Pontificia Universidad Católica de Valparaíso.
Facultad de Ingeniería.
Escuela de Ingeniería Informática.

“Aplicación de secuencias de filtros de imágenes enfocados a la resolución de laberintos con robots Lego Mindstorms ®”

Jonathan Alberto Albornoz Román.
Andrés Felipe Morales Farías.

Profesor Guía: **Guillermo Cabrera Guerrero.**

Profesor Co-referente: **Ricardo Soto De Giorgis.**

Carrera: **Ingeniería de Ejecución en Informática.**

Diciembre 2010.

Índice

Resumen

En el presente trabajo de título se efectúan diversas actividades comenzando con una fase de investigación, construcción de un laberinto constituido por bases, conectores y paredes de madera y para la construcción del robot se utiliza un kit Lego Mindstorms NXT 2.0. Posteriormente, se desarrolla un software que permita al robot escapar del laberinto y así, recopilar los resultados de las experiencias efectuadas durante el proceso del mismo y, las conclusiones a las cuales se ha llegado finalmente.

Cabe destacar que se diseña y construye tanto el laberinto como el robot. El laberinto contiene elementos esenciales como son rectas, curvas y bifurcaciones. Con respecto al robot, debió ser dotado de los mecanismos necesarios para que éste se desplace dentro del laberinto de manera eficiente.

El desarrollo del software consiste en primer lugar en lograr una captura del laberinto para luego aplicar una secuencia de filtros, teniendo al menos tres opciones de secuencia. Con la secuencia se procesa la imagen, logrando distinguir el camino del laberinto con respecto a lo que es pared. Luego de aplicar la secuencia de filtros para procesar la imagen, el laberinto será resuelto con aquellos algoritmos que nos aseguren al menos una solución. Posterior a esto, se entregarán las instrucciones al robot mediante bluetooth para que se desplace y logre salir del laberinto.

Todo lo anteriormente descrito, permite establecer tres secuencias de filtros, en donde cada una de ellas procesa la imagen dependiendo de la luminosidad con que sea tomada y aumentar el porcentaje de éxito en los algoritmos de resolución de laberintos. Esto se logra realizando diversas pruebas cuyos resultados son analizados comparativamente con los obtenidos en el trabajo de título de Camilo Espinoza.

Palabras claves: algoritmos de resolución de laberintos, robot Lego Mindstorms, secuencia de filtros, procesamiento de imágenes, Matlab.

Abstract

Several activities were made in this degree title work. Starting with a research phase, then, the construction of a labyrinth made by bases, connectors and wood walls, and finally the construction of a robot, using a Lego Mindstorms NXT 2.0 kit. Subsequently, a software was developed, to let the robot get out from the labyrinth in order to gather the results of the experiences carried out during the process, and to reach final conclusions.

It is to note, that the robot as well as the labyrinth were built and designed. The labyrinth is composed by essential elements like straight roads, curves and bifurcations. Concerning to the robot, it had to be equipped with the mechanism needed to make it move inside the labyrinth in an efficient way.

The software development consists in the first place into getting an image from the labyrinth in order to apply a sequence of filters, having at least three sequences options. Those

sequences are used to process the image, getting to distinguish the path of the labyrinth from the wall. After applying the sequence of filters to process the image, the labyrinth will be solved with those algorithms that assure us at least one solution. Then, instructions will be given to the robot trough bluetooth, in order to make him move and get out from the labyrinth.

All previously described, let us establish three filter sequences, where, each of them process the image, depending on the lightness with they were taken and increase the percentage of success in the labyrinth solving algorithms. This is managed by making several tests, and their results are comparatively analyzed with those obtained in the Camilo Espinoza's degree title work.

Keywords: labyrinth solving algorithms, Lego Mindstorms robot, sequence of filters, image processing, Matlab.

Lista de Figuras

Lista de Tablas

1. Introducción

El trabajo de título a desarrollar en las siguientes páginas se denomina "Aplicación de secuencias de filtros de imágenes enfocados a la resolución de laberintos con robots Lego Mindstorms ®". En él se interrelacionan la informática, considerando la resolución de laberintos y el procesamiento de imágenes con la robótica.

La informática se refiere al procesamiento automático de información mediante dispositivos electrónicos y sistemas computacionales. Por otro lado, la robótica es aquella rama dentro de la ingeniería que se ocupa de la aplicación de la informática al diseño y al uso de máquinas con el objeto de realizar determinadas funciones o tareas. Es decir, la robótica es la ciencia y la tecnología de los robots, que se ocupa del diseño, manufactura y aplicaciones de los robots que crea. En la robótica se combinan varias disciplinas al mismo tiempo, como ser la mecánica, la electrónica, la inteligencia artificial, la informática y la ingeniería de control.

La resolución de laberintos consiste en aplicar métodos que permitan encontrar una o más rutas o caminos desde un punto de partida hasta la salida del laberinto. En tanto, el procesamiento de imágenes tiene como objetivo mejorar el aspecto de las imágenes y hacer más evidentes en ellas ciertos detalles que se desean destacar.

Una de las motivaciones que llevan a desarrollar este trabajo de título es que los conceptos mencionados anteriormente, de naturalezas distintas, deben ser investigados para luego poder relacionarlos y así, lograr el objetivo general planteado en este trabajo que es "Resolver un laberinto con robots Lego Mindstorms aplicando distintas secuencias de filtros de imágenes".

Otra motivación importante es que basándose en la investigación realizada en un trabajo de título anterior, se pretende mejorar la aplicación ampliando la cantidad de secuencias de filtros para optimizar las métricas de rendimiento y, de esta manera, potenciar el proceso que integra la resolución de laberintos, el procesamiento de imágenes y el desplazamiento del robot. Con respecto a este último punto es que se decide cambiar el diseño del robot con el fin de mejorar las métricas mencionadas anteriormente.

El trabajo de título está estructurado en tres fases: la primera de ellas es la fase de investigación, luego el diseño y construcción de un laberinto y de un robot. Finalmente, el desarrollo de un software que permita procesar imágenes y entregar instrucciones al robot para escapar del laberinto.

A continuación, se proporcionan los objetivos que se pretenden desarrollar en el presente trabajo de título, información relacionada con algoritmos de resolución de laberintos, procesamiento de imágenes, robótica y el desarrollo detallado del mismo desde la construcción del laberinto hasta la aplicación final.

2. Metodología

En el siguiente apartado se definen los objetivos del trabajo de título, explicando la metodología y herramientas que se utilizarán para su correcto desarrollo. Cabe destacar que algunos de los objetivos a mencionar han sido tratados previamente en la investigación desarrollada por Camilo Espinoza, egresado de la carrera de Ingeniería Civil Informática de esta casa de estudios.

2.1. Objetivo general

Resolver un laberinto con robots Lego Mindstorms aplicando distintas secuencias de filtros de imágenes.

2.2. Objetivos Específicos

- Analizar herramientas, metodologías y tecnologías para el desarrollo del trabajo de título.
- Desarrollar módulos de procesamiento de imágenes, resolución de laberintos, comunicación y locomoción robot.
- Establecer un análisis comparativo entre los algoritmos de resolución de laberintos.

2.3. Contextualización

Se debe comenzar dejando claro que este trabajo de título está enfocado a perfeccionar lo realizado el año 2009 por Camilo Espinoza, egresado de la carrera de Ingeniería Civil Informática de la Pontificia Universidad Católica de Valparaíso, titulado "Resolución de un laberinto mediante un Lego Robot".

Primero se debe destacar que en ambos trabajos de título los conceptos básicos son los mismos: laberintos, robótica y procesamiento de imágenes. Si damos una mirada general al desarrollo de ambos trabajos son similares en estructura, realizando cambios para intentar marcar diferencias en los resultados que se esperan obtener.

El trabajo de título anterior buscaba "encontrar" una solución aplicando una sola secuencia de filtros al procesar la imagen, sin saber si realmente era la secuencia más efectiva o idónea para el desarrollo, posteriormente, se resolvía el laberinto y se entregaban las instrucciones al robot.

Este trabajo de título comienza en las secuencias de filtros a aplicar, debido a que se harán pruebas y se establecerán al menos tres secuencias distintas, cada una con sus particularidades. Luego de aplicar la secuencia de filtros para procesar la imagen, el laberinto se resolverá con aquellos algoritmos que aseguren al menos una solución. Posterior a esto, se entregarán las instrucciones al robot para que salga del laberinto. Con respecto a este último punto, el robot cambia su diseño en relación al anterior buscando mejorar la performance de éste al interior del laberinto.

Los resultados obtenidos anteriormente son los siguientes:

Tabla 2.1. Resultados trabajo de título anterior

Algoritmo	**Mejor Tiempo**	**Peor Tiempo**
Ratón Aleatorio	00:05:30	-
Seguidor de Paredes	00:08:30	00:13:30
Algoritmo de Tremaux	00:05:30	00:20:00
Llenador de callejones sin salida	00:05:30	00:09:30
Llenador de callejones ciegos	00:05:30	00:05:30
Inundación de laberinto	00:05:30	00:05:30

A partir de estos resultados, se espera poder mejorarlos y generar un informe con un análisis detallado de los mismos.

2.4. Metodología y Herramientas

En cuanto a las metodologías se puede establecer lo siguiente:

- Reuniones semanales de los responsables del trabajo de título para distribuir y organizar tareas.

- Reuniones periódicas con profesor tutor.

- Visitas al laboratorio de robótica cuando se estime para realizar pruebas.

- Con respecto al modelo de procesos a utilizar, se usará un modelo "híbrido" debido a que la base será el modelo iterativo incremental, pero al momento de desarrollar algunos módulos como el de resolución de laberintos se aplicará reutilización, con lo cual se pretende ahorrar tiempo en el desarrollo del trabajo de título.

  La filosofía detrás del modelo es poder desarrollar un sistema de manera incremental, permitiendo al programador sacar ventaja de lo que ha aprendido a lo largo del desarrollo anterior, incrementando, versiones entregables del sistema. En cada iteración, se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema.

  El modelo de proceso consta de dos etapas y un elemento clave propuesto por Craig Larman (Larman, 2003):

  - ❖ Etapa de inicialización: la meta de esta etapa es crear un producto para que el usuario pueda interactuar, y por ende, retroalimentar el proceso.

  - ❖ Etapa de iteración: esta etapa involucra el rediseño e implementación de una tarea de la lista de control de proyecto y, el análisis de la versión más reciente del sistema.
    La meta del diseño e implementación de cualquier iteración es ser simple, directa y modular, para poder soportar el rediseño de la etapa o como una tarea añadida a la lista de control de proyecto.
    Con respecto a las modificaciones, estas debiesen ser más fáciles de realizar conforme avanzan las iteraciones.

  - ❖ Lista de control de proyecto: esta lista contiene un historial de todas las tareas que necesitan ser realizadas. Incluye nuevas funcionalidades para ser implementadas y áreas de rediseño de la solución ya existentes. Esta lista de control se revisa periódicamente como resultado de la fase de análisis.

- Con respecto al enfoque de programación se opta por un enfoque orientado a objetos, debido a que agiliza el desarrollo de software, facilita el trabajo en equipo, fomenta la reutilización y la extensión del código.

En cuanto a las herramientas a utilizar se puede establecer lo siguiente:

- El entorno de desarrollo integrado será Matlab para la programación, que es un software matemático, con su propio lenguaje de programación "M", desarrollado por MatWorks.

- El otro entorno de desarrollo integrado que se utilizará es Eclipse, que es un entorno multiplataforma de código abierto que se usará netamente para desarrollar lo relacionado con JAVA.

- Con respecto a ambos entornos de desarrollo integrado existe compatibilidad al momento de integrar, ya que lo desarrollado en Matlab puede ser usado en Eclipse y viceversa.

- El modelado del sistema en lenguaje UML se utilizará la herramienta StarUml, que es un modelador de diagramas UML de código abierto y flexible.

- En cuanto al robot construido con piezas Lego, tiene su propio sistema operativo "LejOS", el cual permite programar el bloque NXT mediante lenguaje JAVA y no depende de un compilador o un sistema operativo para ser reemplazado.

- Para el procesamiento de imágenes y la captura de imagen mediante webcam, se utilizarán las Toolbox integradas en Matlab, denominadas "Toolbox para el procesamiento de imágenes" y "Toolbox para la adquisición de imágenes".

- Para la comunicación computador – robot, se realizará mediante bluetooth y se utilizará la API de JAVA Icommand 0.7 complementado con la librería de comunicaciones rxtxcomm 2.1-7.

2.5. Propuesta de solución

La propuesta de solución básicamente consiste en que teniendo un laberinto que resolver, se requiere construir un Robot Lego que posea los mecanismos suficientes para desplazarse dentro del laberinto, logrando finalmente, escapar de éste.

Conjuntamente a lo anterior, se da flexibilidad al usuario para capturar la imagen del laberinto ya sea mediante el uso de una cámara web instalada sobre él o, en caso contrario, utilizar una imagen que ya está almacenada en el computador. Una vez logrado lo anterior, el usuario tendrá la posibilidad de escoger la secuencia más idónea, tomando en cuenta el nivel de luminosidad y calidad de la imagen.

Para alcanzar la solución propuesta, es necesario completar de forma ordenada la siguiente serie de actividades:

1. Construcción física de un laberinto.
2. Construcción de un Robot Lego.
3. Captura y procesamiento de imágenes.
4. Implementación de una interfaz de usuario.
5. Implementación de algoritmos de resolución de laberinto.
6. Implementación de mecanismos de comunicación.
7. Implementación de mecanismos de locomoción.
8. Integración.

3. Estado del arte.

3.1. Resolución de laberintos.

A continuación, se nombrarán y explicarán los distintos algoritmos existentes para poder resolver un laberinto. Cabe resaltar que los algoritmos que no garanticen una solución exitosa del laberinto se excluirán, ya que el fin del trabajo de título es analizar los tiempos de resolución de los laberintos y compararlos, por lo que no interesa tener un algoritmo que no garantice encontrar la salida en forma exitosa en la gran mayoría de los casos. Un ejemplo de este tipo de algoritmos es el ratón aleatorio, que trabaja al azar, llega a una bifurcación del laberinto y toma aleatoriamente un camino pero, al no trabajar con memoria no puede recordar el recorrido realizado anteriormente, lo que puede generar loops de tiempo infinito y nunca encontrar la salida. Con lo anteriormente explicado, se comenzarán a detallar los algoritmos que garanticen una solución como mínimo (en el caso de los laberintos con más de una salida).

3.1.1. Algoritmos Exploratorios.

3.1.1.1. Seguidor de pared.

Es un algoritmo muy simple de resolver, pero el laberinto debe estar configurado de tal manera que todas las paredes se conecten entre sí o con el límite externo del laberinto. Si mantenemos la configuración mencionada, este algoritmo siempre encontrará la solución, por lo tanto, podemos tomarlo en cuenta.

El procedimiento para encontrar la solución es el siguiente: Al comenzar, se toma una dirección al azar y luego se comienza a recorrer toda la pared. El sentido que tiene este método es que en el caso de que sea una persona, ésta nunca se perdería y si no pudiese encontrar la salida, volvería al punto inicial de partida (Pullen, 2010).

En la imagen que se muestra a continuación, podemos ver que el algoritmo toma una dirección u otra y la recorre hasta encontrar la salida.

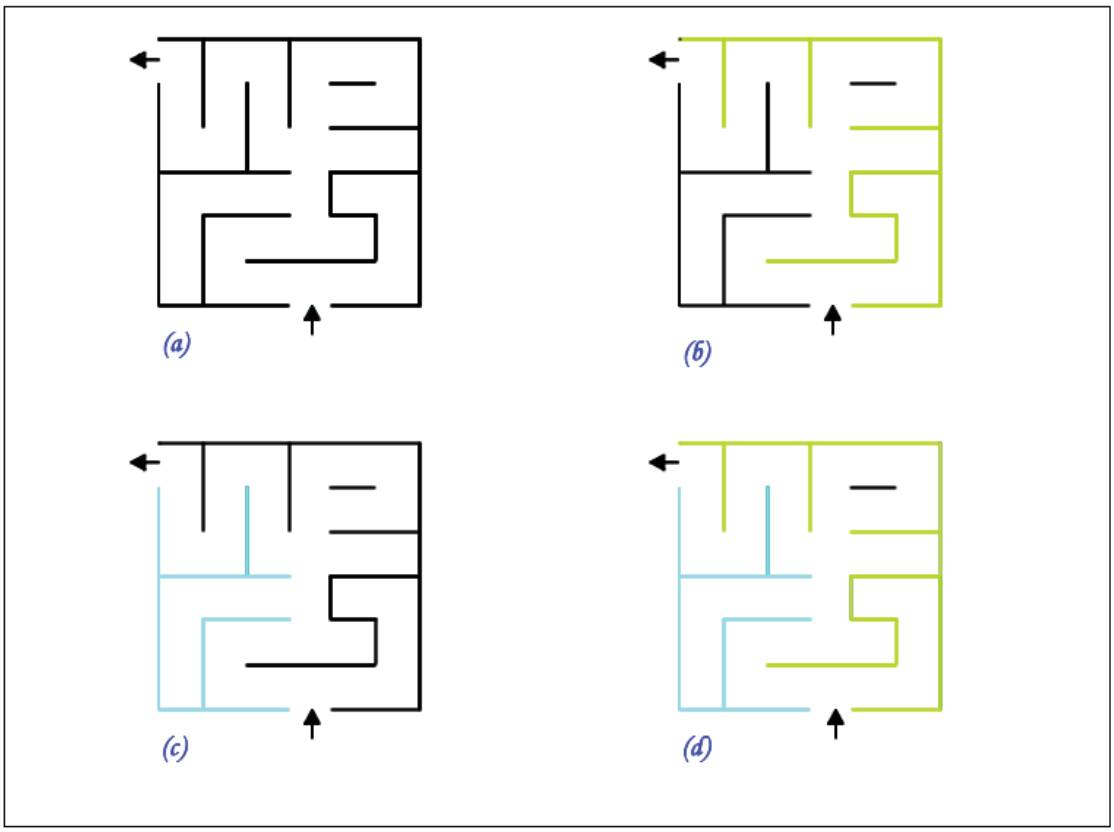


Figura 3.1 Algoritmo seguidor de pared.

3.1.1.2. Backtracker recursivo.

Este algoritmo garantiza encontrar una solución, pero lo que no garantiza es encontrar la solución óptima.

La manera en que trabaja dicho algoritmo es la siguiente: si se encuentra en una pared o en un área que ya haya sido visitada se retorna un fallo. Por otra parte, si se encuentra en la meta se retorna éxito. La idea del algoritmo es que llegado a un punto que no sea la meta, se trate de recorrer el laberinto recursivamente en las cuatro direcciones. Cuando se intenta una nueva dirección se traza una nueva línea en la ruta, y en el caso que se retorne un fallo esta línea se borra, llegando a un momento determinado que tendrá que regresar a un punto, y cuando se ejecute la marcha atrás, la idea es marcar con un valor especial el camino recorrido, para así posteriormente no recorrerlo nuevamente, en esa dirección (Pullen, 2010).

3.1.1.3. Algoritmo de Tremaux.

Es sin duda uno de los algoritmos más eficientes en la resolución de algoritmos de laberintos ya que garantiza la solución de éste para cualquier configuración que tenga. El algoritmo de Tremaux utiliza memoria ya que necesita colocar indicadores en los caminos para ver si se ha transitado por ellos o no. Al llegar a un punto que tenga dos opciones de caminos, el algoritmo toma cualquier dirección aleatoriamente, luego sigue marcando su ruta; si se llegase a encontrar con un camino sin salida, éste se devuelve agregando otra marca al camino (quedando con una doble marca). Si se va transitando por un camino no marcado y se encuentra con una marca, se debe tomar como un callejón sin salida para evitar dar vueltas en círculo dentro del laberinto. En el caso de que se esté devolviendo por un camino (generando una doble marca) y se encuentre una bifurcación marcada, se debe tratar de seguir por un camino no marcado. En el caso que no se pueda, se debe tomar el camino que tenga una sola marca (Pullen, 2010).

En la imagen que sigue podemos ver como la línea roja avanza (va marcando) y la línea azul es la que hace el backtrack (marcación doble).

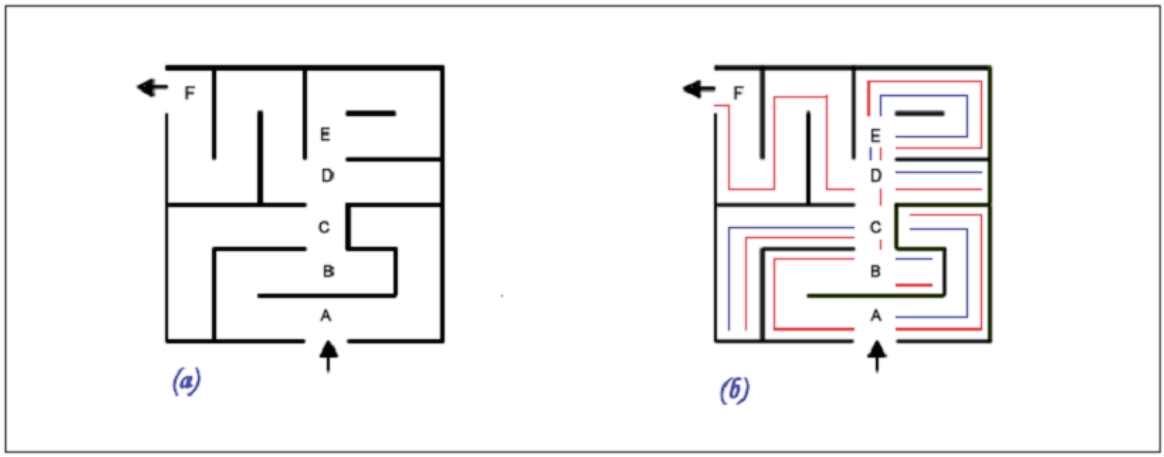


Figura 3.2 Algoritmo de Tremaux.

3.1.2. Algoritmos de Análisis de Caminos.

3.1.2.1. Llenador de callejones sin salida.

Este método lo que hace es llegar hasta el final de un camino sin salida, luego comienza a llenar recursivamente los caminos sin salida generados por el llenado anterior. Este proceso se realiza hasta que finalmente no existan caminos sin salida, lo cual genera un área exclusiva que contiene la solución del laberinto. En el caso que el laberinto no posea caminos sin salida, el algoritmo no realizará ninguna tarea.

Si el laberinto procesado en su configuración no contiene partes disjuntas (puntos que provoquen caminos en círculos), la nueva área de búsqueda corresponderá exactamente a la solución del laberinto. En el caso contrario, analizaremos a continuación un algoritmo que toma en cuenta las partes disjuntas, siendo más exacto que este método (Pullen, 2010).

