ARQUITECTURA HÍBRIDA DE NAVEGACIÓN PARA ROBOT PIONEER PD3X

[*]Jorge Parra [**]Saúl Piña

Enero 2015

Resumen

En el presente informe se da a conocer de manera detallada el diseño, desarrollo e implementación de la arquitectura de navegación híbrida AuRA para el robot Pionner P3DX, con el objetivo de planificar rutas entre dos puntos cualesquiera en un ambiente conocido. La arquitectura construye el camino más corto aplicando el Algoritmo de Dijkstra sobre el diagrama de Voronoi asociado al entorno. Se segmentó el espacio a través de la Triangulación de Delaunay. Por otra parte, se explica el funcionamiento e instalación sobre sistemas operativos Windows o Linux de la aplicación de software desarrollada. Se exponen los resultados y conclusiones del conjunto de pruebas realizadas para validar la arquitectura. En los resultados alcanzados se encontró que el robot tiene la capacidad de dirigirse por el espacio efectivamente más no completamente eficiente. También se pudo observar que el robot responde satisfactoriamente a obstáculos inesperados.

Palabras clave: Robótica, ARIA, AuRA, Arquitectura Híbrida, Pioneer P3DX.

[*] *Decanato de Ciencias y Tecnología, Universidad Centroccidental Lisandro Alvarado, Barquisimeto, Venezuela, thejorgemylio@gmail.com*

[**] *Decanato de Ciencias y Tecnología, Universidad Centroccidental Lisandro Alvarado, Barquisimeto, Venezuela, sauljabin@gmail.com*

Introducción

La robótica móvil es un campo de investigación que supone la integración de numerosas disciplinas entre las que resaltan la electrónica, ingeniería mecánica, inteligencia artificial, estadística, entre otros. Tiene como objetivo guiar el curso de un robot móvil sobre un entorno con obstáculos conocido o desconocido, a través de un sistema que integra la percepción del entorno mediante sensores, toma de decisiones y acciones.

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno, la planificación de una trayectoria libre de obstáculos, y la conducción del vehículo a través de la referencia establecida. Es necesario que se incluya la capacidad de reaccionar ante situaciones inesperadas.

La arquitectura AuRA propone un enfoque de control donde se fusiona los paradigmas deliberativo y reactivo, integra todas las tareas expuestas anteriormente, por ende, le permitirá al robot móvil responder a eventos inesperados, desenvolverse en entornos no estructurados y a la vez cumplir con los objetivos de la misión. La implementación de esta arquitectura supone un conocimiento previo del entorno, el cual servirá como insumo para la planificación global de la misión.

En el presente trabajo se expone todo el proceso de implementación de la arquitectura híbrida AuRA sobre el robot móvil P3DX, con un razonador espacial principalmente basada en el diagrama de Voronoi y en el algoritmo voraz Dijkstra para el cálculo de la ruta óptima.

Pioneer P3DX

El Pionner P3DX (Figura 1) es un robot ligero ideal para ser usado en el interior de laboratorios o aulas de clases. El robot cuenta con una rueda y un motor ubicados a cada lado del mismo. Además, tiene una rueda en la parte posterior que le ayuda a mantener el equilibrio. Igualmente, el Pionner P3DX posee sensores de ultrasonido o sonares, empleados para detectar obstáculos en el ambiente. Es uno de los robots más popularmente usados en las áreas de la educación

e investigación. Puede ser fácilmente personalizado y mejorado, integrándolo con accesorios de diversa índole. Por otra parte, posee un conjunto completo de aplicaciones y librerías que sustentan el desarrollo de proyectos de investigación.

Figura 1: Robot Pioneer P3DX.

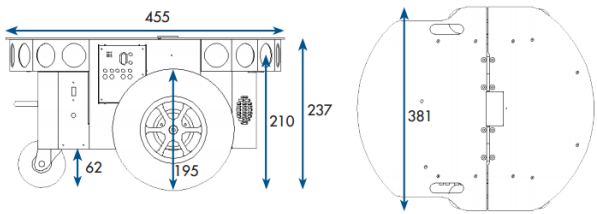


Figura 2: Dimensiones en milímetros del robot Pioneer P3DX.

Especificaciones del Pioneer P3DX

Construcción

Dimensiones: 45,5 cm de largo por 38,1 cm de ancho (Figura 2).
Cuerpo: 1,6 mm de aluminio.
Neumáticos: de caucho relleno de goma.

Operación

Peso del Robot: 9 kg.
Carga operativa: 17 kg.

Potencia

Tiempo de ejecución: 8-10 horas con 3 baterías.
Tiempo de carga: 12 horas o 2,4 horas.
Baterías: Soporta hasta 3 a la vez.
Tensión: 12 V.
Capacidad: 7,2 Ah.

Arquitectura AuRA

La arquitectura AuRA [1] (Autonomous Robot Architecture) introducida por Ronald Arkin, plantea un enfoque híbrido de navegación para robots móviles. Está constituida principalmente por dos capas, una deliberativa basada en la inteligencia artificial tradicional y otra reactiva basada en teoría de esquemas. En la Figura 3 se puede observar la representación del AuRA, la capa deliberativa la componen el *planificador de misión*, *razonador espacial* y el *secuenciador del plan*. En el nivel más alto se encuentra el *planificador de misión*, representa la interfaz con el humano y permite establecer los objetivos, las restricciones y los parámetros en los que debe operar el robot. El *razonador espacial* puede ser referido como navegador, este usa la información cartográfica almacenada para construir la ruta deseada que el robot debe seguir para completar la misión. El *secuenciador del plan* puede ser comparado con un piloto, convierte la ruta generada por el *razonador espacial* al conjunto de comportamientos (esquemas) que debe ejecutar la capa reactiva. En la capa reactiva, el *control de esquema* es el responsable de llevar a cabo y monitorear en tiempo real el conjunto de comportamientos que el secuenciador ordena ejecutar, cada comportamiento está asociado a un esquema perceptual, en otras palabras, cada estímulo genera una respuesta.

