E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Prototipo de robot mapeador mediante el uso del algoritmo GraphSLAM

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor del proyecto: Mikel Abaurrea Mateo

Director del proyecto: Miguel Pagola Barrio

Pamplona, 25 de Julio de 2015

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

Me gustaría hacer una serie de agradecimientos a las personas que me han ayudado a hacer que todo esto sea posible:

- A Miguel Pagola Barrio, por inspirarme para realizar este proyecto, dirigirlo y guiarme durante el desarrollo de éste de manera tan eficaz.
- A Edurne Barrenechea Tartas por haber sido un gran apoyo como miembro de nuestro profesorado y por toda la ayuda que nos ha ofrecido durante el transcurso de toda la carrera.
- A mis padres y mi hermano, por haber hecho posible que haya cursado la ingeniería y por todo su apoyo que me han dado en todo momento durante el transcurso de ésta y en el camino que he recorrido para llegar hasta este momento.
- A mi pareja Ana Osés Losarcos por haberme dado el apoyo moral y la confianza que necesitaba durante el desarrollo de todo el proyecto.
- A mis compañeros de promoción, en especial a Iulen Muñoz Murillo, Guillermo Rubio Briñol y Daniel Huarte Latasa por su papel como compañeros de carrera y de amigos durante estos cuatro años.

Índice

1. Introducción....4
1.1 Navegación automática....6
1.2 Propósito del proyecto....7

2. Robótica Móvil....8
2.1 Exploración de entornos desconocidos y generación de mapas....8
2.2 Tipos de robots móviles....9
2.3 Robots móviles con ruedas....9
2.4 El Robot mOway....11
2.4.1 Módulos de expansión Robot del mOway....12
2.4.1.1 Módulo de Radiofrecuencia....13
2.4.1.2 Módulo de cámara....14

3 Algoritmo GraphSLAM....15
3.1 Explicación general....15
3.2 Navegación del robot....16
3.3 Detección de cruces....16
3.3.1 Tipos de cruces....17
3.4 Landmark....20
3.4.1 Identificación de Landmarks....20
3.4.1.1 Diseño de los Landmarks....21
3.4.1.2 Procesamiento de Imagen sobre los landmarks....23
3.5 Toma de decisión....29
3.5.1 Búsqueda en profundidad....30
3.5.2 Representación de los cruces como nodos....31
3.5.3 Aplicación de la búsqueda en profundidad para el mapeo completo....32
3.5.4 Extracción de caminos mediante búsqueda en profundidad....34
3.5.4.1 Inconvenientes de utilizar la búsqueda en profundidad....36

4. Mapeo visual....37
4.1 Odometría....37
4.2 Odometría en nuestro proyecto....38
4.3 Subsanación de errores sistemáticos....39
4.4 Mapeo de obstáculos....41

5. Arquitectura del Software....43
5.1 Módulo de navegación y mapeo....43
5.2 Módulo de Visión Artificial....43
5.3 Diagramas de transición de estados....44
5.4 Comunicación entre los módulos de Navegación/Mapeo y Visión Artificial....46

6. Resultados experimentales....47
6.1 Diferentes pruebas....47
6.2 Análisis de los resultados obtenidos....49

7. Problemas con el proyecto....49

8. Conclusiones y mejoras....52

9. Bibliografía....53
9.1 Bibliografía básica....53
9.2 Bibliografía on-line....53

1. Introducción

Hoy en día la Inteligencia Artificial es un campo cuyo papel en la vida cotidiana, en el entorno empresarial y en el ámbito científico y de investigación, tiene un peso extremadamente importante.

Disponemos de todo tipo de aparatos y herramientas automatizadas que nos facilitan muchísimas tareas. Éstos, precisan de una inteligencia que les permita comprender su entorno y la composición de los trabajos que tienen llevar a cabo. Los ejemplos de dichos aparatos abarcan desde sistemas de mantenimiento de hogares inteligentes (el campo de la domótica), vehículos capaces de navegar por ellos mismos, y electrodomésticos autónomos como aspiradores que se encargan de limpiar nuestras casas automáticamente.

Este proyecto está centrado en la navegación automática de un robot en un entorno desconocido para él. Su objetivo será generar un mapa virtual de la instancia en la que se encuentre, que en este caso será un laberinto, y que una vez realizado sea capaz de encontrar el camino más corto desde un punto del laberinto hasta otro lugar de éste. Para ello, tendremos que otorgarle de la inteligencia necesaria para que pueda realizar el procesamiento de imagen que recibirá por su cámara, para que junto con el resto de sus sensores sepa capaz de situarse en el laberinto a la vez que lo va mapeando. Una vez tenga generado el mapa, también deberá ser capaz de viajar entre dos puntos, que elegiremos que sean el punto de salida del robot y el final del laberinto.

El actor principal será el robot mOway, un robot programable con varios sensores de proximidad y con la capacidad de poderle añadir módulos adicionales, como por ejemplo, una videocámara. El lenguaje de programación que vamos a utilizar para desarrollar el proyecto es Python, ya que es el lenguaje con el que trabaja nuestro robot. Además, disponemos de librerías gratuitas de procesamiento de imagen tales como openCV, la cual utilizaremos en este proyecto para realizar todo lo referente a la parte de visión artificial.

Nuestro entorno de pruebas será un laberinto construido con cartón-pluma. El diseño del laberinto estará planteado de tal forma que presentará todos los posibles cruces e intersecciones que se pueden encontrar en un laberinto normalmente con el fin de poner a prueba a nuestro robot.

El algoritmo principal de navegación del robot, se basa en el algoritmo de GraphSLAM (Simoultaneos Localization And Mapping) que como su nombre indica trata de localizar el robot en el mapa e ir generándolo simultáneamente apoyándose en el uso de grafos y árboles.

1.1 Navegación automática

Actualmente existen todo tipo de robots cuyo propósito principal es el de desplazarse de manera autónoma dentro de un espacio específico.

Un claro y conocido ejemplo de esta tecnología es el vehículo Google Car, un automóvil que es capaz de conducir sin necesidad de un tripulante desde un punto a otro, respetando en todo momento señalizaciones de tráfico y evitando posibles peligros. Esto lo consigue localizándose en una serie de mapas (obtenidos utilizando la tecnología de Google Maps). Gracias a una serie de sensores y cámaras con los que está equipado, es capaz de situarse en los dichos mapas para conducir a través de ellos con el fin de llegar al destino solicitado.

Figura 1.1: Vehículo Google Car

Como he explicado anteriormente, estos robots precisan de unos mapas para poder desplazarse por su entorno, pero existen otro tipo de máquinas cuya función es ciertamente la de generar estos mapas por los que luego podrán navegar de manera automática.

1.2 Propósito del proyecto

El objetivo que se desea conseguir es que el robot mOway sea capaz de mapear virtualmente una instancia desconocida para él, es este caso un laberinto, y que una vez haya terminado, sepa llegar de cualquier punto del laberinto a otro distinto.

Este proyecto es un buen ejemplo para aplicar principios teóricos básicos sobre la inteligencia artificial en el ámbito de la robótica en un entorno de mundo real, tales como la visión artificial o los algoritmos de búsqueda. Como se ha dicho antes, es un prototipo que propone una solución a un problema establecido que en este caso el de mapear una instancia desconocida para el robot y navegar por ella. Esta idea se podría aplicar a futuros proyectos más concretos adecuando las partes necesarias para que se ajusten a los posibles requisitos más específicos de ese caso.

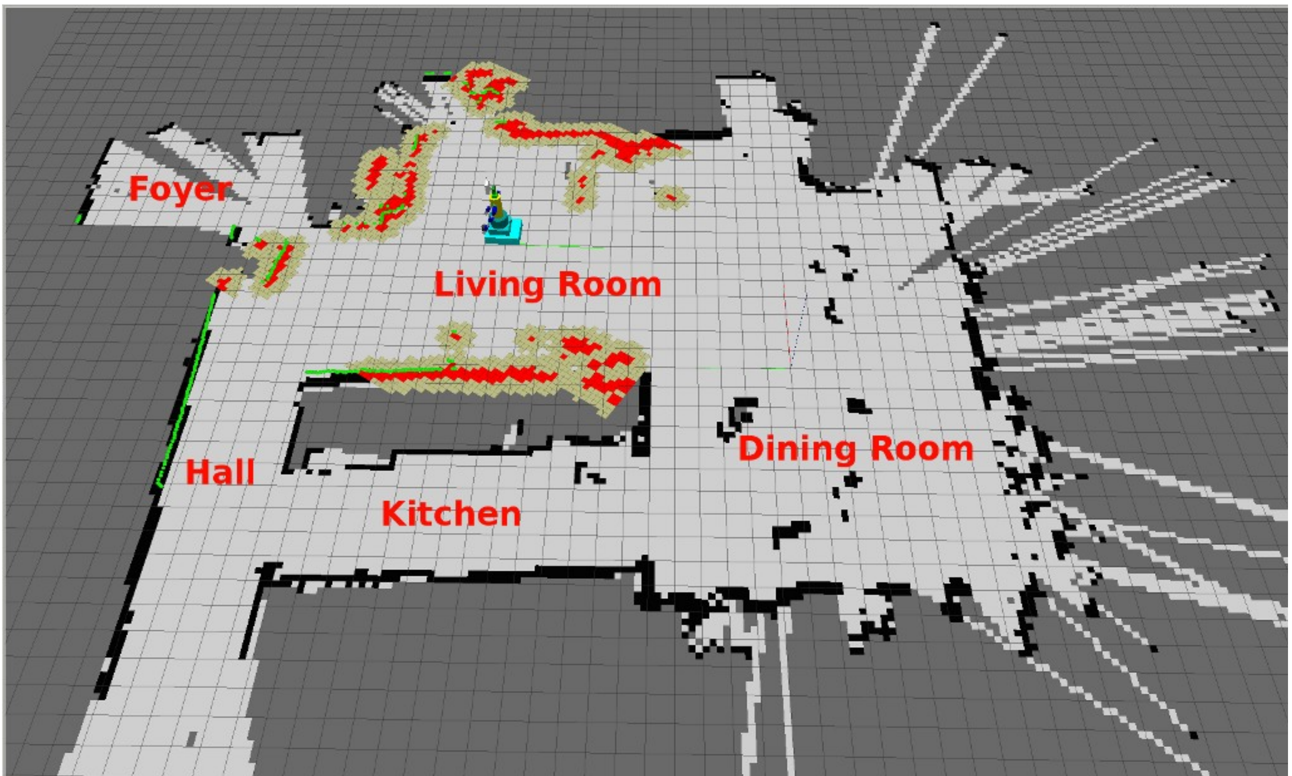


Figura 1.2: Mapeo de una casa

2. Robótica Móvil

2.1 Exploración de entornos desconocidos y generación de mapas

Uno de los temas más activos actualmente dentro del mundo de la robótica es el referente al mapeo y la exploración de entornos desconocidos. Las soluciones a este problema están en continua evolución, desarrollando nuevas metodologías para solucionarlo de manera más precisa y así poder obtener mapas de mayor calidad. Esta necesidad surge de diversas situaciones en las que es necesario cartografiar un espacio por los que por algún motivo es preciso dejarlo en manos de un robot, ya sea por motivos de seguridad en los que sería peligroso para un ser humano acceder a ciertos lugares para explorarlos, o simplemente por lograr una manera más rápida y precisa de llevar a cabo esta tarea.

En la minería por ejemplo, se utilizan robots los cuales recorren y generan mapas detallados de las galerías que exploran para que así los ingenieros mineros sepan las características de las vetas y cuevas en las que están trabajando y puedan diseñar sus planes de explotación de acuerdo a los resultados obtenidos.

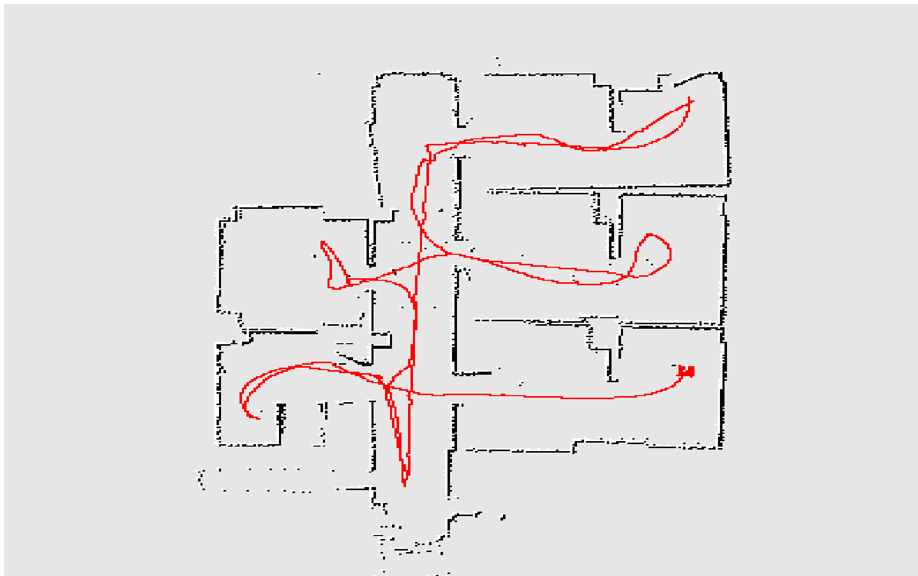


Figura 2.1: Cartografía generada por un robot mapeador

Otro ejemplo más cotidiano es el de los robots aspiradores que cada vez más frecuentemente podemos encontrar en los hogares de las personas. Estos electrodomésticos precisan de un mapa de las habitaciones por las cuales deberán moverse para así poder limpiarlas, por lo que su primera tarea es cartografiar el domicilio para que puedan utilizar el mapa que necesitarán cuando tengan que limpiar los diferentes lugares de la casa.

2.2 Tipos de robots móviles

Puesto que necesitamos que el robot que vamos a utilizar se desplace de alguna manera, explicaremos los diferentes tipos básicos de robots móviles que existen, ya que podemos encontrar una infinidad de mecanismos para el desplazamiento desarrollados para diversos casos.

Los tipos más comunes son:

- Robots con ruedas
- Robots con cintas de oruga
- Robots artrópodos/con apéndices

Ya que nuestro robot va a ser un prototipo, utilizaremos el sistema más sencillo y en un futuro, si se quisiera ampliar el proyecto o trasladarlo para utilizarlo en alguna circunstancia más concreta en la que el entorno precise de otro tipo de desplazamiento, como por ejemplo un terreno abrupto funciona mejor con sistemas de cintas de oruga, bastaría con simplemente modificar la funcionalidad de navegación para que se adecúe a su nuevo modo de desplazamiento.

2.3 Robots móviles con ruedas

Dentro de los robots con ruedas, podemos encontrar distintas variantes en función de la disposición de las ruedas, pero aquí nos centraremos en los robots móviles con ruedas de tipo diferencial ya que esa es la disposición que utiliza nuestro robot mOway. Dispone de dos ruedas motrices y un pivote de apoyo.

En esta configuración, los elementos están colocados de la siguiente manera:

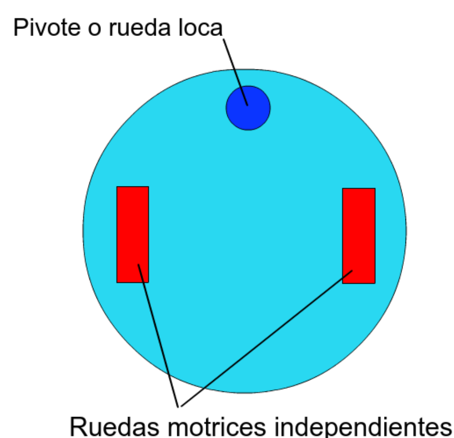


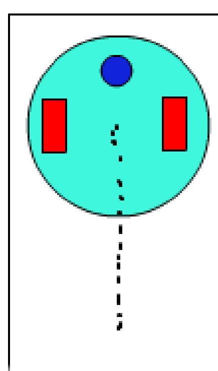
Figura 2.2: Disposición ruedas robot diferencial

Las dos ruedas laterales están alineadas con el eje central del robot. Estas ruedas disponen de motores independientes lo cual le permite a cada una de ellas moverse a velocidades diferentes respecto de la otra, con lo que se consigue realizar los distintos tipos de movimientos. Para avanzar en línea recta, ambas ruedas girarán a la misma velocidad y para hacer giros, una más rápido que la otra. El robot mOway dispone de unos comandos específicos en los que le podemos concretar distancias de avance, grados de giro, etc, de los cuales ya hablaremos en detalle más adelante.