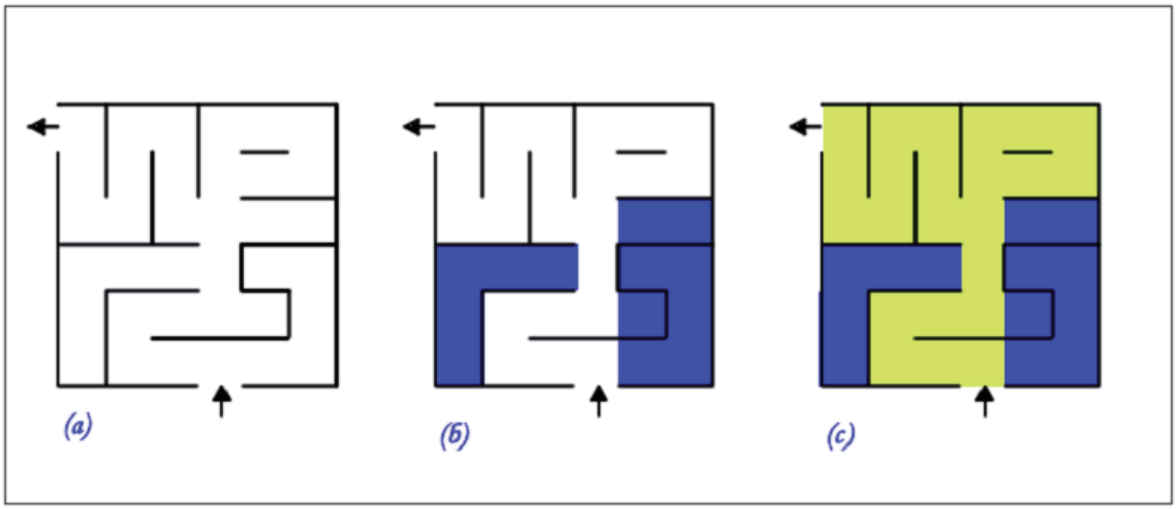


Figura 3.3 Algoritmo llenador de callejones sin salida.

3.1.2.2. Llenador de callejones ciegos.

Este algoritmo es una extensión del llenador de callejones sin salida, ya que toma en consideración los caminos creados por partes disjuntas. Funciona de igual forma que su pariente, transita por los caminos ciegos y los va llenando recursivamente hasta llegar a la meta.

El concepto de callejón ciego indica que para llegar a la meta se debe realizar backtrack, podríamos decir que todos los caminos sin salida son callejones ciegos. Además, esta definición incluiría a los caminos en círculos. Este método es sin duda más lento que el llenador de callejones sin salida, pero también es mucho más efectivo, ya que al eliminar los callejones sin salida y los caminos en círculos, deja exclusivamente el área que corresponde a la solución del laberinto (Pullen, 2010).

A pesar que este método deja el área de solución del laberinto, no toma en cuenta la solución óptima, en caso de que exista más de un camino a la salida. Un método que toma en cuenta este punto importante, es el algoritmo de inundación de laberinto.

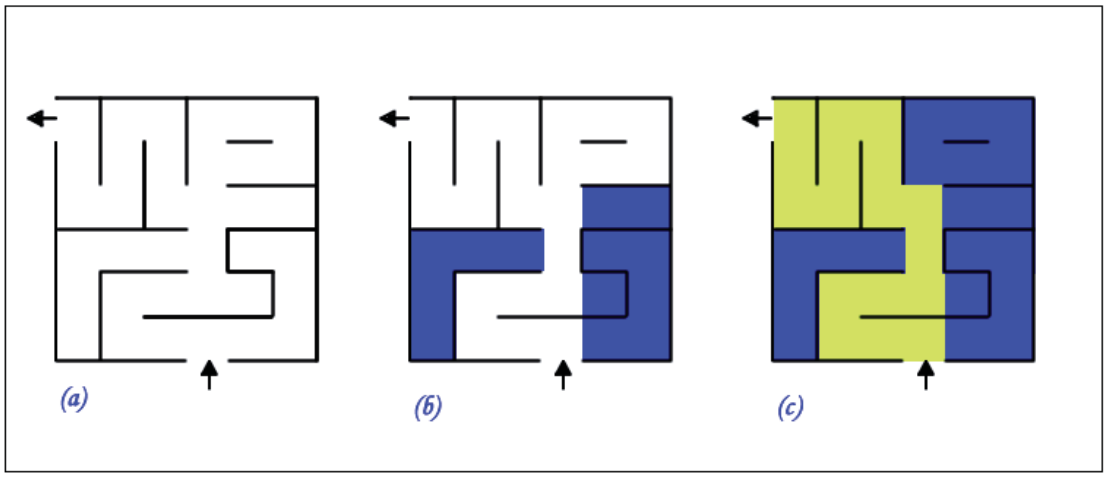


Figura 3.4 Algoritmo llenador de callejones ciegos.

3.1.2.3. Inundación de laberinto.

Este algoritmo comienza llenando todo el camino por donde avanza , cuando se encuentra con una bifurcación, el flujo de llenado no debe elegir una dirección, sino que se divide en dos o más flujos siguiendo cada uno por su propia ruta. Un punto importante es que a pesar que los flujos se dividan, siempre avanzan a la misma velocidad. Los flujos que no puedan continuar, significa que encontraron callejones ciegos. Cada una de las "secciones de agua" del laberinto tienen memoria sobre quién las llenó, lo que permite realizar el backtrack una vez encontrada la solución. Como todos los flujos avanzan a la misma velocidad, la solución óptima será el flujo que primero encuentre la salida. Si bien este método es más lento que los otros porque ocupa más procesamiento de la máquina, es el método más eficiente, ya que encuentra la solución óptima del problema (Pullen, 2010).

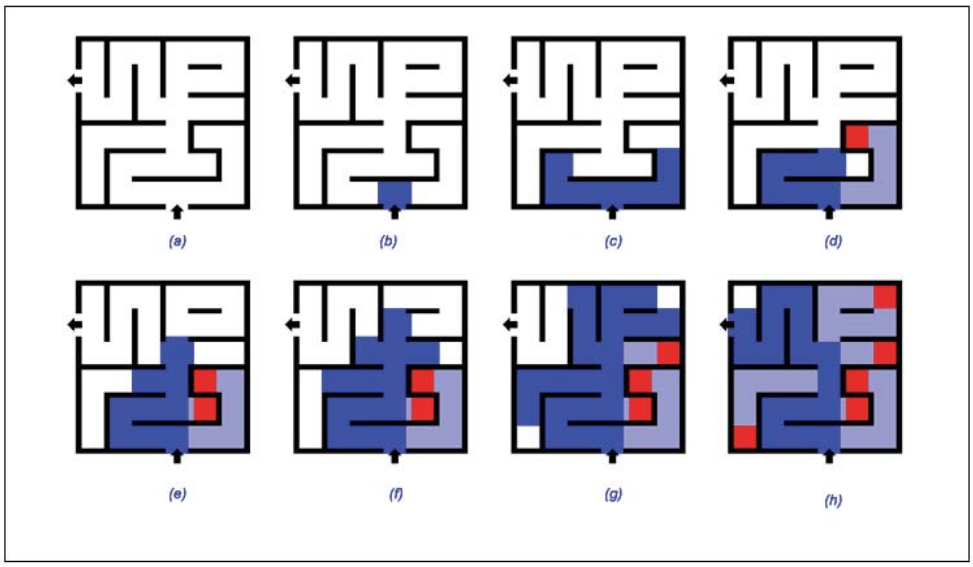


Figura 3.5 Algoritmo de inundación de laberinto.

La figura anterior muestra paso a paso el proceso de inundación del laberinto, dividiéndose los flujos de "agua" a medida que encuentra bifurcaciones, marcando los callejones sin salida.

Se puede concluir definitivamente que estos son los algoritmos de resolución de laberintos más conocidos y que garantizan casi siempre encontrar la solución. Para el caso específico de este trabajo de título, debemos contar sólo con algoritmos y configuraciones de laberintos que nos permitan llegar a una solución. Por lo anteriormente expuesto, no se forzará una configuración que no brinde algún tipo de solución, ya que no se busca la resolución del laberinto, sino que la comparación del análisis con las distintas secuencias de filtros que se le aplicarán.

3.2. Robótica.

3.2.1. Definición.

Según la real academia española, robótica es la ciencia que aplica informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales. Lo cierto es que la robótica combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control (Real Academia Española, 2010).

3.2.2. Antecedentes.

Antiguamente los robots eran conocidos con el nombre de autómatas y, a su vez, la robótica no era reconocida como ciencia, es más, la palabra robot surgió mucho tiempo después del origen de los autómatas a raíz de la popularidad que logró el término "robota" que significa servidumbre en la obra "Robots Universales Rossum", escrita por Karel Capek en 1920.

¿Cómo nace la idea de la robótica? Simple, desde el principio de los tiempos el hombre ha deseado crear vida artificial, es por eso que se ha empeñado en crear seres artificiales que le acompañen en el diario vivir, seres que realicen tareas repetitivas, pesadas o difíciles de ejecutar por un ser humano. En un inicio se creaban autómatas como un pasatiempo y con materiales que se encontraban en cualquier parte como lo son: maderas resistentes, metales y otros materiales moldeables que pudiesen servir en la creación de autómatas, a su vez, utilizaban la fuerza bruta para poder realizar sus movimientos.

El término robótica es acuñado por Isaac Asimov, quien la define como la ciencia que estudia los robots. A su vez, Asimov creó también las tres leyes de la robótica que son:

1.- Un robot no debe dañar a un ser humano, por su inacción, dejar que un ser humano sufra daño.

2.- Un robot debe obedecer las órdenes que son dadas por un ser humano, excepto si estas órdenes entran en conflicto con la primera Ley.

3.- Un robot debe proteger su propia existencia hasta donde esta protección no entre en conflicto con la primera o segunda Ley.

3.2.3. Clasificación.

Existen diversos criterios para clasificar a los robots como son: según su arquitectura, según su cronología, según nivel de inteligencia, según nivel de control, según nivel de lenguaje de programación. Las clasificaciones más interesantes son:

- Según su arquitectura. (Asociación de Robótica y Domótica España, 2010):

1.- Poliarticulados: En este grupo se encuentran los robots manipuladores como son los industriales. La característica principal es que son básicamente sedentarios y están estructurados para mover sus elementos terminales en un determinado espacio de trabajo.

Figura 3.6 Robot poliarticulado de palatización de cargas unitarias.

2.- Móviles: este grupo se caracteriza por poseer gran cantidad de desplazamiento, seguir su camino por telemando o guiándose por la información captada por sus sensores. Su tarea es asegurar el transporte de piezas de un punto a otro en la cadena de fabricación.

Figura 3.7 Robot para transportar material radioactivo.

3.- Androides: son robots que intentan reproducir el comportamiento del ser humano. Son un grupo actualmente muy poco evolucionado y en fase de experimentación.

Figura 3.8 Robot Androide Trompetista.

4.- Zoomórficos: este grupo se caracteriza por su sistema de locomoción que trata de imitar a diversos seres vivos. Sus aplicaciones a futuro son en el campo de la exploración espacial y el estudio de los volcanes.

Figura 3.9 Robot Zoomórfico: Perro.

5.- Híbridos: este grupo lo componen esos tipos de robots que en cierta medida combinan características de las categorías ya expuestas anteriormente. Por ejemplo, un dispositivo segmentado articulado y con ruedas, es al mismo tiempo uno de los atributos de los Robots móviles y de los Robots zoomórficos.

- Según el nivel de lenguaje de programación (Asociación de Robótica y Domótica España, 2010):

1.- Sistemas guiados: son los sistemas en los cuales el usuario conduce al robot a través de los movimientos que debe ir realizando.

2.- Sistemas de programación de nivel-robot: son los sistemas en los cuales el usuario escribe un programa especificando los movimientos y la forma en que deben reaccionar los sensores.

3.- Sistemas de programación nivel-tarea: son los sistemas donde el usuario especifica la acción por sus acciones sobre los objetos que el robot manipula.

3.2.4. Aplicaciones.

En cuanto a las aplicaciones de la robótica son muy variadas, pero generalmente se usan para hacer trabajos peligrosos o tareas difíciles, como por ejemplo: soldaduras al arco, exploraciones espaciales, recubrimiento con spray, inspección y mantenimiento de tuberías que llevan petróleo, gas o aceites en las plataformas oceánicas, siembra y poda de viñedos, etc. Todo esto, con el fin de lograr mejorar la productividad en algunos casos o simplemente proteger al ser humano de sustancias peligrosas.

3.2.5. Robótica en Chile.

La mayor parte de la robótica en nuestro país apunta a la automatización de procesos industriales. En el campo de la minería es donde más se ha desarrollado, precisamente, porque es un área donde es necesario reemplazar al hombre para ciertas tareas de exploración. Rambal ha creado robots pequeños para la inspección de ductos, cavernas y derrumbes de la mina El Teniente.

Uno de los robots más sofisticados que hay en Chile es el diseñado hace casi 15 años por Luis Cerda del Centro de Investigaciones Mineras y Metalúrgicas. Mide alrededor de 15 metros, con forma de araña, que se introduce en los molinos del Servicio Agrícola y Ganadero de Chuquicamata y cambia los revestimientos desde el interior.

Otra empresa que también ha diseñado robótica para la minería es Mechanical Studio, que trabajó en asociación con Codelco para desarrollar sistemas de automatización. Hoy están asociados con la compañía suiza Macrowiss, junto a quiénes trabajan en el diseño y construcción de robots para uso militar. Los chilenos diseñan acá y los suizos los construyen y comercializan en Europa. Una de sus creaciones es el "Tankbot", un pequeño tanque con tracción 6 x 6 que puede cargar distintos dispositivos como cámaras, armas automáticas o extintores.

Pero la robótica chilena no se limita sólo a la minería, ya que una de las creaciones más exitosas de Mechanical Studio es “Sentinel”, una lámpara de vigilancia que tiene una mini cámara en su interior. Este aparato, de uso en lugares públicos y empresas en general es muy útil como medida de seguridad, ya que permite que un intruso no sepa que está siendo observado.

La industria pesquera es otra que ha encontrado en los robots una solución a un grave problema que los afecta, que consiste en las fecas de los peces y los alimentos no consumidos que se depositan en el fondo del mar. Expertos de la Universidad de Concepción junto con el Instituto de Fomento Pesquero, diseñaron una máquina de manejo remoto que es capaz de bajar a 30 metros bajo el agua y limpiar estos desechos, tal como lo haría una aspiradora.

3.3. Robot Lego.

3.3.1. Antecedentes.

Los Robots Legos comenzaron como un juego de robótica creados inicialmente para niños, los cuales contienen elementos básicos de las teorías robóticas, como son el ensamblado de piezas y la programación y ejecución de acciones del robot. Sus primeros pasos fueron en el año 1998 y con el tiempo se comenzó a transformar en una herramienta educativa y de investigación, lo que posteriormente permitió el trabajo en conjunto de Lego y MIT, para confeccionar la herramienta principal con la que se trabajará, que son los Robots Lego Mindstorms. Esta unión demuestra la relación que existe entre la industria y la investigación académica, resultando muy beneficiosa para ambos campos.

Con el tiempo aparecieron sensores, aplicaciones y modelos caseros de este tipo. Los Robots Legos están compuestos de una parte física confeccionada con piezas de Lego y una parte lógica y electrónica denominada "bloques programables". Este bloque se encarga del robot y de su firmware básico para controlar los dispositivos que se conecten a éste.

3.3.2. Robot Lego Mindstorms NXT.

El Robot Lego Mindstorms en su versión NXT es el robot que se va a utilizar para el desarrollo del trabajo de título. Éste cuenta con un brick o bloque programable con un micro controlador ARM7 de 32 bits, además posee una velocidad de procesamiento de 48Mhz, memoria RAM de 64Kb, memoria flash de 256Kb y conexión por bluetooth (Lego Group, 1999). A diferencia de su bloque antecesor el RCX, el NXT posee mayores capacidades de ejecución de programas, evitando que los procesos inherentes de varios paquetes de datos colisionen y produzcan errores en la ejecución del software.

Figura 3.10 Bloques Programables. (a) Bloque programable NXT, (b) Bloque programable RCX.

Además posee cuatro tipos de sensores básicos:

- Sensor de tacto.
- Sensor de sonido.
- Sensor de patrones de luz.
- Sensor ultrasónico.

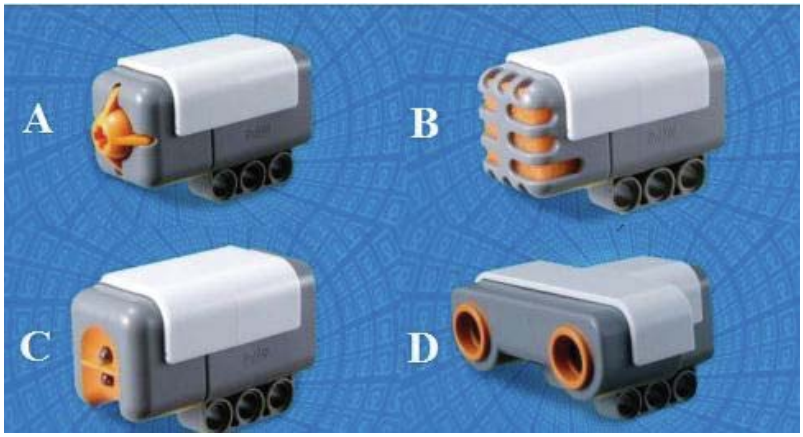


Figura 3.11 Sensores disponibles para la serie MindStorms NXT. (a) Sensor de tacto, (b) Sensor de sonido, (c) Sensor de patrones de luz, (d) Sensor ultrasónico.

A continuación, se explicarán brevemente en qué consiste cada uno de estos sensores y cómo actúan frente al medio (Lego Group, 2008).

- **Sensor de tacto:** Posee un switch en la parte frontal que es activado al tacto con tres estados: Presionado (sin soltar), suelto (por defecto) y tocado (presionado y soltado posteriormente).

- **Sensor de Sonido:** Este sensor permite detectar los decibeles (presión del sonido) emitidos por el ambiente. Puede detectar decibeles ajustados (dbA) que son los que detecta el oído humano, así como también puede detectar decibeles estándar dB, que son los sonidos que no son percibidos por el oído humano. Los resultados entregados por este sensor se miden en porcentaje de 0% a 100% en dónde la máxima es de 90 dB.

- **Sensor de patrones de luz:** Este sensor permite detectar en el ambiente los brillos y contraste, en otras palabras, diferenciar imágenes claras de oscuras. También permite medir las propiedades reflectantes de objetos que se encuentren al alcance del sensor. Este análisis lo hace mediante un lente que posee en la parte frontal, además de un led que ilumina el ambiente frente al sensor. Además de detectar lo visible por el ojo humano, también es capaz de detectar luces invisibles como son las luces ultravioletas.

- **Sensor ultrasónico:** Este sensor permite detectar los objetos físicos que se encuentran al alcance del sensor. Utilizando el mismo principio que los murciélagos, emite un sonido casi imperceptible al oído humano y calcula la distancia midiendo el tiempo que se demora este sonido en llegar, rebotar y volver. El sensor puede llegar a medir hasta 255 cm con una precisión de +-3cm.

Además de los sensores que posee el robot, también cuenta con un mecanismo que le permite realizar acciones, llamados servomotores, los cuales permiten al robot realizar movimientos en el ambiente. Con un tacómetro, los robots pueden medir en forma fácil y precisa, los movimientos que se realizan para poder agregarle exactitud a los mismos.

Figura 3.12 Servomotor robot Lego Mindstorms.

Otro de los componentes de Lego Mindstorms son las ruedas, que permiten que el bloque programable pueda moverse en un espacio real e interactúe con el medio que lo rodea. Existen diversos tipos de ruedas, algunas permiten mayor estabilidad y velocidad, otras permiten el movimiento del robot en zonas reducidas.

Figura 3.13 Tipos de ruedas robot Lego.

En cuanto a la programación posee un entorno de programación Java llamado LejOS NXJ. LejOS NXJ que se compone de un firmware que incluye una máquina virtual Java, además posee una biblioteca de clases de Java (class.jar) que implementan al LejOS NXJ.

Como ventajas tenemos que:

- Utiliza el lenguaje Java estándar.
- Es compatible con la programación orientada a objetos.
- Permite selección de entornos de desarrollo profesionales como Eclipse y Netbeans.
- Tiene soporte multiplataforma. (Windows, Linux y Mac OS X).

Figura 3.14 Ejemplo de robot Lego Mindstorms.

3.3.3. Configuración del robot.

La gran cantidad de variedades de piezas disponibles de Legos permiten contar con innumerables configuraciones de Robot Lego, adecuándose a las necesidades de la investigación.

Los avances de los Robot Legos han sido de gran magnitud, llegando en la actualidad a participar en la confecciones de robots humanoides. También se desarrolló una nueva versión de Robots Mindstorms (la versión 2.0), que incluye entre otros sensores de color.

3.4. Procesamiento digital de imágenes.

3.4.1. Antecedentes.

El procesamiento digital de imágenes aparece tardíamente en la historia de la computación, alrededor de los años 80. Antes de pensar en dicho procesamiento había que desarrollar hardware y los sistemas operativos gráficos que permitieran hacerlo, lo cual era muy costoso. Es por ello que las investigaciones se restringían al campo de las ciencias de la computación, matemáticas y astronomía. Actualmente, debido a lo barato que resulta el hardware y las diversas aplicaciones de software desarrolladas, el campo de investigación es mucho más amplio llegando al control industrial, medicina, telecomunicaciones, robótica, etc. (Paz, 2007).

El procesamiento digital de imágenes ha evolucionado rápidamente debido a varios factores:

- Los sistemas digitales de imágenes son capaces de adquirir imágenes con un rango dinámico más amplio que el ojo humano o películas fotográficas. El ojo puede distinguir poco menos de 100 tonalidades de gris, dependiendo del contraste, mientras que los sistemas digitales pueden utilizarse para representar varios cientos e incluso miles de tonos de gris.

- Una imagen digital puede presentar una gran cantidad de información en forma compacta y sencilla. Una imagen digital contiene varios millones de bits de información que pueden ser mostrados en una sola impresión fotográfica o en un monitor.

- Los computadores pueden procesar y manipular imágenes utilizando métodos que resultarían inalcanzables de otra forma.

- El avance de la tecnología de computadores ha permitido desarrollar tecnología de procesamiento de imágenes. La tendencia a la rapidez y bajo costo de los elementos computacionales, así como la gran capacidad de dispositivos de almacenamiento, han hecho posible el procesamiento de imágenes.

Cuando se habla de procesamiento de imágenes se refiere al conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información en ellas. Estas técnicas se aplican en distintas etapas del procesamiento de imágenes.

