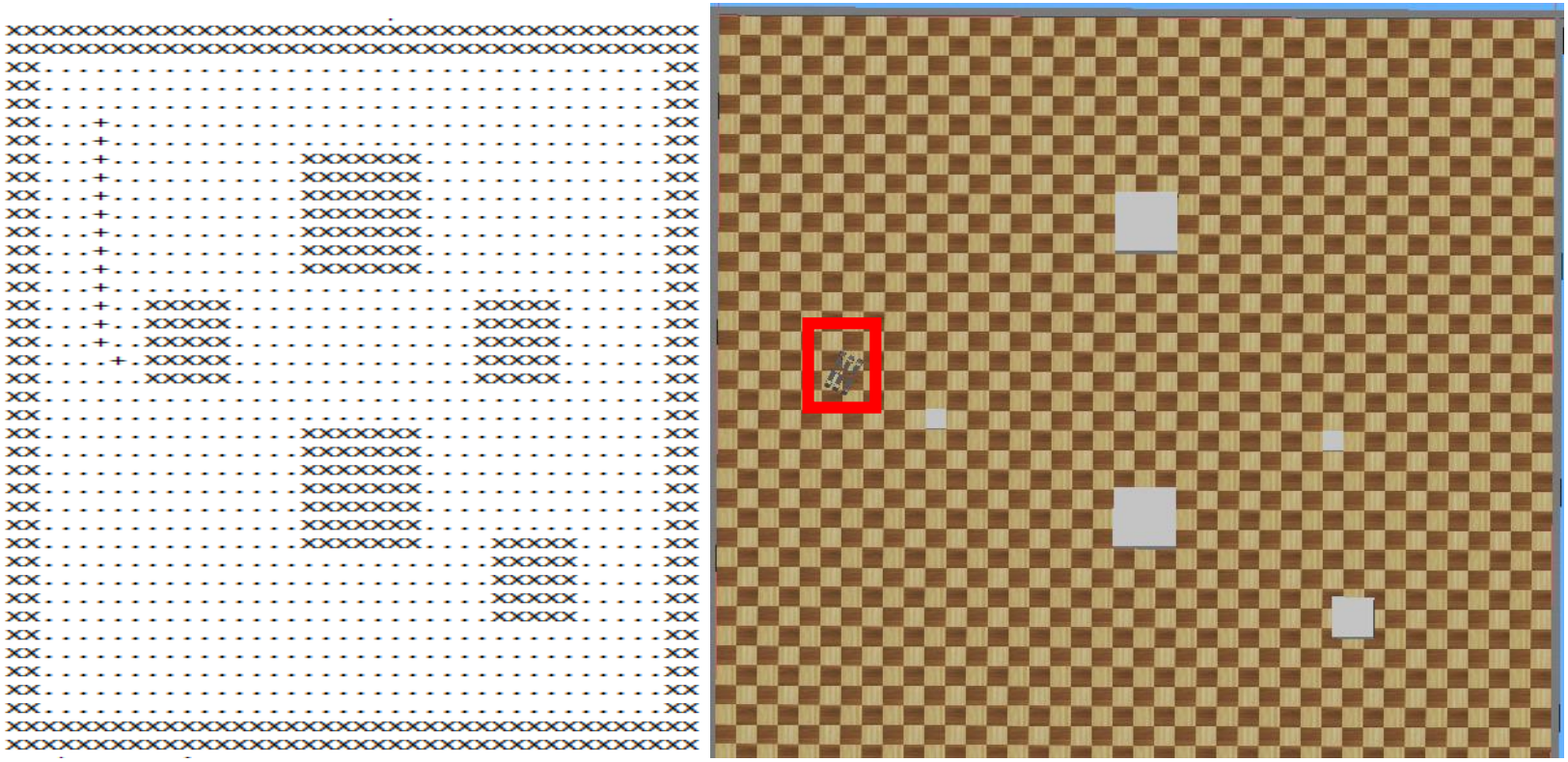

3x
Resultado 1
3x
Resultado 1
3x
Resultado 1
3x
Resultado 1
3x
Resultado 1
3x
Resultado 1