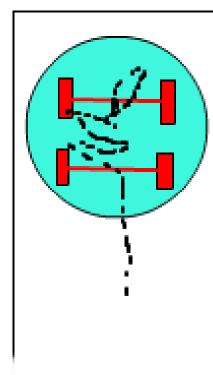
El resto del robot descansa sobre el pivote o “rueda loca” situado en la parte delantera del aparato. Éste se desliza sobre el suelo sin modificar la trayectoria establecida por las ruedas traseras y a la vez aporta el equilibrio necesario para que nuestro robot pueda “sostenerse sobre dos ruedas”.

La ventaja de este tipo de disposición de ruedas es que es un sistema holonómico, es decir, le es posible cambiar su dirección en cualquier momento sin necesidad de rotar previamente. Esto nos permite hacer giros sobre el propio eje del robot, lo cual simplifica la navegación del robot de manera considerable. Es mucho mas sencillo el girar sobre nuestro propio eje cuando sea necesario y reemprender un rumbo recto en comparación a tener que realizar trayectorias parabólicas

Un claro de ejemplo de esto es la situación en la que el robot se encuentra con un callejón sin salida. Si utilizásemos un robot no holonómico, a la hora de dar media vuelta tendríamos que realizar una serie de maniobras que requerirían de un tiempo extra además de una serie de requisitos en el entorno como por ejemplo que el espacio del callejón sea el suficiente como para que el robot pueda maniobrar hasta conseguir darse la vuelta por completo. Sin embargo, con nuestro robot que sí que es holonómico, la única operación necesaria antes de volver a emprender el rumbo es el de rotar sobre su propio eje 180º.



Robot Holonómico:
Paso 1: Girar 180º sobre su eje
Paso 2: Avanzar



Robot no Holonómico:
Paso 1: Girar eje delantero a la izquierda y avanzar
Paso 2: Girar eje delantero a la derecha y retroceder
Paso 3: Girar eje delantero a la izquierda y avanzar
Paso 4: Girar eje delantero a la derecha y retroceder
Paso 5: Enderezar eje y avanzar

Figura 2.3: Diferencia de trayectoria entre un sistema Holonómico y uno que no lo es

2.4 El Robot mOway

El robot mOway es un robot educacional y de investigación, diseñado y producido por la compañía *Moway Robotics* localizada en Erandio, Bizkaya. El objetivo principal de mOway es ser una herramienta útil tanto para quienes se introducen por primera vez en el mundo de la robótica como para quienes ya tienen experiencia y desean realizar aplicaciones complejas de robótica colaborativa.

Figura 2.4: Robot mOway con el módulo de cámara web instalado.

Moway está gobernado por el microprocesador PIC18F86J50. Dispone de varios sensores de distancia infrarrojos, así como uno sonoro y lumínico a parte de otros elementos como diversas luces leds o un acelerómetro. Para su desplazamiento utiliza dos ruedas con motores independientes controladas por I2C. Además también posee un bus de expansión SPI/I2C con el que se le pueden añadir complementos, que en nuestro caso serán la videocámara y el módulo de comunicación por radiofrecuencia.

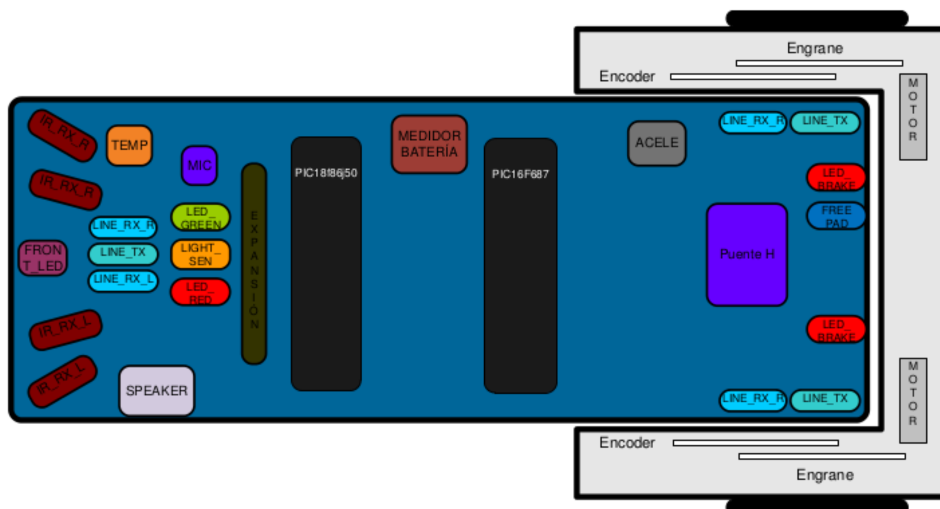


Figura 2.5: Diagrama de los elementos del robot mOway

En la siguiente tabla se especifican sus características principales:

Nombre componente	**Descripción del elemento**
Procesador	Microcontrolador PIC18F86J50 trabajando a 4Mhz
Memoria	RAM: 8KB FLASH: 144KB
Motores	2 motores tipo stepper independientes, velocidad máxima del robot: 10cm/s
Sensores IR	4 sensores infrarrojos con un alcance máximo de 8 centímetros
Sensor optoreflectivo	1 sensor infrarrojos inferior para el seguimiento de líneas
LEDs	1 LED frontal de iluminación, 2 LEDs superiores verde y rojo, 2 LEDs rojos traseros
Sensor intensidad de luz	Sensor APDS-9002 lumínico direccional, mide varianzas en la intensidad de la luz
Micrófono	Micrófono CMC-5042PF-AC de CUI INC operativo desde 100 Hz hasta 20 KHz.
Altavoz	Altavoz CMT-1102 de CUI INC capaz de reproducir tonos desde 250 Hz hasta 5,6 KHz en intervalos pares de 100 ms.
BUS de expansión SPI/IC2	BUS de expansión para módulos extra
Batería	Batería LI-PO con autonomía máxima de 2 horas

Tabla 1. Características principales mOway

2.4.1 Módulos de expansión Robot del mOway

Como hemos mencionado anteriormente, el robot mOway dispone de un BUS de expansión para añadir módulos de circuitos para ampliar las posibilidades que ofrece. Para este proyecto vamos a utilizar dos módulos al mismo tiempo, ya que uno de estos tiene otro puerto de expansión para poder conectarle otro más mientras se utiliza el primero.

Los módulos que vamos a utilizar son:

- Módulo de comunicación por radiofrecuencia.
- Módulo de cámara.

2.4.1.1 Módulo de Radiofrecuencia

El módulo BZI-RF2GH4 permite al robot mOway comunicarse de manera inalámbrica con otros robots mOway o con un PC mediante el RFUSB. El RFUSB es un receptor USB que se conecta al ordenador para que el robot pueda intercambiar datos con él.

Las principales características del BZI-RF2GH4 son:

- Bajo consumo.
- Frecuencia de trabajo de 2.4GHz.
- Potencia de emisión entre -18 y 0 dBm.
- Velocidad de transmisión entre 1 y 2 Mbps.
- 128 canales de transmisión seleccionables por el bus SPI

Figura 2.5: Módulos de radiofrecuencia

La razón de utilizar este complemento en el proyecto es que el robot tiene un procesador poco potente y una memoria bastante escasa, por lo que dejaremos que un PC actúe a manera de cerebro del mOway. El robot se encargará de obtener los datos del entorno a través de sus sensores para que el ordenador los procese y le envíe en respuesta las ordenes que el robot deberá realizar.

2.4.1.2 Módulo de cámara

El módulo de cámara permite captar en el ordenador lo que el robot está "viendo" en cada momento. Las imágenes enviadas por la cámara son recibidas en el PC por radiofrecuencia a través del receptor de vídeo mOway Camera Board.

Las principales características del módulo de cámara son:

- Receptor de vídeo con interfaz USB 2.0
- Decodificador Multi-Standar
- Filtro anti-aliasing
- Vídeo en color RGB
- 4 Canales de emisión/recepción (Debe ser el mismo que el del módulo de radiofrecuencia si se utilizan simultáneamente)
- Varias resoluciones de captura.

Figura 2.6: Receptor de vídeo mOway Camera Board

Gracias a este módulo, podremos procesar las imágenes recibidas por la cámara para que mediante técnicas de visión artificial podamos obtener información útil del entorno que ayudarán a la correcta navegación del robot por el laberinto.

En este proyecto en concreto, se utilizarán, para mediante el reconocimiento de una serie de landmarks repartidos por el laberinto, el robot sepa si ya había visitado o no un cruce en concreto.

3 Algoritmo GraphSLAM

3.1 Explicación general

Como indica su nombre, el algoritmo GraphSLAM es un algoritmo que mapea y se localiza en una instancia simultáneamente (Simultaneous Localization And Mapping) y que además utiliza grafos para llevar acabo este cometido.

En nuestro caso, el robot mOway deberá recorrer un laberinto completamente desconocido para él. Mientras avance por él, al mismo tiempo tendrá que ir generando un mapa virtual para que una vez haya terminado de recorrer todo el laberinto, pueda utilizarlo para saber llegar de un punto a otro sin perderse.

Lo primero en que debemos enseñarle al robot es el saber moverse por el laberinto. Para ello tendrá que ser capaz de recorrer los pasillos estando lo más centrado posible en éstos. También deberá girar a la izquierda o a la derecha en los pasillos si así se requiere. El siguiente paso es que una vez alcance una intersección de pasillos, gracias al uso de sus sensores de proximidad sepa reconocer el tipo de cruce que es en cada caso. Una vez detecte el tipo de cruce, tendrá que comprobar si ya lo había visitado anteriormente. Para ello, tendrá que observar el landmark que habrá en cada cruce, los cuales tendrán asignada una identificación única. Si al llegar a un cruce el robot no ve ningún landmark, gracias a que conoce el tipo de cruce en el que se encuentra, sabrá colocarse para poder ver el landmark. Cuando ya sepa si había visitado o no la intersección, deberá tomar la correspondiente decisión para continuar recorriendo el resto del laberinto. Repetirá este proceso hasta que halla visitado todos los lugares del laberinto.

Todo esto tendrá que realizarlo mientras que simultáneamente va generando un mapa del recorrido que va completando sobre el laberinto. En este proceso generará dos mapas diferentes. El primero será un mapa lógico en forma de árbol donde cada nodo representará cada una de las intersecciones del laberinto y que utilizará para saber si ha recorrido todo el laberinto o no recorriéndolo con una búsqueda en profundidad.

El otro mapa, será una representación visual del laberinto en la pantalla del ordenador. Para construir este mapa, en cada muestreo del robot, éste medirá y evaluará los datos recibidos por sus sensores de proximidad. En una matriz donde todos sus elementos inicialmente valdrán cero, utilizando la posición actual del robot, las mediciones de sus sensores y su orientación, determinará la distancia de los obstáculos y cambiará los valores correspondientes de esa matriz al valor 1 en el caso de que haya obstáculo en esa posición. Una función irá recorriendo toda esta matriz y en el

caso de que un elemento de esta valga 1, pintará un recuadro verde en la pantalla en la posición que esté asignada a ese elemento para que así se vaya formando visualmente el mapa del laberinto.

Al terminar de recorrer todo el laberinto, el robot emprenderá un recorrido hasta el punto que se le haya indicado y completará el trayecto sin perderse.

3.2 Navegación del robot

La parte de navegación en la lógica del robot es la encargada en gobernar el correcto rumbo de este a lo largo de su recorrido por el laberinto.

Nuestro robot deberá mantenerse siempre lo más centrado posible entre las paredes de los pasillos por los que valla avanzando para llegar en una posición óptima a las intersecciones para que le resulte más sencillo identificar el tipo de cruce que es. Para ello, iremos corrigiendo el rumbo en función de los datos recibidos de los sensores laterales del robot.

Como es lógico, a mayor medida de un sensor, la reorientación del mOway deberá ser mayor ya que el error del rumbo es directamente proporcional a la cercanía a una pared. Los ángulos que debe girar para corregir su rumbo vienen dados por las siguientes expresiones, tanto como giros para la izquierda como para la derecha:

$$\Theta l(Sr) = Cte * Sr \qquad \Theta r(Sl) = -Cte * Sl$$

donde ***Sr*** y ***Sl*** son los estímulos recibidos por los sensores derechos e izquierdos respectivamente. ***Cte*** es una constante que viene determinada por la relación entre la anchura media de los pasillos y el tamaño propio del robot. En nuestro caso hemos obtenido experimentalmente que el valor 0.25 es el que mejor funciona. Con estas operaciones nuestro robot tratará de centrarse constantemente en el pasillo que se encuentre. Continuará avanzando hasta que se encuentre con una intersección, donde tendrá que conseguir el siguiente objetivo, detectar el tipo de cruce que es.

3.3 Detección de cruces

Cuando el robot alcance una intersección lo primero que tendrá que hacer es detectar el tipo de cruce que es ya que cada uno de estos cruces implica una serie de posibles decisiones que el robot podrá realizar para continuar su camino. Para ello, utilizando los sensores de proximidad, buscaremos aperturas en las paredes para así poder catalogar la intersección. Dentro del laberinto, hemos catalogado una serie de de cruces que detallaremos a continuación.

3.3.1 Tipos de cruces

- Tipo 0:

  Un pasillo sin salida, camino cortado. En este caso el robot deberá rotar sobre sí mismo 180º y actualizar su orientación. A continuación reemprenderá el viaje. Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Frontal derecho: Mayor o igual que 80
  - Frontal izquierdo: Mayor o igual que 80
  - Lateral derecho: Mayor que 60
  - Lateral Izquierdo: Mayor que 60

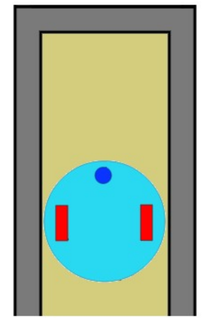


Figura 3.1: Cruce tipo 0

- Tipo 1:

  Giro a la derecha. En este caso, el robot deberá rotar sobre sí mismo 90º hacia la derecha y actualizar su orientación. Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Frontal derecho: Mayor o igual que 70
  - Frontal izquierdo: Mayor o igual que 70
  - Lateral derecho: Menor que 55
  - Lateral Izquierdo: Mayor que 45

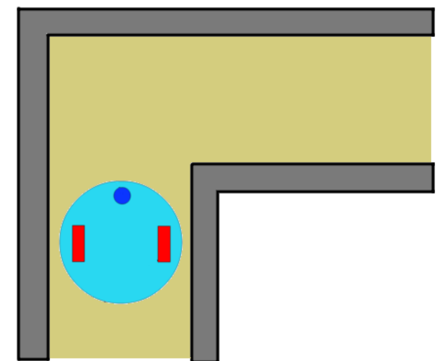


Figura 3.2: Cruce tipo 1

- Tipo 2:

  Giro a la izquierda. En este caso, el robot deberá rotar sobre sí mismo 90º hacia la izquierda y actualizar su orientación. Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Frontal derecho: Mayor o igual que 70
  - Frontal izquierdo: Mayor o igual que 70
  - Lateral derecho: Mayor que 45
  - Lateral Izquierdo: Menor que 55

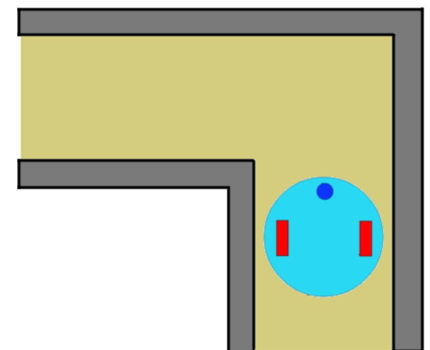


Figura 3.3: Cruce tipo 2

- Tipo 3:

