Diseño e Implementación de un Robot Didáctico para Resolución de Laberintos

Sandra P. OSORIO, Luisa A. ALVANEZ

Trabajo de grado para optar al título de Ingeniero Electrónico

Manuel J. BETANCUR
Doctor en Ingeniería

Universidad Pontificia Bolivariana
Escuela de Ingenierías
Facultad de Ingeniería Eléctrica y Electrónica
Ingeniería Electrónica
Medellín
2013

DEDICATORIA

A mis padres, Luz Mary Quiceno y Rafael Albanés, y a mi hermana, Elizabeth Albanés, agradeciendo su constante apoyo y motivación durante el desarrollo de su pregrado.

Luisa Alvanez.

A mis padres, Dora Nelly Graciano y Luis Bernardo Osorio, a mi hermanita, Diana Marcela Osorio y a mi novio, Luis Miguel Ariztizábal cuyo apoyo ha sido incondicional.

Sandra Osorio.

AGRADECIMIENTOS

Al profesor Doctor Manuel Betancur por su constante acompañamiento en el proceso de elaboración del presente trabajo de grado. A Luis Miguel Aristizabal, compañero, amigo y asesor, quien brindo su apoyo, sabiduría y experiencia para que este proyecto fuera finalizado. Al laboratorio de ingeniería eléctrica y electrónica y todos sus colaboradores, encabezados por su director Armando, quienes nos facilitaron las herramientas y lugares de trabajo necesarios en el proceso de pruebas. A nuestra familia por su constante apoyo y a nuestros amigos, que siempre fueron una voz de aliento en todo este proceso.

Luisa Alvanez.

Le doy gracias a Dios por haberme acompañado y guiado a lo largo de mi carrera y por permitirme culminar esta etapa de mi vida rodeada de personas que me quieren y me valoran. A mis papás porque me dieron la oportunidad de tener una excelente educación, a mi hermana porque siempre pude contar con ella en todo momento. A mis amigos, sobre todo a Pilar Alzate y Jorge Cardona, por ayudarnos a seguir adelante, porque siempre estuvieron conmigo en mis angustias y en mis alegrías, no solo en este trabajo sino en el transcurso de toda la carrera.

A nuestro director de trabajo de Grado Dr. Manuel J. Betancur por su gran colaboración, por compartir sus conocimientos, por orientarnos, acompañarnos y corregirnos semana tras semana para llegar a ser cada vez mejores ingenieras.

A la universidad por brindarnos siempre el espacio y los materiales necesarios para llevar a cabo todas las actividades propuestas durante el desarrollo de este proyecto. A nuestros jurados Ing. Iván Mora e Ing. Jorge Cock porque siempre que los necesitamos sacaron de su tiempo y compartieron sus conocimientos y experiencias para orientarnos. Le agradezco también al Ing. Mecánico Juan Alberto Ramírez Macías por colaborarnos con el formato del presente documento.

De una manera muy especial agradezco a mi novio Luis Miguel Aristizábal, porque desde que lo conocí no ha hecho sino alegrar mi vida, gracias por todo el amor que me das, por enseñarme todo lo que sabes, por que se que siempre puedo contar contigo, porque desde el comienzo del proyecto me orientaste, acompañaste y confiaste en mi.

Sandra Osorio.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

GLOSARIO

ARDUINO: Es una plataforma de hardware de código abierto, basada en una sencilla placa con entradas y salidas, analógicas y digitales, en un entorno de desarrollo que está basado en el lenguaje de programación *Wiring* [7].

ARQUITECTURA DE CONTROL: Es la manera de organizar un sistema de control. Impone restricciones a los métodos para resolver el problema de control [8].

CICLO DE TRABAJO: Porcentaje de tiempo que permanece en nivel alto una señal binaria periódica [9].

EDUCATRÓNICA: Inclusión de robots, sistemas mecatrónicos y tecnologías de la información y la comunicación, en ambientes educativos con el fin de fortalecer y facilitar la adquisición de conocimientos en las diferentes áreas de la ciencia [10].

FIRMWARE: Bloque de instrucciones de programa para propósitos específicos, grabado en una memoria de tipo no volátil (ROM, EEPROM, FLASH), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo [11].

HARDWARE: Conjunto de los componentes que integran la parte material de un sistema informático [12].

ROBOT: Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas [12].

ROBÓTICA: Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales [12].

ROBUSTO: Fuerte, vigoroso, firme, bien fabricado, difícil de romper [12].

SENSOR: Dispositivo que detecta una determinada acción externa, temperatura, presión, entre otros, y la transmite adecuadamente [12].

SISTEMA AUTÓNOMO: Sistema que puede actuar con un alto grado de independencia y con un mínimo de restricciones por parte de otros sistemas [13].

SOFTWARE: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación [14].

VEHÍCULOS GUIADOS AUTOMÁTICAMENTE: también conocidos como AGV (del inglés *Automated Guided Vehicle*) Sistema no tripulado de transporte de carga normalmente propulsado mediante electricidad almacenada en baterías. Estos vehículos pueden ser programados para recoger una carga en un punto y transportarla hasta otra localización [15].

VISUAL C#: Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft. Diseñado para crear una amplia gama de aplicaciones que se ejecutan en un entorno de desarrollo visual [16].

RESUMEN

En este trabajo de grado se realiza una descripción detallada del proceso de diseño mecánico, electrónico y de software del sistema robótico Ómicron. El robot Ómicron debe solucionar tres tipos de laberinto que hacen parte de la Olimpiada Robótica A+D, de la Universidad Pontificia Bolivariana. Se describe cada una de las etapas por las que se pasó hasta la obtención de un dispositivo que cumpliera las especificaciones de robustez, reprogramabilidad y fácil manejo que se requerían. También se explica en que consiste el software para facilitar la reprogramación del robot: ÓmicronTool, así como una descripción en detalle de los algoritmos implementados en el robot para la resolución de los laberintos.

Palabras clave:

Educación, Robots Móviles.

ABSTRACT

In this final report, a detailed description of mechanical, electronical and software design of the Omicron Robotic System is made. Omicron robot must solve three types of mazes, which are part of the Universidad Pontificia Bolivariana's A+D Robotics Olympics. Every stage that happened before the creation of a device which fulfilled with specifications of reliability, reprogrammability and ease of use is described. Also, OmicronTool software, made to facilitate reprogramation of the Omicron Robot, is explained, and finally, a detailed description of implemented algorithms within the robot for maze solving is made.

Keywords:

Education, Mobile Robots.

INTRODUCCIÓN

La robótica en la actualidad es un área que se considera de gran importancia gracias a que ha permitido muchos avances tecnológicos y a que sus aplicaciones son diversas. Es por esto que se debe incentivar cada vez más a las personas para que se interesen en esta área a temprana edad y se vean cada vez más familiarizados con ella. El robot Ómicron fue realizado con propósitos didácticos específicamente con el fin de brindar un acercamiento al público en general a las áreas del conocimiento de la ingeniería, específicamente la electrónica y la robótica. Será implementado como una herramienta pedagógica que permitirá la creación de un ambiente de aprendizaje lúdico, y así fomentar e incentivar el diseño y la construcción de este tipo de robots.

El semillero del Grupo de Automática y Diseño A+D de la Universidad Pontificia Bolivariana (UPB) viene realizando desde el año 1999 la Olimpiada Robótica A+D, la cuál reúne a estudiantes y aficionados de todo el país para concursar en las diferentes categorías. Uno de los propósitos de este proyecto es promocionar la olimpiada, permitiendo proyectar una imagen sobre los avances en la electrónica y la capacidad que tiene esta institución para desarrollar actividades de tal amplitud y así darle visibilidad.

Específicamente, este trabajo de grado tiene como objetivo diseñar e implementar un robot recolector robusto capaz de resolver tres tipos de laberintos de forma autónoma, que sirva para propósitos didácticos y de visibilidad ante el público general. Se debe asegurar la robustez, el fácil manejo, la capacidad de soportar el uso continuo y de informar el estado de batería. Y por último, se propondrá el reglamento de una nueva categoría de competencia que se llevaría a cabo en la Olimpiada Robótica A+D de la UPB, para la cual se desarrollará un software capaz de generar laberintos aleatorios.

En este documento se presentará un recorrido histórico a cerca de la robótica móvil y su importancia en la educación. También se realizará una descripción detallada del diseño y la construcción del robot Ómicron, la cual contiene información de la parte física, del software utilizado para la programación de las diferentes categorías de laberinto y de los circuitos electrónicos. Por último se presentará el software desarrollado para la categoría de laberinto experto con su respectivo reglamento.

1. MARCO REFERENCIAL

1.1. Antecedentes

1.1.1. Origen de la Idea

La idea surgió ante la necesidad de adquirir un robot didáctico para motivar y promover la participación en los eventos que el grupo de Automática y Diseño A+D forme parte. Uno de estos eventos, y el más importante para este proyecto, es la Olimpiada Robótica A+D organizada cada año en la UPB; Allí el robot servirá para ejemplificar algunos de los objetivos que deben ser cumplidos en las competencias.

1.1.2. Estado del Arte

El hombre siempre ha querido imitar los movimientos y funciones de los seres vivos, es por esto que ha buscado crear máquinas que realicen estas acciones. Para los griegos, estas máquinas se llamaban autómatos, palabra de la cual se deriva la actual autómata, que se refiere a máquinas con capacidad de adoptar la forma y movimientos de los seres vivos. En los siglos XVII y XVIII se crearon dispositivos mecánicos que se asemejan a los robots actuales, creados en su mayoría por relojeros con el objetivo de entretener a la gente. También se crearon autómatas antropomorfos capaces de desarrollar diferentes acciones como: tocar instrumentos, escribir, dibujar y pronunciar ciertas palabras. Paralelamente en Japón se construye una muñeca de comportamiento automático llamada *Karakuri* que se podía mover en una dirección y servir el té; Este es quizás el antecesor más directo a los AGV [17].

Robots Móviles Autónomos. Este tipo de robots, también llamados AGV son dispositivos autónomos capaces de guiarse por distintos entornos de forma automática. El primer robot móvil, Electro-Light-Sensitive Internal-External (ELSIE), se creó en Inglaterra en 1953. Mediante una realimentación mecánica, este robot seguía una fuente de luz sin ningún otro tipo de inteligencia incorporado [18]. En 1968 se creó el robot *Shakey*, el primer robot con cámara incorporada que era capaz de explorar el entorno mediante un mapa tipo rejilla y transportar objetos pesados [18].
En la época de los ochentas se produjo una creciente demanda comercial, debido a que sus aplicaciones cada vez eran mayores. A continuación se mencionan algunos ejemplos:

- *Robots industriales.* En 1988, la empresa Clanning Equipments fabricó el *Robokent*, para labores de limpieza en espacios amplios. Este robot posee un sistema de detección tipo sonar y un control automático que le indica la ruta establecida. Éste fue el robot industrial de más éxito en Norteamérica [19].
  Actualmente, la empresa iROBOT trabaja en ir mejorando cada vez más su robot aspirador *Roomba* creado en el 2003. Este robot cuenta con numerables sensores para detectar paredes, muebles y lugares de mayor suciedad. Su versión más reciente tiene una pared virtual que le ayuda mejorar la navegación [20].
- *Hospitales.* La empresa Helpmate Robotics Inc. fabricó un robot enfermero llamado *Helpmate*, que recorre los corredores de los hospitales, y les lleva la comida a los pacientes. Posee un sistema de navegación y ubicación por sonar, y sensores de ultrasonido para detectar obstáculos [19].

- *Investigación espacial.* En 1974 la NASA inicia un programa para la creación de plataformas capaces de reconocer ambientes donde el hombre no puede llegar, es así como surgió *Mars Rover*, que tiene un brazo mecánico, cámaras, un láser, y una brújula giroscópica para localizarse [19]. Actualmente desarrollan, junto con General Motors, un robot llamado *Robonaut2* (R2), un robot astronauta que tiene la apariencia de humanoide de la cintura para arriba. Es manipulado a distancia, sus manos le permiten hacer labores de precisión, posee visión dual y su parte inferior se puede acoplar a un vehículo de exploración espacial llamado *Centauro2*. En febrero del 2011 R2 fue lanzado al espacio, dando señales de "vida" en septiembre de ese mismo año [21].
- *Seguridad.* La empresa Demming Mobile Robotics creó el robot de seguridad *Denning Sentry*, el cual detecta intrusos y patrulla en cualquier instalación. Al detectar que sus baterías se están agotando, se dirige él sólo a la estación de carga. Tiene micrófono, una cámara de TV y transmisores inalámbricos para enviar la información a la estación [19].

Otros campos en los que se destacan actualmente los robots móviles autónomos son: Actividad militar, fabricación de juguetes, extintor de incendios y de propósito didáctico. Este último se ampliará a continuación.

Robótica Pedagógica. El carácter multidisciplinario de la robótica, que reúne áreas del conocimiento como la mecánica, la electricidad, la electrónica, la informática y la inteligencia artificial; la hace una herramienta didáctica de gran valor, propiciando un ambiente tecnológico en el cual el estudiante puede adquirir inductivamente los conocimientos científicos, a la vez que los relaciona entre sí para llegar a la solución de los problemas que se le presentan. La robótica didáctica conduce así a la apropiación de los conceptos aprendidos y motiva al estudiante a la participación activa en el proceso pedagógico, propiciando el desarrollo de competencias tales como la autonomía, la iniciativa, la responsabilidad, la creatividad, el trabajo en equipo, la autoestima y el interés por la investigación [10]. En la actualidad existen en el mercado kits especiales para la robótica pedagógica, listos para ser usados en las aulas de clases de niños y adolescentes. Entre éstos se encuentran:

- *LEGO Education.* Referente mundial en tecnología y robótica, con contenidos y actividades guiadas para alumnos y profesores durante todo el proceso educativo, desde la educación primaria hasta la universidad [22].
- *Robotis OLLO Education.* Es un sistema de iniciación a la robótica educativa a partir de 7 años, flexible, escalable, programable y educativo para diseñar y construir robots, despertando el interés por la ciencia y la tecnología [22].
- *FischerTechnik.* Es un sistema de construcción modular flexibe y escalable utilizado para jugar, modelar y simular con realismo sistemas mecatrónicos. Es de gran ayuda a la educación, ya que replican los elementos utilizados en máquinas y dispositivos reales [22].
- *MOway Educatio.* Es una herramienta pedagógica que permite al estudiante descubrir la programación a través de un software sencillo e intuitivo con el que se controla el robot y sus dispositivos de entrada y salida, desarrollando desde un primer momento sus propios programas y sus propios circuitos electrónicos [22].

- *Robotis BIOLOID.* Es un sistema de kits de robots educativos escalables y reconfigurables con el que se aprende lo básico de estructuras y principios de las articulaciones de robots. Es un kit para todas las edades, ya que también sirve como un curso completo de lenguaje C.[22].
- *VEX IQ.* Es una plataforma de robótica diseñada para transformar el aprendizaje de las Ciencias, Tecnología, Ingeniería y Matemáticas de estudiantes desde los 8 años, ya que es intuitiva y no requiere herramientas. Fomentan el trabajo en equipo, el liderazgo y la resolución de problemas entre grupos.[22].
- *Parallax.* Esta empresa ha estado comprometida con la educación, ya que proporciona a estudiantes e ingenieros el material que necesitan para aprender a programar microcontroladores, proporciona también una amplia variedad de textos y kits de robótica [23].

Competencias de Robótica. En los últimos años las competencias de robótica han alcanzado gran popularidad, debido a la importancia que en la actualidad se le da al conocimiento y el desarrollo de las nuevas tecnologías, motivando a la creación y fortalecimiento de las facultades de electrónica, automatización mecatrónica y control de las universidades en todo el mundo, incentivando así a la participación en las competencias que se organizan. A pesar de la complejidad que conlleva la creación de un robot, cada vez la participación en las competencias comienza en edades más tempranas, gracias a la inclusión de la robótica didáctica en la instituciones educativas de todo el mundo, llevando a la creación de competencias especialmente enfocadas para aquellos adeptos que dan sus primeros pasos en el mundo de la robótica. A continuación se mencionan algunas de las competencias a nivel mundial.

- *Roborodentia.* Se lleva a cabo cada mes de Abril en la Universidad politécnica Estatal de California, Calpoly, EUA desde hace 18 años, y en cada versión el reto que deben afrontar los participantes cambia. Actualmente el reto consiste en ubicar dos robots en un área parecida a una cancha de futbol, en la cual cada robot debe recoger el máximo de pelotas de tenis posibles y lanzarlas en la portería del oponente [24]. En sus inicios, la competencia consistía en que los robots debían resolver un laberinto y recoger dos pelotas de tenis para depositarlas en el lugar definido como la meta (ver Figura 1). De ahí que la categoría de laberinto de la Olimpiada Robótica A+D se haya basado en esta competencia.

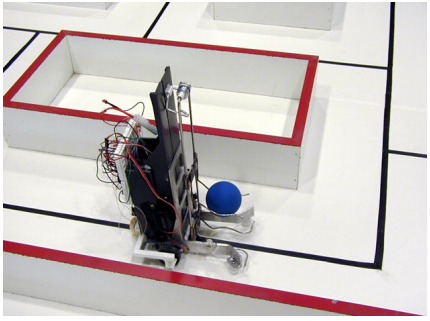


Figura 1. Competencia Roborodentia en el año 2000 [1]

- *Intelligent Ground Vehicle Competition (IGVC).* Llevada a cabo cada año desde 1992 en la Universidad Oakland, EUA. Consisten en desarrollar un vehículo terrestre autónomo que debe seguir una carrera con complicados obstáculos, y deben cumplir ciertos requerimientos de diseño y movilidad. [25].
- *Micromouse.* Estas competencias consisten en crear pequeños robots que resuelven un laberinto específico en el menor tiempo posible. Un robot que participe en este concurso es denominado *micromouse*[26]. Se han celebrado desde 1979 y se llevan a cabo en países de todo el mundo. Dos de las más importantes son: *All Japan Micromouse Contest* e *IEEE Micromouse competition.*
- *AUVS Ground Robotics Competition.* El objetivo de esta competencia es construir un vehículo completamente autónomo capaz de navegar en torno a una pista agreste delimitada con líneas blancas. Existen múltiples obstáculos en el camino que el robot debe evitar.[26].
- *International Fire-Fighting home robot contest.* llevada a cabo cada año desde 1994 en la universidad Trinity College, EUA. La meta de esta competencia es construir un Robot controlado por computadora que pueda desplazarse a través de una pista que simula ser una casa, encontrar una vela encendida y apagarla en el menor tiempo posible. [26].
- *FIRST LEGO League (FLL).* Competencia internacional de tecnología para jóvenes que se lleva a cabo desde 1998 en más de 70 países, involucrando a más de 250.000 jóvenes en el mundo. La competencia consiste en emplear el sistema de robots LEGO Mindstorms NXT para resolver los desafíos que se plantean anualmente. [27].
- *RoboGames.* Es reconocida por el libro de Guinness Récords como la competencia de robots más grande del mundo y está inspirada en los juegos olímpicos que celebran los humanos. Acoge más de 50 eventos diferentes, entre ellos: fútbol, sumo, combate, lucha libre, resolución de laberintos, extinción de incendios, etc. Se celebra cada año desde 2004 en San Mateo, EUA [28].
- *Eurobot.* El concurso inició en 1998 y se realiza anualmente en Europa. La competencia consiste en que los equipos deben construir robots autónomos, que recogen los elementos en una determinada área de juego, dependiendo de las reglas que son renovadas cada año [29].
- *Robot al parque.* Este concurso se lleva a cabo en Bogotá desde hace ocho años, busca convocar a la comunidad académica del área de la robótica a resolver los problemas presentados por el comité organizador: Racing Ring UD (Robots seguidor de línea y pared) y Recolector de Café [30].
- *Latin America robotics Competition (LARC).* Se llevó a cabo en Perú del 21 al 27 de Octubre del 2013. Esta competencia está dividida en diferentes categorías: *IEEE Latinoamérica*: SEK (Robots fabricados sólo con kits educativos), *RoboCup Latin-American*: Robots de pequeño tamaño jugadores de futbol, Humanoides, simulaciones en 2D y 3D de partidos de fútbol.Y por último *RoboCup Junior*: dirigida sobre todo a niños, incluye robots rescatistas, robots jugadores de fútbol y presentaciones de robots bailarines [31].