Fortalezas de la Arquitectura AuRA

Modularidad, flexibilidad, generalización e hibridación, son las características resaltantes de esta arquitectura.

Modularidad, es altamente modular debido a su diseño en capas,

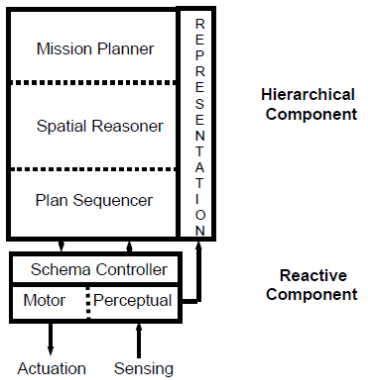


Figura 3: Representación de arquitectura AuRA.

lo que permite intercambiar o mejorar los componentes sin afectar el funcionamiento general.

Flexibilidad, esta arquitectura permite integrar diferentes modelos de inteligencia artificial para la construcción de rutas, aprendizaje y adaptación.

Generalización, puede ser empleado en distintos dominios como la manufactura, navegación, entre otros.

Hibridación, fusiona los paradigmas deliberativo y reactivo, lo que le permite al robot interactuar con eventos inesperados mientras intenta satisfacer sus metas.

Entorno de Desarrollo

Para llevar a cabo la presente investigación se desarrolló una aplicación cliente con las siguientes herramientas de software:

ARIA (Advanced Robot Interface for Applications) [2], es una librería desarrollada en el lenguaje de programación C++ para todas las plataformas MobileRobots/ActivMedia. Provee el conjunto de herramientas necesarias para controlar y monitorear de forma dinámica al robot y sus elementos. Además, esta librería incluye implementaciones llamadas *wrapper* en los lenguajes de programación Python y

Java, siendo este último el lenguaje de desarrollo para este investigación.

MobileSim [3], es un software para la simulación de plataformas MobileRobots/ActivMedia. Permite usar la librería ARIA y sus wrappers. Es un ambiente ideal para realizar experimentación y pruebas de los algoritmos desarrollados antes de implementarlos en los robots reales.

Eclipse IDE [4], es un entorno de desarrollo integrado para múltiples lenguajes de programación en los que destacan Java y C++. Provee las herramientas necesarias para el desarrollo, depuración y compilación de aplicaciones.

Java [5], es un lenguaje de programación de propósito general, orientado a objetos y de alto nivel.

Configuración del Entorno

Antes de configurar el entorno de desarrollo es necesario descargar el código fuente del proyecto, este está disponible al público con licencia de software **MIT** en la siguiente dirección:
`https://bitbucket.org/sauljabin/ariajava-p3dx`.

Requisitos

1. Sistema operativo de 32bits.
2. Java 7 o mayor.
3. Eclipse 4 o mayor.

Configuración del Entorno Sobre Windows

1. Instalar `MobileSim-0.7.2-1.exe`, disponible en:
   `http://robots.mobilerobots.com/wiki/MobileSim`.
2. Instalar `ARIA-2.7.6.exe`, disponible en:
   `http://robots.mobilerobots.com/wiki/ARIA`.
3. Agregar a la variable de entorno `PATH` la ruta:
   `C:\Program Files\MobileRobots\ARIA\bin\`.

4. Dirigirse a la ruta `C:\Program Files\MobileRobots\ARIA\bin\`, cambiar el nombre del archivo `AriaVC10.dll` a `Aria.dll`.

5. Importar proyecto a eclipse.

6. Agregar al `Java Build Path` el wrapper Java de ARIA que se encuentra en la ruta:
   `C:\Program Files\MobileRobots\ARIA\java\Aria.jar`.

7. Verificar que la variable `PATH_LIBARIA` del archivo `CONFIG.props` sea igual a: `C\:\\Program Files\\MobileRobots\\ARIA\\bin\\`.

Configuración del Entorno Sobre Debian/Ubuntu

1. Descargar el archivo `MobileSim-0.7.3+gcc4.6.tgz`, disponible en:
   `http://robots.mobilerobots.com/wiki/MobileSim`.

2. Ejecutar los siguientes comandos por consola para instalar MobileSim:

```
$ tar xzvf MobileSim-0.7.3+gcc4.6.tgz
# mv -f MobileSim-0.7.3 /usr/local/MobileSim
# ln -s -f /usr/local/MobileSim/MobileSim /usr/bin/mobilesim
```

3. Descargar el archivo `ARIA-2.8.1+gcc4.6.tgz`, disponible en:
   `http://robots.mobilerobots.com/wiki/ARIA`.

4. Ejecutar los siguientes comandos por consola para instalar ARIA:

```
$ tar xzvf ARIA-2.8.1+gcc4.6.tgz
# mv -f Aria-2.8.1 /usr/local/Aria
# echo '/usr/local/Aria/lib' >/etc/ld.so.conf.d/aria.conf
# ldconfig
```

5. Importar proyecto a eclipse.

6. Agregar al `Java Build Path` el wrapper Java de ARIA que se encuentra en la ruta: `/usr/local/Aria/java/Aria.jar`

7. Verificar que la variable `PATH_LIBARIA` del archivo `CONFIG.props` sea igual a: `/usr/local/Aria/lib/`

Implementación de la Arquitectura

Capa Deliberativa

Planificador de misión

Con el fin de planificar la misión, se desarrolló una interfaz gráfica intuitiva que permite configurar los parámetros generales, cargar la información cartográfica y además presentar la ruta óptima que deberá recorrer el robot (Figura 4).