3.4.2. Etapas del Procesamiento digital de imágenes.

En el procesamiento digital de imágenes se pueden distinguir cinco etapas que son:

- Adquisición de la imagen.
- Preprocesamiento.
- Segmentación.
- Representación y descripción.
- Reconocimiento e interpretación.

Estas etapas se pueden agrupar en 3 niveles de procesamiento: procesamiento de bajo nivel (adquisición de la imagen, preprocesamiento), procesamiento de nivel intermedio (al procesamiento de bajo nivel se le agrega la etapa de segmentación) y procesamiento de alto nivel (al procesamiento de nivel intermedio se le agregan las etapas restantes). A continuación, se muestra una imagen donde se aprecia la interacción entre las distintas etapas del procesamiento digital de imágenes (González, 2008).

Figura 3.15 Etapas del procesamiento digital de imágenes.

3.4.2.1. Adquisición de la imagen.

El primer paso para el procesamiento de imágenes digitales es la adquisición de la imagen a procesar; esto quiere decir, digitalizarla. Ahora para entender lo que es la digitalización se debe mirar desde un punto de vista físico, donde una imagen puede considerarse como un objeto plano cuya intensidad luminosa y color pueden variar de un punto a otro. Si se trata de imágenes monocromas (blanco y negro), se pueden representar en una función continua f (x,y), en donde (x,y) son sus coordenadas y el valor de f es proporcional a la intensidad luminosa (nivel de gris) en ese punto (González, 2008).

Para obtener una imagen que pueda ser tratada por el computador es preciso someter a la función f (x,y) a un proceso de discretización, tanto en las coordenadas como en la intensidad. A este proceso se le denomina digitalización.

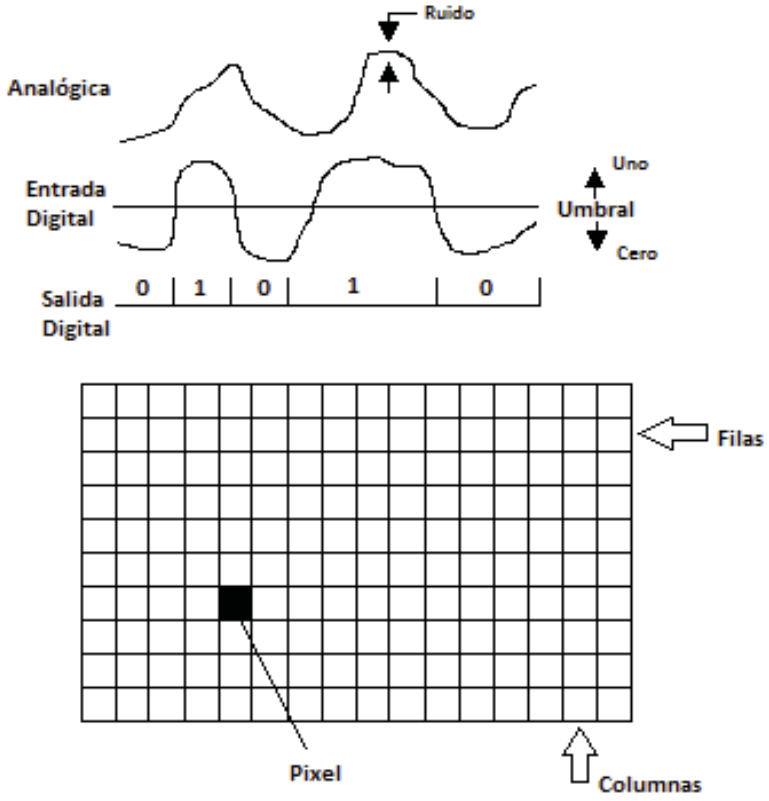


Figura 3.16 Discretización de una imagen.

La salida del proceso puede ser un único valor (escala de grises) o bien un vector con tres valores RGB, que se corresponden con la intensidad de color rojo (R), verde (G) y azul (B). Las imágenes en escala de grises que solo tienen blanco y negro se llaman imágenes binarias.

Una imagen puede considerarse como una matriz, cuyos índices de fila y columna identifican un punto de la imagen. En tanto, el valor del correspondiente elemento de la matriz indica el nivel de gris de ese punto. Los elementos de una distribución digital de este tipo se denominan píxel o pels, que son sus abreviaturas de la denominación inglesa "Picture elements". Una imagen en escala de grises es representada por una matriz bidimensional de m x n elementos en donde n representa el número de pixeles de ancho y el m el número de pixeles largo. Por otro lado, una imagen de color RGB es representada por una matriz tridimensional m x n x p, donde m y n tienen el mismo significado que para el caso de las imágenes de escala de grises, mientras que p representa el plano que para RGB puede ser 1 para el rojo (R), 2 para el verde (G) y 3 para el azul (B).

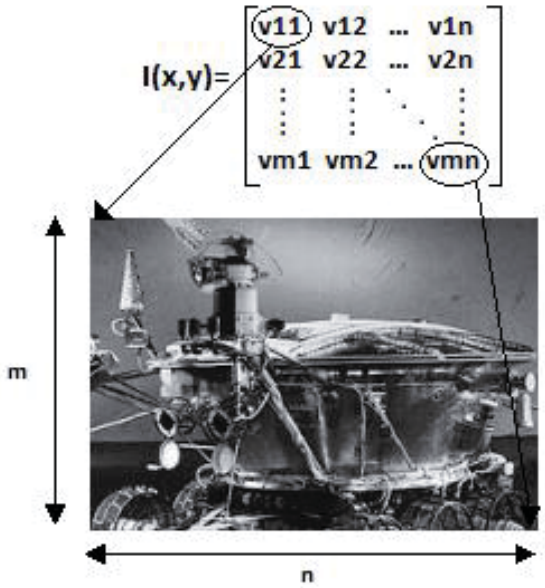


Figura 3.17 Representación de una imagen en escala de grises.

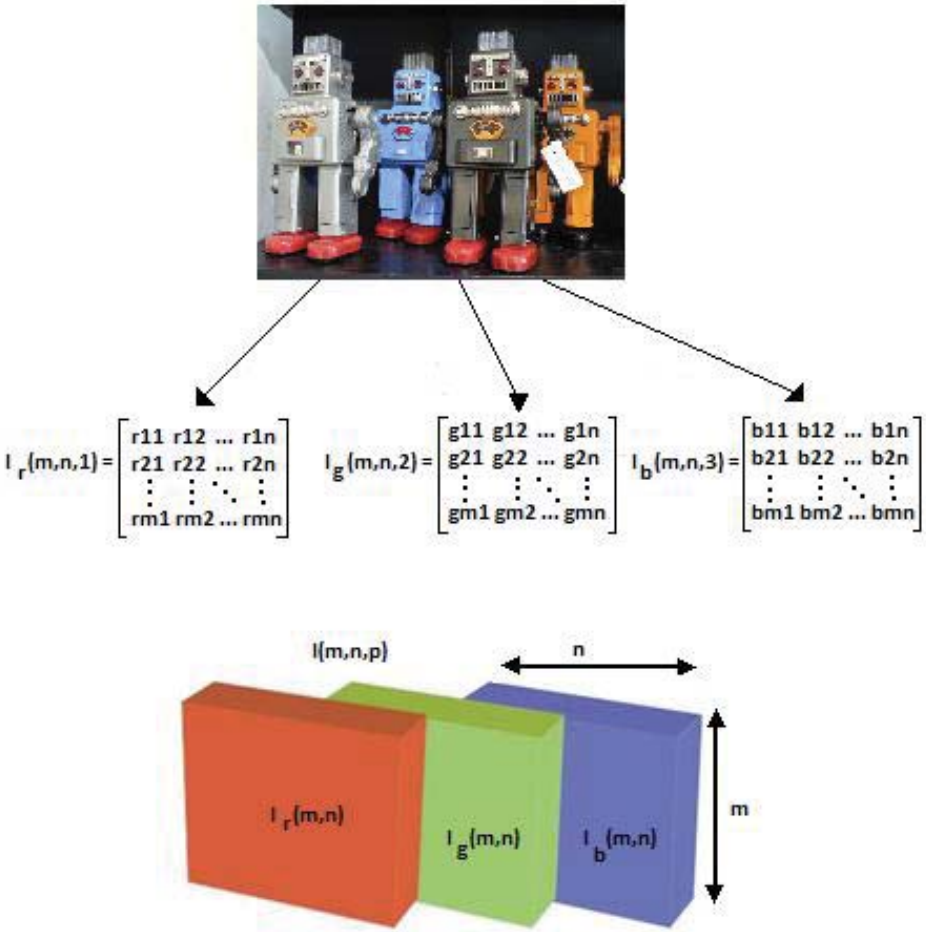
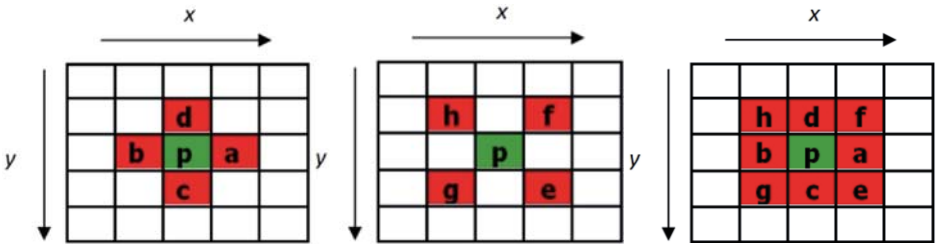


Figura 3.18 Representación de una imagen a color RGB.

3.4.2.2. Preprocesamiento.

El segundo paso para el procesamiento de imágenes digitales es el preprocesamiento que consiste en mejorar la imagen, de manera tal, de hacer evidentes ciertos detalles que se deseen destacar de modo de incrementar la posibilidad de éxito de los siguientes procesos (González, 2008).

Antes de conocer la categorización de filtros en esta etapa es necesario comprender los conceptos de convolución y vecindad. La convolución es una matriz con ciertos valores que se aplica a una imagen, con el fin de obtener una nueva imagen filtrada. Lo que hace básicamente la convolución es modificar el color de un píxel en función del color de los pixeles vecinos. Con respecto a sus vecinos tenemos 3 tipos de vecindades (González, 2008) que se pueden apreciar en la imagen que sigue:



a) b) c)

Figura 3.19 Vecindades de un píxel. (a) Vecindad horizontal-vertical, (b) Vecindad diagonal, (c) Vecindad ocho, que es la unión de las dos vecindades anteriores

El modo de aplicar la convolución es el siguiente: se multiplica el color de un píxel y el de sus pixeles vecinos por una matriz. Esta matriz se llama máscara de convolución que se mueve por cada uno de los pixeles de la imagen original, en donde cada píxel que queda bajo la matriz se multiplica por un valor de la matriz y el resultado se divide después por un valor específico. El resultado es el nuevo color del píxel que cae en el centro de la matriz. Ahora se verá un ejemplo para mostrar el procedimiento usando la siguiente máscara de convolución.

1	1	1
1	4	1
1	1	1

Figura 3.20 Ejemplo de máscara de convolución.

Las máscaras pueden tener tamaños arbitrarios pero, las más usadas son las de 3x3, ya que son rápidas y toman en consideración el valor píxel del mismo y sus 8 vecinos (vecindad ocho). Ahora para aplicar lo mencionado anteriormente, se debe colocar la máscara sobre la imagen y multiplicar los valores de color por 1 o 4. El resultado, se suma y divide por 12 que es en este caso, la suma de los elementos de la máscara de convolución. A continuación, se presenta la imagen en escala de grises sobre la cual se aplicará la máscara de convolución (Paz, 2007).

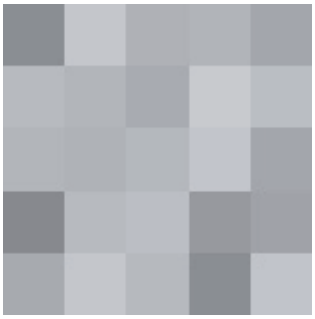


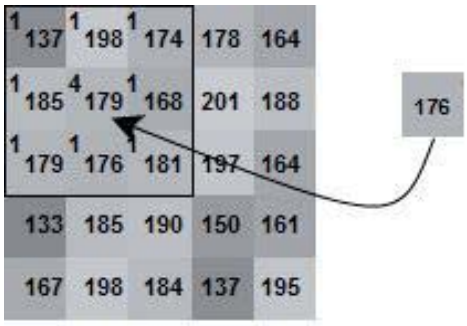
Figura 3.21 Imagen en escala de grises.

La imagen anterior tiene para cada píxel un cierto valor, según su tonalidad de gris que es necesario saber para poder explicar el proceso de convolución.

137	198	174	178	164
185	179	168	201	188
179	176	181	197	164
133	185	190	150	161
167	198	184	137	195

Figura 3.22 Imagen con valores de sus tonalidades.

Ahora si se coloca la máscara sobre los pixeles de la imagen y el primer píxel central que obtenemos es el (2,2), luego al desplazarnos con la máscara a la derecha el siguiente píxel es el (2,3) y así sucesivamente. En la imagen que sigue se puede apreciar el nuevo valor que toma el píxel (2,2) y su respectivo cálculo.



$$y(2,2) = (1x137+1x198+1x174+1x185+4x179+ 1x168+1x179+1x176+1x181) / 12$$

$$y(2,2) = 2114 / 12$$

$$y(2,2) = 176$$

Figura 3.23 Aplicación de máscara de convolución.

Una vez explicados los conceptos de convolución y vecindad, se expondrán los métodos para mejorar la imagen que se pueden dividir en dos categorías: los métodos de mejora en el dominio espacial y los métodos de mejora en el dominio de la frecuencia. Los métodos de la primera categoría consisten en la manipulación directa de los pixeles de la imagen, mientras que los de la segunda categoría corresponden a técnicas basadas en la representación de los pixeles a través de una transformación hacia el dominio de la frecuencia, y usa como operador de mapeo o transformación a la Transformada de Fourier (TdF). A continuación, se detallan usos de los filtros anteriormente nombrados.

a.- Filtrado Espacial.

Los filtros espaciales tienen como objetivo modificar la contribución de determinados rangos de frecuencias de una imagen. El término espacial se refiere al hecho que el filtro se aplica directamente a la imagen y no a una transformada de la misma, es decir, el nivel de gris de un píxel se obtiene directamente en función del valor de sus vecinos (González, 2008).

Los filtros espaciales pueden clasificarse basándose en su linealidad: filtros lineales y filtros no lineales. A su vez, los filtros lineales pueden clasificarse según las frecuencias que dejen pasar:

- Filtros Paso-Bajas: Son utilizados en la reducción de ruido; suavizan y aplanan un poco las imágenes y como consecuencia se reduce o se pierde la nitidez. En inglés son conocidos como Smoothing Spatial Filters.

- Filtros Paso-Altas: Estos filtros son utilizados para detectar cambios de luminosidad. Son utilizados en la detección de patrones como bordes o para resaltar detalles finos de una imagen. En inglés son conocidos como Sharpening Spatial Filters.

- Filtros Paso-Banda: Son utilizados para detectar patrones de ruido. Estos filtros casi no son usados ya que generalmente eliminan demasiado contenido de la imagen. Sin embargo, los filtros paso-banda son útiles para aislar los efectos de ciertas bandas de frecuencias seleccionadas sobre una imagen. De esta manera, estos filtros ayudan a simplificar el análisis de ruido razonablemente, independiente del contenido de la imagen.

Algunas técnicas que se logran con los filtros espaciales son el suavizado, continuidad de bordes y mejoramiento de la nitidez.

- **Suavizado.**

Los filtros paso-bajas son utilizados para difuminar y reducir ruido en las imágenes; a este proceso se le conoce en inglés como smoothing. El difuminado (blurring) se emplea en etapas de preprocesamiento, desde la eliminación de pequeños detalles hasta la extracción de objetos y rellenado de pequeños huecos en líneas y curvas. La reducción de ruido puede ser completada por el difuminado usando filtros lineales o bien con un filtrado no lineal (González, 2008).

(a) (b) (c)

Figura 3.24 Ejemplo de Suavizado. (a) imagen original sin ruido, (b) imagen con ruido, (c) imagen suavizada.

La salida de un filtro paso-bajas lineal simplemente es un tipo de promedio de los pixeles contenidos en la vecindad de la máscara del filtro. Estos filtros son frecuentemente llamados filtros promediadores. La idea detrás de estos filtros es de reemplazar cada píxel en la imagen por un promedio de los niveles de gris de los vecinos definidos por la máscara del filtro.

- **Filtros basados en derivadas de la función Gaussiana.**

Los filtros basados en derivadas Gaussianas fueron en un principio obtenidos de manera heurística. Sin embargo estos filtros tienen un fundamento matemático y un comportamiento muy similar al sistema de visión humano (HVS); por ello son muy importantes ya que se especializan en la detección de cambios bruscos como los bordes.
La función Gaussiana (González, 2008) tiene la siguiente expresión:

$$f(x) = Ne^{-ax^2}$$

Donde N es una constante de normalización que depende de a pero no de x. Hay varias posibilidades para el exponente a, en la forma estándar tenemos:

$$a = \frac{1}{2\sigma^2}$$

Luego la primera derivada de Gaussiana tiene la siguiente forma:

$$f(x) = -N2a(a)e^{-ax^2}$$

Mientras que la segunda derivada tiene la siguiente forma:

$$f(x) = N2a(2ax^2 - 1)e^{-ax^2}$$

Las siguientes figuras muestran las cuatro primeras derivadas de la función Gaussiana con N = σ =1:

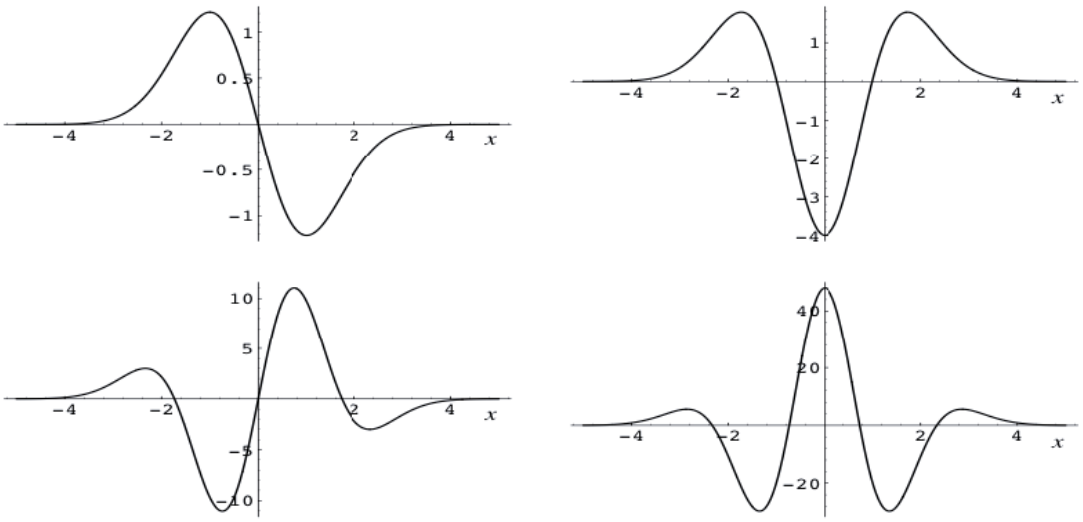


Figura 3.25 Primeras cuatro derivadas de Gaussianas.

Las derivadas de una función digital o discreta son definidas en términos de diferencias. La forma discreta de la primera derivada de Gaussiana se puede expresar de la siguiente manera: f (x) − f (x−1)

Ahora bien, si se desease diseñar un filtro espacial tal que (González, 2008):

$$g(x) = h(x) * f(x) \text{ y } g(x) = f(x) - f(x-1)$$
$$f(x) = f(x) * \delta(x)$$

Y

$$f(x-1) = f(x-1) * \delta(x-1)$$

Por lo tanto, el filtro deseado h(x) tiene la siguiente expresión: $h(x) = \delta(x) - \delta(x-1)$ y tiene la siguiente representación en el plano:

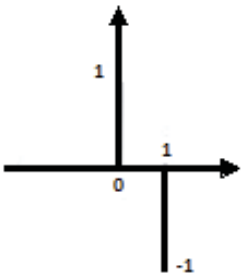


Figura 3.26 Representación h(x) = δ(x)−δ(x−1) en el plano

Ahora $h(x) = \delta(x) - \delta(x-1)$ puede escribirse como: $h(x) = [1\ -1]$. Si se agrega un cero a al filtro anterior, es decir: $h(x) = [1\ 0\ -1]$ podemos reescribir a h(x) como:

$$h(x) = \delta(x+1) - \delta(x-1)$$

Y su representación en el plano sería:

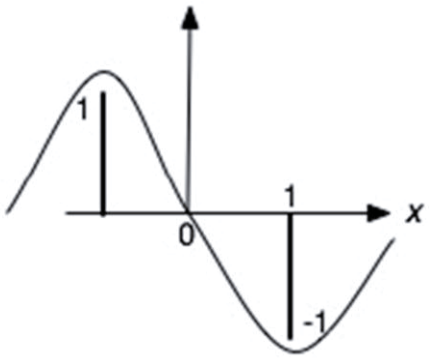


Figura 3.27 Filtro $h(x) = \delta(x+1) - \delta(x-1)$.

El filtro $h(x) = \delta(x+1) - \delta(x-1)$ posee ya un comportamiento discreto de la primera derivada de Gaussiana en el plano continuo y estaríamos en la posibilidad de detectar cambios en la dirección x.

Estos filtros detectores de discontinuidades o cambios, basados en las derivadas de Gaussianas, tienen regularmente aplicación en tareas previas a la segmentación como detectores de bordes.

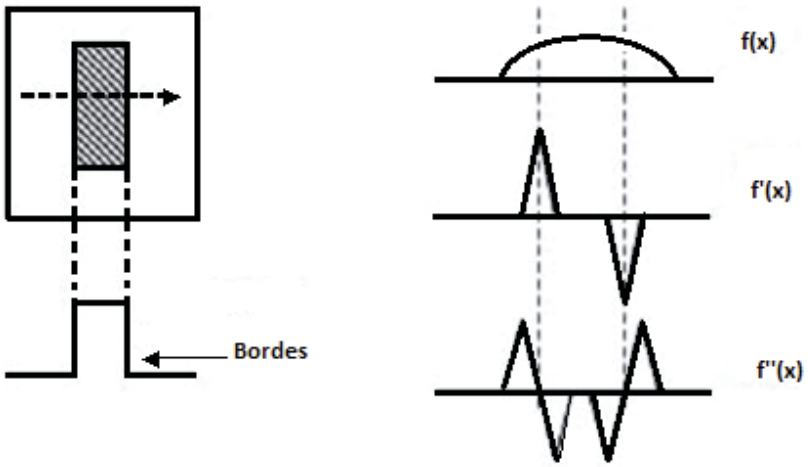


Figura 3.28 Detectores de bordes.