Robots de resolución de laberintos. A nivel mundial, las competencias que implican la creación de un robot para resolver

laberintos son muy populares, y en algunas de ellas, no solo se trata de resolver el laberinto en el menor tiempo posible, sino que también se debe cumplir con algunos objetivos como apagar incendios, o recolectar objetos. En Colombia estas competencias se han venido realizando desde hace muchos años. Casos como el robot *Caterpillar*, que ha participado en la Olimpiada Robótica A+D, ha sido un caso de estudio para este trabajo debido a su excelente desempeño. *Caterpillar* fue construido con láminas de acrílico, y posee un mecanismo tipo pala para la recolección de pelotas de tenis de campo. También a nivel nacional, Cubas et al. [32] desarrollaron dos robots que cumplen objetivos similares a los de este prototipo [32]. Y *Mercaba* [4], que por más de 10 años ha ido evolucionando y obteniendo múltiples méritos por su buen desempeño. Estos son sólo algunos de los casos de éxito de robots laberinto, sin embargo éstos robots no están orientados al uso didáctico.

1.2. Arquitectura de control

El primer paso para diseñar una arquitectura de control adecuada es definir los requerimientos del problema a solucionar. Ollero [2] propone ciertos requerimientos generales para arquitecturas en robótica: aspectos como la programabilidad, grado de autonomía y fiabilidad son de vital importancia en este proyecto, ya que este robot es una plataforma educativa, por lo que debe poder resolver por sí mismo distintos casos de un mismo tipo de problema a criterio del usuario de manera satisfactoria, además debe ser robusto tanto en hardware como en software. Características como adaptabilidad, eficiencia y capacidad de evolución no son prioritarias en este proyecto, no queriendo decir que no se tengan en cuenta. Una vez establecidos los requerimientos de la arquitectura, se puede indagar acerca de los modelos de referencia y arquitecturas de control que se adecúen al problema.

1.2.1. Arquitectura de Referencia

Existe una gran variedad de arquitecturas para robots móviles, sin embargo muchas de estas se basan en modelos de referencia muy generalizados que definen la estructura básica de cada sistema de control. Se escogieron dos modelos que son altamente compatibles con los requerimientos del proyecto y se presentan a continuación:

Arquitectura Centralizada. En este tipo de arquitecturas, las tareas del sistema son llevadas a cabo por módulos, e.g., percepción, planificación de trayectorias, control de movimientos, los cuales se comunican con un módulo central que tiene toda la información relevante del robot y su entorno (Figura 2).

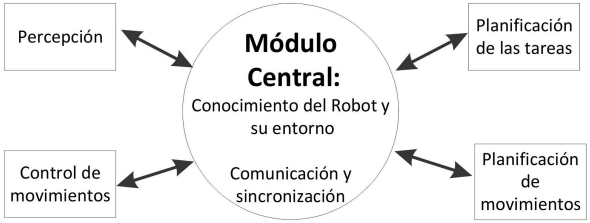


Figura 2. Arquitectura centralizada [2]

Este módulo se encarga de comunicar y sincronizar todos los elementos del sistema.

Arquitectura Descentralizada. A diferencia de la arquitectura centralizada, este tipo de arquitectura no posee un módulo central con las características del anterior, sino que los módulos se comunican entre sí según lo requieran, y se define la periodicidad con la que se realizan estas interacciones (Figura 3).

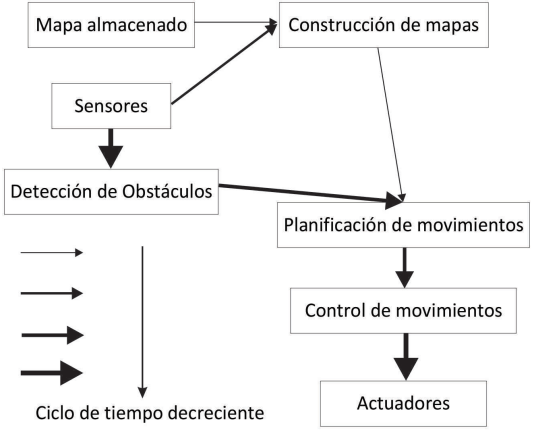


Figura 3. Arquitectura descentralizada [2]

1.2.2. Arquitectura Implementada

Según la teoría anterior, la arquitectura que más se acomoda a la de este proyecto es la arquitectura descentralizada, debido a que el robot tendrá tareas que serán mucho más frecuentes que otras, y la comunicación entre funciones será según se requiera, no hay un módulo general que posea toda la información y se encargue de sincronizar a los demás.

En la Figura 4 se puede observar la arquitectura implementada para este trabajo de grado. Ésta presenta cambios respecto al modelo de referencia propuesto por Ollero [2] para que se ajustara a los requerimientos del proyecto.

El modelo consta de dos partes: la primera parte se desarrolla a nivel de software, se implementará un programa con interfaz gráfica que servirá como herramienta para la reprogramación de las rutas de laberinto, aquí se van a ejecutar unas funciones de planeación de ruta, constructor y autocompletado de laberintos que se explicarán en detalle en la sección 4.

La segunda parte del modelo se encuentra en el robot (hardware). La información del mapa del laberinto y la ruta que debe seguir generadas por el programa, se enviarán al robot por medio de módulos de comunicaciones. Posteriormente esta información la recibe el módulo de planificación de movimientos, el cuál se encarga de verificar que la ruta seguida por el robot sea la correcta usando funciones de planeación y de monitoreo de su ubicación. Este módulo a la vez utiliza otros módulos que están más ligados a controlar el hardware del robot, e.g., el módulo para control de movimientos, que contiene la función de seguidor de linea, la de control de velocidad, que se encarga de mantener la velocidad constante,

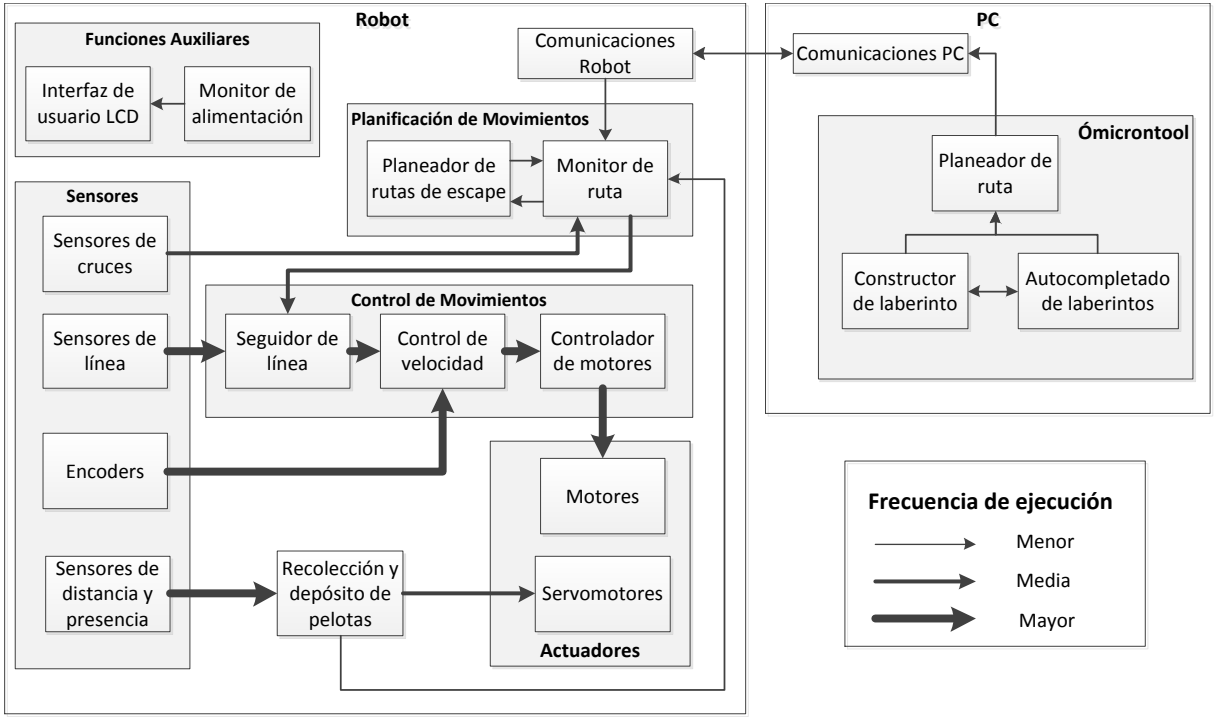


Figura 4. Arquitectura Implementada

y la función del controlador de motores que es el hardware para manejar los actuadores.

Otro de los módulos a nivel de hardware es el de sensores, por medio del cual se adquiere información del entorno. Existen varios sensores para ejecutar tareas diferentes, como los encoders para el control de velocidad, los sensores de línea y cruces, y sensores de distancia y presencia. Estos últimos son los que realimentarán el módulo de recolección y depósito de las pelotas, que a su vez determinarán cuando deben funcionar los servomotores. El último módulo que presenta la arquitectura de control es el de funciones auxiliares, en el cual se realiza un monitoreo de la alimentación del robot que se presentará al usuario por medio del visualizador de cristal líquido, más conocido por sus siglas en inglés LCD (*Liquid Crystal Display*).

2. DISEÑO MECÁNICO

En el desarrollo del proyecto se realizaron diferentes diseños mecánicos partiendo del bosquejo de una idea inicial que fue evolucionando hasta la obtención del modelo final. Este proceso de diseño se detalla en la Figura 5, donde se muestran los modelos en 3D realizados en cada etapa. Estos modelos fueron realizados usando el software libre para modelado en 3D *Google SketchUp*.

En la Figura 5a se puede identificar que el diseño que se tenía al inicio del proyecto es notoriamente diferente a los modelos presentados en etapas posteriores. Éste contaba con un mecanismo de recolección de pelotas que consistía en una pala que desde un segundo nivel empujaba la pelota hasta un contenedor ubicado en el interior del robot, el cual contaba con una rampa que facilitaba la salida de las pelotas una vez se abriera una puerta ubicada en la parte posterior del robot. Este modelo se descartó debido a que, después de la observación del funcionamiento de diferentes robots en las Olimpiadas Robóticas A+D [33], se identificaron dificultades a la hora de guiar la pelota al contenedor y requería que el robot diera un giro de 180° para poder depositar las pelotas en el depósito, problemas que no se presentaban o se reducían notoriamente con otros diseños.

Se determinó entonces que los métodos de recolección más exitosos correspondían a una pala en forma de tenedor que recogía la pelota desde la parte inferior (Figura 6a) y una pinza que al cerrarse atrapaba la pelota (Figura 6b). Para ambos casos el almacenamiento de pelotas se debe ubicar en la parte superior del robot. Debido a que representaba mayor facilidad mecánica manteniendo buenos resultados en su operación se optó por usar la pala como método de recolección y un depósito con un mecanismo que permitía que las pelotas pudieran ser llevadas a la meta desde la parte frontal del robot.

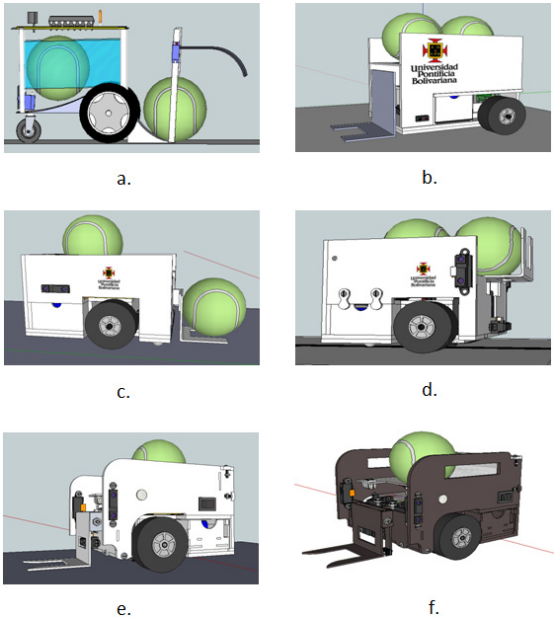
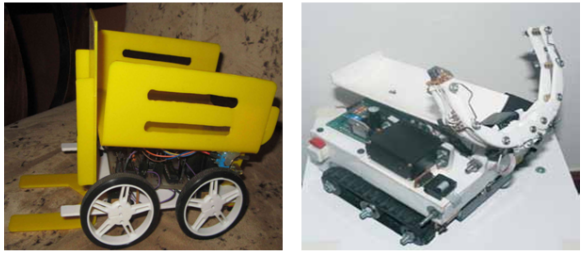


Figura 5. Modelos 3D realizados en el proceso de diseño



a. b.

Figura 6. Métodos de recolección con mayor éxito en Olimpiadas Robóticas A+D: a) Robot Caterpillar [3]; b) Robot Mercaba [4]

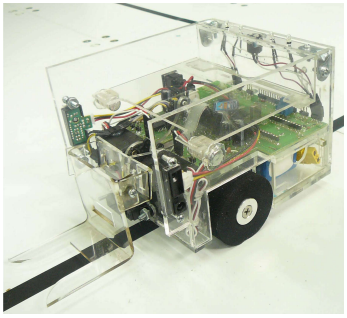


Figura 7. Primer prototipo del robot Ómicron

Una vez elegido el método de recolección y almacenamiento se pudo realizar un primer modelo (Figura 5b). A medida que se avanzaba en el diseño del robot se determinó que era más conveniente el uso de tracción delantera (Figura 5c) para una mejor distribución del centro de masa. Luego al determinar la ubicación y tipo de sensores se obtuvo un primer modelo completo del robot (Figura 5d) que podía ser ensamblado, obteniendo así el primer prototipo funcional que se muestra en la Figura 7. Gracias a este primer prototipo se pudo realizar ajustes en las medidas para garantizar que el robot se moviera de manera fluida dentro del laberinto, se mejoraron los mecanismos de recolección y depósito de pelotas, como también el mecanismo que mantiene en su lugar el cajón interior que contiene la batería (mecanismos que serán explicados en detalle posteriormente), obteniendo así un nuevo modelo (Figura 5e) caraterizado por su modularidad.

El modelo final se puede observar en la Figura 5f.

2.1. Elementos y Sistemas

El robot Ómicron está compuesto por varios mecanismos que en su conjunto permiten que éste cumpla con los objetivos y especificaciones planteados en los reglamentos de la categoría laberinto de la Olimpiada Robótica A+D [33]. A continuación se describe cada uno de ellos.

2.1.1. Sistema de locomoción

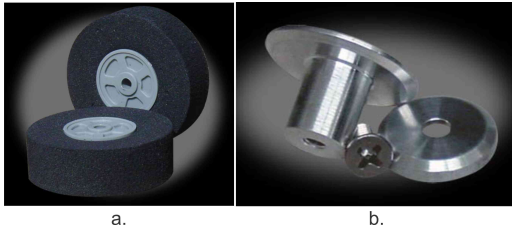
El sistema de locomoción debe facilitarle al robot el desplazamiento sobre una superficie horizontal de madera aglomerada con un recubrimiento de formica blanca estándar con presencia de desniveles y, al mismo tiempo, debe tener la capacidad de subir y bajar una rampa de 16° de inclinación. Para que el robot pueda moverse de manera fluida en un espacio bajo estas condiciones se determinaron los componentes y mecanismos que se presentan en las siguientes subsecciones.

Motorreductores y Ruedas. Para el sistema de tracción era indispensable el uso de motores que le brindaran al robot el torque necesario para subir la rampa presente en algunos de los laberintos a resolver, e igualmente tuvieran un tamaño reducido que permitiera la realización de un diseño lo más compacto posible. El motorreductor que se muestra en la Figura 8 cumple con las características anteriormente planteadas, además de tener un consumo de corriente a 12 V en frenado de 1.5 A aproximadamente, dato importante a la hora de realizar el diseño electrónico del robot. A continuación se resumen las características principales de los motorreductores utilizados.

- Rango de operación: 4.8 V - 14.8 V
- Velocidad a 12 V: 120 RPM
- Torque a 12 V: 5.8 kg-cm
- Corriente en frenado: 1500 mA
- Tamaño (L×A) = 41×16 mm
- Peso: 23.7 g

Se buscó que las ruedas brindaran la adherencia necesaria para subir y bajar la rampa sin deslizarse por ella, y que

Figura 8. Motorreductor Implementado



a. b.

Figura 9. Ruedas y *Hubs* [5]

su tamaño garantizara que los sensores detectores de línea ubicados debajo del robot estuvieran a una distancia adecuada para facilitar su lectura sin llegar a tocar la superficie del suelo. Se escogieron entonces ruedas de compuesto de espuma de caucho de un diámetro de 5.7 cm como las que se muestran en la Figura 9a.

Para poder asegurar una fijación firme entre los motores y las ruedas fue indispensable el uso de *Hubs*, (Figura 9b).

En la Figura 10 se puede ver un modelo en 3D del sistema compuesto por motoreductor, hub y rueda. Además es posible observar el encoder, que está pegado al hub y permite implementar un control de velocidad para los motores. Esta última característica del sistema será profundizada en la sección 3.

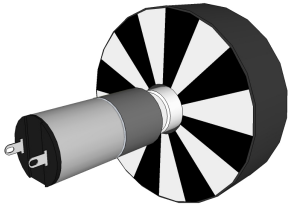


Figura 10. Modelo en 3D del sistema de locomoción

Rueda de Apoyo. Los motoreductores ubicados en la parte delantera del robot, como se ve en la Figura 12, son los únicos elementos que le brindan la capacidad de moverse dentro del laberinto, por lo tanto es necesario la presencia de un punto de apoyo en la parte posterior del robot, que al mismo tiempo tenga la capacidad de rodar para permitir el movimiento en la dirección definida por los motorreductores. Para esto se usó una rueda de apoyo o rueda loca como la mostrada en la Figura 11, que consta de una bola de metal de 1.34 cm de diámetro, una carcasa negra que sujeta la bola mediante dos

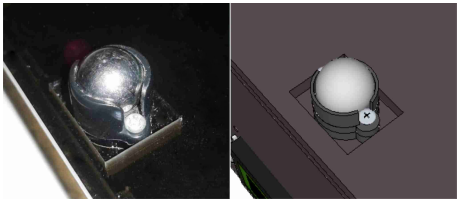


Figura 11. Rueda de apoyo o rueda loca

tornillos, y dos espaciadores de distinto tamaño que permiten

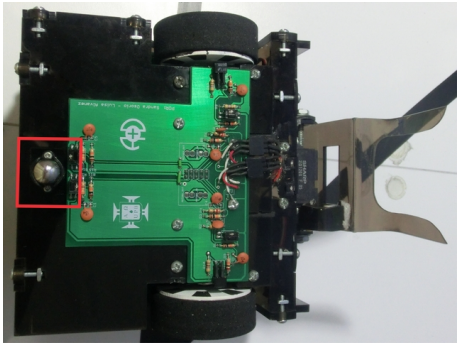


Figura 12. Ubicación de la rueda loca en la parte posterior del robot

posicionarla a diferentes alturas para así asegurar que el robot se encuentre correctamente nivelado. La ubicación de la rueda loca se puede observar en la Figura 12.

Piezas de sujeción de motorreductores. Para asegurar que los motores se unieran rígidamente con el chasis del robot fue necesario el diseño de unas piezas de sujeción hechas a medida que impidiera que los motores se movieran. Estas piezas de acrílico, con un espesor de 1.6 cm una y de 0.8 cm la otra, tienen un orificio en su centro en el cual se ubica el motor y está atravesado en uno de sus lados por un perno, que al ser apretado permite la unión entre el chasis y el motor pero sin impedir que sean extraídos posteriormente en caso de que sea necesario reemplazarlos o repararlos. En la Figura 13 se muestra el modelo con sus dimensiones de esta parte del sistema.