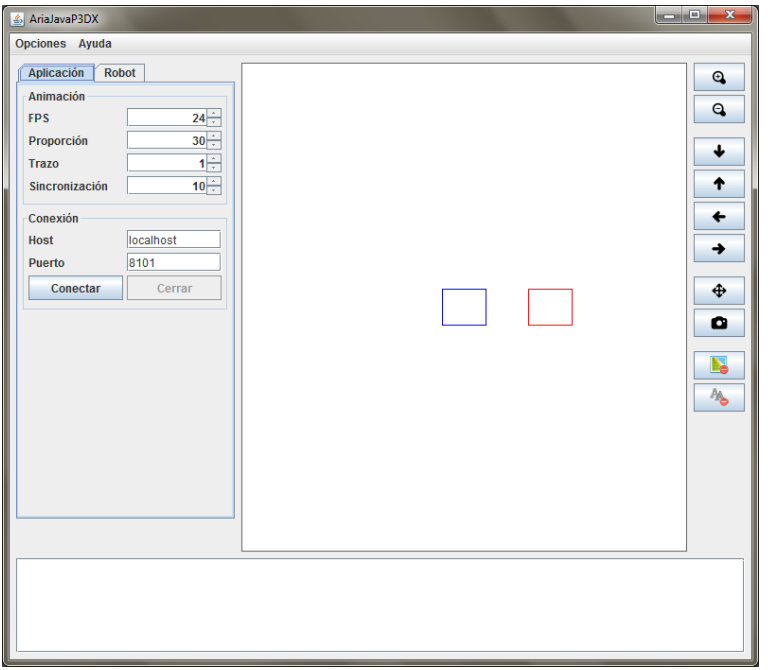


Figura 4: Planificador de misión.

Por otra parte, es posible cargar mapas a través de archivos de texto plano con extensión *.map*, este formato de archivo es compatible con MobileSim y puede ser generado o editado por Mapper3 [6].

En el Listado 1 se puede apreciar un ejemplo de como está estructurado un archivo *.map*, cabe destacar que en este formato las

longitudes son medidas en milímetros.

Listado 1: Ejemplo archivo mapa.

```
1  2D-Map
2  Cairn: RobotHome -5000 -5000 0 "" ICON "Home"
3  Cairn: Goal 5000 5000 0 "" ICON "Goal"
4  LineMinPos: -6000 -6000
5  LineMaxPos: 6000 6000
6  NumLines: 4
7  LINES
8  -6000 6000 6000 6000
9  -6000 -6000 6000 -6000
10 -6000 6000 -6000 -6000
11 6000 6000 6000 -6000
```

A continuación se describen las líneas que conforman el mapa:

Línea 1: `2D-Map`, indica el tipo de mapa.
Línea 2: `Cairn: RobotHome -5000 -5000 0 "" ICON "Home"`, representa la posición inicial del robot.
Línea 3: `Cairn: Goal 5000 5000 0 "" ICON "Goal"`, representa la posición objetivo.
Línea 4: `LineMinPos: -6000 -6000`, indica el punto de inicio del mapa.
Línea 5: `LineMaxPos: 6000 6000`, indica el punto fin del mapa.
Línea 6: `NumLines: 4`, indica la cantidad de líneas (obstáculos) que contiene el mapa.
Línea 7: `LINES`, indica el inicio de la sección de obstáculos.
A partir de la línea 8: sección de obstáculos.

La Figura 5 muestra el resultado del archivo *.map* descrito en el Listado 1, se puede observar el recuadro externo especificado desde la línea 7, el punto de inicio es representado por el rectángulo azul y el objetivo por el rectángulo rojo.

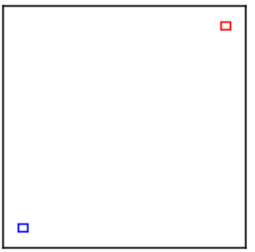


Figura 5: Resultado de ejemplo archivo mapa Listado 1.

Razonador Espacial

El razonador espacial es el encargado de construir la ruta que deberá recorrer el robot para llegar al objetivo. En esta implementación se diseñó un razonador espacial que calcula la ruta óptima mediante 3 etapas (Figura 6).

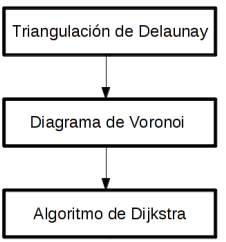


Figura 6: Etapas del razonador espacial.

Como se muestra en la Figura 6, en la primera etapa se realiza una segmentación del mapa a través de la **Triangulación de Delaunay**, esto divide el espacio en secciones pequeñas y conectadas. En segundo lugar se genera un **Diagrama de Voronoi** a partir de la triangulación, este diagrama representará el mapa en un grafo conectado. Por último, se aplica el **Algoritmo de Dijkstra** sobre el Diagrama de Voronoi con el fin de calcular la ruta óptima desde un punto de inicio hasta un punto objetivo dentro del grafo.