3x
Resultado 1

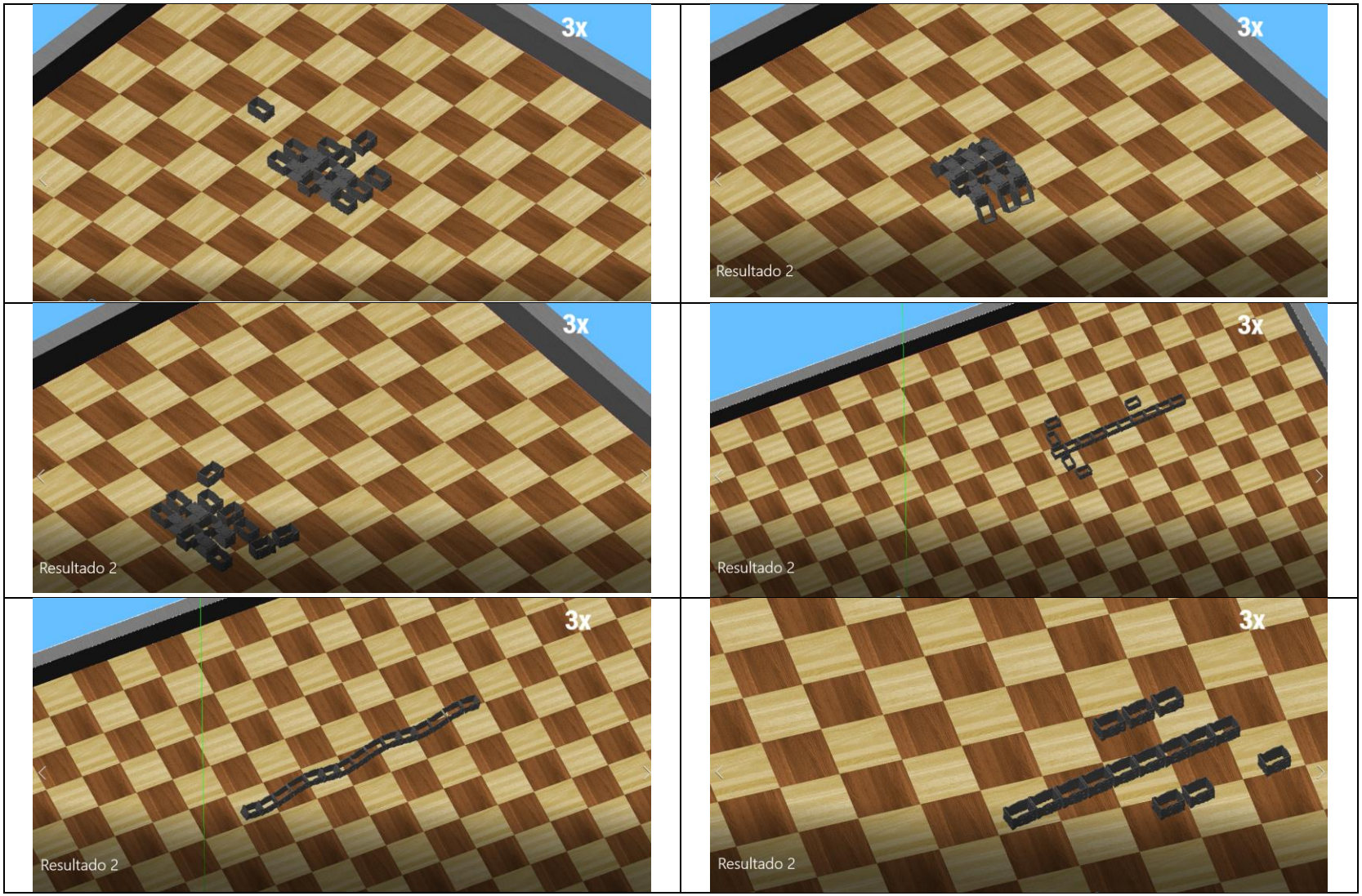

3x
Resultado 1

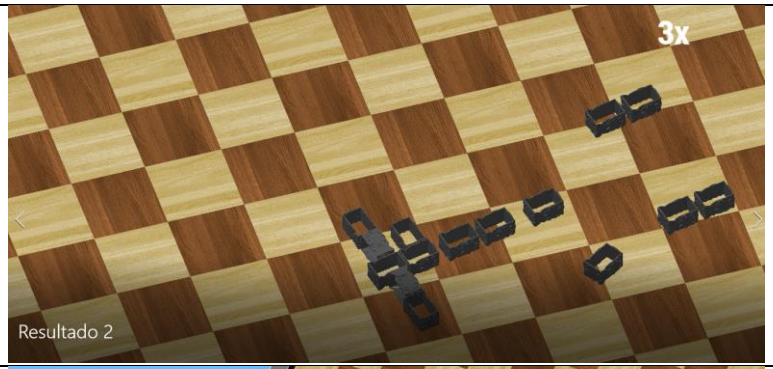
Anexo 19. Resultado 2 obtenido – resumen fotográfico

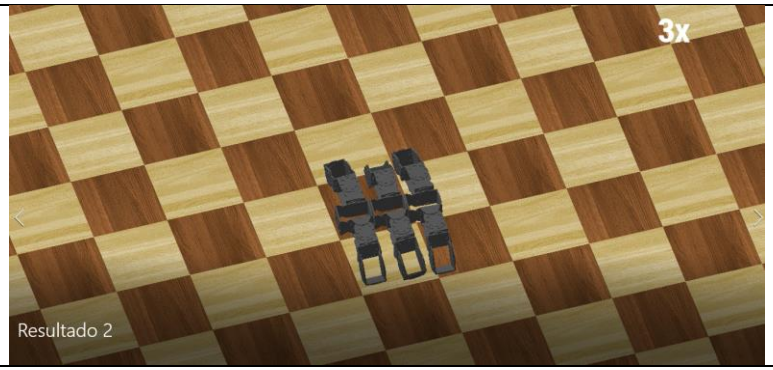
Resultado	Fila Objetivo	Columna Objetivo	N° nodos trayectoria	Número de rectas	Cambios de arquitectura	Tiempo total Minutos -3X
2	19	7	14	3	2	34:03

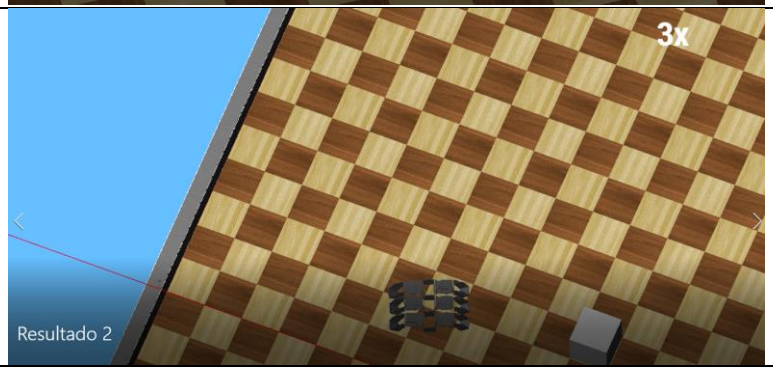


```
[PRUEBA1] RESULTADO ELECCION DE ARQUITECTURA(S) SOBRE LA TRAYECTORIA PLANTEADA:
[PRUEBA1] Recta 1 - Pendiente:90, Cantidad de Nodos: 3, arquitectura:2, nodo que inicia:1 - fila/columna: 6, 6
[PRUEBA1] Recta 2 - Pendiente:90, Cantidad de Nodos: 9, arquitectura:1, nodo que inicia:4 - fila/columna: 9, 6
[PRUEBA1] Recta 3 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:13 - fila/columna: 18, 6
```