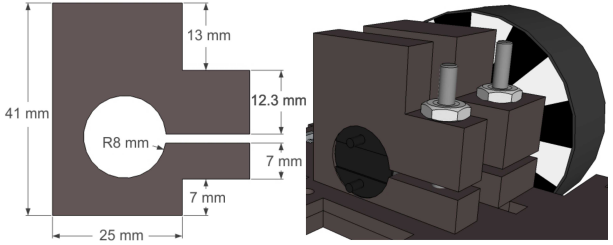


Figura 13. Piezas de sujeción de motorreductores

2.1.2. Mecanismo de recolección y almacenamiento de pelotas

Para todas las categorías de laberinto que Ómicron debe resolver, es un objetivo principal que el robot pueda identificar, recolectar y transportar dos pelotas de tenis ubicadas a lo largo de su recorrido, y luego puedan ser depositadas en el depósito. Para esto es indispensable el diseño de mecanismos que permitan el proceso de recolección y almacenamiento de las pelotas, buscando que representen la mayor facilidad y eficiencia en su implementación. Es así como se llega al diseño de los mecanismos que se muestran en las próximas subsecciones.

Pala. El mecanismo implementado para la recolección de las pelotas está conformado por una pala en forma de tenedor, como se muestra en la Figura 14. La pala está constituida por tres piezas de acrílico (ver Figura 15) para permitir que tenga el comportamiento mecánico deseado y facilitar su fabricación sin perder rigidez.

Figura 14. Pala recolectora de pelotas vista lateral (izquierda) y vista superior (derecha)

La pieza 3 es la que permite la conexión entre la pala y el servomotor que realiza el movimiento necesario para la recolección de las pelotas, en la Figura 16 se ilustra este movimiento. Esta pieza se adhiere a la pieza 2 permanentemente usando cloruro de metileno, para luego ser unida con la pieza 1 por medio de un tornillo sin fin asegurado con tuercas en ambos extremos de la pala.

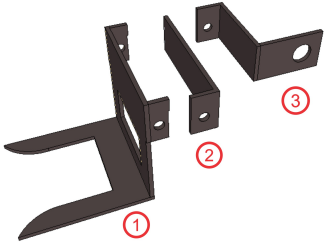


Figura 15. Piezas que conforman la pala

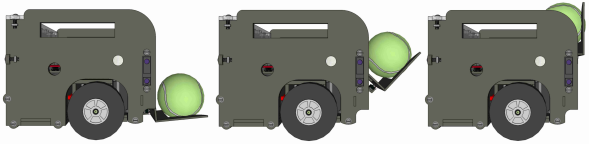


Figura 16. Movimiento realizado por la pala en el proceso de recolección de las pelotas

La pieza 1 es la que recoge la pelota y tiene un orificio rectangular en el centro para permitir la correcta lectura del sensor de distancia.

Mecanismo para Almacenamiento de Pelotas. Después de recoger las pelotas, éstas deben ser almacenadas hasta que el robot llegue a la meta final. Para esto se diseñó un contenedor ubicado en la parte superior, donde se pueden colocar las pelotas después de ser recolectadas y al mismo tiempo pueden ser depositadas en la meta final sin necesidad de que el robot realice un giro de 180°. Se puede ver la manera en que llega la pelota al contenedor en la Figura 17.

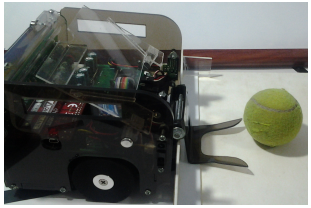


Figura 17. Forma de llevar la pelota al contenedor para ser almacenada

El contenedor del robot está formado por tres piezas de acrílico cristal que se muestran en mayor detalle en la Figura 18.

La pieza 1 tiene forma de ese “s” y se sujeta a las paredes del robot mediante tornillos. Su función es darle soporte al contenedor sin que el robot pierda la cualidad de ser modular.

El piso móvil del contenedor se identifica como la pieza 2, que tiene la capacidad de cambiar su inclinación gracias a un par de ejes atornillados a las paredes laterales del robot.

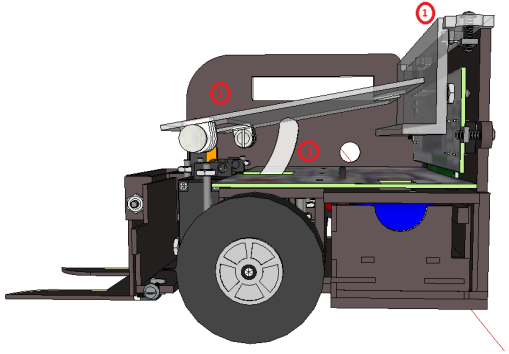


Figura 18. Partes que conforman el contenedor de pelotas

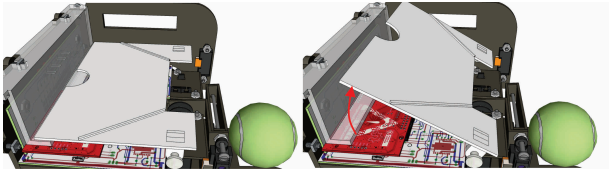


Figura 19. Piso móvil del contenedor de pelotas

En la Figura 19 se puede observar la pieza 2 con mayor detalle, y se puede identificar que posee una guía en forma de semicírculo utilizada para facilitar el movimiento manual del piso del contenedor en caso de ser necesaria la revisión de alguno de los componentes internos del robot. De igual manera son visibles la guías para las pelotas ubicadas a ambos costados del piso, con estos se busca que las pelotas se distribuyan correctamente dentro del contenedor.

Para el movimiento automatizado del piso del contenedor se usa la pieza 3, ver Figura 18, la cual se encuentra unida al eje del servomotor cuyo cambio de posición se ve reflejado en un movimiento hacia arriba y abajo en la pieza, empujando o dejando caer el piso del contenedor. La inclinación del piso lograda gracias a este movimiento facilita la salida de las pelotas para poder ser depositadas en la meta final.

Servomotores. Para todos los mecanismos descritos en esta sección es fundamental el uso de servomotores, uno para el movimiento de la pala, y otro para lograr la inclinación del contenedor. El servomotor elegido para el cumplimiento de estos objetivos es de referencia Futaba S3004 y sus principales características se presentan a continuación.

- Dimensiones: 40 × 20 × 36 mm
- Torque a 4.8 V: 3.2 kg-cm
- Velocidad a 4.8 V: 0.23 s/60º
- Peso: 37 g

Para el correcto funcionamiento de los servomotores es fundamental que éstos se encuentren fijos rígidamente al chasis pero sin olvidar la facilidad de desarmado del robot y del reemplazo

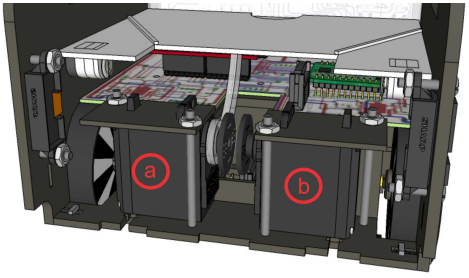


Figura 20. Ubicación de los servomotores para el contenedor (a) y la pala (b) y sistema de sujeción

de piezas en caso de ser necesario. Es por esto que se diseñó un sistema de sujeción de servomotores como los que se muestran Figura 20.

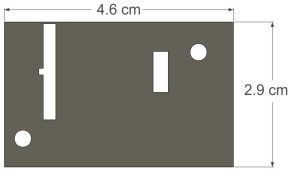


Figura 21. Piezas de sujeción de los servomotores

Este mecanismo de sujeción consta de dos piezas de acrílico en las cuales encajan los servomotores y son asegurados por dos pernos. En la Figura 21 se muestra la forma de la pieza de sujeción y sus dimensiones.

2.2. Chasis del Robot

Para que la estructura mecánica presente un buen desempeño, se debe escoger adecuadamente los materiales a emplear. Es por esto que con base en lo encontrado durante la revisión del estado del arte acerca de robots similares se determinó que el material más adecuado es el acrílico, debido a que es rígido, liviano y permite realizar cortes a laser con una resolución milimétrica a un costo relativamente bajo, lo que facilita el modelamiento de cada una de las piezas.

Debido a que uno de los objetivos de Ómicron es servir para propósitos didácticos, es muy importante que el robot se pueda armar y desarmar a criterio del usuario, ya sea para mantenimiento o reposición de piezas o simplemente como un proceso educativo. Uno de los problemas presentados en la etapa inicial con el diseño del robot fue que no se tuvo en cuenta este aspecto, ya que el chasis no se podía desarmar, es decir, todas las paredes de acrílico, incluido el piso base eran una misma pieza, ya que estaban construidas con láminas de acrílico fundidas unas a otras con cloruro de metileno (Figura 7). Es por esto que se tuvo que rediseñar el chasis de modo que fuese modular. En la Figura 22 se puede observar la segunda versión de Ómicron implementada físicamente y modelada en 3D con sus dimensiones (12.4 cm alto $\times$ 15 cm ancho $\times$ 15.1 cm largo).

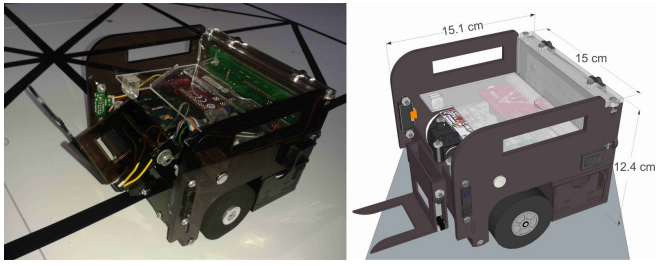


Figura 22. Diseño CAD del Robot Ómicron (derecha) e implementación física (izquierda)

El chasis del robot está formado por varias piezas de acrílico de diferentes espesores, y el color elegido ha sido un negro translúcido: *Humo 23* para casi todas las piezas, excepto para las que conforman el contenedor, las cuáles son transparentes o color *Cristal* para que el usuario pudiese observar el circuito de control sin necesidad de desarmar el robot. Los planos de todas las piezas en acrílico se encuentran en el archivo adjunto planosOmicron.cdr

2.2.1. Sistema modular

Para lograr que el prototipo fuera modular, se diseñaron las piezas de tal manera que unas encajaran con otras, y por medio de pernos y tuercas se sujetaran, para que de este modo el usuario pueda desarmar y armar el robot las veces que sea necesario.

Los 15 pernos utilizados para este fin son phillips cabeza redonda de 1/8", y las tuercas son cuadradas. En la Figura 23 se pueden observar las dimensiones del apoyo de las tuercas en las piezas de acrílico, así como su implementación física. Para

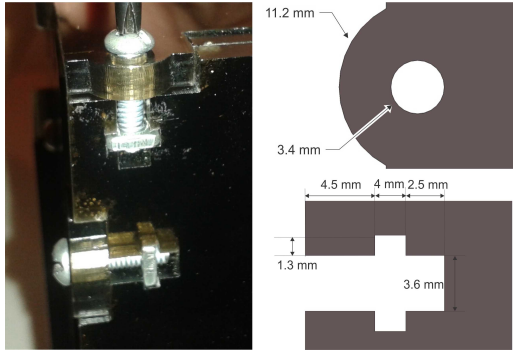


Figura 23. Método de ensamble

asegurar un buen ajuste entre las piezas se realizaron muescas en éstas de modo que encajen unas con otras, tal como se puede ver en la Figura 24.

Al unir todas las piezas del robot de la manera anteriormente descrita se obtiene como resultado el chasis del robot que se muestra en la Figura 25.

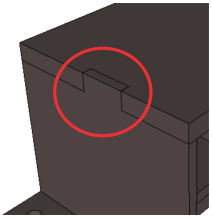


Figura 24. Encaje entre las piezas del chasis

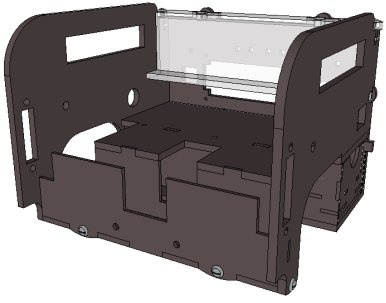


Figura 25. Chasis completo del robot

2.2.2. Pisos

El chasis cuenta con dos pisos, uno inferior, en el cual se asegura los motores (motorreductores y servomotores) y el circuito de sensores de línea, y otro superior, que sirve de soporte para el circuito de control. El piso inferior está formado por dos piezas de acrílico unidas permanentemente, se fabricaron como dos piezas individuales para mantener las confiabilidad del corte y para lograr la forma y espesor requeridas y realizar los canales para el ajuste con pernos. Estas piezas de los pisos se muestran individualmente en la Figura 26, donde las piezas 1 y 2 corresponden al piso inferior y la 3 al superior.

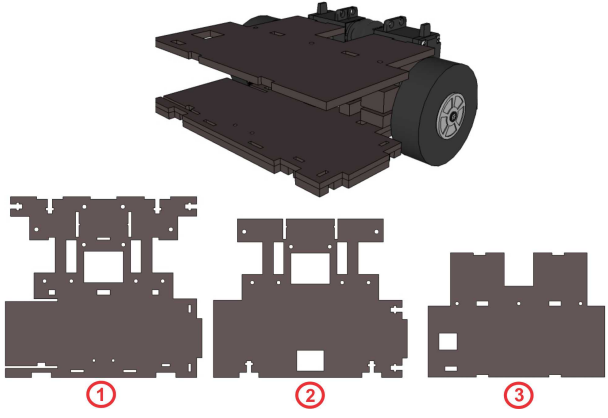


Figura 26. Pisos del chasis

2.2.3. Paredes

En el chasis del robot hay dos clases de paredes, las principales que son exteriores y las secundarias que se encuentran al interior del robot. Cada una de estas paredes brinda el soporte a los componentes del robot. En la Figura 27 se muestran las tres paredes principales.

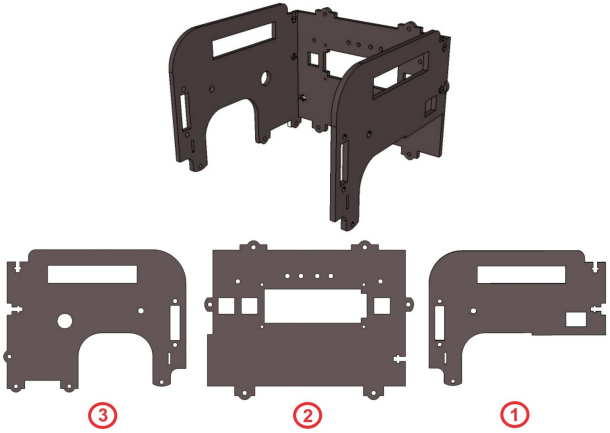


Figura 27. Paredes Principales

La pared 1, ubicada a la derecha de la imagen, presenta un orificio en el cual se soporta el interruptor de encendido general del robot, y otro para el sensor de distancia derecho. En la pared 2 que está en la parte trasera se asegura el circuito que contiene el LCD, los pulsadores para navegación en el menú y los leds indicadores. Por último la pared 3, a la izquierda tiene un orificio para la conexión del cable USB (*Universal Serial Bus*) que permite la comunicación entre el microcontrolador y el PC a la hora de reprogramar el robot. Esta pared, al igual que la pared derecha, da soporte a un sensor de distancia. Además de las paredes descritas hay tres que cumplen la función de soportar el piso superior del robot y el circuito de control. Estas paredes se muestran en la Figura 28. Por último, hay una pared más que soporta el sensor de detección de las pelotas ubicado en la parte delantera del robot, como se muestra en la Figura 29.

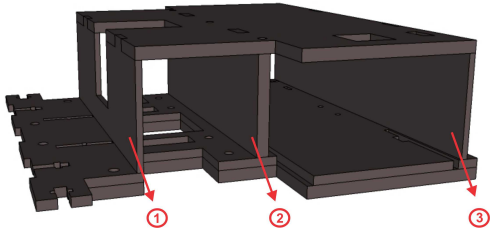


Figura 28. Paredes Secundarias

2.2.4. Compartimiento para batería

Debido a los cuidados necesarios para el buen mantenimiento y funcionamiento de la batería se hizo necesario diseñar un

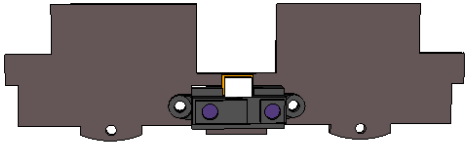


Figura 29. Pared delantera

compartimiento independiente que la contenga y que al mismo tiempo brinde facilidad a la hora de extraerla para que pueda ser cargada. Este compartimiento se muestra en la Figura 30. Al compartimiento se le agregó un seguro para que no se

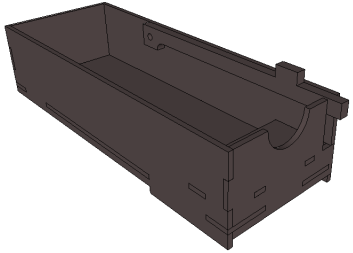


Figura 30. Compartimiento que contiene la batería

abriera accidentalmente en caso de que el robot sea inclinado y por lo tanto la batería caiga y se produzcan daños en la alimentación del robot. Este seguro funciona mediante una pieza de acrílico flexible asegurada en una de las paredes del compartimiento que encaja en un agujero ubicado en el piso superior del robot, en la Figura 31 se muestra el sistema de seguro del compartimiento. Para abrir el compartimiento y

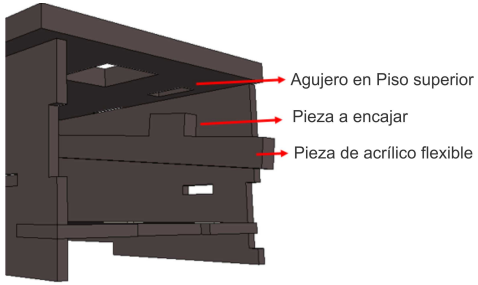


Figura 31. Sistema de seguro para el compartimiento de la batería

acceder a la batería se debe presionar la pieza de acrílico que sobresale de éste y jalarlo con cuidado desde la muesca ubicada en la pared frontal hasta que se logre liberar el seguro. Para cerrarlo se sigue un proceso similar sólo que se debe empujar el compartimiento hasta lograr que la pieza del seguro encaje en el agujero del piso superior del compartimiento.

3. DISEÑO ELECTRÓNICO

En robótica los componentes electrónicos son una parte fundamental ya que estos le proporcionan un alto grado de autonomía al robot. Para esto, debe ser dotado de un "cerebro" que le permita comprender las tareas que le han sido asignadas, además debe recibir información del entorno que lo rodea y del estado en el que se encuentra. En esta sección se describe el proceso de selección de los componentes electrónicos que se utilizaron para hacer posible dicha autonomía. También se describe el proceso para la elaboración de los circuitos electrónicos, las opciones que se consideraron y las decisiones que finalmente se tomaron.

3.1. Sensores

Los sensores se encargan de percibir una magnitud de una variable real y convertirla a una señal entendible para el microcontrolador. Existen diversos tipos de sensores, y las señales que miden pueden ser muy variadas, como por ejemplo temperatura, distancia, posición, velocidad, etc. Ómicron debe estar equipado con diferentes tipos de sensores para poder navegar por el laberinto guiándose con la ayuda de una línea negra y detectando obstáculos para así poder tomar otra ruta; también debe poder detectar las pelotas de tenis para recolectarlas y tener un conocimiento global de la posición en la que se encuentra. A continuación se presentan los diferentes tipos de sensores que se utilizaron en Ómicron:

3.1.1. Sensores de línea y cruces

Para la detección de la línea negra que guía al robot, se seleccionó el sensor fotoeléctrico de referencia QRD1114 [34], que consta de un diodo emisor de luz y un fototransistor que trabaja como receptor y su funcionamiento se basa en la capacidad de reflexión que presenten los objetos. Este tipo de sensores es muy usado en los robots seguidores de línea debido principalmente a su sensibilidad y a su tamaño compacto. Se utilizaron seis sensores ubicados en la parte inferior del robot como muestra la Figura 32. Los sensores que se encuentran en

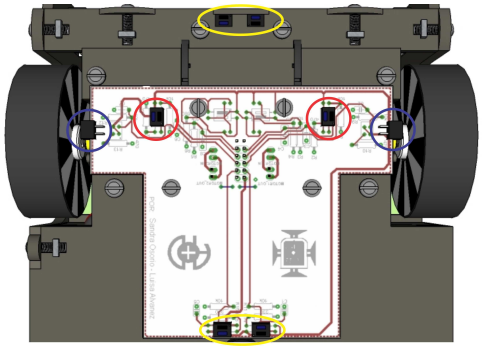


Figura 32. Ubicación de los sensores de línea y de cruces, vista inferior

el centro (señalados en amarillo) son para la detección de la

línea negra y los sensores de los extremos (señalados en rojo) son para la detección de los cruces.