1. Triangulación de Delaunay

Una triangulación es la división de una superficie o polígono plano en un conjunto de triángulos, con la restricción de que cada lado del triángulo se reparta entre dos triángulos adyacentes [7].

La condición de Delaunay dice que una red de triángulos es una triangulación de Delaunay si todas las circunferencias circunscritas de todos los triángulos de la red son vacías, es decir, la circunferencia circunscrita de cada triángulo de la red no contiene otros vértices aparte de los tres que definen el triángulo. Esa es la definición original para espacios bidimensionales, y es posible ampliarla para espacios tridimensionales usando la esfera en vez de la circunferencia circunscrita. Esta condición asegura que los ángulos del interior de los triángulos son lo más grandes posibles, la longitud de los lados de los triángulos es mínima y la triangulación formada es única.

Por lo tanto, la triangulación de Delaunay es una red de triángulos que se caracteriza por formar los triángulos más equiláteros posibles, es decir, maximiza el mínimo ángulo de los triángulos. De esta forma, los puntos más próximos entre sí, estarán conectados por una arista, en la que los triángulos resultantes serán lo más regulares posibles [7, 8, 9, 10]. En la figura Figura 7 se puede observar un ejemplo de triangulación de Delaunay. Esta técnica tiene gran aplicación en la generación de gráficos en 3D por computadora.

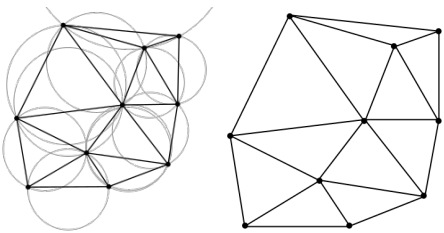


Figura 7: Ejemplo de Triangulación de Delaunay.

Las propiedades de la triangulación son las siguientes.

- La triangulación es unívoca si ningún círculo rodeado por los vértices de un triángulo contiene otros puntos del espacio.
- El ángulo mínimo dentro de todos los triángulos está maximizado y la longitud de los lados de los triángulos es mínima.
- Las aristas que pertenecen al perímetro de la triangulación conforman el cierre convexo de la nube de puntos. Estas aristas formarán parte de un solo triángulo, mientras que las aristas interiores a la triangulación van a formar parte de dos triángulos. Los triángulos que conforman el cierre convexo suelen ser alargados.

2. **Diagrama de Voronoi:**

   Un Diagrama de Voronoi de un conjunto de puntos en el plano, también conocido como teselaciones de Voronoi, no es más que la subdivisión del mismo en regiones formadas por los lugares más próximos a cada uno de los puntos [9, 10]. Estos objetos fueron estudiados por Dirichlet, Voronoi y Thiessen. El diagrama debe cumplir las siguientes propiedades:

   - El diagrama de Voronoi está compuesto fundamentalmente por puntos generadores, aristas y vértices.
   - Dos puntos pi y pj son vecinos si comparten una arista. Una arista es la bisectriz perpendicular del segmento pi , pj.
   - Un vértice es un punto equidistante a tres puntos y es la intersección de tres aristas.
   - Una región de Voronoi es un polígono convexo o es una región no acotada.
   - Una región de Voronoi es no acotada si su punto generador pertenece a la envolvente convexa de la nube de puntos.

- Dentro del círculo con centro en un vértice de Voronoi y que pasa por 3 puntos generadores no puede existir ningún otro punto generador.

Algunas de las aplicaciones del Diagrama de Voronoi son las siguietes: Posicionamiento de torretas en telefonía móvil. Control aéreo. Modelizador del crecimiento de bosques mediante diagramas de Voronoi. Reconstrucción de curvas. Empacamiento de objetos circulares.

La triangulación de Delaunay con todos los circuncentros es el grafo dual del diagrama de Voronoi: los circuncentros son los vértices de los segmentos del diagrama [11]. En la Figura 8 se observa la triangulación con todas las circunferencias circunscritas y sus centros. Conectando los centros de las circunferencias circunscritas se produce el diagrama de Voronoi (Figura 9).

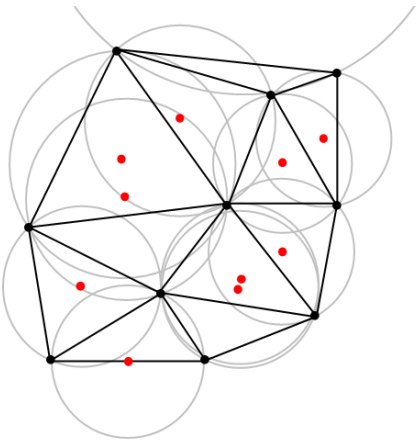


Figura 8: Ejemplo de Circuncentros de la Triangulación de Delaunay.

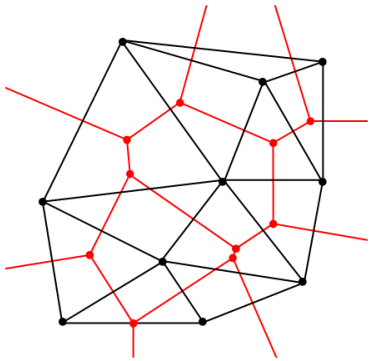


Figura 9: Ejemplo de Diagrama de Voronoi.

