VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"SISTEMA DE NAVEGACIÓN PARA INVIDENTES UTILIZANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL Y FUSIÓN SENSORIAL"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRÍA EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

LUIS FRANCISCO RAMÓN SANABRA SERNA

DIRECTOR DE TESIS: DR. JESÚS SAVAGE CARMONA

México, D.F. **2009.**

SISTEMA DE NAVEGACIÓN PARA INVIDENTES UTILIZANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL Y FUSIÓN SENSORIAL

PRESENTA:
LUIS FRANCISCO RAMÓN SANABRA SERNA
DIRECTOR DE TESIS:
DR. JESÚS SAVAGE CARMONA

27 de julio de 2009

"Sólo por que un hombre no tenga vista en sus ojos, no significa que no tenga visión"

Stevie Wonder

AGRADECIMIENTOS

A mis padres *José Francisco Sanabra Marroquín* y *María de Jesús Serna López* y a mis hermanas *Diana* y *Jazmine*. Por todo el apoyo que me han brindado a distancia durante estos años, desde el inicio de mi carrera hasta la fecha.

A mi director de tesis, el *Dr. Jesús Savage Carmona*. Por el gran apoyo que he recibido de su parte desde el inicio del posgrado y por la confianza que demostró depositar en mí; y quien, más que un tutor, se convirtió en un invaluable amigo.

A mis colegas, los ingenieros Baudel Calvillo, Luis Díaz e Isaac Velázquez del Instituto Tecnológico de Tijuana. Con quienes inicié este interés y esta pasión por aportar un granito de arena a una comunidad tan lamentablemente marginida como lo es la gente con capacidades diferentes; razón por la cual decidí tomar un tema dirigido a las personas invidentes.

A los sinodales: *Dra. Ana Lilia Concepción Laureano Cruces*, *Mat. María Concepción Ana Luisa Solís González Cosío*, *Dr. Sergio Marcellin Jacques* y el *Dr. Boris Escalante Ramírez*. Por haberse tomado el tiempo y dedicación de revisar minuciosamente el presente trabajo y cuyos comentarios y sugerencias resultaron valiosos para la buena conclusión del mismo.

A mis compañeros y amigos del posgrado. Por su ánimo y apoyo incondicional a lo largo de este periodo.

RESUMEN

El presente trabajo presenta el diseño del prototipo de un sistema orientado a guiar a personas invidentes a desplazarse fácilmente y sin ayuda externa por entornos desconocidos.

Para lograr esto, el sistema implementa un sistema de visión con 3 cámaras simultáneas, utilizando ARToolKit Plus, el cual es capaz de detectar marcas, previamente colocadas en el entorno en una posición bidimensional predeterminada, y calcular la distancia de cada cámara con respecto a una marca; de esta manera utilizando una técnica de localización por triangulación se calcula la posición de la persona. Para guiar al invidente se tiene un mapa topológico del ambiente; el cual, utilizando teoría de árboles K-Dimensionales (árboles-KD) se encuentra el nodo más cercano al invidente y utilizando el algoritmo de Dijkstra se encuentra la ruta más corta desde este nodo al nodo destino que se pretende alcanzar por el invidente. Una vez el invidente habiendo iniciado su curso hacia el nodo destino, el sistema es capaz de guiarlo en todo momento, basándose en los sistemas GPS, para esto, se cuenta con una brújula digital que es capaz de saber la orientación del invidente y de esta manera calcular la dirección del ángulo en relación al nodo destino y evitar así que la persona se desvíe. Tanto la localización por triangulación y la corrección de ángulos se calculan en todo momento para asegurar que la persona invidente alcanzará su destino final.

Palabras clave: Triangulación, Invidentes, ARToolKit Plus, Dijkstra, Árboles KD, Corrección de Curso, Mapa Topológico.

Índice general

Lista de algoritmos

Índice de figuras

Índice de cuadros

INTRODUCCIÓN

Vivimos en un mundo donde la tecnología está avanzando cada día a pasos agigantados; desde la invención de la electricidad, el mundo ha cambiado y revolucionado tecnológicamente. Desde que ir de un continente a otro resultaba una travesía de meses, hasta hoy en día que sólo toma unas pocas horas.

Ante estos avances tecnológicos, muchos aprovechados para bien, otros no, han ayudado a la humanidad a tener un estilo de vida más sencillo, para algunos dependiente, pero más comoda sin lugar a dudas.

Sin embargo, vemos con lástima, que estos avances tecnológicos siempre van orientados a lo que resulte atractivo para el mercado; aquello que no resulte en fuertes ganancias económicas para las grandes corporaciones, será difícilmente de interés.

El mercado tecnológico orientado a personas con capacidades diferentes no es, sin lugar a dudas, la excepción. Estas personas han sido, hasta cierto punto, marginadas de todo avance tecnológico con el que se cuenta actualmente. Ha sido con muchos esfuerzos que algunos gobiernos han optado por invertir parte del gasto público en herramientas tecnológicas orientadas a este tipo de personas.

Es por esto que, como trabajo de tesis, se ha decidido desarrollar un sistema orientado a la ayuda de personas invidentes. Como una manera de realizar un aporte a una sociedad tan marginada que merece también ser atendida.

El presente trabajo lleva por título "Sistema de Navegación para Invidentes utilizando Técnicas de Inteligencia Artificial y Fusión Sensorial". Sistema de Navegación para Invidentes, como resulta altamente intuitivo su nombre, es debido a que se ha desarrollado un sistema de guía para la libre navegación de un invidente, esto es, que una persona invidente sea capaz de, desde el lugar donde se encuentra, llegar a cualquier destino en un ambiente semi controlado. Utilizando las bases de los sistemas de localización GPS, pero sin la infraestructura altamente costosa de éstos; donde en lugar de satélites que indiquen la distancia y posición del receptor, se tengan marcas de ARToolKit Plus las cuales el receptor sabrá sus distancias y por lo tanto, con únicamente información local, podrá obtener su localización precisa. Utiliza Técnicas de Inteligencia Artificial como lo son el algoritmo de Dijkstra para encontrar el camino más corto entre todos los puntos del mapa del entorno embebido en el sistema, así como búsquedas

en árboles bidimensionales para encontrar el vecino más cercano, esto es, el nodo del mapa más cercano al que se encuentra el invidente en un determinado momento. Por último, Fusión Sensorial, ya que el sistema estará constituido por varios sensores; se tendrá una brújula digital capaz de medir la orientación del invidente con respecto al norte, así como tres cámaras capaces de detectar patrones defindos en el entorno; y una salida auditiva que será la salida del sistema hacia el usuario.

La presente tesis empieza en su primer capítulo definiendo el planteamiento del problema, los objetivos que persigue el proyecto, así como la metodología a utilizar y las hipótesis planteadas.

El segundo capítulo contiene un enfoque humano de lo que es un invidente y la perspectiva de ellos hacia el mundo que los rodea y una breve explicación de la importancia que es en estos tiempos el atender la problemática de las personas con capacidades diferentes; seguido de algunos conceptos básicos, así como términos y definiciones que se estarán manejando durante el desarrollo de esta tesis; posteriormente, en dicho capítulo, se muestra información sobre la magnitud de la discapacidad visual a nivel mundial y su distribución, de manera tal que se vea el impacto que traerá el presente trabajo y los beneficios que aportará a esta comunidad. También contiene información sobre lo que existe en la actualidad, como herramientas que ayuden a la persona invidente a desenvolverse en su vida cotidiana y como el presente trabajo aportará mayor significado a lo anteriormente descrito.

El tercer capítulo se refiere a la arquitectura del sistema que se propone en esta tesis, las herramientas que serán utilizadas, algoritmos y demás componentes, así como una explicación de la razón de uso.

El desarrollo del proyecto, es decir, la programación de interfaces con los dispositivos involucrados, la implementación de los algoritmos, etc. serán vistos en el cuarto capítulo de esta tesis.

En el capítulo 5 se encuentran las pruebas realizadas y resultados obtenidos; estas pruebas son realizadas con usuarios en un entorno semi controlado a fin de analizar el comportamiento de éstos y así mismo se presentan los ajustes correspondientes que tuvieron que ser realizados.

En el capítulo 6 y último, se describen las conclusiones que se encontraron en la realización de este proyecto; así como trabajos a futuro y mejoras que se considerarían importantes para el mismo.

Capítulo 1

PLANTEAMIENTO DEL PROBLEMA

1.1. OBJETIVOS GENERALES

Proponer y diseñar un sistema de navegación y localización para invidentes, utilizando tecnología de visión por computadora y otras herramientas de hardware, que les permita la libre navegación y ubicación por cualquier entorno (desconocido por el invidente pero conocido por el sistema) en el que se encuentran . De manera tal que puedan mejorar su calidad de vida y su autonomía.

1.2. OBJETIVOS PARTICULARES

- Establecer un dispositivo de navegación para evidentes eficaz y de uso cotidiano.
- Aplicar técnicas de búsqueda y clasificación de información que permitan realizar el proceso de representación de medio ambiente de una manera eficaz y rápida.
- Mejorar la calidad de vida y autonomía de las personas invidentes.
- Otorgar un aporte significativo de la tecnología al servicio de personas con capacidades diferentes.
- Despertar el interés de otros estudiantes de licenciatura y postgrado en la investigación y desarrollo de proyectos de tecnología computacional para un área tan poco atendida en México como es la de los invidentes u otras discapacidades.

1.3. CONTRIBUCIÓN Y RELEVANCIA:

La tecnología orientada a la ayuda de personas con capacidades diferentes ha sido poco atendida o no muy atractiva para los grandes corporativos tecnológicos.

La realización de este proyecto contribuirá al desarrollo de dispositivos tecnológicos orientados a personas con capacidades diferentes, utilizando algoritmos de inteligencia artificial, tales como los algoritmos de Dijkstra y Árboles KD para encontrar el vecino más cercano, basados en métrica euclidiana, orientado a la clasificación y agrupamiento de grandes cantidades de información bi-dimensional.

1.4. METODOLOGÍA

- Se utilizarán herramientas de realidad aumentada y visión por computadora que puedan detectar patrones y distancias de los mismos con respecto a las cámaras.
- Se utilizarán técnicas de inteligencia artificial que permitan la rápida localización y navegación por un entorno.
- Se analizará el comportamiento habitual de las personas invidentes, detectando las necesidades que se presentan en ellos, de manera tal que el proyecto cumpla con sus necesidades en un criterio amplio.
- Se estudiará el resultado en cada persona para analizar el impacto y alcance que tiene el sistema en los usuarios.

1.5. HIPÓTESIS

Teniendo como preámbulo los objetivos que se desean conseguir y la metodología utilizada para ello; se plantean varias hipótesis que se buscarán satisfacer durante el proceso del presente trabajo.

- Diversas técnicas de localización y navegación empleadas en robots, pueden ser adaptadas para emplearse en el desarrollo de herramientas para la ayuda de personas con capacidades diferentes.
- Puede desarrollarse un sistema de localización y navegación a partir de marcas en el ambiente.
- Este sistema puede ser económico, utilizando componentes ya existentes de software y hardware gratuito ó relativamente económicos.
- Los sistemas de Realidad Aumentada, capaces de detectar cierto tipos de marca con visión, son una buena herramienta para utilizar en el presente trabajo.

- Una persona invidente es capaz de llegar a cualquier destino rápidamente si se tiene un sistema que lo esté guiando en todo momento.
- Puede tenerse un sistema completamente audio/táctil que sea utilizable por una persona invidente, permiténdole en todo momento la entrada y salida de información.

Capítulo 2

ESTADO DEL ARTE

2.1. EL INVIDENTE

"Resulta trivial decir que el mundo de una persona invidente es en sí un mundo que está desprovisto de visión, de luz, de color, es un mundo en el que la información transmitida por otros sentidos cobra una vital y esencial importancia. Dada una estimación, el vidente recibe alrededor del 85 % de la información a través del canal visual, este tipo de información es globalizada y se realiza a una velocidad considerable, eso hace comprender, entonces, que su atención se dirija selectivamente hacia el análisis de estos estímulos visuales, obviando la información que recibe de otras vías sensoriales" [Nuñ07].

De esta manera, analizando el caso de un invidente, podemos entender que es considerablemente distinto. Con la ausencia del sentido visual; las sensaciones auditivas, olfativas, táctiles y térmicas pasan a ocupar un lugar preeminente en su experiencia sensorial. Su experiencia sensorial del mundo es, por tanto, cualitativamente diferente. En lugar de ser un mundo de sonidos, olores, texturas, temperaturas, donde la información la recibe a través de la actividad de su propio cuerpo y a través de la información verbal. En estos dominios sensoriales, la cantidad y calidad de información que se recibe es significativamente diferente. Existen muchas nociones, para los videntes comunes, que carecen totalmente de significación para los invidentes. Un caso típico es el color, sin embargo no es el único; puede pensarse en otras nociones, como la perspectiva (representación grafica de líneas paralelas que se unen en el infinito) cuya captación, para el dominio sensorial del tacto, es muy compleja o prácticamente imposible. Pero eso no es todo, de igual manera existen fenómenos naturales difícilmente accesibles a la experiencia sensorial directa: el vuelo de los pájaros, el movimiento de los peces, los paisajes. Un caso práctico es la idea que puede formarse un invidente de las otras personas con las que se relaciona; cuando su percepción de ellas tiene que estructurarla en base a su voz o al contacto fugaz al estrecharle la mano. Más difícil aún sería indagar como serán sus sueños.

La diferencia entre un vidente y un invidente en ese sentido, radica en que el vidente integra las informaciones de los otros sentidos formándose una imagen visual de las experiencias, mientras que en el invidente, esta imagen visual está ausente, pero no la imagen mental.

Por tanto, con base en lo anterior, el presente proyecto tiene como fin diseñar un sistema orientado a auxiliar a personas invidentes con el desempeño de sus actividades cotidianas, y por lo tanto, mejorar significativamente su calidad de vida.

2.2. ANTECEDENTES

A fin de desarrollar el presente proyecto, se debe tener una base de conocimientos generales y conceptuales sobre la problemática que se aborda, dicha base de conocimiento se presenta a continuación:

2.2.1. LA CEGUERA

La ceguera es la pérdida del sentido de la vista. La ceguera puede ser total o parcial; existen varios tipos de ceguera parcial dependiendo del grado y tipo de pérdida de visión, como la visión reducida, el escotoma, la ceguera parcial (de un ojo) o el daltonismo.

2.2.2. TIPOS DE CEGUERA

La ceguera se clasifica dependiendo de donde se ha producido el daño que impide la visión. Éste puede ser en:

- Las estructuras transparente del ojo, como las cataratas y la opacidad de la córnea.
- La retina, como la degeneración macular asociada a la edad y la retinosis pigmentaria.
- El nervio óptico, como el glaucoma o la diabetes.
- El cerebro.

2.2.3. CAUSAS DE LA CEGUERA

La ceguera puede ser causada por [WHO09c]:

2.2.3.1. Enfermedades

De acuerdo con la estimación de la OMS en el 2002 , las causas más comunes de ceguera alrededor del mundo son:

- Cataratas
- Glaucoma
- Uveítis
- Degeneración macular
- Opacidad corneal
- Tracoma
- Retinopatía diabética
- Anormalidades y daños

Los accidentes, especialmente en los menores de 30 años, hacen perder la vista generalmente en uno de los ojos. Personas con daños en el lóbulo occipital, a pesar de tener intactos los ojos y nervios ópticos, tendrían ceguera parcial o total.

2.2.3.2. Defectos Genéticos

- Las personas con albinismo usualmente sufren de deterioro a la vista extendido al grado de ceguera parcial, aunque pocos presentan ceguera total.
- La amaurosis congénita de Leber puede causar total o gran pérdida de visión desde el nacimiento o la infancia.
- Aniridia. Falta congénita del iris del ojo.
- Recientes descubrimientos en el genoma humano han identificado otras causas genéticas de baja visión o ceguera. Una de ellas es el síndrome de Bardet-Biedl.

2.2.3.3. Envenenamiento

Ciertos químicos como el metano, encontrado en el alcohol metílico, utilizado por alcohólicos como sustituto económico de la bebida alcohólica.

2.2.3.4. Otros

La malnutrición junto a las enfermedades son las causantes principales de la ceguera

2.3. RAZONES Y JUSTIFICACIONES

2.3.1. Magnitud de la Discapacidad Visual

Las estimaciones desde los años 90's, así como nuevos datos basados en la población global en el 2002 muestran una reducción en el número de gente quien es invidente o visualmente discapacitada, y quienes son invidentes por efecto de enfermedades infecciosas, pero un incremento en el número de gente que es invidente por condiciones relacionadas a un lapso de vida largo [WHO09a].

- Globalmente, en el 2002 más de 161 millones de personas eran discapacitadas visualmente, de las cuales 124 millones tenían visión deficiente y 37 millones eran invidentes. Sin embargo, el error refractivo como causa de una discapacidad visual no fue incluida, lo cual implica que la magnitud global actual de discapacidad visual es mayor.
- A nivel mundial por cada persona invidente, un promedio de 3.4 personas tienen visión deficiente, con variaciones regionales en el rango de 2.4 a 5.5.
- Estas estadísticas son las estimaciones científicas mejor alcanzables de la carga global de discapacidad visual y son el resultado de nuevos estudios realizados en todas las regiones pertenecientes a la Organización Mundial de la Salud.

2.3.2. Distribución de la Discapacidad Visual

De acuerdo con [WHO09b], la distribución de la discapacidad visual es la siguiente:

- Por edad: La discapacidad visual no está igualmente distribuida a través de los distintos grupos de edad. Más del 82 % de toda la gente quien es invidente tienen 50 años o más, aunque ellos representan solo el 19 % de la población mundial. Debido al número esperado de años vividos en la ceguera, la ceguera infantil, con un estimado de 1.4 millones de niños invidentes menores a 15 años.
- Por género: Estudios indican consistentemente que en cada región del mundo, y en todas las edades, las mujeres tienen, significativamente, un riesgo más alto de ser discapacitadas visuales que los hombres.

	Región Africana	Región de las Américas	Región del Este Mediterraneo	Región Europea	Región Sudeste de Asia	Región del Pacífico Oeste	Total
Población	672.2	852.6	502.8	877.9	1,590.80	1,717.50	6,213.90
# gente invidente	6.8	2.4	4	2.7	11.6	9.3	36.9
% de ceguera total	18 %	7 %	11 %	7 %	32 %	25 %	100 %
# con visión deficiente	20	13.1	12.4	12.8	33.5	32.5	124.3
# con deficiencia visual	26.8	15.5	16.5	15.5	45.1	41.8	161.2

Cuadro 2.1: Habitantes con problemas visuales en el mundo[WHO09b]

- Geográficamente: La discapacidad visual no está distribuida uniformemente a lo largo del mundo. Más del 90 % de los discapacitados visuales en el mundo viven en países en vías de desarrollo.

2.4. LA TECNOLOGÍA EN LA DISCAPACIDAD VISUAL

En la actualidad, la tecnología ha mejorado notablemente la calidad de vida de las personas con discapacidad visual, la invención de las computadoras y el desarrollo a pasos agigantados que ha tenido el Internet, favorece el acceso a la información por personas invidentes quienes pueden contar con software desarrollados para adaptar los sitemas a las necesidades del invidente. Los siguientes son algunos de los aportes de la tecnología en el tema de la discapacidad visual.

2.4.1. Computadoras e Internet

2.4.1.1. Magnificadores de Imagen

Esta es una herramienta especialmente útil para las personas con baja visión, debido a que permite ver el contenido de la pantalla gracias a la amplificación de las partes seleccionadas de la imagen, es un proceso rápido y bastante efectivo [dPPR08].

2.4.1.2. Lectores de Pantalla

Son aplicaciones de software las cuales permiten el acceso al texto que existe en la computadora, esta herramienta identifica aquello que se muestra en pantalla y lo que representa lo transmite al usuario mediante sintetizadores de texto a voz, íconos sonoros, o una salida braille [dPPR08].

2.4.1.3. Impresoras Braille

Estas impresoras reflejan la información mediante líneas de pines que sobresalen de una superficie, formando los diferentes signos. En el caso de la impresora Braille, los pines se insertan en una placa con orificios, dejando en medio el papel con los puntos remarcados en alto relieve, el cual, luego es sacado renglón a renglón, mediante mecanismos de tracción o empuje, de modo que al salir del papel refleja la información que ha sido enviada por la computadora en el sistema de lecto-escritura Braille[dPPR08].

2.4.1.4. Ayudas para permitir la Lectura

Reconocimiento Óptico de Caracteres Es una herramienta muy útil para que la persona con déficit visual acceda a documentos escritos, esto se hace a través de un escáner que se une a un sistema de programas de reconocimiento óptico de carácteres y un sintetizador de voz que lee el contenido del documento [dPPR08].

Libros Digitales Son discos grabados por un locutor del Sistema digital de información accesible (Daisy, por sus siglas en inglés), permite que quien lo lee ponga marcas para resaltar lo que le llama la atención, indicar el lugar donde suspendió la lectura o devolverse para repasar alguna línea

Funcionan con un reproductor especial con el que se puede modificar el tono, el volumen y la velocidad de lectura del sintetizador de voz. En el mundo hay una biblioteca de unos seis mil títulos en este formato [dPPR08].

2.4.1.5. Tecnología para Discapacitados en la Ciudad

Semáforos Sonoros Son semáforos adaptados tecnológicamente con sonidos que permiten que la persona con limitación visual reconozca auditivamente en que momento debe cruzar la calle y en que momento detenerse [dPPR08].

Señalización en Braille Se refiere al hecho de que las edificaciones de la ciudad y del servicio al público cuenten con señalización braille en diferentes lugares para que la persona con discapacidad visual pueda orientarse, cada día son más comunes los ascensores que cuentan con información en Braille, facilitando así el uso autónomo de este servicio [dPPR08].

2.5. HERRAMIENTAS TRADICIONALES PARA LA AYUDA DEL DISCAPACITADO VISUAL

2.5.1. Bastón Blanco

Es una especie de vara hecha de aluminio (o algún otro material), ya sea fija o flexible, el cual es de color blanco con extremo inferior rojo, a fin de distinguir fácilmente a una persona invidente y otorgarle prioridad de paso [BRA08].

2.5.2. Perros Guía

Para el entrenamiento de perro-guía, son usados en su mayoría la raza Pastor Alemán, Labrador Retriever y Golden Retriever, aunque también se han entrenado, pero muy escasamente, Collies, Dálmatas, Doberman y varias otras. A la fecha existen varias escuelas de perros guía para invidentes en países como Sudáfrica, Alemania, Australia, Bélgica, Canadá, España, Estados Unidos de Norteamérica, Francia, Holanda, Inglaterra, Irlanda, Israel, Italia, Japón, Noruega, Suiza, y otros más. Estos países también cuentan con la legislación adecuada, además de otros muchos países que sin tener una escuela de perros-guía, tienen la suficiente conciencia humana como para permitirles el paso [I.A08].

Capítulo 3

ARQUITECTURA DEL SISTEMA

Actualmente, los avances tecnológicos nos permiten tener al alcance sistemas de visión en tiempo real utilizados generalmente en la Robótica o en sistemas para Realidad Aumentada. Partiendo del principio de que un robot, como tal, no puede ver; se necesitan sistemas adicionales que le permitan la libre navegación por cualquier entorno en el que se encuentre.

Han sido desarrollados a lo largo de las décadas, muchos sistemas a fines y muy variados: Se habla de sensores sonares y campos potenciales, basados en el comportamiento natural de ciertos animales (como el murciélago y la hormiga, respectivamente); visión por computadora, que utilizan técnicas de procesamiento digital de imágenes, entre otros.

Es este último (visión por computadora) el que resulta de interés para el presente tema de tesis. Donde, utilizando sistemas ya existentes de visión por computadora a través de cámaras digitales, mapas topologicos, planeadores de rutas y algoritmos de localización; se utilizará dicha tecnología para desarrollar un sistema integral que sea capaz de orientar a la persona invidente a desenvolverse libremente en un entorno desconocido para él, pero previamente conocido para el sistema .

La Figura 3.1, explica de una manera gráfica como está constituido el sistema. El sistema tiene un componente de localización, el cual, utilizando visión por computadora y una brújula digital, podrá determinar la posición y orientación exacta del invidente en un entorno conocido por el sistema.

De esta manera, el invidente solicitará al sistema, por medio de un teclado braille, las instrucciones a seguir para ir de un punto a otro (ej.: del punto en donde se encuentra hacia los baños, de la taquilla del metro a los andenes); antes de iniciar cualquier proceso, el sistema debe saber en donde se encuentra actualmente el invidente, esto es posible gracias al módulo de localización.

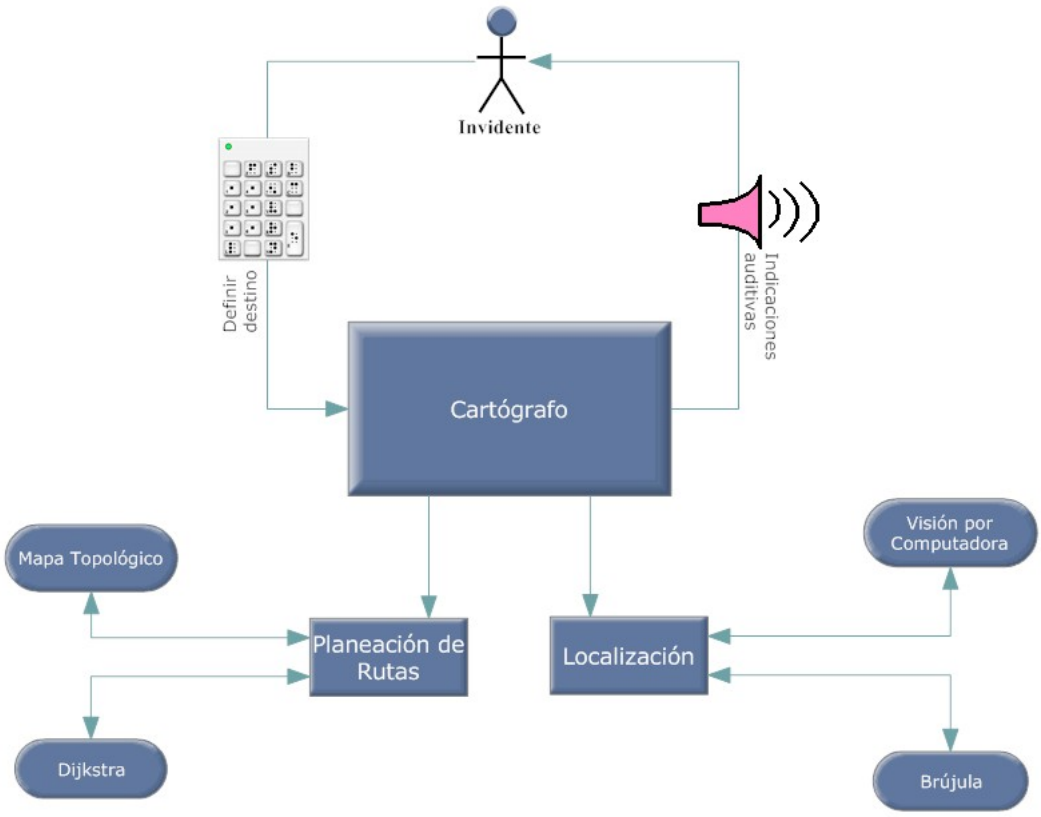


Figura 3.1: Diagrama de Bloques del funcionamiento del sistema

Posteriormente, el módulo de planeación de rutas entrará en proceso para calcular, con el mapa topólogico del lugar y usando el algoritmo de Dijkstra, la mejor ruta a seguir para llegar al destino deseado.