Los sensores que están señalados en azul son empleados como encoders para la medición de las vueltas dadas por cada llanta del robot (ver sección 3.4.2.).

La conexión de cada uno de los QRD1114 se presenta en la sección 3.6.1. La salida de los sensores es una señal análoga entre 0 V y el nivel de voltaje de la alimentación lógica (5 V), donde 0 V se da cuando el sensor está detectando la línea negra y 5 V cuando detecta la superficie blanca.

3.1.2. Sensores de distancia y presencia

Se instalaron tres sensores de distancia de referencia SHARP GP2D12 [35], cuyo rango de visión se encuentra entre 10 y 80 cm de distancia, con tiempos de respuesta de máximo 45 ms. Dos de estos están ubicados a cada lado del robot para la detección de las paredes que forman parte del laberinto (Figura 33a), y el tercero se instaló en la parte frontal del robot (Figura 33b) para la detección de pelotas, obstáculos y depósito.

La detección de las pelotas de tenis que se deben recolectar durante el recorrido requirió de otro sensor para poder diferenciarlas de las paredes y de los obstáculos del laberinto, por lo que se implementó un sensor tipo barrera (Figura 33c) que está compuesto de dos partes, un LED infrarrojo emisor de un haz de luz, y un detector de proximidad de referencia SHARP IS471F. Ambos se encuentran ubicados en lados opuestos pero de manera alineada para que el haz de luz emitido incida directamente sobre el detector, de modo que la detección se realiza si el haz es interrumpido. Se seleccionó este tipo de sensor debido a que son inmunes a la luz externa, ya que modulan el haz de luz de su emisor, y su salida es completamente digital. La instalación de este sensor requirió de dos láminas de aluminio en la parte frontal del robot, como se puede observar en la Figura 33.

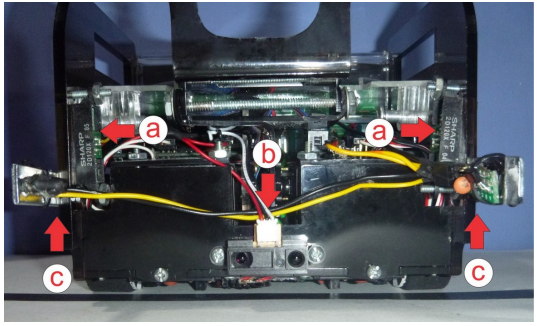


Figura 33. Ubicación de los sensores de distancia y presencia, vista frontal

3.2. Microcontrolador

Durante el diseño del robot se consideró el uso de diferentes microcontroladores, a los cuales se les realizó un análisis para determinar cuál de ellos tenía las características más apropiadas para el cumplimiento de los objetivos planteados. Ini-

cialmente se diseñó un prototipo usando un microcontrolador Freescale MCF51JM128VQH, ya que se tenía experiencia en su uso y programación debido al proceso de aprendizaje a lo largo de uno de los curso del pregrado. Éste se soldó directamente en el circuito de control, pero debido a que se presentaron graves problemas con este primer circuito, dejándolo completamente inservible, en el proceso de rediseño se evaluó la posibilidad de usar otro tipo de controlador.

Se comenzaron a hacer pruebas con el microcontrolador Arduino UNO ya que éste ofrecía un entorno de desarrollo muy accesible, de fácil programación y al formar parte de una tarjeta de desarrollo independiente simplificaba notablemente el diseño del circuito, ya que incluía la comunicación serial por USB y el circuito para la correcta programación de éste. Se descartó el uso de este controlador debido a que no ofrecía las características de memoria y puertos análogos indispensables para la aplicación y la tarjeta de Arduino que cumplía estas características tenía un tamaño superior al permitido para la adaptación del diseño electrónico y mecánico.

Teniendo en cuenta la experiencia obtenida con las tarjetas de desarrollo Arduino, se buscó otro microcontrolador que ofreciera las mismas ventajas pero con una capacidad superior. Así se pudo encontrar la tarjeta ChipKit UNO32 que, además de ser compatible con Arduino y ser más económica, ofrecía en una placa de tamaño reducido las características que se necesitaban para la aplicación. Ésta se caracteriza por tener un microcontrolador Microchip PIC32MX32OF128H. Después de realizar pruebas con este controlador, se llegó a la conclusión de que era el adecuado para este trabajo y se realizó el nuevo diseño del circuito de control.

Tabla 1. Comparación de los microcontroladores

Dispositivo	Freescale ColdFire	Arduino UNO	ChipKit UNO32
Microcontrolador	MCF51JM 128VQH	ATmega 328	PIC32 MX32OF128H
Hardware libre	NO	SI	SI
Software libre	NO	SI	SI
Velocidad del reloj (MHz)	50	16	80
Memoria Interna (kB)	128	16	128
RAM (kB)	16	2	16
E/S Digitales	51	14(6 PWM)	28 (5 PWM)
E/S Análogas	12	6	12
E/S de comunicaciones	4	2	4
Temporizadores internos	9/32-bit	8/16-bit	16/32-bit
Complejidad de implementación	Alta	Baja	Baja

En la Tabla 1 se resumen las características principales de los microcontroladores con los que se trabajó. De los dispositivos comparados el único que no cumple con los requisitos es el Arduino UNO, por lo que se descartó, como se había dicho anteriormente. De los otros dos comparados, la decisión final fue la placa ChipKit UNO32 como el controlador apropiado para la realización del proyecto, ya que es sencillo de implementar e incluye circuito de programación del microcontrolador, regulador y comunicación serial. En la Figura 34 se muestra una

imagen de la tarjeta de desarrollo usada y un esquema de la distribución de sus componentes y los pines disponibles para el ingreso y salidas de datos al microcontrolador.

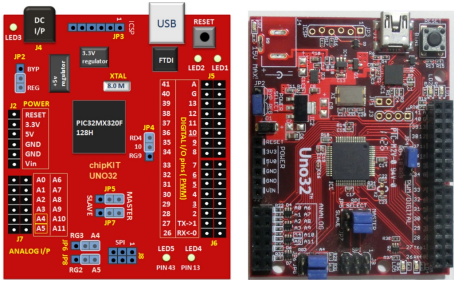


Figura 34. Tarjeta de desarrollo ChipKit UNO32

En la Tabla 2 se muestra la distribución de los pines disponibles, realizada teniendo en cuenta las características que deben presentar las señales de entrada y salida del microcontrolador para un óptimo funcionamiento.

Una de las características a resaltar de la tarjeta de desarrollo ChipKit UNO32 es el entorno de programación, que es compatible con Arduino y presenta mucha simplicidad no solo en el manejo del software, sino que cuenta con gran variedad de librerías que facilitan notablemente la programación y a las cuales se puede acceder de forma gratuita. En la Figura 35 se muestra el entorno de desarrollo MPIDE que se usa para la programación de todas las funciones del robot.

Tabla 2. Distribución de pines para las diferentes señales requeridas en la implementación del proyecto

# Pin	Señal	Tipo
A0	Sharp lateral derecho	Entrada análoga
A1	Sensor de línea trasero izquierdo	Entrada análoga
A2	Sensor de línea trasero derecho	Entrada análoga
A3	Lectura de voltaje batería	Entrada análoga
A4	Sharp frontal detector de pelota	Entrada análoga
A5	Sharp lateral izquierdo	Entrada análoga
A6	Sensor línea delantero izquierdo	Entrada análoga
A7	Sensor línea delantero derecho	Entrada análoga
A8	Sensor cruce izquierdo	Entrada análoga
A9	Sensor cruce derecho	Entrada análoga
A10	Sensor encoder derecho	Entrada análoga
A11	Sensor encoder izquierdo	Entrada análoga
3	Enable puente H motor derecho	Salida PWM
5	Enable puente H motor izquierdo	Salida PWM
9	Control 1 puente H motor derecho	Salida Digita
10	Control 2 puente H motor derecho	Salida Digita
11	Control 1 puente H motor izquierdo	Salida Digital
12	Control 2 puente H motor izquierdo	Salida Digital
28	Pin de Lectura/Escritura (R/W) del LCD	Salida Digital
31	Pulsador menú derecho	Entrada Digital
32	Pulsador menú Enter	Entrada Digital
34	Sensor IS471F detector de pelota	Entrada Digital
38	Pulsador menú izquierdo	Entrada Digital
35	Led1	Salida Digital
36	Led2	Salida Digital
37	Led3	Salida Digital
41	Led4	Salida Digital

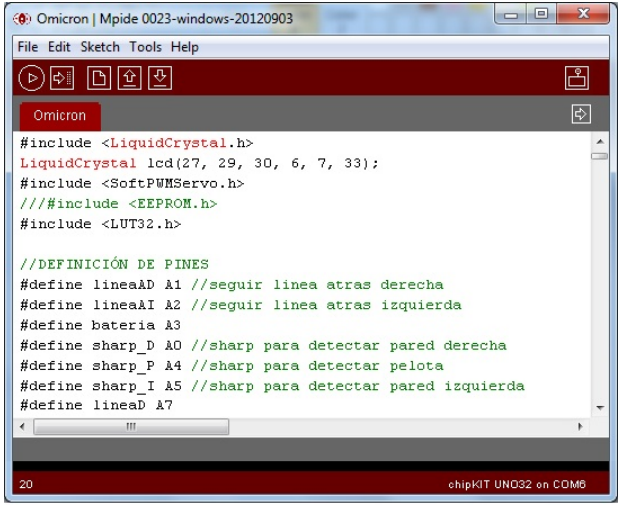


Figura 35. Entorno de desarrollo para la programación del robot

3.3. Alimentación

Uno de los puntos más importantes en el diseño del robot, tanto mecánico como electrónico, es la definición de la alimentación, ya que a partir de ésta se puede determinar no sólo el tiempo de autonomía del robot, sino que al ser el elemento con mayor tamaño y peso dentro de éste, influyen notablemente las decisiones de diseño a tomar. Se comenzó el proceso de selección de la batería basándose en los robots que han participado en la Olimpiada Robótica de la universidad, y en estos predominaba el uso de pilas recargables AA NiMH. Esta opción se descartó al comprobar que necesitaban una gran cantidad de baterías, alrededor de seis, y al mismo tiempo, tenían bajo rendimiento. Por lo tanto se comenzó una búsqueda de una alimentación más apropiada indagando por aquellas baterías usadas en aplicaciones como aeromodelismo y automodelismo (ver Tabla 3) y así se determinó que la batería de Litio Polímero (Li-Po) era la que mejor se adaptaba a los requerimientos del robot.

Tabla 3. Características generales de las baterías usadas comúnmente en aeromodelismo y automodelismo [6]

	Níquel Cadmio (Ni Cd)	**Níquel Metal Hidruro (NiMH)**	**Litio ion (Li-Ion)**	**Litio Polímero (Li-Po)**
Voltaje por celda (V)	1.25	1.25	3.6	3.7
Densidad de energía (Wh/kg)	45-80	60-120	110-160	100-130
Potencia específica (W/kg)	150	250-1000	1800	3000+
Autodescarga por mes (%)	20	30	10	10
Ciclos de carga/descarga	1500	300-500	500-1000	300-500

3.3.1. Batería

Se seleccionó la batería tipo Li-Po porque ofrece grandes ventajas que se enlistan a continuación:

- Permiten llegar a altas densidades de energía. Poseen una densidad de energía de entre 5 y 12 veces más a las de Ni-Cd o Ni-MH, a igualdad de peso. A igualdad de capacidad, las baterías de Li-Po son cuatro veces más ligeras que las de Ni-Cd.
- No sufren el efecto memoria. Pueden cargarse sin necesidad de estar descargadas completamente sin la reducción de su vida útil.
- Tienen una tasa de descarga de energía superior a las demás baterías conocidas.
- Su tamaño y peso las hacen muy útiles para equipos pequeños que requieran potencia y duración
- Mayor voltaje por celda en comparación con otras baterías recargables, por lo que se puede obtener altos voltajes usando menor cantidad de celdas.
- Escasa resistencia interna, lo que permite aprovechar casi el 100% de la energía almacenada.

En la Figura 36 se muestra una imagen con las dimensiones de la batería implementada: Li-Po Turnigy de 1300 mAh, y en la siguiente lista se resumen sus principales características.

- Capacidad mínima: 1300 mAh
- Configuración: 3 celdas en serie - 11.1 V
- Descarga constante: 20C
- Descarga pico: 30C
- Peso: 115 g
- Tamaño: 8.1 × 3.6 × 2.1 cm

Figura 36. Batería Li-Po Turnigy de 1300 mA

- Conector de carga: JST-XH
- Conector de descarga: XT60

La capacidad de la batería es la que indica la cantidad de energía que puede llegar a almacenar, para este caso en específico se tiene que es capaz de entregar 1.3 A de manera continua durante una hora. Teniendo en cuenta que el gasto de corriente de la suma de todos los circuitos del robot es aproximadamente 600 mA, se obtiene que la duración de la batería es de alrededor de 2 horas. Ahora, la duración promedio para la solución de los laberintos básico, avanzado y experto es de 15 minutos, por lo tanto el robot está en la capacidad de realizar 8 recorridos antes de necesitar cargar su batería. Para aumentar la autonomía del robot se recomienda el uso de una batería adicional con las mismas características de la anterior, para que se remplace inmediatamente la batería descargada

mientras ésta es cargada nuevamente, ya que su tiempo de carga es de 1 hora, es suficiente para siempre poder contar con una batería de respaldo totalmente cargada.

Aunque las Li-Po ofrecen muchas ventajas, también es necesario tener en cuenta aspectos como:

- No admiten carga rápida. Tardarán como mínimo una hora en cargarse completamente.
- Se necesita un cargador específico para esta clase de batería, incluso para carga lenta.
- No toleran cortocircuitos, sobrecargas o temperatura excesiva.
- No toleran abusos, como descargas profundas o sobreconsumo.
- Requieren de un cuidado especial para evitar disminución de su vida útil y evitar lesiones. Para conocer las normas generales y algunas recomendaciones sobre su uso se puede remitir al manual de Ómicron.

3.3.2. Cargador

Se debió tener especial cuidado en la selección del cargador adecuado para la batería implementada, no sólo por los riesgos ante su manipulación, sino también buscando la mayor simplicidad en su uso, ya que la mayoría de cargadores tiene controles que pueden ser complicados para personas con poca experiencia en su manejo, lo que aumenta la posibilidad de un uso inadecuado de éste y por lo mismo, un aumento de los riesgos en su manipulación disminuyendo la vida útil de la batería y causando daños personales o materiales. Por estas razones se seleccionó el cargador HobbyKing E4, debido a que no sólo es fácil de usar, sino que mantiene balanceadas las celdas mientras carga la batería. En la Figura 37 se muestra una imagen del cargador seleccionado y a continuación se resumen sus características. Para mayoi información de los cuidados y la forma de conexión del cargador y la batería ver Anexo A.

Figura 37. Cargador HobbyKing E4

- Alimentación: 11 a 15 V
- Celdas a cargar: 1 a 4 celdas conectadas en serie
- Tipo de batería: Li-Po
- Corriente de carga: 100 a 4500 mA
- Potencia de carga: 40 W
- Dimensiones: 104×62×31 mm

3.3.3. Regulador

El regulador de voltaje usado para el circuito electrónico es el Turnigy SBEC 5 A que se muestra en la Figura 38. Las siglas BEC vienen del inglés *Battery Eliminator Circuit* que hacen referencia a la idea de eliminar la necesidad de una batería diferente para servos y control, y otra para motores [36]. Con este circuito se unifica y una sola batería puede dar servicio a ambas necesidades, esta característica lo hace especialmente útil para el proyecto. Además los BEC ofrecen una salida de voltaje constante, sin caídas ni picos.

Figura 38. Regulador de voltaje TURNIGY SBEC 5 A

Los SBEC o *switching* BEC cuentan con la característica de ser un regulador conmutado, por lo que tiene alrededor de un 80% de eficiencia, el doble que un regulador lineal [37]. Las características deL SBEC utilizado se resumen a continuación.

- Tipo de regulador: Conmutado
- Protección de entrada: Protección ante inversión de polaridad
- Entrada: Entre 8 y 26 V (2 a 7 celdas de batería Li-Po)
- Salida: Selectiva, 5 o 6 V a 5 A

Es importante aclarar que se realizó una pequeña modificación en el regulador para lograr que su salida fuera permanentemente 5 V, de esta forma se evitaría que por error los componentes del robot fueran alimentados con más voltaje del que pueden tolerar y causar daños graves en estos.

3.4. Control de motores

3.4.1. Drivers de Motores

Los drivers seleccionados para el control de cada uno de los motores de DC que tiene Ómicron son circuitos integrados de referencia Freescale MC33887 [38] (ver Figura 39). Este controlador es de tipo Puente H y las características más relevantes se presentan a continuación:

- Voltaje de operación: 5 V a 28 V
- Temperatura de operación: -40 °C a 125 °C
- Máxima corriente de operación: 5 A
- Máxima frecuencia de las señales de entrada: 10 kHz

Para la selección de este dispositivo, se realizaron pruebas a los motores para conocer los picos de corriente máximos que pueden presentar (2 A), y se buscó que el driver tuviere la capacidad de soportar el doble de esta corriente. Otros factores decisivos fueron: facilidad de uso, tamaño compacto y otras prestaciones como por ejemplo la capacidad de monitorear el consumo de corriente del motor que está controlando.

Para la variación de velocidad de los motores requerida en el control de velocidad se utilizó la técnica de Modulación por

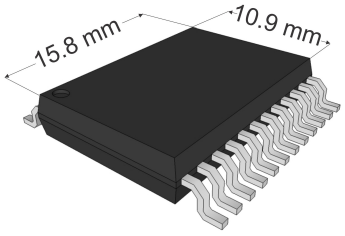


Figura 39. Puente H Freescale MC33887

Ancho de Pulso (Pulse Width Modulation o PWM), en la que se modifica el ciclo de trabajo de una señal periódica para controlar la cantidad de energía promedio proporcionada al motor. Para poder implementar este método se deben tener en cuenta básicamente dos variables: frecuencia y ciclo de trabajo; la frecuencia de la señal no debe ser muy baja ya que el motor podría alcanzar a encenderse y apagarse completamente durante cada ciclo y no se apreciaría el efecto promedio, y tampoco debe ser muy alta ya que superaría el límite de operación del puente H; el ciclo de trabajo debe poderse variar en pasos lo suficientemente pequeños como para permitir el control adecuado de velocidad [9].

Para implementar la técnica PWM en el robot se puso a prueba una herramienta que hace parte del entorno de desarrollo Arduino y que podía cumplir con las condiciones previamente descritas: la función analogWrite. Esta función le permite al microcontrolador generar señales de tipo PWM en el puerto que se desee, a una frecuencia de 490 Hz, con incrementos/decrementos del ciclo de trabajo mínimos de 0.39% respecto a un ciclo completo. La prueba realizada consistió en generar las señales por medio de esta función, para luego conectarlas a las entradas Enable de cada puente H, variando el ciclo de trabajo desde 0% hasta 100%, y observar que la variación de velocidad del robot fuese suave y continua.