3. **Algoritmo de Dijkstra:** Este algoritmo fue diseñado por Edsger Dijkstra en 1959 [12, 13, 14], se puede categorizar en la familia de algoritmos voraces y posee una complejidad de $O(n^2)$. Tiene como objetivo hallar todos los caminos con costo mínimo desde un nodo inicial a todos los demás nodos dentro de un grafo ponderado. Básicamente lo que hace este algoritmo es recorrer los nodos adyacentes a un nodo específico, si encuentra un nodo con un alguna alternativa más óptima actualiza su peso y le indica cual es su nuevo nodo ancestro .

   Se explicará el algoritmo a través de su pseudocódigo (Algoritmo 1). Se tienen los siguietes elementos:

   - G: el grafo
   - Q: cola de prioridad
   - s: nodo inicial

   Se inicializan todos los pesos acumulados de los nodos en infinito y todos los nodos ancestros en nulo. Luego se agrega el nodo inicial a la cola de prioridad. Se procede a recorrer la cola de prioridad, mientras esta sea distinta de vacío se toma el primer elemento. Posteriormente, se verifica el peso acumulado de cada nodo adyacente al primer elemento en la cola, si el peso

acumulado del nodo actual del ciclo es mayor al peso del primer elemento (u) más el peso entre ambos nodos, se actualiza el peso acumulado y el nodo ancestro del nodo actual. Por último, se agrega a la cola de prioridad el nodo adyacente.

Algoritmo 1: Algoritmo de Dijkstra

```
Datos: Nodo inicial s
Resultado: Ruta óptima
Inicializar Q y G
para u ϵ G.getNodos() hacer
    u.setPesoAcumulado(INFINITO)
    u.setNodoAncestro(NULL)
s.setPesoAcumulado(0)
Q.adicionar(s)
mientras Q ≠ ∅ hacer
    u = Q.getPrimero()
    para v ϵ u.getNodosAdyacentes() hacer
        si v.getPesoAcumulado() >u.getPesoAcumulado() +
        G.getPesoEntreNodos(u,v) entonces
            v.setPesoAcumulado(u.getPesoAcumulado() +
            G.getPesoEntreNodos(u,v))
            v.setNodoAncestro(u)
            Q.adicionar(v)
```

Secuenciador del Plan

El secuenciador del plan es el encargado de dirigir el robot utilizando la información generada por el planificador. Tiene la responsabilidad de colocar al robot en el ángulo correcto para dirigirse a la siguiente meta, y además, le indica cuando moverse. Lleva el control de que metas han sido cumplidas y cuales no. En el Algoritmo 2 se observa el pseudocódigo de la implementación del secuenciador.

Algoritmo 2: Secuenciador del Plan

```
Point position = schema.getPosition()
double distance = position.distance(nextGoal)
double desiredAngle = calculateDesiredAngle(position,
nextGoal)
double angleTurn = desiredAngle - angle
si distance <RobotErrorDistance entonces
    schema.stop()
    nextGoal = goalControl.poll()
sinó, si Math.abs(angleTurn) >RobotErrorAngle entonces
    schema.turn(angleTurn)
sino
    schema.move(distance)
```

Capa Reactiva

Esquema de Percepción

El robot percibe en el entorno a través de los siguientes sensores:

Sonar: permite establecer la distancia entre su ubicación y los obstáculos cercanos. En la Figura 10, se observa la disposición del sonar.

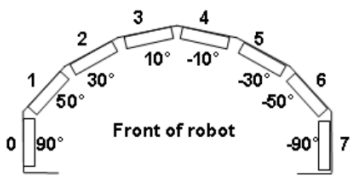


Figura 10: Disposición del sonar del robot.

Sensor Odométrico: permite estimar el cambio en la posición del robot a través del giro de las ruedas, utiliza como punto de referencia el punto de inicio.

Esquema de Control

El esquema de control son el conjunto de comportamientos integrados que permiten controlar el robot.

El esquema de percepción y de control implementados se pueden apreciar en el Listado 2.

Listado 2: Implementación de Esquemas.

```
1  public class ArSchemaController {
2     private ArRobotMobile robot;
3
4     public ArSchemaController(ArRobotMobile robot) {
5        this.robot = robot;
6     }
7
8     public void turn(double angle) {
9        robot.setDeltaHeading(angle);
10    }
11
12    public void move(double speed) {
13       robot.setVel(speed);
14    }
15
16    public void stop() {
17       robot.setVel(0);
18    }
19
20    public Point getPosition() {
21       Point point = new Point(robot.getX(), robot.getY(), "");
22       return point;
23    }
24
25    public double getAngle() {
26       double angle = robot.getTh();
27       return angle;
28    }
29
30    public void sleep(int sleepTime) {
31       ArUtil.sleep(sleepTime);
32    }
33
34    public double rangeObstacle(double startAngle, double endAngle) {
35       double range = robot.getRangeSonar().currentReadingPolar(startAngle, endAngle);
36       return range;
37    }
38
39    public boolean detectObstacle(int distance, double angle) {
40       double range = robot.getRangeSonar().currentReadingPolar(-angle, angle);
41       return range < distance;
42    }
43
44 }
```

Herramienta de Software Desarrollada

Se desarrolló una herramienta computacional llamada **AriaJavaP3DX**, que provee los elementos gráficos necesarios para configurar

y monitorear el comportamiento del robot Pioneer P3DX.