  Intersección "T". En este caso, el robot podrá girar hacia la izquierda o a la derecha. Actualizará su orientación 90º a la derecha o a la izquierda acorde a la decisión que haya tomado. Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Lateral derecho: Menor que 15
  - Lateral Izquierdo: Menor que 15

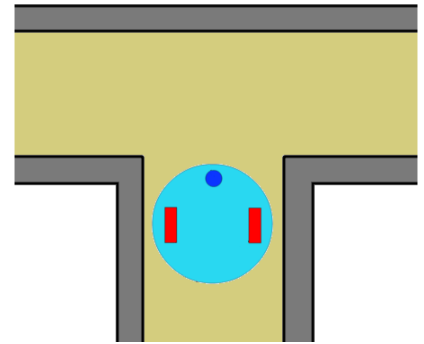


Figura 3.4: Cruce tipo 3

  Una vez obtenidas estas lecturas el robot avanzará hasta el centro de la intersección y a continuación se deben dar las siguientes condiciones en el resto de sensores:

  - Frontal derecho: Entre 15 y 69
  - Frontal izquierdo: Entre 15 y 69

- Tipo 4:

  Intersección "izquierda-delante". En este caso primero, el robot podrá girar hacia la izquierda o continuar recto. Actualizará su orientación 90º a la la izquierda si efectúa un giro y la mantendrá igual si continúa recto. Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Frontal derecho: Menor o igual que 5
  - Frontal izquierdo: Igual a 0
  - Lateral derecho: Entre 20 y 75
  - Lateral Izquierdo: Igual a 0

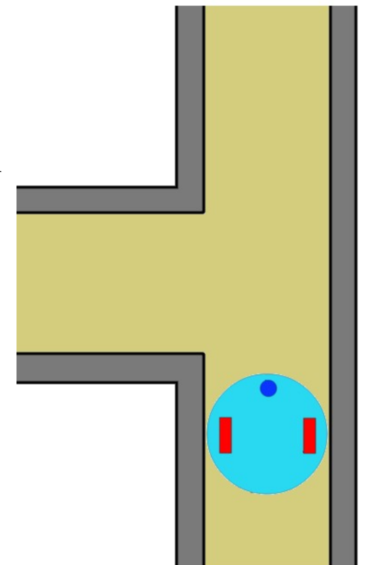


Figura 3.5: Cruce tipo 4

- Tipo 5:

  Intersección “derecha-delante”. En este caso, el robot podrá girar hacia la derecha o continuar recto. Actualizará su orientación 90º a la la derecha si efectúa un giro y la mantendrá igual si continúa recto. Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Frontal derecho: Igual a 0
  - Frontal izquierdo: Menor o igual que 5
  - Lateral derecho: Igual a 0
  - Lateral Izquierdo: Entre 20 y 75

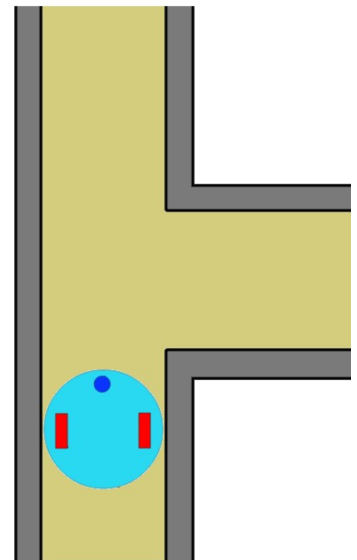


Figura 3.6: Cruce tipo 5

- Tipo 6:

  Intersección “cruz”. En este caso, el robot podrá girar hacia la izquierda, a la derecha o continuar recto. Actualizará su orientación 90º a la derecha o a la izquierda si efectúa uno de estos giros y la mantendrá igual si continúa recto Este tipo de cruce se detecta cuando los sensores devuelven las siguientes mediciones:

  - Lateral derecho: Menor que 5
  - Lateral Izquierdo: Menor que 5

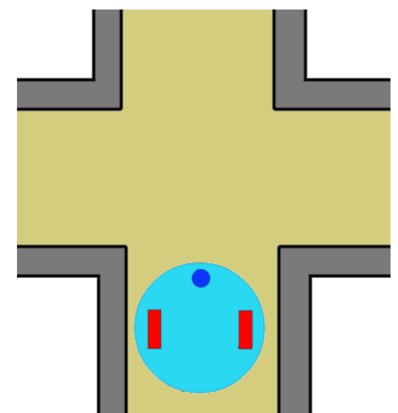


Figura 3.7: Cruce tipo 6

  Una vez obtenidas estas lecturas el robot avanzará hasta el centro de la intersección y a continuación se deben dar las siguientes condiciones en el resto de sensores:

  - Frontal derecho: Menor que 2
  - Frontal izquierdo: Menor que 2

Una vez el robot conozca el tipo de cruce en el que se encuentra, lo siguiente que deberá saber es si ya había visitado esa intersección con anterioridad o es la primera vez que la alcanza. Para lograr esto, reconocerá la ID asignada al landmark de ese cruce de la manera que detallaremos a continuación.

3.4 Landmark

Un landmark es un elemento del entorno físico en el que se encuentra el robot que es fácilmente identificable. En algunos casos se utilizan algoritmos para la extracción y creación de landmarks a partir de objetos de mundo en el que se desenvuelve el robot. En otros casos, como es el de este proyecto, se han diseñado previamente unos landmarks los cuales enseñaremos a reconocer al robot para que éste les asigne una identificación única. Con esto, podrá evitar un gran problema que se le presentaba, el de saber si al llegar a un cruce, ya lo había visitado antes o en cambio es la primera vez que llega a él.

3.4.1 Identificación de Landmarks

En este momento el robot ya conocerá el tipo de cruce en el que se encuentra. Lo siguiente que tiene que hacer es saber si ya había visitado esa intersección o no y para ello deberá identificar el landmark asignado a ese cruce. Hemos decidido otorgar un landmark único únicamente a los cruces cuyo tipo tenga la posibilidad de decidir entre más de un camino (es decir a los de tipo 3, 4, 5 y 6). Esta decisión se toma basándonos en que solo necesitará saber si había visitado un cruce anteriormente en el caso de que pueda tomar algún tipo de decisión.

Ahora el robot se encontrará en una de estas dos situaciones: Que el landmark esté dentro del campo de visión de la cámara (esto ocurre en la intersección de tipo 3) o el caso contrario, que no lo esté (esto sucede en las intersecciones de tipo 4, 5 y 6) ya que el landmark estará pegado en la pared de la intersección. Por lo tanto, si el robot se encuentra en un cruce de tipo 3 ya podrá obtener la identificación del landmark sin necesidad de realizar ningún movimiento extra, sin embargo, en el resto de casos deberá enfocar el landmark para poder proceder a extraer su ID.

Para conseguir esto, hemos desarrollado una función que introduciéndole el tipo de cruce en el que se encuentra el robot, le ordena una serie de movimientos necesarios para que el landmark acabe dentro del campo de visión de la cámara. A continuación ponemos el pseudocódigo de esta función:

```
function focusLandmark(int intersectionType, robot):
    if (intersectionType == 4):
        //Colocamos el robot en el centro de la intersección
        robot.avanzar(100);
        //Encaramos el robot hacia la pared girando 90º a la derecha
        robot.girarDerecha(90);
        //Retrocedemos 10 centímetros para enfocar el Landmark
```

```
        robot.avanzarAtras(100):
        return;
    if (intersectionType == 5):
        //Colocamos el robot en el centro de la intersección
        robot.avanzar(100);
        //Encaramos el robot hacia la pared girando 90º a la izquierda
        robot.girarIzquierda(90);
        //Retrocedemos 10 centímetros para enfocar el Landmark
        robot.avanzarAtras(100):
        return;
```

Como se puede apreciar en el fragmento de pseudocódigo, no están contemplados los cruces de tipo 6. La razón de esto es que, por la manera que hemos decidido colocar los landmarks en las paredes, en las intersecciones de este tipo no disponemos de una superficie donde pegarla. Por este motivo, como simplificación, hemos decidido que en nuestros laberintos únicamente pueda haber una intersección de este tipo, a la cual le asignaremos automáticamente una ID única sin necesidad de leer un landmark. También cabe mencionar que las intersecciones de tipo 0 o caminos cortados asignará una ID 0 a todos ellos, ya que son irrelevantes a la hora de alcanzar nuevos puntos.

3.4.1.1 Diseño de los Landmarks

Como hemos explicado anteriormente, vamos a diseñar unos Landmarks para que el robot pueda identificarlos y asignarle una ID a cada cruce ya que necesitará saber si lo había alcanzado anteriormente o no.

Vamos a hacer que todos los landmarks tengan dos elementos en común, ya que ayudará al robot a saber que estaba viendo un landmark y que tiene que proceder a obtener información de él. Estos elementos serán dos cruces negras colocada una sobre otra en un espacio vertical a cierta distancia. A continuación se muestra la plantilla básica:

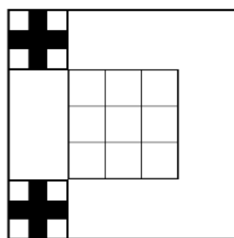


Figura 3.8
Plantilla Landmark

Codificación de la información

Una vez el robot sepa que está viendo un landmark, se fijará en el recuadro del centro. Este recuadro estará dividido en 3x3 casillas, las cuales podrán estar pintadas o no de color negro. La idea es, que el robot sepa reconocer si cada una de estas 9 está rellena, y a partir de esa información construirá una matriz de 3x3 donde sus elementos serán 0 si la casilla era blanca y 1 si era negra. Una vez esté completa la matriz, gracias a una función que programaremos se obtendrá un número en binario a partir de los elementos de la matriz que pasaremos a entero y este será el ID asignado a ese landmark en concreto.

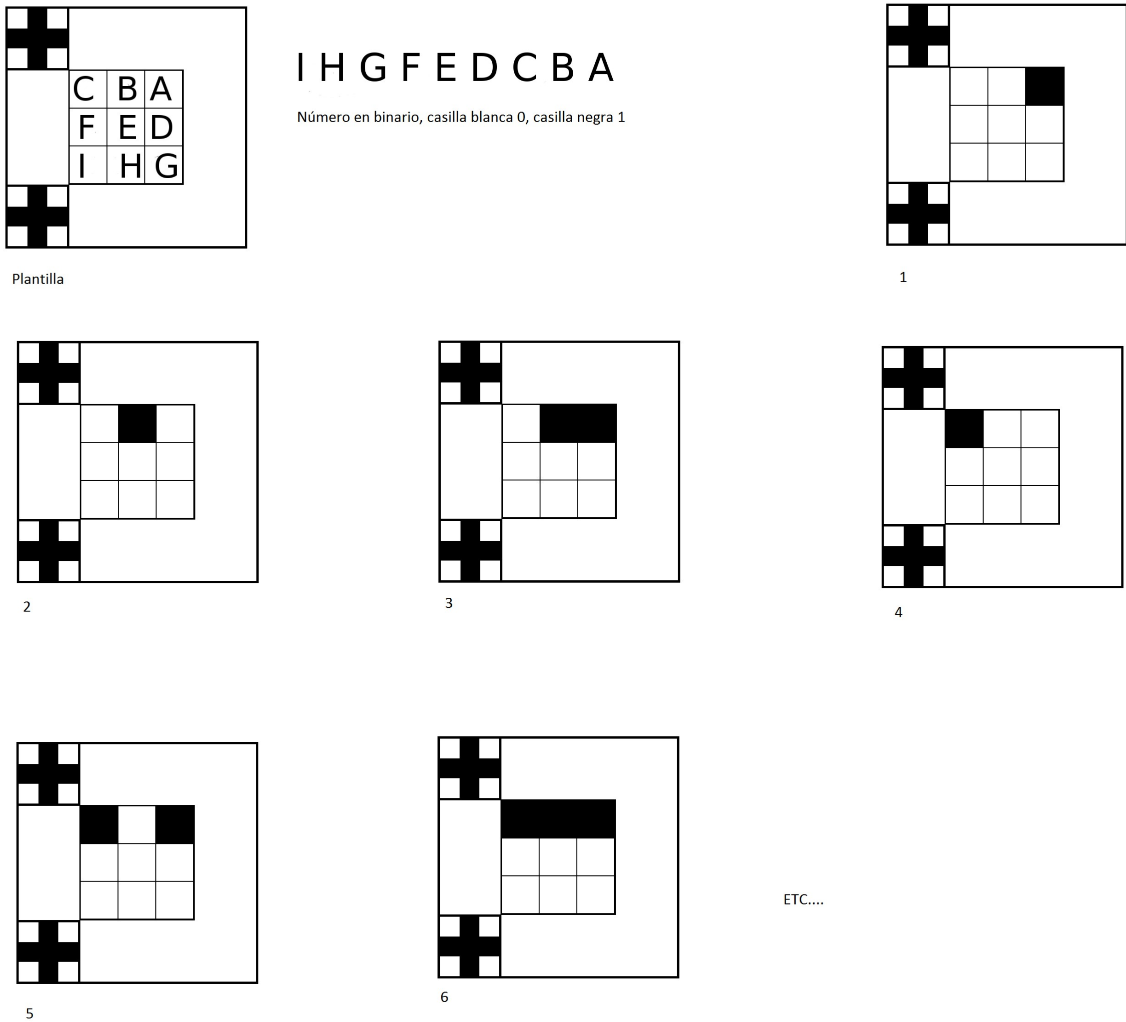


Figura 3.9: Diseño de los landmarks

Utilizando este método tenemos muchas ventajas frente al prediseñar ciertos patrones que corresponderían a ciertas identificaciones. Las más notorias son que el robot no tendrá que poseer conocimiento previo de sobre cómo son los patrones prediseñados y a que ID están ligados cada uno, en términos de programación no necesitaremos tantas máscaras como patrones hubiésemos predefinido (además habría que multiplicarlo por los diferentes tamaños de ventana), si no que sin necesidad de ninguna podemos obtener hasta 2^8 posibles combinaciones (256) de IDs.

3.4.1.2 Procesamiento de Imagen sobre los landmarks

Una vez diseñados los landmarks y definida la codificación de la información en estos, el siguiente paso es el de enseñar al robot a extraer dicha información cuando vea un landmark a través de su cámara. Para conseguir esto, utilizaremos una serie de técnicas de procesamiento de imagen y visión artificial con el fin de que el robot sea capaz de obtener el ID de cada una de las landmarks que se encuentre y que sigan el diseño descrito anteriormente.

Librería OpenCv

OpenCv (Open source Computer Vision) es una librería enfocada al procesamiento de imagen de uso libre tanto para uso académico como para uso comercial. Está implementada para varios lenguajes de programación así como C, C++, Java y Python. Además es multi-plataforma y se puede utilizar en Windows, Linux, Mac OS, Android e iOs. Está centrada en el alto rendimiento y en las aplicaciones de tiempo real, por lo que es perfecta para utilizar en este proyecto para trabajar con todo lo referente a procesamiento de imagen y visión artificial. En el apartado de bibliografía disponemos de un enlace a toda su documentación.

Segmentación de la imagen

La segmentación de imagen en el ámbito de la visión artificial, es el proceso encargado en dividir y catalogar los diferentes píxeles que componen una imagen en clases diferentes con en objetivo de simplificar la imagen entrante y que así sea más sencillo extraer la información de ésta.

Uno de los principales usos de la segmentación es el de separar los objetos que aparecen en la imagen, con respecto al fondo. En nuestro caso será este el objetivo que buscaremos con la segmentación, el de obtener los elementos del landmark y separarlos del fondo, ya que son estos los que aportan información y el fondo es irrelevante para nosotros.