En la Figura 40 se pueden observan las señales generadas desde el microcontrolador, ordenadas de izquierda a derecha, la primera con ciclo de trabajo de 3.9%, la segunda con 39% y la tercera con 94%. Durante las pruebas se observó que el movimiento del robot variaba de manera suave y sin cambios abruptos, por lo que se determinó que este método de control de motores es aceptable para implementar el control de velocidad.

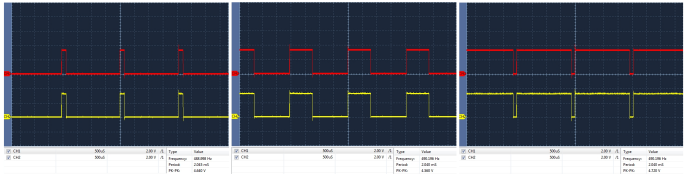


Figura 40. Señales de prueba para puentes H

3.4.2. Encoders

Los encoders son codificadores que generan señales digitales en respuesta al movimiento. Debido a que se necesitaba un control de velocidad a la hora de subir y bajar la rampa

presente en la categoría avanzados, y a la hora de saber cuanto se ha desplazado el robot para conocer su ubicación, se implementaron dos encoders. Para la construcción de los encoders se utilizó en cada una de las ruedas del robot (Figura 32) un sensor infrarrojo de referencia QRD1114 y un disco hecho en acrílico con un total de 16 franjas blancas y negras alternas con un diámetro un poco menor que el de la llanta (5.4 cm). Se eligieron estos sensores debido a su tamaño, y a que funcionan muy bien a la hora de detectar cambios de blanco a negro, que es lo que se necesita para el buen funcionamiento de los encoders. La expresión

$$R = \frac{P}{N} \tag{1}$$

permite calcular la resolución para conocer cuanto se ha avanzado en un cambio de blanco a negro, donde R es la resolución, P es el perímetro de la llanta y N es el número de franjas que tiene el encoder. Reemplazando los valores particulares en 1 se obtuvo que la resolución es de 1.12 cm.

3.5. Controles de Usuario

Para facilitar la interacción entre el usuario y el robot se vio la necesidad de usar dispositivos que permitan la selección de su modo de operación y que al mismo tiempo sirvieran de apoyo en su proceso de programación. Para lograr dichos objetivos se agregaron al robot una serie de leds, pulsadores y un LCD a través de los cuales se podía suministrar y adquirir información.

3.5.1. Indicadores Led

Para facilitar el proceso de posicionamiento del robot al comenzar a resolver un laberinto, se agregaron 4 leds que indiquen la detección de línea de cada sensor. La relación entre los leds y el sensor de línea correspondiente se muestra en la Tabla 4 teniendo en cuenta la distribución de los Leds mostrada en la Figura 41. Esta relación cambia en los leds 1 y 4 cuando el robot está en movimiento, indicando ahora los cruces detectados y no la línea.

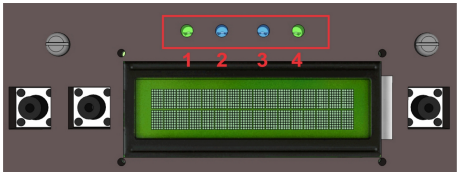


Figura 41. LEDs indicadores de detección de línea

Tabla 4. Configuración Leds Indicadores

Led	Color	Sensor correspondiente
1	Verde	Sensor de línea trasero izquierdo
2	Azul	Sensor de línea delantero izquierdo
3	Azul	Sensor de línea delantero derecho
4	Verde	Sensor de línea trasero derecho

3.5.2. LCD y pulsadores

Debido a que el robot Ómicron debe permitir tres modos distintos de operación (laberinto básico, avanzado y experto), además de mostrar de manera ininterrumpida el porcentaje de carga de la batería, se hizo indispensable el uso de dispositivos que permitieran visualizar esta información y que facilitara la selección del modo de operación, por lo que se utilizó un LCD 16×2 y tres pulsadores distribuidos de la manera en que se muestran en la Figura 42. En esta imagen también se puede

Figura 42. Dispositivos de entrada y salida para navegación en el menú

observar el mensaje de bienvenida que se muestra una vez se encienda el robot, se resalta la zona donde se informa el porcentaje de carga de la batería, que siempre estará ubicado en la zona demarcada por el cuadro rojo.

El LCD se trabajó a 4-bits y se manejó con la ayuda de la librería de Arduino llamada LiquidCrystal, que permite utilizar caracteres estándar y caracteres independientes como el de la batería.

3.6. Circuitos Impresos

Ómicron cuenta con tres tarjetas de circuitos impresos o PCB (del inglés *printed circuit board*) para facilitar el diseño modular: tarjeta de control, de sensores y de interfaz de usuario. Las tres tarjetas están interconectadas entre sí y actúan en conjunto. En esta sección se explicarán los principales componentes de cada una, y se presentarán las diferentes etapas de diseño.

Para la creación de las 3 PCB se realizaron los siguientes pasos:

- Se realizaron pruebas a cada uno de los componentes en una protoboard y se verificó su correcto funcionamiento.
- Se diseñaron los circuitos electrónicos en la versión de evaluación del programa *Cadsoft EAGLE*, que es un programa que permite crear esquemas de circuitos electrónicos y a partir de éstos obtener, el diseño del circuito impreso a una o dos caras.
- Una vez finalizado el diseño del esquemático, se procede a dar un tamaño específico a cada tarjeta y a ubicar los componentes dependiendo de donde se necesiten. Posteriormente comienza el proceso de enrutamiento. Para la tarjeta de control fue necesario realizar el circuito en doble capa debido a la gran cantidad de componentes.
- La fabricación de los PCB fue llevada a cabo por personal especializado en la elaboración de circuitos impresos para garantizar que las pistas y el material fuesen resistentes y de gran calidad.
- Por último se procedió a soldar cada componente.

Los diagramas esquemáticos y el diseño final de los PCB se pueden encontrar en el Anexo B.

3.6.1. *Tarjeta de sensores*

La tarjeta de sensores presentó tres versiones hasta llegar al diseño final. En la Figura 43a se puede observar el primer prototipo, pero debido a que se realizaron cambios en la parte mecánica del carro, las posiciones de los sensores y de los huecos que se encargan de sujetar la tarjeta se vieron afectadas, es por esto que se diseñó otra (Figura 43b), que tenía el mismo diseño, pero con una ubicación diferente de los sensores, conectores y huecos de sujeción. Debido a que no se estaba presentando un buen comportamiento en la seguida de línea (ya que los sensores se encontraban en el mismo eje de giro) se realizó otra tarjeta (Figura 43c) con dos sensores más para la seguida de línea en la parte trasera, y los sensores de línea delanteros se ubicaron por fuera de la tarjeta para que pudiesen quedar lo más adelante posible. El PCB final tiene un tamaño de 10.6 × 8.8 cm.

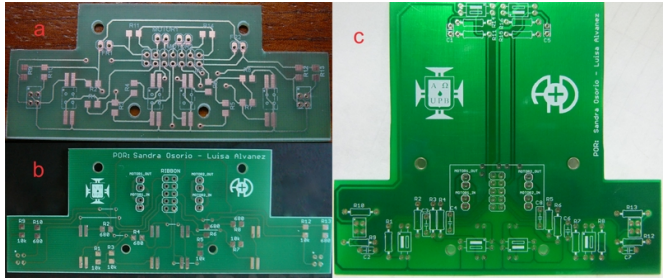


Figura 43. Tarjetas de sensores implementadas

La tarjeta de sensores completa en su versión final se puede observar en la Figura 44, donde se han enumerado los principales componentes, que se describen en la Tabla 5.

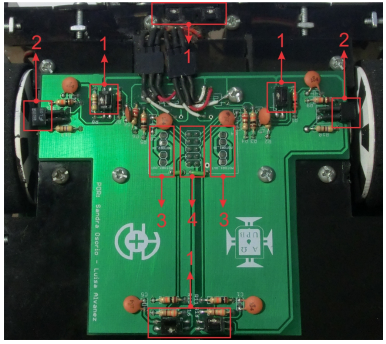


Figura 44. Circuito impreso de los sensores de línea

3.6.2. *Circuito Interfaz de Usuario*

Este circuito es el único que no presenta modificaciones, ya que en un inicio, solo se pensaba tener dos tarjetas: la de control y la de sensores, pero debido a que la tarjeta de control debía contener muchos componentes, se implementó una tercera tarjeta para facilitar el proceso de enrutamiento y permitir que el robot fuese más modular. En la Figura 45 se puede observar la tarjeta con los componentes soldados, los cuales se explican en la Tabla 6.

Tabla 5. Listado y descripción de componentes de la tarjeta de sensores

#	Tipo	Descripción
1	Sensores QRD1114	Detectan la línea negra y los cruces que permiten que el robot se guíe por el laberinto (ver sección 3.1.1.).
2	Sensores QRD1114	Realizan la lectura de los encoders (ver sección 3.4.2.).
3	Conectores Ribbon hembra	Allí se conectan las señales de los dos motores de DC. No van directamente a la tarjeta de control para aumentar la vida útil del cable de los motores.
4	Conector Ribbon 10 pines macho	Se encarga de comunicar la tarjeta de sensores con la tarjeta de control

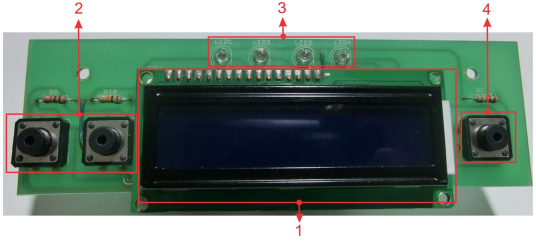


Figura 45. Circuito impreso de la interfaz de usuario

La PCB se ensambla por medio de 4 tornillos a la pared de acrílico de la parte posterior del robot, y tiene un tamaño de 13.4 × 4 cm.

Tabla 6. Listado y descripción de componentes de la tarjeta de interfaz de usuario

#	Tipo	Descripción
1	Display LCD	Permite visualizar el estado de la batería y la modalidad en la que está trabajando el robot (ver sección 3.5.2.).
2	Pulsadores	Permiten al usuario desplazarse por el menú del LCD
3	Indicadores led	Indica si los sensores de línea y cruces están detectando línea negra (led encendido) o superficie blanca (led apagado). Esenciales para la etapa de pruebas.
4	Pulsador ENTER	Permite al usuario interactuar con la interfaz de usuario.

3.6.3. Circuito de Control

Este es el circuito principal, ya que a él convergen las señales de las otras dos tarjetas y actúa como plataforma de interconexión entre la unidad de procesamiento (ChipKIT UNO32) y los demás elementos del sistema de control. Al igual que la tarjeta de sensores, ésta presentó dos modificaciones drásticas en cuanto al diseño. La primera PCB (Ver Figura 46a) se diseñó con el microcontrolador Freescale DEMO JM de 32 bits, y su fabricación fue con un proveedor que no ofrecía alta calidad en los circuitos, por lo que se presentaron problemas de continuidad en las rutas, además no presentaba la tecnología que se emplea para comunicar las capas de un circuito impreso por medio de agujeros, llamado *Through-Hole*

Technology (THT), por lo que fue necesario soldar puentes para asegurar la unión eléctrica de ambas capas.

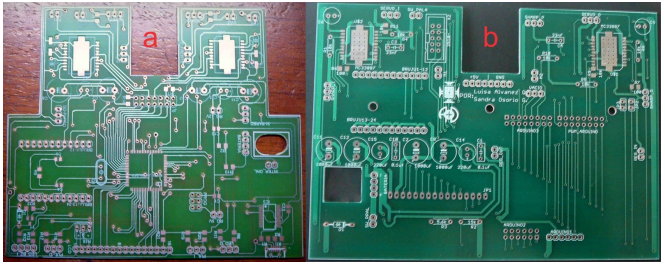


Figura 46. Circuitos impresos implementados de la tarjeta de control

Fue debido a estos inconvenientes en la fabricación que se decidió cambiar de proveedor, que, aunque más costoso, garantizaba una alta calidad en los circuitos. La versión 2 del circuito de control (Ver Figura 46b) presenta también modificaciones en cuanto al tamaño, y al diseño electrónico debido a la decisión de cambiar de microcontrolador. Además se le agregó una malla de tierra para reducir el ruido eléctrico. La PCB se encuentra ubicada sobre una placa de acrílico que actúa como segundo piso del robot. El tamaño de ésta es de 14.2 × 10 cm.

En la Figura 47 se puede observar la tarjeta de control completa con sus principales componentes enumerados. En la Tabla 7 se hace la descripción de los componentes previamente numerados.

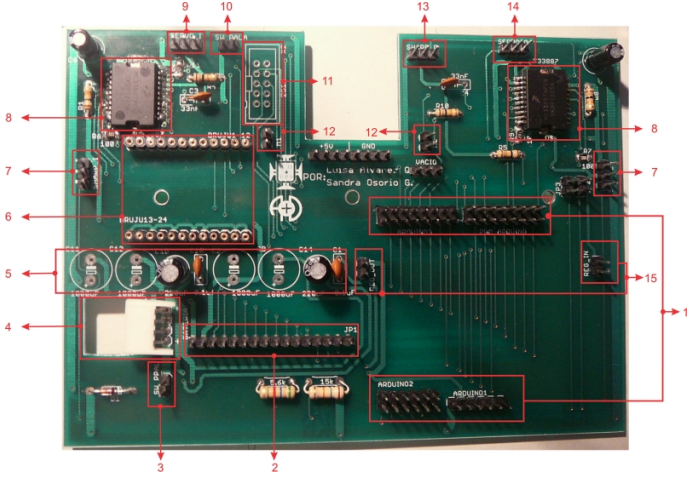


Figura 47. Circuito impreso de la tarjeta de control

Tabla 7. Listado y descripción de componentes de la tarjeta de control

#	Descripción
1	Conectores Ribbon macho: sirven como interconexión entre la unidad de procesamiento (ChipKit Uno 32) y los demás componentes de la tarjeta de Control. Se utilizaron conectores para que en caso de daños, la plataforma Chipkit se pueda reemplazar fácilmente.
2	Regleta de 16 pines macho: conecta las señales que llegan de la tarjeta de interfaz de usuario con la tarjeta de control.
3	Regleta de 2 pines macho: conexión para el interruptor de encendido general.
4	Conector jst: usado para conectar la batería LiPo de 12 V (ver sección 3.3.1.), que es la encargada de alimentar todo el robot.
5	Capacitores: sirven para filtrar ruidos de alta frecuencia en las líneas de alimentación (12 y 5 V).
6	Conectores maquinados: para insertar la brújula electrónica. Esta brújula viene en el kit de Ómicron, pero no se implementará en este trabajo de grado. Se deja como extensión para futuros trabajos con este robot.
7	Regleta de 3 pines macho: para conectar los sensores SHARP GP2D12 que sirven como detectores de las paredes laterales del laberinto (ver sección 3.1.2.).
8	Puentes H Freescale MC33887: drivers de los motores de DC (ver sección: 3.4.1.).
9	Regleta de 3 pines macho: para conectar el servo encargado del movimiento del contenedor (ver sección 2.1.2.).
10	Conector macho de 2 pines para el sensor IS para la detección de pelotas.
11	Conector Ribbon 10 pines macho: encargado de comunicar la tarjeta de sensores con la tarjeta de control.
12	Conectores Ribbon Macho 2 pines: para la señal de los motores, previamente conectados a la tarjeta de sensores.
13	Regleta de 3 pines macho: para conectar el sensor SHARP GP2D12 para detección de pelotas y de paredes frontales (ver sección 3.1.2.).
14	Regleta de 3 pines macho: conecta el servo encargado del movimiento de la pala para la recolección de las pelotas (ver sección 2.1.2.).
15	Regulador de voltaje SBEC Turnigy (ver sección 3.3.3.).

4. PROGRAMACIÓN Y SOFTWARE

4.1. Asistente para la creación de laberintos y planeación de Rutas: ÓMICRONTOOL

Como se mencionó en el capitulo 1.2., la arquitectura de control consta de dos partes: la primera a nivel de software, y la segunda a nivel de hardware. En este capítulo se describe lo referente a la parte que se desarrolla a nivel de software, que es la aplicación encargada de facilitarle al usuario la reprogramación de las rutas de laberinto de una manera sencilla.

La aplicación, llamada "Ómicrontool", es una herramienta de asistencia al usuario para el manejo del robot Ómicron creada en el entorno de desarrollo integrado, llamado también IDE (sigla en inglés de *Integrated Development Environment*) Microsoft Visual C# 2010 Express, que posee una interfaz gráfica y permite esquematizar cualquier laberinto que pueda ser construido para las Olimpiadas Robóticas A+D.

Se decidió trabajar en el IDE Microsoft Visual C# debido a que provee una ayuda gráfica para programar en el lenguaje C#, es orientado a objetos, y es un lenguaje sencillo, similar al C, del cual se tenían bases.

En Ómicrontool no solo se pueden esquematizar los laberintos, adicionalmente el usuario tiene la posibilidad de utilizar la herramienta de "Completar Laberinto" de manera que el programa automáticamente complete el laberinto de forma aleatoria. Una vez dibujado el laberinto a resolver, se puede también programar la ruta que se desea que siga el robot dentro del laberinto de manera sencilla con la herramienta "Planeador De Ruta". En la Figura 48 se puede observar la ventana principal del programa.

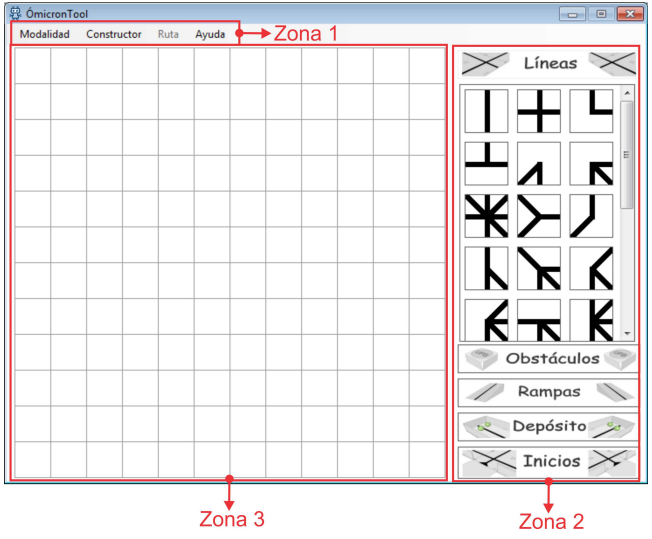


Figura 48. Ventana principal de ÓmicronTool

Esta ventana se encuentra dividida en tres zonas:

- Zona 1: Corresponde a la barra de Menú, la cual da acceso a las características principales del programa como: Modalidad, Constructor, Ruta y Ayuda.

- Zona 2: Área Botonera, en la cual se podrá escoger cualquier combinación de líneas que se necesite para el diseño de los laberintos de la Olimpiada Robótica A+D. Se tuvieron en cuenta todas líneas a 45° y 90° de la horizontal.
- Zona 3: Zona de dibujo, es el área donde se va a poder esquematizar el laberinto que se desee. Esta área está compuesta por 12 × 12 celdillas.

Para una mejor comprensión de lo que representa una celdilla en ÓmicronTool, es importante recordar que el laberinto de la UPB es modular y se encuentra conformado por celdas cuadradas de 61 × 61 cm^2. Cada celda está dimensionalmente dividida en 4 celdillas de 30.5 × 30.5 cm^2, como se muestra en la Figura 49. Estas celdillas no son visibles físicamente en el laberinto, son divisiones realizadas en el diseño y corresponden a las imágenes disponibles en el programa.

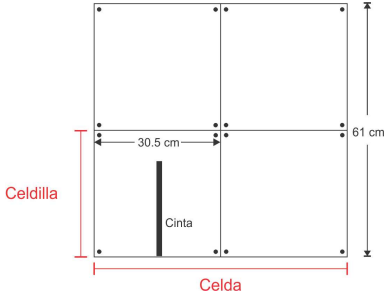


Figura 49. Pieza modular del laberinto de la UPB

La estructura básica del programa se puede observar en la Figura 50, y se explicará a continuación.