Como se muestra en las Figuras 11 y 12, la ventana principal está compuesta por 8 elementos básicos:

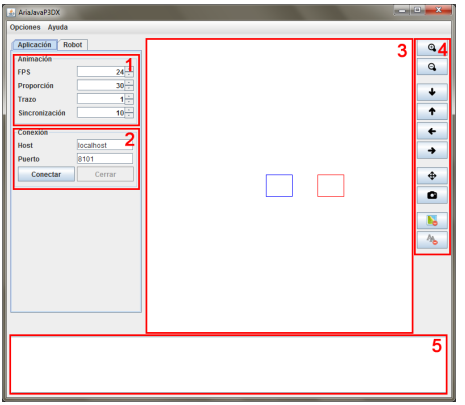


Figura 11: Partes de la ventana principal.

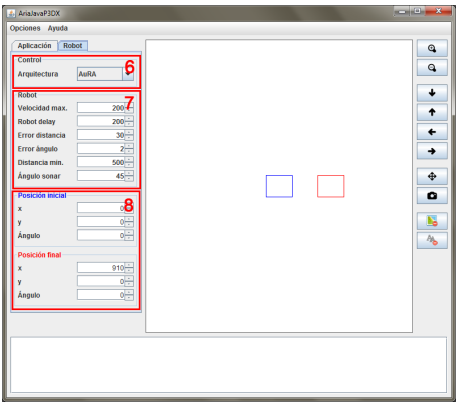


Figura 12: Partes de la ventana principal.

1. Esta sección permite configurar la animación a través de los siguientes items: **FPS**, cuadros por segundo (en inglés Frames per second), se refiere a la frecuencia en que se actualiza la

imagen de la animación. **Proporción**, es la proporción del mapa en la animación con respecto a la medida original en milímetros. **Trazo**, es el grosor del trazado de las imágenes en la animación. **Sincronización**, es la frecuencia en segundos en que se verifica la posición real del robot y se actualiza la posición del robot animado.

2. En el panel **Conexión** se establece la dirección IP y el puerto de conexión con el robot.

3. En la región central de la ventana se localiza la animación, se puede observar un ejemplo en la Figura 15.

4. En el lateral derecho de la ventana se encuentran una serie de botones que permiten manipular el mapa. En primer lugar se puede hacer zoom sobre el mapa (Figura 13a). Además se puede mover la ubicación del mapa hacia los 4 puntos cardinales (Figura 13b). Por otra parte, se puede centrar el mapa (Figura 13c) y hacer una captura de imagen, al presionar el botón de captura de imagen (ícono cámara fotográfica) se muestra la ventana de guardar imagen (Figura 14). Los últimos botones (Figura 13d) permiten mostrar u ocultar la ruta óptima calculada y activar o desactivar el antialiasing.

5. En la parte inferior de la ventana se encuentra la consola, permite observar los mensajes que se generan en la ejecución de la aplicación.

6. En este panel se puede establecer la arquitectura de control del robot. En el ítem **Arquitectura**, se puede elegir entre la arquitectura AuRA o un comportamiento solamente reactivo.

7. En este panel se manipula el conjunto de variables que permiten calibrar el comportamiento del robot, estas son:

   a) **Velocidad max.:** velocidad máxima alcanzada por el robot.

   b) **Robot delay:** tiempo de espera entre acciones.

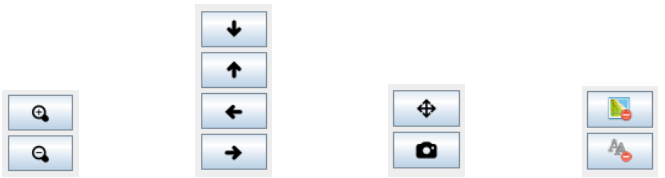
c) **Error distancia:** distancia máxima permitida entre el robot y los objetivos.

d) **Error ángulo:** diferencia máxima permitida entre el ángulo del robot y el ángulo deseado.

e) **Distancia min:** distancia mínima tolerada entre el robot y un obstáculo.

f) **Ángulo sonar:** rango de detección del sonar.

8. Esta última sección está destinada a asignar la posición de inicio y fin del robot.



(a) Zoom **(b)** Mover **(c)** Capturar **(d)** Ocultar

Figura 13: Acciones para manipulación del mapa.

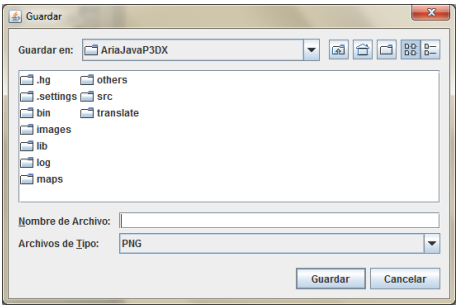


Figura 14: Ventana para guardar captura del mapa.

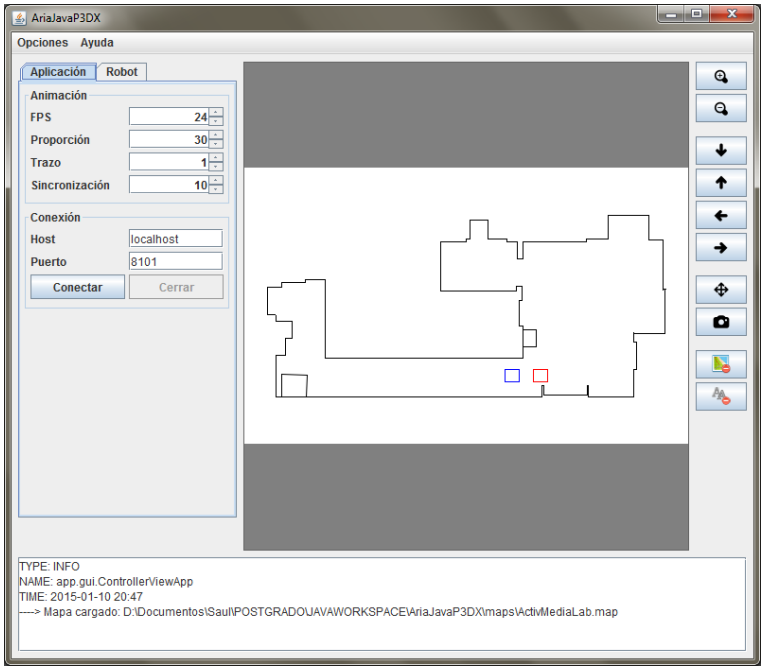


Figura 15: Mapa cargado.

