CAPÍTULO 5

Implementación

Una vez hecho el diseño del sistema, lo que sigue es implementar cada parte mencionada en este. Para la implementación se utilizó el lenguaje JAVA con el paquete jdk1.3 . Dada la naturaleza del lenguaje Java que es programación orientada a objetos, la representación de cada parte del diseño se implementó a manera de clases.

5.1 Implementación de la manipulación del robot.

Lo primero que se implementó fue una representación del robot. Lo que se quiere es que el proyecto funcione con una gran variedad de robots, por lo que se necesita generalizar el uso de los robots, como se mencionó en el diseño. Así que se creó una clase que representa a un robot.

La implementación del robot consiste de dos partes, una que contiene los movimientos y otra que maneja al sensor.

5.1.1 Robot

La clase que se creó para representar al robot se llama MyRobot.java y contiene los atributos y métodos básicos para que la manipulación del robot a través del algoritmo sea de la manera más sencilla.

5.1.1.1 Atributos

La clase MyRobot.java tiene los siguientes atributos:

- Sensor: será el medio de detección de obstáculos.
- Configuration: una configuración consta de una coordenada (X,Y) y una orientación. Esta configuración será el centro del robot.

5.1.1.2 Métodos

La clase MyRobot.java tiene los siguientes métodos principales:

- MyRobot(x,y): este es un constructor que inicializa los atributos y crea una instancia de MyRobot con centro en (x,y).
- Forward(distance): aquí se mueve al robot hacia delante una distancia “distance”.
- Backward(distance): sucede lo mismo que con Forward pero en sentido inverso.
- Left(angle): rota “angle” grados hacia la izquierda.
- Right(angle): rota “angle” grados hacia la derecha.

5.1.2 Sensor

La clase que representa al sensor del robot se llama Sensor.java y trata de manera sencilla manipular los datos provenientes del sensor para la creación de obstáculos. La implementación que se llevó a cabo fue simulando un sensor de contacto, ya que al hacer uso del robot LEGO no se pudo obtener una retroalimentación en tiempo real del robot a la computadora.

La clase Sensor.java tiene un método principal que es el siguiente:

- Contact(): Aquí se prueba si el sensor detecta algún objeto cercano al robot. La implementación directa de este método varía dependiendo del tipo de robot que se utilice y del tipo de sensores.

5.2 Implementación del algoritmo.

La implementación del algoritmo se realizó en dos clases principales que son:

- PathFinder.java, el cual se encarga de crear un "RoadMap" inicial con nodos libres de colisión.
- RoadMap.java, que se encarga de guardar los nodos iniciales y los secundarios.

5.2.1 PathFinder

En esta parte se crea un posible camino inicial que podría seguirse. El camino es un grafo que consta de nodos. Los nodos son creados libres de colisión de acuerdo a los obstáculos que se sabe existen; de haber obstáculos que no se tenga información, entonces se podrían crear nodos colisionando con dichos obstáculos pero que a la hora de intentar a esos nodos se eliminarían ya que se detectaría que existe algún obstáculo con el cual colisionan.

La clase PathFinder se utiliza únicamente al inicio de la exploración. Después se trabaja con RoadMap.

5.2.1.1 Atributos

Los atributos de PathFinder son:

- RoadMap: un objeto de tipo RoadMap en el cual se guardarán los nodos creados en PathFinder. En la clase RoadMap se manipulan los nodos y se crean nuevos nodos cuando es necesario.
- Width: el ancho del área a explorar.
- Height: lo largo del área a explorar.
- knownObstacles: los obstáculos que de alguna manera se tienen identificados al momento de iniciar la búsqueda.

5.2.1.2 Métodos

La clase PathFinder cuenta con un método principal:

- createRoadMap(): con este método se inicia la división del área total en regiones lo suficientemente pequeñas para suponer que el robot al estar en el centro de una región puede identificar lo que hay a su alrededor (obstáculos u objetivos). Si al crear un nodo en el centro de la región se encuentra en colisión, se procede a dividir la región en 9 regiones para tratar de reacomodar al nodo dentro de la región. Si aún así el nodo colisiona entonces esa región no podrá ser visitada.

Una vez creados todos los nodos se procede a conectarlos entre sí mediante el método RoadMap.conectNodes(), el cual conecta a cada nodo con los n nodos más cercanos posibles. El número de conexiones elegidas fueron 5.