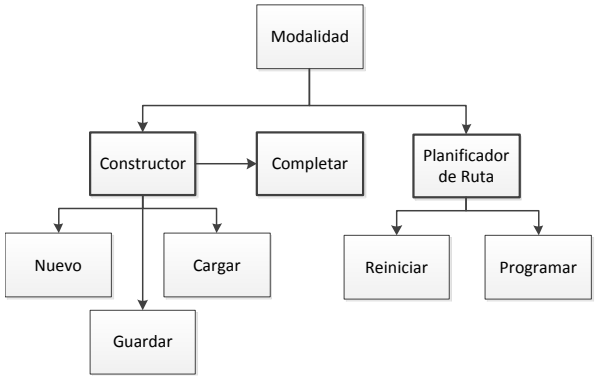


Figura 50. Estrucura básica del programa Ómicrontool

4.1.1. Constructor de Laberintos

En esta modalidad, el programa permite dibujar el laberinto que el robot deberá recorrer. Para esto, lo primero que se debe hacer es seleccionar en la barra de menú Modalidad > Constructor. Cuando esté seleccionada, se habilitarán otras opciones en la barra de menú, tales como Nuevo Laberinto, Guardar Laberinto, Cargar Laberinto y Completar Laberinto. Para entender mejor cada una de estas funciones, el lector

puede remitirse al manual de usuario de ÓmicronTool que se encuentra en el Anexo C.

Para que se pudiese esquematizar el laberinto, se dividió la zona de dibujo en una matriz de celdillas, sobre las cuales el usuario irá ubicando las imágenes que se encuentran disponibles en la zona botonera en el lado derecho de la ventana principal.

Debido a que el laberinto no solo se compone de líneas continuas, se dividieron todas las imágenes por clases, las cuales son: líneas, obstáculos, depósitos, rampas e inicios; con lo cuál ÓmicronTool cuenta en total con 306 imágenes diferentes que el usuario puede ubicar en una celdilla.

4.1.2. Planeador de Rutas

Esta modalidad permite al usuario indicar el camino que Ómicron va a recorrer sobre el laberinto que se tuvo que haber esquematizado previamente. Para esto se debe seleccionar en la barra de menú Modalidad > Planeador de Rutas, y luego ir haciendo clic en las celdillas en el orden que desee que el robot se mueva (comenzando con la celdilla que representa el inicio y finalizando en la celdilla que representa el depósito). Es una manera sencilla de programar, sin embargo se deben seguir ciertas reglas que se encuentran en el manual (Anexo C) para que la programación sea exitosa.

Como resultado de dar clic a las celdillas que indicarán el recorrido del robot, el programa genera un vector que va almacenando valores que corresponden a la dirección que éste debe seguir. El usuario podrá ver este vector en la ventana emergente "Prog. Ruta", donde también se puede visualizar, a medida que se selecciona una celdilla, la dirección a la que va a moverse el robot (referenciado desde el mismo robot), por ejemplo, si aparece "derecha", significa que el robot debe girar 90° en sentido horario de la posición en la que se encuentra en el momento.
Un ejemplo de esto se muestra en la Figura 51, donde se presenta un tramo de laberinto con una ruta trazada, y a la derecha la ventana "Prog. Ruta", en la cual se puede observar que el vector a enviar es de 12 posiciones, lo que indica que el usuario ha dado 12 clics, y que el robot debe seguir la trayectoria trazada por los cuadros que cambiaron de color, como lo indican las flechas rojas. En la modalidad Planeador de

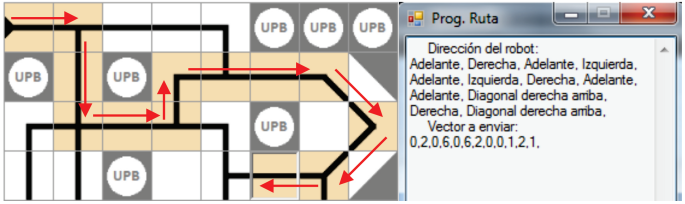


Figura 51. Ejemplo de una ruta en Ómicrontool

Ruta también se encuentra habilitada la opción "Programar Ruta", que es la que permite enviar la información que el robot necesita para poder solucionar el laberinto indicado. Esta información consta de lo siguiente:

- Vector con la ruta que debe seguir el robot
- Posición de la celdilla de inicio y la celdilla de depósito

- Vector con la información del laberinto dibujado

En la subsección 4.2.1. se ampliará más el tema de la comunicación entre el robot y el programa Ómicrontool.

4.1.3. Autocompletado de Laberintos

Es una herramienta que forma parte de la modalidad Completar Laberinto, y es de gran importancia, ya que por medio de ésta se generarán los laberintos aleatorios que deberán resolver los robots concursantes en la categoría expertos de la Olimpiada Robótica A+D.

Para comenzar a utilizar esta herramienta se debe ir a la barra de menú Constructor > Completar Laberinto, donde saldrá una ventana emergente (Figura 52) con algunas opciones que ayudarán al programa a resolver el algoritmo para completar automáticamente el laberinto iniciado previamente por el usuario.

El algoritmo para el autocompletado de laberintos se basó en el de Búsqueda en Profundidad, más conocido por sus siglas en inglés DFS (*Depth First Search*) [39], que utiliza algoritmos de grafos, donde se restringe al laberinto para que sea tratado como un árbol (en el sentido de teoría de grafos). Se eligió este algoritmo, ya que es de las técnicas de generación de laberintos más comunes debido a que es sencillo de implementar.

Se realizaron algunas modificaciones al algoritmo DFS debido a que en este programa, la generación del laberinto no comienza desde cero, sino que el programa debe entender las celdillas que el usuario haya ingresado antes de utilizar esta herramienta y desde ahí generar uno aleatorio, es decir, la herramienta no solo es para la generación de laberintos, sino para autocompletar uno ya empezado.

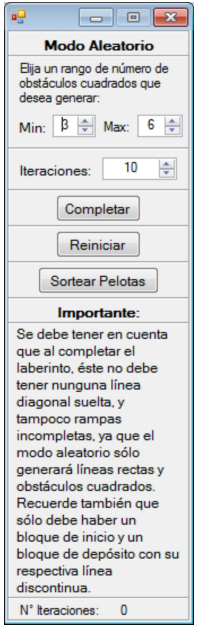


Figura 52. Ventana Modo aleatorio

En los siguientes ítems se mencionan las restricciones para que el algoritmo trabaje de forma adecuada, y más adelante se numerará la explicación del algoritmo utilizado.

- Debe existir una celdilla de Inicio (únicamente una).
- Debe existir una celdilla de Depósito (únicamente una) seguida de la celdilla con línea discontinua.
- Si el usuario ha colocado celdillas con diagonales, debe cerrarlas, es decir, las celdillas deben terminar en líneas horizontales o verticales, ya que el algoritmo no genera diagonales
- Si el usuario ha colocado celdillas con rampas, debe terminarlas, ya que el algoritmo no genera rampas.
- Todas las celdillas vacías en el momento previo a la generación deben conformar un único conjunto de celdillas vacías continuas, de manera que el algoritmo pueda recorrer todo el laberinto y no existan zonas aisladas o inaccesibles.

Para entender el algoritmo utilizado, basado en DFS, se debe tener en cuenta que el área de trabajo se definió como una matriz de celdas de 12 × 12, y que cada celda posee tres propiedades denominadas: Tipo, líneas y Usuario.

- **Tipo:** Esta propiedad tiene dos estados, Visitado o No visitado.
- **Línea:** Esta característica representa el camino o cinta negra que deberá seguir el robot. Su valor está entre 1 y 255 dependiendo de la forma del camino. En la Figura 53 se puede observar con un ejemplo cómo se conforma dicho número.
- **Usuario:** Esta propiedad informa si la celda ha sido puesta por el usuario o no. Sirve en caso de que la persona desee volver a completar el laberinto, ya que se borrarán todas las celdas generadas pero no las puestas por el usuario.

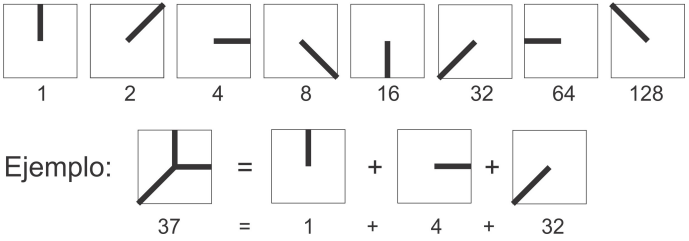


Figura 53. Ejemplo de la propiedad: Línea

Teniendo esto en cuenta, a continuación se explica el algoritmo:

1. Revisar cuales celdas fueron puestas por el usuario, es decir, a las celdas que no están vacías se le asigna la propiedad: Usuario.
2. De manera aleatoria se elige el número de obstáculos de acuerdo al rango que el usuario haya especificado. Luego, también de manera aleatoria se ubica dicho número de obstáculos en celdas que se encuentren vacías.
3. Se le asigna la propiedad línea a cada celda de acuerdo a la imagen que tengan. Si la celda está vacía, el valor de línea es cero.
4. Se le asigna a cada celda la propiedad: Tipo, donde todas las celdas puestas por el usuario, y las que contienen los

obstáculos son marcadas como Visitadas, mientras las celdas vacías se marcan como No visitadas.
5. Se elige la primera celda No visitada como "celda actual" y se marca como Visitada. A la vez se va almacenando cada celda que se va visitando.
6. Mientras que no estén todas las celdas marcadas como Visitadas se debe hacer lo siguiente:
   - Se busca de manera aleatoria una celda vecina a la celda actual que esté marcada como No visitada.
   - Si la encuentra, la celda siguiente se convierte en la celda Actual y se marca como Visitada, pero si no la encuentra, es decir, si todos los vecinos ya han sido visitados, debe devolverse a la celda anterior y convertirla en celda Actual.
   - Se modifica la propiedad Línea de la celda actual de acuerdo al camino seleccionado y a cómo están caracterizadas las celdas vecinas. Esta propiedad es la que se traduce posteriormente en imágenes.

4.2. Programa en Arduino

El programa que permite a Ómicron solucionar los laberintos se describirá en esta sección. Una estructura básica de este programa se puede observar en la Figura 54. En general, ésta consta de un menú, que se visualiza al en el LCD una vez se encienda el robot, y que permite seleccionar las diferentes modalidades de operación, como la programación de la ruta desde el software ÓmicronTool, y si se desea solucionar el laberinto básico, avanzado o experto.

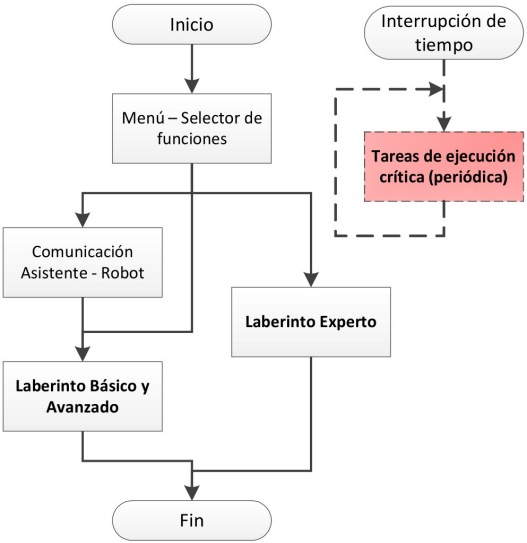


Figura 54. Estructura general del programa en Arduino

4.2.1. Comunicaciones Asistente - Robot

Debido a que el robot Ómicron tiene la capacidad de resolver varios laberintos diferentes sin necesidad de ser reprogramado, es indispensable que pueda almacenar los datos correspondientes a los laberintos conocidos de manera no volátil, es decir, que permanezcan a pesar de que el microcontrolador

sea desenergizado. El microcontrolador de la tarjeta de desarrollo ChipKit UNO32 cuenta con una memoria EEPROM virtual simulada a partir de su memoria Flash y que está libre para ser usada por el usuario. La capacidad de esta memoria EEPROM es de 512 bytes y puede ser sobrescrita como mínimo 1000 veces sin tener restricciones en la lectura. Esta memoria EEPROM virtual es la que se utiliza para almacenar los vectores que envía por serial el software ÓmicronTool al robot. En la Figura 55 se presenta un diagrama que muestra la interacción entre el software ÓmicronTool ejecutado en el computador y el microcontrolador del robot.

Debido a que se cuenta sólo con 512 direcciones de memoria disponibles y que el mayor número que puede almacenar es 255, se hizo necesario realizar modificaciones al vector enviado por el software antes de ser almacenado. En primer lugar se realizó la división de la memoria, para que se pueda identificar a qué vector corresponden los datos almacenados. Esta distribución se especifica en la Tabla 8.

De esta tabla se puede concluir que existe para el almacenamiento de las rutas un espacio reducido. Ante este problema se vio la necesidad de diseñar una codificación que permita ahorrar espacio en la memoria EEPROM virtual, pero que permita la recuperación de todos los datos en el momento en que se necesite leer de nuevo la ruta. Inicialmente se debe conocer qué tipo de datos se debe almacenar para poder determinar la forma correcta de codificarlos. El vector que se codifica es el vector Ruta, cuyos valores posibles y su significado se muestran en la Tabla 9.

Tabla 8. Distribución de memoria EEPROM virtual

Posiciones de memoria	**Número de posiciones asignadas**	**Vector al cual se le asignó memoria**
0 a 143	144	Mapa laberinto básico
144 a 287	144	Mapa laberinto avanzado
288 a 397	109	Ruta laberinto básico
398 a 399	2	Coordenadas de inicio y depósito del laberinto básico
400 a 509	109	Ruta laberinto avanzado
510 a 511	2	Coordenadas de inicio y depósito del laberinto avanzado

Tabla 9. Valores posibles dentro del vector ruta

Valor	**Hacia qué celdilla debe dirigirse el robot**
0	Adelante
1	Diagonal derecha arriba
2	Derecha
3	Diagonal derecha abajo
4	Atrás
5	Diagonal izquierda abajo
6	Izquierda
7	Diagonal izquierda arriba
10	Rampa adelante
12	Rampa derecha
14	Rampa atrás
16	Rampa izquierda

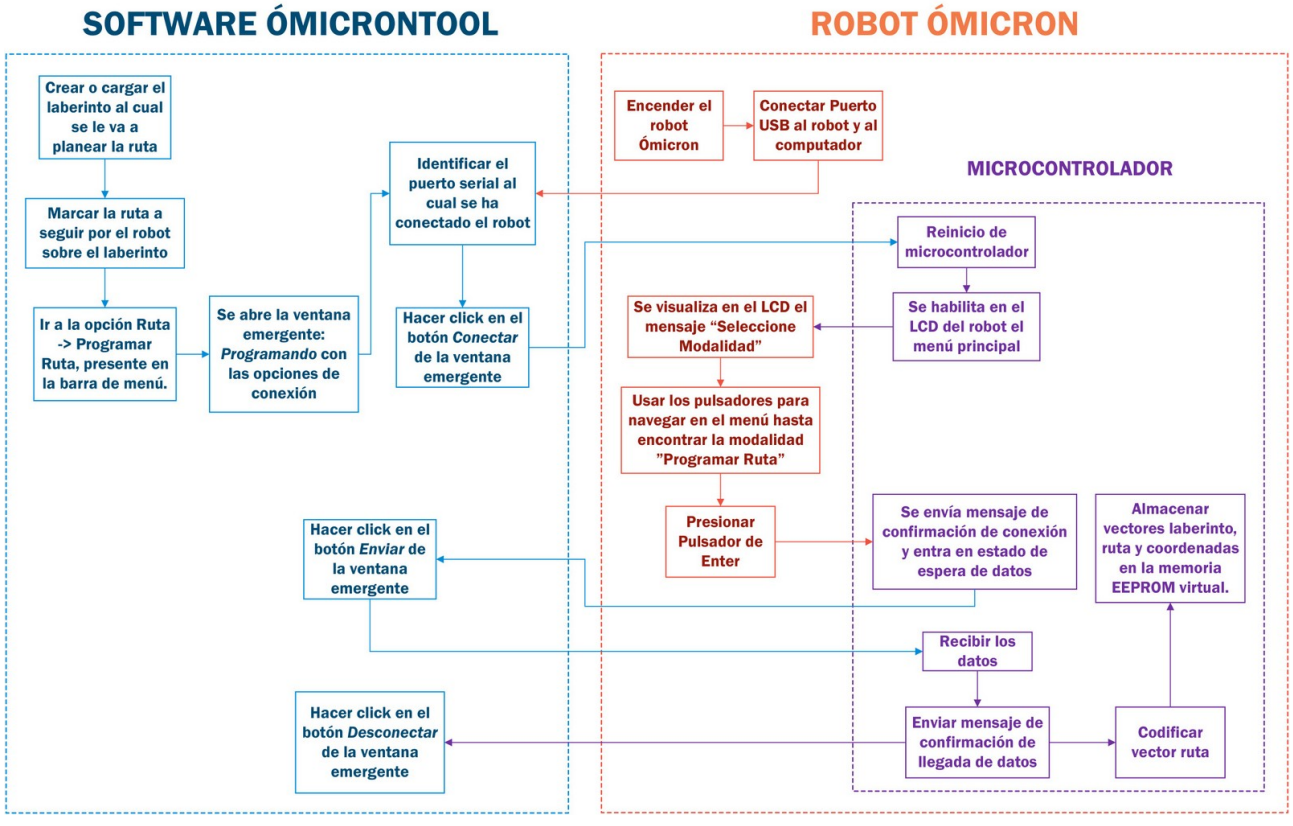


Figura 55. Interacción entre el robot Ómicron y el software ÓmicronTool

Teniendo en cuenta esta información se crearon las siguientes reglas de codificación:

1. Si el valor es igual a cero, se suman la cantidad de ceros consecutivos en la ruta, hasta llegar a un máximo de 24. Multiplicar el valor obtenido por 10 y sumar 8. El número máximo de ceros consecutivos que se pueden sumar es 24, ya que si se le suma un cero más el número resultante de la codificación es 258, número que no se puede almacenar en EEPROM. Se debe sumar 8 ya que este número no hace parte de la lista de valores posibles en el vector Ruta y es el que indica en la decodificación, que el número representa una secuencia de ceros.
2. Si el valor es igual a 2, el valor siguiente es menor que 5, y el siguiente menor que 9,se multiplica el primer número por 100, el segundo por 10 y se suman junto con el tercer número. Estas especificaciones tan precisas se deben a la necesidad de asegura que el número que resulte de la codificación no sea mayor a 255.
3. Si el valor es igual a 1, el valor siguiente es menor que 9, y el siguiente menor que 9, se multiplica el primer número por 100, el segundo por 10 y se suman junto con el tercer número.
4. Si el valor de la ruta es menor que 10 y el valor siguiente también, y si además no cumplen las reglas 2 y 3, se multiplica el primer número por 10 y se le suma el segundo.
5. Si el valor es menor que 10 y no cumplen las reglas 2, 3 y 4, se almacena el número sin ninguna modificación
6. Si el valor es igual a 10, se suman la cantidad de 10 consecutivos en la ruta, hasta llegar a un máximo de 24. Multiplicar el valor obtenido por 10 y sumar 9. El número máximo de dieces consecutivos que se pueden sumar es 24, ya que si se le suma un 10 más el número resultante de la codificación es 259, número que no se puede almacenar en EEPROM. Se debe sumar 9 ya que este número no hace parte de la lista de valores posibles en el vector ruta y es el que me indica en la decodificación, que el número representa una secuencia de dieces.
7. Si el valor es mayor que 10, se almacena el número sin ninguna modificación.
8. Si el valor es igual a 100, se cambia por el número cero.

Siguiendo las reglas planteadas anteriormente se puede lograr codificar la ruta de tal forma que se reduzca a más de la mitad el espacio requerido para almacenar la ruta asegurándose al mismo tiempo que los datos puedan ser decodificados y llevados a su estado original.