3x
3x
Resultado 2
3x
Resultado 2
3x
Resultado 2
3x
Resultado 2
3x
Resultado 2

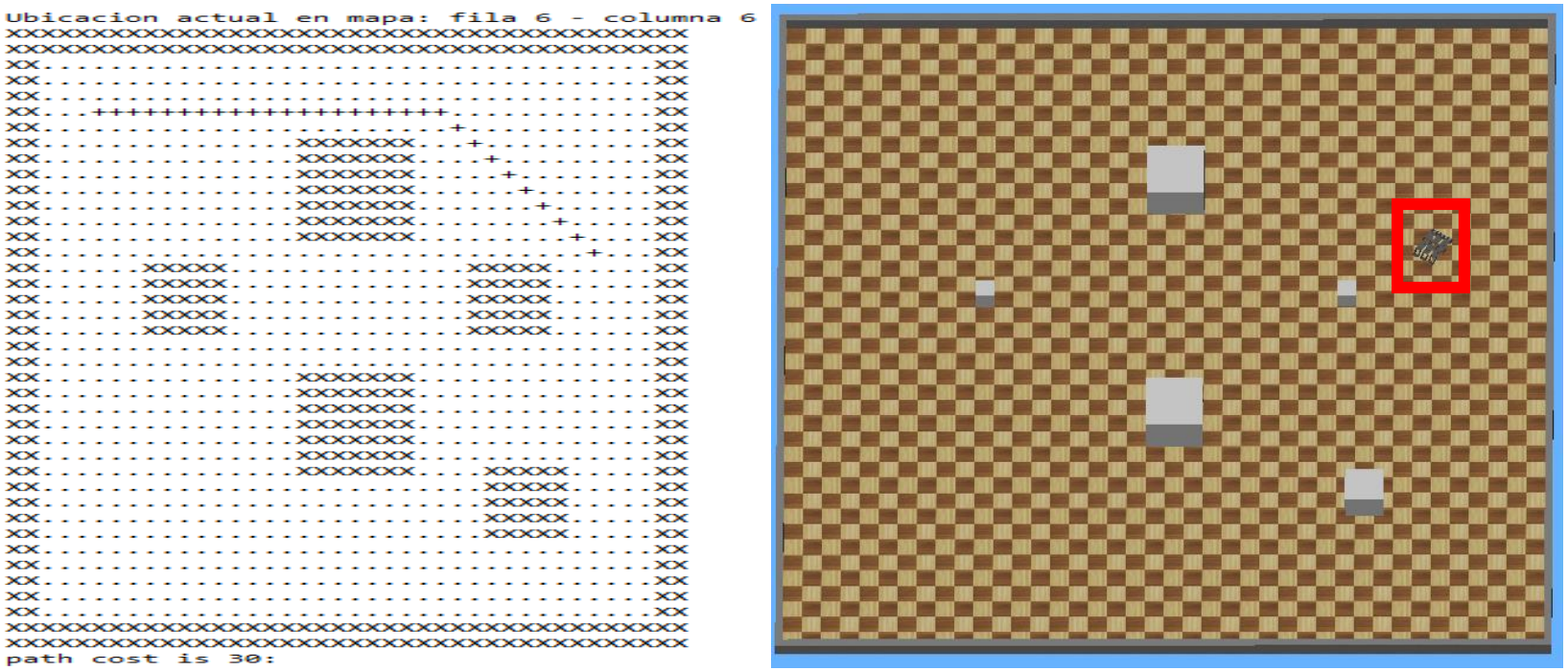

3x
Resultado 2


3x
Resultado 2

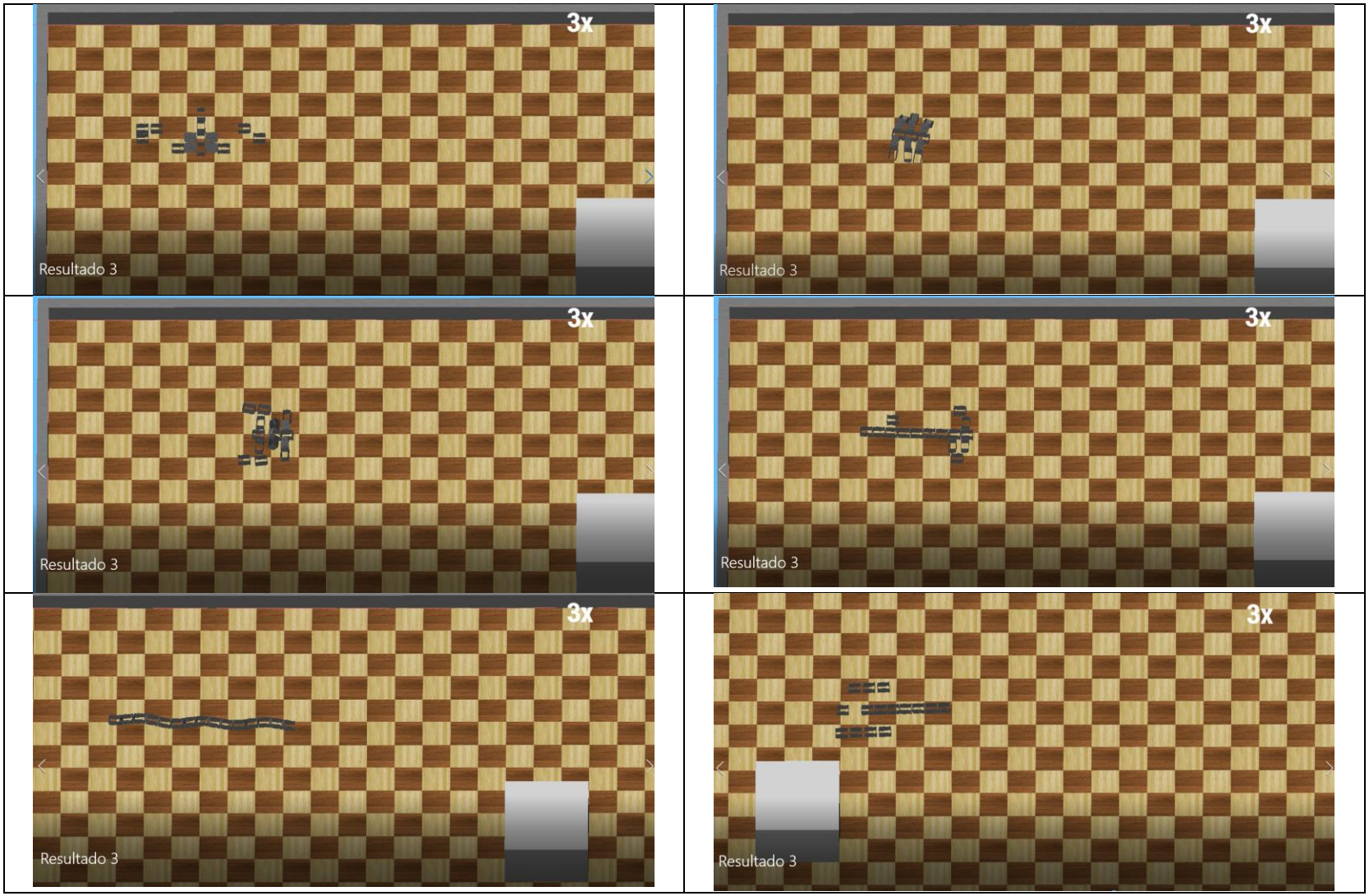

3x
Resultado 2