- **Mejoramiento de la nitidez.**

El mejoramiento de la nitidez o de la calidad visual de una imagen basada en los filtros unsharp masking tiene bastante importancia en el procesamiento digital de imágenes. Si se quisiera traducir al español unsharp masking podría interpretarse como enmascaramiento de imagen borrosa. Se basa en el hecho que se tiene una imagen borrosa (con pendiente pequeña) y se le resta una pendiente aún más pequeña (fPB). A su vez, esto va multiplicado por un factor k al que se recomienda tome valores entre 1 y 3 (González, 2008).

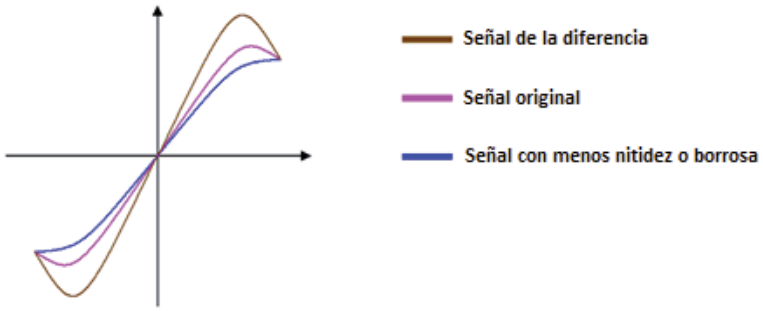


Figura 3.29 Unsharp masking

Después se suman la señal de la diferencia con la original para obtener una pendiente mayor y por lo tanto, más resolución. Lo anterior se puede expresar como:

$$k\,[\,f(x,y) - fPB(x,y)] + f(x,y) = f(x,y) * hUM$$

Donde hUM(x,y) está definido como: $hUM(x,y) = \delta(x,y) + k\delta(x,y) - khPB(x,y)$

Con lo que tendríamos la forma final para los filtros unsharp masking en la siguiente expresión: $hUM(x,y) = (1+k)\delta(x,y) - khPB(x,y)$

De la ecuación anterior se puede notar que, si modificamos el filtro paso bajas fPB entonces el filtro unsharp masking también cambiará. Esto implica que podemos tener varios filtros unsharp masking.

b.- Filtrado por Frecuencia

El teorema de la convolución es hacer la correspondencia entre el filtrado espacial y el filtrado en el dominio de la frecuencia. El proceso en el cual se mueve una máscara de un píxel a otro píxel sobre una imagen y calculamos una cantidad en cada píxel tiene su fundamento en dicho teorema. Formalmente, la convolución discreta de dos funciones f (x,y) y h(x,y) de tamaño M × N es denotada por f (x,y) ∗ h(x,y) y es definida por la expresión (González, 2008):

$$f(x,y) * h(x,y) = \frac{1}{MN} \sum_{m=0}^{m-1} \sum_{n=0}^{n-1} f(m,n) h(x-n, y-n)$$

Sea F(u,v) y H(u,v) las transformadas de Fourier de f (x,y) y h(x,y) respectivamente. El teorema de la convolución indica que f (x,y) ∗ h(x,y) y F(u,v) H(u,v) están relacionadas de la siguiente manera:

$$f(x,y) * h(x,y) \Leftrightarrow F(u,v)H(u,v)$$

Y de manera análoga que:

$$f(x,y)h(x,y) \Leftrightarrow F(u,v) * H(u,v))$$

El filtrado en frecuencia sirve principalmente para observar las características de los filtros al obtener su espectro y de esa manera, saber si se trata de filtros paso-bajas, paso-altas, etc. Otro factor por el cual es preferible hacer filtrado en frecuencia es la aceleración del algoritmo para calcular la Transformada Discreta de Fourier, usando para esto las implementaciones de la Transformada Rápida de Fourier.

3.4.2.3. Segmentación

El tercer paso para el procesamiento de imágenes digitales es la segmentación, que siendo definida de manera amplia, es saber que es cada objeto en la imagen y para eso es necesario saber que grupo de pixeles corresponden a un mismo grupo. Es una de las etapas más complejas en el procesamiento digital de imágenes. Las técnicas de segmentación más conocidas son las basadas en detección de bordes y segmentación de regiones (González, 2008).

a.- Detección de bordes.

Con respecto a la detección de bordes, podemos decir que los bordes de una imagen digital se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Suministran una valiosa información sobre las fronteras de los objetos y puede ser utilizada para segmentar la imagen, reconocer objetos, etc.

La mayoría de las técnicas para detección de bordes se agrupan en torno a las derivadas, en donde, la primera derivada indica la presencia de un borde, mientras la segunda derivada indica si un píxel pertenece a la parte “clara” u “oscura” del borde.

a.1.- Máscara del cálculo de la gradiente.

En el caso de la máscara del cálculo de la gradiente, asociada a la primera derivada, los operadores más utilizados son los de Roberts, Prewitt, Sobel, Frei-chen y Canny.

- **Operador de Roberts.**

Las máscaras usadas por este operador son (Paz, 2007):

Gradiente fila

0	0	0
0	0	1
0	-1	0

Gradiente columna

-1	0	0
0	1	0
0	0	0

Figura 3.30 Máscara convolución operador Roberts

Obtiene buena respuesta ante bordes diagonales. Ofrece buenas prestaciones en cuanto a localización. El gran inconveniente de este operador es su extremada sensibilidad al ruido y, por tanto, tiene pobres cualidades de detección.

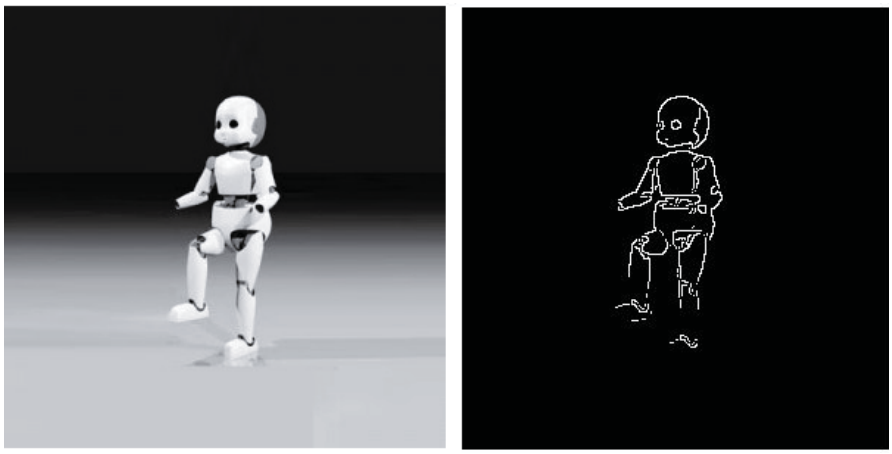


Figura 3.31 Detección de bordes con operador de Roberts

- **Operadores de Prewitt, Sobel y Frei-Chen.**

Los tres operadores pueden formularse de forma conjunta con las siguientes máscaras de convolución mostradas a continuación (Paz, 2007):

Gradiente fila

$$\frac{1}{2+K}$$

1	0	-1
K	0	-K
1	0	-1

Gradiente columna

$$\frac{1}{2+K}$$

-1	-K	-1
0	1	0
1	K	1

Figura 3.32 Mascaras de convolución operadores Prewitt, Sobel y Frei-Chen.

En el operador Prewitt (K=1) se involucran a los vecinos de filas / columnas adyacentes para proporcionar mayor inmunidad al ruido.

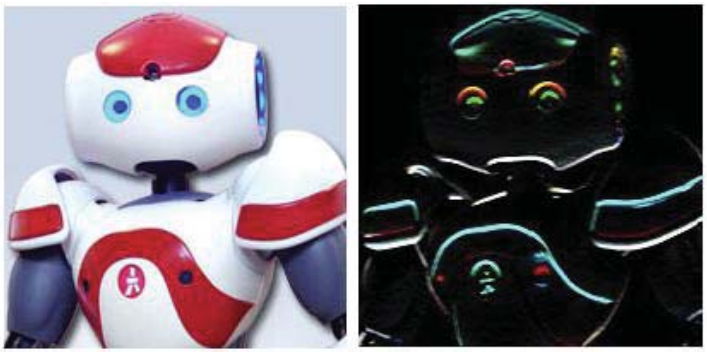


Figura 3.33 Detección de bordes con operador Prewitt.

El operador Sobel (K=2) se supone que es más sensible a los bordes diagonales que el de Prewitt, aunque en la práctica hay poca diferencia entre ellos.

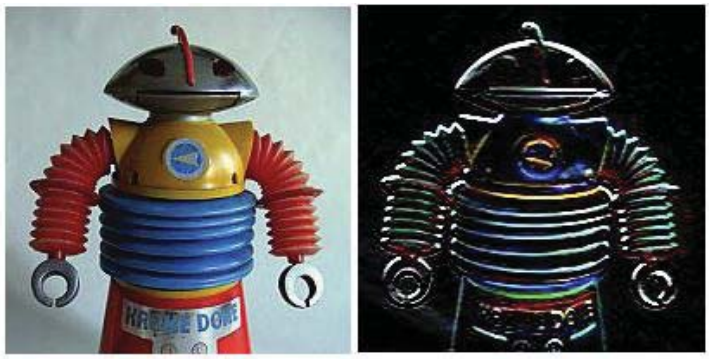


Figura 3.34 Detección de bordes con operador Sobel.

Frei-Chen (K= √2), el gradiente es el mismo para bordes verticales, horizontales y diagonales.

- **Operador de Canny.**

Es uno de los operadores más potentes y por lo mismo uno de los más usados. Este operador, se caracteriza por estar optimizado para la detección de bordes diferenciales, consta de 4 fases: obtención del gradiente, supresión no máxima al resultado de la gradiente, histéresis de umbral a la supresión no máxima y cierre de contornos.

Obtención del gradiente: en este paso se calcula la magnitud y orientación del vector gradiente en cada píxel.

Para la obtención del gradiente, lo primero que se realiza es la aplicación de un filtro Gaussiano a la imagen original con el objetivo de suavizar la imagen y tratar de eliminar el posible ruido existente. Sin embargo, se debe de tener cuidado de no realizar un suavizado excesivo, pues se podrían perder detalles de la imagen y provocar un pésimo resultado final. Este suavizado se obtiene promediando los valores de intensidad de los pixeles en el entorno de vecindad con una máscara de convolución de media cero y desviación estándar σ. En la figura siguiente, se muestran dos ejemplos de máscaras que se pueden usar para realizar el filtrado Gaussiano.

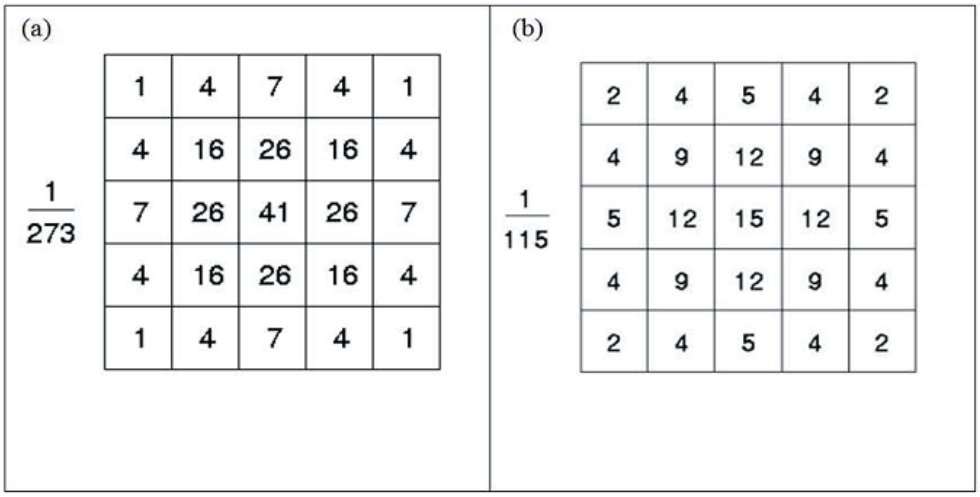


Figura 3.35 Máscaras de convolución recomendadas para obtener el filtro Gaussiano.

Una vez que se suaviza la imagen, para cada píxel se obtiene la magnitud y módulo (orientación) del gradiente, consiguiendo así dos imágenes.

Supresión no máxima al resultado del gradiente: en este paso se logra el adelgazamiento del ancho de los bordes obtenidos con el gradiente, hasta alcanzar bordes de un píxel de ancho.

Las dos imágenes generadas en el paso anterior sirven de entrada para obtener una imagen con los bordes adelgazados. El procedimiento es el siguiente: se consideran cuatro direcciones identificadas por las orientaciones de 0º, 45º, 90º y 135º con respecto al eje horizontal. Para cada píxel se encuentra la dirección que mejor se aproxime a la dirección del ángulo de gradiente.

Posteriormente, se observa si el valor de la magnitud de gradiente es más pequeño que al menos uno de sus dos vecinos en la dirección del ángulo obtenida en el paso anterior. De ser así, se asigna el valor 0 a dicho píxel; en caso contrario, se asigna el valor que tenga la magnitud del gradiente (Paz, 2007).

Histéresis de umbral a la supresión no máxima: en este paso se aplica una función de histéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

La imagen obtenida en el paso anterior suele contener máximos locales creados por el ruido. Una solución para eliminar dicho ruido es la histéresis del umbral.

El proceso consiste en tomar la imagen obtenida del paso anterior, tomar la orientación de los puntos de borde de la imagen y tomar dos umbrales, el primero más pequeño que el segundo. Para cada punto de la imagen se debe localizar el siguiente punto de borde no explorado que sea mayor al segundo umbral. A partir de dicho punto seguir las cadenas de

máximos locales conectados en ambas direcciones perpendiculares a la normal del borde siempre que sean mayores al primer umbral. Así se marcan todos los puntos explorados y se almacena la lista de todos los puntos en el contorno conectado. Es así como en este paso se logra eliminar las uniones en forma de Y de los segmentos que confluyan en un punto.

***Cierre de contornos*:** en este paso final lo que se realiza es cerrar los contornos que pudiesen haber quedado abiertos por problemas de ruido.

Un método muy utilizado es el algoritmo de Deriche y Cocquerez. Este algoritmo utiliza como entrada una imagen binarizada de contornos de un píxel de ancho. El algoritmo busca los extremos de los contornos abiertos y sigue la dirección del máximo gradiente hasta cerrarlos con otro extremo abierto .

El procedimiento consiste en buscar para cada píxel uno de los ocho patrones posibles que delimitan la continuación del contorno en tres direcciones posibles. Esto se logra con la convolución de cada píxel con una máscara específica. Cuando alguno de los tres puntos es ya un píxel de borde se entiende que el borde se ha cerrado, de lo contrario, se elige el píxel con el valor máximo de gradiente y se marca como nuevo píxel de borde y se aplica nuevamente la convolución. Estos pasos se repiten para todo extremo abierto hasta encontrar su cierre o hasta llegar a cierto número de iteraciones determinados.

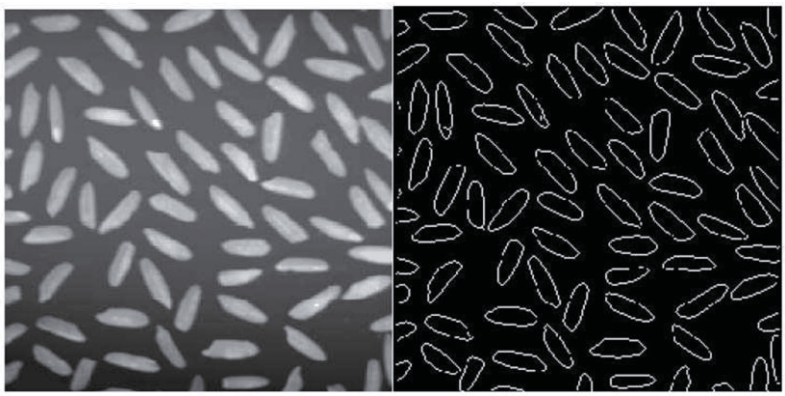


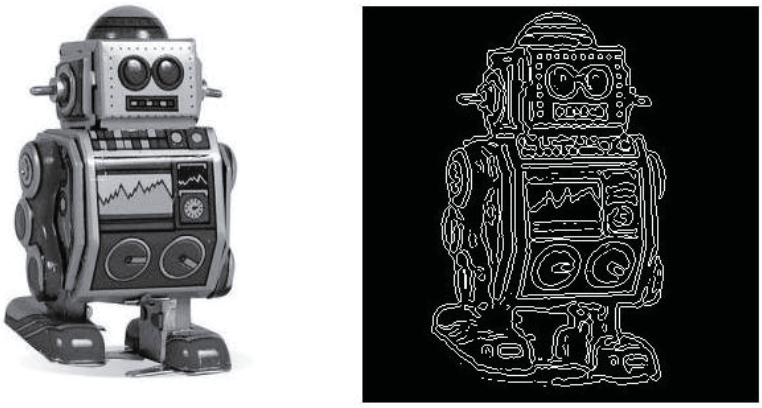
Figura 3.36 Resultado de la detección de bordes mediante el operador de Canny.

a.2.- Máscara Laplaciana

Está asociada a la segunda derivada, podemos distinguir 2 aplicaciones (Paz, 2007):

- **Detección de cruces por cero en el filtro Laplaciano**

Este filtro se utiliza para mejorar la nitidez de la imagen por su capacidad de captar altas discontinuidades como resultado de aplicar la segunda derivada. Adicionalmente, detectar un borde es detectar el cruce por cero de la segunda derivada de la imagen.



a) b)

Figura 3.37 Aplicación de detección de cruces por cero en el filtro Laplaciano. a) Imagen original, b) Imagen después de aplicar detección de cruces por cero en el filtro Laplaciano.

- **Laplaciana de la Gaussiana**

Como su nombre lo indica es el resultado de aplicar la segunda derivada a una función Gaussiana. Existen diferentes formas para emular la segunda derivada representada a través de este método, para ello debe existir un centro positivo y un entorno de valores negativos que posteriormente se aproximan a cero en la medida que se aleja del centro de la máscara. Debe cumplirse la condición de que la suma de los coeficientes sean cero para cuando el nivel de gris sea semejante, el filtro devuelva cero (fondo de la imagen).

Aunque el Laplaciano responde a transiciones en la intensidad de la imagen, se emplea en pocas ocasiones en la práctica para la detección de bordes, ello es debido a que un operador de segunda derivada es muy sensible a la presencia de ruido y puede producir bordes dobles.

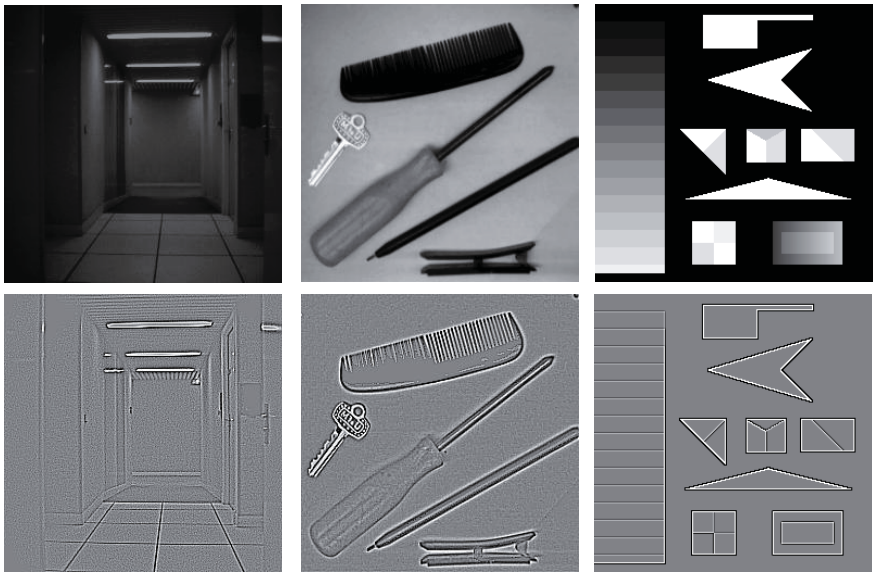


Figura 3.38 Aplicación filtro Laplaciano de la Gaussiana. Arriba imagen original en escala de grises, abajo, la imagen con el filtro aplicado.

b.- Segmentación de regiones

b.1.- Métodos de crecimiento de regiones.

Es un procedimiento iterativo que consiste en unir regiones adyacentes que cumplan con criterios como son el nivel de intensidad, varianza de los niveles de la región y textura de la región. Cuando finaliza el algoritmo dejan de haber regiones adyacentes que cumplan los criterios anteriormente nombrados (Paz, 2007).

b.2.- Procedimientos de separación-unión.

El procedimiento se basa en el "Quad-tree" de una imagen, esto quiere decir la organización jerárquica de una imagen. Es así que:

- Cada nodo contiene la intensidad media de un conjunto de pixeles.
- El nodo raíz contiene la intensidad media de la imagen.
- Las hojas del árbol contienen la intensidad de cada píxel.
- El nodo padre de cuatro hojas en un nodo 2x2 y contiene su intensidad media.
- Si las hojas tienen igual intensidad, no se ponen explícitamente en el quad-tree.

Una vez que se tiene el quad-tree se le aplica un algoritmo de profundidad superior a pixeles en dos pasos: unión y separación.

- **Proceso de unión:** Si las cuatro hojas de cada nodo cumplen criterios de uniformidad, se unen borrándose del quad-tree. El padre hereda las propiedades. En caso de no cumplirse los criterios de uniformidad no se unen.

- **Proceso de separación:** Si las cuatro hojas de cada nodo no cumplen criterios de uniformidad, se dividen y vuelven a aparecer en el "quad-tree". Si se cumplen los criterios de uniformidad, siguen como estaban.

La gran desventaja de esta técnica de segmentación de regiones es su alto coste computacional, por lo cual queda fuera del desarrollo de este trabajo de título.

3.4.2.4. Representación y descripción - Reconocimiento e interpretación.

El cuarto paso para el procesamiento de imágenes digitales es la representación y descripción, también llamada selección de características, donde se trata con extracción de rasgos que resulta en alguna información cuantitativa de interés o características que son básicas para diferenciar una clase de objetos con otra. Por otro lado el reconocimiento es el proceso que etiqueta o asigna un nombre a un objeto, basándose en la información que proveen sus descriptores y la interpretación, involucra la asignación de significado a un conjunto de objetos reconocidos (González, 2008).

4. Desarrollo.

4.1. Construcción del laberinto.

La construcción del laberinto se lleva a cabo fundamentalmente con tres elementos: conectores, bases y paredes. Estos elementos son de ensamble rápido y fácil, lo que da gran flexibilidad para la construcción del laberinto.

A continuación, se puede apreciar el ensamble de los elementos nombrados anteriormente.