5.2.2 RoadMap

La clase RoadMap representa un conjunto de vías interconectadas por las cuales el robot puede transitar. Si en algún momento dado no es posible alcanzar un nodo de manera directa, entonces se procede a realizar una re-planeación, la cual está implementada en esta clase.

5.2.2.1 Atributos

Los atributos de esta clase son:

- Nodes: Vector de nodos que conforman a este RoadMap. Cada nodo contiene una configuración (x,y,angulo) y una lista de los nodos a los que está conectado.
- actualNode y lastNode: el nodo actual y el anterior que se han visitado. Esto sirve a la hora de realizar una re-planeación ya que es necesario saber de dónde se viene y a dónde se quiere ir.

5.2.2.2 Métodos

Los métodos principales de esta clase son:

- conectNodes(nConections, obstacles): conecta a los nodos entre sí conectando cada nodo a los nConections nodos más cercanos comprobando que sea factible pasar de nodo a nodo sin colisionar con algún obstáculo dentro de los reconocidos en obstacles. Para cada nodo se calcula la distancia entre el nodo y los demás, después se procede a ordenar la lista de distancias para este nodo y de ahí se toman las nConections más cercanas. En caso de que no se pueda realizar la conexión a un nodo debido a que un obstáculo obstruye el camino, entonces el contador de nodos conectados no avanza, sino hasta que se haya logrado conectar a un nodo.

- conectNodeToRoadMap(nodo, nConections, obstacles): conecta un nodo dado al grafo. La conexión se lleva a cabo de la misma forma que en el método anterior.
- createNodesBetween(pPartida, pFinal): este es el método de re-planeación. Se crean nodos secundarios que ayuden a llegar del nodo pPartida al nodo pFinal. Esta situación se da porque la conexión entre nodos es mediante el trazo de líneas, si es posible trazar una línea entre nodos entonces se dice que éstos se pueden conectar; pero a la hora de realizar el avance con el robot resulta que por algún motivo el robot chocará con algún obstáculo, entonces se busca crear un camino nuevo para llegar al nodo destino. A continuación se ilustran los casos contemplados para llevar a cabo una re-planeación:

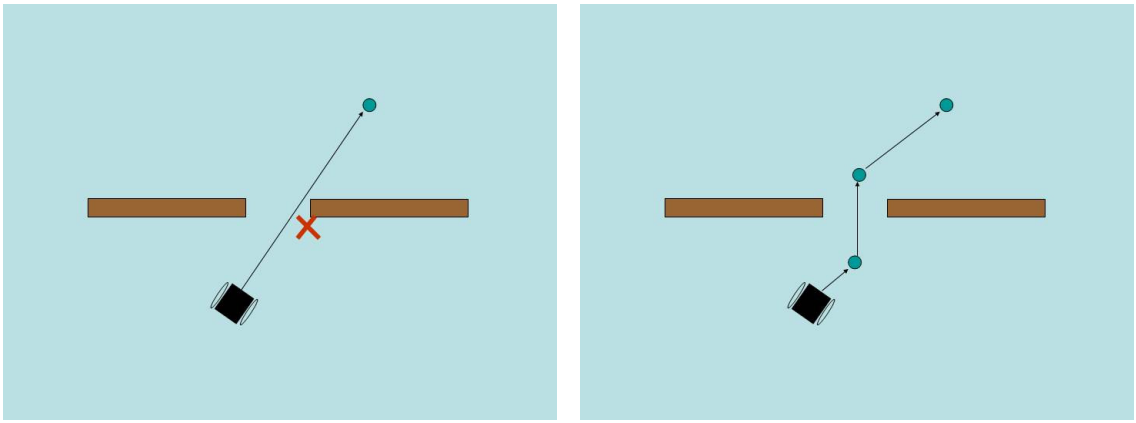

Fig. 5.1) y 5.2) Se tienen dos nodos unidos por una línea, pero a cuando se quiere hacer pasar al robot por este camino se ve que colisionará, en este caso la colisión es con un solo obstáculo.