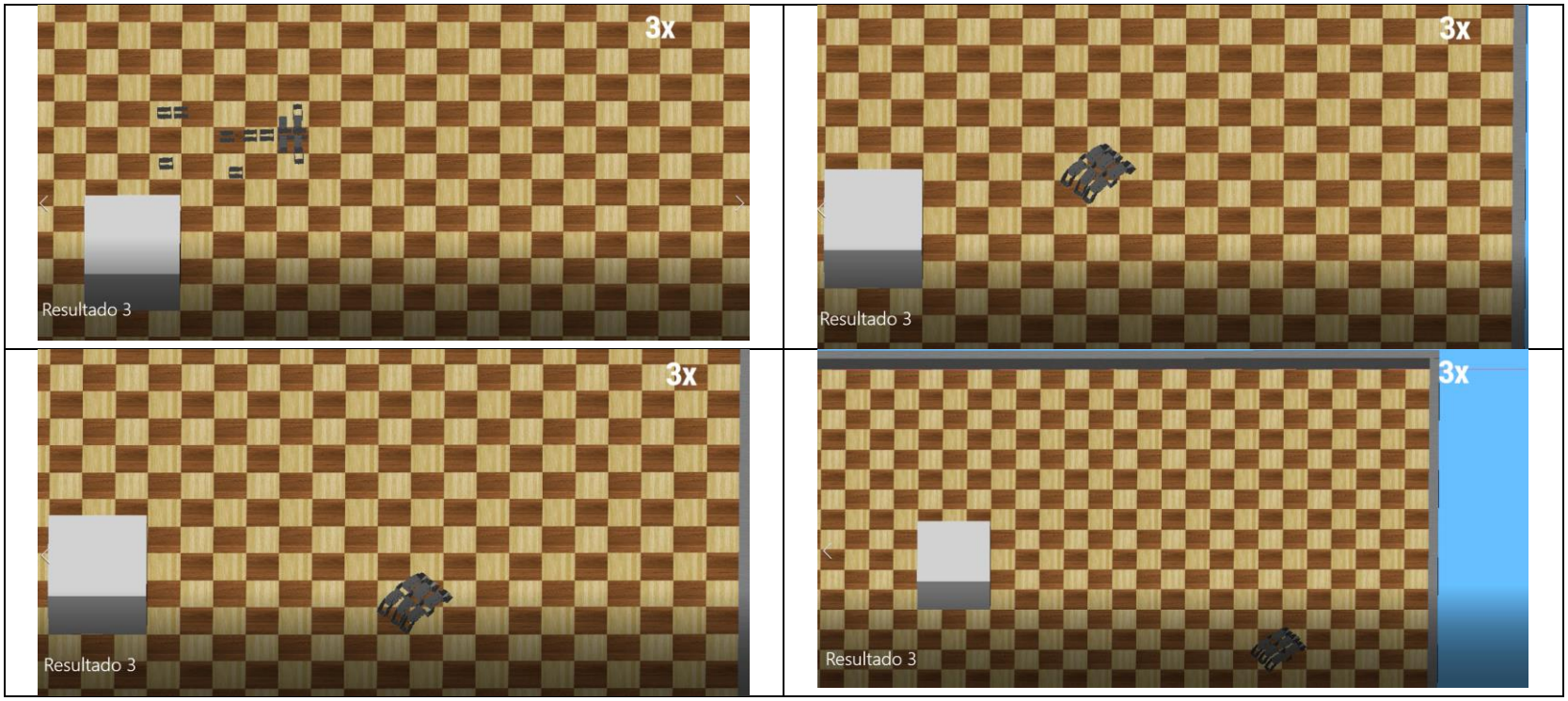
Anexo 20. Resultado 3 obtenido – resumen fotográfico

Resultado	Fila Objetivo	Columna Objetivo	N° nodos trayectoria	Número de rectas	Cambios de arquitectura	Tiempo total Minutos -3X
3	15	35	30	4	2	34:44