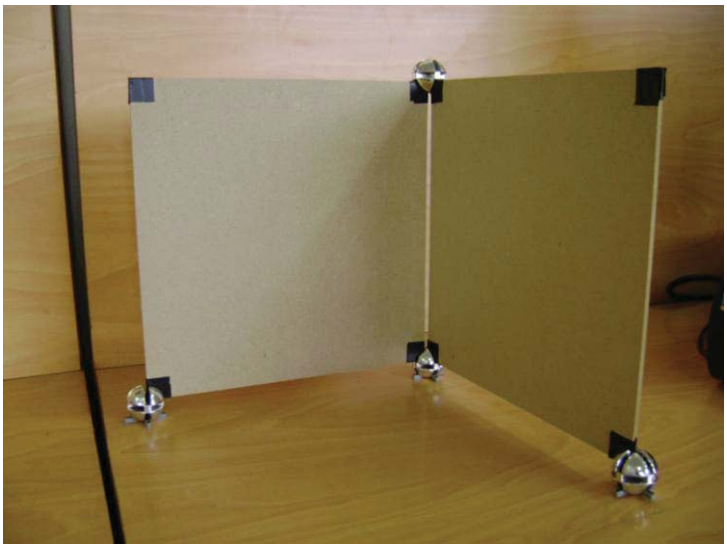


Figura 4.1 Ensamble de paredes del laberinto vista interior.

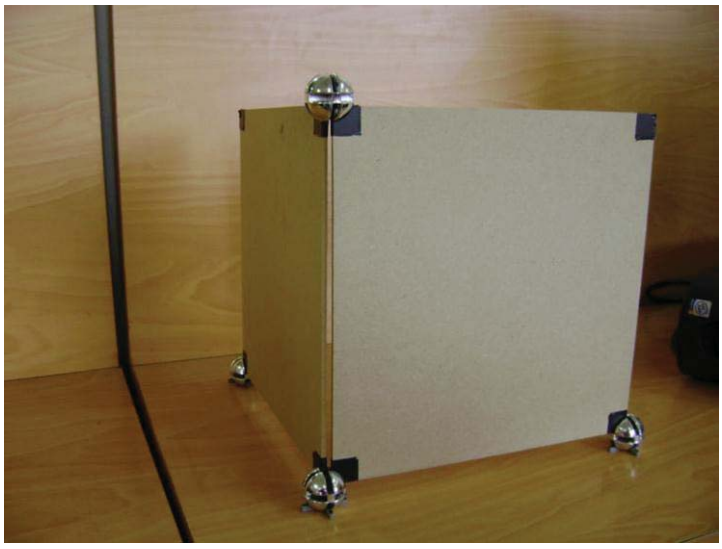


Figura 4.2 Ensamble de paredes del laberinto vista exterior.

Las paredes del laberinto son de madera terciada de 25 centímetros de alto, 25 centímetros de ancho y 0.4 centímetros de espesor. A modo de optimizar la unión pared-conector, se ha puesto cinta adhesiva de color negro en las esquinas de las paredes.

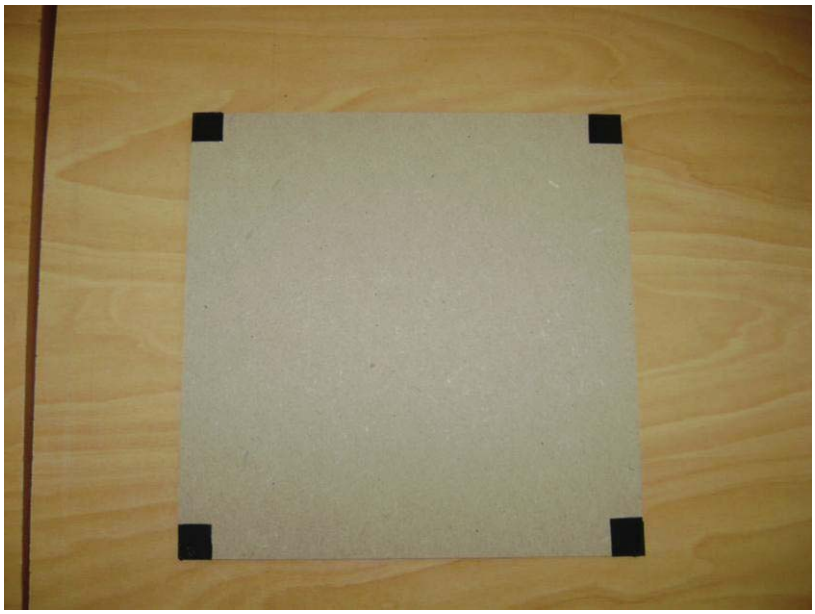


Figura 4.3 Pared del laberinto.

Con respecto a los conectores que se utilizarán en la construcción de laberinto, son los mismos que se usan para construir repisas de vidrios. Las dimensiones de los conectores son 3 centímetros de diámetro, con canales de 0.4 centímetros Estos canales forman ángulos de 90 grados.

Figura 4.4 Conectores del laberinto.

Con respecto a las bases, son separadores plásticos para cerámicas de 0.4 centímetros. Estas bases van ubicadas debajo de los conectores y le proporcionan estabilidad. Tanto la base como el conector tienen un calce perfecto.

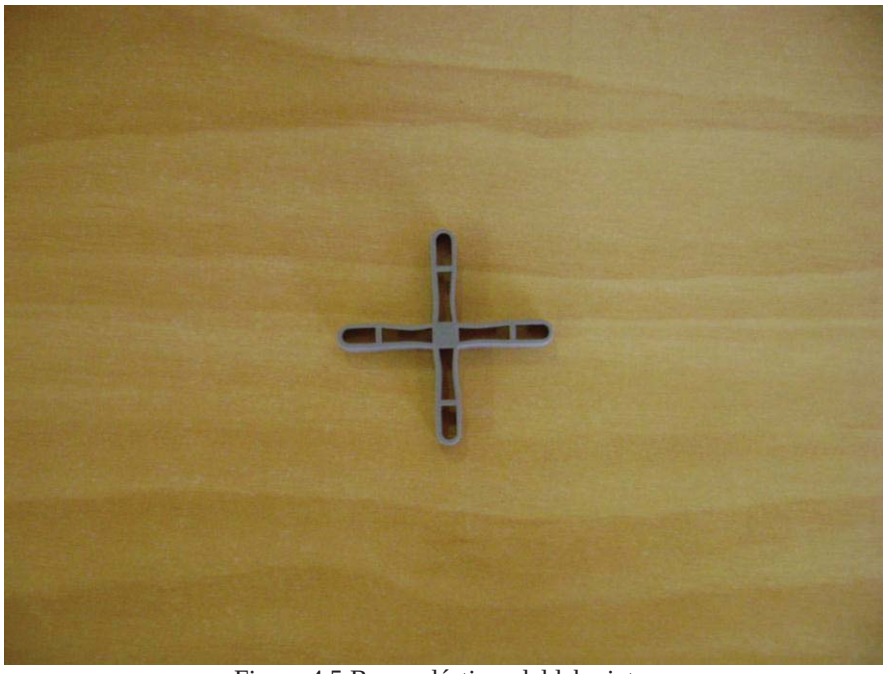


Figura 4.5 Bases plásticas del laberinto.

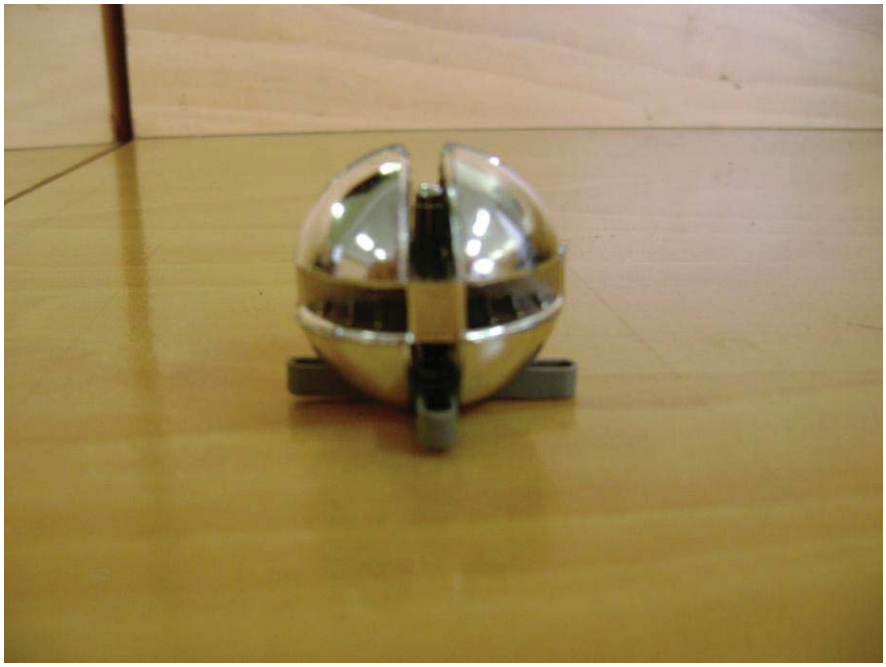


Figura 4.6 Base plástica y conector juntos.

El laberinto a construir debe poseer al menos siete elementos fundamentales: caminos rectos, curvas, bifurcaciones, caminos cerrados, caminos en círculo, una entrada y una salida. Tanto los elementos de caminos cerrados y en círculo agregan complejidad a la determinación del camino solución.

La configuración del laberinto posee 17 caminos rectos, 14 curvas, 5 bifurcaciones, 5 caminos cerrados, 1 camino en círculo, 1 entrada y 1 salida. A continuación, se puede apreciar el laberinto construido con la configuración mencionada anteriormente.

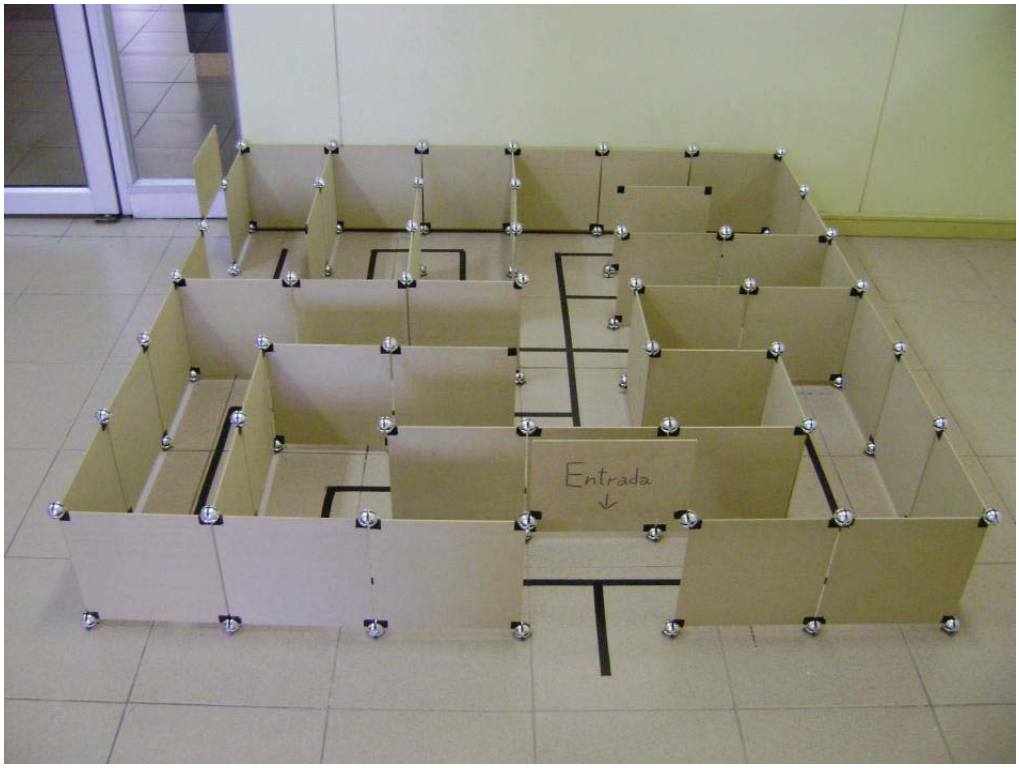


Figura 4.7 Configuración del laberinto.

Las imperfecciones presentes en la configuración del laberinto son principalmente:

- Imperfecciones del terreno, sobre el cual está construido el laberinto.
- Coeficiente de roce no constante en los distintos tramos del laberinto.
- Inexactitud de la distancia entre las distintas paredes.

4.2. Diseño y construcción Robot Lego Mindstorms.

Es indispensable para el trabajo de título que el robot esté correctamente equipado, para ello se dota de sensores que le permitan ubicarse dentro del laberinto y mecanismos de locomoción para que pueda desplazarse dentro del mismo.

Primero, los sensores de ultrasonido serán los encargados de proporcionar "visión" al robot, los cuales utilizan el mismo principio que los murciélagos, se emite un sonido casi imperceptible al oído humano y calcula la distancia midiendo el tiempo que se demora este sonido en llegar, rebotar y volver. El sensor puede llegar a medir hasta 255 centímetros de distancia, con una precisión de +-3 centímetros.

Segundo, con respecto a los mecanismos de locomoción, se utilizarán ruedas anchas "balloon" que permiten conjugar estabilidad y velocidad. El tamaño de ellas aún es una incógnita que se pretende revelar una vez construido el robot y se realicen las pruebas de rigor.

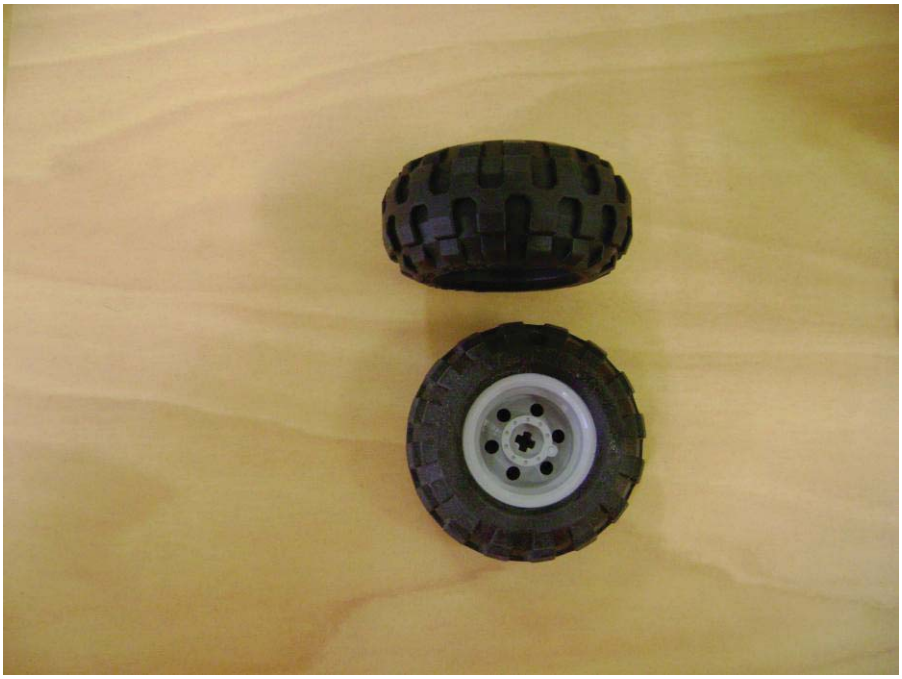


Figura 4.8 Ruedas Balloon.

El siguiente prototipo nos permite tener una idea de la ubicación de los sensores, servomotores, NXT y tipos de ruedas. Si bien no es el modelo definitivo a construir y falta pulir ciertos detalles, es bastante cercano a lo que será el modelo final. El prototipo fue desarrollado con el programa Lego Digital Designer.

Figura 4.9 Prototipo robot vista frontal.

Figura 4.10 Prototipo robot vista lateral.

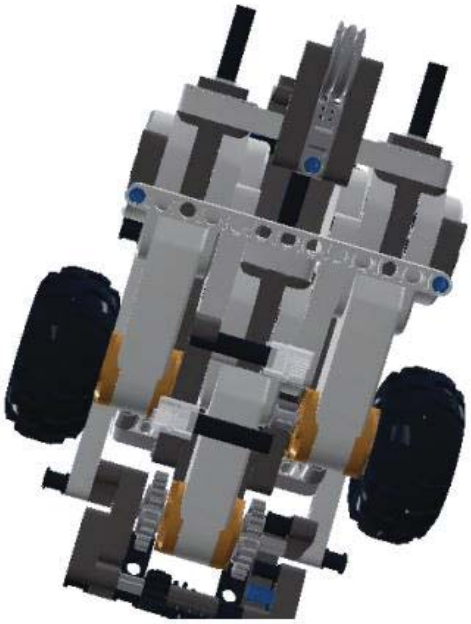


Figura 4.11 Prototipo robot vista inferior.

En estas últimas dos imágenes se puede apreciar lo que denomina la “rueda loca” que ayuda a dar los giros. Esta rueda queda libre y se mueve en todas las direcciones.

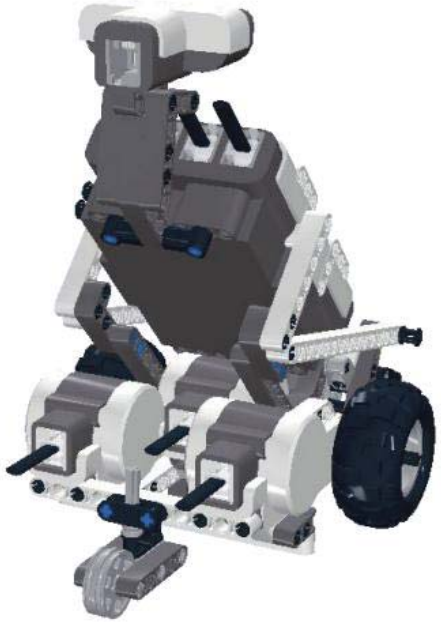


Figura 4.12 Prototipo robot vista posterior.

Finalmente el robot construido cuenta con 2 servomotores, los cuales manejarán el movimiento de las ruedas y 3 sensores ultrasónicos ubicados por sobre el bloque programable, donde su tarea será detectar una posible colisión y así ayudar al desplazamiento del robot dentro del laberinto. Las medidas del robot son 19 centímetros de alto, 13.8 centímetros de ancho y 15 centímetros de largo.

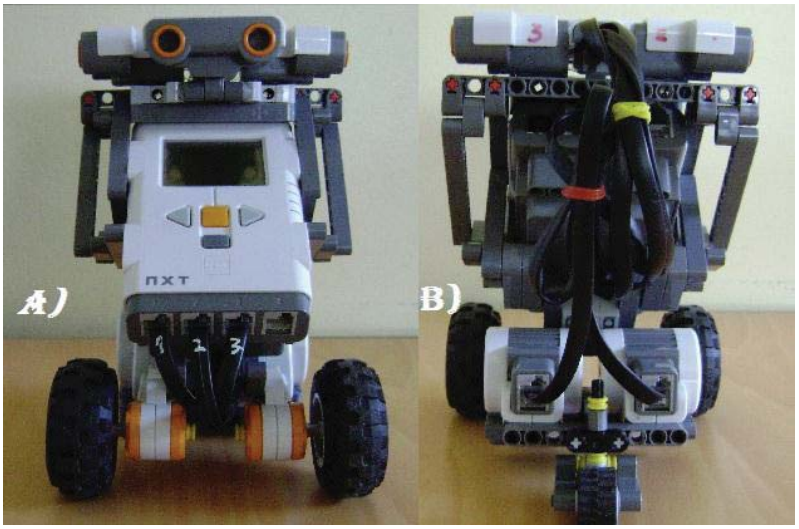


Figura 4.13 Robot, vista frontal y posterior. (a) Vista frontal, (b) Vista posterior.

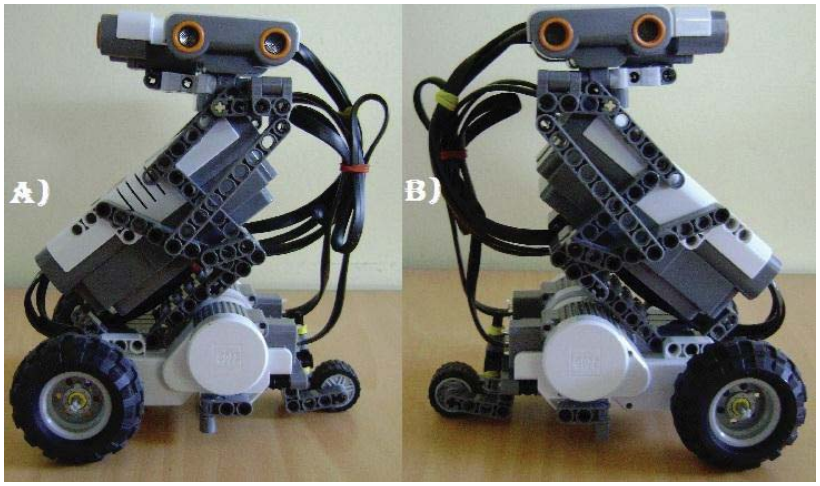


Figura 4.14 Robot, vistas laterales. (a) Vista lateral izquierda, (b) Vista lateral derecha.

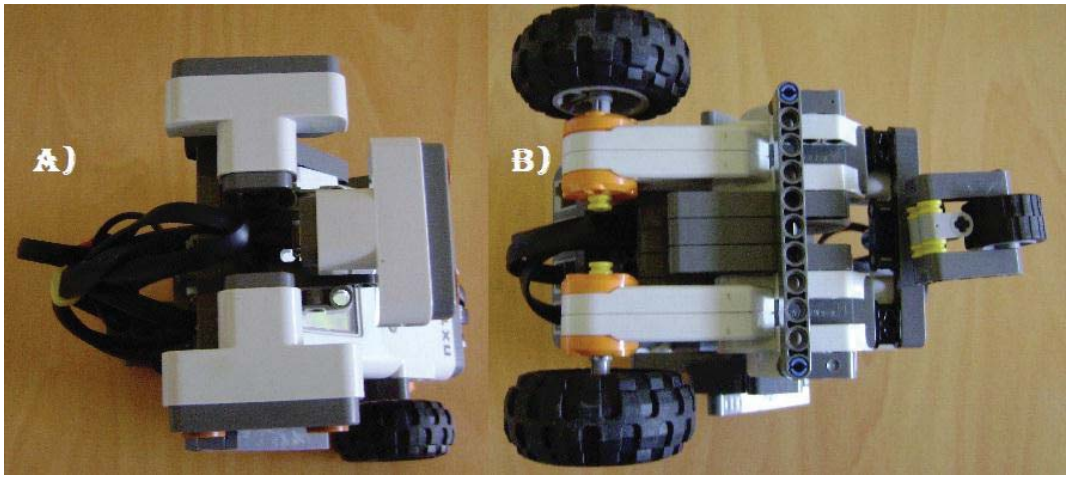


Figura 4.15 Robot, vista desde arriba y abajo. (a) Vista desde arriba, (b) Vista desde abajo.