Al principio, las imágenes recibidas de los landmarks del robot serán algo parecido a esto:

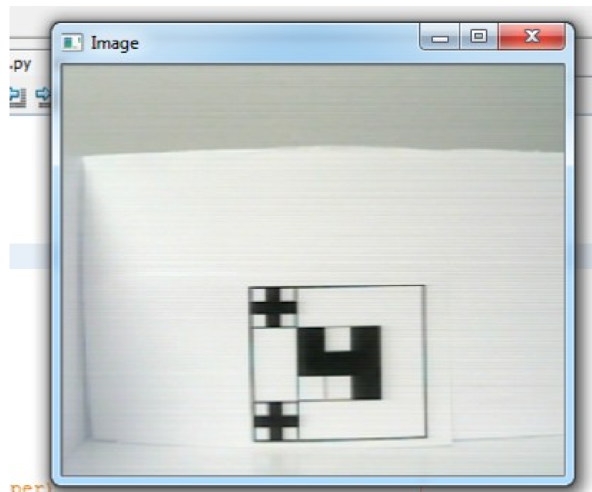


Figura 3.10: Imagen recibida por la cámara

El primer paso que debemos realizar es el de pasar la imagen que el robot percibe por la cámara a escala de grises, ya que en este caso el color no nos aporta información alguna y vamos a hacer una segmentación de blancos y negros.

Segmentación de tipo Otsu

Este método de segmentación se basa en la varianza de los niveles de gris de la imagen. El método de Otsu calcula el valor umbral de tal manera que la varianza dentro de cada segmento sea lo más pequeña posible y al mismo tiempo la varianza sea lo más grande posible entre segmentos diferentes. Para ello se calcula el cociente entre ambas varianzas y se busca un valor umbral para el que este cociente sea máximo.

La razón de utilizar este método de segmentación en el proyecto es que funciona muy bien para imágenes que están uniformemente iluminadas. Obviamente esto es así ya que es un método basado en la varianza de grises, y si la imagen está uniformemente iluminada, la dispersión de grises será a causa del contenido de la imagen y no por la iluminación. Como se puede apreciar en la imagen mostrada anteriormente, las imágenes recibidas por la cámara del robot están iluminadas de una manera bastante uniforme, debido a que las propias paredes del laberinto igualan la luz que se refleja en sus paredes, luego utilizando este método se conseguirán buenos resultados.

Una vez segmentada la imagen, es recomendable aplicar algún filtro para reducir el posible ruido resultante de la segmentación. En este caso utilizaremos un filtro bilateral. Utilizaremos éste en concreto, ya que es un filtro no lineal que se caracteriza por preservar los bordes y eliminar ruido. Nos interesa que los bordes no se modifiquen para que el robot pueda detectar con facilidad las cruces que le indicarán que está viendo un landmark.

Tras aplicar la segmentación de Otsu y posteriormente el filtro bilateral para reducirle el ruido, la imagen resultante será similar a la siguiente:

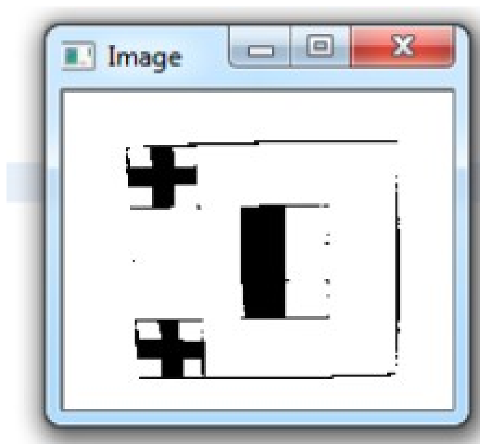


Figura 3.11: Imagen segmentada y con filtro aplicado

Extracción de información de la imagen

Ahora la imagen ya estará lista para su procesamiento y extracción de la información que nos interesa, en este caso su identificación.

Lo primero que el robot deberá hacer es verificar que la imagen que esté recibiendo en cada momento contenga o no un landmark. Para ello, tendrá que comprobar si la imagen contiene dos cruces, las cuales como hemos explicado en el diseño de las balizas, indican que en la imagen se encuentra un landmark. Esto lo lograremos pasando unas matrices de diversos tamaños a las cuales llamaremos máscaras, cuyos elementos "representen" la forma que estamos buscando en la imagen, en nuestro caso, una cruz. Trabajaremos con cuatro tamaños de ventana distinto:

- Tamaño de ventana 15x15
- Tamaño de ventana 21x21
- Tamaño de ventana 27x27
- Tamaño de ventana 33x33

En estas matrices, pondremos a 1 los elementos que deberían ser píxeles blancos y a 0 los píxeles que deberían ser negros. Como ejemplo, supongamos que utilizaríamos un tamaño de ventana 3x3, la matriz resultante sería de la siguiente manera:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Con cada máscara, se irán desplazando pixel a pixel por la imagen y comprobando que los elementos de la máscara y de la imagen coincidan. Si coinciden, es que existe una forma como la de la matriz en esa parte de la imagen. Por motivos de ruido y que las imágenes recibidas por la cámara del robot no son perfectas, no se pedirá que coincidan todos los elementos de la máscara, si no un 85% de coincidencias. Así pues, el total de coincidencias mínimo viene definido por la siguiente fórmula:

$$f(s) = s^2 * 0.85$$

siendo *s* el tamaño de ventana utilizado en ese momento. Para decidir que en la imagen hay un landmark, se deben detectar dos de estas cruces, pero como no le pedimos el 100% de coincidencias en los píxeles, es posible que una misma cruz sea detectada más de una vez con los píxeles adyacentes a éste. Por eso, no se considerará una segunda cruz como detectada, si la distancia entre el pixel del centro de la primera cruz detectada y la segunda sea igual o superior a 2 por el tamaño de ventana que se esté utilizando, ya que físicamente en el diseño de las landmarks, están separadas por esa distancia.

Una vez el robot detecte dos cruces, sabrá que está viendo un landmark y tiene que procesar la imagen para obtener el ID de éste. Para ello obtendrá una subimagen de la imagen cuyo origen estará en la esquina inferior derecha de la primera cruz detectada, y que tendrá un tamaño de la distancia entre la parte inferior de las cruces. Así obtendremos un a imagen que abarque exactamente la parte del landmark donde se encuentra la codificación de la ID.

Ahora, sólo queda repetir el mismo proceso que hemos hecho para buscar las cruces en la imagen completa, pero en este caso buscaremos máscaras que “representen” cuadrados de un tercio del tamaño de la subimagen, ya que esta está dividida en 3x3 elementos. Recorreremos la imagen esta vez no pixel a pixel, si no en 9 posiciones que coincidirá donde podrá o no estar pintado de negro ese elemento. Si está pintado, en una matriz de 3x3 pondremos un 1, en caso contrario un 0. La matriz resultante obtendrá la codificación en binario del ID correspondiente a ese landmark. Ahora solo falta decodificar esa información para obtener la ID en entero, para ello utilizaremos una función cuyo pseudocódigo es el siguiente:

```
function getLandmarkID(mat):
    String binaryNumber;
    for (i = 0; i < 3; i++){
        for(j = 2; j  >= 0; j++){
            binaryNumber.append(mat[i,j]);
        }
    }
    return binaryStringToInt(binaryNumber);
```

En el caso de que no se encuentren las cruces que determinan la existencia de un landmark en la imagen, se repetirá el proceso con el resto de tamaños de ventana, siempre comenzando por el tamaño más pequeño y terminando por el más grande, ya que el tamaño de ventana utilizado en cada caso quedara condicionado por la distancia del landmark al objetivo de la cámara del robot ya que a más lejos se encuentre, más pequeño será el tamaño de ventana utilizado, y es lógico que comience a ver el landmark a distancia a lo largo de un pasillo del laberinto.

Una vez identificado el landmark, se mostrará la información por pantalla sobre la imagen analizada y también devolverá el valor del ID asignado a ese landmark para que el robot ya pueda saber si había visitado antes ese cruce. En el caso de que no se detecte un landmark, la ID devuelta será siempre 0.

A continuación se mostrarán una serie de diversos landmarks en los que se extrae su ID aplicando los procedimientos explicados anteriormente, utilizando diferentes tamaños de máscara.

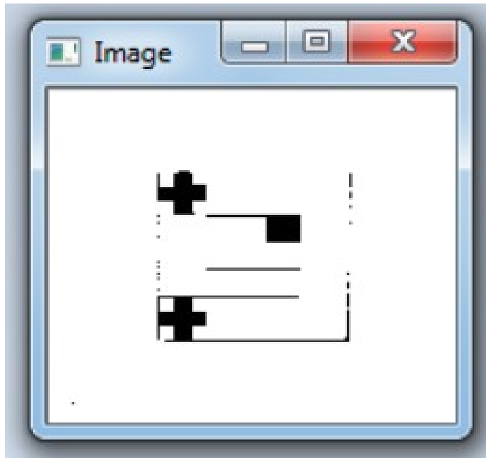


Figura 3.12: ID 1 Segmentada

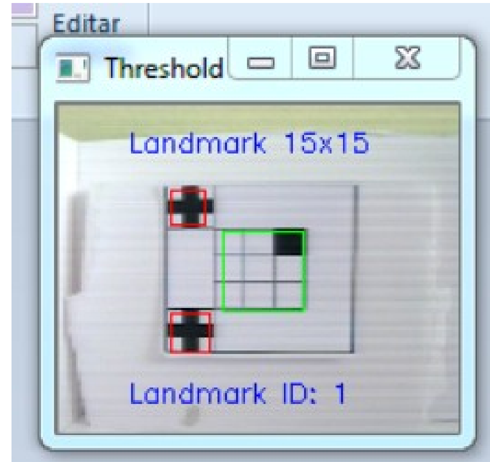


Figura 3.13: ID 1 a tamaño 15x15

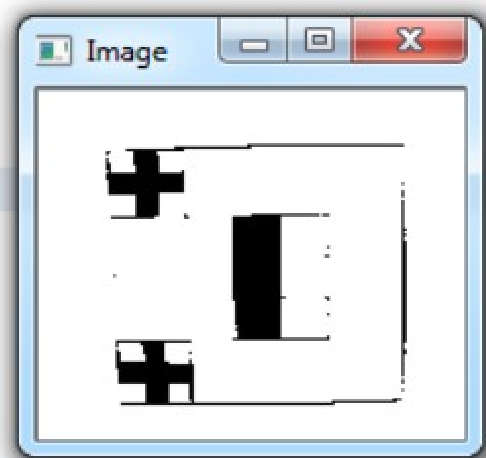


Figura 3.14: ID 146 segmentada

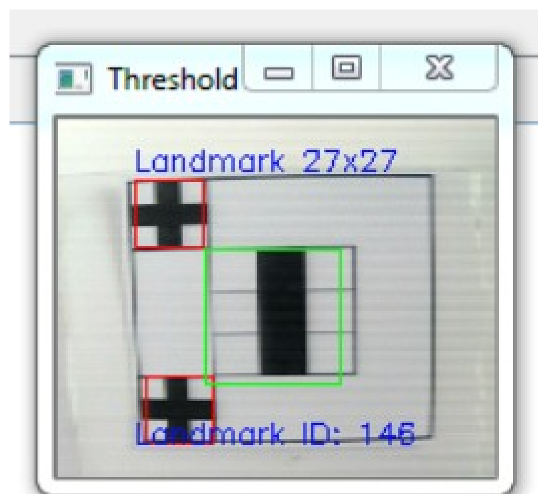


Figura 3.15: ID 146 a tamaño 27x27

Ahora el robot ya sabrá la identificación única de cada cruce, por lo que lo siguiente que deberá hacer es tomar la decisión de cual seguirá el siguiente camino a tomar. Para ello, antes tendremos que recolocar al robot en la posición original en la que encontró por primera vez el cruce. Esto quiere decir, que si llega a un cruce y al leer su ID ve que es una intersección en la cual ya había estado, el robot deberá recolocarse de tal manera que acabe en la misma posición en la que se encontró por primera vez en el cruce para así poder tomar correctamente la siguiente decisión. En el caso que sea la primera vez que visita ese cruce, deberá colocarse en la posición con la que se haya encontrado con ese cruce. Para ello, utilizaremos la siguiente función:

```
function relocateRobot(int originalIntersectionType, robot):
    if originalIntersectionType == 3:
        //Avanzamos 10 centímetros el robot
        robot.avanzar(100);
    if originalIntersectionType == 4:
        //Avanzamos 10 centímetros el robot
        robot.avanzar(100);
        //Giramos el robot 90º a la izquierda
        robot.girarIzquierda(90);
    if originalIntersectionType == 5:
        //Avanzamos 10 centímetros el robot
        robot.avanzar(100);
        //Giramos el robot 90º a la derecha
        robot.girarDerecha(90);
```

3.5 Toma de decisión

En este punto del algoritmo, el robot tendrá que tomar la decisión de cual será el siguiente camino que tendrá que recorrer. Nos interesa que las decisiones que tome le lleven a recorrer todos los pasillos del laberinto para así poder generar un mapa completo de este y que además lo haga de la manera más eficiente posible, es decir, pasando el mínimo de veces posibles por los distintos cruces del laberinto.

Existe el popular método que cumple que ir siempre con la mano izquierda pegada a la pared de un laberinto tarde o temprano se encontrará la salida, es decir, en nuestro caso siempre que se encuentre un cruce, girar a la izquierda. Es cierto que al final se alcanza la salida del laberinto, pero en nuestro caso esta idea no nos servirá ya que nosotros inicialmente lo que queremos no es alcanzar la salida, si no recorrer todo el laberinto para poder mapear todos los caminos de éste y así poder mapearlo. A continuación, explicaremos un algoritmo con el que lograremos este fin.

3.5.1 Búsqueda en profundidad

La búsqueda en profundidad es un tipo de búsqueda que no utiliza heurística utilizada en la inteligencia artificial y ciencias de la computación para recorrer árboles o grafos completamente. La idea de esta búsqueda es la de ir desarrollando los hijos de cada nodo hasta que ya no se pueda descender más, subir un nivel, y desarrollar el siguiente hijo de ese nivel que faltaba por desarrollar, volviendo a repetir el proceso hasta que no se pueda descender más. El algoritmo terminará cuando vuelva hasta el nodo raíz, lo que querrá decir que ya no quedan más hijos por desarrollar en todo el árbol y este habrá sido recorrido completamente. En este pequeño ejemplo se puede ver el funcionamiento descrito con el orden de acceso a los diferentes nodos del árbol:

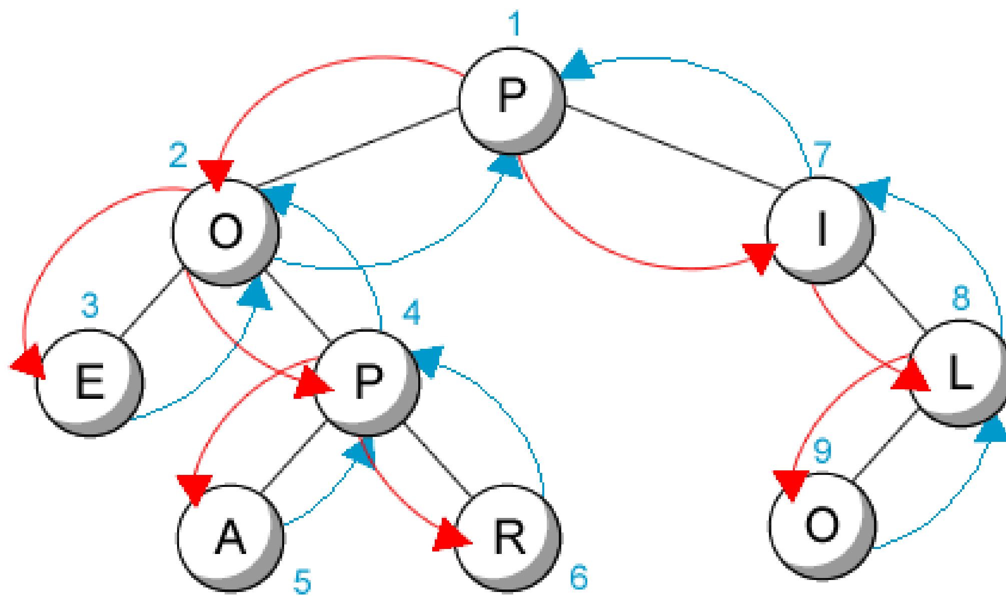


Figura 3.16: Árbol recorrido con búsqueda en profundidad

Como podemos ver es ideal para lograr nuestro objetivo, ya que si tratamos a los cruces como nodos y a sus posibles decisiones como sus posibles hijos, podremos generar un mapa lógico completo que el robot luego podrá interpretar para llegar de un nodo a otro sin perderse. El robot iría desarrollando todos los cruces siguiendo la ideología del algoritmo de búsqueda en profundidad hasta que terminaría llegando al punto de inicio, lo cual significaría que habría terminado de mapear todo el laberinto.