Es importante mencionar que, de acuerdo a que el proyecto está orientado a personas con discapacidad visual, no puede pensarse en un sistema con salida visual, como es común en la mayoría de los sistemas, para esto, se tendrá la siguiente forma de salida auditiva. De esta manera el sistema le guiará mediante señales de audio y voz la ruta que debe de seguir para llegar a dicho destino.

3.1. VISIÓN POR COMPUTADORA

La visión por computadora ha crecido rápidamente a lo largo de los últimos años, produciendo herramientas que permiten el entendimiento de la información visual, especialmente para escenas que no vengan acompañadas con información estructurada o descriptiva. La meta principal de las investigaciones realizadas sobre visión por computadora es proveer a la computadora de habilidades de percepción visual de manera tal que puedan ser capaces de sensar el ambiente, entender los datos sensados, tomar las acciones apropiadas y aprender de esta experiencia para mejorar el desempeño a futuro.

3.1.1. Realidad Aumentada

Los términos realidad virtual y ciberespacio se han vuelto muy populares en las comunidades de investigación en las últimas dos décadas. La población, en su mayoría, asocia estos términos con la posibilidad tecnológica de sumergirse dentro de un mundo completamente sintético y generado por una computadora; algunas veces también es utilizado el término ambiente virtual. En un ambiente virtual, nuestros sentidos, tal como la vista, el olfato, el tacto y el sentido auditivo son controlados por una computadora, mientras nuestras acciones influyen a producir los estímulos virtuales.

Algunos autores definen la Realidad Aumentada como un caso especial de la Realidad Virtual; mientras que otros autores, por el contrario, argumentan que la Realidad Aumentada es un concepto más general y ven a la Realidad Virtual como un caso especial de ésta [BR05].

El hecho es que, en contraste con la Realidad Virtual tradicional, en la Realidad Aumentada el ambiente real no está completamente suprimido; por el contrario, éste juega un papel sumamente importante. En lugar de sumergir a la persona en un mundo completamente sintético, la Realidad Aumentada embebe suplementos sintéticos en el ambiente real.

Por lo tanto, en base a lo anterior, puede obtenerse por concepto de Realidad Aumentada como un campo de investigación computacional el cual se basa en la

combinación del mundo real y datos generados por computadora. Actualmente, la mayoría de las investigaciones de Realidad Aumentada están relacionadas con el uso de imágenes de video en vivo el cual es digitalmente procesado y "aumentado" mediante técnicas de graficación por computadora.

3.1.2. ArtoolKit

ARToolKit es una librería de software desarrollada en C y C++ la cual permite a los programadores desarrollar aplicaciones de Realidad Aumentada. Dicha librería está siendo desarrollada en University of Washington HIT Lab [KB99].

Una de las partes más complejas en el desarrollo de Realidad Aumentada es el calcular en tiempo real el punto de vista de tal manera que las imágenes virtuales que se embeberán estén exactamente alineadas con los objetos del mundo real. ARToolKit utiliza técnicas de visión por computadora para calcular la posición y orientación real de la cámara y la posición y orientación relativa de las marcas (las cuales se hablará detalladamente en el próximo tema); de tal manera que se permita al programador dibujar los objetos virtuales sobre dichas marcas.

3.1.3. Landmarks (Marcas)

Las aplicaciones hechas en ARToolKit permiten que imágenes virtuales sean sobrepuestas sobre la escena del mundo real. La clave para que esto sea posible de manera eficiente es usar cuadrados negros con determinado patrón, el cual se definirá como marcas.

La Figura 3.2 muestra tres ejemplos de diseños o dibujos que pueden ser utilizados como una marca válida por ARToolKit; para que una marca sea válida, solo debe de cumplir con dos reglas: a) ser cuadrada y b) el cuadrado debe ser negro y blanco, el color negro debe ser el primer color a partir de los bordes. Lo resaltante de este tipo de marcas es que puede ser utilizado cualquier diseño en ella, desde simples líneas rectas sin sentido, hasta caracteres en otros idiomas o símbolos, incluso logotipos[KB99].

Figura 3.2: Ejemplos de figuras que pueden ser utilizadas como una marca válida y reconocida por ARToolKit

3.1.4. Funcionamiento de ArtoolKit

El modo de operación de ARToolKit es el siguiente[KB99]:

Algorithm 3.1 Modo de operación de ARToolKit

- La cámara captura video del mundo real y lo envía a la computadora.
- El sistema busca, en cada cuadro del video, marcas válidas.
- Si una marca es encontrada, el sistema realiza ciertas operaciones matemáticas para calcular la posición de la cámara relativa a la marca encontrada.
- Una vez que la posición es obtenida, se dibuja algún modelo gráfico desde la misma posición.
- El modelo es dibujado en la parte superior del video del mundo real de manera tal que parece estar fijado a la marca cuadrada.
- El video "aumentado" se muestra como salida en pantalla, de manera tal que cuando es observado por el usuario, puede apreciar modelos gráficos adheridos al mundo real.

La Figura 3.3 muestra el flujo del funcionamiento de ARToolKit para reconocer una marca y dibujar en ella un modelo gráfico. Es importante señalar que, para el presente tema de tesis, resulta irrelevante la salida virtualmente aumentada del sistema; lo que es de gran relevancia es la capacidad que tiene ARToolKit de encontrar la posición y orientación que existe entre la cámara y la marca que se está observando[KB99].

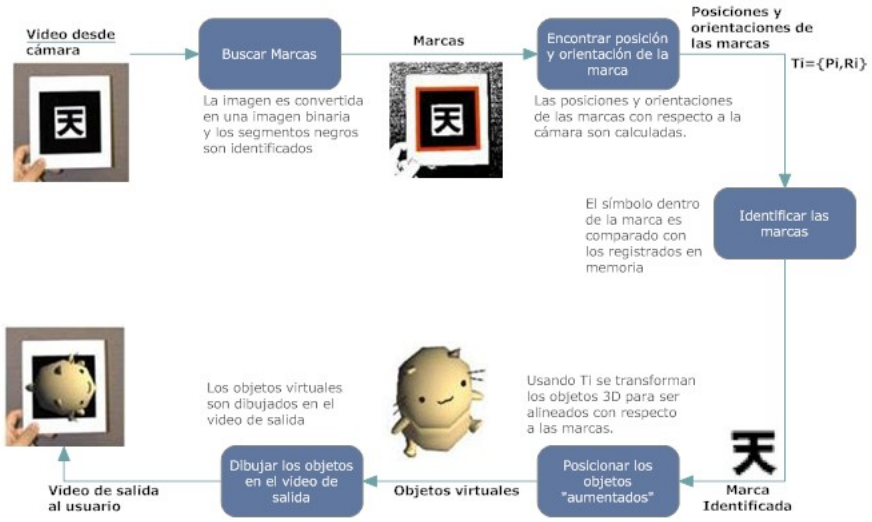


Figura 3.3: Flujo del funcionamiento interno de ARToolKit

Las características más resaltantes que aporta ARToolKit y por lo que se ha decidido utilizarlo en el presente trabajo, son las siguientes[KB99]:

- Un framework sencillo para la creación de aplicaciones de realidad aumentada.
- Librería multiplataforma (Windows, Linux, Mac OS X, SGI)
- Soporte para otros lenguajes (JAVA, Matlab).
- Rutinas sencillas para calibración de cámaras.
- Seguimiento de la posición y orientación de la cámara con respecto a las marcas.
- Posibilidad de utilizar cualquier marca cuadrada en blanco y negro como patrón.
- Tiempo de respuesta inmediato.

Entre las múltiples características con las que cuenta este sistema, aquellas que resultan de mayor importancia son el poder tener de manera precisa y rápida la posición y orientación que existe entre la cámara y una marca que se encuentre en cualquier lugar en el entorno dentro del rango de visión de esta y el poder utilizar cualquier patrón como marca válida para ser reconocida por ARToolKit; esto significa que señalamientos ya existentes y utilizados en la vida cotidiana, como pueden ser una señal de ALTO, de NO ENTRAR o simplemente el logotipo de alguna empresa o entidad pública, pueden ser marcas reconocibles por el sistema.

La Figura 3.4 muestra como dos señalamientos que suelen ser comúnmente utilizados en la vida cotidiana pueden ser fácilmente transformados a marcas

válidas y reconocibles por ARToolKit; de esta manera se demuestra que estos diseños pueden ser utilizados en cualquier ambiente, cualquier lugar, en instituciones públicas y privadas, parques y escuelas.

Figura 3.4: "Conversión de señalamientos cotidianamente utilizados a marcas válidas de ARToolKit"

3.1.5. ARToolkitPlus

ARtoolkitPlus es una biblioteca de Realidad Aumentada desarrollada por el Instituto para Visión y Gráficas por Computadora como parte del proyecto Handhled Augmented Reality, de la Universidad de Graz en Austria[WS07].

Sus características son:

- API basada en clases
- Hasta 4096 marcadores basados en identificador (inspirado por ARTag)
- Marcadores con borde de ancho variable
- Nuevo algoritmo de detección de pose
- Optimizaciones para su uso en dispositivos móviles
- Aritmética de punto fijo
- Soporte para formatos de píxel de cámaras de dispositivos móviles, como RGB565 y YUV
- Umbralizado automático a partir del último patrón detectado
- Compensación de viñetado
- Tablas de búsqueda para funciones trigonométricas y corrección de distorsión

Los marcadores basados en identificadores no requieren entrenamiento ni empatado de imagen, esta última es una operación que en ARToolkit tiene un costo de 4*M*N, donde 4 es el número de rotaciones de la imagen, en pasos de 90°, que se almacenan por patrón; M es la cantidad de patrones en la base de datos y N

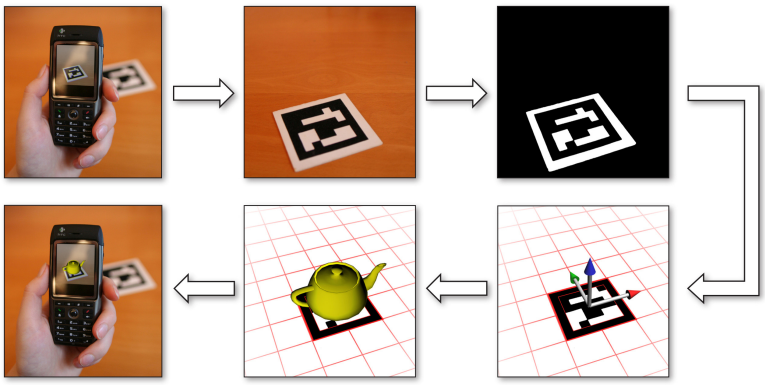


Figura 3.5: ARToolkitPlus

el número de patrones visibles; además la codificación de los identificadores es robusta a rotaciones a pasos de 90°[WS07].

La detección de marcadores empieza con una detección de bordes simple, mediante umbralizado de la imagen por un valor constante, seguido de una detección de cuadrángulos; se rechazan las áreas que sean muy grandes o muy pequeñas. Enseguida, las áreas interiores de los cuadrángulos restantes son normalizados por una transformación de perpectiva, generando un conjunto de subimágnes que son revisados contra un conjunto de patrones conocidos. Al detectar un patrón, se usan los bordes para una estimación burda de la pose de la cámara. En el siguiente paso la parte de la rotación de la pose es refinada mediante ajuste de matriz[WS07].

ARToolkitPlus no tiene funciones para capturar imágenes o dibujar objetos gráficos, la biblioteca solamente extrae la información de la pose de la cámara a partir de una imagen de escala de grises, representada por una matriz de valores enteros, la cual puede provenir de un archivo de imagen o el flujo de imágenes de una cámara. Para propósitos de esta tesis, se usaron cámaras de vídeo para pasar imágenes a ARToolkitPlus.

3.1.6. ARToolKit vs ARToolKit Plus

ARToolKit detecta los patrones asociados a una marca de la siguiente manera: se encuentran grupos de pixeles conectados que estén por debajo de cierto nivel (umbral), se encuentran los contornos de estos grupos y se identifican a los grupos con 4 contonrnos rectos como marcas potenciales. Las esquinas de los grupos se usan para remover la deformación de perspectiva y llevar el patrón a una vista frontal canónica, así como para definir una homografía para muestrear una cuadrícula de NxN valores de escala de grises[KB99].

Cada marcador que se agrega a ARToolKit se almacena en un archivo de

patrones como una matriz de 16x16 valores de escala de grises, la cual es rotada a pasos de 90° para generar otras tres matrices. Estas cuatro matrices forman los patrones asociados a un marcador. Se puede tomar la misma imagen bajo diferentes condiciones de iluminación para generar otras cuatro matrices de valores de escala de grises, las cuales se almacenan en el archivo de patrones. ARToolKit permite hasta tres niveles de iluminación, lo que da un total de 12 matrices de escala de grises[KB99].

El tiempo de búsqueda de un patrón en ARToolKit es de $4 * M * N$, donde cuatro son las rotaciones de un mismo patŕon a pasos de 90°, N es la cantidad de marcadores encontrados en la imagen y M es la cantidad de patrones conocidos por el programa; esto se debe a que cada marcador debe ser empatado contra cada patrón conocido y sus rotaciones, para determinar si éste tiene algún patrón asociado[WS07].

En el caso de ARToolKitplus, la identificación de patrones para un marcador se hace mediante un método basado en bordes donde los pixeles de los bordes son umbralizados y enlazados en segmentos, que, a su vez, son agrupados en cuadrángulos. Las esquinas de la frontera del cuadrángulo se usan para crear un mapeado de homografía para muestrear en interior del marcador, al cual también se agrupan los segmentos de los bordes de los cuadros que representan los bits de datos del marcador para obtener el número del marcador.

El tiempo de búsqueda patrones en ARToolKitPlus es de N, que es la cantidad de marcadores que aparecen en la imagen, esto se debe a que el sistema extrae los datos de los marcadores sin necesidad de hacer empatado de imágenes. La API de ARToolKitPlus permite al programador restringir la cantidad máxima de patrones que puede extraer de una imagen, por lo que un ambiente lleno de marcadores no representa un impacto significativo en el desempeño del programa[WS07].

	ARToolKit	ARToolKitPlus
Tipos de patrones	Imágenes	Identificadores
Método de detección de patrones	Umbralizado en escala de grises	Umbralizado binario
Método de identificación de patrones	Empatado de imágenes	Detección de identificador
Entrenamiento previo de patrones	Sí	No
Cantidad de variantes por patrón	4	0
Tiempo de búsqueda de patrones	4*M*N	N
Cantidad máxima de patrones en programa	El tiempo de búsqueda no permite poner un número elevado de patrones.	512

Cuadro 3.1: Comparación entre ARToolKit y ARToolKitPlus

3.1.7. OpenCV

La captura de vídeo por cámara se hizo mediante la biblioteca OpenCV de Intel, que es una API enfocada a Visión Computacional en tiempo real, la cual proporciona estructuras de datos para operaciones con matrices e imágenes, así como funciones para procesamiento de imágenes, visión computacional, aprendizaje de máquina, dibujado sobre mapas de bits, interfaces gráficas de usuario y captura de vídeo[Int99].

La biblioteca OpenCV es capaz de manejar varias cámaras a la vez, utilizando un índice entero para referirse a cada una de ellas en el momento en que son inicializadas. Esta funcionalidad es importante, ya que el sistema usa 3 cámaras para triangular la posición del usuario. Por cada imagen que se captura por una cámara, se hace una conversión a escala de grises, seguida de un ajuste de histograma para arreglar condiciones de poco contraste en la imagen; a esta imagen se le hace un ajuste manual de brillo y contraste para facilitar la detección de los marcadores, usualmente los valores altos de contraste resultan en imágenes a blanco y negro.

3.2. BRÚJULA DIGITAL

Una brújula es un instrumento de navegación para determinar la dirección relativa hacia los polos magnéticos de la Tierra. Consiste en un apuntador magnetizado (usualmente marcado hacia el Norte) libre para alinearse así mismo con el campo magnético de la Tierra. La cara principal de la brújula generalmente sobresalta los puntos cardinales norte, sur, este y oeste.

A lo largo del tiempo, se han desarrollado dispositivos derivados de brújulas que no dependen del campo magnético de la Tierra (conocido comúnmente en tales casos como “norte real” en alternativa a “norte magnético” manejado para las brújulas tradicionales). Una brújula giroscópica puede ser usada para encontrar el norte real sin ser afectada por campos magnéticos en el entorno, circuitos eléctricos cercanos o ciertas masas metálicas que aporten magnetismo al entorno. La tecnología en el ámbito de las brújulas ha resultado en el desarrollo de una brújula electrónica digital, conocida tambien como brújula giroscópica de fibra óptica. Este dispositivo se utiliza frecuentemente como un subsistema opcional en los sistemas GPS.

La brújula digital que se utilizará en la arquitectura de este sistema, es el Sensor Brújula HiTechnic NXT, cuyo nombre oficial en inglés es: “HiTechnic NXT Compass Sensor For LEGO® Mindstorms® NXT”.

Figura 3.6: La primera generación del Brick Lego Programable [Wik09]

3.2.1. LEGO® Mindstorms® NXT

Lego Mindstorms es una linea del conjunto de productos para robótica de Lego. Combina dispositivos programables con motores eléctricos, sensores, partes plásticas de construcción tradicional y de locomoción tales como engranajes, ejes y vigas. La primera versión comercial de Lego Mindstorms fue liberada en 1998 y conocida comercialmente como, en inglés, Robotic Invention System (RIS). La versión actual fue liberada en 2006 como Lego Mindstorms NXT. La figura 3.6 muestra una imagen de lo que fuera el primer Brick Programable que Lego sacara al mercado.

Lego Mindstorms NXT reemplaza la primera generación del kit Lego Mindstorms (RIS). Viene originalmente con un software de programación llamado NTX-G, sin embargo, existe una variedad de otros lenguajes tales como: NXC, NBC, RobotC, y BricxCC.

3.2.1.1. NXT Brick

El componente principal del NXT es una pequeña computadora en forma de un pequeño ladrillo (de ahí el origen de su nombre) llamado NXT Brick. Este dispositivo puede tomar entradas de hasta cuatro sensores y controlar hasta tres motores, por medio de cables RJ12, muy similares pero incompatibles con los cables de teléfono RJ11. El brick tiene un pantalla LCD monocromática de 100x64 píxeles y cuatro botones que pueden ser usados para navegar en una interface de usuario utilizando menues jerárquicos. También cuenta con bocinas que permiten reproducir archivos de sonido en frecuencias de 8kHz. La energía la obtiene de 6 baterias AA (1.5V cada una).

Especificaciones Técnicas
Microcontrolador de 32-bit ARM7
256 Kbytes FLASH, 64 Kbytes RAM
Microcontrolador AVR de 8-bit
4 Kbytes FLASH, 512 Byte RAM
Bluetooth
Puerto USB (12 Mbit/s)
4 puertos de entrada, cable 6-vias digital platform
3 puertos de salida, cable 6-vias digital platform
100 x 64 pixel LCD graphical display
Procesador Principal de 32-Bits AT91SAM7S256 (256 KB de memoria flash, 64KB RAM).

Cuadro 3.2: Espeficicaciones Técnicas [LEG08]

Figura 3.7: La nueva generación del Brick Lego Programable, el NXT [LEG08]

La figura 3.7 muestra el modelo de última generación, el Mindstorms NXT, con los diferentes sensores que pueden ser conectados para su utilización.

Programación del NXT

Programas muy básicos y sencillos pueden escribirse directamente utilizando el menú del NXT. Programas más complicados y archivos de sonido pueden ser transferidos a este utilizando el puerto USB o de manera inalámbrica mediante Bluetooth. Mediante Bluetooth también se pueden copiar archivos entre dos brick NXT, e inclusive, algunos teléfonos móviles pueden ser usados como control remoto.

NXT-G

NXT-G v1.0 es el software de programación que viene con el NXT. Este

software es adecuado para programación básica, tal como son motores, entradas de sensores (de tacto, de luz, brújulas), realizar cálculos, y utilizar estructuras de programacion simples y control de flujos.

Ventajas y Desventajas de NXT-G

Ventajas	Desventajas
NTX-G es fácil de instalar en Windows XP, Vista y Mac.	Los programas escritos con NXT pueden ser de mayor tamaño que aquellos creados con otros lenguajes de programación (ej: de 2 Kb a 12Kb).
NXT-G puede transferir datos via Bluetooth o mediante cable USB	Los programas toman relativamente mayor tiempo en cargar que los creados con otros lenguajes
NXT-G provee una interfaz gráfica "drag-and-drop" fácil de usar.	Cuando se crean programas muy largos, NXT-G tiende a colapsarse y perder los datos no guardados

Cuadro 3.3: Ventajas y Desventajas de NXT-G [LEG08]

3.2.2. NXT Compass Sensor

Como se comento anteriormente, el NXT viene con soporte para distintos tipos de sensores, de tacto, de luz, de color y, entre otros, un sensor brújula digital que mide el campo magnético de la tierra y y devuelve un ángulo de 0 a 359 grados, siendo 0° la orientación exacta al norte. La dirección a la que está apuntando actualmente es calculada al grado discreto más cercano y se actualiza 100 veces por segundo. La Brújula se conecta a través del puerto de sensores que viene equipado con el NXT y utiliza el protocolo de comunicación digital I^2C. Es importante mencionar que este sensor no es desarrollado por Lego, sino por HiTechnic; sin embargo, cumple con los estándares de los sensores Mindstorms para ser compatibles con todos los elementos de éste.

Figura 3.8: HiTechnic NXT Compass Sensor (Brújula)[HiT09]

3.3. LOCALIZACIÓN

3.3.1. Triangulación

La triangulación es un método para determinar la posición de un objeto basado en rangos de medidas simultáneas desde tres puntos en coordenadas conocidas. Este es un método muy utilizado comúnmente no sólo en localización de robots móviles, sino también en cinemática, aeronáutica, graficación por computadora, entre otros. Esto puede ser expresado trivialmente como el problema de encontrar la intersección de tres circunferencias. La figura 3.9 muestra tres circunferencias con centros P1, P2, P3 y con radios r1, r2, r3, respectivamente, los cuales se intersectan en el punto B, la línea d es la distancia en x entre P1 y P2, el punto A es la intersección entre P1 y P2, la línea i es la distancia en x de P1 y P3 y la línea j es la distancia en y entre P1 y P3 y entre P2 y P3.

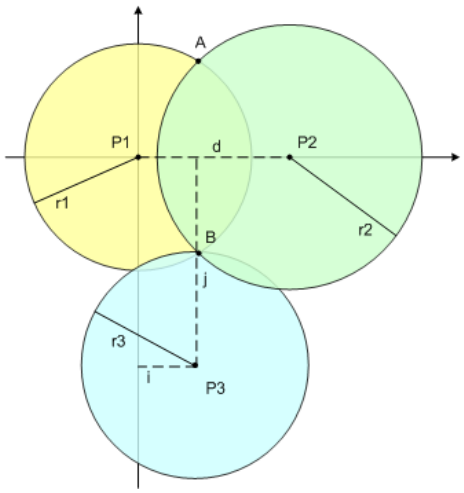


Figura 3.9: Intersección de tres circunferencias que dan lugar a la triangulación

Convirtiendo la figura anterior a un sistema de ecuaciones cuadráticas, se tiene lo siguiente:

$$\left.\begin{array}{l}(x-x_1)^2+(y-y_1)^2=r_1^2\\(x-x_2)^2+(y-y_2)^2=r_2^2\\(x-x_3)^2+(y-y_3)^2=r_3^2\end{array}\right\} \tag{3.1}$$

Donde x,y son las coordenadas del punto a encontrar (punto B), $x_i, y_i, i = 1, 2, 3$ son las coordenadas para cada uno de los centros y $r_i, i = 1, 2, 3$ son los radios de cada una de las circunferencias, en otras palabras, es la distancia existente entre el punto B con el centro de cada una de las circunferencias.

Tomando (3.1) para n circunferencias, tenemos para cada circunferencia:

$$(x-x_n)^2+(y-y_n)^2=r_n^2 \tag{3.2}$$

3.3.2. Aplicaciones de la Triangulación

La triangulación como método de localización de un punto bidimensional (o en muchos casos hasta tridimensional) no radica sólo en una idea abstracta. Además de las aplicaciones antes mencionadas, una aplicación bastante común es en sismología.

En sismología, los sismólogos encuentran el epicentro de los terremotos utilizando esta técnica. Los terremotos envían dos tipos de ondas de choque que viajan a diferentes velocidades. Determinando la demora en tiempo entre la primer onda y la segunda, puede determinarse que tan lejos ha ocurrido el terremoto.

Antes de adentrarse más profundamente en la resolución matemática, se presenta un análisis gráfico. Tan sólo sabiendo la distancia, se define un círculo alrededor de la estación donde la medición tuvo lugar. La figura 3.10 muestra la estación s1 la cual ha calculado la distancia d1 con la que se encuentra del epicentro. El terremoto pudo haber ocurrido en cualquier lugar a lo largo del perímetro de este círculo.

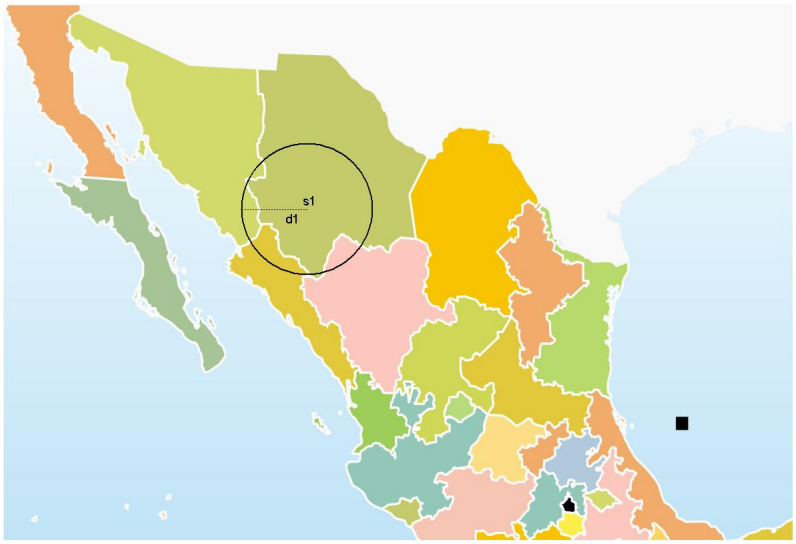


Figura 3.10: Medición de la distancia del epicentro con una sóla estación

Si dos estaciones en diferentes lugares toman la medida de este terremoto, esto reduciria las posibilidades de lugar de ocurrencia a sólo dos, que son los puntos donde ambas circunferencias se traslapan. La figura 3.11 muestra una nueva medición por parte de la estación S2, la cual se encuentra a una distancia d2 del epicentro; con las mediciones de ambas estaciones, se tienen dos círculos, puede observarse que existen dos puntos (ep1, ep2) en donde se traslapan, siendo estos los posibles puntos donde ocurrió el epicentro.