4.3. Relación entre módulos.

A modo de dejar clara la relación y dependencia entre los distintos módulos se construye el siguiente diagrama de componentes:

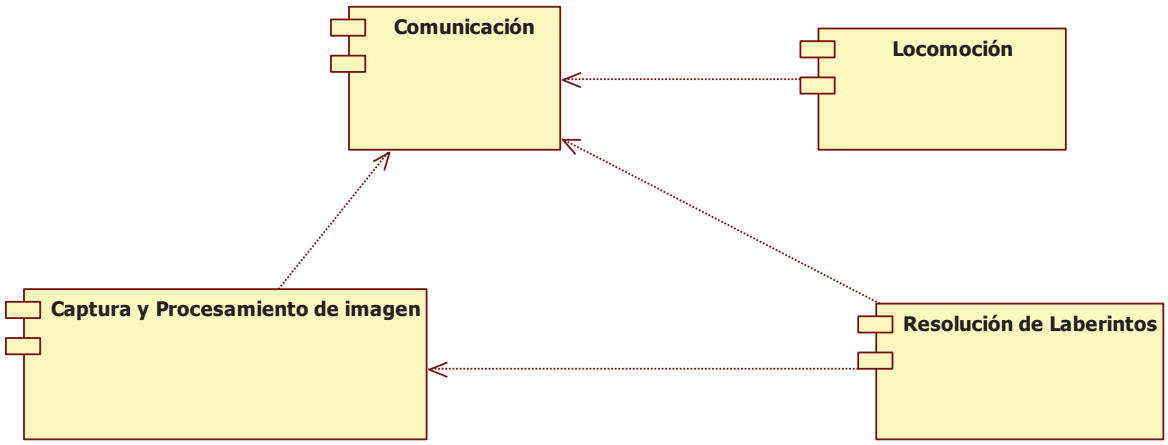


Figura 4.16 Diagrama de componentes.

En el diagrama de componentes podemos apreciar que la solución se compone principalmente de cuatro módulos principales que son:

1. Captura y Procesamiento de imagen. En este módulo los métodos principales son:
   - "AplicarSecuencia", donde se escoge entre las tres secuencias disponibles y se aplica una de ellas.
   - "Reducir", donde se reduce la imagen para una posterior resolución de laberinto.
   - "Cargar imagen", donde se puede buscar y cargar una captura de laberinto que se haya tomado anteriormente.

2. Resolución de Laberintos. En este módulo los métodos principales son:
   - "Resolver_Tremaux". En este método se resuelve el laberinto respetando el algoritmo de Tremaux.
   - "Resolver_Css". En este método se resuelve el laberinto respetando el algoritmo de Callejones sin salida.
   - "Resolver_Cc". En este método se resuelve el laberinto respetando el algoritmo de Callejones ciegos.

3. Comunicación. En este módulo los métodos principales son:
   - "CargarWebcam", donde se captura la imagen del laberinto mediante la cámara web para un posterior procesamiento.
   - "Dar_Instrucciones". Este método establece la comunicación vía bluetooth entre el computador y el robot, para que este último, se desplace dentro del laberinto según el algoritmo de resolución elegido.

4. Locomoción. En este módulo los métodos principales son:
    - “Avanzar”.
    - “Avanzar_Izquierda”.
    - “Avanzar_Derecha”.
    - “Avanzar_Abajo”.
    - “Obtener_Distfront”. Este método obtiene la distancia entre el sensor frontal del robot y la pared del laberinto, evitando así, posibles colisiones.
    - “Obtener_Distizq”. Este método obtiene la distancia entre el sensor izquierdo del robot y la pared del laberinto, evitando así, posibles colisiones.
    - “Obtener_Distder”. Este método obtiene la distancia entre el sensor derecho del robot y la pared del laberinto, evitando así, posibles colisiones.

En primer lugar, el módulo Captura y Procesamiento de imagen depende del módulo de Comunicación, ya que al no existir comunicación entre la cámara y el computador, no podremos realizar la captura del laberinto.

En segundo lugar, se tiene el módulo de Resolución de Laberintos que depende en primera instancia del módulo Captura y Procesamiento de imagen, ya que si éste no le entrega la matriz binaria de la imagen procesada, no podrá aplicar los algoritmos de resolución de laberintos. Por otro lado también depende del módulo de Comunicación, ya que una vez resuelto el laberinto necesita comunicarse con el robot para entregarle las instrucciones y éste escape del laberinto.

Finalmente el módulo de Locomoción depende también del módulo Comunicación para poder recibir las instrucciones de escape del laberinto y poner así en marcha al robot.

Se puede concluir entonces que el módulo más importante es de Comunicación, ya que el resto depende de él para poder ejecutar las diferentes tareas. Este módulo maneja los protocolos USB para conectar la cámara con el computador y el protocolo bluetooth para la conexión computador – robot.

4.4. Captura y Procesamiento de imágenes

El primer módulo a desarrollar dentro del trabajo de título es el de Captura y Procesamiento de imágenes. Para la captura de imágenes se usará la Toolbox de Matlab "Toolbox para la adquisición de imagen", mientras que para procesarla, se usará "Toolbox para el procesamiento digital de imágenes"

Para poder procesar la imagen lo primero que se debe hacer es capturarla. Para ello se dispone de una cámara web USB conectada al computador que será ubicada sobre el laberinto a una altura aproximada de 3.10 metros, soportada por un brazo plegable que tiene una extensión de 75 centímetros. A continuación se puede apreciar tanto el robot, laberinto, brazo y cámara.

Figura 4.17 Entorno completo: Robot, laberinto, brazo y cámara.

Luego de ser capturada, debe ser procesada para rescatar características deseables de la imagen, remover elementos que puedan interferir con el procesamiento. Para todo esto se aplicarán diversos filtros agrupados en secuencias que han sido implementadas para el segundo incremento. Cabe destacar que las paredes al ser tan delgadas (4 milímetros de espesor) no eran captadas por la cámara, por lo mismo, se colocó en el camino del laberinto cinta adhesiva negra.

Las características deseables a rescatar en la imagen son los caminos marcados con la cinta adhesiva negra y los elementos a remover, son principalmente luces y sombras que se generan en la imagen.

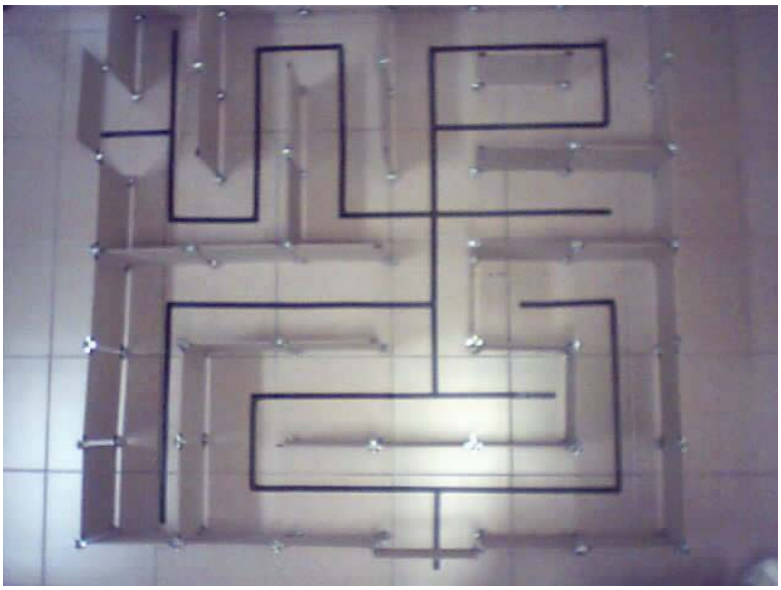


Figura 4.18 Vista superior del laberinto.

4.4.1. Primer Incremento.

El primer incremento captura la imagen desde la cámara web y aplica filtros de manera unitaria, no como secuencias como será en el segundo incremento. Como se puede observar en la siguiente imagen del primer incremento, se observa en tiempo real lo que sucede por la cámara web. Esto ayudará a enfocar y regular de buena manera la cámara antes de tomar una captura que será posteriormente procesada.

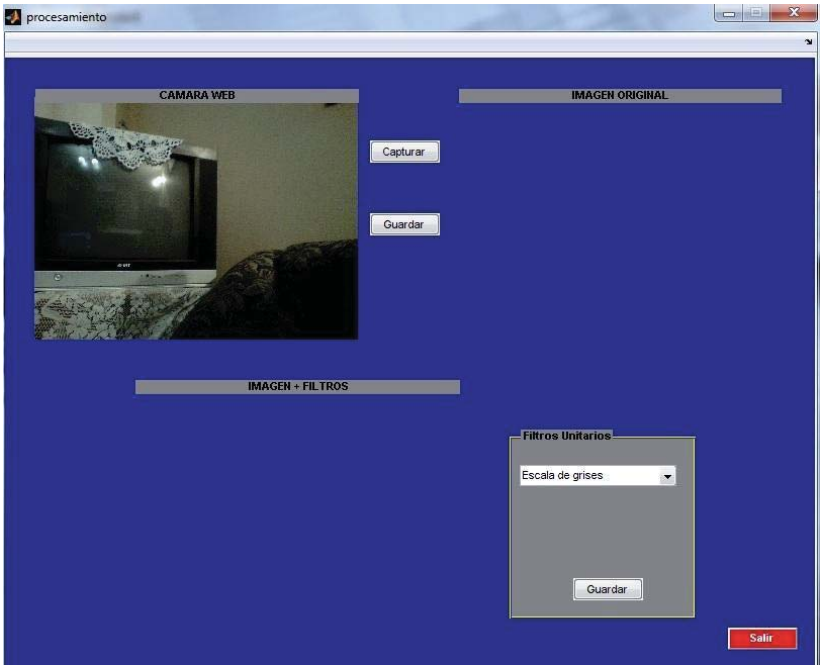


Figura 4.19 Cámara web.

Una vez que esté bien enfocada la cámara web, se procede a capturar la imagen la que aparecerá bajo la etiqueta de "imagen original" e "imagen + filtros". Cabe destacar que el botón "Capturar" puede ser usado las veces que se estime necesario hasta lograr la captura de una imagen correcta.

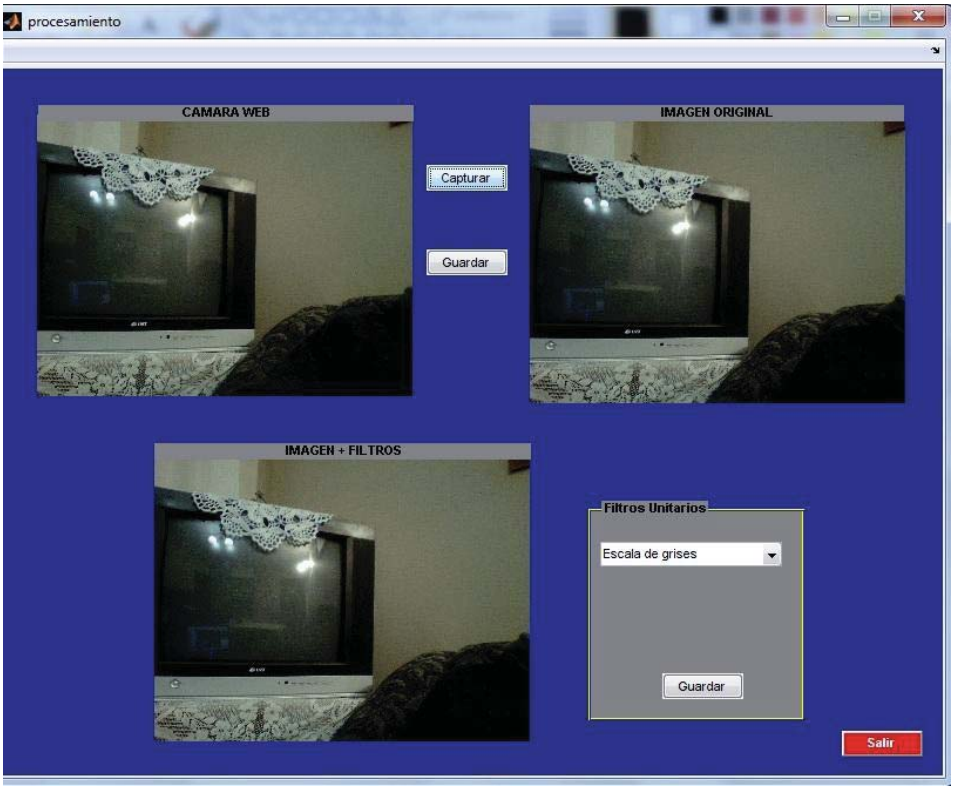


Figura 4.20 Capturas de cámara web.

Luego a la "Imagen + filtros", se le pueden ir aplicando una serie de filtros, que por el momento son unitarios, pero cuando se establezcan las secuencias de filtros, serán reemplazados por ellas.

Los filtros que se encuentran implementados a prueba son: Escala de grises, Sobel, Prewitt, Roberts, Log (Laplacian of Gaussian), Zerocross, Umbral.

Figura 4.21 Menú Popup de filtros.

Finalmente, se muestran algunos resultados de aplicaciones de filtros a la imagen original, que fueron guardadas con el botón integrado en el panel de filtros unitarios.

En esta primera imagen se muestra el efecto que provoca aplicar el filtro Escala de grises (Ver Figura 4.22), donde se pasa de una imagen a color capturada por la cámara a una con diversas tonalidades de grises.

Figura 4.22 Aplicación de Escala de grises.

En esta segunda imagen se muestra el filtro Sobel (Ver Figura 4.23), que su principal objetivo es resaltar bordes como se aprecia en la imagen. Para poder aplicar este filtro necesariamente debe aplicarse antes el de Escala de grises.

Figura 4.23 Aplicación de filtro Sobel.

En la tercera imagen se aplica el filtro Zerocross (Ver Figura 4.24) que al igual que el anterior detecta bordes, a grandes rasgos busca las zonas donde hace cambios rápidos de tono el píxel y los define como fronteras, o sea, bordes. Al igual que el filtro anterior necesita de la Escala de grises, previamente.

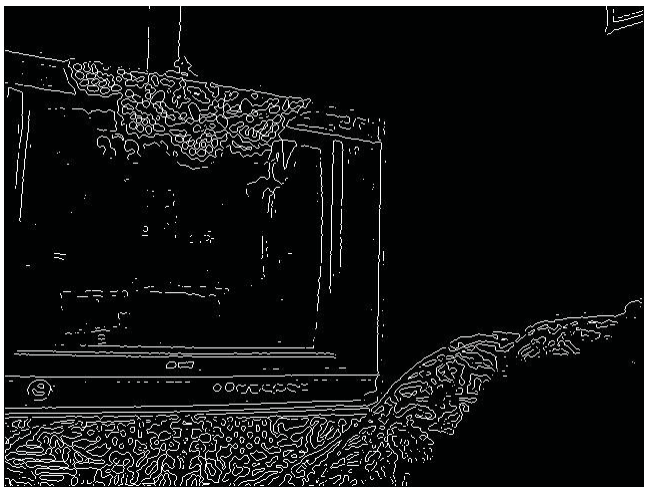


Figura 4.24 Aplicación de filtro Zerocross.

Finalmente el filtro de Canny detecta bordes (Ver Figura 4.25). Idealmente debe aplicarse después de algún algoritmo que reduzca el ruido para así obtener resultados satisfactorios. Para poder aplicar este filtro necesariamente debe aplicarse antes el de Escala de grises.

Figura 4.25 Aplicación de filtro Canny.

Una vez instalada la cámara web sobre el laberinto, se harán las pruebas correspondientes aplicando los filtros sobre el objetivo real del trabajo de título que es el laberinto y es ahí, donde se podrán establecer las diversas secuencias de filtros a ser utilizadas en el trabajo de título. Se cree que uno de los filtros llamado a ser importante en estas secuencias es el filtro de umbralización, que además puede ser regulada la intensidad con que desea ser aplicada.

Finalizado este primer incremento se intentó cambiar de entorno de desarrollo (Octave por Matlab) en lo que se refiere a la captura y procesamiento de imágenes. Octave si bien es muy similar a Matlab en diversos ámbitos, desde el cómo se presenta el entorno al usuario hasta los comandos utilizados para la programación, presentó numerosos problemas de compatibilidad al momento de "especializarse" en un área como era la captura y procesamiento de imágenes. El llamado 99% de compatibilidad entre ambos entornos de desarrollo, claramente se restringe siempre y cuando no se usen las toolbox especializadas.

4.4.2. Segundo Incremento.

Para este segundo incremento ya se cuenta con la cámara instalada por encima del laberinto, además se instala un foco halógeno para lograr una buena iluminación del laberinto y a su vez, una mejor captura del mismo. Finalmente se agruparon los filtros en tres secuencias, que se encuentran en proceso de ajustes y pruebas para obtener los resultados deseados.

La siguiente imagen muestra que existen dos opciones para ingresar una imagen al software y que ésta sea procesada. La primera de ellas es mediante cámara web donde se puede apreciar lo que muestra la cámara en tiempo real y realizar capturas del laberinto (Ver figura 4.27).

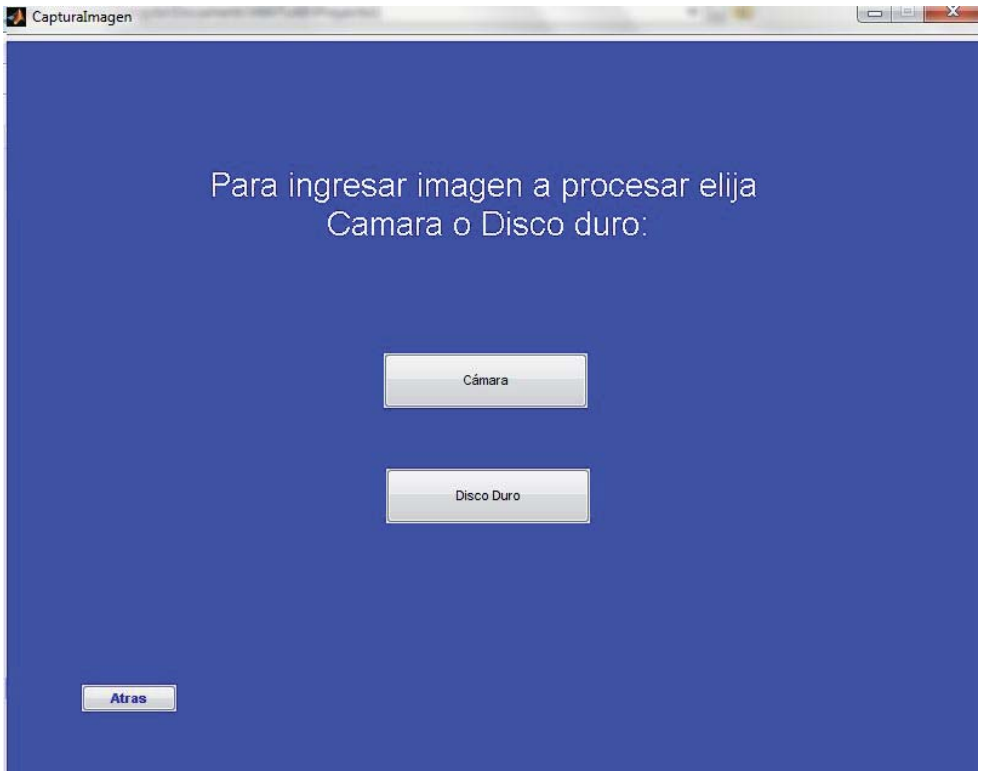


Figura 4.26 Opciones de ingreso de imagen.

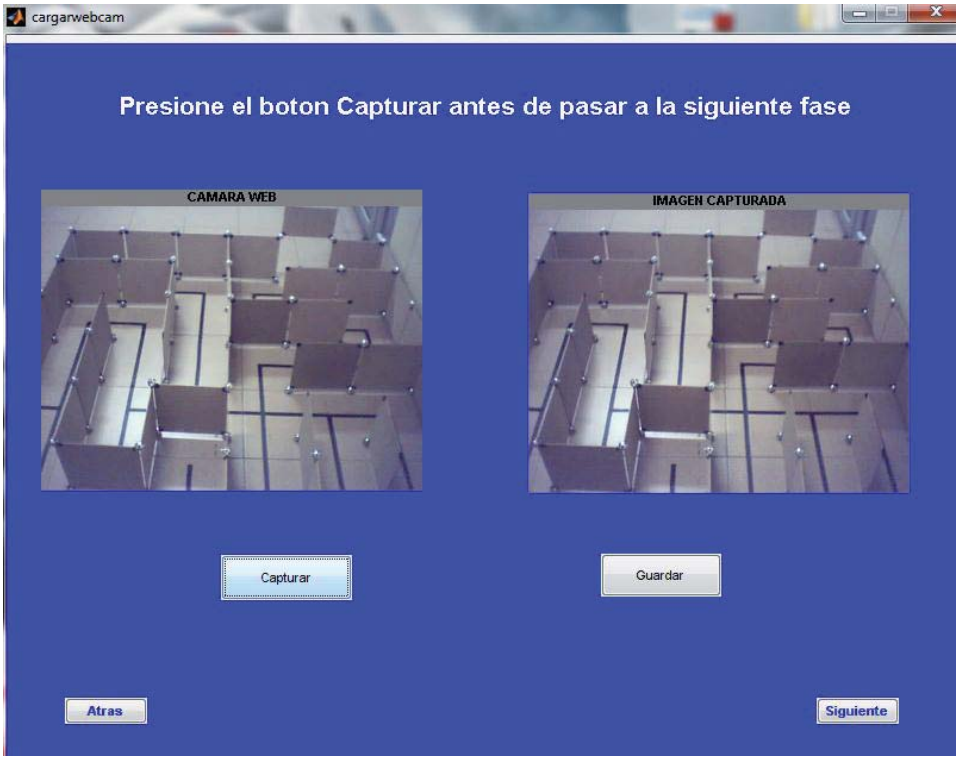


Figura 4.27 Capturar imagen cámara web.

La segunda opción, es cargar la imagen desde el computador que se había capturado en otra ocasión (Ver figura 4.28).

Figura 4.28 Cargar imagen disco duro

Finalmente existe la opción de aplicar secuencias pre-establecidas a la imagen que se ha cargado. Por ejemplo, en la secuencia 1 se aplican los filtros de: escala de grises, aumento de brillo, umbralización (factor 0.6), Sobel.

Figura 4.29 Escoger secuencia.