El proceso de decodificación consiste en tomar los valores codificados en unidades, decenas y centenas, y a partir de allí seguir las siguientes reglas de decodificación:

1. Si el valor codificado es igual a cero, el valor del vector ruta es 100.
2. Si el valor codificado es mayor o igual a 100 y menor o igual a 255, y además las unidades son iguales a 8, se multiplica el valor de las centenas por 10 y se le suma en valor de las decenas y se le agrega al vector ruta el número de ceros consecutivos indicados por el valor resultante.
3. Si el valor codificado es mayor o igual a 100 y menor o igual a 255, y además las unidades son iguales a 9, Se multiplica el valor de las centenas por 10 y se le suma en valor de las decenas y se le agrega al vector ruta el número de dieces consecutivos indicados por el valor resultante.

4. Si el valor codificado es mayor o igual a 100 y menor o igual a 255, y no cumple ni la regla 2 ni la 3, se agregan los valores de las centenas, decenas y unidades al vector Ruta en ese mismo orden.
5. Si el valor codificado es mayor o igual a 11 y menor o igual a 17, no se requiere decodificación, se agrega el valor al vector ruta.
6. Si el valor codificado es mayor o igual a 18 y menor o igual a 99, y además las unidades son iguales a 8, se le agrega al vector ruta el número de ceros consecutivos indicados por el valor de las decenas.
7. Si el valor codificado es mayor o igual a 18 y menor o igual a 99, y además las unidades son iguales a 9, se le agrega al vector ruta el número de dieces consecutivos indicados por el valor de las decenas.
8. Si el valor codificado es mayor o igual a 18 y menor o igual a 99, y no cumple ni la regla 6 ni la 7, se agregan los valores de las decenas y las unidades al vector ruta en ese mismo orden.
9. Si el valor codificado es mayor o igual a 1 y menor o igual a 7, no se requiere decodificación, se agrega el valor al vector ruta.

4.2.2. Solución de Laberinto Básico y Avanzado

El diagrama básico para la solución de estos tipos de laberintos se puede observar en la Figura 56. Se basa en una máquina de estados finita para garantizar el orden de las tareas que se deben cumplir en cada etapa.

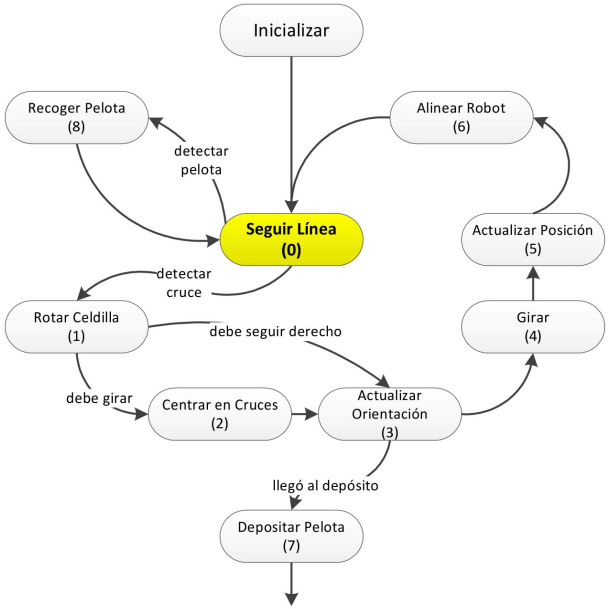


Figura 56. Estructura general del laberinto básico y avanzado

En la inicialización se toma información necesaria para desempeñar correctamente las siguientes funciones. Las variables que se inicializan son: Posición absoluta, Orientación

absoluta, vector Laberinto, vector Ruta y vector Coordenadas. Estos tres últimos se explicaron en la sección anterior. La posición y la orientación absoluta son variables que el sistema está pendiente de actualizar siempre.

- Posición Absoluta: Esta variable indica en qué celdilla del laberinto se encuentra el robot, por lo que su valor va de 0 a 143, siendo 0 la primera celdilla de la parte superior izquierda de la zona de dibujos de ÓmicronTool, mientras que 143 es la última celdilla de la parte inferior derecha (Ver Figura Figura 57).
- Orientación Absoluta: indica la dirección del robot en la celdilla que en que se encuentre actualmente, pero, a diferencia de la información del vector Ruta, esta orientación no es referenciada respecto al robot, sino respecto al laberinto. Debido a que en una celdilla, el robot tiene como máximo 8 posibles caminos, esta variable va de 0 hasta 7 como indica la Figura 57. Cabe resaltar que el cero de la orientación absoluta coincide con el norte del laberinto definido en ÓmicronTool, no con el norte terrestre.

Tareas de ejecución crítica. El robot Ómicron es un sistema que depende de varios estímulos provenientes del medio para completar las tareas que se le programen, tales como seguir una línea, medir la distancia recorrida, detectar la presencia de pelotas, paredes, entre otras. Lo anterior implica que la medición de estas variables externas debe realizarse de manera periódica e ininterrumpida, para lo cual se hace uso de las herramientas de hardware con las que cuenta el controlador del robot. El microcontrolador utilizado en el robot Ómicron

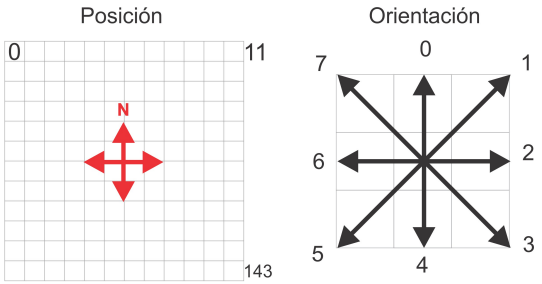


Figura 57. Posición y Orientación Absoluta

cuenta con varios temporizadores que están en la capacidad de medir intervalos de tiempo de manera precisa, y que al hacer uso de esta propiedad pueden garantizar la ejecución de cierto número de instrucciones de manera periódica, mediante la interrupción del flujo de programa principal. Particularmente, se sacó provecho de esta función de interrupción por tiempo para leer todos los sensores con una periodicidad de 2 ms, definiendo así un periodo de muestreo general del sistema.

En la Figura 58 se observan las distintas funciones que se ejecutan con cada interrupción del flujo de programa principal, así como la porción de tiempo que tarda cada una en evacuarse. Es evidente que la porción de tiempo empleado en las funciones críticas no es excesiva, ocupando solamente el 5% del tiempo total del ciclo de 2 ms, por lo cual no interfiere con las tareas que se estén ejecutando en el flujo principal.

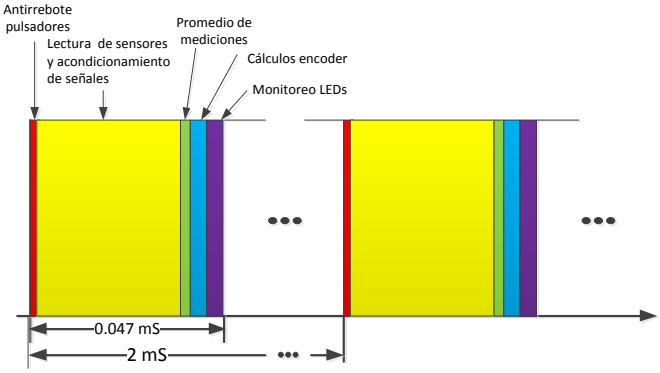


Figura 58. Diagrama de tiempos

Seguidor de línea. El sistema de seguidor de línea en el robot Ómicron es un elemento esencial para la ejecución de movimientos por parte del robot, ya que, con base en la información aportada por los sensores y de las acciones realizadas por los actuadores, permite guiar al robot a través del laberinto. En otras palabras, el sistema de seguidor de línea es, en esencia, un sistema en el cual se busca controlar la posición del robot respecto a un eje.

Se puede definir al algoritmo del seguidor de línea implementado, visto desde la perspectiva de los sistemas de control, como de tipo *On-Off*, uno de los métodos de control más simples existentes. Fue posible usar este método de control principalmente por la alta sensibilidad del sistema ante cambios, ya que esto permite corregir el error en la variable controlada a tiempo y por ende, evitar que éste se amplifique y desestabilice el robot.

Rotar Celdilla. La ruta que debe seguir el robot Ómicron se basa en direcciones referenciadas respecto a la orientación del robot (sección 4.1.2.). Para que el robot pudiese interpretar estas direcciones, se implementó como parte del algoritmo una función que permite rotar de manera virtual la celdilla en la cual el robot se encuentra, y de esta manera estandarizar las direcciones especificadas previamente en la ruta.

Se observa en la Figura 59 un ejemplo de la función Rotar Celdilla, analizado respecto al laberinto y desde la perspectiva del robot, y además se resalta en amarillo la celdilla a rotar.

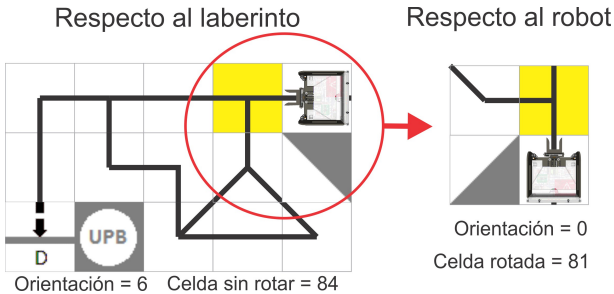


Figura 59. Ejemplo de la función Rotar Celdilla

Centrar en Cruce. Una vez el robot determina qué ruta debe tomar en la celdilla que se encuentra, debe proceder a la ejecución de los movimientos correspondientes. El primer paso para tomar el camino correcto es ubicar el centro de la celdilla y alinear el centro de giro del robot con éste. En el estado anterior se obtuvo la información del cruce en el que se encuentra, y dependiendo del tipo de cruce, el robot debe realizar alguna de las siguientes acciones: avanzar, retroceder o quedarse quieto. En la Figura 60, se pueden observar las tres técnicas usadas para centrar el robot en los diferentes cruces:

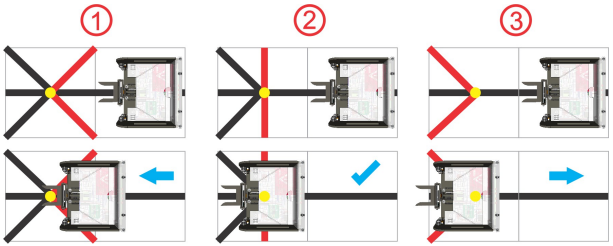


Figura 60. Ejemplo de la función Centrar en Cruce

1. Cuando las celdillas contienen alguna de las líneas resaltadas en rojo, el robot debe avanzar una distancia determinada para llegar al centro.
2. En este caso, el robot ya se encuentra en el centro del cruce.
3. Aquí, el robot debe retroceder una distancia determinada y así volver al centro del cruce.

Sólo hay un caso en el cuál el robot no se debe centrar en los cruces, y es cuando éste debe seguir derecho ignorando cualquier otra línea.

Actualizar Orientación Y Girar. El giro se realiza dependiendo del valor del vector Ruta en la posición de la celdilla actual. En la Figura 61 se puede observar el diagrama de flujo de esta función. Como se puede observar, la distancia recorrida, dada por la lectura de los encoders, es de vital importancia para diferenciar los giros de 45° y 90°.

Actualizar Posición. Esta función es la que se encarga de actualizar la posición Absoluta del robot mencionada anteriormente (ver Figura 57). Depende de la orientación absoluta, y se ejecuta una vez se haya detectado un cruce o se haya recorrido una distancia mayor a la del tamaño de una celdilla, indicando que el robot ya se encuentra en otra celdilla. Dicha distancia se calcula con la ayuda de los encoders.

Alinear Robot. Esta función se realiza inmediatamente luego de que se termina un giro y se actualiza la posición, y permite reubicar el robot en la línea y estabilizarlo para continuar su camino. Se encarga también de que el robot no vuelva a detectar cruces falsos mientras se estabiliza.

Recoger y Depositar Pelotas. En todo el recorrido, el robot siempre está pendiente de la detección de las pelotas que debe recolectar por todo el laberinto. Para ello, cuando hay una

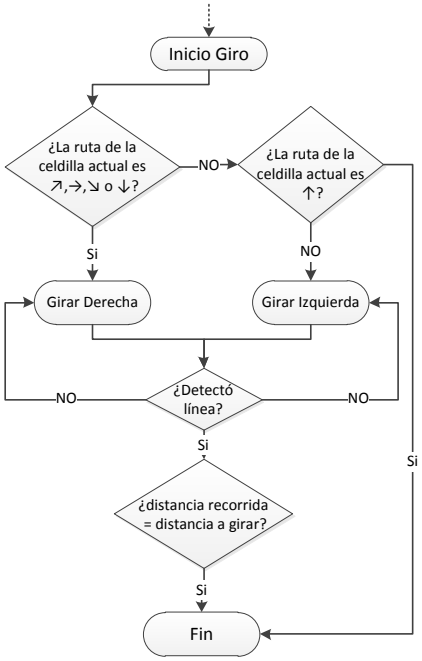


Figura 61. Diagrama de flujo de la función Girar

detección, el robot debe retroceder para bajar la pala, ya que Ómicron siempre anda con la pala arriba para evitar choques.

Aprovechando los sensores de línea en la parte trasera, la reversa no se realiza a ciegas, sino siguiendo línea pero en sentido contrario. Una vez el robot reversa por determinado tiempo, baja la pala y procede a seguir línea hasta detectar nuevamente la pelota. En esta nueva detección el robot sube la pala sin necesidad de parar, y sigue su recorrido nuevamente.

Cuando el robot detecta la celdilla de depósito, entra en el estado Depositar Pelotas, el cuál utiliza el sensor de distancia frontal para reconocer que ha llegado al borde del depósito y paso seguido, baja la pala e inclina el depósito, permitiendo que las dos pelotas sean liberadas.

4.2.3. Solución de Laberinto Experto

A partir del año 2013 se implementó, también como parte de este trabajo de grado, una nueva modalidad de la categoría laberinto en la Olimpiada Robótica A+D. En resumen, ésta consiste en que el robot ingrese a un laberinto desconocido y encuentre dos pelotas que luego deberá llevar hasta un depósito que es, al mismo tiempo, la salida o meta del laberinto. Para conocer los detalles de esta competencia leer Anexo D.

Para la solución de este nuevo laberinto se realizó un programa basado en el algoritmo *Backtracking*, también conocido como Vuelta atrás, el cual consiste en una búsqueda sistemática de la solución del problema, es decir, intentar todas las posibilidades hasta llegar a su solución. Se caracteriza principalmente en que en caso de no encontrar solución por una de las posibilidades elegidas, vuelve a estados anteriores en busca de

opciones diferentes [40]. Un esquema básico del algoritmo de *Backtracking* se muestra en la Figura 62.

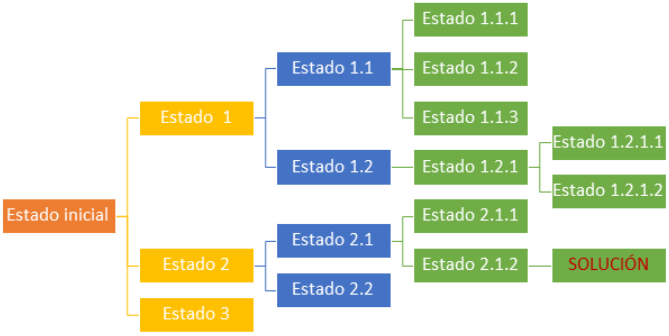


Figura 62. Esquema básico del algoritmo *Backtracking*

El esquema de este algoritmo se asemeja a las ramas de un árbol, que se va extendiendo hasta llegar a un estado donde no es posible continuar avanzando, en este punto se debe regresar a un estado anterior donde exista posibilidades de seguir extendiendo las ramas. Este proceso continúa hasta llegar a encontrar la solución del problema.

Para la solución del laberinto experto es indispensable tener en cuenta que no se debe únicamente llegar a la salida, sino que antes de llegar a ella se debe encontrar dos pelotas, por lo tanto, para la solución del problema es necesario cumplir con tres objetivos diferentes dentro del laberinto. Por esta razón es necesario modificar el algoritmo de *Backtracking* para adaptarlo a lo que se exige en esta competencia. Un diagrama general del método de solución diseñado se muestra en la Figura 63.

Para comprender la lógica del programa implementado es necesario entrar en mayor detalle en el código del programa, comenzando con explicar las variables usadas para almacenar la secuencia de giros ejecutados por el robot y la forma en que se realiza el recorrido del mapa. Por lo que a continuación se explicará la función de las principales variables presentes en el programa:

- *distancia_cruce*. Variable de tipo entero: Indica la distancia promedio presente entre dos cruces consecutivos, medidos en número de señales enviadas por el encoder.
- *conteo_cruce_D*. Variable de tipo entero: Indica la distancia recorrida por el robot, medida por las señales enviadas por el encoder derecho. Se reinicia, cada vez que se determina que el robot se encuentra en una nueva coordenada dentro del laberinto.
- *conteo_cruce_I*. Variable de tipo entero: Indica la distancia recorrida por el robot, medida por las señales enviadas por el encoder Izquierdo. Se reinicia, cada vez que se determina que el robot se encuentra en una nueva coordenada dentro del laberinto.
- *conteo_cruce*. Variable de tipo entero: Indica la distancia recorrida por el robot, determinada promediando el valor de las dos variables anteriores. Indispensable para determinar cambio de ubicación dentro del laberinto.