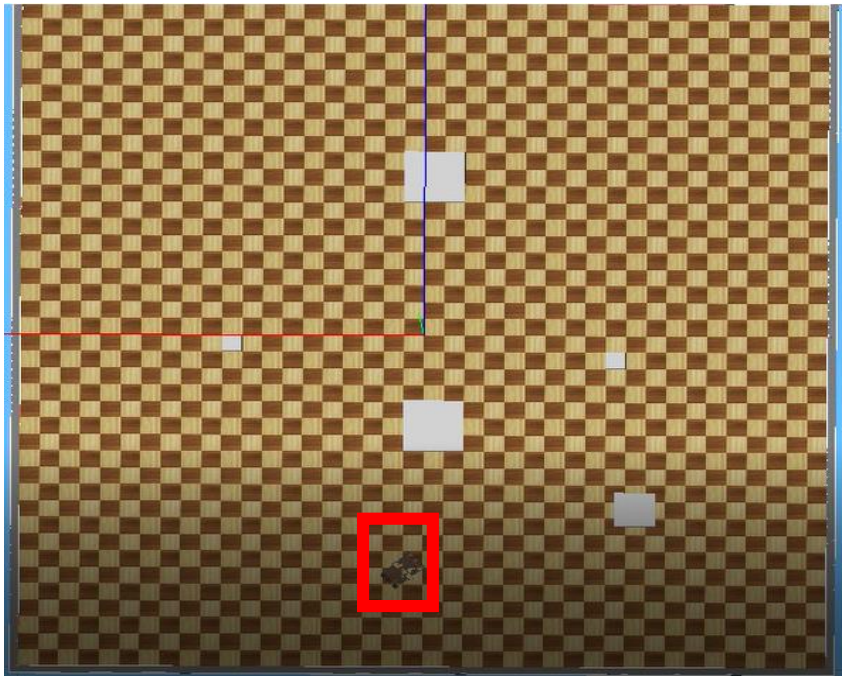

```
RESULTADO ELECCION DE ARQUITECTURA(S) SOBRE LA TRAYECTORIA PLANTEADA:
Recta 1 - Pendiente:0, Cantidad de Nodos: 3, arquitectura:2, nodo que inicia:1 - fila/columna: 6, 6
Recta 2 - Pendiente:0, Cantidad de Nodos: 17, arquitectura:1, nodo que inicia:4 - fila/columna: 6, 9
Recta 3 - Pendiente:45, Cantidad de Nodos: 8, arquitectura:2, nodo que inicia:21 - fila/columna: 6, 26
Recta 4 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:29 - fila/columna: 14, 34
```


3x
Resultado 3
3x
Resultado 3
3x
Resultado 3
3x
Resultado 3
3x
Resultado 3
3x
Resultado 3


3x
Resultado 3
3x
Resultado 3
3x
Resultado 3
3x
Resultado 3

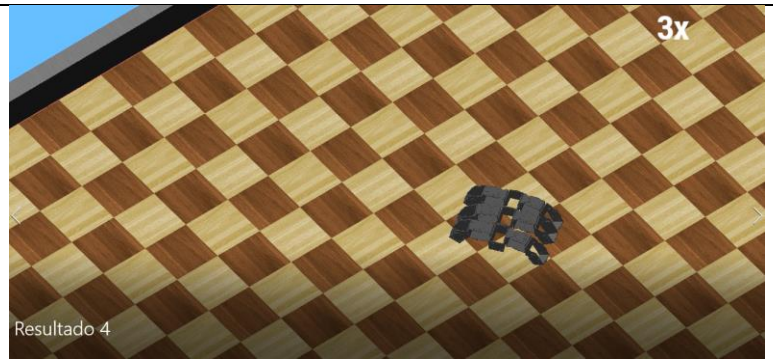
Anexo 21. Resultado 4 obtenido – resumen fotográfico

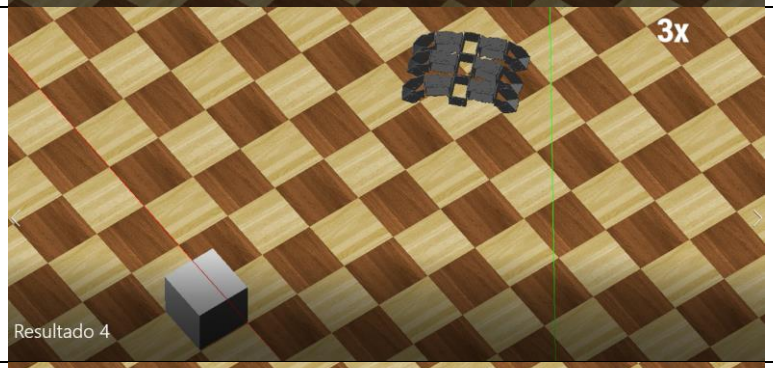
Resultado	Fila Objetivo	Columna Objetivo	N° nodos trayectoria	Número de rectas	Cambios de arquitectura	Tiempo total Minutos -3X
4	35	20	30	8	2	36:10

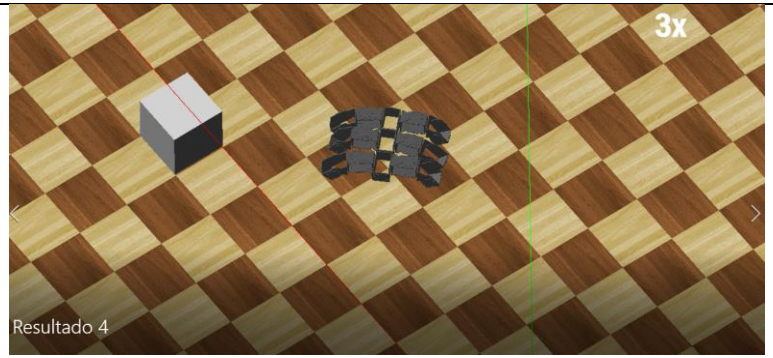


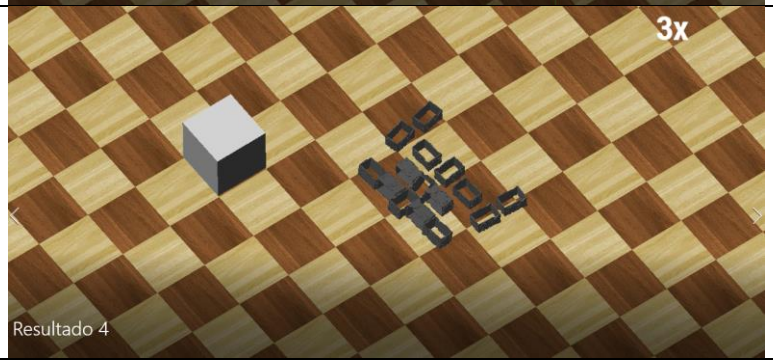
```
RESULTADO ELECCION DE ARQUITECTURA(S) SOBRE LA TRAYECTORIA PLANTEADA:
Recta 1 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:1 - fila/columna: 6, 6
Recta 2 - Pendiente:90, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:2 - fila/columna: 7, 7
Recta 3 - Pendiente:45, Cantidad de Nodos: 2, arquitectura:2, nodo que inicia:3 - fila/columna: 8, 7
Recta 4 - Pendiente:90, Cantidad de Nodos: 4, arquitectura:2, nodo que inicia:5 - fila/columna: 10, 9
Recta 5 - Pendiente:45, Cantidad de Nodos: 5, arquitectura:2, nodo que inicia:9 - fila/columna: 14, 9
Recta 6 - Pendiente:90, Cantidad de Nodos: 10, arquitectura:1, nodo que inicia:14 - fila/columna: 19, 14
Recta 7 - Pendiente:45, Cantidad de Nodos: 5, arquitectura:2, nodo que inicia:24 - fila/columna: 29, 14
Recta 8 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:29 - fila/columna: 34, 19
```