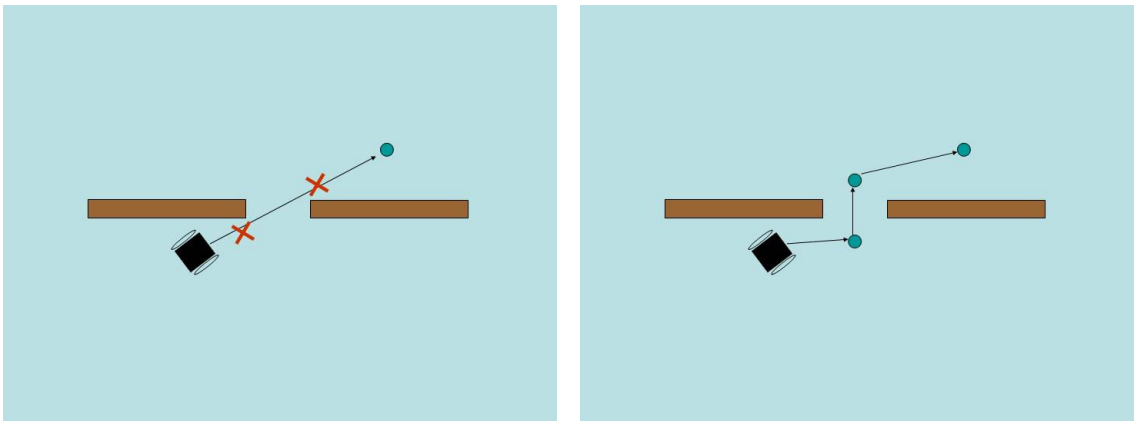

Fig. 5.3) y 5.4) En este caso la colisión es con dos obstáculos, la re-planeación nos dará una ruta en forma de "S".

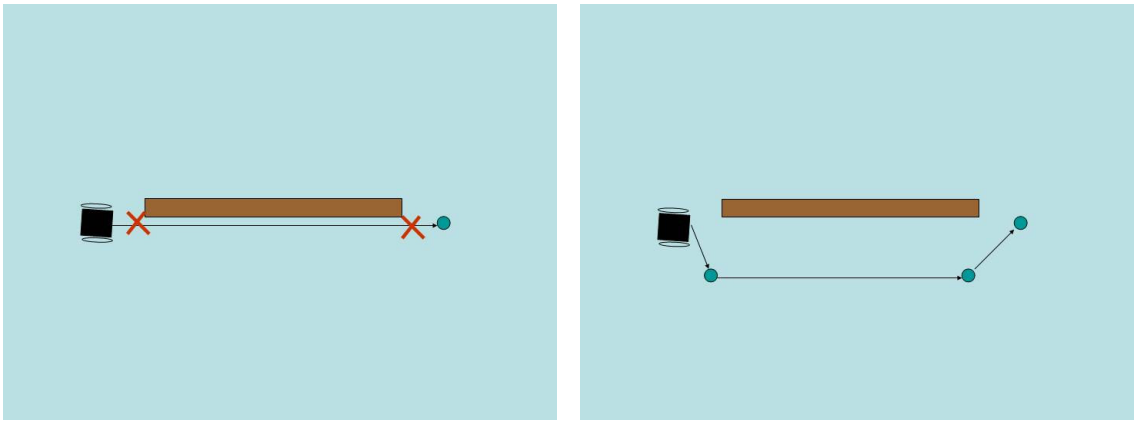

Fig. 5.5) y 5.6) La colisión es con un solo obstáculo pero es a lo largo del mismo. Para este caso se trata de alejar un poco al robot del obstáculo para poder pasar libremente en una línea paralela al obstáculo.

5.3 Implementación del cerebro.

El cerebro, como se mencionó en el diseño, se encargará de ser el intermediario entre algoritmo y robot, a la vez que realizará el mapeo necesario en el caso de detección de obstáculos. Aquí se llevarán a cabo las operaciones de análisis del área y toma de decisiones.

5.3.1 Brain.

La implementación del cerebro se hizo en una clase llamada Brain.java, la cual contiene los siguientes atributos y métodos.

5.3.1.1 Atributos

Los atributos de esta clase son:

- Robot: es una instancia de la clase MyRobot, y servirá para poder ordenar al robot las rutas decididas por el cerebro.
- knownObstacles: los obstáculos conocidos y a sea que el robot los haya identificado o hayan sido proporcionados desde el inicio de la búsqueda.

- newKnownObstacles: aquí se guardarán los obstáculos creados a partir de otros obstáculos, o sea los *bloques* reconocidos por el espacio estrecho que existe entre obstáculos y que impide que el robot pueda transitar entre ellos.
- PathFinder: Sirve para crear un grafo inicial libre de colisiones, a partir del cual se planeará la exploración.
- PATH: es un Vector de nodos que han sido visitados. Sirve para conocer la ruta que ha seguido el robot.
- RoadMapInicial: copia del RoadMap creado por PathFinder. De esta manera aquí se tendrán los nodos contemplados de manera inicial sin tomar en cuenta los nodos secundarios generados en la re-planeación. Servirá para poder comparar el RoadMap que se generó al inicio con el RoadMap final con rutas re-planeadas.