3.5.2 Representación de los cruces como nodos

Como hemos mencionado en el apartado anterior, nos interesa representar a los cruces del laberinto como nodos con los cuales iremos generando un árbol. A continuación mostramos la representación de la clase nodo:

```
class Nodo:
    def __init__(self,landmarkID,intersection):
        self.landmarkID = landmarkID
        self.intersection = intersection
        self.left = None
        self.center = None
        self.right = None
        if intersection == 0:
            self.landmarkID = 0
            self.left = False
            self.center = False
            self.right = False
        elif intersection == 3:
            self.center = False
        elif intersection == 4:
            self.right = False
        elif intersection == 5:
            self.left = False
```

Figura 3.17: Clase nodo

Como se puede apreciar, cada nodo tendrá 5 atributos:

- landmarkID: Identificación de ese nodo en concreto.
- intersection: Tipo de intersección en el que originalmente se encontró ese nodo.
- left: Posible hijo izquierdo de ese nodo
- center: Posible hijo central de ese nodo
- right: Posible hijo derecho de ese nodo

A la hora de crear un nuevo nodo, se le indicará la ID que se ha extraído del landmark de ese

cruce, y el tipo de intersección en el que ha sido detectado. En función de que tipo de cruce sea, se modificarán los valores de su posibles hijos, inicializando a *False* los que no sean posibles. Por ejemplo, en un cruce de tipo 0 que es un callejón sin salida, todos sus posibles hijos se igualarán a *False* ya que este tipo de cruces no puede tener más hijos.

Además, esta clase posee una serie de métodos propios que proporcionan posibilidades para trabajar con el árbol que se ira construyendo. Por ejemplo, existe un método que indicándole la ID del padre, buscará recursivamente con una búsqueda en profundidad a lo largo del árbol e indicándole en que hijo quiere añadir un nodo, le añade un nodo con una ID y un tipo de cruce.

3.5.3 Aplicación de la búsqueda en profundidad para el mapeo completo

Ahora que ya tenemos una manera de representar los cruces en forma de nodos, tenemos que aplicar la ideología de la búsqueda en profundidad para recorrer todos los nodos (o cruces en el caso del laberinto) hasta recorrerlo completamente y volver al punto de inicio lo que indicará al robot que el mapeo ha sido completado.

Cada vez que el robot llegue a un cruce, identifique el tipo, obtenga el ID de éste y se recoloque para acabar en la posición original con la que se encontró ese cruce, es momento de que tome la siguiente decisión. Para ello, se obtendrá el nodo actual pasándole la ID extraída del landmark. Si el nodo no existiera en el árbol, se añadiría un nuevo hijo con la ID leída, al nodo padre actual en su rama correspondiente a la última decisión tomada y a continuación, se desarrollaría el primer hijo del nodo recién añadido que fuese desarrollable, es decir, que fuese igual a *None.* En el caso de que el nodo ya existiera en el árbol, se desarrollaría el siguiente hijo que fuera posible (el siguiente igual a *None*). Cuando no quede ningún hijo desarrollable en el nodo actual, el robot dará media vuelta y emprenderá el viaje hasta llegar al nodo que actualmente es el padre, para repetir el proceso y tratar de desarrollar sus hijos restantes. El proceso de intentar desarrollar el siguiente nodo, siempre seguirá un orden de izquierda a derecha, es decir, en primera instancia se intentará girar a la izquierda, luego seguir por el centro y por último se intentará girar a la derecha.

Cuando el robot fuese escalando por los nodos al no quedar más hijos desarrollables en éstos, en el momento en el cual llegase al punto de inicio, significaría que ya no habría más nodos para abrir en todo el árbol, por lo que el mapeo estaría finalizado.

Así pues, al finalizar de recorrer el laberinto nuestro robot tendrá completo un árbol que para él será un mapa lógico que lo represente. En nuestro caso, el árbol resultante del laberinto en el que hemos efectuado las pruebas será el siguiente:

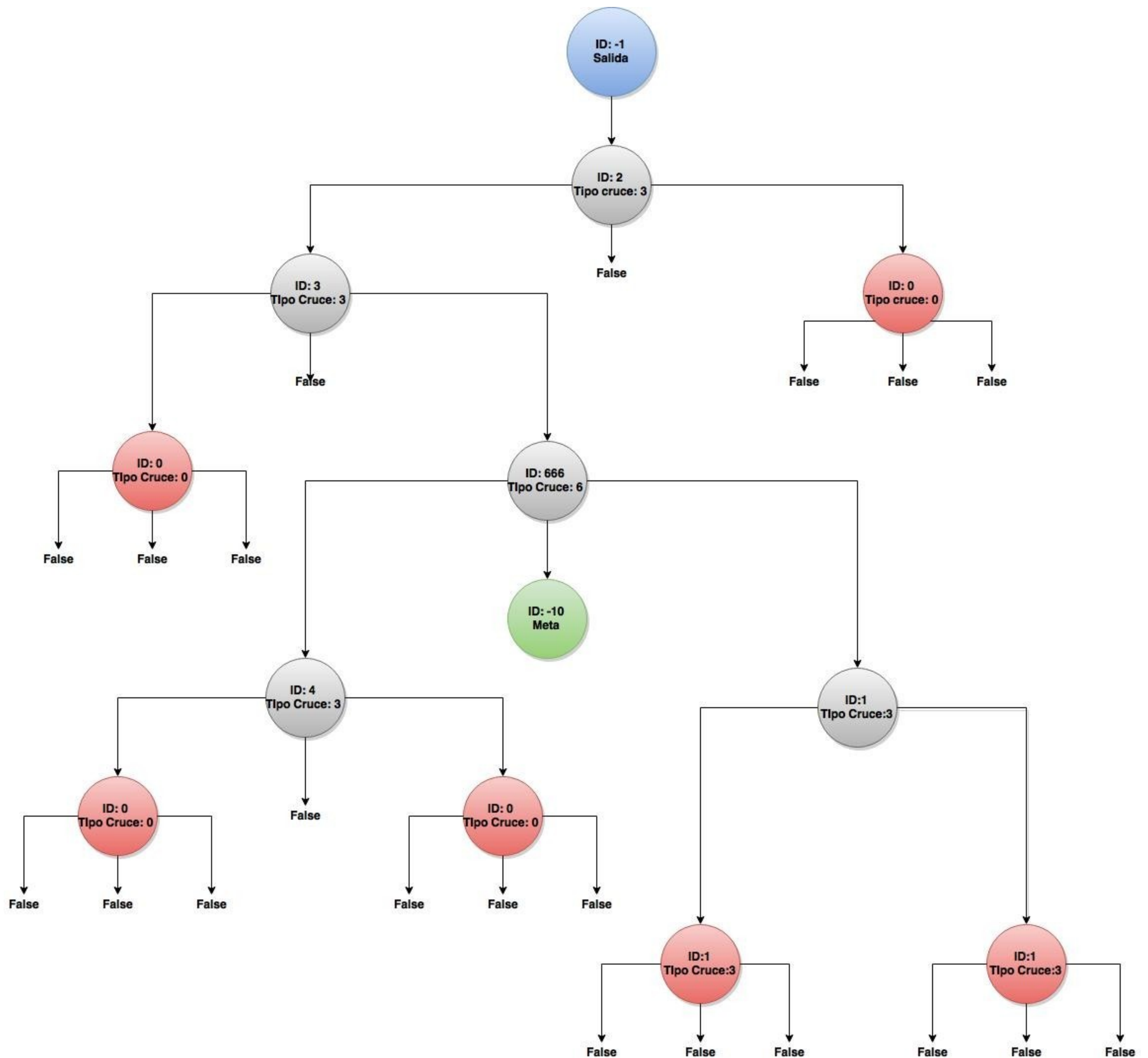


Figura 3.18: Árbol resulntante del mapeo del laberinto.

Como se puede apreciar, de cada cruce guardará su ID correspondiente y el tipo de cruce con el que originalmente llegó a esa intersección, amén de los siguientes cruces que alcanzará el robot si tomara la decisión de girar a la izquierda, seguir recto, o girar a la derecha, respectivamente. En la imagen también vemos que hemos añadido 2 nodos especiales, que son la salida y la meta, aunque podrían omitirse si utilizamos un cruce como salida y otro como meta.

A continuación, se muestra una imagen del laberinto que usaremos en las pruebas y del cual se generará el árbol mostrado anteriormente en el proceso que el robot va mapeando el mismo.

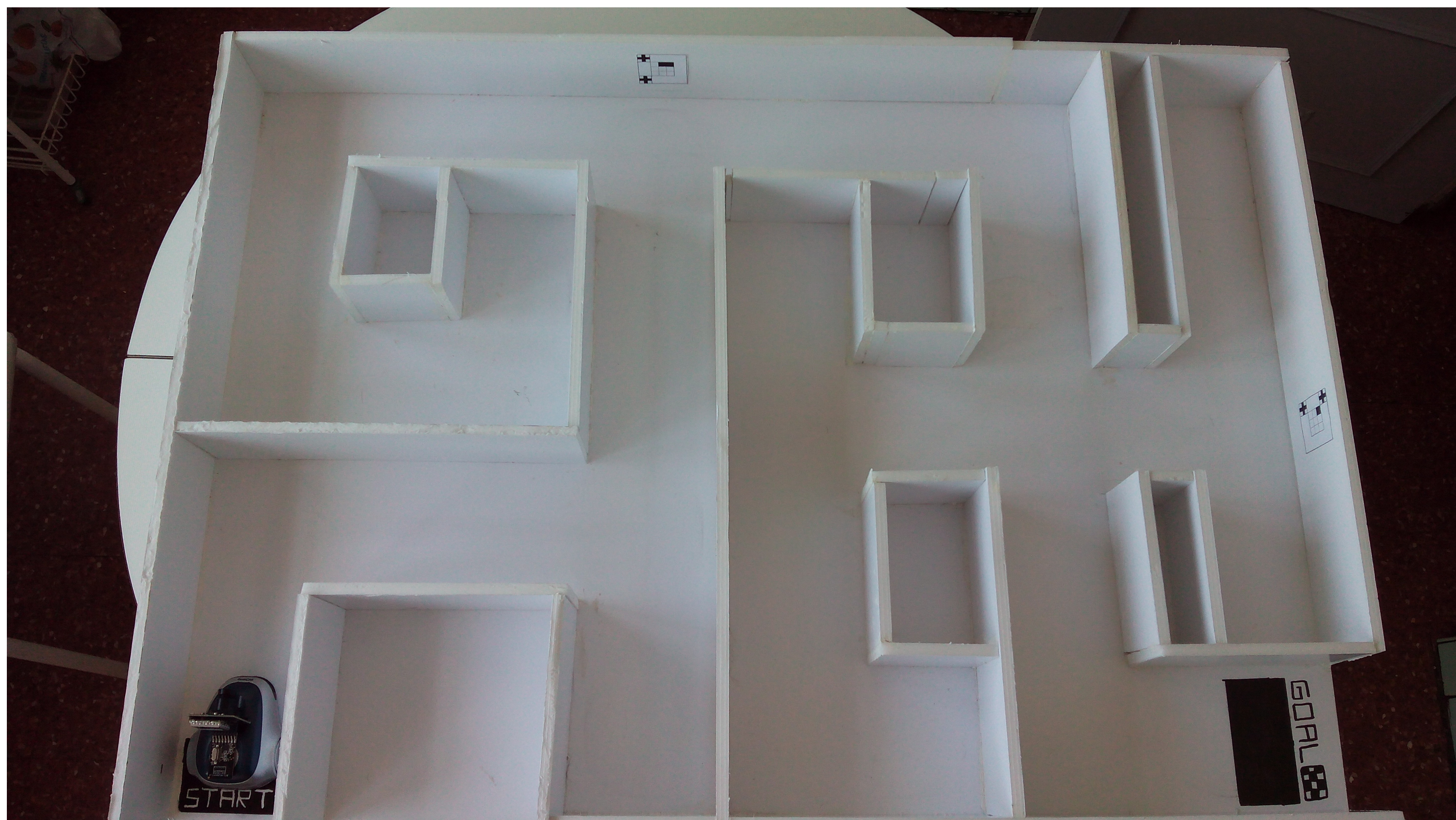


Figura 3.19: Laberinto utilizado en el desarrollo del proyecto.

Así pues, una vez el robot ya disponga de un mapa lógico del laberinto que acaba de recorrer, lo siguiente que deberá hacer es ser capaz de obtener el camino entre dos puntos indicados, que en nuestro caso serán la salida y la meta a partir de este mapa.

3.5.4 Extracción de caminos mediante búsqueda en profundidad

Como hemos mencionado anteriormente, en este punto del algoritmo el robot se encontrará en el mismo punto que estaba al comenzar el mapeo. Ahora, si le indicamos un punto a alcanzar deberá ser capaz de utilizar el mapa para encontrar el camino hasta él.

Partiendo del árbol generado durante el proceso de mapeo, nos interesa quedarnos con un sub-árbol de éste , que una únicamente la salida con la meta. A continuación mostramos este sub-árbol:

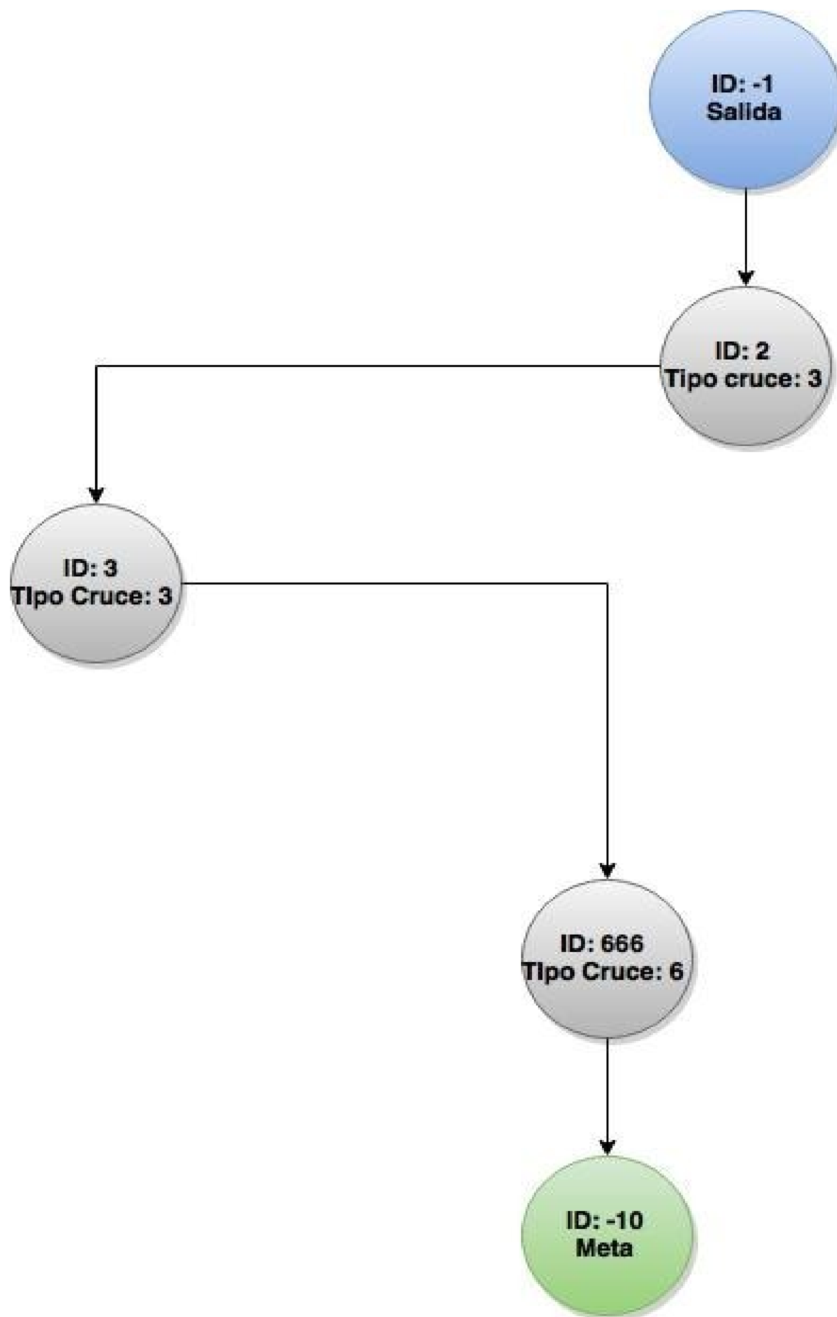


Figura 3.20: Árbol solución del laberinto

Como se puede observar en este nuevo grafo, lo que tiene que hacer nuestro robot para alcanzar la meta es lo siguiente:

1. El robot comenzará su marcha en la salida
2. En la intersección cuya ID es 2, deberá girar a la izquierda.
3. En la intersección cuya ID sea 3, deberá girar a la derecha.
4. En la intersección cuya ID sea 666, es decir, en la de tipo todas direcciones, deberá seguir recto.
5. El robot finalmente alcanzará la meta.