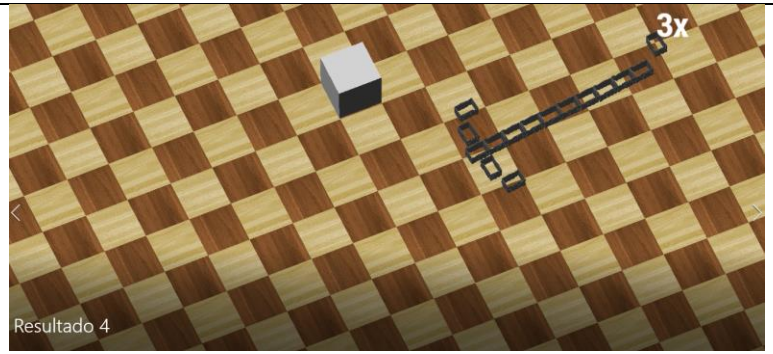

3x
Resultado 4

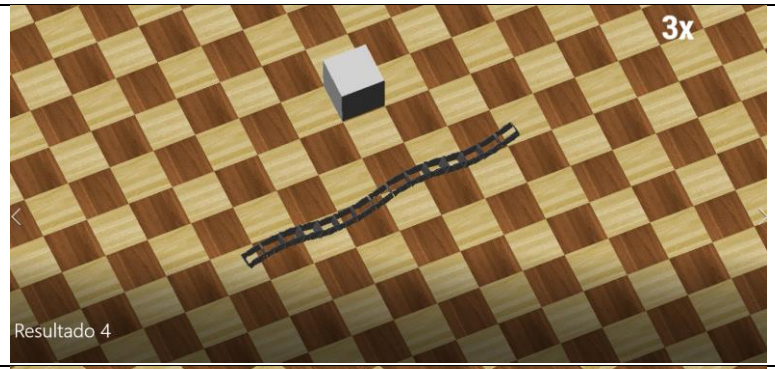

3x
Resultado 4

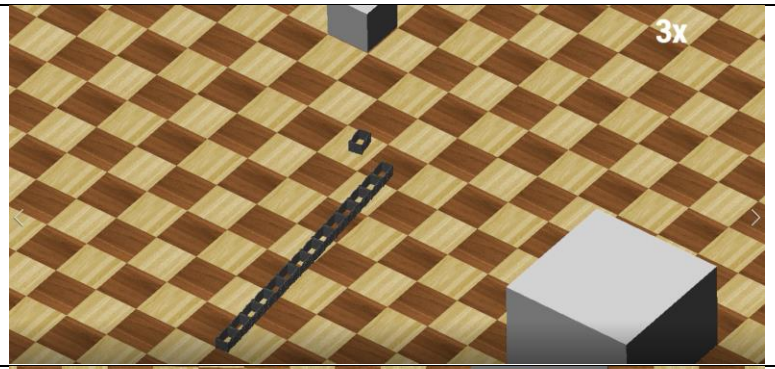

3x
Resultado 4

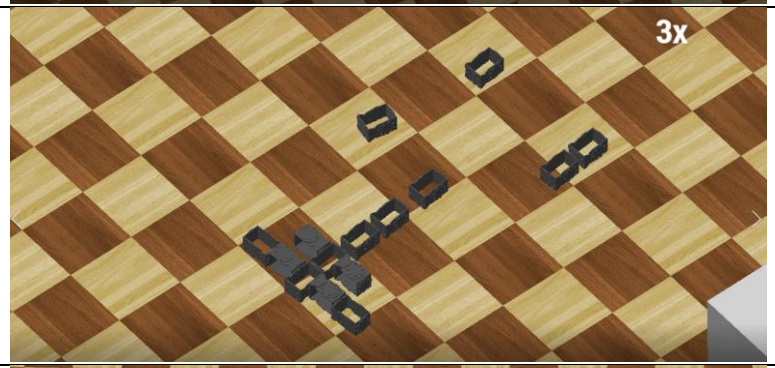

3x
Resultado 4


3x
Resultado 4


3x
Resultado 4


3x
Resultado 4


3x


3x

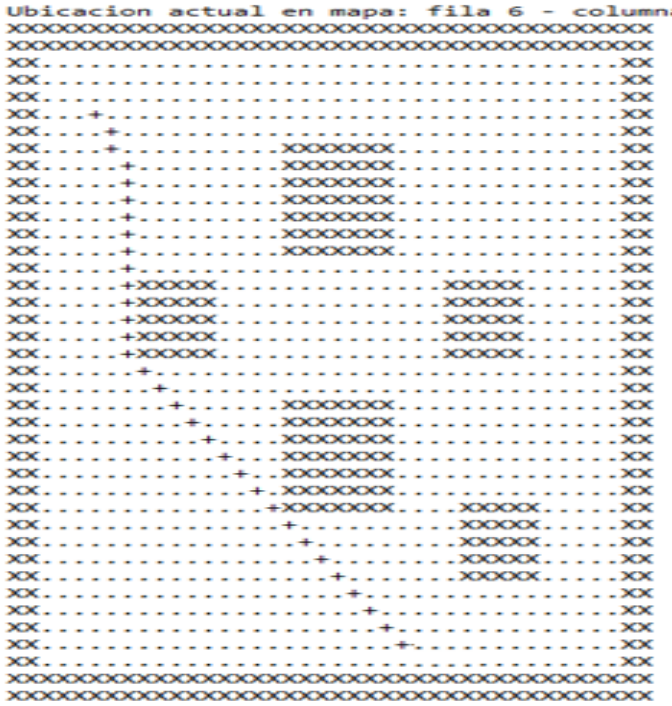