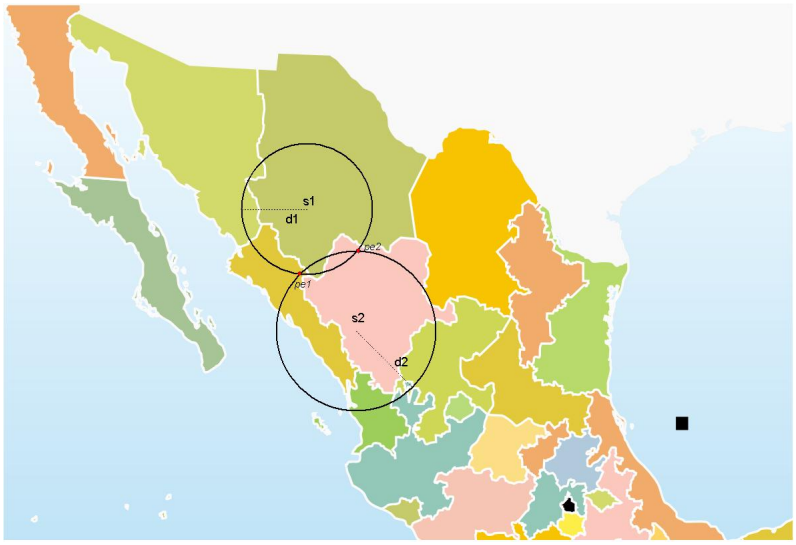


Figura 3.11: Medición de la distancia del epicentro con dos estaciones

Sin embargo, teniendo una tercer medida en otra estación precisaría el punto exacto el cual es, de los dos puntos posibles, el verdadero epicentro. La figura 3.12 muestra una tercera estación s3 que ha calculado una distancia d3 del epicentro. La intersección de las tres circunferencias, muestran el punto exacto en donde ha ocurrido el epicentro.

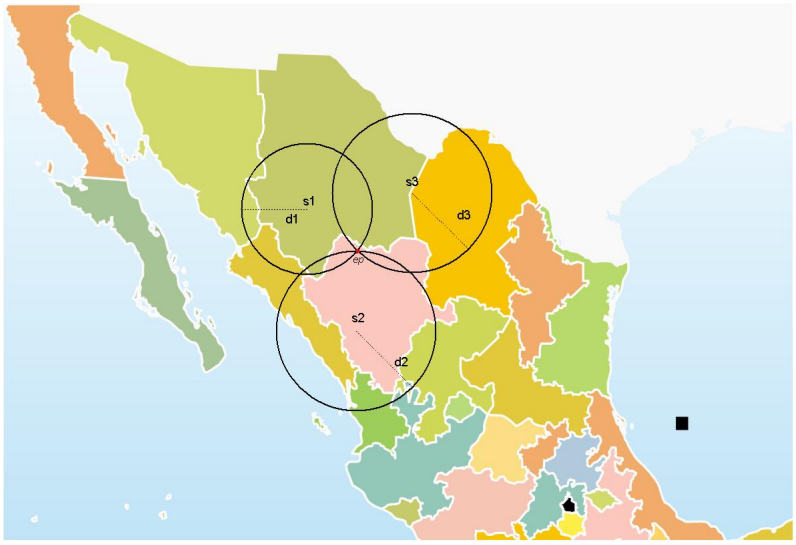


Figura 3.12: Medición de la distancia del epicentro con tres estaciones

Una vez analizado el problema gráficamente, podría parecer muy trivial la manera de encontrar el punto donde el epicentro tuvo lugar; sin embargo, detrás de todo esto se encuentra una resolución matemática. Un método de resolución es resolviendo sistemas de ecuaciones no lineares simultaneas. Sin embargo, existe un método más simple y menos costoso computacionalmente hablando.

Realizando la resolución con tres circunferencias que nos darán como resultado un punto bi-dimensional; se procede de la siguiente manera:

Retomando la ecuación 3.1. Se toman dos de las tres ecuaciones de los círculos y se substrae una de la otra, el resultado será la ecuación de la línea pasando a través de los dos puntos en donde intersectan ambos círculos. La gran ventaja de hacer esto, es que el resultado es, por tanto, una ecuación lineal, la cual es más sencilla de manejar.

Lo siguiente es, substraer cualquier otro par de las tres ecuaciones de círculos para obtener así una segunda ecuación lineal. La intersección de estas dos lineas es la localización buscada. Ambas ecuaciones lineales pasarán através de dos puntos de intersección entre dos círculos, y uno de estos puntos es el punto buscado.

3.3.3. Encontrar el epicentro de un terremoto con triangulación

Para demostrar su correcto funcionamiento se usará un ejemplo:

La estación de investigación sismológica A está localizada en las coordenadas (100,100) de un plano bidimensional, la estación B en las coordenadas (160, 120) y la estación C en las coordenadas (70, 150). Los análisis de las onda de choque S y P indican que un particular terremoto ocurrió a 50 km de la estación A, 36.06 km de la estación B, y 60.83km de la estación C. Con esta información, se debe de localizar el epicentro del terremoto.

Lo primero es encontrar las ecuaciones para los tres círculos, usando las distancias obtenidas de cada estación como radio. Por lo tanto, tomando la ecuación 3.1 tenemos:

$$\begin{array}{rl} A: & (x-100)^2 + (y-100)^2 = 50{,}00^2 \\ B & (x-160)^2 + (y-120)^2 = 36{,}06^2 \\ C: & (x-70)^2 + (y-150)^2 = 60{,}83^2 \end{array} \tag{3.3}$$

Estas ecuaciones pueden expandirse a:

$$\begin{array}{rl} A: & x^2 - 200x + 10000 + y^2 - 200y + 10000 = 2500 \\ B & x^2 - 320x + 25600 + y^2 - 240y + 14400 = 1300 \\ C: & x^2 - 140x + 4900 + y^2 - 300y + 22500 = 3700 \end{array} \tag{3.4}$$

Lo siguiente es tomar dos pares de ecuaciones y substraerlas. Hay tres combinaciones únicas para hacer esto, sin embargo, sólo se necesitan dos. Téngase en cuenta

que al hacer la substracción de cualquier ecuación con otra, los términos x^2 y y^2 se cancelan, quedando sólo ecuaciones lineales.

$$\begin{array}{ll} A-B: & 120x - 15600 + 40y - 4400 = 1200 \\ B-C & -180x + 20700 + 60y - 8100 = -2400 \end{array} \tag{3.5}$$

Despejando y en ambas ecuaciones se tiene:

$$A - B: \quad y = -3x + 530 \tag{3.6}$$

$$B - C: \quad y = 3x - 250 \tag{3.7}$$

Hasta este momento, ya se tienen los dos puntos posibles donde ha ocurrido el terremoto. Para encontrar el punto exacto, se debe encontrar la intersección de las dos ecuaciones lineales. Entonces se tiene:

$$\begin{array}{l} A - B = B - C \\ -3x + 530 = 3x - 250 \\ 780 = 6x \\ x = 130 \end{array} \tag{3.8}$$

Substituyendo x en la ecuación 3.7tenemos:

$$\begin{array}{l} y = 3(130) - 250 \\ y = 140 \end{array} \tag{3.9}$$

Por lo tanto, el epicentro del terremoto es (130, 140)

La localización por triangulación también se extiende a casos tridimensionales,

de hecho, es así como funcionan el Global Position System (GPS). Cada satelite GPS tiene un reloj atómico de alta precisión. Midiendo el retraso de la velocidad de la luz de las trasmisiones entre el GPS y el receptor (aproximadamente 50 milisegundos)

3.3.4. Triangulación para localización de una persona invidente en un entorno semi controlado

Una vez analizado lo anterior, puede llevarse esta técnica a un contexto de intereses para la presente tesis. Supóngase que una persona invidente se encuentra en un apartamento (fig: 3.13) en una posición (x,y) desconocida. Con el sistema aquí propuesto, se detectan tres marcas distintas en diferentes posiciones (unidades dadas en metros). Las marcas detectadas son las siguientes:

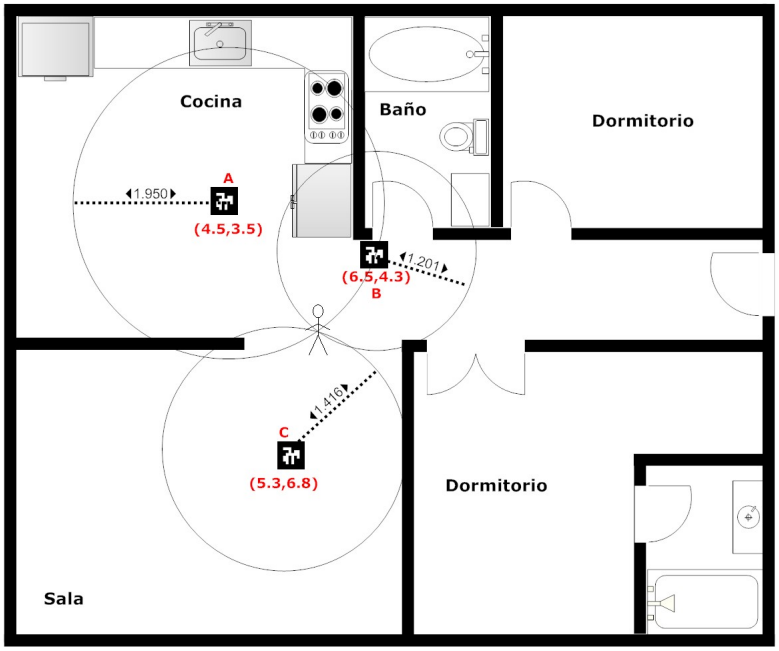


Figura 3.13: Triangulación de un invidente en un entorno semi-controlado

Marca A, con posición (4.5, 3.5) a una distancia de 1.950 mts del invidente.

Marca B, con posición (6.5, 4.3) a una distancia de 1.201 mts del invidente.

Marca C, con posición (5.3, 6.8) a una distancia de 1.416 mts del invidente.

Con esta información, aplicando triangulación, es posible encontrar la posición del invidente.

Aplicando la ecuación 3.1 se tiene:

$$\begin{array}{ll} A: & (x-4{,}5)^2+(y-3{,}5)^2=1{,}950^2 \\ B & (x-6{,}5)^2+(y-4{,}5)^2=1{,}201^2 \\ C: & (x-5{,}3)^2+(y-6{,}8)^2=1{,}416^2 \end{array} \tag{3.10}$$

Expandiendo 3.10:

$$\begin{array}{ll} A: & x^2-9x+20{,}25+y^2-7y+12{,}25=3{,}8025 \\ B: & x^2-13x+42{,}25+y^2-8{,}6y+18{,}49=1{,}442 \\ C: & x^2-10{,}6x+28{,}09+y^2-136y+46{,}24=2{,}005 \end{array} \tag{3.11}$$

Tomando dos pares de ecuaciones y substrayéndolas:

$$\begin{array}{ll} A-B: & 4x-22+1{,}6y-6{,}24=2{,}3605 \\ B-C: & -2{,}4x+14{,}16+5y-27{,}75=-0{,}563 \end{array} \tag{3.12}$$

Despejando y en ambas ecuaciones se tiene:

$$A-B: \quad y=\frac{30{,}60-4x}{1{,}6} \tag{3.13}$$

$$B-C: \quad y=\frac{13{,}027+2{,}4x}{5} \tag{3.14}$$

Lo siguiente es encontrar la intersección de las dos ecuaciones lineales 3.13 y 3.14 para encontrar el punto exacto en donde se encuentra el invidente. Por lo tanto, tenemos:

$$\begin{array}{l} A-B=B-C \\ \frac{30{,}60-4x}{1{,}6}=\frac{13{,}027+2{,}4x}{5} \\ x=5{,}54 \end{array} \tag{3.15}$$

Substituyendo x en la ecuación 3.14 tenemos:

$$\begin{array}{l} y=\frac{13{,}027+2{,}4(5{,}54)}{5} \\ y=5{,}26 \end{array} \tag{3.16}$$

Por lo tanto, se tiene que la posición del invidente es (5.54, 5.26).

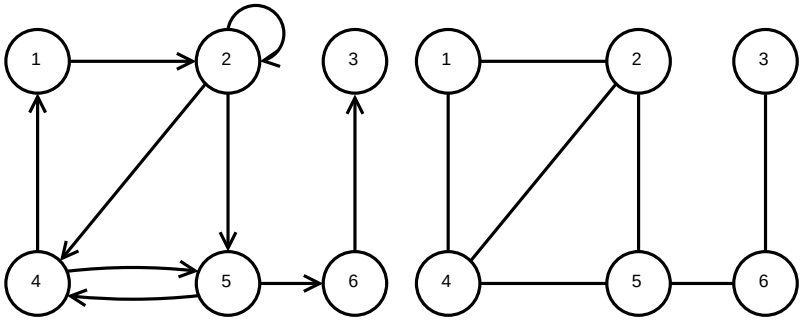
3.4. MAPA TOPOLÓGICO

Para guiar al invidente, el sistema tiene un mapa que contiene todas las rutas sin obstáculos por donde puede caminar el invidente, a partir de la cual obtiene rutas para llegar de un lugar a otro. Para representar este mapa dentro del sistema, su contenido se representa con un grafo.

Un grafo es una estructura de datos formada por una cantidad de objetos de datos, cada uno es llamado vértice. Cualquier vértice puede estar conectado con cualquier otro y a las conexiones de les llama aristas. Los grafos se describen por la forma en que las aristas se conectan a los vértices[CLRS01].

Un **grafo dirigido** (o digrafo) G es un par (V, E), donde V es un conjunto finito de datos y E es una relación binaria entre elementos de V. El conjunto E es llamado el **conjunto de aristas** de G, y a sus elementos se les llama **aristas**. En un grafo dirigido es posible tener aristas que conecten a un vértice con si mismo (Fig. 3.14a). Por convención, las aristas de un grafo dirigido se representan con la notación (u, v), y (u, v) y (v, u) representan aristas diferentes[CLRS01].

En un grafo **no dirigido** $G = (V, E)$, el conjunto de aristas E consiste en pares no ordenados de vértices, más que en pares ordenados. Una arista es un conjunto $\{u, v\}$, donde $u, v \in V$ y $u \neq v$, lo cual impide la existencia de ciclos que conecten a un vertice consigo mismo, por lo que cada arista consiste de exactamente dos vértices distintos (Fig. 3.14b). Por convención, las aristas de un grafo no dirigido se representan con la notación (u, v), y (u, v) y (v, u) representan la misma arista[CLRS01].



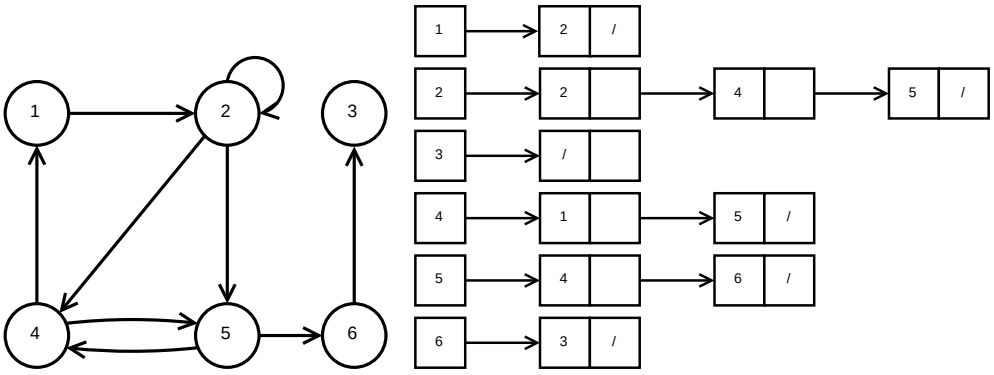
(a) Grafo Dirigido (b) Grafo No Dirigido

Figura 3.14: Tipos de grafos.

Si (u, v) es una arista en un grafo $G = (V, E)$, se dice que el vertice v es **adyacente** al vértice u. Cuando se trata de un grafo no dirigido, la relación de adyacencia es simétrica. En el caso de un grafo dirigido, si v es adyacente a u, la relación se escribe como $u \rightarrow v$.

Un grafo no dirigido es **conexo** si cada par de vértices está conectado por una ruta, mientras que en caso de un grafo dirigido, si todos los pares de vértices son alcanzables entre si, se le denomina **fuertemente conexo**.

Hay dos formas estándares para representar un grafo: como una colección de listas de adyacencia o como una matriz de adyacencia. Cualquiera de estas formas son aplicables a grafos dirigidos y no dirigidos. La representación como colección de listas de adyacencia se prefiere porque proporciona una forma compacta de representar grafos poco densos, donde la cantidad de aristas es mucho menor que el cuadrado del número de vertices. Cuando el grafo es denso – que cantidad de aristas se acerce al cuadrado del número de vértices – o se quiera averiguar de forma rápida si hay una arista entre dos vértices, es preferible la representación por matriz de adyacencia. Para el software desarrollado en esta tesis, se utilizó una representación por matriz de adyacencia[CLRS01].



(a) Grafo dirigido con 6 vertices.

(b) Representación como lista de adyacencia.

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	1
6	0	0	1	0	0	0

(c) Representación como matriz de adyacencia.

Figura 3.15: Representación de un grafo dirigido.

La representación de matriz de adyacencia de un grafo, asume que los vértices están numerados $1, 2, \ldots, |V|$ de alguna forma arbitraria. Entonces, la matriz de adyacencia de un grafo consiste en una matriz $A = (a_{ij})$ de tamaño$|V| \times |V|$ tal que

$$a_{ij} = \begin{cases} 1 & si\ (i,j) \in E, \\ 0 & en\,caso\,contrario. \end{cases}$$

La definición anterior es para aristas que sólo indican la relación entre dos vértices. Para el caso de un grafo con aristas ponderadas (con peso), la matriz $A = (a_{ij})$ es tal que

$$a_{ij} = \begin{cases} \neq 0 & si\ (i,j) \in E, \\ 0 & en\,caso\,contrario. \end{cases}$$

El mapa topológico del sistema es representado por un grafo con aristas ponderadas, cuyos vértices representan los diferentes puntos a donde se puede llegar dentro

de la escena, los cuales tienen información de su ubicación en el mundo real. Esta información espacial es utilizada para asignar peso a las aristas, que es igual a la distancia euclidiana entre dos vértices conectados[CLRS01].

3.5. PLANEACIÓN DE RUTAS

3.5.1. Algoritmo de Dijkstra

El invidente es guiado en la casa o en el ambiente donde se encuentre mediante un mapa que contiene un grafo de las conexiones entre cada habitación y demás puntos existentes, cada punto es un vértice del grafo, sin embargo, existen diversos vértices entre un punto y otro; la conexiones entre vértices son las aristas del grafo.

Para encontrar la ruta más corta entre cuartos, se usa el algoritmo de Dijkstra, el cual es un algoritmo que encuentra la ruta más corta de un vértice s a un vértice t de un grafo finito conexo G. Describiéndo el algoritmo de manera coloquial, éste acomoda los vértices de G en orden creciente de distancia ponderada desde s *[Wal00]*.

El algoritmo se inicializa poniendo los pesos de todas las aristas $w(x, y)$ a ∞ y el peso del vértice s se pone a cero. Después, se ordenan los vértices de G como $Ss_{k=}\{s_0 (= s), s_1, s_2 \ldots, s_n\}$ de tal forma que las distancias ponderadsa $w(s, s_1), w(s, s_2) \ldots$ estén en orden no decreciente, poniendo una etiqueta temporal $\ell(x)$ a cada miembro x de $V(G)$, el conjunto de vértices del grafo, $\ell(x)$ es el límite superior de la longitud ponderada más corta de s a x. En el siguiente paso, por cada x que no esté en s_k, el algoritmo cambia el valor de $\ell(x)$de la longitud ponderada de la ruta más corta (s, x) que tiene una sola arista en S_k, se elige S_{k+1}para un vértice x tal que $\ell(x)$ es minimizada [Wal00].

El proceso con todos los vértices va de la siguiente forma [Wal00]:

- $s_0$es el vértice origen de la ruta s a t.
- $s_1$es el vértice cuyo peso $w(s, x)$es mínimo.
- El vértice $s_2$se encuentra seleccionando por cada vértice, excepto s o s_1, ya sea la arista con peso más bajo sx o la ruta ss_1x, y así sucesivamente. Cuando se etiqueta un vértice, su valor-ℓ no cambia y su valor final es igual a $w(s, x)$.
- Cuando se etiqueta un vértice s_{k+1}, se puede definir un vértice s_i que lo precedió. Así puede encontrarse la ruta más corta a partir de t, recorriendo todos los predecesores hasta llegar a s_0.

El algoritmo tiene un enfoque voraz porque siempre toma la arista con el peso más pequeño posible para generar el árbol de recubrimiento mínimo [CLRS01].

Algorithm 3.2 Algoritmo de Dijkstra

DIJKSTRA (Grafo G, nodo_fuente s)

```
// inicializamos todos los nodos del grafo. La distancia de cada nodo es infinita
// y los padres son NULL
for u ∈ V[G] do
   distancia[u] = INFINITO
   padre[u] = NULL        distancia[s] = 0
//encolamos el nodo_fuente s
   Encolar (cola, grafo)
   mientras cola no es vacía do
// OJO: Se extrae el nodo que tiene distancia mínima y se conserva la condición
// de Cola de prioridad
   u = extraer_minimo(cola)
   for v ∈ adyacencia[u] do
      if distancia[v] > distancia[u] + peso (u, v) do
         distancia[v] = distancia[u] + peso (u, v)
         padre[v] = u
```

3.5.2. Arbol k-d

Un problema asociado con la navegación en este sistema es saber dónde empezar para obtener la ruta más corta hacia el destino, ya que es posible que el invidente esté localizado en un punto del mapa donde no haya un vértice del grafo y se requiere mover al invidente al punto más cercano para calcular la ruta más corta hacia el destino.

La forma más simple consiste en calcular la distancia euclidiana entre el invidente y cada vértice del grafo y mover al invidente al punto donde la distancia sea la más cercana. Este enfoque tiene complejidad lineal, por lo que el tiempo de búsqueda se vuelve directamente proporcional a la cantidad de vértices del grafo.

Una mejor solución es usar una estructura de búsqueda llamada arbol k-d, que es un árbol binario cuyos nodos pueden tener cualquier número de claves, las cuales dividen el espacio mediante un hiperplano asociado a cada nivel de profundidad del árbol. El hiperplano divide el espacio y sirve como referencia para insertar futuros nodos, comparando si éstos deben quedar a la derecha o a la izquierda de un hiperplano. Cada nodo que se inserta en el árbol divide el plano donde se encuentra con un hiperplano alineado con el eje coordenado correspondiente a ese nivel de profundidad[Ben75].

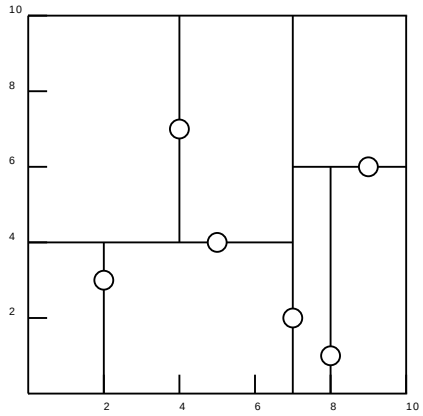


Figura 3.16: Árbol k-d

El árbol k-d se usa para búsquedas del vecino más cercano, mediante un recorrido en el árbol y calculando la distancia entre cada nodo y un punto en el espacio, viendo si ésta queda dentro de un radio predeterminado. La búsqueda termina ya sea cuando no se encontró nodo alguno o cuando se encontró un nodo cuya distancia euclidiana era la más corta con respecto al punto en el espacio[Ben75].

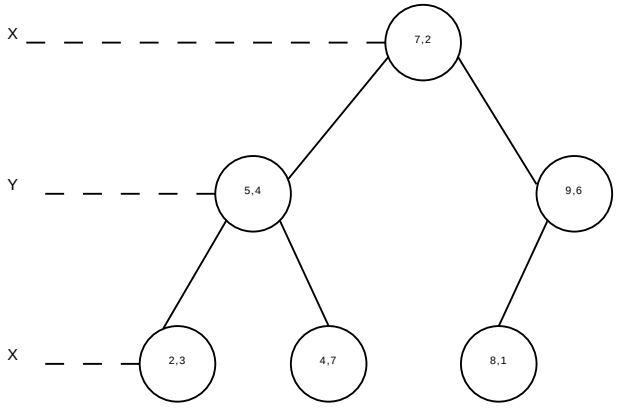


Figura 3.17: Árbol k-d

3.5.3. Guiando al Invidente

Una vez que se encuentra el vértice del mapa más cercano a la ubicación del invidente, éste se usa para obtener la ruta más corta mediante Dijkstra y para empezar a mover al invidente, por lo que es necesario acercarlo a la ubicación del vértice.

Para saber la distancia y la orientación del invidente con respecto al vértice, ilustrados en la figura 3.18, se obtiene la distancia euclidiana entre ambas ubicaciones, así como el angulo. La distancia euclidiana es la raíz cuadrada de la suma de los cuadrados de las distancias en cada uno de los ejes coordenados de cada punto (Ec. 3.17), mientras que el ángulo es el arcotangente del cociente de la distancia de ambos puntos sobre el eje Y entre la distancia de ambos puntos sobre el eje X (Ec. 3.18).

Para propósitos de esta tesis, se utilizó una versión modificada de la función de arcotangente, la cual es conocida en la biblioteca de matemáticas de lenguaje C como atan2, que usa valores con signo para X e Y para determinar el ángulo, el cual está restringido entre $+180°$ y $-180°$, como indica la figura 3.19. De esta forma es más fácil indicar los giros porque son relativos a cero grados.

$$\Delta x = x_{punto} - x_{ciego}$$
$$\Delta y = y_{punto} - y_{ciego}$$

$$d = \sqrt{\Delta x^2 + \Delta y^2} \tag{3.17}$$

$$\alpha = atan\left(\frac{\Delta y}{\Delta x}\right) \tag{3.18}$$

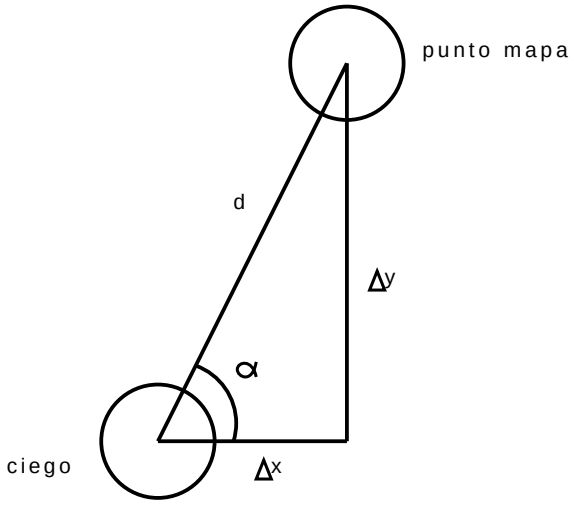


Figura 3.18: Orientación y dirección entre 2 puntos

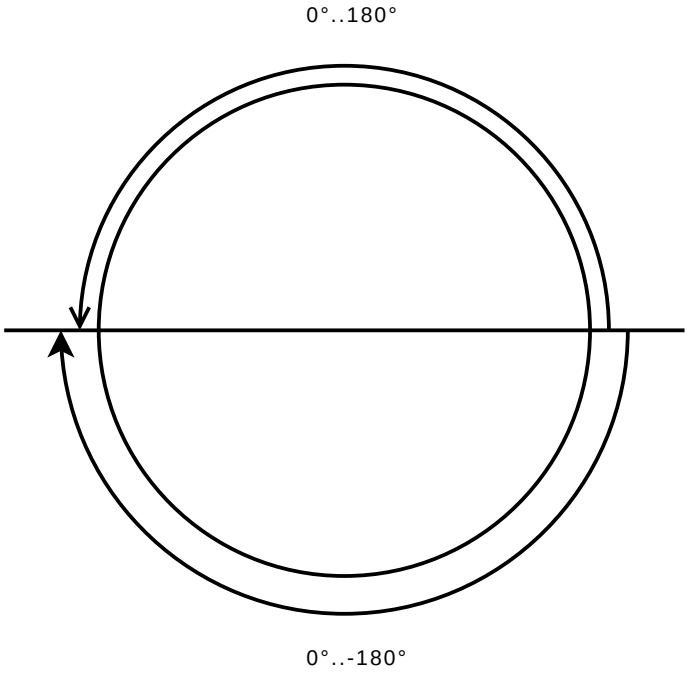


Figura 3.19: Ángulos obtenidos por atan2

El ángulo obtenido se utiliza para alinear la brújula con el vértice y el invidente empiece a caminar. Con cada paso, se va calculando la diferencia entre el ángulo de orientación del invidente y el ángulo entre el invidente y el vértice (Ec. 3.19) y se le ordena que gire en sentido contrario a su ángulo de desviación para corregir su curso (Ec. 3.20), y de esa manera pueda seguir el curso hacia el vértice, como se indica en la figura 3.20. Al llegar a un vértice del grafo, el ángulo de orientación cambia al ángulo entre los siguientes dos vértices del grafo de la ruta y se repite el procedimiento anterior de desplazamiento y retroalimentación de dirección y sentido hasta que el invidente llega a su destino.

$$\Delta\alpha = \alpha_{punto} - \alpha_{ciego} \tag{3.19}$$

$$giro = \begin{cases} \Delta\alpha < -umbral & giro\, izquierda \\ \Delta\alpha > umbral & giro\, derecha \end{cases} \tag{3.20}$$

Donde $umbral = 2°$.