Para conseguir esto, realizaremos una búsqueda en profundidad sobre el árbol que representa el mapa del laberinto. Conforme la búsqueda vaya recorriendo el árbol, irá generando una lista de tuplas, donde cada tupla estará formada por la ID de cada cruce, y la decisión que ha tomado para alcanzar el siguiente nodo (la decisión estará representado con el número 1, 2 o 3, representando estos el girar a la izquierda, continuar recto, o girar a la derecha respectivamente).

Si la búsqueda no converge en solución, esa lista se descartará y encaso contrario, es decir, el caso en que se encuentre el camino al nodo deseado, se devolverá la lista con los nodos y las decisiones necesarias para llegar hasta este. En nuestro caso, la lista a la cual llamaremos *pathToNode* contendrá los siguientes datos:

$$pathToNode=[(2,1),(3,3),(666,2)]$$

Nuestro robot irá sacando el primer elemento de esta lista y al llegar a una intersección, este comprobará que la ID extraída del landmark de ese cruce, coincida con la ID que indique elemento actual de la lista. En el caso que sean iguales, significará que no ha habido ningún problema y que el robot se encontrará en el nodo que realmente debería estar, por lo que tomará la decisión que el elemento de la lista le indique. En el caso de que las ID no coincida, significará que algo ha ido mal y el robot se ha perdido, por lo que se detendrá y se mostrará un mensaje de error por la pantalla indicando que algo ha ido mal.

Este proceso se irá repitiendo hasta que la lista esté vacía y el robot haya llegado a la meta.

3.5.4.1 Inconvenientes de utilizar la búsqueda en profundidad

El problema de utilizar el algoritmo de la búsqueda en profundidad para encontrar el camino hasta un determinado nodo, es que dado que este tipo de búsqueda no utiliza ninguna heurística para realizarla, la solución que encuentre no tiene por que ser la óptima, es decir, no tiene por que ser el camino más corto para alanzar ese punto. El camino que la búsqueda en profundidad devuelva será el primero que encuentre. En nuestro caso, si que será el camino óptimo, pero esto sucede porque también es el único camino que lleva hasta la meta.

Otro inconveniente, es que este tipo de búsqueda es computacionalmente costosa. En nuestro caso no causa ningún inconveniente ya que nuestro árbol no va a tener muchos nodos, pero si el laberinto fuese gigante podría suponer un problema.

4. Mapeo visual

Lo siguiente que nos interesará será el generar una representación visual del mundo real y del entorno en el cual trabaja el robot mOway en nuestro ordenador, en principio, con motivos únicamente estéticos para observar la estructura que presenta el laberinto , aunque este nuevo mapa se podría utilizar para ciertas mejoras en el funcionamiento robot, por ejemplo el de buscar caminos óptimos entre dos puntos distintos, es decir, el camino más corto. Hablaremos más en detalle sobre esto en el apartado de conclusiones y mejoras.

Lo primero que el robot tiene que hacer para poder continuar con el proceso de mapeo, es saber en que punto del plano que será el mapa se encuentra en cada momento con respecto a su posición inicial. En este momento entra en juego el estudio de la odometría.

4.1 Odometría

En el ámbito de la robótica móvil, concretamente en aquella que engloba los vehículos cuyo desplazamiento se realiza con ruedas, la odometría es el estudio de la estimación de la posición de estos robots con respecto a su posición inicial. Para realizar esta estimación se usa información sobre la rotación de las ruedas para estimar cambios en la posición a lo largo del tiempo.

El problema es que esta información se ve afectada por muchos errores que hacen que no sea nada precisa, por lo que la posición en la que supuestamente el robot se encuentra se aleje en gran medida de la realidad. Podemos diferenciar dos tipos de errores principales:

Errores sistemáticos

Son aquellos errores debidos a las propias imperfecciones del robot. Algunos de los ejemplos más comunes son:

- Las ruedas son de diámetros diferentes
- Las ruedas están mal alineadas.
- La información recibida por los sensores que cuentan las vueltas de las ruedas no es perfecta.

Errores no sistemáticos

Son aquellos errores que se producen por factores externos al robot, como puede ser por desperfectos en el terreno en los que el robot se desenvuelve. Los casos mas frecuentes son los siguientes:

- Desnivel del terreno
- Fuerzas externas que modifiquen la trayectoria del robot
- Giros imprecisos debido a diversos factores como por ejemplo el patinaje de ruedas

Como es comprensible, los errores más graves son los que más imprecisión añaden a la posición del robo. Éstos son los errores sistemáticos, ya que se acumulan constantemente a lo largo de la navegación del robot.

4.2 Odometría en nuestro proyecto

El método que utilizaremos para estimar la posición del robot mOway con respecto a la posición inicial utilizará la distancia que ha recorrido el robot en cada intervalo de tiempo y la orientación que tenga en dicho intervalo.

Obtendremos el total de la distancia que ha recorrido en ese intervalo y junto con la orientación que posea en ese momento el robot, calcularemos el vector que determinará su nueva posición, donde el inicio del vector será su posición actual y el extremo de éste sera su nueva posición.

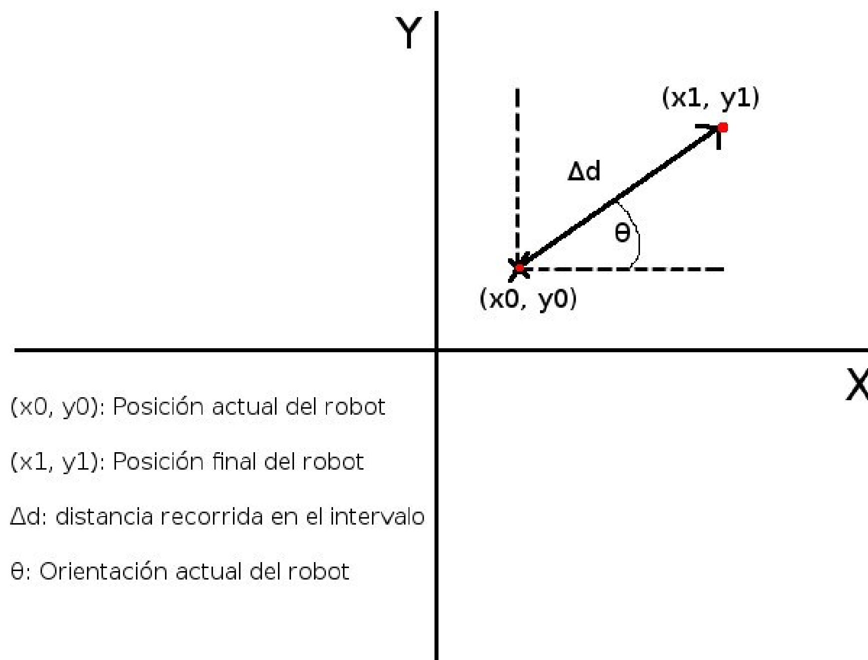


Figura 4.1: Vector que determina la nueva posición del robot

Así pues, teniendo la posición actual del robot, la distancia que ha recorrido en un intervalo de tiempo y la orientación de éste, la manera de calcular el nuevo punto es la siguiente:

$$x1 = x0 + \cos(\theta) * \Delta d \qquad y1 = y0 + sen(\theta) * -1 * \Delta d$$

La razón de invertir el signo en la ecuación para calcular *y1*, es que en la pantalla de un ordenador las coordenadas de Y son opuestas a como funcionan en un sistema de eje cartesiano.

4.3 Subsanación de errores sistemáticos

Como ya hemos mencionado anteriormente, el robot sufre de una serie de errores que hacen que el cálculo de su posición vaya siendo cada vez peor a lo largo del tiempo. Hemos explicado que los errores que más afectan en este sentido, son los errores sistemáticos ya que se van acumulando continuamente mientras el robot avanza. Por eso, trataremos de reducir estos errores de la siguiente manera:

Ya que es un error que aumenta conforme el paso del tiempo, trataremos de ir acotando las mediciones a lo largo del tiempo de tal manera que estos errores tendrán menos espacio para ir incrementando. Cada vez que el robot se encuentre con un cruce en concreto por primera vez, este se guardará en que posición se encontraba cuando lo vio originalmente, así, cuando vuelva a éste al terminar de desarrollar sus hijos, obtendrá la posición en la que lo había visto originalmente y se la asignará a la posición actual del robot. De esta manera, descartaremos todos los errores que el robot ha acumulado desarrollando su hijo y solo contará con la cantidad de error que hubiese acumulado hasta antes de encontrarse por primera vez con esa intersección.

A continuación mostraremos un par de ejemplos en los que pondremos al robot una pequeña parte del recorrido del laberinto original en la que tendrá que volver hasta su posición original tras haberlo recorrido completamente. En el primer ejemplo, no aplicaremos el método descrito anteriormente para reducir los errores sistemáticos, mientras que en el segundo si que lo usaremos, para poder ver así las diferencias entre uno y otro.

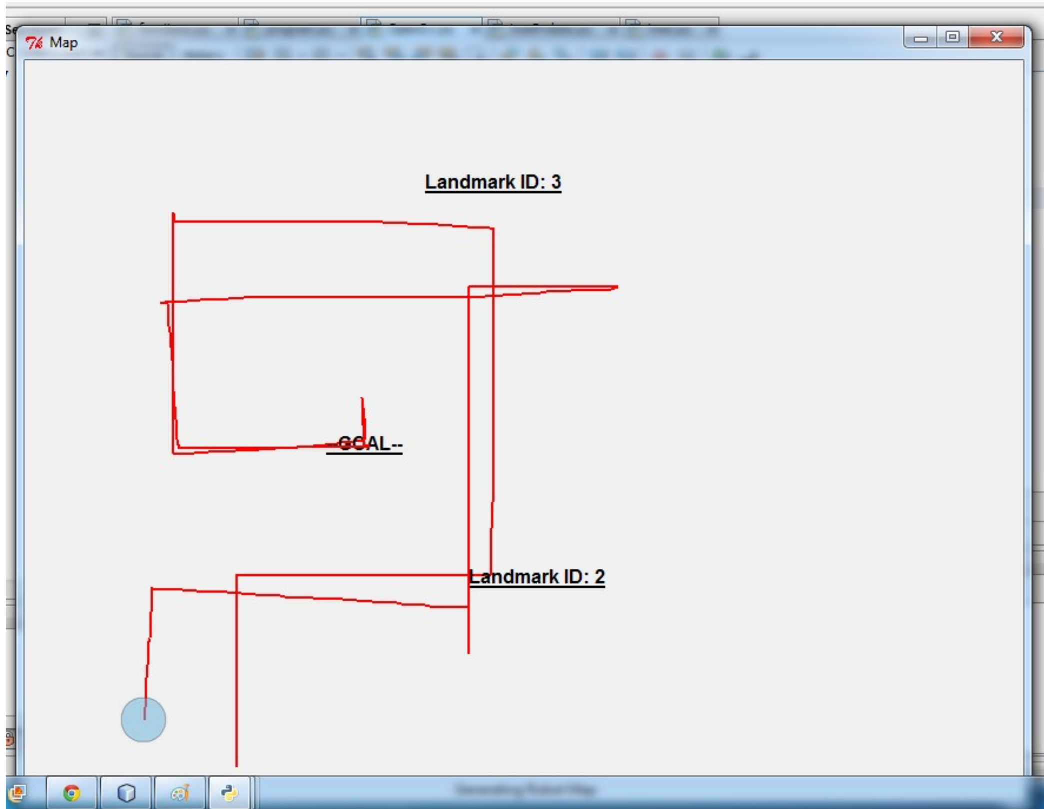


Figura 4.2: Odometría con errores sistemáticos

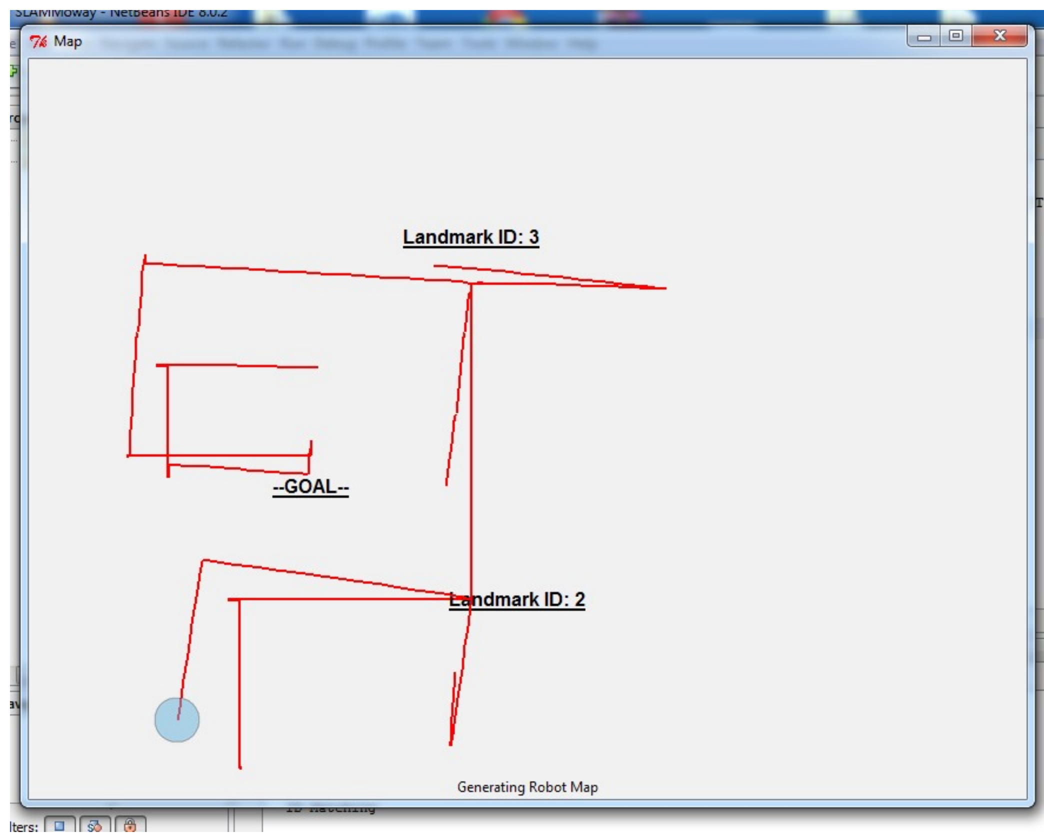


Figura 4.3: Odometría con un grado menor de errores sistemáticos

Como se puede apreciar a simple vista, las diferencias entre las posiciones del robot que tiene todo el error sistemático y el que las trata de subsanar en cierta parte acumulándolas en menor medida es notoria. Lo primero que podemos observar es que las posiciones finales en el segundo caso es mas cercana a la posición inicial por lo que es más acertado con su posición real. Además, el recorrido que se representa con las líneas rojas en el caso que reducimos los errores se acerca más a la forma que tienen los pasillos que el robot ha recorrido.