3x

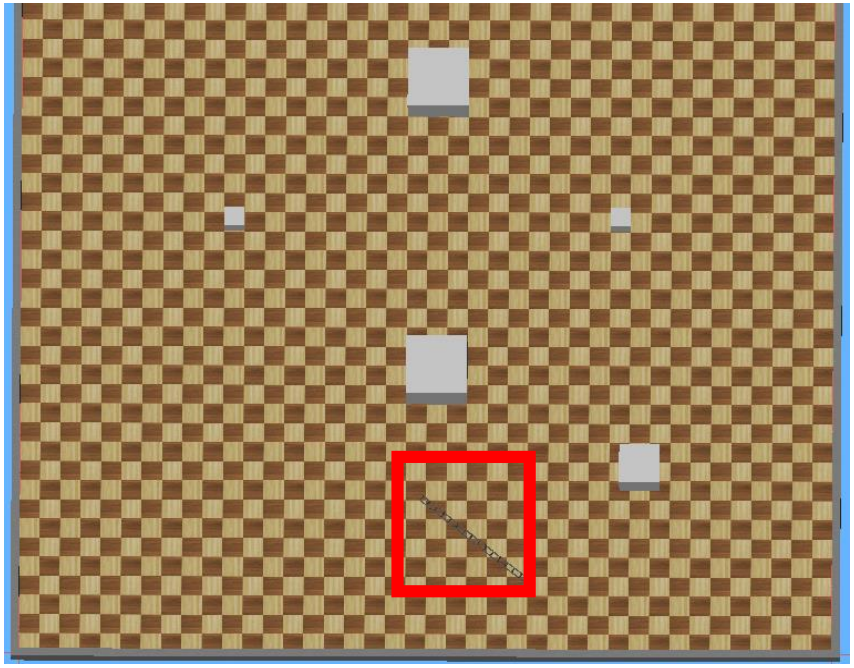

3x
Resultado 4


3x
Resultado 4

Anexo 22. Resultado 5 obtenido – resumen fotográfico

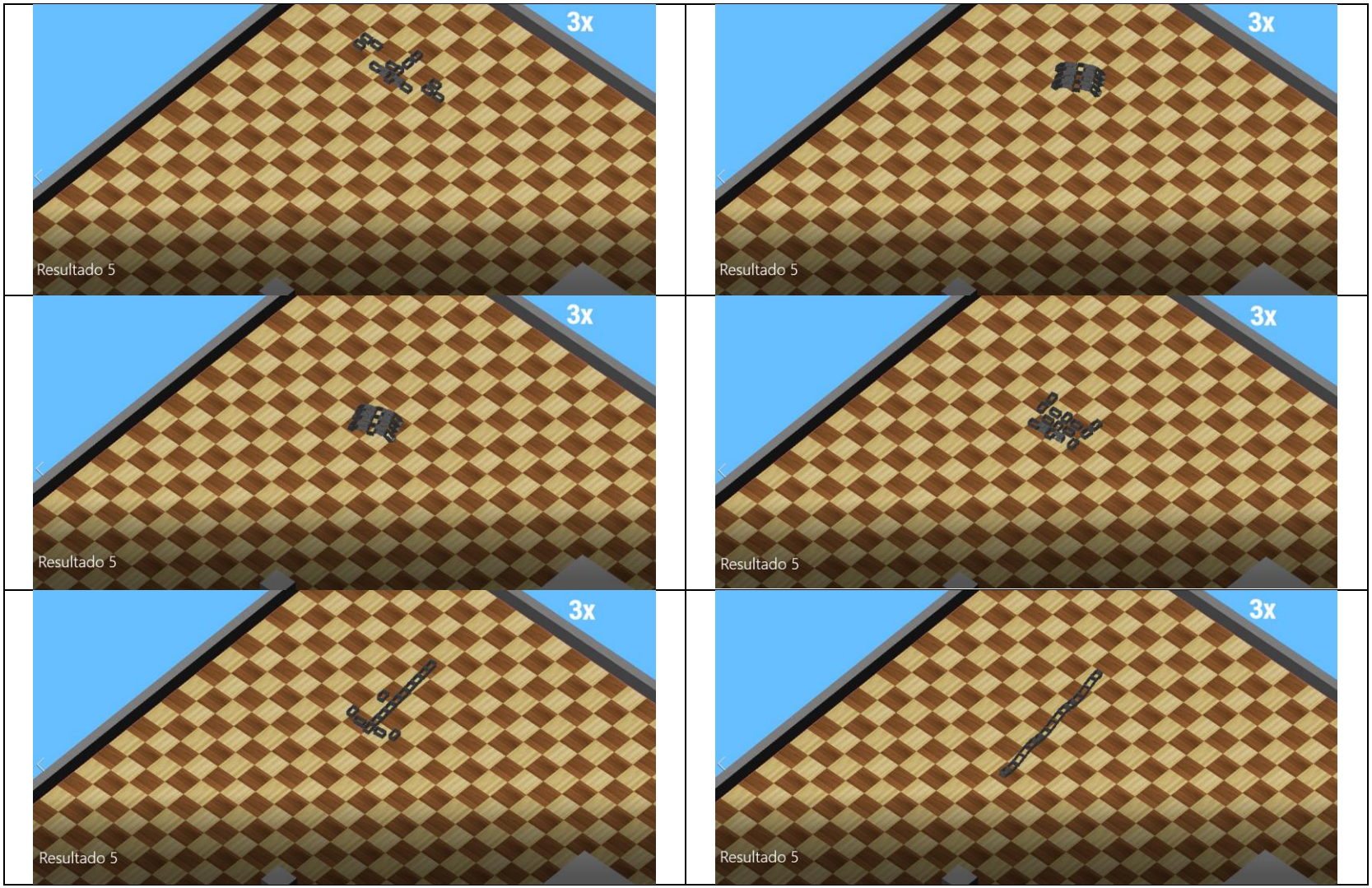
Resultado	Fila Objetivo	Columna Objetivo	N° nodos trayectoria	Número de rectas	Cambios de arquitectura	Tiempo total Minutos -3X
5	37	25	32	6	4	47:09