Se da este margen de umbral debido a que la mayor parte del tiempo el invidente no quedará alineado completamente con el objetivo.

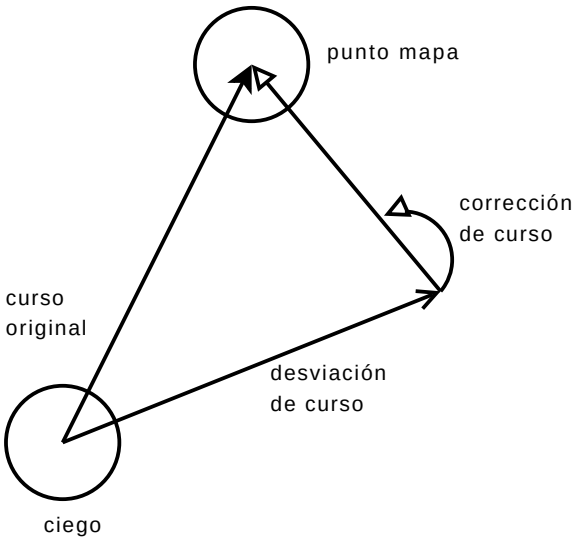


Figura 3.20: Cambio de curso

3.6. TECLADO BRAILLE

Esta es una de las partes más escenciales del sistema, pues es lo que permitirá al invidente poder comunicarse con el sistema. Para este fin, se utilizará un teclado braille el cual fue desarrollado en el 2004 con unos colegas del *Instituto Tecnológico de Tijuana* [SSL04].

El Teclado Braille consta de 16 teclas con relieve, seis de las cuales representan la celda braille, constituida por una matriz de tres renglones dos columnas, numeradas ascendentemente en forma vertical; las 15 teclas restantes estarán destinadas a funciones específicas del sistema.

El usuario, a partir de las 6 teclas que tiene, las cuales simulan la celda braille, realizará las pulsaciones de cada una de ellas, acorde al carácter que desea capturar, siguiendo obviamente la secuencia de numeración de una celda braille; una vez terminada la combinación de teclas, presionará una tecla de función1 que le indique al sistema que el usuario ha terminado de realizar las combinaciones; en este momento, el sistema procesa la combinación de teclas y realiza la codificación al carácter correspondiente de acuerdo al código ASCII; para capturar otro carácter, basta con repetir las mismas operaciones hasta obtener la o las palabras deseadas.

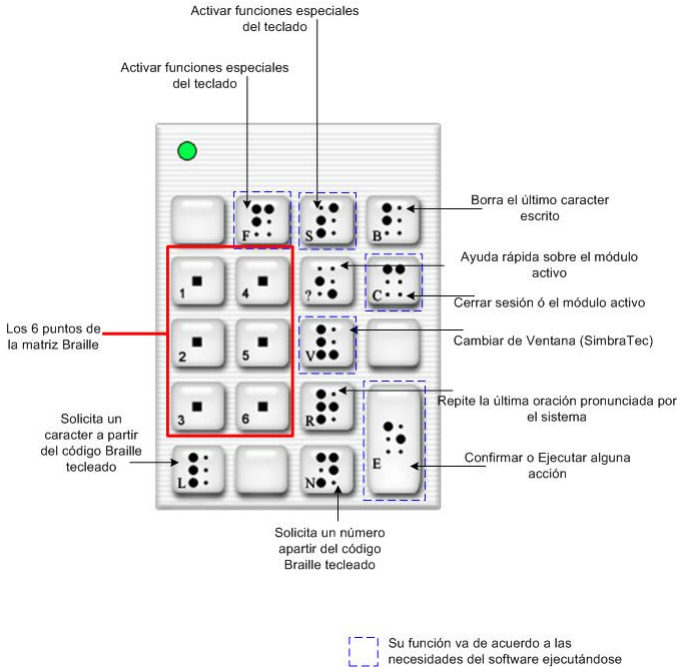


Figura 3.21: Diseño y distribución del teclado braille [SSL04]

La figura 3.21 muestra el diseño y distribución del teclado braille que se utilizará en el presente proyecto; en dicho diseño pueden apreciarse el funcionamiento de cada tecla así como la manera de escribir el alfabeto. El teclado también cuenta con teclado de funciones semejantes a un teclado normal de computadora, que van de F0 a F9. Para más información sobre el teclado braille, consultar el apéndice A.

El teclado permitirá al invidente ejecutas acciones que requiera por parte del teclado, las acciones básicas que estarán disponibles mediante el teclado son las siguientes:

- Obtener listado de los destinos posibles a donde puede desplazarse.
- Elegir uno de los destinos para obtener asistencia guiada por parte del cartógrafo.
- Cancelar cualquier operación en el momento deseado y reelegir destino.
- Preguntar al cartógrafo por los lugares más cercanos a su posición actual.

Capítulo 4

IMPLEMENTACIÓN

En este capítulo se define como se llevó a cabo la implementación del sistema. El sistema es totalmente orientado a objetos, lo que facilita su diseño y su implementación. Este sistema ha sido desarrollado de manera modular.

A continuación se define y explica como se implementó cada uno de los módulos que conforman dicho sistema para finalmente, en la última sección de este capítulo definir la integración de los módulos y la interacción que tienen cada uno de ellos para hacer posible los objetivos definidos en esta tesis.

4.1. LOCALIZACIÓN

4.1.1. Mapa del Ambiente

El mapa del ambiente juega uno de los roles más importantes en el Cartógrafo. Gracias al mapa el sistema puede saber: la posición en la que se encuentra la persona invidente, las rutas para ir de un punto origen a un punto destino, la posición de las marcas y la orientación del invidente relativa al ambiente. Sin esta parte del sistema, sería imposible determinar la información anterior. La información del mapa, así como del posicionamiento de las marcas, definición de puntos para obtención de rutas más cortas y demás información necesaria para la localización se almacenan en archivos XML cuyas estructuras se explica en apartados posteriores. Es importante mencionar que este sistema se limita a trabajar sobre mapas de dos dimensiones (x, y), descartando el vector Z utilizado en nuestro ambiente tridimensional.

4.1.1.1. Escenario

Lo principal en el desarrollo del cartógrafo, antes de definir y codificar estructuras de programación, es definir la representación del mapa que define el escenario

de navegación del invidente; esto es, establecer en el sistema la representación de los diferentes elementos que conforman un entorno conocido (habitaciones, pasillos, etc). Cada elemento que conforma el mapa, debe estar relacionado a una posición bidimensional única para cada una; sean paredes, obstáculos, u otros elementos.

4.1.1.2. Representación en XML del escenario

La información del escenario se almacena en un archivo XML cuya definición de esquema es el siguiente:

```
<?XML version="1.0" encoding="UTF-8"?>
<xs:schema XMLns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
        <xs:element name="stage" type="stageType">
                <xs:annotation>
                        <xs:documentation>Define el elemento que contiene
                 la información del escenario
            </xs:documentation>
                </xs:annotation>
        </xs:element>
<xs:complexType name="stageType">
<xs:sequence>
                        <xs:element name="polygon" type="polygonType"
                  minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="units" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:enumeration value="milimeters"/>
                                        <xs:enumeration value="decimeters"/>
                                        <xs:enumeration value="meters"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="dimensionX" type="xs:decimal" use="required"/>
                <xs:attribute name="dimensionY" type="xs:decimal" use="required"/>
        </xs:complexType>
        <xs:complexType name="polygonType">
                <xs:annotation>
                        <xs:documentation>Un polígono estará conformado
                  por al menos dos líneas necesarias para
                                dibujar este elemento
                  </xs:documentation>
                </xs:annotation>
                <xs:sequence minOccurs="2" maxOccurs="unbounded">
                        <xs:element name="line" type="lineType"/>
                </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:complexType name="lineType">
                <xs:attribute name="x" type="xs:decimal" use="required"/>
                <xs:attribute name="y" type="xs:decimal" use="required"/>
        </xs:complexType>
</xs:schema>
```

El esquema anterior produciría un archivo XML como el siguiente:

```
<?XML version="1.0" encoding="UTF-8"?>
<stage units="decimeters" dimensionX="85.00" dimensionY="130.00"
       XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="stage.xsd">
```

```
        <polygon name="Pared 1">
                <line x="2.5" y="0.0"/>
                <line x="2.5" y="50.00"/>
                <line x="23.51" y="50.0" />
                <line x="23.51" y="0.00" />
                <line x="2.5" y="0.00" />
        </polygon>
        <polygon name="Pared 2">
                <line x="60.0" y="50.0"/>
                <line x="52.13" y="50.00" />
            <line x="52.13" y="0.00" />
                <line x="23.51" y="0.00"/>
                <line x="23.51" y="50.0" />
                <line x="39.38" y="50.0" />
        </polygon>
        <polygon name="Pared 3">
       <line x="1.70" y="130.0" />
            <line x="1.70" y="72.40" />
            <line x="39.38" y="72.40"/>
        </polygon>
</stage>
```

El elemento raíz **stage** define el inicio del documento, requiere los atributos: **units** que indica las unidades métricas en las que está dada la información del escenario, este atributo puede tener tres valores posibles (milimeters, decimeters, meters); los atributos de tipo decimal **dimensionX** y **dimensionY** son utilizados para indicar la delimitación del escenario con respecto al origen (0,0) esto significa que ningún elemento dentro del mapa podrá estar fuera de estos límites.

El elemento hijo **polygon,** que tiene una ocurrencia de 0 a n veces, define el polígono que representa un objeto en el escenario; tiene un único atributo **name** que indica el nombre del objeto que se está representando. Los elementos hijos a polygon, **line,** con una ocurrencia de 2 a m, son una serie de puntos bidimensionales que indican los lados del polígono; al menos dos puntos deben existir para limitar a que un polígono sea al menos de un sólo lado; los atributos decimales **X** y **Y** indican la posición bidimensional del punto.

4.1.1.3. Representación en Memoria del Escenario

Una vez que se ha definido el esquema del archivo XML que contendrá la configuración del mapa; este debe ser cargado en memoria para que la información pueda ser utilizada por el sistema, de manera tal que se realicen los cálculos necesarios.

La figura 4.1 muestra el diagrama de clases que encapsulan la información del escenario para que pueda ser utilizada por el sistema, ya sea para realizar cálculos de localización, navegación o simplemente para su graficación. La clase **TPunto** representa el elemento mínimo de información almacenada, representa, como su nombre lo indica, un punto bidimensional en el escenario con coordenadas x,y. La clase **TPolygon** representa un polígono en el escenario, para ello contiene como parte de sus atributos, una lista de puntos (**TPunto**) los cuales indican su forma gráfica, también cuenta con un atributo name que es utilizado para su identificación. La clase **TUnits** encapsula las distinas unidades métricas que son

soportadas en el escenario, si el XML indica una unidad métrica distinta a las que se soportan en esta clase, se asigna la unidad ***UNKNOWN*** y los valores se toman como metros. La clase **TStage** es la clase principal de esta parte del sistema y es quien encapsula la información completa de todo el escenario; esta clase contiene los polígonos que representan elementos en el escenario, contiene la unidades métricas en que se representa los datos, así como las dimensiones del escenario que sirven de limitantes para éste. Por último, la clase **TStageXML**, que hereda de **TStage**, y es la encargada de realizar el parseo del archivo XML y crear así una instancia del escenario con todos sus atributos.

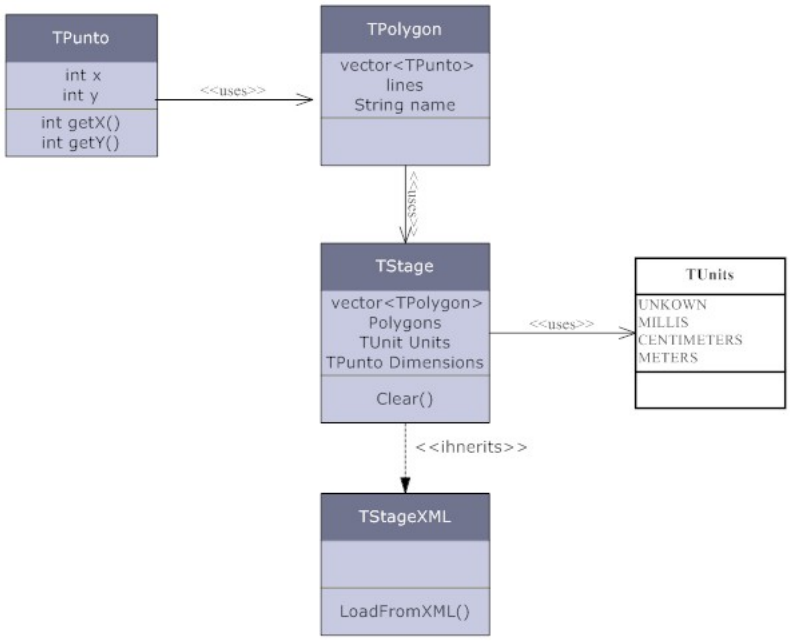


Figura 4.1: Diagrama de clases del escenario

4.1.1.4. Representación Gráfica del Escenario

El presente trabajo de tesis no intenta tener un enfocarse en desarrollo y técnicas complejas de graficación; sin embargo, resulta muy útil y eficiente para el desarrollo del sistema el poder interpretar los datos de manera gráfica. Es por eso que se ha agregado al sistema una interfaz gráfica sencilla que permita ver la interacción del sistema con los datos de entrada y los procesamientos y resultados que arroja. La representación gráfica del escenario se desarrolló utilizando Canvas, la cual es proporcionada por la libreria VCL de C++ Builder 6. Canvas permite hacer dibujados bidimensionales sencillos; lo cual resulta suficiente para el objetivo de esta tesis.

Utilizando la libreria de Canvas antes mencionada y el diagrama de clases que se describe en el punto anterior, es posible dibujar los polígonos que representan el escenario para poder así tener una percepción visual del mismo. La figura 4.2 muestra un ejemplo de escenario, que es dibujado con canvas apartir de un archivo XML como el anteriormente mencionado, que contiene tres habitaciones completas, dos habitaciones semicompletas y un pasillo; para casos de este ejemplo, este sería el escenario conocido por el sistema donde el invidente puede navegar. En secciones posteriores se explicarán otros elementos que tienen un rol importante para la localización del invidente.

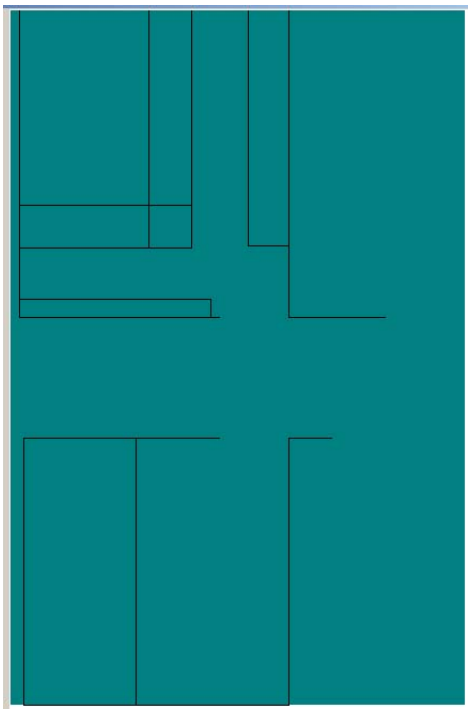


Figura 4.2: Gráfico del escenario dibujado con Canvas

4.1.2. Brújula

La brújula NXT está conectada al "brick" NXT Mindstorm de LEGO, que a su vez viene integrada con una interfaz de comunicación inalámbrica Bluetooth. Utilizando este estándar, es que es posible realizar una comunicación entre un software de computadora y la brújula. A continuación se explican los procesos que fueron necesarios seguir para su implementación; para obtener información más técnida y detallada sobre la programación de las interfaces, consulte el apéndice B.

4.1.2.1. Comunicación por Puerto Serial.

Cuando se realiza el emparejamiento del NXT con la computadora, el sistema operativo le asigna un número de puerto COM con el que se hará la comunicación. Una vez realizado lo anterior, se diseña una clase que sea capaz de realizar comunicaciones al puerto serial mediante el puerto asignado, esta comunicación debe ser bidireccional; por lo que el objeto debe ser capaz de escribir datos en el puerto (comando de petición de lectura) y leer del mismo (lectura de la brújula). Lo anterior entra en el concepto de cliente/servidor, donde el NXT realiza las funciones del servidor de orientación y la clase diseñada es el cliente quien realiza peticiones al servidor para obtener la lectura actual de la brújula. La figura 4.3 representa la comunicación que se realiza entre el cliente con el brick NXT; la computadora cliente se conecta inalámbricamente mediante un puerto COM utilizando Bluetooth como medio de transmisión de datos, esta envía un comando de petición de lectura, el NXT regresa una respuesta con la lectura actual de la brújula la cual es un número entero con valores posibles de 0 a 359.

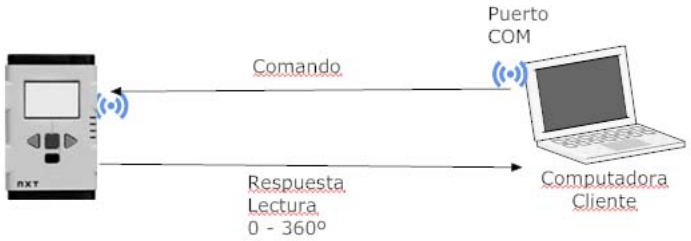


Figura 4.3: Comunicación Bluetooth entre Cliente y Servidor NXT

4.1.2.2. Interfaz de la Brújula

Como se mencionó anteriormente, el sistema de NXT proporciona una interfaz de comunicación inalámbrica por medio de Bluetooth. Dicha interfaz encapsula una conexión de puerto serial que es interpretada por el sistema operativo de la computadora cliente como tal; de esta manera, el sistema cliente por medio de comandos AT, es capaz de realizar consultas a la brújula y obtener una respuesta.

4.1.2.3. Información de lectura devuelta por la brújula.

Una vez que el servidor (NXT) recibe la petición de lectura por parte de la clase cliente, este devuelve un dato de tipo entero que tiene un valor de 0 a 359

grados (ya que $360^o = 0^o$). Donde 0^o representa una lectura exactamente en el Norte, 90^o una lectura exactamente en el Este, 180^o una lectura exactamente en el Sur y 270^o una lectura exactamente en el Oeste. De esta manera es que es interpretada la información devuelta por la brújula. La figura 4.4 muestra cinco lecturas dadas por la brújula y su representación en el plano cartesiano.

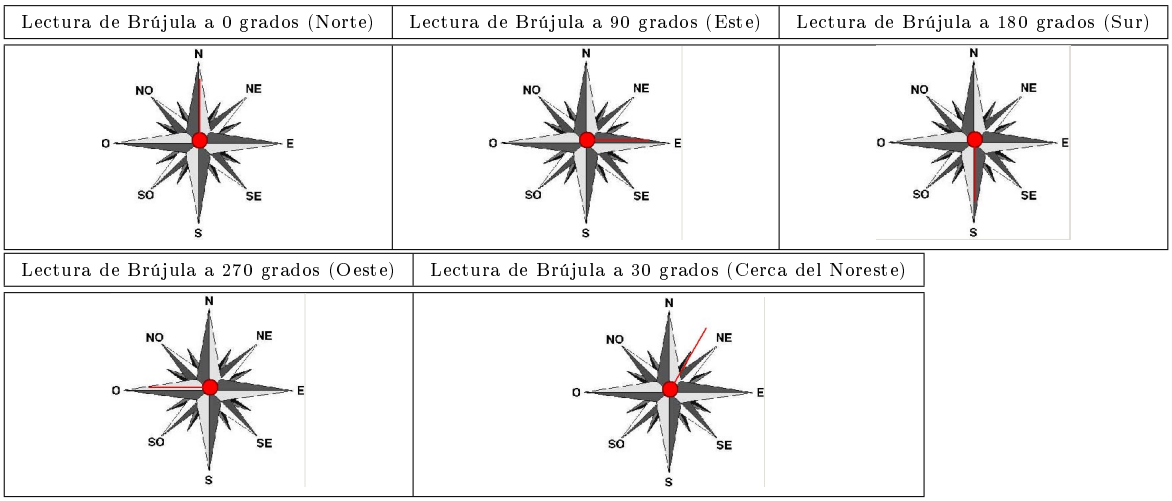


Figura 4.4: Representación en plano cartesiano de lecturas obtenidas por la brújula

4.1.2.4. Diagrama de clases del cliente de brújula

Para hacer posible realizar la comunicación con la brújula del NXT, se ha desarrollado una entidad cliente cuyas operaciones se encapsulan en tres clases (Serial, Sensor, Compass). La figura 4.1muestra el diagrama de clases con sus respectivos métodos y atributosp que conforman el cliente que se comunica con la brújula. La clase "Serial" es la de más bajo nivel y encapsula los procedimientos necesarios para realizar la conexión al puerto serial utilizando un atributo "port". La clase "Sensor" encapsula los métodos de escritura y lectura en el puerto serial. Por último la clase "Compass" encapsula los métodos de inicialización de la comunicación requerida entre el cliente y el dispositivo y la lectura respectiva a la brújula.

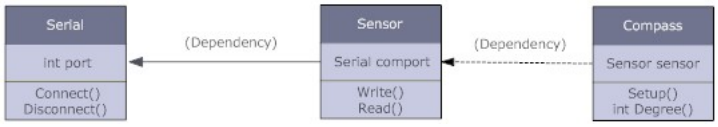


Figura 4.5: Diagrama de clases del cliente de la brújula NXT

4.1.2.5. Aplicación cliente

Basándose en el diagrama de clases anterior, puede desarrollarse una interfaz de usuario que sea capaz de interacutar con la brújula. Se desarrolló una interfaz la cual permite al usuario capturar el puerto COM que ha sido asignado por el sistema operativo para conectarse a la brújula. Dicha interfaz permite al usuario ver gráficamente los valores devueltos por la brújula en un plano cartesiano en tiempo real.

Para hacer esto posible, se ha programado un hilo que, con una frecuencia de 500 milisegundos, envía comandos de lectura a la brujula; esto pemite que la lectura de los datos de la brujula sea interpretado por el usuario como en tiempo real sin saturar el procesador realizándolo con mayor frecuencia.

La figura 4.6muestra la interfaz gráfica diseñada, la cual se comunica con la brújula NXT y muestra en tiempo real la lectura de la misma en un plano cartesiano fácilmente comprensible por el usuario. La seccion a) permite al usuario capturar el puerto COM que ha sido asignado por el sistema operativo; cabe mencionar que cuando un dispositivo serial es conectado a una computadora, el sistema operativo de la misma puede asignarle un puerto COM distinto en cada caso dependiendo de la disponibilidad de puertos que se tengan, es por eso que este valor se ha dejado como entrada al usuario. La seccion b) son botónes que permiten al usuario iniciar o detener la lectura de la brújula. La seccion c) es la parte que permite al usuario ver gráficamente la lectura actual registrada por la brújula utilizando los valores mencionados anteriormente.

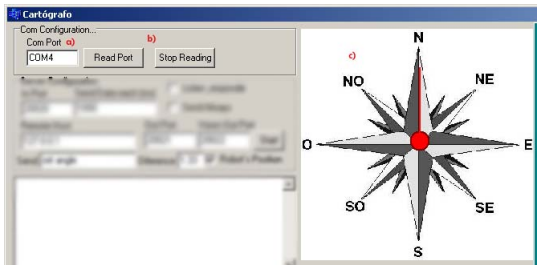


Figura 4.6: Ventana gráfica de comunicación y lectura con la brújula

Es importante mencionar, que hasta el momento, sólo se ha hablado de la parte gráfica de salida al usuario; en secciones posteriores se hablará del papel que juega internamente la lectura de la brújula para lograr el objetivo deseado.

4.1.3. ARToolKit Plus

Como se ha venido mencionando en esta tesis, el objetivo principal no es desarrollar un sistema de visión o reconocedor de patrones; sin embargo, el presente proyecto

no podría ser posible sin la existencia de un buen sistema de visión, ya que forma parte principal para lograr nuestros objetivos.

Por lo tanto, utilizaremos esta herramienta poderosa de visión y de reconocimiento de patrones. ARToolKit era en un principio la herramienta idónea a utilizar, sin embargo, se encontraron ciertos problemas como que la cantidad de marcas que podían ser reconocidas en ARToolKit era muy limitada; ARToolKit ya viene con un sistema para el manejo de cámaras utilizando DSVideolib, GLUT y OpenGL, lo que lo hacía muy pesado; sobre todo en el manejo de tres cámaras simultáneas.

Tras investigar arduamente en otras posibilidades, se encontró una mejor opción que no afectaría mucho el tiempo de desarrollo y el trabajo que ya se tenia realizado hasta el momento, ARToolKit Plus. En el capítulo anterior se definen las razones por las cuales se decidio utilizar esta herramienta.

A diferencia de ARToolKit, ARToolKit Plus no viene integrado con ningún sistema para el control de cámaras; lo que lo hace más ligero computacionalmente hablando; así mismo está optimizado para funcionar en dispositivos portátiles, por lo que sus requerimientos de recursos computacionales son considerablemente menores en comparación con ARToolKit.

4.1.3.1. Integración de OpenCV en ArtoolkitPlus

ARToolkitPlus fue diseñado para extraer la información de la pose de los marcadores a partir de imágenes en escala de grises, las cuales pueden provenir de archivos en disco o cámaras de vídeo.

Para esta tesis, se utilizó OpenCV[Int99] para usar cámaras de vídeo como entrada de imágenes para el rastreador de ARToolkitPlus. Un detalle acerca del rastreador de ARToolkitPlus es que las imágenes de entrada deben estar representadas como un arreglo unidimensionales de valores enteros entre 0 y 255. Esto representa un potencial problema con OpenCV, debido a que sus estructuras de datos para representar imágenes tienen información adicional que no requiere el rastreador.