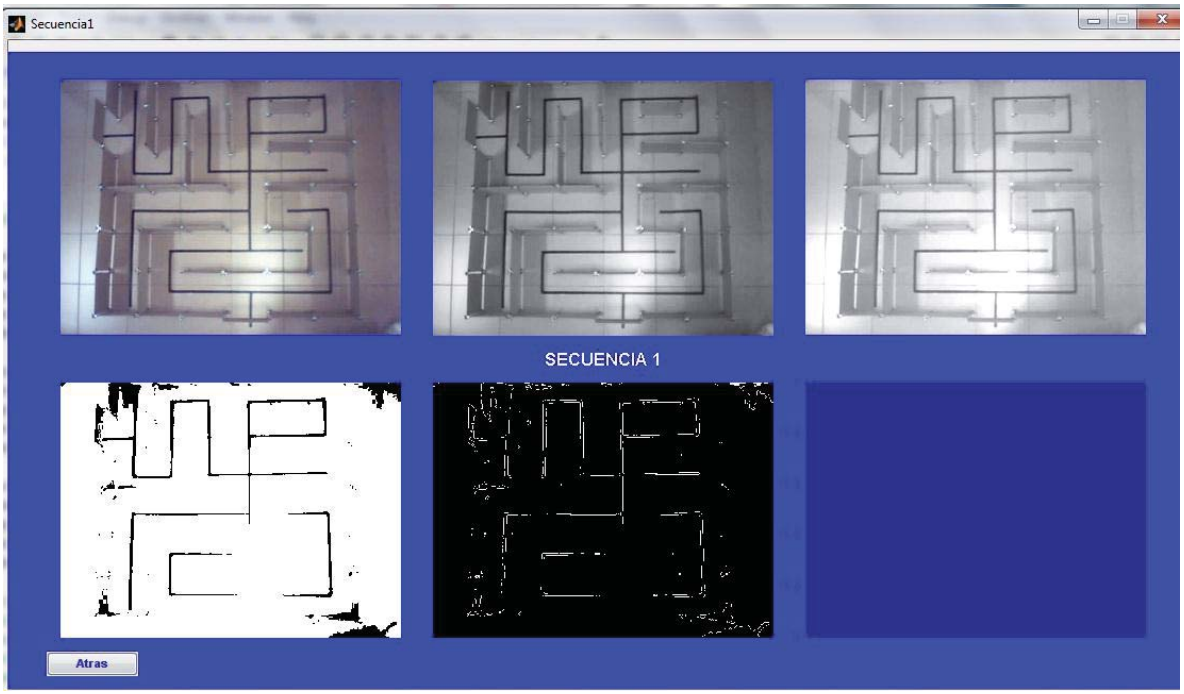


Figura 4.30 Aplicación secuencia 1.

4.4.3. Tercer Incremento.

Para este tercer incremento están completamente desarrolladas las secuencias de filtros de imágenes. Finalmente en la secuencia número 1 se aplican los siguientes filtros: Escala de grises, Aumento de brillo (+50%), umbral (0.75), Detector de bordes Canny y Desenfoque Gaussiano (15,3). Todo lo anterior se puede ver en la imagen que sigue.

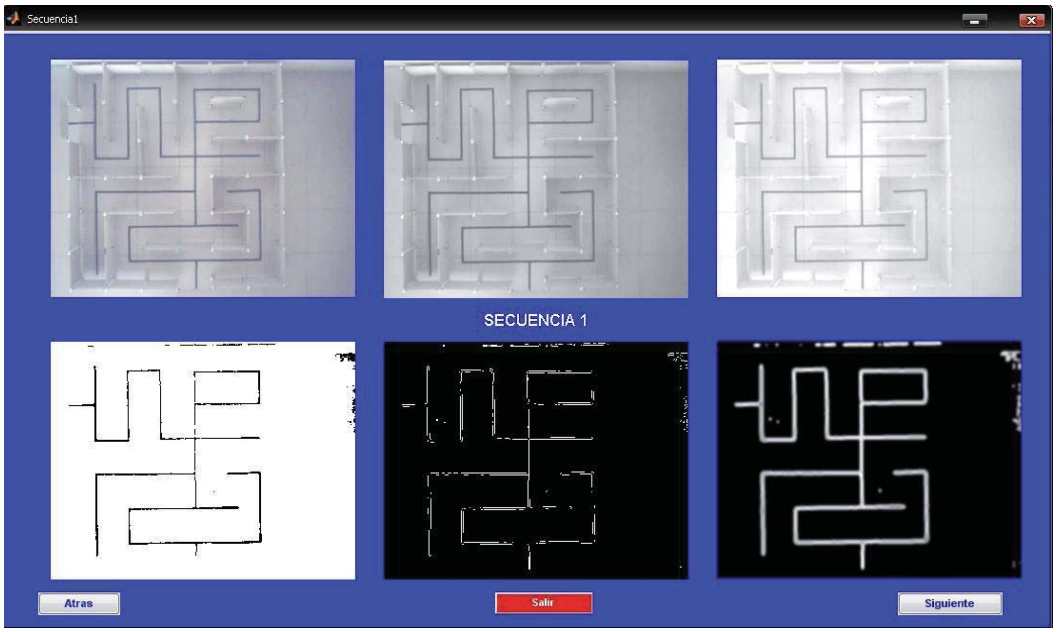


Figura 4.31 Secuencia 1.

En la secuencia número 2 se aplican los siguientes filtros: Escala de grises, Desenfoque Gaussiano (14, 0.5), Disminución de brillo (-142%), umbral (0.5), Detector de bordes Canny y Desenfoque Gaussiano (15,190). El séptimo filtro se muestra en el primer cuadrante. El detalle se puede apreciar en la imagen que sigue.

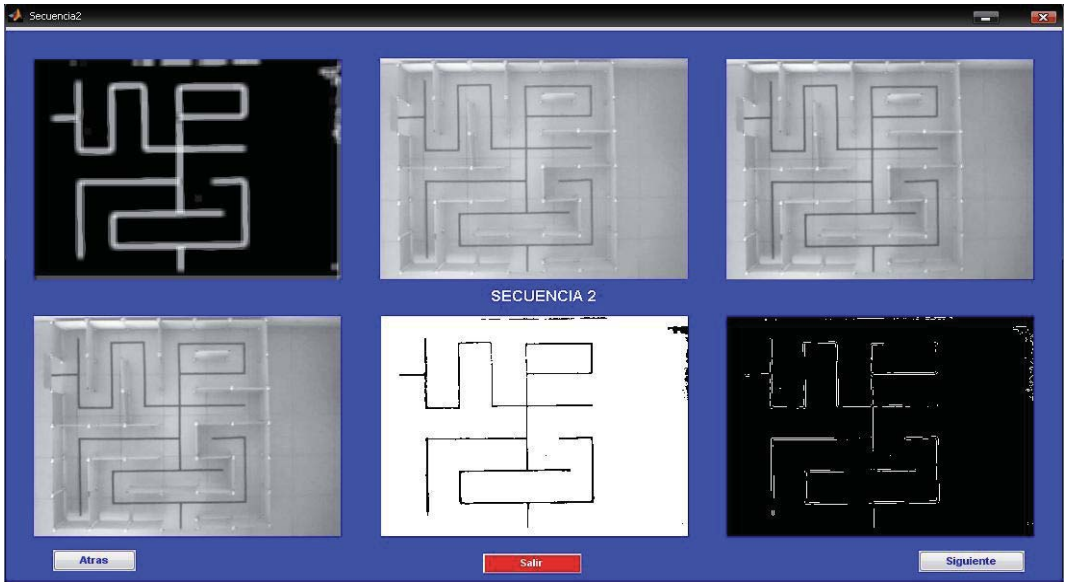


Figura 4.32 Secuencia 2.

En la secuencia número 3 se aplican los siguientes filtros: Escala de grises, umbral (0.55), Detector de bordes Sobel, Desenfoque Gaussiano (7,20), Detector de bordes Sobel, Desenfoque Gaussiano (1,20), Detector de bordes Sobel y Desenfoque Gaussiano (20,100). Desde el séptimo filtro comienza a mostrarse sobre los filtros aplicados inicialmente. La idea en los últimos filtros de repetirlos es que se va logrando engrosar el camino del laberinto. El desarrollo de cada filtro puede apreciarse en la imagen que sigue.

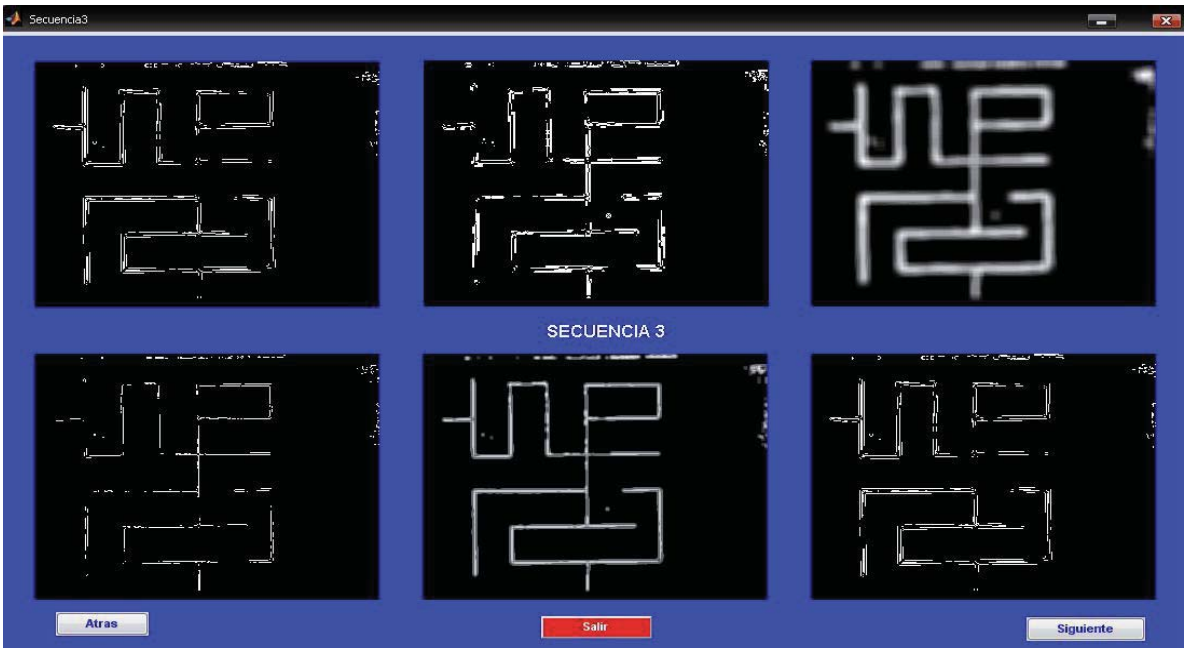


Figura 4.33 Secuencia 3.

Al finalizar el tercer incremento se produjo un problema con Matlab, al usar Windows 7 de 64 bits no se carga el compilador que genera los ejecutables (que se necesitaba para la integración con Java), pero al bajar a Windows XP de 32 bits no hubo problema alguno.

4.4.4. Otras consideraciones.

Uno de los principales problemas al realizar la captura de la imagen, es que dependiendo de la hora del día y debido a la ubicación física del laberinto se generan numerosas sombras, bajos contrastes que complican el procesamiento de la misma. Es por eso que la iluminación es un factor que afecta considerablemente la complejidad de los algoritmos, por lo que una buena iluminación permite simplifica el análisis.

La verdad es que existen variadas técnicas de iluminación (Valiente, 2006), pero se probarán solo tres que son aplicables a la naturaleza del trabajo de título, descartando la técnica de iluminación a contraluz, estructurada y coaxial, algunos por lo difícil de montar y otros por el alto coste. Por lo tanto, las técnicas que se analizarán son:

- Iluminación ambiental: Los objetos no están expuestos de manera directa a una fuente de luz, de modo que se tiene una iluminación uniforme, ya que los haces de luz inciden de forma constante en cada objeto.

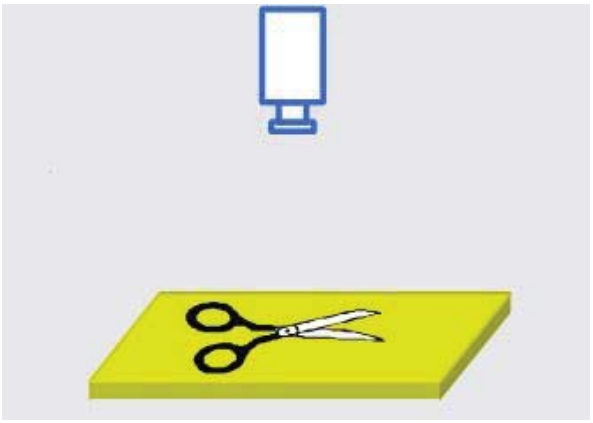


Figura 4.34 Iluminación ambiental.

- Iluminación direccional: Se aplica iluminación directamente a un objeto, de modo que el haz luminoso incida sobre el objeto para aprovechar la formación de sombras y de esta manera resaltar sus características.

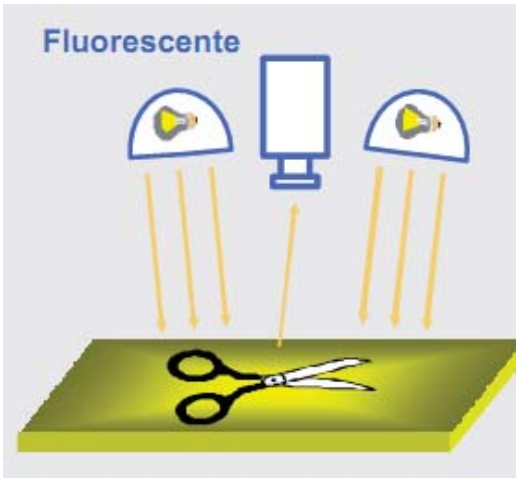


Figura 4.35 Iluminación direccional.

- Iluminación difusa: Los haces luminosos inciden sobre el objeto desde todas las direcciones, de modo que no se generan sombras y como consecuencia de esto, se obtiene un mínimo de contraste.

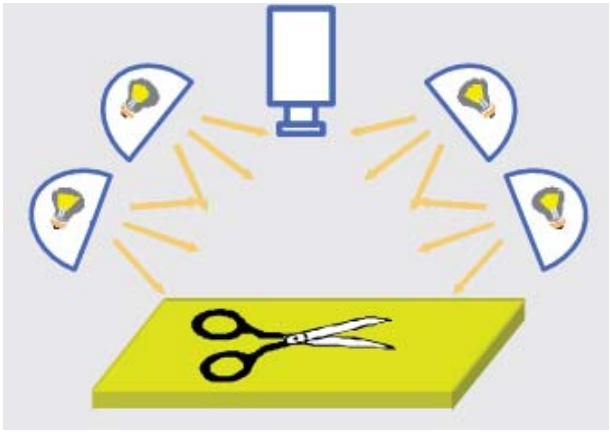


Figura 4.36 Iluminación difusa.

Luego de diversas pruebas realizadas con las tres técnicas de iluminación elegidas, se optó por seleccionar la "técnica de iluminación ambiental", ya que se obtiene una iluminación uniforme y adecuada para una buena captura de la imagen del laberinto que posteriormente será procesada.

4.5. Resolución de Laberintos.

Además, se desarrolló el módulo de resolución de laberinto, en donde el proceso comienza al recibir una imagen binaria y esta debe transformarse a matriz binaria para poder ser procesada y encontrar solución al laberinto.

4.5.1. Transformación imagen binaria a matriz binaria.

La transformación de imagen a matriz se hará con Java y sus librerías de manejo de imágenes. En este caso en particular con la librería java.awt.image. Entre las capacidades de procesamiento de esta librería podemos rescatar tres grandes interfaces (Oracle, 2003):

- ImageProducer.
- ImageConsumer.
- ImageObserver.

Las clases que implementa el interfaz ImageProducer sirven como fuentes de pixeles. Los métodos de estas clases pueden generar los pixeles a partir de la pantalla, o puede interpretar cualquier otra fuente de datos, como un fichero GIF. No importa como genere los datos, el principal propósito de un productor de imágenes es proporcionar pixeles a una clase ImageConsumer.

Los productores de imágenes operan como fuentes de imágenes y, el modelo productor/consumidor que se sigue en el tratamiento de imágenes es el mismo que se utiliza en el modelo fuente/receptor de eventos.

En particular, los métodos declarados en el interfaz ImageProducer hacen posible que uno o más objetos ImageConsumer se registren para mostrar su interés en una imagen. El productor de la imagen invocará a los métodos declarados en el interfaz ImageConsumer para enviar pixeles a los consumidores de imágenes.

Un productor de imágenes puede registrar muchos consumidores de sus pixeles de la misma forma que una fuente de eventos puede registrar múltiples receptores de sus eventos. En el interfaz ImageProducer hay varios métodos para manipular la lista de consumidores, por ejemplo, para añadir nuevos consumidores interesados en los pixeles del productor o para eliminar consumidores de la lista.

El interfaz ImageConsumer declara métodos que deben ser implementados por las clases que reciben datos desde un productor de imágenes. El principal de estos métodos es setPixels().

Las principales clases a ser utilizadas son 6 y ayudarán a generar dicha matriz binaria (Oracle, 2003).

- Clase ColorModel.

Dependiendo de cómo se quiera representar la imagen en pantalla, un solo píxel puede contener mucha información o casi ninguna. Por ejemplo, si se quieren dibujar una serie de líneas del mismo color sobre un fondo de un color diferente, la única información que debe contener cada píxel es si debe estar encendido o apagado, es decir, que solamente sería necesario un bit de información para representar al píxel.

En el otro extremo de la información que puede recoger un píxel se encuentran los gráficos complejos de los juegos de ordenador, que requieren 32 bits para representar un píxel individual. Estos 32 bits se dividen en grupos de cuatro octetos de bits o bytes. Tres de estos grupos representan la cantidad de colores fundamentales: rojo, verde y azul que forman parte del color del píxel; y el cuarto byte, normalmente conocido con byte alpha, se utiliza para representar la transparencia, en un rango de 0 a 255, siendo 0 la transparencia total y 255 la opacidad completa.

En teoría, este formato permite la utilización de 16 millones de colores en cada píxel individual, aunque puede ser que el monitor o la tarjeta gráfica no sean capaces de visualizar electrónicamente tal cantidad de colores. Cada uno de los extremos se puede considerar como un modelo de color. En Java, un modelo de color determina la cantidad de colores que se van a representar dentro del AWT. La clase ColorModel es una clase abstracta que se puede extender para poder especificar representaciones de color propias.

Un modelo de color es necesario porque hay muchos métodos que reciben un array de bytes y convierten esos bytes en pixeles para su presentación en pantalla, es decir, convierten los bytes de datos en pixeles visuales sobre la pantalla. Por ejemplo, con un modelo de color directo simple, cada grupo de cuatro bytes se podría interpretar como la representación de un valor del color de un píxel individual. En un modelo de color indexado simple, cada byte en el array se podría interpretar como un índice a una tabla de enteros de 32 bits donde cada uno de esos enteros representa el valor del color adscrito al píxel.

El AWT proporciona tres subclases de ColorModel: IndexColorModel, ComponentColorModel y PackedColorModel.

- Clase ColorModel: Subclase IndexColorModel:

Cuando se utilizan imágenes de alta resolución, se consume gran cantidad de memoria; por ejemplo, una imagen de dimensiones 1024x768, contiene 786.432 pixeles individuales. Si se quieren representar estos pixeles con sus cuatro bytes en memoria, se necesitarán 3.145.728 bytes de memoria para esta imagen.

Para evitar este rápido descenso de memoria, se han incorporado varios esquemas de forma que las imágenes se puedan representar de forma razonable sin comprometer demasiado el uso de memoria. Un esquema muy común consiste en localizar un pequeño número de bits

para cada píxel (8 bits por ejemplo) y luego utilizar los valores de los pixeles como un índice a una tabla de enteros de 32 bits que contenga el subconjunto de todos los colores que se utilizan en la imagen. Por ejemplo, si se utilizan ocho bits para representar un píxel, el valor del píxel se puede utilizar como índice a una tabla de contenga hasta 256 colores que se utilicen en la imagen. Estos 256 colores se pueden seleccionar entre los millones disponibles, conociéndose a este subconjunto como la paleta de colores de la imagen.

- Clase DirectColorModel:

Esta clase se extiende a la clase PackedColorModel e implementa el formato completo de color de 32 bit, donde cada píxel está formado por los cuatro bytes, representando el canal alpha y las cantidades de rojo, verde y azul que componen cada píxel

- Clase FilteredImageSource:

Esta clase es una implementación del interfaz ImageProducer que toma una imagen y un objeto de tipo ImageFilter para generar una nueva imagen que es una versión filtrada de la imagen original. Las operaciones de filtrado pueden realizar una gran variedad de operaciones, como son: el desplazamiento y sustitución de colores, la rotación de imágenes, etc.

- Clase ImageFilter:

Esta clase es una implementación del interfaz ImageConsumer. Además de los métodos declarados en el interfaz, hay métodos para implementar un filtro nulo, que no realiza modificación alguna sobre los datos que se le pasan.

Hay varias subclases que extienden esta clase base para permitir la manipulación de la imagen, como son RGBImageFilter, CropImageFilter, ReplicateScaleFilter y AreaAveragingScaleFilter.

- Clase MemoryImageSource:

Esta clase es una implementación del interfaz ImageProducer que utiliza un array de datos para generar los valores de los pixeles de una imagen. Dispone de varios constructores que son:

- Ancho y alto en pixeles de la imagen que va a ser creada.
- Modelo de color que se empleará en la conversión. Si el constructor no requiere este parámetro, utilizará el modelo de color RGB por defecto.
- Un array de bytes o enteros conteniendo los valores a ser convertidos en píxel según el modelo de color especificado.
- Un offset para indicar el primer píxel dentro del array.
- El número de pixeles por línea en el array.
- Un objeto de tipo Hashtable conteniendo las propiedades asociadas de la imagen, en caso de que exista.

En todos los casos, el constructor genera un objeto ImageProducer que se utiliza como un array de bytes o enteros para generar los datos de un objeto Image. Esta clase puede pasar múltiples imágenes a consumidores interesados en ellas, recordando a la funcionalidad multiframe que ofrece el formato gráfico GIF89a para producir animaciones.

- Clase PixelGrabber:

Esta es una clase de utilidad para convertir una imagen en un array de valores que corresponden a sus pixeles. Implementa el interfaz ImageConsumer, que puede ser acoplado a un objeto Image o ImageProducer para realizar cualquier manipulación de esa imagen, y en el fondo, es la que específicamente se ocupará para poder hacer la transformación a la matriz binaria.

4.5.2. Obtención de matriz simplificada para aplicación de algoritmos.

Si bien se ha obtenido una matriz con los valores 0 y 1 de cada píxel de la imagen binaria, se necesita limpiar de "manchones" y reducir lo más posible la matriz para poder aplicar los algoritmos sobre ella de manera simple, y poder enviar las instrucciones al robot con la ruta de la solución óptima.