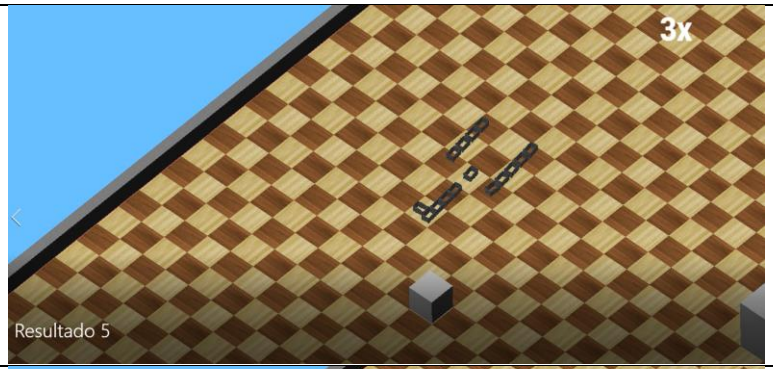


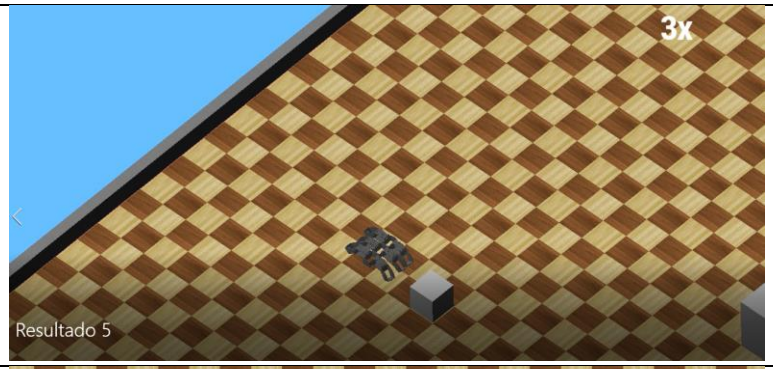


path cost is 32:

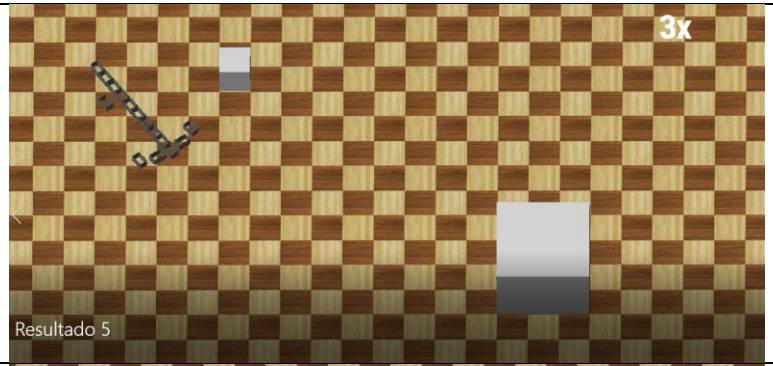
RESULTADO ELECCION DE ARQUITECTURA(S) SOBRE LA TRAYECTORIA PLANTEADA:
Recta 1 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:1 - fila/columna: 6, 6
Recta 2 - Pendiente:90, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:2 - fila/columna: 7, 7
Recta 3 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:3 - fila/columna: 8, 7
Recta 4 - Pendiente:90, Cantidad de Nodos: 11, arquitectura:1, nodo que inicia:4 - fila/columna: 9, 8
Recta 5 - Pendiente:45, Cantidad de Nodos: 1, arquitectura:2, nodo que inicia:15 - fila/columna: 20, 8
Recta 6 - Pendiente:45, Cantidad de Nodos: 16, arquitectura:1, nodo que inicia:16 - fila/columna: 21, 9

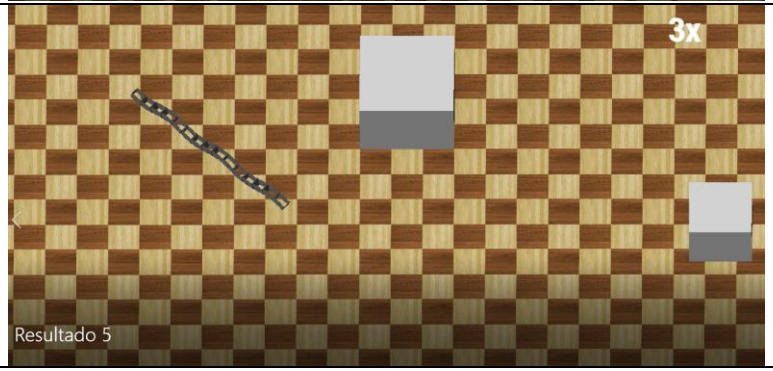

3x
Resultado 5
3x
Resultado 5
3x
Resultado 5
3x
Resultado 5
3x
Resultado 5
3x
Resultado 5

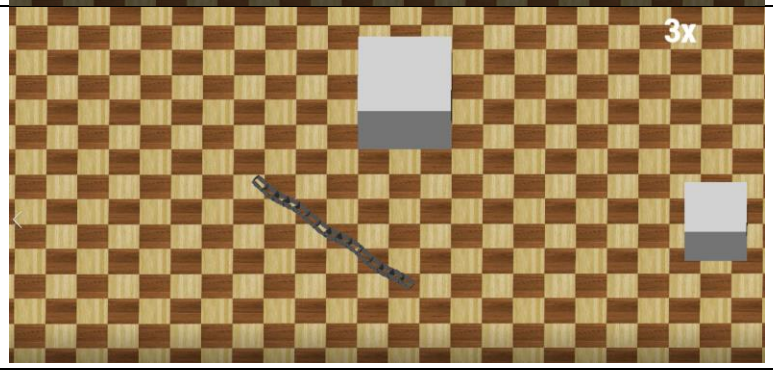

3x
Resultado 5


3x
Resultado 5


3x
Resultado 5


3x
Resultado 5


3x
Resultado 5


3x