Menú Opciones

A través de este menú (Figura 16) se puede seleccionar y cargar un mapa ingresando a la opción **Cargar Mapa**, esta, desplegará una ventana de búsqueda (Figura 17). La opción **Ver configuración** permite visualizar la lista de valores de la configuración general de la aplicación (Figura 18). Por último la opción cerrar permite terminar la aplicación.

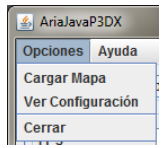


Figura 16: Menú opciones.

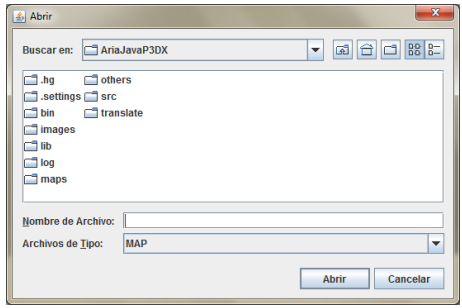


Figura 17: Ventana para seleccionar mapa.

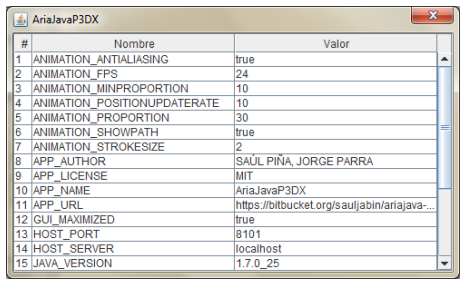


Figura 18: Ventana para ver los atributos de configuración.

Menú Ayuda

A través del menú ayuda (Figura 19) se puede acceder a la documentación de la aplicación, además se puede ingresar a la ventana de créditos en la opción **Sobre AriaJavaP3DX** (Figura 20).

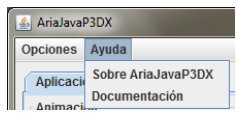


Figura 19: Menú Ayuda.

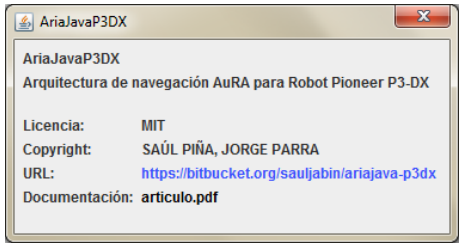


Figura 20: Ventana para ver la información de la aplicación.

Experimentación

Para el proceso de experimentación se empleó la herramienta de simulación MobileSim, y se digitalizó el plano del Edificio K del Decanato de Ciencias y Tecnología de la Universidad Centroccidental Lisandro Alvarado. Se diseñaron dos experimentos con la finalidad de observar el comportamiento exhibido por el robot simulado frente a un escenario con y sin obstáculos, además detectar la configuración ideal para la navegación. Los experimentos consisten en ejecutar seis veces el algoritmo configurando los parámetros de navegación con diferentes valores.

La información espacial para los experimentos se cargó en un archivo *.map*, el resultado puede observarse en las Figuras 21 y 22.

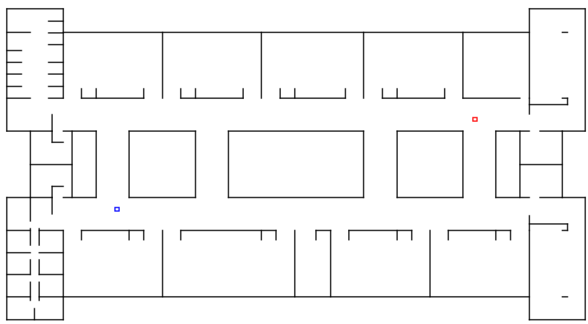


Figura 21: Mapa del Módulo K.

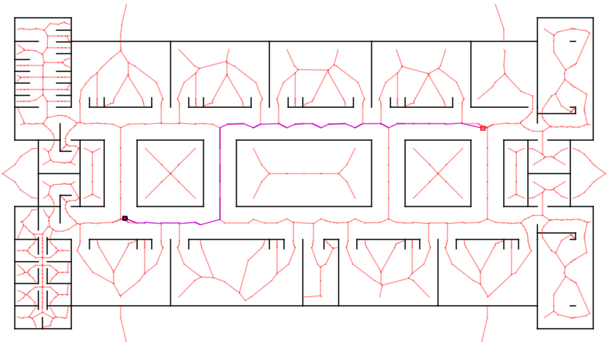


Figura 22: Mapa del Módulo K con Ruta de Navegación.