Con el fin de proporcionar arreglos unidimensionales en escala de grises al rastreador de ARToolkitPlus, se deben hacer los siguientes pasos por cada cuadro del flujo de vídeo:

1. Se genera una copia del cuadro capturado.
2. La copia del cuadro se convierte a escala de grises.
3. Se copian a un arreglo unidimensional todos los elementos de la matriz de los píxeles que forman a la copia del cuadro.

Como parte de la integración entre OpenCV y ARToolkitPlus, se creó una clase de captura de vídeo que puede devolver un arreglo unidimensional de valores

de escala de grises, así como imágenes en formato de OpenCV para mostrar la imagen que se está capturando.

Los arreglos unidimensionales que regresa la clase se pasan al rastreador de ARToolkitPlus para la estimación de pose de los marcadores. Al principio se trató de utilizar un rastreador para todas las imágenes que mandaban las tres cámaras, pero el comportamiento del rastreador no fue el esperado, ya que se utilizaba la primer imagen capturada en todas las llamadas a la función del rastreador, aún cuando las imágenes eran diferentes; así que la decisión final de diseño fue asignar un rastreador a cada cámara.

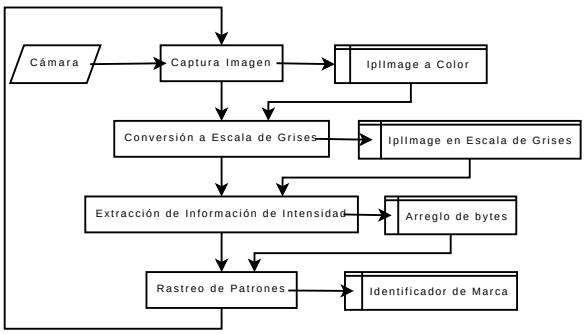


Figura 4.7: Diagrama de Integración de Open CV con ARToolKit Plus

4.1.3.2. Calibración de las cámaras

Las propiedades internas de una cámara: coordenadas del centro de la imagen, distancias focales en los ejes X e Y y factores de distorsión en los ejes X e Y, sirven para estimar con la mayor precisión posible la distancia entre una cámara y una marca de ARToolkitPlus, así como corregir la distorsión de imagen que se produce por la lente de la cámara.

Al proceso en el que se obtienen las propiedades internas de la cámara, también llamadas parámetros extrínsecos, se le llama calibración, el cual consiste en tomar una serie de imágenes de una estructura geométrica conocida, para estimar la forma en que la óptica de la cámara altera las imágenes y así poder revertirlas.

ARToolkitPlus no tiene herramientas para calibrar cámaras de vídeo, los desarrolladores recomiendan usar los programas de calibración de ARToolkit o el Camera Calibration Toolbox para Matlab.

Considerando que no se disponía de MatLab, se utilizó el programa de ARToolkit para calibrar en un solo paso, el cual utiliza una hoja de papel con un patrón de 24 puntos, acomodados en 4 filas de 6 puntos cada uno (Fig. 4.8a).

Al iniciar el programa de calibración, se pregunta al usuario cuál es la distancia entre los puntos del patrón, la cual en una hoja tamaño carta es de 4 cm, éste es uno de los valores conocidos para poder estimar los parámetos de la cámara.

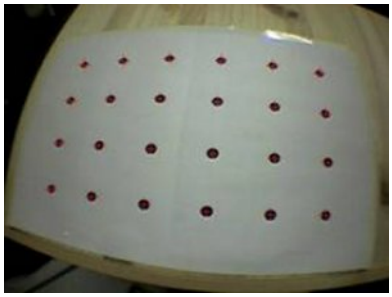
1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Cuadro 4.1: Orden de los puntos en el patrón de calibración

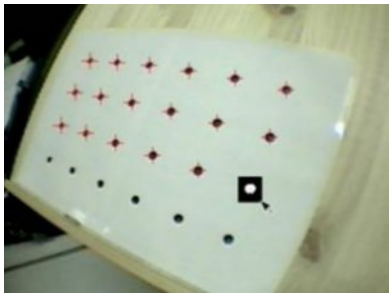
Una vez que se introduce este valor, aparece una ventana que muestra el flujo de imágenes que captura la cámara, la cual debe estar apuntando hacia el patrón de tal forma que todos los puntos sean visibles. En este momento se hace clic con el botón izquierdo el ratón para capturar una imagen del flujo de vídeo. Sobre la imagen se marcan los puntos de calibración, arrastrando el ratón para dibujar un rectángulo que cubra cada punto (Fig. 4.8b); los puntos deben ser cubiertos en el orden que marca la Tabla 4.1.

Una vez que se seleccionan los 24 puntos, se hace clic con el botón izquierdo del ratón para guardar los puntos y volver a activar el flujo de vídeo. El procedimiento se repite para 5-10 imágenes, las cuales deben ser tomadas desde diferentes ángulos y posiciones. Una vez cubierto ese requisito, se hace clic con el botón derecho para detener la captura de imagen e iniciar el proceso de calibración.

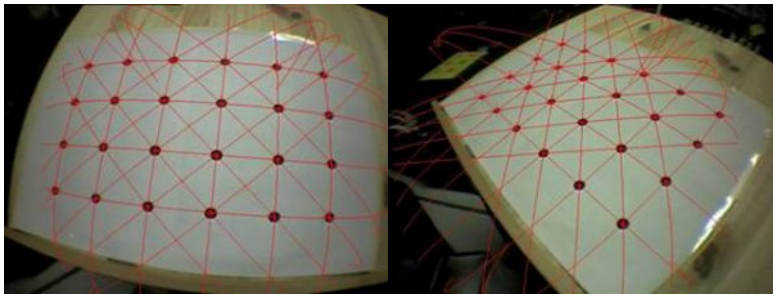
Los cálculos del proceso de calibración toma un momento para completarse, al final se obtienen los valores del centro de la imagen en X e Y y el factor de distorsión de la cámara, así como una demostración visual de la precisión de los parámetros de la cámara (Fig. 4.8c)[Lam03]



(a) Patrón de calibración



(b) Seleccionado puntos



(c) Calibración correcta

4.1.3.3. Comunicación con sistemas externos

El sistema de visión no puede ser, por ningún motivo, cerrado; este debe ser capaz de comunicarse con otros sistemas (en nuestro caso el cartógrafo) para notificarles la información que está viendo en cada momento. ARToolKit plus no viene integrado con ningún sistema de comunicación, por lo que una de las cosas que tuvieron que ser agregadas fue, precisamente, la integración de sockets UDP que permitan al sistema de visión estar en comunicación con el mundo exterior.

Cuando el sistema de visión recibe la petición por parte de cartógrafo, este contruye una cadena de texto con la información que le arrojan sus tres cámaras. La figura 4.8 muestra el flujo de operaciones que se realizan cuando ARToolKit Plus procesa la información de cada una de sus cámaras para detectar los patrones que se encuentran en cada rango de visión y calcular así las distancias que existen a los mismos. Nótese que de acuerdo al flujo, el sistema es capaz de funcionar para N cantidad de cámaras, sin embargo, para propositos de este proyecto, ARToolKit Plus siempre está configurado para trabajar únicamente con tres cámaras.

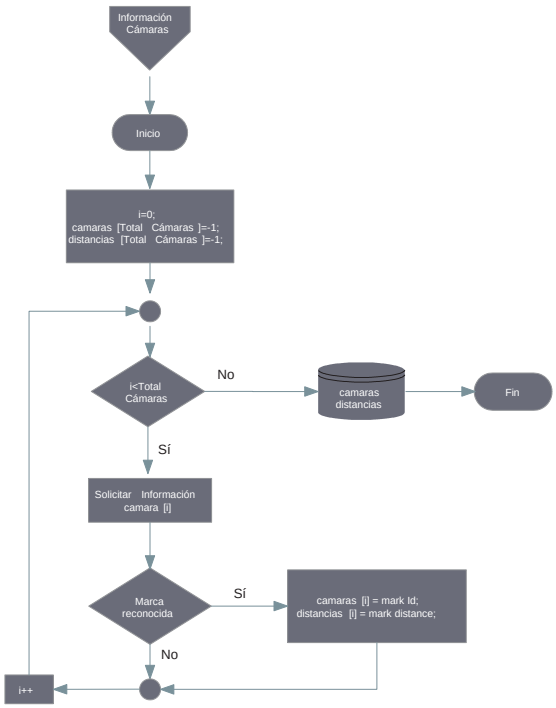


Figura 4.8: Flujo del proceso de obtención de información por parte de las cámaras

En un principio, este proceso resultaba lento; esto se debía a que el sistema de visión esperaba a recibir la solicitud, para entonces procesar los registros de las cámaras, realizar reconocimientos de patrones de marcas y cálculos de distancias, posteriormente construir una respuesta de las tres cámaras y enviarlas de regreso al cartógrafo. Para resolver este problema, se decidió realizar este proceso complejo antes mencionado de manera asíncrona, utilizando un hilo, el sistema de visión recopila información de sus cámaras y realiza los procesos descritos con anterioridad. De esta manera, cuando el cartógrafo envía la solicitud, la visión puede regresar una respuesta de manera casi inmediata. Para asegurar que la información resultante sea en tiempo real, este proceso asíncrono se realiza en un cierto periodo considerable de tiempo. Cuando la visión detecta que ha pasado un cierto lapso de tiempo sin recibir una petición, este proceso se pausa para ahorrar así recursos de cómputo, el proceso se activa nuevamente en la siguiente petición.

4.1.4. Triangulación

La triangulación es una de las partes escenciales del sistema, es aquí donde se realiza el proceso más importante de la localización. Como se mencionó

anteriormente, el sistema de visión está encargado de enviar al cartógrafo información de las marcas que están siendo detectadas y la distancia que se encuentra el sujeto de cada una de éstas. Con dicha información, el cartógrafo la procesa utilizando cálculos de triangulación para obtener la posición bidimensional del sujeto.

Como se mencionó en el capítulo anterior, para poder encontrar la posición de un punto determinado, es necesario conocer por lo menos la distancia que existe de este punto hacia otros tres puntos alrededor, la limitante de estos tres puntos es que no basta con saber cuales distancias existen hacia ellos, sino también se debe conocer la posición precisa que tienen en el escenario.

Por lo tanto, en base a lo anterior, se tiene un sistema de visión que es capaz de detectar e identificar una marca en el ambiente así como calcular la distancia que existe entre la cámara y dicha marca. Lo siguiente, para poder realizar la triangulación, es conocer la posición (x, y) en la que se encuentran cada una de estas marcas.

4.1.4.1. Las marcas en el escenario

Las marcas en el escenario tendrán una posición arbitraria, por lo que al momento de definir el lugar en el que se encontrarán, se debe conocer su posición (x,y) exacta; la forma de conocer esto es midiendo desde un origen para cada uno de los ejes; este origen deberá ser el mismo que se definió como origen del escenario; esto se hace para todas y cada una de las marcas. Un primer problema que se encontró, al definir estas marcas y hacer la medición de su posición, fue las grandes distancias que pueden existir entre el origen del escenario con algunos de los puntos más alejados del mismo.

Para ejemplificar lo anterior, puede imaginarse una casa con distintas habitaciones; como sólo puede tenerse un origen, se define una de las esquinas de la casa como dicho origen, sin embargo, tenemos que una de las habitaciones se encuentra en el extremo opuesto de este punto. Este problema puede verse gráficamente en la figura 4.9 donde se tiene un origen para toda la casa, posteriormente se definen marcas en cada una de las habitaciones; puede observarse que realizar una medición entre el origen y las marcas A y B puede no ser tan complicado, sin embargo, para las marcas C y D realizar una medición puede ser algo realmente complejo, dada las distancias tan alejadas que existen entre los puntos y las paredes que hay.

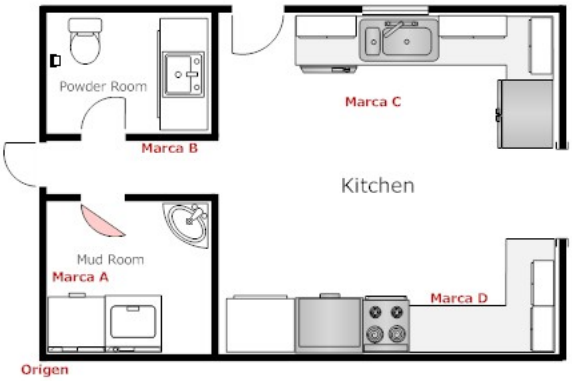


Figura 4.9: Problema para realizar mediciones entre el origen y cada una de las marcas.

Para resolver este problema, se ha decidido dividir el escenario en zonas, donde cada zona tiene su "origen local" y las mediciones a las marcas se realizan en relación a este origen de la sección donde pertenecen. La figura 4.10 muestra como ha sido dividido el escenario en distintas zonas, cada zona tiene un origen local quien a su vez depende del origen global (origen del escenario) de esta manera, cada marca puede medirse y encontrar posición x,y respecto al origen de la zona en la que se encuentra y es así como se almacena.

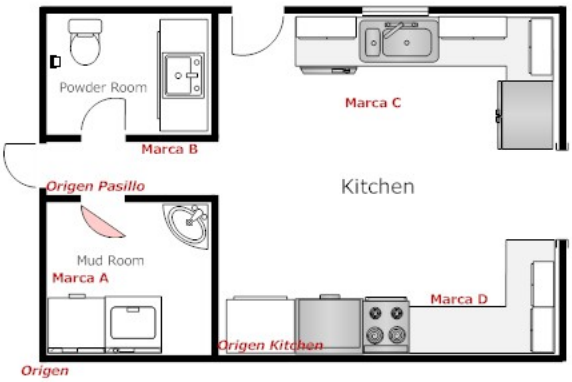


Figura 4.10: Escenario dividido en zonas para resolver problema de medición.

4.1.4.2. Representación de las marcas en archivo

La representación de la información anterior se hace mediante archivo XML donde se indican todas las zonas que existen en el escenario, así como la posición x,y del origen local de cada una; en este archivo posteriomente se definen las marcas que se encuentran en cada zona así como su posición relativa.

La información anterior se almacena en un archivo XML cuyo esquema es el siguiente:

```
<?XML version="1.0" encoding="UTF-8"?>
<xs:schema XMLns:xs="http://www.w3.org/2001/XMLSchema"
          elementFormDefault="qualified"
          attributeFormDefault="unqualified">
         <xs:element name="marks" type="marksType">
                  <xs:annotation>
                           <xs:documentation>Define the marks information</xs:documentation>
                  </xs:annotation>
         </xs:element>
         <xs:complexType name="marksType">
                  <xs:sequence maxOccurs="unbounded">
                           <xs:element name="zone" type="zoneType"/>
                  </xs:sequence>
         </xs:complexType>
         <xs:complexType name="zoneType">
                  <xs:sequence maxOccurs="unbounded">,
                           <xs:element name="mark" type="markType"/>
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string"/>
                  <xs:attribute name="x" type="xs:float" use="optional" default="0.0"/>
                  <xs:attribute name="y" type="xs:float" use="optional" default="0.0"/>
         </xs:complexType>
         <xs:complexType name="markType">
                  <xs:attribute name="name" type="xs:string" use="required"/>
                  <xs:attribute name="x" type="xs:float" use="required"/>
                  <xs:attribute name="y" type="xs:float" use="required"/>
                  <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
         </xs:complexType>
</xs:schema>
```

En el esquema puede apreciarse que existe un tipo de dato raíz (marksType) el cual es una colección de zoneType, este último tipo de dato encapsula las marcas que contiene así como la posición de cada una de sus ellas y de su origen local.

El esquema anterior produciría un archivo XML como el siguiente:

```
<?XML version="1.0" encoding="UTF-8"?>
<marks XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="marks.xsd">
   <zone name="mud room" x="0.0" y="0.0">
         <mark id="1" name="Marca A" x="2.5" y="50.0" />
         </zone>
   <zone name="kitchen" x="30.0" y="0.0">
         <mark id="5" name="Marca C" x="53.13" y="72.4" />
            <mark id="6" name="Marca D" x="40.51" y="72.4" />
         </zone>
   <zone name="pasillo" x="0.0" y="10.0">
        <mark id="10" name="Marca B" x="16.5" y="50.0" />
   </zone>
</marks>
```

Puede apreciarse en el archivo XML de ejemplo como se configuran las zonas que existirán en el escenario, así como la posicion x,y en donde se encuentra su origen local; es importante mencionar que estas coordenadas se definen en referencia al origen real del escenario, este origen siempre tendrá las coordenadas 0,0 ya que es apartir de este punto donde se definen las coordenadas de todos los objetos involucrados en dicho escenario.

Siguiendo la definición del esquema, puede observarse que es necesario siempre tener al menos una zona y es aquí donde se definirán las marcas. Es posible que exista el caso en donde se tenga un escenario el cual no requiere ser dividido en zonas, para esto, únicamente basta con declarar una zona con origen local en (0,0).

Posteriormente se definen las marcas que se encuentran dentro de cada zona. Cada marca cuenta con los siguientes atributos: ***id***, este es un identificador único para cada marca, el sistema de visión envía un identificador de la marca que está observando, el cartografo utiliza este identificador para empatarlo con las marcas definidas y saber así la posición en la que se encuentra; ***name***, se utiliza como manera de identificar más fácilmente las marcas que se tienen, este dato es arbitrario y dado por el usuario; ***x***, coordenada en x de la marca con respecto al origen local de su zona; ***y***, coordenada en y de la marca con respecto al origen local de su zona.

4.1.4.3. Programando el cálculo de la triangulación.

Una vez definidas las marcas y sus respectivas posiciones en el escenario, es ya posible programar el cálculo de la triangulación. Para esto se ha programado un analizador que lea el archivo XML y convierta los valores a estructuras de memoria. Una vez teniendo esta información procesable en memoria, se procede a realizar los cálculos pertinentes.

En el capítulo anterior se explicó en que consiste el método de triangulación y como funciona éste. Se mencionó la información que debe conocerse previamente para poder realizar la localización, la cual consiste en:

1. Un punto bidimensional p el cual es la incognita a encontrar.
2. Tres puntos c_1,c_2 y c_3 con posiciones bidimensionales conocidas.
3. Tres distancias d_1, d_2y d_3 correspondientes a las distancias existentes entre p con c_1,c_2 y c_3, respectivamente.
4. Observando los dos apartados anteriores en términos de geometría, se tiene que c_i y d_i forman una circunferencia, siendo c_i el centro y d_i el radio para la *i-ésima* circunferencia.
5. El punto p, que es la incognita, es a su vez la intersección de las tres circunferencias anteriormente mencionadas.

En base a lo anterior se deduce que se cumplen todos los requisitos previos, por lo que lo consiguiente es representar lo anterior en estructuras de datos que sean procesables. La figura 4.11 muestra el diagrama de clases necesarias para que el cálculo de la triangulación sea posible.

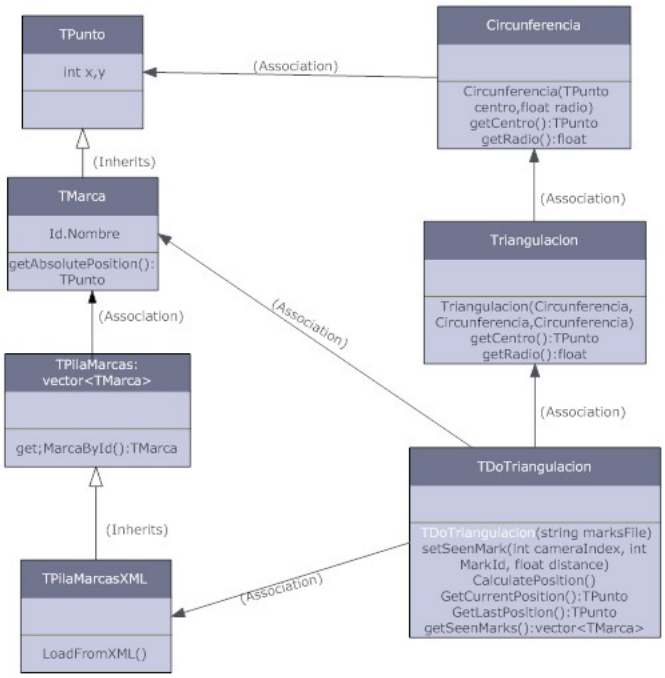


Figura 4.11: Diagrama de clases para el cálculo de la triangulación.

- La clase *TPunto* es utilizada en diferentes procesos del sistema y encapsula los valores para las coordenadas (x,y).
- Debido a que, como se ha venido mencionando, una marca contiene una posición bidimensional en el escenario (x,y), se tiene una clase *TMarca*, que hereda de *TPunto*, la cual agrega el identificador de la marca, el nombre y una posición absoluta (*getAbsolutePosition*) que devuelve un *TPunto* de la posición absoluta con respecto al origen del escenario; los métodos x(), y() devuelven la posición con respecto al origen local.
- La clase *TPilaMarcas* es un vector de *TMarca* que permite almacenar n cantidad de éstas y acceder a alguna de ellas en particular por medio de su identificador. La clase *TPilaMarcasXML* hereda de *TPilaMarcas*, sin embargo, agrega la particularidad de realizar el parseo del archivo XML definido anteriomente para de esta manera crear las instancias de *TMarca* en memoria.
- Para realizar la triangulación se deben tener al menos tres circunferencias cuyos radios intersecten en algún punto que se desea encontrar; para ello, se tiene la clase *Circunferencia*, que encapsula la información de la posición del centro de la misma y su radio. Hasta el momento puede apreciarse que, si se tiene un objeto *TMarca* y una distancia r a cierto punto,

esto se convierte en un objeto *Circunferencia*. Esta clase será la utilizada posteriomente para realizar el cálculo.

- Una vez habiendo definido las clases anteriores, se tiene una de las clases principales; la clase *Triangulacion* que es la encargada de realizar los cálculos de la triangulación y regresar un punto resultante. Para esto, recibe como parámetros en su constructor las trés circunferencias; esta clase se encargará entonces de encontrar la intersección de los tres radios que será el punto o posición resultante.

- Por último, se tiene una clase que se encarga de construir y encapsular a las demás, así como de llamar los métodos necesarios para realizar los cálculos pertinentes. La clase *DoTriangulacion* realiza lo antes descrito. Su constructor recibe la ruta del archivo XML que contiene la configuración de las marcas, con esto se crea una instacia *TPilaMarcasXML*. El método *setSeenMark* es utilizado para notificar que marca está siendo vista, a que distancia y por medio de cual de las tres cámaras; recibe como parámetros el índice de la cámara, el identificador de la marca y la distancia; llamando este método para las 3 cámaras del sistema de visión, se crean las tres circunferencias necesarias. El método *CalculatePosition* es el corazón de esta clase pues se encarga de instanciar los objetos necesarios para que se realice el cálculo de la triangulación. El método *getSeenMark* devuelve un vector con referencias a las marcas que han sido vistas por el sistema de visión. El método *getCurrentPosition* devolverá un objeto *TPunto* que indica la posición resultante de realizar la triangulación, si por algún motivo el sistema no ha podido realizar el cálculo (no hay suficiente información de la visión, etc) este método regresara la posición (-1,-1). El método *getLastPosition* devolverá la última posición que pudo ser calculada por la triangulación antes de la posición actual.

4.1.5. Comunicación ARToolKit - Cartógrafo

Hasta el momento se tienen dos entidades, cada una realizando una función muy específica. Por una parte una de ellas es la encargada de la visión y detectar patrones en el ambiente, así como saber la distancia a la que se encuentran los mismos. Por otro lado, se tiene una entidad que es capaz de procesar esta información para devolver un dato específico y de gran importancia para esta tesis: la posición de una persona.

4.1.5.1. Solicitar Información al sistema de visión

Es necesario definir e implementar una comunicación entre ambas entidades, donde el cartógrafo realizará una petición al sistema de visión y éste le responderá con la información de las marcas que están siendo detectadas por cada una de sus cámaras.

La comunicación se desarrolló a través de sockets UDP; se decidió utilizar este protocolo ya que presenta una mayor velocidad en la transmisión de la información al no utilizar algoritmos de corrección de errores; para cuestiones de la presente tesis lo anterior no presenta ningún problema.

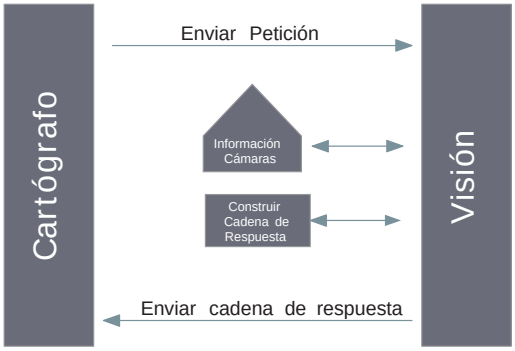


Figura 4.12: Esquema de comunicación entre el sistema de visión y el cartógrafo.

La figura 4.12 muestra el flujo que hace posible la comunicación antes mencionada. El cartógrafo envía un comando al sistema de visión solicitándole la información de lo que ve en este momento; la visión a su vez envía una respuesta con la información solicitada.

4.1.5.2. Procesar Información de la Visión

Cuando el cartógrafo ha recibido la cadena de respuesta por parte del sistema de visión, este deberá realizar un análisis de esta información para posteriormente procesarla.

La información que se recibe del sistema de visión es la registrada por tres cámaras, por lo tanto vienen tres segmentos con el mismo formato de datos. El formato es el siguiente:

cameraIndex,markId,distance	**Donde:** **cameraIndex:** Índice de la cámara que está registrando la información, el índice para la primer cámara es 0. **markId:** Índice de la marca reconocida. **distance:** La distancia en milímetros de la marca con respecto a la cámara

Este fragmento de información es enviado por cada cámara que tenga registrado el sistema de visión; por lo que para el caso del sistema de visión utilizado, la cadena completa de respuesta es la siguiente:

```
#cameraIndex,markId,distance;cameraIndex,markId,distance;cameraIndex,markId,distance
```

Se puede apreciar que el formato se repite tres veces, ya que son tres las cámaras utilizadas, separadas con punto y coma. La cadena inicia con numeral para indicar que la cadena que se está enviando es una respuesta a la solicitud recibida y no cualquier otro tipo de información.

Como se ha venido mencionado, esta información se envía por cada cámara de la visión, la cual expresa las marcas que se están reconociendo, por ejemplo, supóngase que el cartógrafo realiza una petición a la visión donde ésta se encuentra actualmente viendo lo siguiente:

- La primer cámara está viendo la marca 3 a una distancia de 120 mm.
- La segunda cámara está viendo la marca 7 a una distancia de 200 mm.
- La tercer cámara está viendo la marca 5 a una distancia de 100 mm.

En tal caso, la respuesta recibida por el cartógrafo sería una cadena como la siguiente:

```
#0,3,120.00;1,7,200.00;2,5,100.00
```

Sin embargo, no siempre ocurrirá que las tres cámaras estén detectando una marca, en diversas ocasiones ocurrirá que una o más cámaras no estén viendo marca alguna. Supóngase el siguiente caso:

- La primer cámara está viendo la marca 5 a una distancia de 40 mm.
- La segunda cámara no está viendo marca alguna.
- La tercer cámara está viendo la marca 10 a una distancia de 150 mm.