5.3.1.2 Métodos

Los métodos principales son:

- startSearch(): método para iniciar la búsqueda. Aquí se realizan las principales tomas de decisión como cuál será el próximo nodo a visitar y saber si se debe o no re-planear la ruta entre dos nodos.
- checkNeighbors(nodo): ante la detección de un nuevo obstáculo se debe checar si los nodos pueden seguir conectados o no. Esta operación se lleva a cabo por medio de este método.

- actualizaNodosLibres(): ante la detección de un obstáculo se hace una revisión de todos los nodos para saber si colisionan o no con el nuevo obstáculo, de estar en colisión el nodo en cuestión será desconectado del grafo para ya no intentar llegar a él.
- conectaNodos(nodo1, nodo2): aquí se busca un camino cualquiera entre dos nodos.
- optimizaUnion(ruta): el parámetro ruta es un Vector de nodos que representan el camino que se planea seguir. En este método se realiza una optimización de la ruta eliminando los nodos que no sean necesarios seguir y tratando de hacer la ruta lo más recta posible.
- goFromTo(from, to): checa si es posible llegar de una configuración "from" a otra "to" de manera directa, si no llama al método de re-planeación RoadMap.createNodesBetween(). En caso de encontrar algún obstáculo aquí se realiza el reconocimiento.
- joinObstacles(obstacle): crea *bloques* a partir de la distancia encontrada entre el obstáculo "obstacle" y los demás obstáculos. El *bloque* creado se agrega a newKnownObstacles.

5.4 Implementación de la visualización.

La parte gráfica del proyecto consiste de dos áreas de visualización. La primera contiene el mundo que será el real, y la segunda será la parte que conoce el robot. En la parte de la visualización del mundo del robot se podrán ver los nodos del RoadMap creado y los nodos que vayan surgiendo con la re-planeación, así como las conexiones existentes.

Existe un menú que presenta opciones básicas para poder manipular la exploración del robot. El menú se divide en cuatro categorías: Archivo, Búsqueda, Mapa y Opciones.

5.4.1 Menú Archivo.

En este menú se presentan 4 opciones:

- Nueva Búsqueda: inicializa las áreas de visualización y remueve los mapas generados por búsquedas anteriores.
- Abrir Mapa: carga un mapa guardado con anterioridad a la primera parte de visualización.
- Guardar Mapa 1: guarda en un archivo todo lo que se encuentre en la primera parte de visualización.
- Guardar Mapa 2: guarda el mapa explorado por el robot en un archivo.

5.4.1 Menú Búsqueda.

Este menú presenta 4 opciones:

- Iniciar Búsqueda: se hace inicio de la búsqueda-exploración en las condiciones en las que se encuentre el mapa del robot.
- Pausar Búsqueda: pausa la búsqueda-exploración para poder resumirla posteriormente.
- Continuar Búsqueda: resume la búsqueda-exploración pausada anteriormente.
- Detener Búsqueda: detiene la búsqueda por completo si dar opción a poder resumirla posteriormente.

5.4.2 Menú Mapa.

Este menú presenta 3 opciones:

- Ver Ruta Recorrida: se desplegará la ruta que el robot recorrió durante su búsqueda-exploración.
- Ver RoadMap: se despliega el RoadMap inicial.
- Dibujar Obstáculos: opción que sirve para crear un ambiente simulado que el robot pueda explorar.

5.4.3 Menú Opciones.

Este menú presenta 5 opciones:

- Búsqueda con Información: se refiere a que se cargará un mapa que servirá como guía para planear la búsqueda-exploración del robot. Los obstáculos contenidos en este mapa son los que conocerá el robot.
- Posición del Robot: sirve para dar las coordenadas (x,y) en las que se desea que el robot inicie su exploración.
- Ver Nodos Principales: visualizar únicamente los nodos que son necesarios visitar para considerar la búsqueda-exploración como satisfactoria.
- Ver Nodos Secundarios: visualizar los nodos creados a partir de las re-planeaciones.
- Ver Conexiones: despliega las conexiones entre nodos.

Con esto se concluye la parte de implementación, la cual fue planeada de manera tal que las futuras mejoras a este sistema comentadas en el siguiente capítulo sean fáciles de añadir.