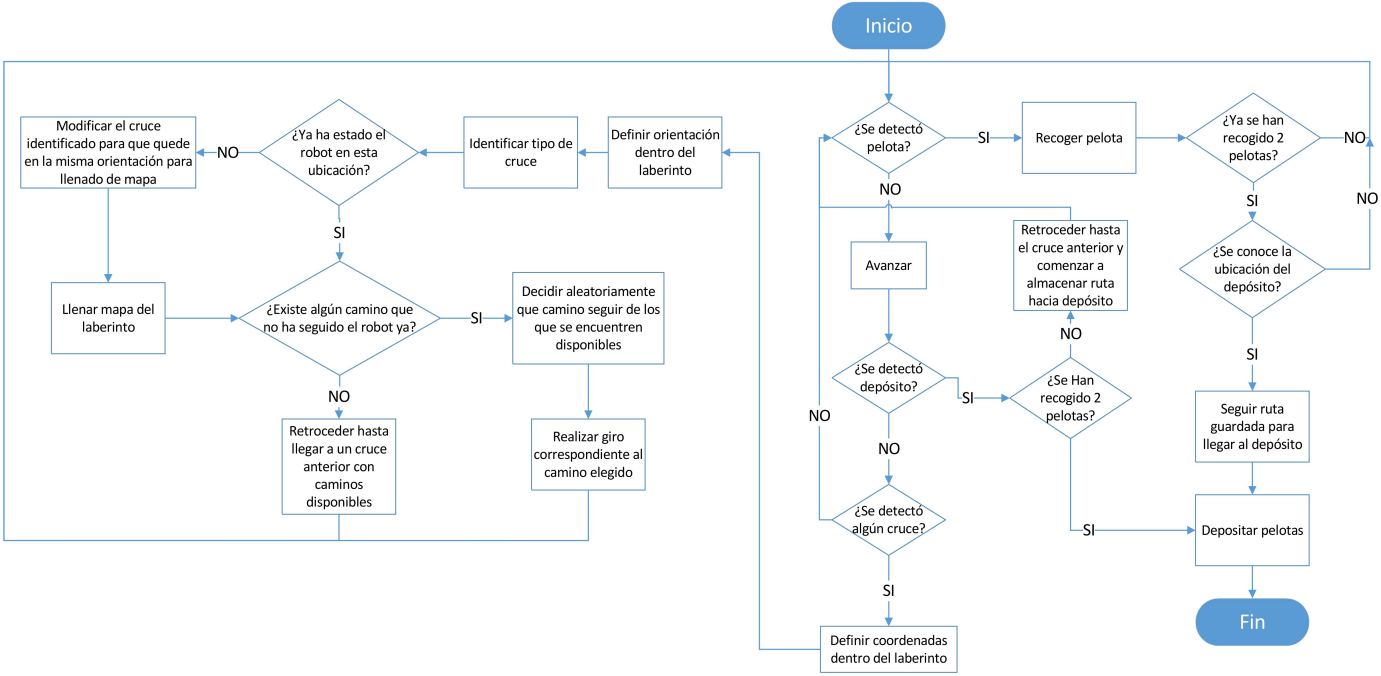


Figura 63. Diagrama general del programa de solución de laberinto experto

- *primer_cruce*. Variable de tipo Booleano: Si se encuentra en estado *true* indica que ya se ha identificado el primer cruce dentro del laberinto y que por lo tanto ya se ha comenzado a llenar el mapa.
- *laberintoExp*. Matriz de bytes de 19 × 19: Contiene el tipo de cruce encontrado en cada una de las ubicaciones o coordenadas dentro del laberinto. Dichos cruces se almacenan teniendo en cuenta la misma orientación para mantener la congruencia del mapa guardado.
- *cruces_posibles*. Matriz de bytes de 19 × 19: Indica qué caminos se encuentran disponibles en cada una de las ubicaciones del mapa. Cada vez que el robot decide irse por un camino en una coordenada específica, se almacena en esta matriz los caminos que no se han tomado y que por lo tanto siguen disponibles.
- *secuencia_Coord*. Matriz de bytes de 400 × 5: Indica la secuencia de los caminos elegidos por el robot durante todo el proceso de solución del laberinto. Almacena las coordenadas y el sentido del robot en cada ubicación, el tipo de cruce identificado y el camino elegido.
- *ruta_a_deposito*. Matriz de bytes de 100 × 2: Si se ha identificado la ubicación del depósito antes de encontrar las dos pelotas, se almacena en esta matriz el camino que se debe seguir para llegar de nuevo al depósito una vez se recojan las pelotas. Se guarda la orientación del robot y el giro que realizó.
- *num_secuencia*. Variable de tipo entero: Indica en qué posición de la matriz *secuencia_Coord* se encuentra.
- *num_sec_deposito*. Variable de tipo entero: Indica en qué posición de la matriz *ruta_a_deposito* se encuentra.
- *num_sec_regreso*. Variable de tipo entero: Indica en qué posición de la matriz *secuencia_Coord* se encuentra en caso de tener que regresar a un cruce anterior en busca de caminos disponibles.
- *linea_adelante*. Variable de tipo Booleano: Toma el estado *true* cuando en un cruce se identifica que existe línea hacia adelante. Vuelve a estado *false* una vez se identifique el tipo de cruce presente en la ubicación actual.
- *regreso*. Variable de tipo Booleano: Toma el estado *true* si el robot se está devolviendo en la secuencia de giros en busca de caminos disponibles. Vuelve a estado *false* una vez encuentre una trayectoria a seguir.
- *deposito_identificado*. Variable de tipo Booleano: Toma el estado *true* si se ha identificado el depósito dentro del laberinto y no se han recolectado las 2 pelotas. Es la variable que indica que se debe comenzar a llenar la matriz *ruta_a_deposito*.
- *detectar_pelota*. Variable de tipo Booleano: Toma el estado *true* cuando se identifica la presencia de una pelota.
- *todas_pelotas_recogidas*. Variable de tipo Booleano: Toma el estado *true* cuando se identifica que se han recolectado las 2 pelotas presentes en el laberinto.
- *continuar_ruta_deposito*. Variable de tipo Booleano: Toma el estado *true* cuando se puede continuar llenando la matriz *ruta_a_deposito* inmediatamente después de que el robot debió retornar en la secuencia en busca de caminos disponibles.
- *regresando_a_deposito*. Variable de tipo Booleano: Toma el estado *true* una vez se determine que se recolectaron todas las pelotas del laberinto y se conoce la ubicación del depósito. Indica que el robot puede volver al depósito.

- *"tiempo_recoger_pelota y tiempo_recoger_pelota_anterior"*. Variables de tipo Long: La diferencia entre ambos tiempos indica el tiempo trascurrido desde la última vez que se intentó recoger una pelota y el momento actual. Indispensable para determinar cuántas pelotas se han recogido.

Ya conocidas las variables que hacen parte de la solución del laberinto experto, se puede entrar en detalle a describir el programa. En primer lugar es necesario aclarar por qué el tamaño de la matriz en la que se almacena el laberinto es 19×19, teniendo en cuenta que el tamaño real del laberinto a solucionar es de máximo 10×10. Se definió este tamaño para la matriz para asegurarse de no leer espacios de memoria no declarados y causar un error en el programa, ya que no se conoce el punto de partida del laberinto ni la orientación inicial del robot (sólo se sabe que comienza en una de las esquinas de la matriz). Es así como la coordenada inicial al comenzar a llenar el mapa del laberinto experto corresponde a la columna 9 fila 9 de la matriz, su punto más central, para así asegurar que aunque se mueva 9 espacios en cualquier dirección, nunca se violará la memoria asignada a la matriz.

Por otro lado, la matriz del laberinto experto no comenzará a llenarse hasta encontrar el primer cruce, es decir, se ignorarán todos los tramos rectos presentes antes de que los sensores de cruce del robot detecten el primer giro posible a la derecha o izquierda. Esto se hace para asegurar que el conteo de los encoders entre cruce y cruce se haga correctamente. Es importante aclarar que a pesar de que se ignore los tramos rectos iniciales en el mapa, si en este recorrido se encuentra una pelota, ésta será recolectada.

Una vez se detecte el primer giro o cruce, el programa comienza a llenar el mapa del laberinto y la secuencia de giros que va realizando. En este punto es primordial explicar en detalle la matriz *secuencia_Coord* y los valores que almacena. En la Figura 64 se puede ver un esquema de la forma que tiene esta matriz.

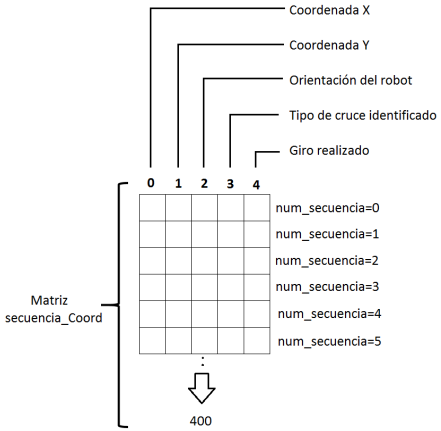


Figura 64. Esquema de matriz *secuencia_Coord*

La matriz *secuencia_Coord* está formada por 5 columnas y hasta 400 filas. El número de la fila está determinado por el valor de *num_secuencia*, que va incrementando su valor cada

vez que se presente un cambio de ubicación del robot dentro del laberinto. En cada una de las columnas se almacenan datos diferentes correspondientes a una misma posición de la secuencia. El primer dato a almacenar son las coordenadas, que indican en qué posición del mapa del laberinto se encuentra el robot. Existen dos coordenadas, la coordenada X, correspondiente a las columnas de la matriz que contiene el mapa del laberinto, y la coordenada Y, que corresponde a las filas.

En la columna 0 de la matriz *secuencia_Coord* se almacena la coordenada X, mientras que en la columna 1 se guarda la coordenada Y del robot dentro del laberinto. Los valores almacenados entonces están en un rango de 0 a 18. Determinar la orientación del robot es vital para poder almacenar el mapa del laberinto y tomar el camino correcto dentro de éste. Existen 4 orientaciones posibles, esquematizadas en la Figura 65.

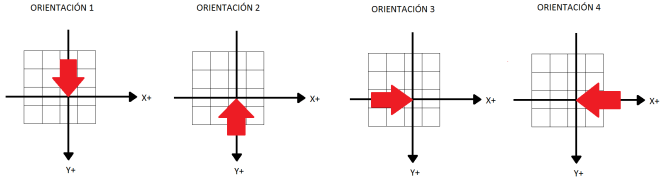


Figura 65. Orientaciones posibles del robot en modo experto

En esta figura es posible ver la manera en la que van dirigidos los ejes de las coordenadas X e Y, los cuales se consideran que crecen en la dirección en que aumentan las filas y las columnas de la matriz del mapa del laberinto, *laberintoExp*. Las orientaciones se almacenan en la columna 2 de la matriz *secuencia_Coord* y puede tomar valores de 1 a 4 de acuerdo a lo siguiente:

- Orientación 1: El robot viene de $Y-$ a $Y+$
- Orientación 2: El robot viene de $Y+$ a $Y-$
- Orientación 3: El robot viene de $X-$ a $X+$
- Orientación 4: El robot viene de $X+$ a $X-$

En el mapa del laberinto, es decir la matriz *laberintoExp*, todos los valores se almacenan como si el robot se encontrara en la orientación 2, para mantener la congruencia del laberinto. Para reconocer en qué clase de cruce se encuentra el robot, se leen los sensores de cruce y línea, identificando en cuál de las direcciones sigue el camino del laberinto. Para almacenar la clase de cruce es necesario codificar la información recibida por los sensores, como se muestra en la Figura 66.

El tipo de cruce se codifica entonces sumando los códigos de los 4 cruces bases, dependiendo de las líneas identificadas por los sensores, tal como se ve en la Figura 66. El tipo de cruce se almacena en la columna 3 de la matriz *secuencia_Coord* y puede tomar valores entre 1 y 15. Luego de identificar el tipo de cruce en la orientación actual del robot, se debe hacer la trasformación de dicho valor, para que éste tenga como referencia la orientación 2 y así pueda ser almacenado en la matriz *laberintoExp*, donde se guarda el laberinto.

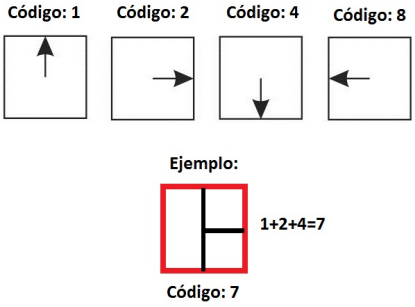


Figura 66. Codificación para identificar y almacenar tipo de cruce en modo Experto

Ya habiendo definido que caminos hay disponibles en la ubicación en que se encuentra el robot, se puede pasar a decir hacia donde debe girar. Se debe entonces decodificar los cruces disponibles para esta posición, almacenados en la matriz *cruces_posibles*, y partiendo de estos decidir de manera aleatoria cual giro realizar. El valor que corresponde a cada uno de los giros que puede realizar el robot se muestra en la Figura 67.

En la columna 4 de la matriz *secuencia_Coord* se almacena el código del giro que realizó el robot, por lo tanto puede tomar valores entre 0 y 3. Una vez terminado el giro que se eligió aleatoriamente se comienza un nuevo ciclo y se almacenan los datos de las siguientes coordenadas en la secuencia. Con los

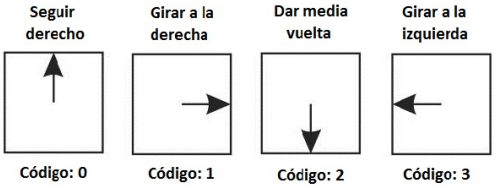


Figura 67. Giros posibles

datos almacenados en cada ciclo, se obtiene la información necesaria para saber las coordenadas y orientación de las siguientes secuencias, de igual manera, es posible saber en qué posición del laberinto se encuentra el robot en todo momento y los posibles caminos que puede seguir. También son indispensables en el momento en que el robot deba devolverse en la secuencia en búsqueda de cruces disponibles en caso de agotar las posibilidades en el camino elegido y para almacenar la ruta que lleva de vuelta al robot al depósito, en caso de que éste se hubiese identificado antes de recolectar todas las pelotas.

5. CONCLUSIONES

El robot Ómicron, en conjunto con el software ÓmicronTool, cumplen el objetivo de ser una herramienta didáctica de fácil manejo con el cual el usuario puede realizar demostraciones para las competencias de laberinto de las Olimpiadas Robóticas A+D y al mismo tiempo es útil para mostrar conceptos básicos de robótica, electrónica, mecánica y programación involucrada en la creación de un robot AGV.

Se realizó un estudio del estado del arte en lo referente a robótica móvil con fines educativos, así como a las distintas competencias de robótica afines a nivel mundial, lo que permitió plantear un punto de partida con base en los casos de éxito más compatibles con los objetivos del proyecto, así como en la literatura disponible.

Se comprobó que las plataformas de desarrollo tipo Arduino proveen la infraestructura adecuada para implementar sistemas de hardware y Firmware, ya que su caracterísitica *Open Hardware* ha facilitado un mejoramiento constante y ha permitido que la comunidad de desarrolladores aumente su funcionalidad por medio de librerías, las cuales aportan funciones que pueden ser necesarias en la aplicación, lo que implica un ahorro de tiempo en el desarrollo del producto

El uso de un entorno de desarrollo de software con ayudas para la creación de interfaces gráficas como *Microsoft Visual C#* permitió acelerar la programación de software. Además, el hecho de que este entorno utilice un lenguaje orientado a objetos facilita la creación de interfaces de usuario amigables y versátiles.

El diseño de un robot requiere una perfecta sincronización entre la parte mecánica, electrónica y la programación ya que deficiencias en una de estas 3 esferas pueden ser compensadas con una optimización de las demás. Pero de igual forma la falla de dos de estas esferas puede representar problemas irreparables en la esfera restante causando que el funcionamiento del robot en general no sea el idóneo. Este fenómeno se evidenció notoriamente en el proceso de lograr que el robot siguiera línea, ya que problemas mecánicos, la transmisión delantera y sensores de línea ubicados alineados con el eje de giro, junto con un problema con los componentes eléctricos, los motores ofrecían un rango de trabajo muy reducido, causó que el robot fuera inestable y difícilmente controlable por lo que no se pudo llegar a un programa para seguir línea óptimo. La posterior solución de los problemas mecánicos y eléctricos permitió que con una programación sencilla el robot siguiera línea correctamente.

Para lograr que un robot móvil seguidor de línea pueda desplazarse de manera fluida y constante, es indispensable que los sensores destinados a detectar la línea estén lo más adelante posible de su eje de giro, de esta forma se puede conocer la posición del robot sobre la línea con la anticipación suficiente para determinar qué acción tomar para seguir adecuadamente el camino marcado.

6. RECOMENDACIONES

El prototipo, resultado del presente proyecto, no pretende ser final y definitivo. Se diseñó pensando en que mantenga su vigencia y pueda evolucionar junto con las Olimpiadas Robóticas A+D. Para lograr esto el robot cuenta con sensores adicionales no utilizados hasta el momento, estos son: 2 sensores de distancia ubicados en las paredes laterales del robot y una brújula electrónica, los cuales fueron incluidos previendo que en el futuro se tendrá una competencia en que los robots no tendrán ruta en cinta negra como guía dentro del laberinto. El robot también es casi en su totalidad desarmable lo que deja abierta la posibilidad de ser modificado físicamente con el fin de adaptarse a nuevas características en los laberintos a resolver.

El método de control utilizado en el robot Ómicron, a pesar de que mantiene estable al sistema, hace que opere a velocidades muy lentas, por lo que se recomienda para trabajos futuros la implementación de un control de posición y velocidad más sofisticado que permita obtener un rendimiento superior al actual.

REFERENCIAS

[1] S. Avery, "Roborodentia project page," 2001.

[2] A. Ollero, *Robótica, Manipuladores y robots mobiles*. Marcombo, 2007.

[3] J. Suarez, "Robot laberinto caterpillar," *XIII Olimpiada Robótica A+D*, 2011.

[4] A. F. López and J. López, "Mercaba," *XIII Olimpiada Robótica A+D*, 2011.

[5] The Robot MarketPlace, "Mechanical and drive," 2013.

[6] I. Buchmann, *Batteries in a Portable World: A Handbook on Rechargeable Batteries for Non-Engineers*. 2011.

[7] Arduino, "¿qué es arduino?," 2013.

[8] M. J. Matarić, "Behavior-based systems: Main properties and implications," in *IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pp. 45–54, 1992.

[9] V. Sieben, "A high power h-bridge," 2003.

[10] E. Ruiz, *Educatrónica: Innovación en el aprendizaje de las ciencias y la tecnología.* Díaz de Santos, 2007.

[11] C. F. Tedesco, *Ascensores electrónicos y variadores de velocidad.* Alsina, 2010.

[12] Real Academia Española, "Diccionario panhispánico de dudas," 2010.

[13] Asesoría Integral, "Glosario," 2012.

[14] IEEE, *IEEE Software Engineering Standard: Glossary of Software Engineering Terminology*. IEEE Computer Society Press, 1993.

[15] J. Sanchiz, "Sistema industrial de múltiples vehículos autónomos guiados por láser," 2006.

[16] Microsoft, Visual Studios, "Visual c sharp," 2013.

[17] A. Barrientos, L. Peñín, C. Balaguer, and R. Aracil, *Fundamentos de Robótica*. Mc graw Hill, 2007.

[18] C. Urdiales, "Robótica. historia y tendencias," *Revista Mundo Electrónico*, pp. 44–47, 1999.

[19] G. Bermúdez, "Robóts móviles: Teoría, aplicaciones y experiencia," 2002.

[20] iROBOT, "irobot roomba 530," 2010.

[21] NASA, "Robonaut wakes up in space," 2013.

[22] Ro-botica, "Robótica educativa," 2012.

[23] Parallax Inc., "Basic stamp and propeller education," 2013.

[24] Roborodentia, "About roborodentia," 2013.

[25] Association for Unmanned Vehicle Systems International Foundation, "The purpose of igvc," 2011.

[26] J. L. Jones, B. A. Seiger, and A. M. Flynn, *Mobile Robots Inspiration to Implementation*. Peters, 1999.

[27] First Lego League, "¿que es la fll?," 2013.

[28] Robogames, "Robogames," 2013.

[29] Eurobot, "Welcome to eurobot," 2013.

[30] Robot al Parque, "Vi robot al parque 2011," 2011.

[31] IEEE, "Xi latin american robotics competition," 2013.

[32] C. Cubas, M. Consuegra, R. López, A. Sierra, J. M., F. Méndez, C. Ochoa, F. Panesso, E. Sotter, F. Manotas, and V. Manotas, "Dispositivos autómatas para navegación, detección recolección de pelotas de tenis en ambientes conocidos," *Ingeniería y Desarrollo, Universidad del Norte*, no. 9, pp. 98–112, 2001.

[33] XV Olimpiada Robótica A+D, "Reglamentos," 2013.

[34] Fairchild Semiconductor Corporation, "Qrd1113 / qrd1114 reflective object sensor," 2013.

[35] SHARP Corporation, "Gp2d12 optoelectronic device," 2013.

[36] S. Vorkoetter, "The battery eliminator circuit," 2013.

[37] R. Villa, "Reguladores conmutados," 2013.

[38] Freescale Semiconductor, "5.0 a h-bridge with load current feedback," 2013.

[39] D. Herzog, "How to build a maze," 2013.

[40] S. Cardona, S. Jaramillo, and M. Villegas, *Técnicas de diseño de algoritmos en Java*. 2008.

AUTORES

Sandra Patricia OSORIO GRACIANO, Nace el 31 de Agosto de 1989 en Medellín, Colombia; Bachiller del colegio Jesús María (2006). Ha tomado clases extracurriculares de inglés, piano, canto y tenis de mesa. En el 2008 hizo parte de la logística de ingeniar. Integrante del Semillero A+D desde el 2009. Ha participado en la Olimpiada Robótica en San Gil (2009), en Robogames, San Francisco (E.U.A., 2011) y en la UPB, Medellín (2011 y 2013). Actualmente es egresada de Ingeniería Electrónica de la UPB.

Luisa Alejandra ALVANEZ QUICENO, Nace el primero de Junio de 1989 en Itagüí, Colombia; Bachiller de la Institución Educativa Santo Ángel (2006). Participó en las Segundas Olimpiadas Medellinenses del conocimiento, en las cuales fue semifinalista (2006) y fue ganadora en el programa Mejores bachilleres de la alcaldía de Medellín (2006). Hace parte del Semillero A+D desde el año 2010. Actualmente es egresada del programa de Ingeniería electrónica de la UPB