En tal caso, la respuesta recibida por el cartógrafo sería una cadena como la siguiente:

```
#0,3,120.00;1,-1,0.00;2,5,100.00
```

Puede apreciarse en la información recibida que el segundo segmento de la cadena contiene un -1 como identificador de marca. Esto siempre sucederá cuando una cámara no pueda detectar ninguna marca en su rango de visión. De esta manera el cartógrafo puede saber no sólo que cámara está viendo que marca, sino cuando éstas no pueden detectar ninguna.

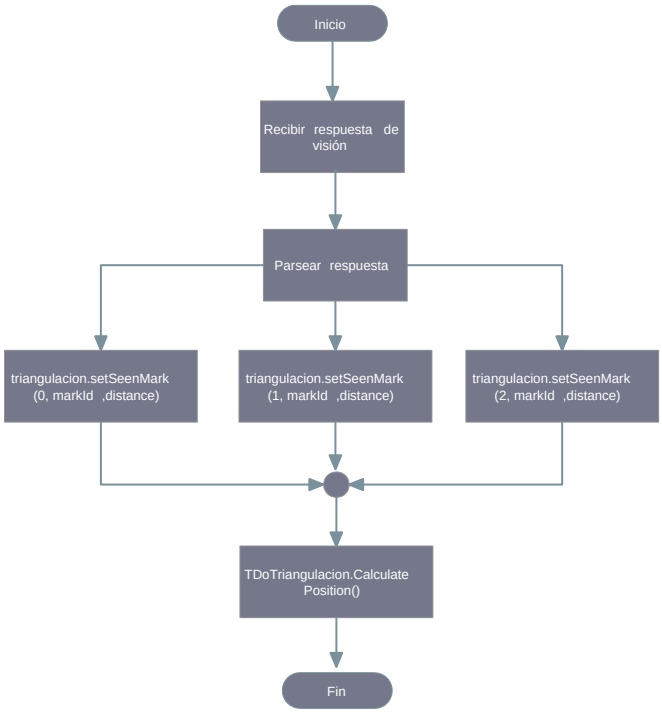


Figura 4.13: Diagrama de flujo de la recepción de respuesta por parte de la visión.

La figura 4.13 muestra un diagrama de flujo de los pasos que se llevan acabo una vez que el cartógrafo ha recibido una respuesta por parte del sistema de visión, este analiza la información y le define a la triangulación las marcas que han sido detectadas con su respectiva distancia; posteriormente se invoca un método para calcular la posición con la información recibida. Si al menos una de las cámaras no detectó una marca, el cálculo de la posición retornará la posición (-1, -1).

4.1.5.3. Representación Gráfica de las Marcas y la Posición del Invidente

Una vez encontrada la posición del invidente por medio de la triangulación, este dato se utilizará para ayudarlo en su navegación, de esto se hablará más a profundidad en la siguiente sección. Para poder tener una mejor apreciación de esta información, se ha decidido graficar esta posición en el mapa que ya se tiene.

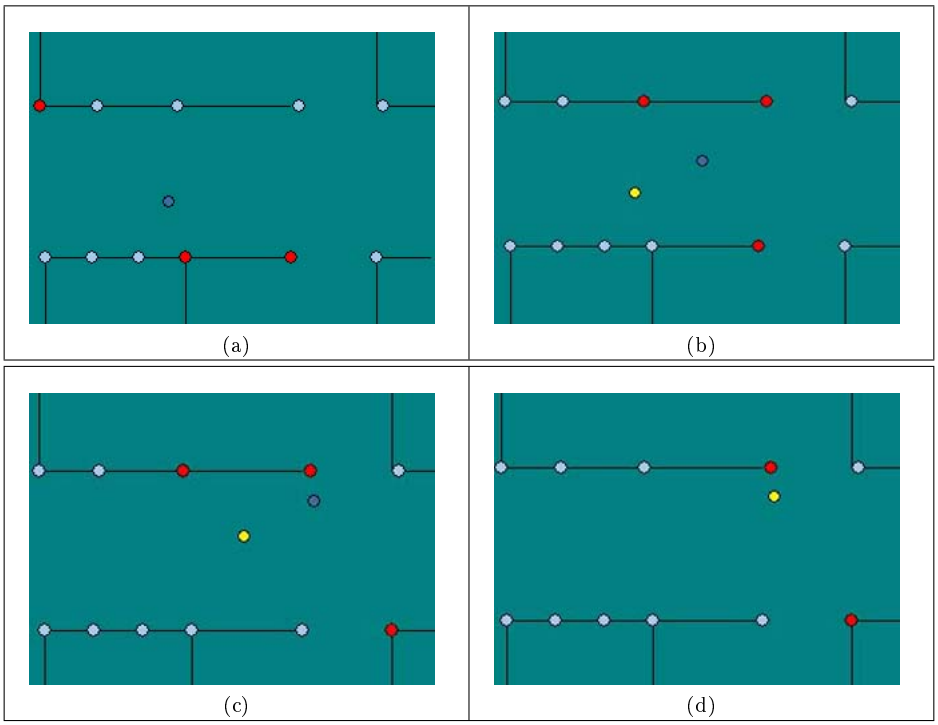


Figura 4.14: Representación gráfica de las marcas detectadas y la posición calculada

El cartógrafo, una vez que ha obtenido la información necesaria para calcular la triangulación, ya es capaz de graficar en el escenario esta información. La figura 4.14 muestra diversas tomas de la información graficada por el cartógrafo. Los círculos de color azul cielo representan las marcas que se encuentran en el escenario, los círculos rojos son marcas también pero que han sido detectadas por la visión; esto es, cada vez que el sistema de visión le hacer saber al cartógrafo cuales marcas son las que está detectando, este las señala de color rojo, de manera tal que se vea visualmente la información de la visión. El círculo azul marino representa la posición del invidente, esto es, la posición encontrada por la triangulación (a).

Cada vez que el invidente cambie su posición, el cartógrafo y la visión trabajan juntos para detectar su nueva posición; el círculo azul marino siempre tendrá la posición actual, que es la calculada, mientras que el círculo amarillo representa la última posición calculada antes de la actual (b); conforme el invidente cambia su posición, cambian así las marcas que la visión detecta, por lo que en cada caso los círculos iluminados con rojo van cambiando indicando las marcas recientemente detectadas (c). Almacenar la última posición calculada es muy útil sobre todo en los casos en los que el sistema no pueda triangularse; se ha venido mencionando frecuentemente que una de las restricciones primordiales para realizar este proceso

es siempre tener tres marcas a la vista; por lo que cuando la visión no detecta las tres marcas, el cartógrafo no puede encontrar la posición; en la figura (d) puede observarse que cuando esto sucede, el círculo azul desaperece; en cuanto existan nuevamente tres marcas detectables por la visión, el cartógrafo puede nuevamente encontrar su posición.

4.2. NAVEGACIÓN

4.2.1. Arbol KD

Para el árbol kd, originalmente se tenía pensado utilizar una biblioteca hecha por Martin F. Krafft, que provee una implementación basada en plantillas STL para C++ para crear árboles kd con cualquier número de dimensiones. Sin embargo, esta biblioteca no es compatible con Borland C++ Builder 6.0, porque su compilador de C++ es incapaz de hacer reemplazos implícitos de tipos de datos al inicializar plantillas. La solución más simple al problema fue crear una clase para árboles kd que usara tipos de datos primitivos de C++, utilizando como base una clase para árboles binarios, a la cual se le agregaron los atributos y funciones para un árbol kd de dos dimensiones con puntos identificados por etiquetas numéricas.

4.2.1.1. Balanceo del árbol

El propósito de balancear el árbol es para optimizar las búsquedas de puntos en el plano, limitando las búsquedas a una sola mitad del árbol. Uno de los problemas con un árbol kd es que no se puede balancer por rotación ya que se puede perder el hiperplano que divide el espacio en cada nivel de árbol, una solución a este problema la propuso [Bær03]en el 2003, la cual consiste en colocar todos los elementos de árbol en un arreglo ordenado e insertarlos de forma recursiva partiendo el arreglo por la mitad en cada recursión.

En el caso de un árbol kd para coordenadas espaciales en dos dimensiones, el arreglo se ordena primero por el eje Y y después por el eje X. La razón de esto es que los niveles pares del arbol kd, incluyendo la raíz, dividen el espacio con un hiperplano en el eje X, mientras que los niveles impares dividen el espacio por el eje Y.

La inserción de los elementos se hace de la siguiente forma: se calcula la posición del elemento en el punto de división del arreglo, el cual se inserta como raiz y se insertan recursivamente las mitades izquierda y derecha del arreglo, calculando el punto de división de la misma forma que la raiz del árbol.

La fórmula para calcular el punto de división del arreglo permite dividir el arreglo de tal forma que el subarreglo a la izquierda sea mayor al subarreglo a la derecha, para que el el árbol esté lleno en su subárbol izquierdo[Bær03].

Las variables que usa la fórmula son las siguientes:

M = potencia de dos más cercana al número de elementos en el arreglo

N = cantidad de elementos en el arreglo

R = Diferencia entre M y N, la cual define la selección de la fórmula para calcular los índices de los árboles izquierdo y derecho dentro del arreglo

LT = Índice del subárbol izquierdo

RT = Índice del subárbol derecho, usualmente es el número de elementos que quedan a partir de LT.

La fórmula para calcular los subárboles se desarrolla de la siguiente forma:

$$R = N - (M - 1) \tag{4.1}$$

$$LT = \begin{cases} \frac{M-2}{2} + R & R \leq \frac{M}{2} \\ \frac{M-2}{2} + \frac{M}{2} & R > \frac{M}{2} \end{cases} \tag{4.2}$$

$$RT = \begin{cases} \frac{M-2}{2} & R \leq \frac{M}{2} \\ \frac{M-2}{2} + R - \frac{M}{2} & R > \frac{M}{2} \end{cases} \tag{4.3}$$

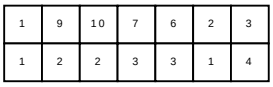
El procedimiento para insertar los datos ordenados en el árbol k-d es el siguiente (Fig. 4.15):

4.2.1.2. Búsqueda de vecinos más cercanos

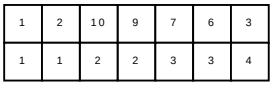
La búsqueda del vecino más cercano en un árbol kd se hace mediante una comparación de la distancia euclidiana entre un punto y los elementos del árbol, cuya diferencia debe ser menor o igual a un radio de búsqueda que se indica al principio de la consulta. La búsqueda se repite recursivamente con un radio cada vez más pequeño hasta recorrer completamente el subárbol en donde inició la búsqueda. Cuando se encuentra el vecino más cercano a un punto, su identificador numérico se utiliza para encontrar la ruta más corta entre ese punto y el punto destino en el espacio mediante algoritmo de Dijkstra.

4.2.2. Algoritmo de Dijkstra

La matriz de adyacencias se llena a partir de un archivo XML que define el mapa topológico mediante puntos en el espacio y las relaciones entre dos puntos. Cuando se llena la matriz de adyacencia se calcula el árbol de las rutas más cortas, mediante algoritmo de Dijkstra, el cual permanece en la memoria del programa hasta que finaliza.

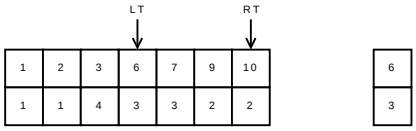


(a) Arreglo de coordenadas 2d sin ordenar

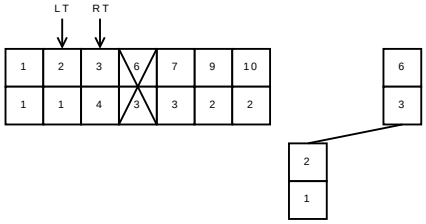


(b) Arreglo de coordenadas 2d ordenado por el eje y

(c) Arreglo de coordenadas 2D ordenado por el eje x

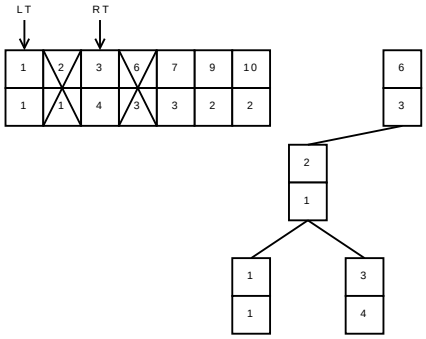


(d) Inserción ordenada en árbol k-d

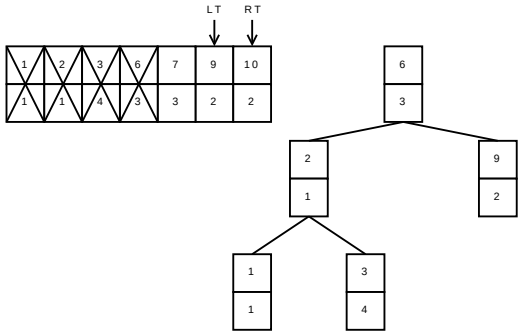


(e) Inserción ordenada en árbol k-d

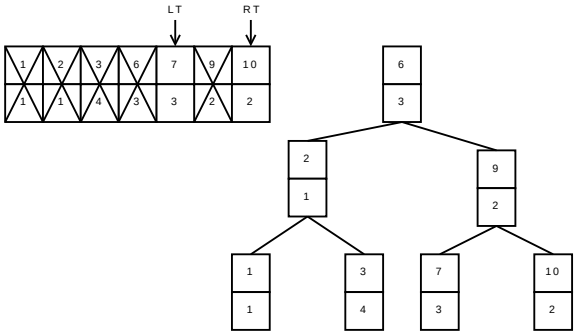
Figura 4.15: Árbol kd balanceado



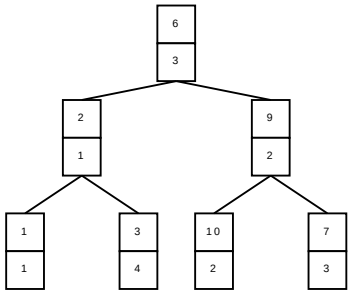
(a) Inserción ordenada en árbol k-d



(b) Inserción ordenada en árbol k-d



(c) Inserción ordenada en árbol k-d



(d) Árbol kd

Figura 4.16: Árbol kd balanceado

Algorithm 4.1 Algoritmo de Inserción en un Árbol KD

1. Se calculan LT y RT aplicando las ecuaciones sobre 4.2 y 4.3 el arreglo completo.
2. El elemento del arreglo referido por LT se inserta en el árbol k-d (Fig. 4.15d).
3. Recursivamente, se calculan nuevos LT y RT para el subárbol izquierdo, $[0 \cdots LT]$, y el subárbol derecho, $[LT + 1 \cdots RT]$ (Fig. 4.15e).
4. El elemento del arreglo referido por LT se inserta en el árbol k-d.
5. Cuando el subárbol tiene un solo elemento, se inserta el árbol k-d.
6. Se repiten los pasos 2 al 5 mientras no se han referido todos los elementos del arreglo.
7. Finalmente se tiene un árbol KD balanceado cuya raíz es el punto que está ubicado lo más cerca posible del centro del conjunto de puntos (Fig: 4.16d).

El algoritmo fue modificado para utilizar pesos de aristas con valores de punto flotante, los cuales se obtienen a partir de las distancias euclidianas entre dos puntos.

4.2.3. Corrección de curso

En la función para corregir el curso se hicieron ajustes a los valores de la brújula, la cual tiene un intervalo de valores entre $[0 \cdots 360°]$, para que pudiera funcionar en un intervalo de ángulos entre $[-180° \cdots 180°]$.

$$\alpha = \begin{cases} \alpha - 360° & si\, 180° < x \leq 360° \\ \alpha + 360° & si\ -180° > x \geq -360° \end{cases} \tag{4.4}$$

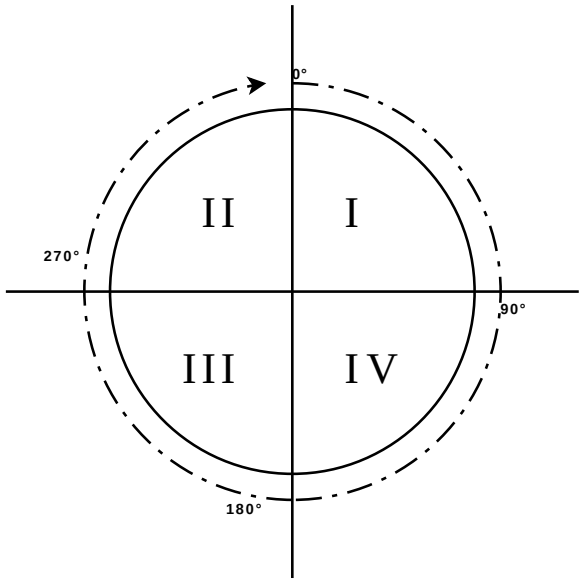
La correción del curso del invidente sigue estos pasos:

1. Actualizar el vector de orientación del invidente con la brújula
2. Actualizar la posición del invidente con las coordenadas del cartografo
3. Calcular el ángulo entre el invidente y el destino en el mapa
4. Indicar la dirección a la que debe voltear el invidente, de acuerdo con la diferencia de ángulos entre la brújula del invidente y el ángulo entre el invidente y el destino.

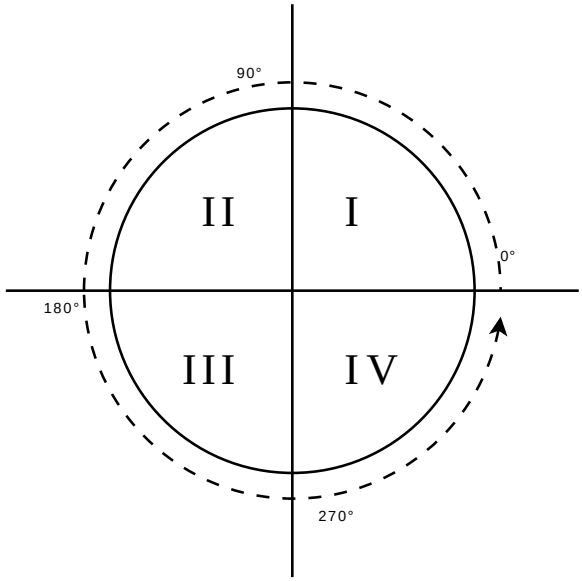
Los pasos se repiten mientras el invidente no llegue al final de la ruta establecida por el planeador de rutas o hasta que el invidente decide cancelar la ruta y seleccionar una nueva.

4.2.4. Ajustando la brújula del sistema de guía

La brújula del Lego MindStorms tiene un sistema angular donde el origen (0°) está alineado con el eje vertical positivo, es decir está apuntando hacia el Norte, y los ángulos se incrementan en sentido horario (Fig. 4.17a), mientras que el software del sistema de guía fue diseñado sobre el sistema angular de un transportador, donde el origen (0°) está alineado con el eje horizontal positivo, es decir está apuntando hacia el Este, y los ángulos se incrementan en sentido antihorario (Fig. 4.17b). Al no ser iguales los sistemas angulares de la brújula y del sistema de guía, es necesario convertir uno de los sistemas en el otro para que la corrección de curso opere con los datos adecuados.



(a) Brújula del Lego Mindstorm



(b) Transportador

Figura 4.17: Sistemas angulares

La solución más simple es convertir el sistema angular de la brújula al sistema angular del software del sistema de guía, ya que de otra manera sería necesario ajustar todas las operaciones asociadas a la corrección de curso al sistema

angular de la brújula. Usando como referencia los cuatro cuadrantes y el sistema angular de un transportador, los ángulos de la brújula progresan en el siguiente orden de cuadrantes: I, IV, III, II (Tab. 4.2).

Cuadrante	Transportador		Brújula	
	Inicio	Fin	Inicio	Fin
I	0°	90°	90°	0°
II	90°	180°	180°	90°
III	180°	270°	270°	180°
IV	270°	0°	0°	270°

Cuadro 4.2: Ángulos por cuadrante

Las operaciones para la conversión de ángulo que dan como resultado un ángulo positivo entre 0° y 360°, según el cuadrante, son las siguientes (Ec. 4.8) :

$$\begin{aligned} \text{I} &= 90° - \alpha \\ \text{II} &= 360° - (\alpha - 90°) \\ \text{III} &= 360° - (\alpha - 90°) \\ \text{IV} &= 360° - (\alpha - 90°) \end{aligned} \tag{4.5}$$

Aquí podemos aplicar una equivalencia del sistema angular de un transportador: cero grados está en la misma posición que trescientos sesenta grados, para que así las restas anteriores tengan las mismas constantes, además se cambia I a $0° - (\alpha - 90°)$ o a $360° - (\alpha - 90°)$, para hacer más evidentes las equivalencias angulares.

$$\begin{aligned} \text{I} &= 0° - (\alpha - 90°) \\ \text{II} &= 0° - (\alpha - 90°) \\ \text{III} &= 0° - (\alpha - 90°) \\ \text{IV} &= 0° - (\alpha - 90°) \end{aligned} \tag{4.6}$$

$$\begin{aligned} \text{I} &= 360° - (\alpha - 90°) \\ \text{II} &= 360° - (\alpha - 90°) \\ \text{III} &= 360° - (\alpha - 90°) \\ \text{IV} &= 360° - (\alpha - 90°) \end{aligned} \tag{4.7}$$

El problema con la primera equivalencia (Ec. 4.6) es que al simplificar las operaciones, la resta queda así $90° - \alpha$, lo cual genera valores negativos en ángulos mayores a noventa grados, lo cual requiere de sumar trescientos sesenta grados al resultado para normalizarlo al sistema angular de un transportador. En el caso de la segunda equivalencia (Ec. 4.7), todos los valores obtenidos son positivos, pero en el intervalo de $[0° \ldots 90°]$ los ángulos obtenidos valen entre $[360° \ldots 450°]$, los cuales valen $[0° \ldots 90°]$ al aplicar la equivalencia angular

$\alpha_c = \alpha \, modulo \, 360°$. Al simplificar las operaciones, éstas quedan de la siguiente forma (Ec. 4.8):

$$\begin{aligned} \text{I} &= 450° - \alpha \\ \text{II} &= 450° - \alpha \\ \text{III} &= 450° - \alpha \\ \text{IV} &= 450° - \alpha \end{aligned} \tag{4.8}$$

Al final, la operación de conversión de sistemas angulares queda de la siguiente forma $\alpha_t = (450° - \alpha_b) \; modulo \, 360°$, el resultado de esta función se pasa al sistema de guía para corregir el curso del invidente.

Ahora se va a demostrar cómo el giro de 450° hace la conversión entre sistemas angulares. Si a cada punto cardinal se le sumara o restara 450°, dos de los puntos cardinales quedarían convertidos a valores que no corresponden al sistema angular de la brújula del Lego Mindstorms (Tab. 4.3).

Para saber la conversión correcta, primero hay que saber cómo cambian los ángulos en los cuatro puntos cardinales de cada sistema angular(Fig. 4.18), los cuales quedan expresados numéricamente de la siguiente forma (Tab. 4.4). El signo del resultado indica en qué sentido se debe hacer el giro de 450° para hacer la conversión correcta del sistema angular: un signo negativo indica vuelta a la derecha mientras que el signo positivo indica vuelta hacia la izquierda, por lo que al aplicar el giro en base al signo del cambio de ángulo, la conversión de sistema angular genera los resultados esperados (Tab. 4.5).

α	$(\alpha + 450°) \, mod \, 360°$	$(\alpha - 450°) \, mod \, 360°$
0°	90°	270°
90°	180	0°
180°	270°	90°
270°	0°	180°

Cuadro 4.3: Conversión directa de sistema angular

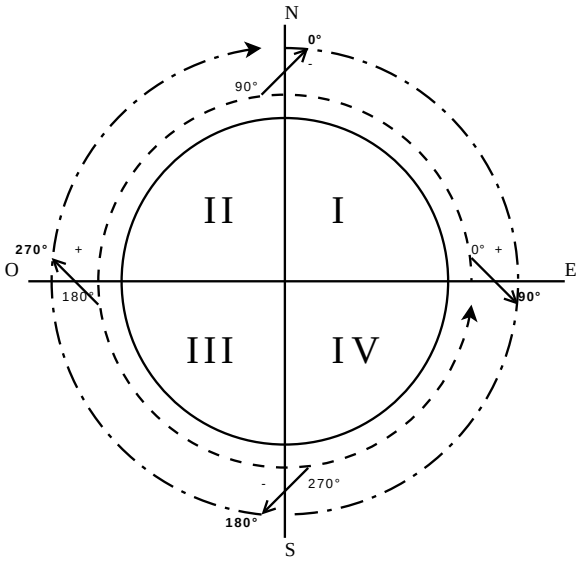


Figura 4.18: Cambio de ángulos entre puntos cardinales

Punto cardinal	Razón de cambio
N	$-90°$
E	$+90°$
S	$-90°$
O	$+90°$

Cuadro 4.4: Cambios de ángulo en puntos cardinales

α	$(\alpha + 450°)\, mod\, 360°$	$(\alpha - 450°)\, mod\, 360°$
$0°$	$90°$	N/A
$90°$	N/A	$0°$
$180°$	$270°$	N/A
$270°$	N/A	$180°$

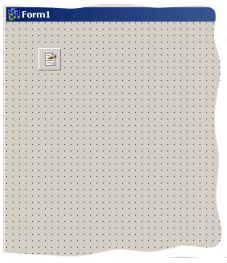
Cuadro 4.5: Conversión de sistema angular basada en el signo de la razón de cambio de sistemas angulares

4.2.5. Integración del Teclado Braille

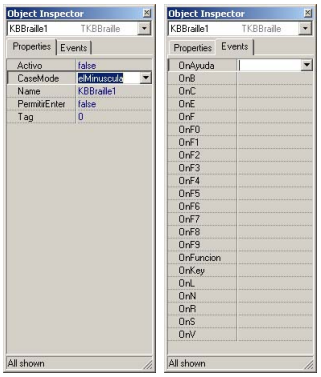
El teclado braille fue desarrollado con un API que permite su fácil integración en cualquier proyecto; esta API viene encapsulada en un componente (llamado

KBBraille) para la VCL de C++ Builder, lo que facilitó aún más su integración al presente proyecto de tesis.

En la figura 4.19 puede observarse como KBBraille se integra rápidamente en una aplicación desarrollada para el entorno C++Builder (Fig: 4.19a). Esto permite de manera sencilla acceder a las propiedades de este componente que son requeridas para manejar distintas opciones del teclado en tiempo de programación y/o ejecución como lo son (Fig: 4.19b): cambiar entre mayúsculas y minúsculas, permitir saltos de línea y activar y desactivar el teclado; para los fines que persigue este proyecto, solo esta última propiedad es útil. Por último, KBBraille también dispone de una serie de eventos programables que se disparan cada vez que el usuario realiza alguna acción en el teclado (Fig: 4.19c). Gracias a lo anterior, de una manera fácil y rápida pueden programarse las acciones que el usuario invidente realizará para que sean interpretadas por el cartógrafo. A manera de ejemplo, si se desea programar el teclado para que cuando el usuario presione la letra B, el cartógrafo le indique al invidente los lugares más próximos a su ubicación actual; sólo hace falta hacer doble click en el evento "OnB", y capturar allí el código necesario para realizar la acción pedida [SSL04].



(a) Componente en una forma



(b) Propiedades(c) Eventos del del ComponenteComponente

Figura 4.19: API en C++ Builder para el Teclado Braille

4.2.6. Salida Auditiva

Como todo sistema, siempre tiene que existir una salida al usuario. Para este propósito, se ha programado una salida especial completamente auditiva que indicará en todo momento lo que está sucediendo en el cartógrafo para que sea notificado al invidente.