Los atributos configurables para la navegación son los siguientes:
Punto inicio: punto origen de navegación.
Punto objetivo: objetivo que el robot debe alcanzar.
Error distancia: distancia máxima permitida entre el robot y los objetivos.
Error ángulo: diferencia máxima permitida entre el ángulo del robot y el ángulo deseado.
Distancia mínima: distancia tolerada entre el robot y un obstáculo.
Ángulo sonar: rango de detección del sonar.

Experimento I, Navegación sin Obstáculos

En el Cuadro 1 se puede apreciar los resultados obtenidos. La tasa de éxito fue de 100 %.

Cuadro 1: Tabla de Resultados Experimento Sin Obstáculos

Nro	Punto Inicio	Punto Objetivo	Error Distancia	Error Ángulo	Éxito	Tiempo
1	(12000,12000)	(50950,21800)	30	2	Si	419,17
2	(48645,24047)	(56392,23638)	30	2	Si	417,29
3	(9221,23000)	(54431,10791)	40	4	Si	455,21
4	(8000,11000)	(40431,16000)	40	4	Si	310,5
5	(39432,20500)	(8800,11000)	60	6	Si	317,64
6	(54432,11500)	(8800,23000)	60	6	Si	429,77

Experimento II, Navegación con Obstáculos

Para el experimento con obstáculos se configuraron las pruebas con las siguientes tolerancias de error: distancia de 30mm y ángulo 2 grados. De este experimento (Cuadro 2) se obtuvo una tasa de éxito de 66,6 %. Se puede interpretar del resultado que distancias de aproximación menores o igual a 300mm no son ideales para el algoritmo de navegación desarrollado.

Cuadro 2: Tabla de Resultados Experimento Con Obstáculos

Nro	Punto Inicio	Punto Objetivo	Distancia Mínima	Ángulo Sonar	Éxito	Tiempo
1	(12000,12000)	(50950,21800)	500	45	Si	540,17
2	(48645,24047)	(56392,23638)	500	55	Si	130,44
3	(9221,23000)	(54431,10791)	400	45	Si	529,08
4	(8000,11000)	(40431,16000)	400	55	Si	383,89
5	(39432,20500)	(8800,11000)	300	45	No	
6	(54432,11500)	(8800,23000)	300	55	No	

Conclusión

La arquitectua AuRA es altamente modular por su diseño, integra la capa reactiva con la capa deliberativa de manera que el robot móvil pueda desenvolverse en entornos no estructurados y a la vez cumplir con los objetivos de la misión. Los resultados presentados en esta investigación muestran que una combinación entre AuRA y un razonador espacial basado en los diagramas de Voronoi genera buenas rutas entre el el punto de partida y el objetivo, más se observó que no son del todo eficientes. El algoritmo voraz Dijkstra cumplió con las expectativas con respecto al tiempo de ejecución, se pudo apreciar que es necesario poco tiempo para determinar las rutas óptimas sobre los grafos, lo que ayuda a minimizar el tiempo de cálculo en el construcción de nuevos caminos al encontrar obstáculos.

Con respecto al desempeño general de la implementación hecha, se puede decir que los resultados fueron buenos, en el experimento realizado sin obstáculos la tasa de éxito fue de 100 %. Por su parte, el experimento con obstáculos obtuvo una calificación de 66,6 %, debido a que dos de las pruebas realizadas resultaron fallidas. La distancia de aproximación mínima de las pruebas erradas fue de 300 milímetros, lo

que sugiere que esta distancia es muy pequeña para el algoritmo desarrollado, por ende se toma como distancia de aproximación deseable 500 milímetros.

Referencias

[1] Arkin, R. y Balch, T. (1997). AuRA: Principles and practice in review. Journal of Experimental & Theoretical Artificial Intelligence. Taylor & Francis. Vol. 9. No. 2-3. Pp. 175-189.

[2] MobileRobots. (2014). ARIA. Disponible en: http://robots.mobilerobots.com/wiki/ARIA.

[3] MobileRobots. (2014). MobileSim. Disponible en: http://robots.mobilerobots.com/wiki/MobileSim.

[4] Eclipse. (2014). Eclipse IDE. Disponible en: https://www.eclipse.org/ide/.

[5] Oracle. (2014). Java. Disponible en: http://www.oracle.com/technetwork/java/index.html.

[6] MobileRobots. (2014). Mapper3. Disponible en: http://robots.mobilerobots.com/wiki/Mapper3.

[7] Triangulación de Delaunay. Disponible en: http://www.dma.fi.upm.es/mabellanas/bigdelone/FrameConceptos.htm

[8] Triangulación de Delaunay. Disponible en: http://www.dma.fi.upm.es/mabellanas/voronoi/delone/delone.html

[9] De Berg, M., Van Kreveld, M., Overmars, M. y Schwarzkopf, O. (2000). Computational geometry. Springer Berlin Heidelberg. Pp. 1-17.

[10] Okabe, A., Boots, B., Sugihara, K. y Chiu, S. (2009). Spatial tessellations: concepts and applications of Voronoi diagrams. John Wiley Sons. Vol. 501.

[11] Wikipedia. Triangulación de Delaunay. Disponible en:
http://es.wikipedia.org/wiki/TriangulaciÃşn_de_Delaunay

[12] Sniedovich, M. (2006). Dijkstra's algorithm revisited: the dynamic programming connexion. Control and cybernetics. Vol. 35. No. 3. Pp. 599-620.

[13] Lavalle, S. (2006). Planning algorithms. Cambridge university press.

[14] Cormen, T., Leiserson, C., Rivest, R. y Stein, C. (2001). Introduction to algorithms. Cambridge: MIT press.