Con la imagen cargada como una matriz binaria, se comienza a recorrer los valores de sus pixeles, estableciendo aproximadamente el ancho del camino de la solución. Con el ancho definido, se puede aplicar un método en la matriz de manera parcial, identificando la mayoría de pixeles que se encuentran en el sector. Por ejemplo, si en un sector de 8x8 hay 60 pixeles negros y 4 blancos, se determina que el sector es negro. Con esto, se puede ir obteniendo una matriz reducida donde el tamaño del camino solución sea idealmente uno o dos pixeles de ancho y no cientos de ellos que pueden dificultar la obtención de la solución.

Figura 4.37 Matriz limpia y primera reducción de 640x480 a 121x112 pixeles.

Con la matriz binaria reducida se pueden aplicar los algoritmos de resolución de laberintos a esta matriz y, una vez resuelta, enviar las instrucciones al robot, siendo éste una caja negra que sólo recibe instrucciones acerca del camino solución.

4.5.3. Ejemplo de resolución de laberinto.

Una vez obtenida la imagen reducida y limpia se procede a resolver el laberinto. Para explicar el procedimiento se crea una matriz ideal en la cual se ilustrará paso a paso el desarrollo del algoritmo creado. Para la resolución del laberinto se usó la siguiente simbología: 0 para identificar una pared, 1 para el camino, 9 para el camino solución y 4 para marcar caminos bloqueados. En la siguiente imagen se puede apreciar la matriz ideal.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1	0	0	0	1	0
0	1	0	1	0	1	0	1	1	1	1	1	0
0	1	0	1	0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	1	1	1	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0	0	0	1	0
0	1	0	1	1	1	1	1	1	1	0	1	0
0	1	0	1	0	0	0	0	0	0	0	1	0
0	1	0	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0

Figura 4.38 Matriz ideal.

En el ejemplo las imágenes mostrarán la resolución del laberinto según el algoritmo de Tremaux. En esta primera matriz se observa que se entró en el laberinto y ante dos posibles opciones, escogerá una en forma aleatoria tal como lo indica el algoritmo de Tremaux.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1	0	0	0	1	0
0	1	0	1	0	1	0	1	1	1	1	1	0
0	1	0	1	0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	1	1	1	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0	0	0	1	0
0	1	0	1	1	1	1	1	1	1	0	1	0
0	1	0	1	0	0	0	0	0	0	0	1	0
0	1	0	1	1	1	1	9	1	1	1	1	0
0	0	0	0	0	0	0	9	0	0	0	0	0

Figura 4.39 Avance dentro del laberinto.

Aleatoriamente se escogió el camino de la derecha que no tiene salida, por lo tanto, debe ser marcado como bloqueado. Ahora, para entender cómo se bloquea dicho camino se hace la siguiente pregunta ¿el pixel que sigue es 1?, de ser así avanza y se marca con un 9. Nuevamente se realiza una pregunta: ¿el pixel que sigue es 1?, la respuesta es no, entonces se pregunta ¿el antecesor es 9? y la respuesta es afirmativa, entonces se concluye que es un camino sin salida y se devuelve bloqueando con 4.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1	0	0	0	1	0
0	1	0	1	0	1	0	1	1	1	1	1	0
0	1	0	1	0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	1	1	1	1	1	0	1	9	9	0
0	1	0	0	0	0	0	1	0	0	0	9	0
0	1	0	1	1	1	1	1	1	1	0	9	0
0	1	0	1	0	0	0	0	0	0	0	9	0
0	1	0	1	1	1	1	9	9	9	9	9	0
0	0	0	0	0	0	0	9	0	0	0	0	0

Figura 4.40 Escoge dirección aleatoria.

En la imagen que sigue observamos que bloqueó el camino sin salida y retornó al inicio de éste y tomó el camino de la izquierda. Así continua avanzando y bloqueando los caminos sin salida a los que ingresa aleatoriamente.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1	0	0	0	1	0
0	1	0	1	0	1	0	1	1	1	1	1	0
0	1	0	1	0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0
0	4	4	4	4	4	4	9	0	4	4	4	0
0	4	0	0	0	0	0	9	0	0	0	4	0
0	4	0	9	9	9	9	9	1	1	0	4	0
0	4	0	9	0	0	0	0	0	0	0	4	0
0	4	0	9	9	9	9	9	4	4	4	4	0
0	0	0	0	0	0	0	9	0	0	0	0	0

Figura 4.41 Bloquea caminos.

Finalmente, se aprecia que se logra bloquear el camino en círculo o también denominado parte disjunta, quedando marcado el camino solución con números 9 que fue la simbología designada.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	4	0	9	9	9	0	4	4	4	4	4	0
9	9	0	9	0	9	0	4	0	0	0	4	0
0	9	0	9	0	9	0	4	4	4	4	4	0
0	9	0	9	0	9	0	4	0	0	0	0	0
0	9	9	9	0	9	9	9	4	4	4	4	0
0	0	0	0	0	0	0	9	0	0	0	0	0
0	4	4	4	4	4	4	9	0	4	4	4	0
0	4	0	0	0	0	0	9	0	0	0	4	0
0	4	0	9	9	9	9	9	1	1	0	4	0
0	4	0	9	0	0	0	0	0	0	0	4	0
0	4	0	9	9	9	9	9	4	4	4	4	0
0	0	0	0	0	0	0	9	0	0	0	0	0

Figura 4.42 Camino solución.

4.6. Locomoción Robot.

Se desarrollaron algoritmos de locomoción para el robot con los movimientos básicos necesarios para que pueda desplazarse dentro del laberinto, apoyado por los sensores ultrasónicos para evitar colisiones. Los movimientos fueron escritos en lenguaje JAVA apoyados por la librerías Icommand 0.7 y lejOS 1.03 Todo lo anteriormente nombrado está en la clase Movimientos.

El algoritmo de locomoción será usado una vez resuelto el laberinto para enviar las instrucciones al robot, pero surge la problemática de cómo serán analizados los giros para evitar que sean erróneos. Por ejemplo, si viene bajando, es giro a la derecha y si viene subiendo es giro a la izquierda. Pero en ambos casos, visto de manera matricial sería un giro a la izquierda, entonces en el caso de venir bajando provocaría un giro erróneo. Para ello se implementó un sistema de coordenadas, Norte – Sur – Este – Oeste, que permite orientarse de la posición que lleva el robot y la dirección real en que debe realizar el giro (ver figura 4.43).

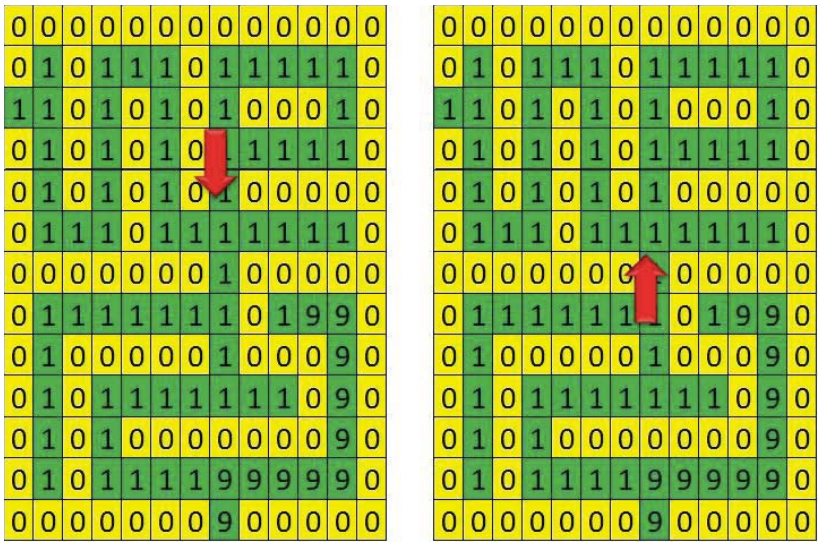


Figura 4.43 Posibilidad de giros erróneos.

Los movimientos básicos son: giro a la izquierda (90º), giro a la derecha (90º), giro (180º), avanzar (avanza hasta que el sensor ultrasónico frontal detecte una pared a 30 centímetros). Para poder ejecutar los movimientos básicos se usa de apoyo la clase Pilot de la librería Icommand que provee los mecanismos necesarios para que el robot pueda desplazarse y ejecutar los movimientos. Para que pueda lograr moverse el robot sin colisionar, dentro de la misma clase Movimientos se implementaron métodos para capturar las distancias que son obtener_distfront, obtener_distizq y obtener_distder.

En cuanto a la corrección de los movimientos del robot mientras se desplaza por el laberinto y para evitar que choque con las paredes mientras avanza, se dispuso de una lectura cada dos pixeles de los sensores ultrasónicos, donde se detendrá y analizará las distancias laterales y la frontal y realizará la corrección hacia donde corresponda, usando los métodos de la clase Movimientos nombrados anteriormente, para luego continuar con las instrucciones de movimiento recibidas de la resolución del laberinto.

4.7. Implementación de mecanismos de comunicación.

La comunicación entre la cámara web y el computador se realiza de forma alámbrica mediante protocolo USB. Por otro lado, el computador y el robot lo hacen de manera inalámbrica mediante la librería Icommand 0.7 apoyado por la librería rxtxcomm que ayuda a reconocer los puertos seriales usados por el bluetooth al conectarse al NXT.

Figura 4.44 Arquitectura de los mecanismos de comunicación.

La cámara que se utilizó para capturar la imagen del laberinto y que se sitúa por encima de él, para poder así entregar una vista aérea de la totalidad del laberinto es la D-Link DSB-C320, con conectividad USB 1.1 y que posee una resolución de 640x480 pixeles, que será la resolución usada para la captura de la imagen. De manera conjunta, para lograr la conexión entre la cámara y el computador se tuvo que utilizar una extensión de cable USB de 3 metros.

Figura 4.45 Cámara D-Link DSB-C320 y extensión USB.

El dispositivo bluetooth que se utilizó para realizar la comunicación entre el computador y el robot es marca Dongle. Entre sus principales características tenemos que soporta las versiones de bluetooth 2.0 e inferiores, interfaz USB y rango de operación 0 - 100 metros.

Figura 4.46 Dispositivo bluetooth Dongle.

La cámara web captura la imagen del laberinto, se envía mediante protocolo USB al computador para que éste la procese y resuelva el laberinto. Finalmente envía las instrucciones mediante bluetooth al robot para que éste escape del laberinto.

Al implementar los mecanismos de comunicación se produjeron problemas asociados a la incompatibilidad de librerías o programas con Windows 7 de 64 bits. Primero el stack bluetooth funcionaba de manera intermitente, lo cual no aseguraba conectividad computador – bluetooth en todo momento como se deseaba, siendo que se probaron al menos tres: Bluesoleil, Toshiba BT y Widcomm. El problema fue resuelto cambiando el sistema operativo a Windows XP de 32 bits.

Habiendo solucionado el problema anterior, se comenzó a probar con la librería que proveería de comunicación computador – robot. PcControl es una librería que viene incorporada dentro de lejOS y haría dicha tarea, pero, fue imposible lograr la comunicación de manera eficiente. La solución fue ocupar Icommand 0.7.

Una vez solucionado el problema de la comunicación se concluyó que no todas las versiones de firmware disponibles para el NXT se comportan de la misma manera al usar Icommand, encontrando el comportamiento deseado en la versiones 1.03 o superiores.

4.8. Integración.

La integración del software consiste principalmente en juntar todo lo desarrollado en el proyecto Java de Eclipse con lo desarrollado en Matlab. Para ello se toma la decisión de llevarlo todo al proyecto Java y, para eso, la parte creada en Matlab es convertida a ejecutable (.exe) y de esta manera llamado desde el proyecto Java. El trozo de código importante que realiza tal acción es el siguiente:

```
Try{
Runtime.getRuntime().exec("Proyecto2");
}
Catch(Exception e){
System.out.println("Error");
}
```

Proyecto2 es el nombre del ejecutable generado en Matlab y que debe ser cargado al proyecto Java para que con el código anterior logre ejecutarse.

Se puede decir que se logró unificar de manera satisfactoria el procesamiento de imágenes, algoritmo de resolución de laberintos y entrega de instrucciones al Robot Lego Mindstorms.

4.9. Resultados.

Los resultados fueron generados a partir de 10 intentos de escape del laberinto, tomando en cuenta el mejor tiempo, el peor tiempo y el porcentaje de éxito al intentar escapar del laberinto y se pueden apreciar en la tabla que sigue.

Tabla 4.1. Resultados.

Algoritmo	Mejor tiempo	Peor Tiempo	% Éxito
Tremaux	02:31	02:58	80%
Llenador de callejones sin salida	03:03	03:22	70%
Llenador de callejones ciegos	02:28	02:38	80%

Como se puede observar en la tabla los algoritmos con mejor desempeño fueron Tremaux y el Llenador de callejones ciegos con un 80% de éxito, esto quiere decir, que de los 10 intentos logró escapar 8 veces con un tiempo mínimo promedio de 02:30 minutos. Por otro lado, el algoritmo llenador de callejones sin salida tuvo un 70% éxito, esto quiere decir que de los 10 intentos logró escapar 7 veces. El tiempo mínimo para este último algoritmo fue 03:03 minutos, registrándose 30 segundos de diferencia con los otros dos algoritmos de resolución. Dicha diferencia de tiempo se explica por el camino solución que toma el algoritmo callejones sin salida, donde incluye la parte disjunta mientras que el llenador de callejones ciegos no incluye dicha parte del laberinto en su camino solución. En el caso de Tremaux por ser aleatorio sus mejores tiempos lo logró sin incluir la parte disjunta y sus peores tiempos claramente si la incluye en su camino o ruta solución. En la imagen que sigue se puede apreciar la diferencia en la ruta solución al considerar o descartar la parte disjunta.

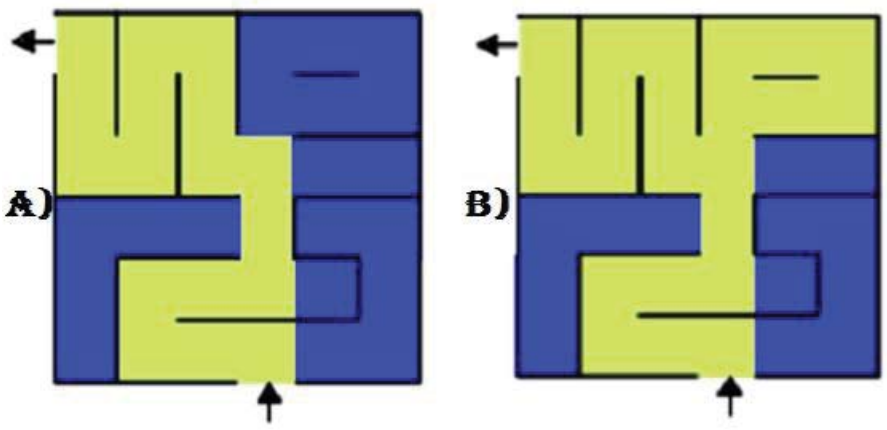


Figura 4.47 Ruta solución. A) Ruta solución sin parte disjunta, B) Ruta solución con parte disjunta.

Con respecto a los resultados que se obtuvieron en el trabajo de título anterior podemos observar un extracto de la tabla 2.2 que se presenta a continuación.

Tabla 4.2. Extracto tabla de resultados 2.1.

Algoritmo	Mejor Tiempo	Peor Tiempo
Algoritmo de Tremaux	00:05:30	00:20:00
Llenador de callejones sin salida	00:05:30	00:09:30
Llenador de callejones ciegos	00:05:30	00:05:30

Al comparar los resultados expresados en ambas tablas podemos apreciar que los mejores tiempos se lograron disminuir 3 minutos aproximadamente, lo que equivale a un 57% en el caso de los algoritmos de Tremaux y llenador de callejones ciegos, mientras que en el caso del algoritmo de llenador de callejones sin salida, el mejor tiempo se disminuye en 02:30 minutos, lo que equivale a un 43%. Ahora, al comparar los resultados de los peores tiempos las mejoras son significativas, ya que en el caso del algoritmo de Tremaux el tiempo se disminuye en 17 minutos, lo que equivale a un 85%, el de callejones sin salida en 6 minutos aproximadamente, con un porcentaje de 65% y el de callejones ciegos en 3 minutos aproximadamente, correspondiente a un 54%.

Finalmente, queda demostrado que se lograron disminuir notoriamente las diferencias entre el mejor tiempo y el peor tiempo del robot al escapar del laberinto, optimizando el desempeño del robot, ya sea recibiendo y ejecutando órdenes y navegando de manera eficiente dentro del laberinto.

Las sugerencias para un trabajo futuro, planteadas en función de los resultados obtenidos y detallados anteriormente son:

- Optimizar la comunicación por bluetooth entre el robot y el computador.
- Agregar al procesamiento de imágenes existente, las etapas de Representación - Descripción y Reconocimiento e Interpretación, ya que con ellas se podría lograr un procesamiento de alto nivel y establecer posibles mejoras.
- Establecer la secuencia más idónea, según rangos de luminosidad, para la captura y posterior procesamiento de la imagen

5. Conclusión.

Al día de hoy, se puede afirmar que se han cumplido los objetivos que se plantearon a inicios del trabajo de título con resultados significativos.

Por un lado, se logró la construcción física tanto del robot como del laberinto. El robot posee los mecanismos suficientes para lograr escapar del laberinto y, a su vez, el laberinto tiene la configuración deseada para realizar las pruebas correspondientes.

Otro punto a resaltar fue durante el desarrollo cuando se intentó cambiar de entorno de desarrollo de Matlab a Octave, en lo que dice relación a captura y procesamiento de imágenes. Octave si bien es muy similar a Matlab en diversos ámbitos, desde el cómo se presenta el entorno al usuario hasta los comandos utilizados para la programación, presentó numerosos problemas de compatibilidad al momento de "especializarse" en un área como era la captura y procesamiento de imágenes. El llamado 99% de compatibilidad entre ambos entornos de desarrollo, claramente se restringe siempre y cuando no se usen las toolbox especializadas.

En cuanto a problemas en el desarrollo del trabajo de título pueden destacarse los que están asociados a incompatibilidad de librerías o programas con Windows 7 de 64 bits. Primero el stack bluetooth funcionaba de manera intermitente, lo cual no aseguraba conectividad computador – bluetooth en todo momento como se deseaba, siendo que se probaron al menos tres: Bluesoleil, Toshiba BT y Widcomm. Segundo, no todas las versiones de firmware se comportan de la misma manera ante una librería. Tercero, de manera obligada se tuvo que usar Icommand 0.7 ante la nula respuesta de PcControl. Finalmente todo esto generó un atraso de alrededor de 2 semanas con respecto a la planificación inicial.

Con respecto a la integración de los módulos, se puede decir que se cumple el objetivo, logrando usar la aplicación completa desde Java, recordando que se usaron dos lenguajes distintos .M por el lado de Matlab y Java por el lado de Eclipse.

Finalmente, podemos decir que los factores que incidieron en mejorar los resultados fueron:

- Se logró establecer tres secuencias de filtros de imágenes eficientes, las cuales facilitaron la aplicación de los algoritmos de resolución de laberintos, debido a que ellas entregaban una imagen adecuada para continuar con el proceso.

- Se logró crear un sistema de navegación para que el robot se desplazara en forma expedita por el laberinto mediante la utilización de tres sensores ultrasónicos que efectuaban lecturas periódicas, evitando así, posibles colisiones del robot contra las paredes del laberinto.

- Cambiar por completo el diseño del robot, donde la elección del tipo de ruedas balloon por sobre las de oruga, ayudó a optimizar su velocidad de desplazamiento arrojando resultados positivos ya que se disminuyeron considerablemente los tiempos de escape del robot del laberinto, hasta en 17 minutos en el caso del algoritmo de Tremaux.

Todo lo anterior se ve reflejado, por ejemplo, en el aumento del porcentaje de éxito a un 80% en el caso del algoritmo de Tremaux y el de llenador de callejones ciegos que fue donde se logró una mejora considerable e importante.

6. Referencias bibliográficas.

[1] J.Abad. (2010, Junio 17). "Modelos de la inteligencia artificial. Interfaces gráficas de usuarios en Matlab". [En línea].
Disponible en: http://www.scribd.com/doc/6539399/Guide.

[2] Asociación de Robótica y Domótica de España (A.R.D.E.). (2010, Abril 18). "Tipos de Robots". [En línea].
Disponible en: http://wiki.webdearde.com/index.php/TIPOS_DE_ROBOTS.

[3] D. Barragán. (2010, Junio 17). "Manual de interfaz gráfica de usuario en Matlab".
[En línea].
Disponible en: http://www.scribd.com/doc/15532859/MANUAL-DEGUI-EN-MATLAB

[4] E. Cuevas. (2010, Mayo 03). "Visión por computador utilizando Matlab y el toolbox de procesamiento digital de imágenes". [En línea].
Disponible en: http://proton.ucting.udg.mx/tutorial/vision/cur sovision.pdf

[5] C. Espinoza. (2010, Agosto 25). "Resolución de un laberinto mediante un Robot Lego ®".
Informe final del proyecto para optar al título profesional de Ingeniero Civil Informático.

[6] R. González y R. Woods (2008), "Digital Image Processing".
Prentice Hall.

[7] C. Larman (2003), "UML y Patrones: Introducción al análisis y diseño orientado a objetos".
Prentice Hall.

[8] Lego Group (2010, Marzo 07). "Productos". [En línea].
Disponible en: http://mindstorms.lego.com/en-us/products/default.aspx

[9] D. Maravall (1994), "Reconocimiento de formas y visión artificial".
Addison – Wesley Iberoamericana S.A.

[10] Oracle (2010, Mayo 17). "Api Java – Package Image v1.4.2". [En línea].
Disponible en: http://download.oracle.com/javase/1.4.2/docs/api/java/awt/image/package-summary.html

[11] J. Paz. (2010, Abril 22). "Generación de imágenes para web con GDI+". [En línea].
Disponible en: http://www2.uacj.mx/Noticias/Publicaciones/generacion_imagenes.htm

[12] W. Pullen. (2010, Marzo 22). "Think Labyrinth!". [En línea].
Disponible en: http://www.astrolog.org/labyrnth.htm

[13] Real Academia Española. (2010, Abril 18). "Diccionario de la lengua española – 22º edición". [En línea].
Disponible en: http://www.rae.es

[14] O. Sánchez (2010, Abril 05). "Modelos, control y sistemas de visión". [En línea].
Disponible en: http://omarsanchez.net/filtdisc.aspx

[15] The Mathworks. "Matlab User Guide". [En línea].
Disponible en: http://www.mathworks.com/help/doc-archives.html

[16] J. Valiente, J. Pérez, G. Andreu y A. Rodas (2010, Agosto 20). "Visión por computador". [En línea].
Disponible en: http://miron.disca.upv.es/vxc/Documentos/vxc-%20Iluminaci%F3n.pdf