Se tienen dos tipos de salida auditiva, una que consiste en ciertos sonidos distintos entre sí que cada uno representa una situación; la otra es mediante voz generada por computadora, para situaciones más específicas.

4.2.6.1. Sonido

El criterio para indicar al invidente cómo corregir su curso es que gire en sentido contrario a la desviación que tiene con respecto al destino en el mapa. Las indicaciones se dan en forma de un sonido que se emite en el oído del lado al que debe voltear. Para esto el sistema usa dos archivos de sonido estéreo, en el que uno de los canales (izquierdo o derecho) tiene un sonido de un clic, el cual sigue sonando hasta que queda alineado con el objetivo. Cuando el invidente llega al destino que se indicó al sistema, éste emitirá un sonido diferente que indica que ya se terminó la ruta. Todos los sonidos son reproducidos a partir de archivos en disco, utilizando una función de la API de Win32 para reproducir sonidos de forma síncrona, es decir, el sonido se reproduce en el momento en que se llama a la función.

4.2.6.2. Voz

Se ha incluido también un sistema de sintetización de voz, más específicamente, se utiliza el TextToSpeech desarrollado por Microsoft. Esto sirve para indicar al invidente mediante voz los puntos disponibles a donde éste pude dirigirse. Toda la información que el invidente necesita conocer, con excepción de la corrección de curso, es dada por este sistema de sintetización de voz.

4.3. CARTÓGRAFO

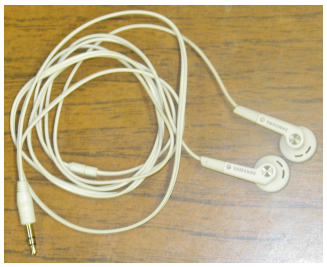
Una vez habiendo implementado cada componente que conforma este proyecto, se presentan en esta sección de manera integrada.

4.3.1. Elementos

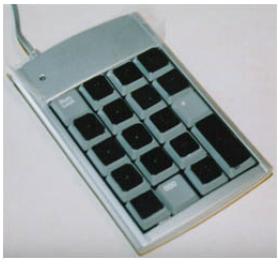
Se habló de que existen tres cámaras que serán las encargadas de hacer posible la triangulación; éstas se integraron en un sombrero (fig: 4.20a), las cuales están en una posición triangular para ampliar el rango de visión de las mismas. El

(a) Sombrero con tres cámaras para localización

(b) Lego Mindstorm NXT con Brújula



(c) Audífonos para Salida Auditiva



(d) Teclado Braille[SSL04]

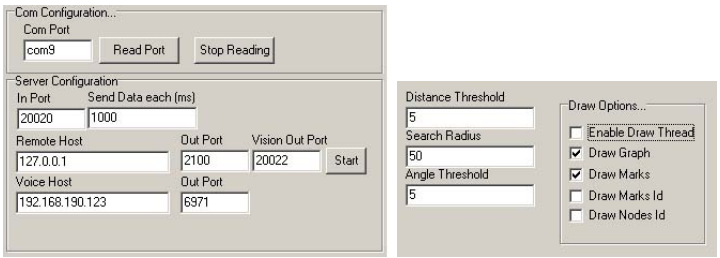
Figura 4.20: Componentes del Cartógrafo.

invidente llevará una mochila (Fig: 4.21) con una computadora portatil pequeña y liviana donde tendrá el sistema cargado. A esta computadora se conectarán las cámaras, el teclado y la brújula. El brick de lego (fig: 4.20b) estará guardado también dentro de esta mochila, dejando por fuera la brújula, la cual irá en el hombro del invidente en dirección de la persona. También se cuenta con unos audifonos (fig: 4.20c) para la salida auditiva del sistema con el usuario; de esta manera el usuario puede conocer por voz los destinos disponibles a donde dirijirse y la debida corrección de curso. Todas las entradas del usuario al sistema serán por medio del teclado braille (fig: 4.20d).

Figura 4.21: Usuario portando el cartógrafo

4.3.2. Flujo de funcionamiento.

Cuando el cartógrafo se inicia por primera vez, varios valores de configuración deben ser capturados (fig:4.22). Se debe configurar el puerto COM por el que se comunicará con la brújula, la comunicación con la visión y el sistema sintetizador de voz (fig: 4.22a). Posteriormente se configuran los valores para el umbral de distancia, el radio de búsqueda para el vecino más cercano y el umbral de ángulo; así como valores para la graficación de la información (fig: 4.22b).



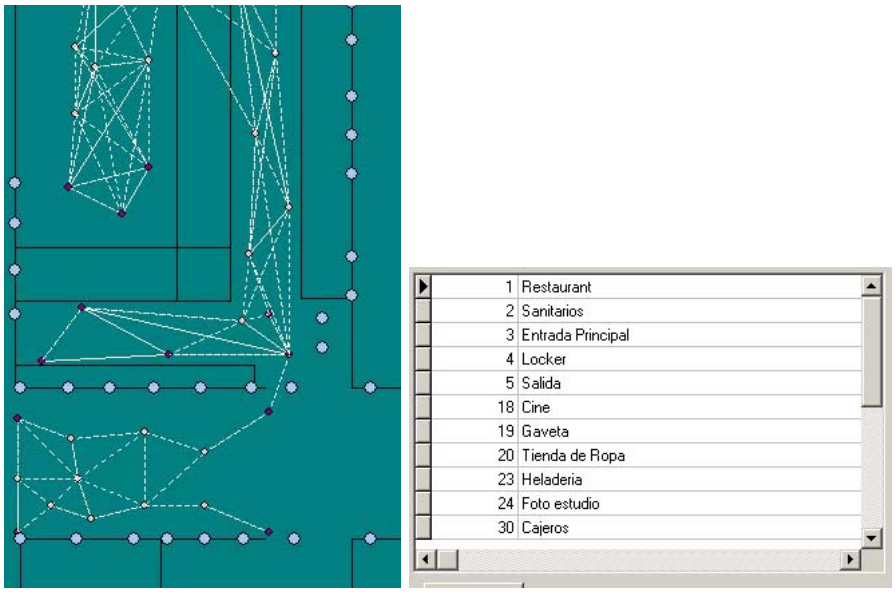
(a) Configuración para Comunicación con Brújula, Visión y Sintetizador de Voz

(b) Configuración de Umbrales y Opciones de Dibujado

Figura 4.22: Parámetros de Configuración

Con los parámetros anteriores propiamente configurados, el sistema ya puede ser utilizado por el usuario. Para esto, se tiene una lista de destinos posibles a los que el usuario puede ir (fig: 4.23b); estos destinos se encuentran dentro del mapa, puede verse en la fig: 4.23a el mapa dibujado, los nodos de color morado

son los destinos disponibles y los nodos blancos son nodos intermedios de ayuda para llegar a dichos destinos. El usuario elije con el teclado braille el destino, una vez hecho esto, el sistema de guía entra en operación para indicarle al usuario los giros que debe realizar y los avances en todo momento hasta llegar a su destino final.



(a) Escenario, Mapa topológico y marcas (b) Destinos disponibles

Figura 4.23: Mapa topológico y destinos disponibles

El cartógrafo, asíncronamente en intervalos de 100 milisegundos, realiza lecturas de la visión para realizar la triangulación y actualizar la posición del invidente, así como su orientación gracias a la brújula. Es importante mencionar que, para que esto ocurra, siempre deberán existir marcas visibles en el entorno.

La figura 4.24 muestra el flujo de operación del cartógrafo desde que el usuario elije un destino hasta que llega al mismo.

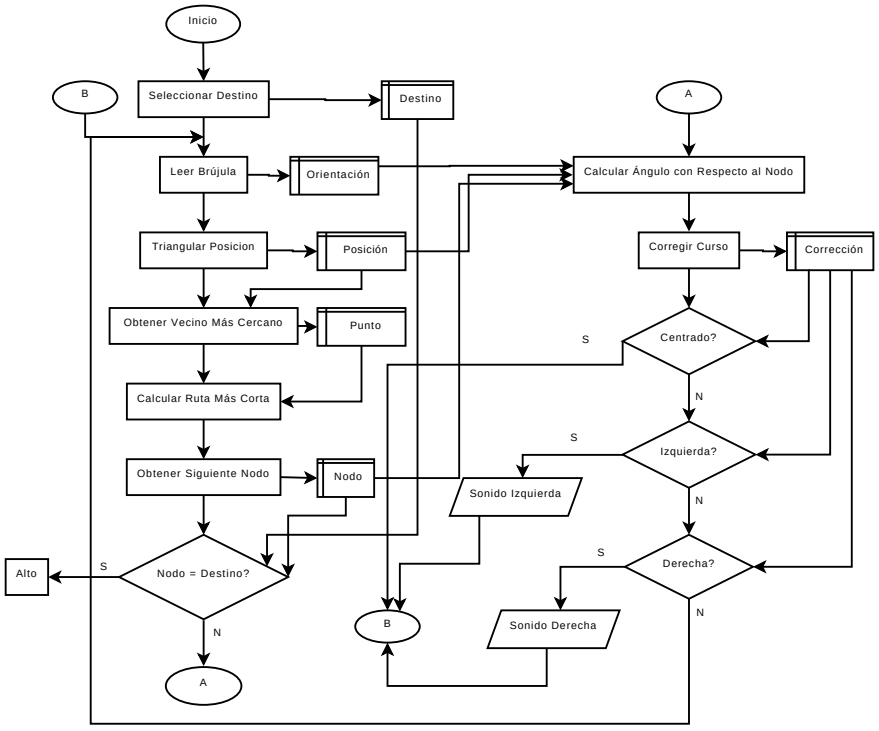


Figura 4.24: Flujo de operación del cartógrafo

Capítulo 5

PRUEBAS Y RESULTADOS

En este capítulo se presentan las pruebas realizadas durante este trabajo que permitieron corroborar la funcionalidad del sistema y encontrar las condiciones para su óptimo desempeño posible.

Para las pruebas que se realizaron del sistema, se pidió la ayuda una personas ajenas al mismo, que no estuvieran familiarizadas con el funcionamiento. De esta manera, con los ojos vendados, se les pidio efectuar las pruebas que a continuación se presentan con sus respectivos resultados.

5.1. LIMITANTES ENCONTRADAS CON EL SISTEMA DE VISIÓN

Antes de realizar pruebas completas con todo el sistema, se decidió realizar pruebas individuales de reconocimiento de patrones por parte del sistema de visión; se encontró lo siguiente, que es muy importante resaltar:

5.1.1. Rango de Visión

Existen algunas limitaciones cuando se trabaja con visión computacional y sistemas de realidad aumentada. Una de las principales limitantes es el hecho de que la marca debe ser completamente visible por la cámara para poder así identificarla. Esto quiere decir, que si en algún momento la marca es parcialmente cubierta por algún otro objeto que se interponga entre ésta y la cámara, el sistema no podrá reconocerla; basta con que sea cubierta una pequeña esquina de la marca para que esto suceda.

Existen también algunas limitantes en cuanto al rango de visión. Entre más grande es el patrón físico (la marca), mayor será la distancia desde la cual

el sistema podrá reconocerla. La Tabla 5.1 muestra algunos rangos de visión aceptados con respecto al tamaño de la marca. Estas mediciones fueron realizadas teniendo la cámara perpendicularmente a la marca y haciéndola retroceder hasta que el sistema dejara de reconocerla.

Tamaño del Patrón (cm)	Rango Máximo (cm)
7	40.6
9	63.5
10.8	86.4
18.7	127.0

Cuadro 5.1: Rango de visión para diferentes tamaños de los patrones

5.1.2. Tipos de Patrones (Frecuencia del Patrón)

Este rango es también afectado por la complejidad del patrón. Esto quiere decir que entre más simple sea el patrón, el desempeño es mejor. Los patrones con sólo regiones cuadradas de blanco y negro (al cual llamaremos patrones de baja frecuencia) son más efectivos. Reemplazando el patrón cuadrado en la distancia de 10.8 cm utilizado anteriormente con un patrón del mismo tamaño pero con mayor complejidad, el rango de visión se redujo de 86.4 a 38.1 cm.

Figura 5.1: Frecuencia de Patrones

La Figura 5.1muestra dos patrones del mismo tamaño pero de diferente complejidad, nótese que aquellos que se consideran de baja frecuencia tienen un diseño más sencillo, generalmente basado en figuras cuadradas; mientras que aquellos con figuras complejas y curvas son considerados de alta frecuencia y tienen un desempeño menor en su utilización al ser de mayor complejidad.

Es importante resaltar que ARToolKit Plus, aunque también puede trabajar con patrones personalizados y de alta frecuencia; cuenta ya con una gama definida de patrones de baja frecuencia; como se ha dicho, estos resultan de mejor desempeño para la visión, aunque sacrifica estética en el ambiente. Para efectos de esta tesis, lo anterior se ha decidido pasar por alto.

5.1.3. Orientación de la Cámara y Luminosidad

La visión también es afectada por la orientación de la marca relativa a la cámara. Mientras la orientación de la marca tiende a ser más horizontal, la cámara tiende

a perder de vista el centro de la marca, por lo que el reconocimiento se vuelve menos eficiente.

Finalmente, y como suele ser obvio en un sistema de visión, el desempeño es afectado por las condiciones de luz. La luz artificial puede crear reflejos en el papel de la marca y por lo tanto hacer más difícil que el sistema encuentre la marca cuadrada.

5.2. OBTENCIÓN DE LOS VALORES DEL SISTEMA DE GUÍA

Durante las etapas iniciales del desarrollo del sistema, se utilizaban valores arbitrarios para el radio de búsqueda del vecino más cercano, el umbral del ángulo de la brújula y el umbral de distancia entre el invidente y un punto del grafo, por lo que era necesario encontrar valores que fueran más acordes con medidas del mundo real y tomaran en cuenta la estructura del entorno y la movilidad del invidente.

Es importante señalar que estos valores estan dados como entrada por parte del usuario, por lo que pueden ser modificados en cualquier momento, sin embargo, las pruebas siguientes se realizan para encontrar los valores adecuados en los que el sistema deberá de funcionar de la manera más óptima.

5.2.1. Radio de búsqueda del vecino más cercano

El valor del radio de búsqueda del vecino más cercano se obtiene a partir del promedio de las distancias entre dos nodos conexos del grafo (Eq. 5.1). Al tomar en cuenta la estructura del grafo para obtener el radio de búsqueda, se obtiene una radio que permite optimizar las búsquedas del vecino más cercano al evitar espacios de búsqueda que son exageradamente grandes o exageradamente pequeños.

$$radio = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}}{n * 2} \tag{5.1}$$

Donde n es la cantidad de aristas cuyo peso sea mayor a cero

El promedio se calcula a partir de las sumas de los pesos de las aristas en la matriz de adyacencia que son mayores de cero, dividido entre el doble de la cantidad total de aristas. La razón para dividir entre el doble de la cantidad de aristas es que como el grado es bidireccional, cada arista del grafo tiene dos instancias dentro de la matriz de adyacencia.

5.2.2. Umbral del ángulo de la brújula

Los invidentes al caminar en línea recta experimentan desviaciones en el ángulo de orientación, que oscilan hacia la izquierda y la derecha. Este detalle se tomó en cuenta a la hora de generar la salida de audio para orientar al invidente, ya que existe la posibilidad de una desviación mayor cuando se haga el giro para corregir el curso .

Para calcular el rango de ángulos (umbral) en el que se considera que el invidente está caminando en línea recta, se hicieron pruebas caminando a ciegas que consistía en caminar en línea recta, dar media vuelta, caminar en línea recta, dar otra media vuelta y caminar en línea recta siguiendo indicaciones verbales de otra persona. Los resultados de la prueba (Fig. 5.2) muestran que cuando se camina a ciegas sin ayuda verbal (Fig. 5.3a, 5.3b), la desviación es menor que cuando se trata de seguir instrucciones (Fig. 5.4); los picos en la gráfica muestran los momentos en que se dió media vuelta.

A partir de los conjuntos de datos donde la caminata se hizo sin ayuda, obtuvo el umbral de desviación mediante el promedio de las diferencias entre dos ángulos consecutivos (Eq. 5.2)

$$\theta = \frac{\sum_{i=2}^{n}(c_i - c_{i-1})}{n} \tag{5.2}$$

5.2.3. Umbral de distancia de proximidad a un punto

En todas las pruebas que se hicieron, se encontró que la persona jamás llega exactamente al punto con las coordenadas específicas, por lo que se maneja un umbran de proximidad que no es otra cosa más que un cierto radio el cual se utiliza como margen para indicar al sistema que cuando la persona esté a una distancia menor o igual de este margen con respecto al punto destino, se consideré que ya se ha alcanzado dicho destino.

Para encontrar un valor óptimo que pudiera satisfacer este problema mencionado, lo que se hizo fue tomar como referencia la longitud que tiene un mosaico en el piso (fig.: 5.5); como todas las pruebas fueron realizadas en el edificio del posgrado de ingeniería, se usaron éstos como modelos. La medida de cada mosaico fue 32 cm

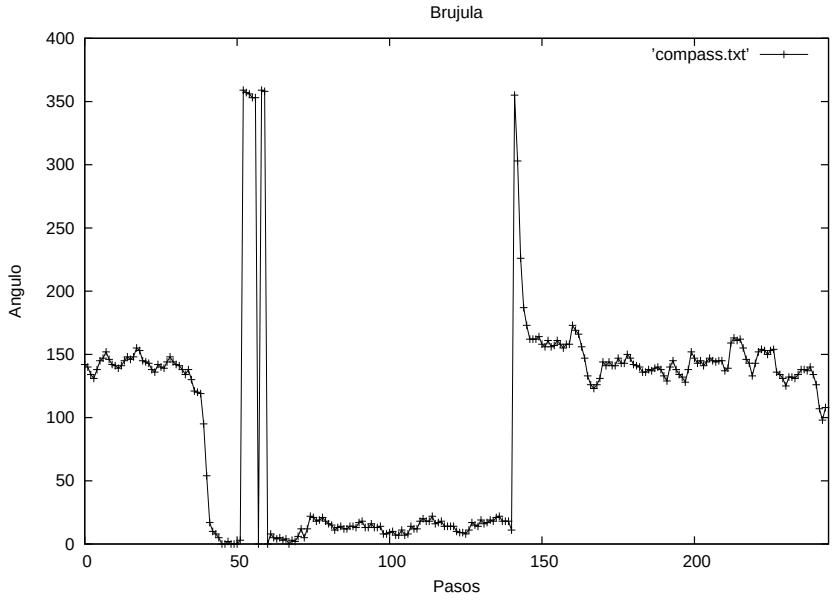
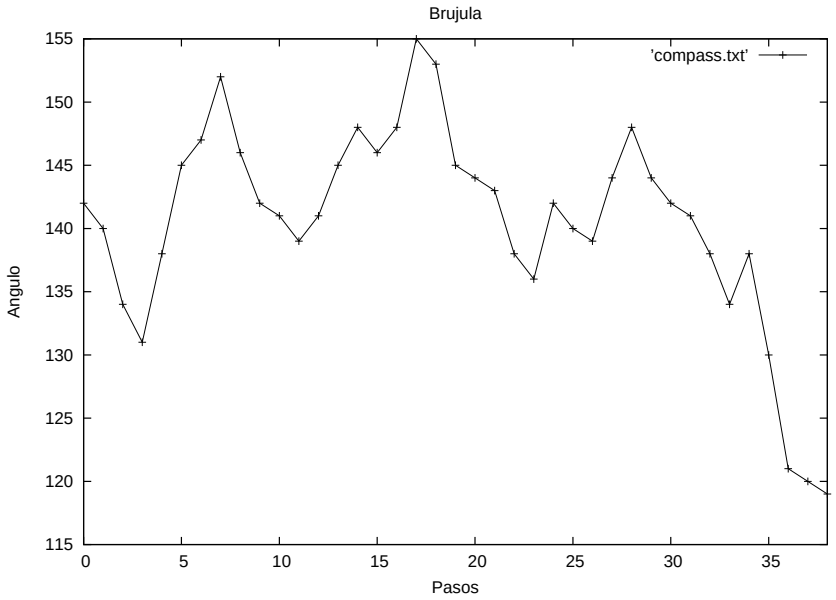
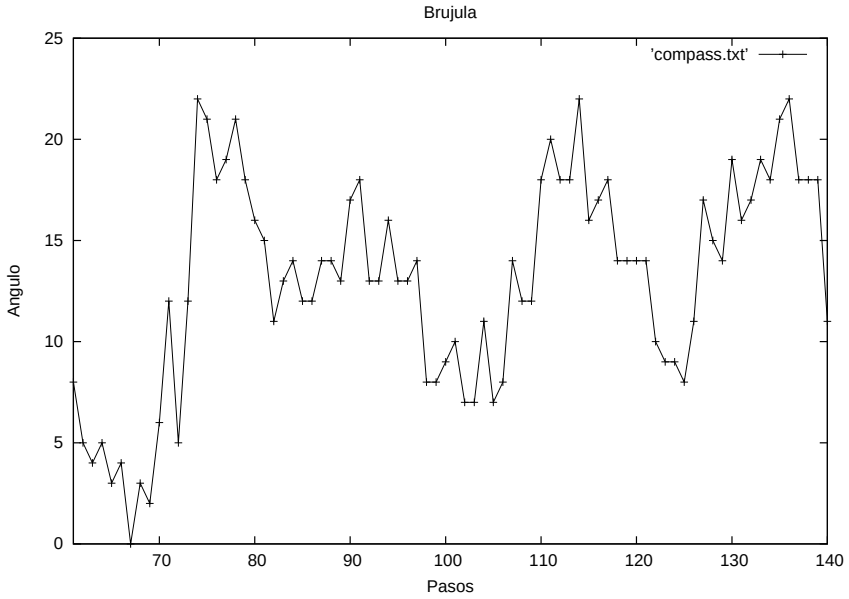


Figura 5.2: Prueba de caminata a ciegas



(a) Sin ayuda verbal



(b) Sin ayuda verbal

Figura 5.3: Muestras de caminatas a ciegas

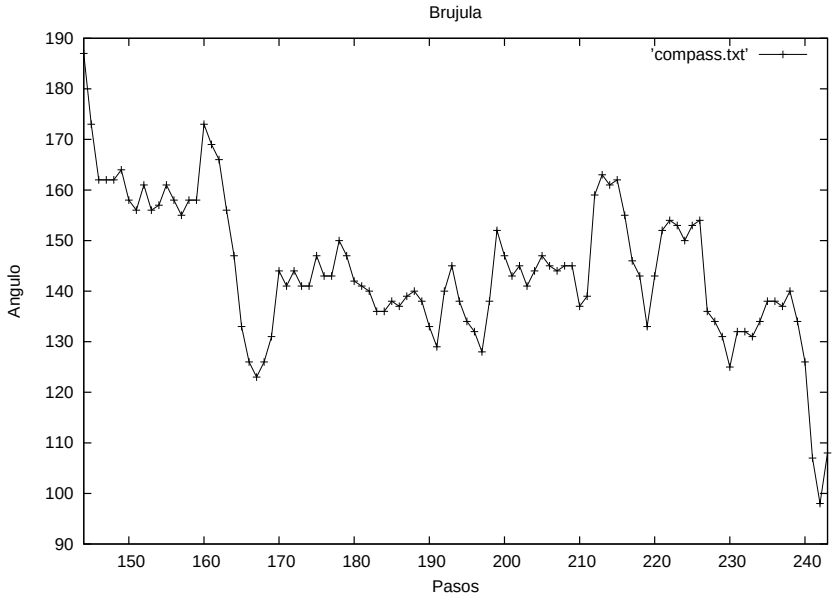
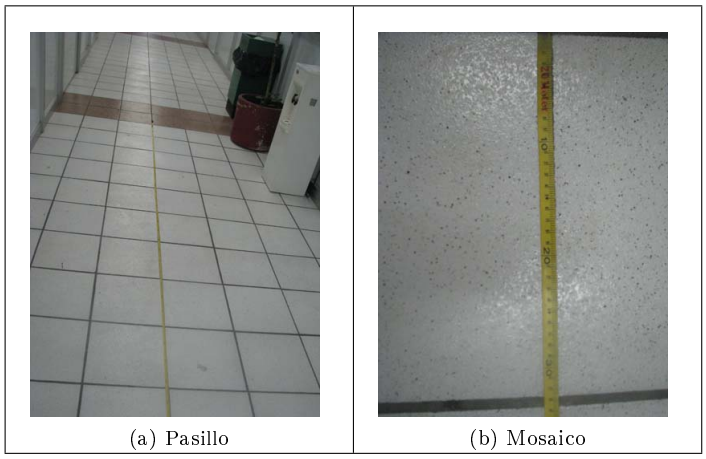


Figura 5.4: Muestras de caminatas a ciegas, con ayuda verbal



(a) Pasillo (b) Mosaico

Figura 5.5: Medidas de los mosaicos de un pasillo

Una vez que se tiene este dato, lo que se prosigue es a analizar la longitud del paso de una persona adulta (fig.: (5.6)). Para realizar esto, lo que se hizo es tomar lecturas de dicha lontigud. Se encontró, como puede ser evidente, que una persona vidente (fig: 5.6a) tiene un paso de mayor lontigud que una persona invidente (fig: 5.6b). Analizando a detalle este comportamiento con distintas personas, se encontró que por cada paso de una persona invidente se avanza en aproximación un mosaico y medio, es decir, que dos pasos es suficiente para avanzar tres mosaicos; sin embargo, para una persona invidente, avanzar un paso le toma apenas la longitud de un mosaico.

Es importante mencionar que estos datos presentaron una ligera variación de persona a persona, pero esta realmente fue mínima, en su mayoría el comportamiento fue similar.

(a) Longitud del paso de una persona vidente (b) Longitud de pasos de una persona invidente

Figura 5.6: Longitud de pasos de una persona vidente e invidente

Por lo tanto, en base al analisis anterior, se encuentra que un umbral de distancia

de proximidad óptimo es de 32 cm o 3.2 dm.

5.3. PROBANDO EL FUNCIONAMIENTO DEL SISTEMA CON USUARIOS

Una vez realizadas las pruebas anteriores se procede a realizar pruebas con los usuarios utilizando sistema completo.

Para esta prueba, se pidió la colaboración de 5 personas videntes que, utilizando vendaje en los ojos, portarían el sistema y ralizarían cierto recorrido.

El objetivo de esta prueba es que con los usuarios utilicen el sistema para realizar un cierto recorrido, el cual será el mismo para todos, y analizar el nivel de aceptación que presentan los usuarios con el sistema.

Básicamente los puntos de interés para esta prueba fueron:

- Eficacia en la localización, sistema de guía y tiempos de recorrido.
- Guía auditiva adecuada, mensajes sonoros cómodos y fáciles de entender por el usuario.
- Comodidad con el sombrero y las cámaras dentro de éste.
- Comodidad e intuitividad con el teclado.
- Interfaz de usuario; esto es, facilidad del usuario para comunicarse con el sistema, realizar peticiones, comprensión de los mensajes de salida e interactividad.
- Localización constante; esto es, que el sistema en todo momento lo tenga localizado con triangulación y el usuario no se pierda.