4.4 Mapeo de obstáculos

Ahora que ya dispondremos de una ubicación aproximada del robot en cada momento lo siguiente que tendremos que hacer es calcular la posición de los obstáculos que pueda haber respecto a la posición del robot. Para ello, utilizaremos los valores que nos devuelvan los sensores de proximidad del robot. Recordamos que el robot mOway dispone de 4 sensores, 2 laterales y 2 frontales.

Lo primero que debemos hacer es crear una matriz donde todos sus elementos inicialmente valdrán cero. En el caso de que el robot detecte un obstáculo, pondremos a 1 la celda de la matriz correspondiente a la posición en el mapa de ese obstáculo. Para saber que posición corresponde a cada elemento de la matriz, dividiremos la variable x de la posición del robot entre el tamaño que querremos que ocupe cada elemento de la matriz (en nuestro caso serán cuadrados de 10 por 10 píxeles) para obtener la fila de la matriz, y la variable y también por el tamaño que vaya a representar.

Para calcular la posición de los obstáculos con respecto al robot, utilizaremos el mismo principio que hemos definido para calcular el vector de la distancia que ha recorrido el robot en un intervalo de tiempo, pero en esta ocasión el módulo del vector vendrá en función de los valores devueltos por los sensores de proximidad. También hay que tener en cuenta que los sensores laterales del robot están dispuestos a 45 grados con respecto al eje central del robot, por lo que a la hora de calcular la posición del robot habrá que añadir a la orientación que el robot en ese momento 45º para el de la izquierda y restarle 45º para el de la derecha.

Para simplificar el proceso, cada vez que el robot alcance una intersección por primera vez, se aplicara sobre esta matriz una plantilla con la forma de ese tipo de cruce y pondrá las casillas correspondientes a 1. Estas plantillas serán aplicadas teniendo en cuenta la orientación del robot ya que podrá llegar a ellos con cualquier orientación.

A continuación mostramos el resultado de un proceso del mapeo visual obtenidos siguiendo este procedimiento:

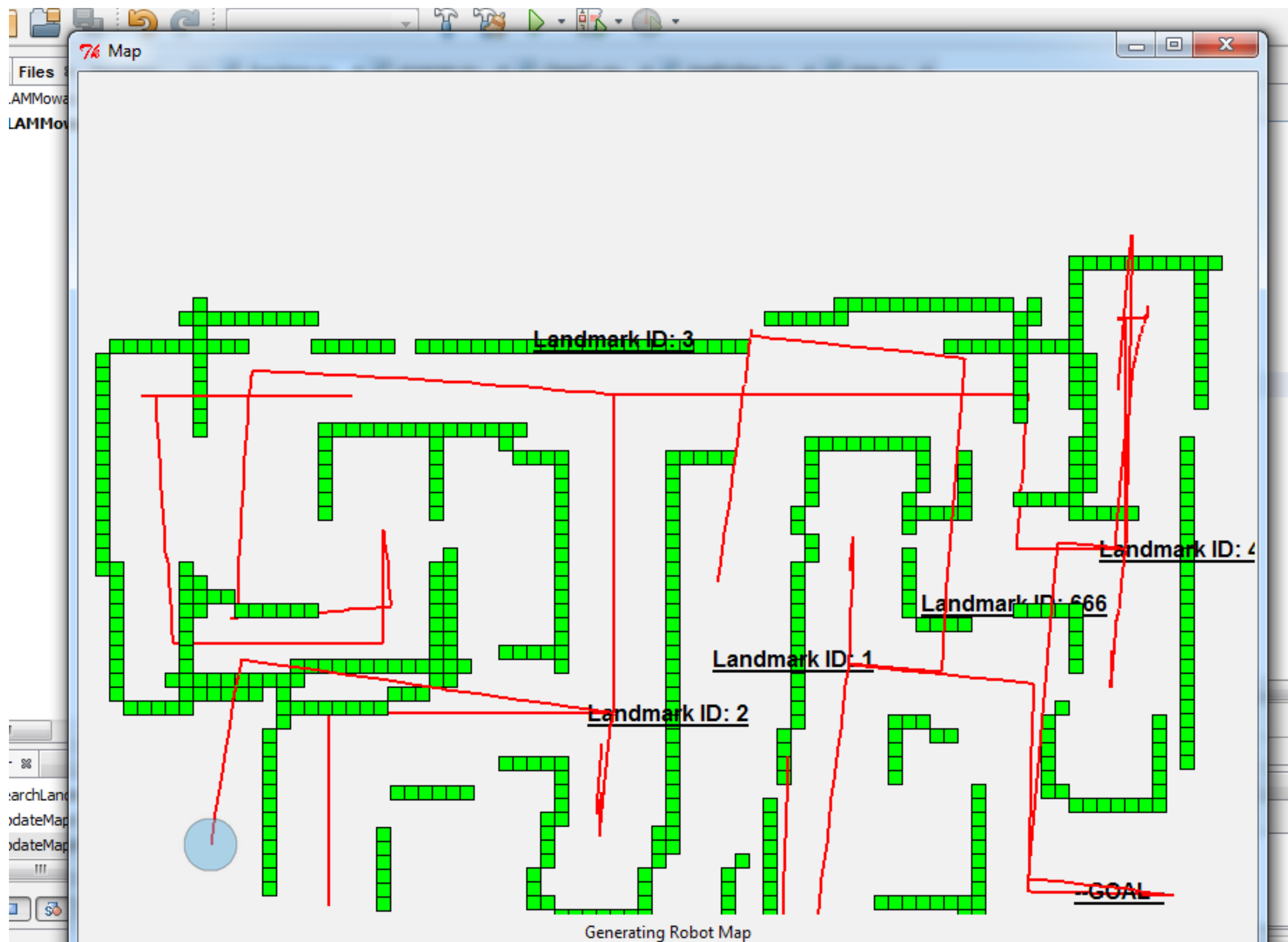


Figura 4.4: Resultado del mapeo visual

Como se puede observar, se aprecia la forma básica del laberinto en el resultado, pero también es fácilmente apreciable que tiene mucho ruido y que se puede mejorar en gran medida la calidad del mapeo. Sobre este tema se hablará mas en detalle en el apartado de conclusiones y mejoras.

5. Arquitectura del Software

En este apartado explicaremos como estarán dispuestos los distintos componentes software del proyecto y detallaremos de que manera interactuarán entre ellos para obtener así los resultados deseados. Para el desarrollo de este proyecto, se han diferenciado 3 partes en concreto:

- Parte de Navegación
- Parte de Mapeado
- Parte de Visión Artificial

Como la navegación y todo lo referente al proceso de mapeo virtual van muy ligados, se ha decidido que estas dos partes permanecerán juntas en un sólo módulo o programa, mientras que la parte de la visión artificial será un módulo totalmente independiente de este primero. La razón de separarlo en dos módulos es que precisamos que la cámara esté capturando imágenes continuamente para que la lectura de los landmarks sea lo más precisa posible. El módulo de navegación con alguna de sus tareas detiene el ciclo principal del algoritmo por lo que no podría actualizarse a tiempo real las imágenes recibidas por la cámara del robot mOway.

5.1 Módulo de navegación y mapeo

Este módulo será el encargado de gobernar el robot a través del laberinto y tendrá que encargarse de todo lo que se refiere al apartado de navegación, como la detección de cruces, y también de la parte de mapeo, que comprende desde el desarrollo del mapa virtual hasta su finalización completa, hasta la extracción de los caminos entre los puntos deseados para que el robot pueda mediante el uso de su navegación alcanzar el punto deseado. Más adelante mostraremos el diagrama de transición de estados básico que sufre el robot dentro de este módulo en concreto.

5.2 Módulo de Visión Artificial

Ese módulo sin embargo, será el responsable de todo lo referente a la visión artificial usada en este proyecto, así como la obtención de imágenes, el procesamiento de estas, extracción de la información de los landmarks y decodificación de esta información. Más adelante mostraremos también en este caso un diagrama de transición de estados básico que sufre la cámara del robot dentro de éste módulo en concreto.

5.3 Diagramas de transición de estados

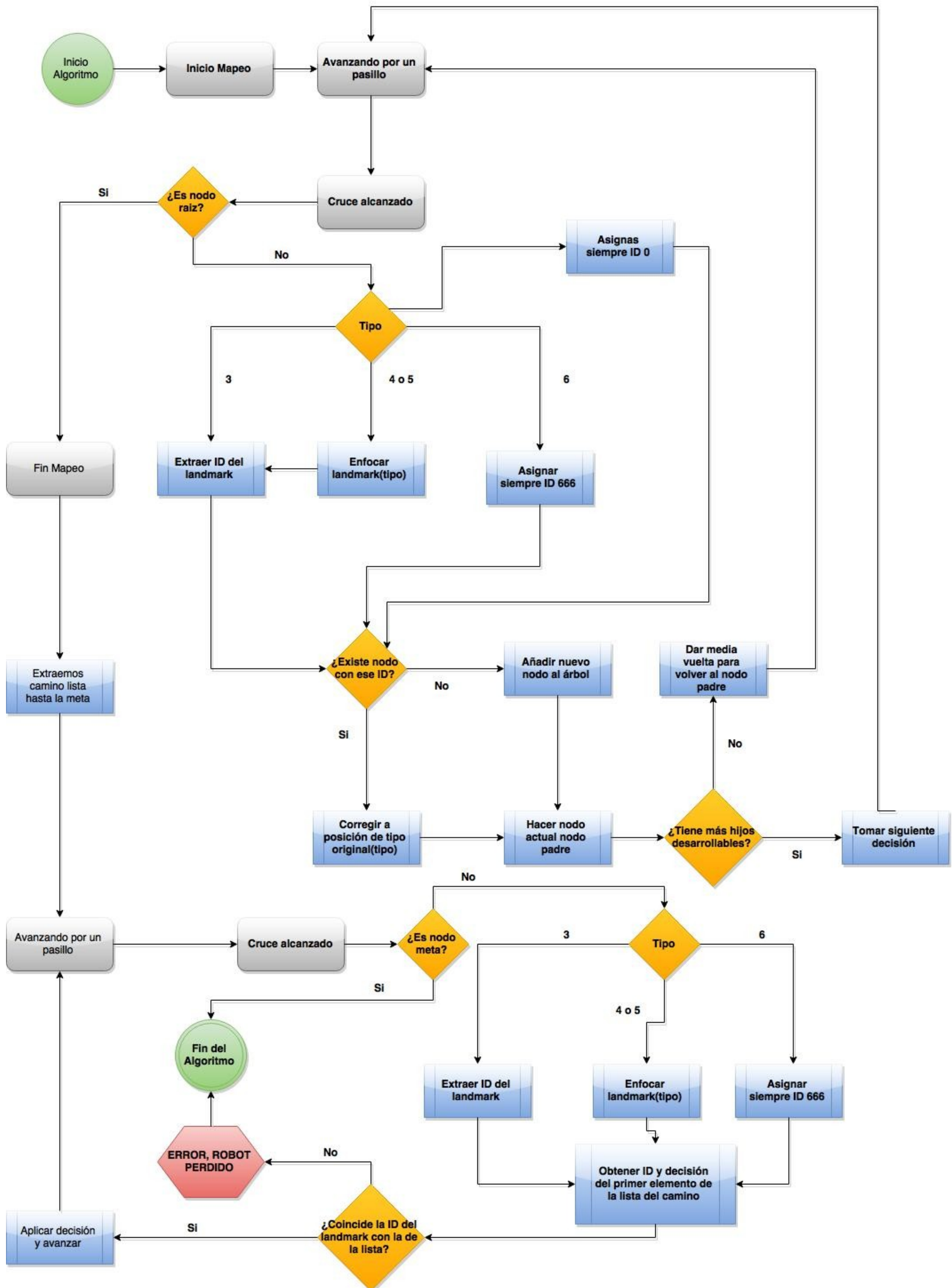


Figura 5.1: Diagrama de transición de estados de navegación y mapeo

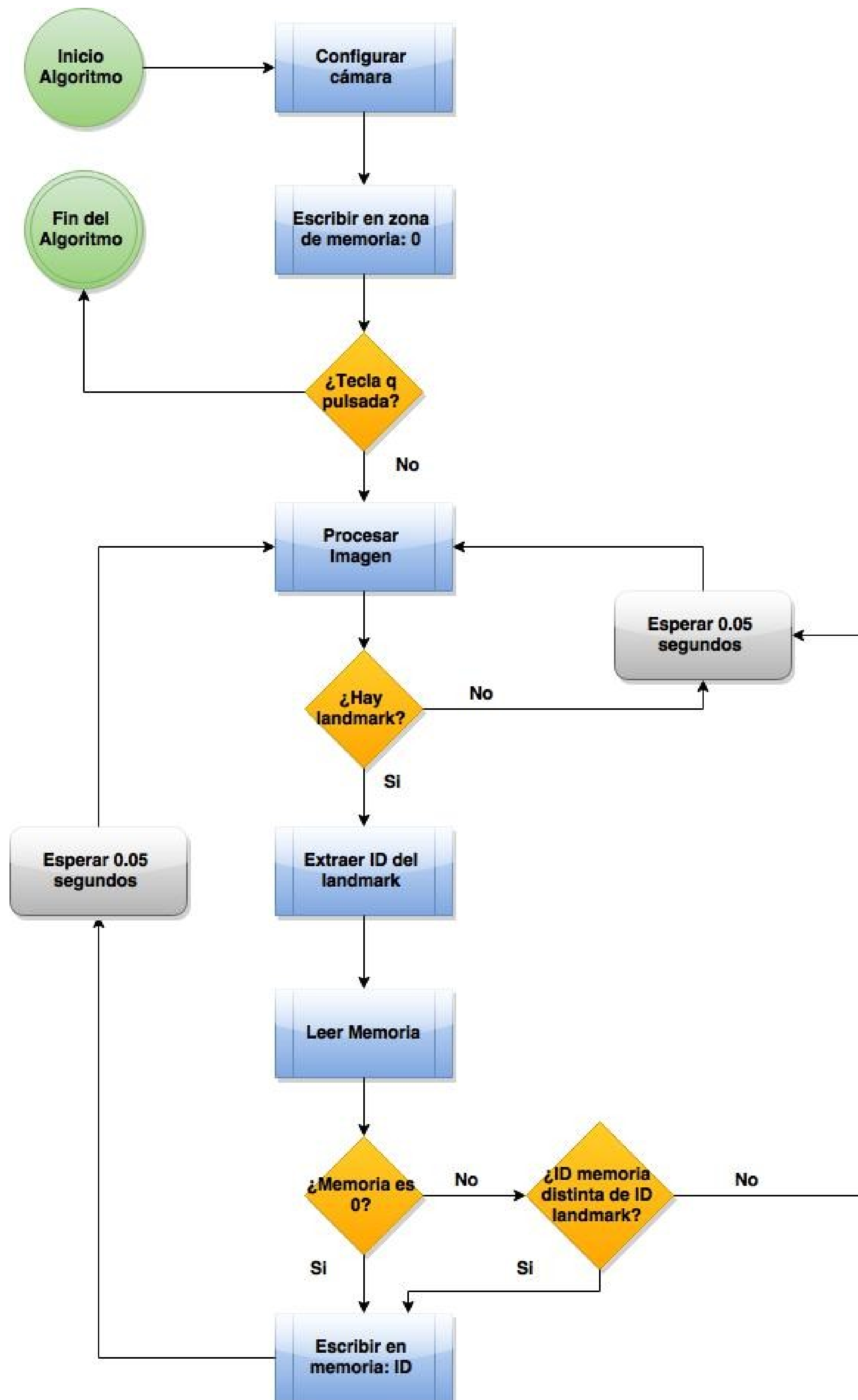


Figura 5.2: Diagrama de transición de estados del módulo de visión artificial

5.4 Comunicación entre los módulos de Navegación/Mapeo y Visión Artificial

Como es obvio, los módulos de Navegación/Mapeo y Visión Artificial deberán comunicarse de alguna manera para así poder transferirse información entre ellos. Para lograr esto, el módulo de visión artificial,al comenzar su ejecución creará una zona de memoria compartida en la que inicialmente escribirá un 0.