	Nada Satisfecho	Medianamente Satisfecho	Satisfecho	Total
Sombrero	60 %	40 %		100 %
Guía auditiva		20 %	80 %	100 %
Teclado			100 %	100 %
Interfaz Usuario			100 %	100 %
Localización Constante		40 %	60 %	100 %

Cuadro 5.2: Resultados de pruebas del sistema con usuarios

Analizando los resultados que arrojó esta prueba (Tabla 5.2) se observa que la *Interfaz de Usuario* presenta una aceptación completa por los usuarios, el usuario fácilmente pudo acceder a las opciones del sistema, elegir destino y

empezar el recorrido. Posteriormente, el *Teclado* presentó también una completa aceptación, al ser pequeño intuitivo y, aún cuando ninguna de las personas está familiarizada con el sistema de escritura braille, les fue fácil de utilizar. En cuanto a la *Guía Auditiva*, se detectó una satisfacción en general, la mayoría de los usuarios se sintieron cómodos con los sonidos q los guían a la derecha o izquierda, con la excepción de una persona que comentó que le resultó en diversas ocasiones confuso. En cuanto al *Sombrero*, los resultados fueron menos alentadores, ya que presentó una insatisfacción de la mayoría de los usuarios, quienes lo consideraron muy robusto y poco prático para utilizarlo en la vida cotidiana; los usuarios recomendaron pensar en algún otro diseño más practico ya que este, junto con los cables de las tres cámaras, resulta poco cómodo. En cuanto a la *Localización*, en general se tuvieron buenos resultados; el sistema fue capaz de localizarse en todo momento para llevarlos a su destino, sin embargo, hubo algunas ocasiones en las que el sistema no podía reconocer alguna de las marcas, por lo que provocaba que el cartógrafo se "perdiera", con algún movimiento del usuario, el sistema de visión volvía a reconocer las tres marcas necesarias y el cartógrafo nuevamente estaba localizado; este problema se presentó en diversos casos, sobre todo en aquellos donde las condiciones de luz sufría cambios o en donde el usuario movía o giraba muy rápido su cabeza.

Un punto que llamó la atención fue el tiempo de recorrido; ya que, en general, se notó que el tiempo de éste resultaba ser de hasta un 300 % más lento que si se hiciera sin el sistema; sin embargo, conforme una persona incrementaba las repeticiones o se familiarizaba más con el cartografo, este tiempo iba en decremento.

Capítulo 6

CONCLUSIONES

De acuerdo con las hipótesis planteadas al inicio de este trabajo, se pudo demostrar experimentalmente que, en efecto, es posible desarrollar un sistema basado en técnicas de localización y navegación de robots; sin embargo, es importante resaltar que una persona jamás será un robot ni se comportará como tal; por lo que es necesario emplear estas herramientas a un nivel más elevado, donde las exactitudes métricas son imposibles, es decir, es imposible ordenarle a un ser humano girar 1.5 radianes y avanzar 3 metros en línea recta.

Fue a su vez gratificante demostrar que no es indispensable contar con grandes sistemas de hardware o software de precio elevado, ha sido suficiente contar con tres cámaras web las cuales cada vez se encuentran de una gran variedad de marcas a precios competitivos; contar con un teclado braille como el utilizado en este trabajo y el cual, como se explica en el apéndice A, resulta también bastante económico.

Utilizar sistemas de Realidad Aumentada (RA), más en concreto ARToolKit Plus, ha resultado, desde cierto punto, satisfactorio para el prototipo desarrollado; sin embargo, como se ha mencionado en el capítulo de pruebas, este sistema presenta muchas limitaciones para el uso que se requiere en este proyecto. Esto es debido a que, en su mayoría, los sistemas de RA están diseñados para funcionar en ambientes donde el patrón se encuentra relativamente cerca a la cámara y ningún otro objeto se interpondrá en la imagen haciendo el patrón invisible.

Este problema presentado con ARToolKit Plus era ya, desde los inicios del proyecto, intuitivo; se sabía de la posibilidad de que una de las principales limitantes que se encontrarían en el sistema sería precisamente el sistema de visión; no sólo por el funcionamiento de ARToolKit Plus y para lo que fue diseñado originalmente, sino por la complejidad y limitantes que aun presentan hoy día los diversos sistemas de visión.

Un excelente sistema de visión que iría acorde con este trabajo, sería aquel que sea capaz de reconocer marcas naturales; esto es, no utilizar patrones impresos

como los que utiliza ARToolKit Plus, sino en base a elementos naturales existentes en el ambiente, se tengan patrones para su reconocimiento y cálculo de distancias. Sin embargo, tópicos como éste podrían ser por sí mismos temas de investigación para doctorado. El Maestro en Ingenieria José Israel Figueroa Angulo, egresado del Posgrado en Ciencia e Ingeniería de la Computación de la UNAM, planteó como proyecto de tesis ([Ang09]) un sistema de RA basado en marcas naturales el cual presenta ventajas con respecto a ARToolKit Plus con el simple hecho de no utilizar patrones artificiales; sin embargo, presenta el mismo problema de los demás sistemas de RA en cuanto al límite en el rango de distancia para el reconocimiento.

Con base en lo anterior, es que se decidió diseñar este sistema de manera tal que fuese independiente del sistema de visión. Esto es, que ante la limitante que presenta ARToolKit Plus, éste puede ser fácilmente reemplazado por cualquier otro sistema de visión o incluso cualquier otro sistema de sensores que sea capaz de calcular las distancias entre la persona y tres puntos definidos, sólo basta con seguir el protocolo de comunicación que tiene el cartógrafo.

De la misma manera ocurre con la brújula, la cual resulta ser el componente más caro dentro del sistema, que apesar de tener una interfaz inalámbrica por medio de bluetooth, esto para la comunicación del cartógrafo resulta transparente puesto que ambos se comunican por medio de un puerto COM serial; por lo que se relega al sistema operativo las operaciones para hacer posible la comunicación vía bluetooth. Esto significa, que bastaría con tener otra brújula capaz de comunicarse con una interfaz serial y que reciba los mismos comandos (los cuales se muestran en el apéndice B) para que funcione de la misma manera.

Debido a las limitantes que se han venido mencionando y lo robusto del prototipo, se decidió que para esta etapa aun no se harían pruebas con personas invidentes. Esto se debe a que las personas con discapacidades diferentes pueden llegar a ser muy sensibles en distintos aspectos; es por esto que se decidió no acudir a la ayuda de estas personas hasta primero probar con personas videntes y analizar el comportamiento de estas, pero sobre todo, hasta tener un mejor sistema de visión que el utilizado en este trabajo. Una vez mejorando esto último (sistema de visión) y teniendo un mejor diseño en la colocación de las cámaras (sombrero), puede ser posible empezar a hacer pruebas con personas invidentes para analizar el comportamiento y el grado de aceptación que presenta el sistema con ellos y, por último, realizar los ajustes que se consideren pertinentes.

Bibliografía

[Ang09] José Israel Figuera Angulo, *Desarrollo de un sistema de realidad aumentada basado en la colocación de objetos virtuales en escenas reales a partir de la obtención de nubes de puntos en una secuencia de imágenes*, Master's thesis, 2009.

[Bær03] J. A. Bærentzen, *On left-balancing binary trees*, aug 2003.

[Ben75] Jon Louis Bentley, *Multidimensional binary search trees used for associative searching*, Commun. ACM **18** (1975), no. 9, 509–517.

[BR05] Oliver Bimber and Ramesh Raskar, *Spatial augmented reality: Merging real and virtual worlds*, A K Peters, Ltd., July 2005.

[BRA08] ASOCIACIÓN CIVIL CONTACTO BRAILLE, *Discapacidad visual con dignidad*, http://www.contactobraille.com/baston.html, 2008.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, 2nd ed., The MIT Press, 2001.

[dPPR08] María del Pilar Pinzón Rueda, *Tecnología en la discapacidad visual*, 2008, "http://www.ladiscapacidad.com/tecnologiaydiscapacidad/0000009a8b05b220b/index.html".

[HiT09] HiTechnic, *Hitechnic products*, http://www.hitechnic.com, 2009.

[I.A08] Discapacitados Visuales I.A.P, *Perros guía*, http://www.perrosguia.org.mx/comoseentrena.html, 2008.

[Int99] Intel, *Opencv*, 1999, "http://sourceforge.net/projects/opencvlibrary/".

[KB99] H. Kato and M. Billinghurst, *Marker tracking and hmd calibration for a video-based augmented reality conferencing system*, Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on, 1999, pp. 85–94.

[Lam03] Phillip Lamb, *Camera calibration*, 2003, "http://www.hitl.washington.edu/artoolkit/documentation/usercalibration.htm".

[LEG08] LEGO, *Mindstorm nxt*, mindstorms.lego.com, 2008.

[Nuñ07] María Angeles Nuñez, *La deficiencia visual*, 2007.

[SSL04] Velázquez Chávez Isaac Sanabra Serna Luis, Díaz Méndez Luis, *Sistema de mensajería braille del tecnológico*, 2004.

[Wal00] W. Wallis, *A beginner's guide to graph theory*, Springer, July 2000.

[WHO09a] World Health Organization WHO, *Magnitude of blindness and visual impairment*, http://www.who.int/blindness/causes/magnitude/en/index.html, 2009.

[WHO09b] ———, *Visual impairment and blindness*, 2009, http://www.who.int/mediacentre/factsheets/fs282/en.

[WHO09c] WHO World Health Organization, *Causes of blindness*, http://www.who.int/blindness/causes/en/index.html, 2009.

[Wik09] Wikipedia, *Lego mindstorms — wikipedia, the free encyclopedia*, 2009, [Online; accessed 19-April-2009].

[WS07] Daniel Wagner and Dieter Schmalstieg, *Artoolkitplus for pose tracking on mobile devices*, February 2007.

Apéndice A

TECLADO Y CÓDIGO BRAILLE

Código Braille.

Los caracteres Braille se forman a partir de la denominada "celda Braille" (fig: A.1), la cual consiste en una matriz de 6 puntos como se muestra en la imagen inferior. A cada uno de estos puntos se asocia un número de 1 a 6 y, dependiendo de cuáles puntos se pongan de relieve, tenemos un carácter distinto, para un total de 64, incluyendo el carácter "blanco", donde no se realza ningún punto, y el que tiene todos los puntos en relieve.

En la figura A.2 se puede apreciar las distancias aproximadas entre puntos de una celda y entre celdas Braille. La altura de estos puntos, aproximadamente 0,5 mm, le confiere el relieve característico a los caracteres Braille.

También pueden encontrarse versiones en un tamaño mayor, especialmente pensado en personas invidentes que tienen problemas para percepción por el tacto, así como para quienes se están iniciando en la lectura Braille.

A continuación se muestra varios grupos de tablas con carácteres braille que se apegan de acuerdo al documento oficial “Código matemático unificado para la lengua castellana”, aprobado por la Reunión de Imprentas Braille de Habla

Figura A.1: Celda Braille[SSL04]

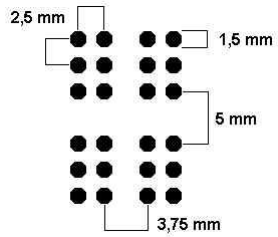


Figura A.2: Medidas de relieves en celdas braille[SSL04]

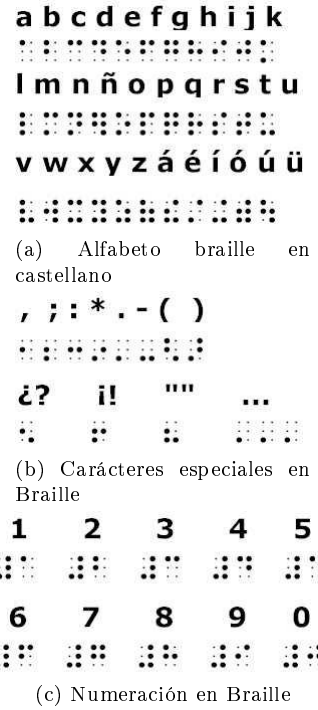
Hispana, en Montevideo, Uruguay, en 1987, en la primera edición de 1988, por Enrique Elissalde.

Los carácteres aquí reseñados corresponden al tipo de letra por defecto, minúscula latina normal (fig: A.3a). Estos caracteres, excepto la ñ, á, é, í, ó, ú y ü, coinciden con los asociados a las letras en los otros idiomas, como el inglés y francés, este último, idioma del país de origen de Louis Braille, inventor del código. Aquí vemos los signos ortográficos más comunes, nótese que los signos de interrogación, admiración y comillas se transcriben igual, sean de apertura o cierre (fig: A.3b). Los dígitos se forman anteponiendo el símbolo de número (3456) a las primeras diez letras del alfabeto (fig: A.3c).

Descripción del Teclado Braille

Para obtener el teclado braille, el cual será requerido para la interacción del usuario con el sistema, partirá de un teclado numérico de computadora (fig: A.4) al cual se le aplicarán relieves que vayan acorde con los estándares braille; de tal manera que pueda ser fácil de identificar y controlar por el usuario, para lograr el máximo comfort y mínimo de errores, se realizó un análisis de eficiencia haciendo pruebas de uso directamente por personas con discapacidades visuales, la propuesta de distribución de teclas que se presenta es totalmente nueva, no tiene antecedentes y es totalmente diferente a la de los dispositivos braille del mercado actual, proporcionando una considerable disminución del costo y un alto grado de desempeño.

Este tipo de teclados son destinados generalmente para computadoras portátiles y no son muy costosos como los teclados especializados braille, resulta relativamente sencillo de hacer una adaptación hardware-software.



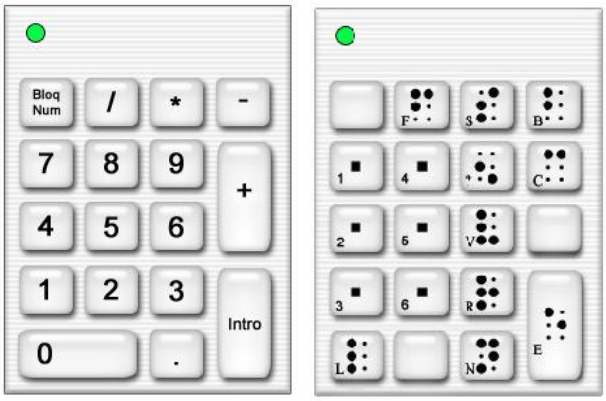
(a) Alfabeto braille en castellano

(b) Carácteres especiales en Braille

(c) Numeración en Braille

Figura A.3: Alfabeto Braille en castellano [SSL04]

Figura A.4: Teclado Numérico[SSL04]



(a) Teclado convencional (b) Teclado con relieves

Figura A.5: Distribución del Teclado[SSL04]

Distribución del Teclado Braille

El Teclado Braille consta de 16 teclas con relieve, seis de las cuales representan la celda braille, constituida por una matriz de tres renglones dos columnas, numeradas ascendentemente en forma vertical; las 15 teclas restantes estarán destinadas a funciones específicas del sistema.

La figura A.5 muestra el esquema de la distribución del teclado acorde a lo arriba mencionado.

Funcionamiento del Teclado y Codificación

Como puede observarse, el teclado numérico está limitado a una pequeña cantidad de teclas, por lo tanto, a fin de poder cubrir el extenso alfabeto castellano, números y símbolos gramaticales; recurriremos a la combinación de teclas que nos ayudarán a lograr nuestro objetivo, teniendo siempre en cuenta el estándar del código braille (fig: A.6).

El usuario a partir de las 6 teclas que tiene, las cuales simulan la celda braille, realizará las pulsaciones de cada una de ellas, acorde al carácter que desea capturar, siguiendo obviamente la secuencia de numeración de una celda braille; una vez terminada la combinación de teclas, presionará una tecla de función que le indique al sistema que el usuario ha terminado de realizar las combinaciones; en este momento, el sistema procesa la combinación de teclas y realiza la codificación al carácter correspondiente de acuerdo al código ASCII; para capturar otro

carácter, basta con repetir las mismas operaciones hasta obtener la o las palabras deseadas.

A continuación se plasma la combinación de teclas numéricas las cuales darán origen a los caracteres del alfabeto.

Letras del Alfabeto

Letra	Combinaciones	Letra	Combinaciones	Letra	Combinaciones
A	7	N	7185	Z	7152
B	74	Ñ	74852	Á	74152
C	78	O	715	É	4182
D	785	P	7418	Í	18
E	75	Q	74185	Ó	182
F	748	R	7415	Ú	41852
G	7485	S	418		
H	745	T	4185		
I	48	U	712		
J	485	V	7412		
K	71	W	4852		
L	741	X	7182		
M	718	Y	71852		

(a) Combinación de teclas para letras del alfabeto

Dígitos Numéricos.

Letra	Combinaciones	Letra	Combinaciones
1	7	6	748
2	74	7	7485
3	78	8	745
4	785	9	48
5	75	0	485

(b) Combinación de teclas para dígitos numéricos

Símbolos Gramaticales

Letra	Combinaciones	Letra	Combinaciones
,	4	(	742
;	41	)	185
:	45	¿?	42
*	15	¡!	415
.	1	"	412
-	12		

(c) Combinación de teclas para símbolos gramaticales

Figura A.6: Combinación de teclas para obtener distintos caracteres en ASCII [SSL04]

Apéndice B

CÓDIGO DE PROGRAMACIÓN DE LA BRÚJULA

A continuación se muestra el código de las clases que hacen posible la programación de la comunicación con la brújula del NXT. La descripción de cada una de estas clases se explica a detalle en la sección 4.1.2 en la página 47.

Clase Serial

```
#ifndef SERIAL
#define SERIAL
#include <windows.h>
#include <iostream>
#include <string>
#define MAX_NR_BYTES 64

using namespace std;

class Serial{
private:
        HANDLE handle;
    BYTE byte[MAX_NR_BYTES];
   DWORD nrBytes;
   COMMTIMEOUTS timeout;
    DCB dcb;
public:
    Serial();
    char * receive(int length);
    int send(char *command, int bytes);
    int connect(char *port);
    int connect(char *port, int set_timeout, int bytesize, int baudrate, int parity);
    int disconnect();
    int flush();
};
#endif
```

```
#include <iostream>
#include <windows.h>
#include "serial.hpp"
#include <stdio.h>
#include <time.h>
#include <string>
#include <stdlib.h>
```

```
using namespace std;
    Serial::Serial(){           }
         int Serial::send(char *command, int bytes){
         int i=0;
         if (handle == INVALID_HANDLE_VALUE){
             return 0;
         }
         while(i<bytes){
             byte[i]=command[i];
             i++;
         }
         if(WriteFile(handle, byte, bytes, &nrBytes, NULL)!=0){
             return 1;
          }
         else
             return 0;
   }

  char *Serial::receive(int length){
         static char answer[MAX_NR_BYTES];
         int i=0;
         nrBytes=0;
         answer[0]='\0';
         if (ReadFile(handle, byte, length, &nrBytes, NULL) == 0){
             answer[0]='e';
             answer[1]='r';
             answer[2]='r';
             answer[3]='o';
             answer[4]='r';
             answer[5]='\0';
             return &answer[0];
         }
         while(i<nrBytes){
             answer[i]=byte[i];
              i++;
         }
         answer[i]='\0';
         return &answer[0];
    }
   int Serial::flush(){
         int i=0;
         clock_t endwait;
         while(nrBytes > 0){
         if(ReadFile(handle, byte, 1, &nrBytes, NULL)==0){
                   return 0;
               }
               i++;
         }
         return i;
    }

 int Serial::connect(char *port){
         return connect(port, 200, 8, 19200, 0);
         }
 int Serial::connect(char *port, int set_timeout, int bytesize,
                           int baudrate, int parity){
     handle = CreateFile(port, GENERIC_READ | GENERIC_WRITE,
                0,0,OPEN_EXISTING,0,0);
         if (handle == INVALID_HANDLE_VALUE){
             return 0;
         }
         dcb.ByteSize = bytesize;/*sets bit-size*/
         dcb.BaudRate = baudrate;/*sets baudrate*/
         /*Sets timeout for reading - 0 = deactivate*/
         timeout.ReadIntervalTimeout=set_timeout;
         timeout.ReadTotalTimeoutConstant=set_timeout;
         dcb.Parity=parity;
         timeout.ReadTotalTimeoutMultiplier=set_timeout;
         SetCommTimeouts(handle, &timeout);
        SetCommState(handle, &dcb);
         return 1;
    }

int Serial::disconnect(){
          if (handle){
            CloseHandle(handle);
            handle = 0;
            return 1;
          }
          else{
             return 0;
          }
    }
```

Clase Sensor

```
#ifndef SENSOR
#define SENSOR
#include <iostream>
#include <string>
#include "serial.hpp"
using namespace std;
//sensor types
enum {NO_SENSOR=0x00, TOUCH=0x01, TEMPERATURE=0x02, REFLECTION=0x03,
      ANGLE=0x04, LIGHT_ACTIVE=0x05, LIGHT_INACTIVE=0x06,
        SOUND_DB=0x07, SOUND_DBA=0x08, CUSTOM=0x09, LOWSPEED=0x0A, LOWSPEED_9v=0x0B,
          NO_OF_SENSOR_TYPES=0x0C};
//sensor moedes
enum {RAW_MODE=0x00, BOOL_MODE=0x20, TRANSITION_MODE=0x40, PERIOD_MODE=0x60,
       PERCENT_MODE=0x80, CELSIUS_MODE=0xA0,
         FAHRENHEIT_MODE=0xC0, ANGLE_MODE=0xE0, SLOPE=0x1F, MODEMASK=0xE0};
class Sensor{
private:
    int port;
    Serial *comport;
    int get_sensor_value();
         int raw;
    int adc;
    int mode;
    int calcu;
           public:
    Sensor();
    Sensor(int input_port, Serial *cp);
    int type_and_mode(int type, int mode);
    int read();
    int read_raw();
    int read_adc();
    int reset();
};
#endif
```

```
#include <iostream>
#include "sensor.hpp"
#include <stdio.h>
#include <time.h>
#include <string>
#include <stdlib.h>
using namespace std;
         Sensor::Sensor(){      }
    Sensor::Sensor(int input_port, Serial *cp){
        port=input_port;
        comport=cp;
    }
    int Sensor::type_and_mode(int type, int mode){
         char command[7];
         command[0]=0x05;  //command length
         command[1]=0x00;  //start of message
         command[2]=0x80;
         command[3]=0x05;
         command[4]=port;
         command[5]=type;//input port
         command[6]=mode;//set mode
         return comport->send(&command[0],7);
        }

        int Sensor::get_sensor_value(){
        char command[5];
        char *answer;
        command[0]=0x03;
        command[1]=0x00;
                  command[2]=0x00;
        command[3]=0x07;
        command[4]=port;
        comport->send(&command[0],5);
            answer=comport->receive(18);
        raw=((0xff & answer[10]) | (answer[11] << 8));
         adc=((0xff & answer[12]) | (answer[13] << 8));
         mode=((0xff & answer[14]) | (answer[15] << 8));
         calcu=((0xff & answer[16]) | (answer[17] << 8));
         return 1;
    }

   int Sensor::reset(){
         char command[5];
         command[0]=0x03;  //command length
         command[1]=0x00;
         //start of message
         command[2]=0x80;
         command[3]=0x08;
         command[4]=port;
         return comport->send(&command[0],5);
         }

   int Sensor::read(){
```

```
        get_sensor_value();
        return mode;
      }

   int Sensor::read_raw(){
        get_sensor_value();
        return raw;
     }
     int Sensor::read_adc(){
        get_sensor_value();
       return raw;
        }
#endif
```

Clase I2C

```
#ifndef I2C_SENSOR
#define I2C_SENSOR
#include <iostream>
#include <string>
#include "serial.hpp"
#include "sensor.hpp"
using namespace std;
class I2C{
private:
    int port;
    Serial *comport;
    int sensor_status;
    int nr_bytes;
    int  status();
    enum {COM_PENDING=0x20, CHANNEL_BUSY=0xE0};
public:
    I2C();
    I2C(int input_port, Serial *cp);
    int sensor_type(int type);
    int  write(char *tx, int tx_length, int rx_length);
    char *read();
    int wait_ans(int num_bytes);
 };
#endif
```

```
#include <iostream>
#include "i2c.hpp"
#include <stdio.h>
#include <time.h>
#include <string>
#include <stdlib.h>
using namespace std;
    I2C::I2C(){     }
    I2C::I2C(int input_port, Serial *cp){
        port=input_port;
        comport=cp;
        sensor_status=1;
        nr_bytes=0;
         }
        int I2C::sensor_type(int type){
         char command[7];
         command[0]=0x05;
 //command length
          command[1]=0x00;
          //start of message
          command[2]=0x80;
          command[3]=0x05;
          command[4]=port;
          command[5]=type;
          command[6]=RAW_MODE;//set mode
          return comport->send(&command[0],7);
        }

   int I2C::status(){
        char command[5];
        char *answer;
        command[0]=0x03;
        command[1]=0x00;
        command[2]=0x00;
        command[3]=0x0E;
        command[4]=port;
        comport->send(&command[0],5);
        answer=comport->receive(6);
        sensor_status=answer[4];
        nr_bytes=answer[5];
        return 1;
```

```
        }

    int I2C::write(char *tx, int tx_length, int rx_length){
        int i=0;
        char command[24];
        while(i < tx_length){
            command[i+7]=tx[i];
            i++;
        }
       command[0]=i+5;
     command[1]=0x00;
                 command[2]=0x80;
        command[3]=0x0F;
        command[4]=port;
        command[5]=i;
        command[6]=rx_length;
        return comport->send(&command[0],i+7);
    }

  int I2C::wait_ans(int num_bytes){
      int bytes_ready=0;
      int status=0;
      int attempts=0;
      do{
            I2C::status();
            attempts++;
            bytes_ready=nr_bytes;
            if(sensor_status < 0 || attempts > 10000){//maybe tune this
                return 0;
            }
       }while(bytes_ready < num_bytes);
       return 1;
    }

     char * I2C::read(){
        char command[5];
        char *answer;
        command[0]=0x03;
        command[1]=0x00;
        command[2]=0x00;
        command[3]=0x10;
        command[4]=port;
        comport->send(&command[0],5);
        answer=comport->receive(24);
        return &answer[5]; //first 5 bytes discarded
       }
#endif
```

Clase Compass

```
#ifndef COMPASS
#define COMPASS
#include <iostream>
#include <string>
#include "serial.hpp"
#include "I2C.hpp"
using namespace std;
enum {COMPASS_ADDRESS=0x02, COMMAND_REG=0x41, DEGREE=0x42, DEGREE_HALF=0x43};
namespace BluetoothCompass {
class Compass{
private:
    I2C sensor;
     public:
    Compass();
    Compass(int input_port, Serial *cp);
    int setup();
    int degree();
    int degreeInverse(); };
}
using namespace BluetoothCompass;
#endif
```

```
#include <iostream>
#include "compassnxt.hpp"
#include <stdio.h>
#include <time.h>
#include <string>
#include <stdlib.h>
using namespace std;
        Compass::Compass(){      }
    Compass::Compass(int input_port, Serial *cp){
        sensor = I2C(input_port,cp);
```

```
}
int Compass::setup(){
    sensor.sensor_type(LOWSPEED);
    Sleep(500);
    degree();
    degree();
    degree();//Otherwise the first measurement fails
    return 1;
}
int Compass::degree(){
    char *answer;
    char command[2];
    command[0]= COMPASS_ADDRESS;
    command[1]= DEGREE;
    sensor.write(&command[0], 2, 2);
    if(sensor.wait_ans(2)){
        answer=sensor.read();
        return (0xff & answer[1])*2 + answer[2]; //returns unsigned value
    }
   return -1;
}
int Compass::degreeInverse(){
    int deg = degree();
    return (deg==-1)?deg:360-deg;
}
```