Como se puede apreciar en su diagrama de transición de estados, la primera ID que extraiga de un landmark la escribirá en la zona de memoria, y a partir de ahí, no volverá a escribir en la zona de memoria a no ser que relea esa ID y vea que es diferente, en cuyo caso lo que hará es actualizar dicho valor, ya que la cámara es más precisa a cortas distancias y es posible que el landmark se haya detectado desde una distancia elevada y puede que haya error en la ID extraída de la baliza.

A su vez, cuando el módulo de navegación y mapeo alcancen un cruce, precisarán de la ID de esa intersección para saber cual es la identificación correspondiente a ese cruce. Para obtenerla, accederá a la zona de memoria compartida con el módulo de visión artificial. En el caso de que esa zona de memoria haya un 0, querrá decir que el robot no habrá logrado ver el landmark y por lo tanto no habrá podido extraer su ID, por lo que tendría que empezar el proceso de reenfoque para tratar de ver el landmark.

Una vez haya extraído de la zona de memoria la ID del cruce, escribirá en esta un 0, lo cual será mi flag para el módulo de visión artificial que le indicará que el módulo de navegación y mapeo ya ha terminado de usar esa ID y que debe proceder a continuar actualizando la zona de memoria con las posibles siguientes ID que extraiga del siguiente landmark que visualice.

6. Resultados experimentales

Por último, procederemos a realizar una serie de pruebas con el robot en un laberinto que hemos construido para llevar a cabo las pruebas con el robot. Nuestro laberinto está construido con un material ligero y rígido llamado cartón-pluma. El motivo de utilizar este material, aparte de su ligereza y de que es fácil trabajar con él, es que tiene un buen nivel de refracción por lo que el funcionamiento de los sensores de proximidad de infrarrojos funciona de manera notoriamente mejor.

El laberinto consta de un total de 13 cruces en total, incluyendo un punto de salida y una meta. Estas intersecciones abarcan los 7 tipos de cruces descritos en el apartado de navegación, por lo que pondrá a prueba si el robot es capaz de reconocerlos correctamente o no. El procedimiento general de las pruebas será hacer un tándem de 3 intentos para cada prueba establecida, comparando así sus resultados para sacar las pertinentes conclusiones de las diferencias entre unas y otras.

6.1 Diferentes pruebas

Prueba 1: Mapeo general

Como primera prueba, ordenaremos al robot que mapee el laberinto al completo, y que al terminar se dirija a la meta del laberinto. Las dos primeras veces, el robot ha conseguido mapear el laberinto correctamente y a continuación obtenido el camino para llegar a la meta sin problemas. En la tercera iteración de la prueba, ha llegado un momento en el que el robot no ha identificado un cruce correctamente por lo que el mapeo final ha sido incorrecto y no ha sabido alcanzar la meta del laberinto. Esto se debe a que el nivel de batería del robot había descendido en gran parte realizándolas dos primeras pruebas, y al disponer de poca energía los sensores de proximidad funcionaban peor por lo que ha terminado equivocándose.

Prueba 2: Mapeo con diferentes condiciones de luz

La siguiente ronda de pruebas, las hemos realizado con 3 condiciones de luz diferentes: Buena iluminación, iluminación intermedia y mala iluminación. El motivo de estas pruebas era el de comprobar hasta que punto es sensible la extracción de la ID de los landmarks ya que como hemos explicado en el apartado de visión artificial utiliza una segmentación de tipo Otsu, la cual funciona bien cuando la imagen esta uniformemente iluminada.

En la prueba con buena iluminación, no ha habido problemas y el robot ha mapeado y encontrado la salida sin ningún problema. En la prueba con iluminación intermedia, el robot ha confundido la ID de la penúltima baliza, lo cual no ha afectado a que encuentre el camino final a la meta porque ese cruce no lleva hasta ella, pero esa información ha sido errónea.

En la prueba con mala iluminación el robot ha confundido 2 landmarks y no ha podido leer otro correctamente.

Prueba 3: Mapeo general con otros robots mOway

Estas pruebas han sido igual que la primera, salvo que en cada una hemos utilizado distintos robots mOway para ver cuan adaptable es nuestro algoritmo a diferentes configuraciones de hardware, concretamente, a diferentes sensores de proximidad. En la primera prueba, el robot era demasiado sensible a los estímulos recibidos por los sensores frontales, y los recibidos por los sensores laterales eran demasiado débiles. En la segunda prueba con el segundo robot, el robot ha conseguido resolver el laberinto con alguna pequeña dificultad causada porque sus ruedas no estaban bien alineadas causándole que al avanzar en línea recta se desplazara hacia el lado izquierdo, lo que le ha supuesto un problema para centrarse del todo en los pasillos del laberinto. En la tercera prueba el robot confundía las intersecciones debido a que los valores que recibía por sus sensores eran muy aleatorios.

Prueba 4: Mapeo general desde distintos puntos del laberinto

En estas pruebas hemos decidido establecer distintos puntos de inicio y de final para comprobar si el robot puede mapear correctamente todo el laberinto independientemente de donde inicie a hacerlo y cual sea su destino final. Los resultados obtenidos han sido los siguientes:

En la primera prueba, el robot ha confundido el tipo de un cruce por lo que el mapeo ha sido erróneo y no ha podido alcanzar el final del laberinto. En la segunda prueba el robot ha logrado mapear y resolver el laberinto sin problemas y lo mismo ha sucedido con la tercera prueba.

Obviando el primer error, que ha sido un fallo que el robot puede cometer en cualquier momento, podemos deducir que el robot es capaz de mapear y recorrer el laberinto sin problemas independientemente de cual haya sido su punto de partida y dónde se encuentre la meta.

6.2 Análisis de los resultados obtenidos

Gracias a los resultados obtenidos en las pruebas es fácil llegar a una serie de conclusiones. La primera es que para que el algoritmo funcione correctamente y llegue a buena solución, depende de muchos factores físicos. Uno muy importante es el nivel de batería, ya que condiciona en gran medida los valores devueltos por los sensores de proximidad. Otro es la correcta iluminación de la habitación, ya que para que la segmentación de Otsu funcione adecuadamente son necesarias buenas condiciones de luz.

Debido a las diferencias presentadas en el uso de distintos robots mOways, se puede llegar a la conclusión que el éxito del algoritmo está muy condicionado por los sensores del robot y el correcto funcionamiento de éste. Esto es algo bastante obvio, ya que la mitad del algoritmo de navegación depende de los valores devueltos por los sensores para identificar los diferentes tipos de intersecciones, por lo que si no se pueden identificar correctamente, el robot no desempeñará la tarea que se le ha encomendado correctamente.

También se ha podido observar que la calidad del mapeo visual en todos los tándem de pruebas no es muy buena. Esto es debido a los problemas que he mencionado que tienen los diferentes sensores. En el apartado de conclusiones y mejoras se habla más sobre este tema y de porqué sería interesante mejorarlo en el proyecto.

7. Problemas con el proyecto

Durante el desarrollo de este proyecto, han surgido numerables problemas que no había sido capaz de prever a la hora de plantear el trabajo. Algunos han sido por desconocimiento del mundo que gira en torno alrededor del proyecto, ya que es la primera vez que me ha tocado trabajar con un robot en el mundo real. Otros han sido totalmente incontrolables por mi parte ya que son cosas que pueden pasar y resolverlas tienen un coste de tiempo importante. A continuación mencionare los principales problemas que he tenido durante el transcurso y la evolución del proyecto.

El primer contratiempo importante que tuve con el proyecto, fue a la hora de empezar a implementar la parte de procesamiento de imagen y visión artificial. Todo surgió de que por algún motivo, las imágenes del proyecto no se recibían correctamente en el ordenador y no me era posible realizar el procesamiento de imagen para extraer la información necesaria de los landmarks.

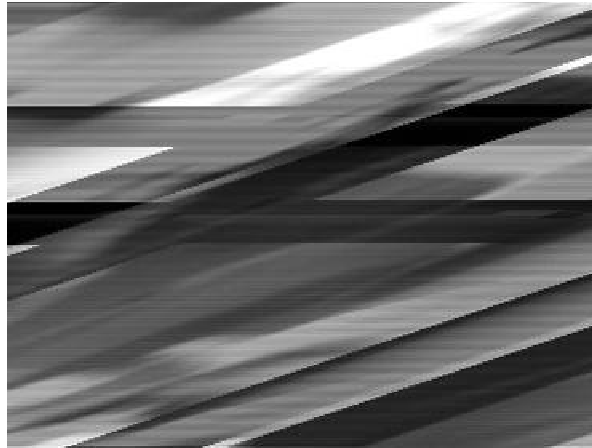


Figura 7.1: Imagen recibida por el robot

En un principio traté de solucionar el problema por mi cuenta buscando por internet y probando diversos métodos, pero el tiempo iba pasando y no conseguía resultados, así que decidí ponerme en contacto con la empresa mOway para pedirles ayuda. Tras un mes de mandarme emails con ellos, probando supuestas soluciones, haciendo citas virtuales con sus empleados con el *Team Viewer*, descubrimos que los drivers que estaban distribuyendo para su receptor de vídeo estaban mal, ya que su implementación era para un formato de vídeo NTSC mientras que la cámara emitía en PAL. Finalmente, me cedieron una versión actualizada de los drivers con los cuales la cámara funcionó sin problemas y pude continuar con el desarrollo del proyecto.

Otro gran problema que he tenido ha sido por cuestiones de hardware del robot mOway. Como he descrito a lo largo de la memoria, el robot dispone de unos sensores de infrarrojos con los cuales miden la distancia y la posición de los obstáculos. El problema viene a partir de que la calidad de estos sensores es pésima. Sus mediciones no son nada precisas, pueden variar de un robot mOway a otro, y no devuelven distancias si no un valor comprendido entre 0 y 99, siendo 0 que no detecta nada y 99 la distancia más próxima.

El alcance de estos sensores es muy escaso, no llega a los 10 centímetros. Un sensor no devuelve la misma cifra que otro sensor cuando ambos tienen un objeto a la misma distancia. Los valores que devuelvan dependen de demasiados factores del mundo en el que se encuentre el robot. Por ejemplo, la superficie en la que se proyectan los rayos infrarrojos es tremendamente determinante para que lo detecte a una mayor o menor distancia. Esto lo solucioné en parte construyendo el laberinto usando como material cartón-pluma, un material con un color blanco uniforme y con una buena refracción de la luz. Otro gran condicionador de la eficacia de estos sensores es el nivel de batería del propio robot. Un mismo sensor te devuelve 80 por ejemplo con un objeto a 2 centímetros de este cuando su batería es del 100%, pero cuando esta está al 70%, este

valor puede ser de 50. He perdido mucho tiempo tratando de acotar lo mejor posible estos valores para la detección de cruces, ya que, debido a la poca calidad de estos sensores, la mayoría de errores en el algoritmo que sufre el robot, es que reconoce el tipo de cruce en el que se encuentra de manera errónea por culpa de éstos.

La información recibida por los sensores de proximidad es muy imprecisa y resulta muy difícil medir distancias con exactitud. Este problema también ha afectado a la calidad del mapeado visual, que no es tan buena como hubiese deseado y aunque representa de manera bastante fiel la forma del laberinto, tiene mucho ruido que no debería aparecer lo que hace que aparente que el resultado sea malo.

También tengo que destacar que la duración de la batería me ha supuesto un límite importante en la velocidad de progresión y avance en el trabajo. A pesar de que en las especificaciones del robot indica que tiene una autonomía máxima de 2 horas, en mi caso, con 5 minutos de uso del robot para efectuar pruebas, el robot apenas disponía de batería y los valores devueltos por sus sensores de proximidad eran incorrectos, por lo que la navegación del robot era imposible. Esto implicaba, que aproximadamente por cada 5 minutos de pruebas reales en el laberinto con el robot, perdía 2 horas y media en las que tardaba en cargarse por completo.

En el transcurso del proyecto, se me estropeó el disco duro del ordenador con el cual estaba trabajando y perdí el proyecto. Por suerte, para evitar este tipo de problemas, todos los viernes hacía una copia de seguridad de lo que tenía realizado hasta el momento y lo subía a la nube, así que perdí algo menos de una semana de trabajo.

En general, no calculé el tamaño del proyecto de manera correcta y ha acabado siendo mucho mayor de lo que pensaba. Por esta razón, me he quedado corto de tiempo y no he podido implementar todas las funcionalidades que tenía en mente además de haber tenido que simplificar varios aspectos del proyecto. En el apartado de conclusiones y mejoras para el proyecto explicaré alguna de las ideas que iba a desarrollar en el robot.

8. Conclusiones y mejoras

Una de las posibles mejoras para aplicar a este proyecto es de hecho una idea que inicialmente se había planteado para incluir dentro de este. Trataría de utilizar el mapeado visual, cuya única funcionalidad actual es la de tratar mostrar una representación que se asemeje al laberinto que ha recorrido el robot, para sobre él aplicar un algoritmo de búsqueda A estrella. Así, en el caso de que existiese más de un posible camino hasta la meta, obtendríamos siempre el más corto, ya que utilizaríamos como heurística la distancia desde cada cruce hasta la meta.

Pero para poder realizar esto, primero deberíamos mejorar en gran medida la calidad del mapeado visual, ya que ahora sólo se asemeja hasta cierto grado a la realidad y para que los resultados obtenidos por el algoritmo A estrella fuesen los idóneos, habría de aplicarse sobre un mapa de calidad. Para lograr un mejor mapeado visual, podríamos probar a utilizar otro modelo de robot, uno cuyos sensores de proximidad fuesen mucho más precisos ya que , como he explicado en la sección de problemas, los del robot mOway dejan mucho que desear. Una buena opción sería usar algún robot con sensores de proximidad basados en la tecnología de ultrasonidos ya que unos sensores de este tipo de calidad estándar pueden medir distancias que rondan entre los 200 y 300 centímetros de alcance máximo, y lo que es más importante, con un error de +/- 1 milímetro.

También se podría tratar de mejorar la odometría del robot. Una opción sería el utilizar un filtro de partículas para hacer una mejor estimación de su posición en cada momento, pero habría que modificar y volver a plantearse de nuevo todo el sistema de landmarks, ya que necesitaría poder verlos en todo momento. Actualmente, esto no es posible ya que en nuestro caso, el robot solo puede ver los landmarks cada vez que alcanza un cruce.

Las conclusiones que he sacado personalmente durante el desarrollo de todo este proyecto son varias. Una de las más importantes es que me he dado cuenta que aplicar un modelo teórico en un ejemplo del mundo real es sumamente difícil. En los modelos teóricos todo es supuesto como el mejor de los casos, la situación es perfecta y no hay nada de ruido. En cambio, en un ejemplo real, nada es como debería ser y los problemas surgen en todo momento. Hay que tener en cuenta muchos elementos los cuales en el modelo teórico se simplifican o directamente se omiten, que hacen que una tarea que inicialmente parecía sencilla, pueda alcanzar un nivel de dificultad elevado.

El realizar este proyecto me ha supuesto una primera toma de contacto con el mundo de la robótica en el cual he podido aplicar de manera práctica diversos conocimientos sobre inteligencia artificial que he ido adquiriendo a lo largo de mi transcurso en la carrera.

9. Bibliografía

9.1 Bibliografía básica

- Manual usuario robot mOway
- Manual programación python mOway
- ANGELES, JORGE. Fundamentals of robotic mechanical systems : theory, methods and algorithms. New York : Springer, third edition (2007).
- USATEGUI ANGULO; MARÍA JOSÉ. Introducción a la robótica: principios teóricos, construcción y programación de un robot educativo. Madrid : International Thomson Editores Spain Paraninfo (2005).
- AYACHE, NICHOLAS. Artificial vision for mobile robots : stereo vision and multisensory perception. Cambridge (Mass.) ; London : MIT Press, (1991)

9.2 Bibliografía on-line

- Documentación OpenCv – http://www.opencv.org
- Documentación TKInter - https://wiki.python.org/moin/TkInter
- Aspectos de programación - https://stackoverflow.com/
- Información general - https://es.wikipedia.org/wiki/Wikipedia:Portada
- Udacity courses of AI - https://www.udacity.com/course/intro-to-artificial-intelligence--cs271