CONSTRUCCIÓN DE MAPAS TRIDIMENSIONALES MEDIANTE UN SENSOR LÁSER ACTIVO

Víctor Prieto Marañón

Facultad de Informática.
Universidad de Las Palmas de G.C.

Facultad de Informática. Universidad de Las Palmas de G.C.

Proyecto fin de carrera

Título: CONSTRUCCIÓN DE MAPAS TRIDIMENSIONALES MEDIANTE UN SENSOR LÁSER ACTIVO
Apellidos y nombre del alumno: Prieto Marañón, Víctor
Fecha : Mes y año en que se presentó la memoria del PFC

Tutor: Cabrera Gámez, Jorge
Cotutor: Domínguez Brito, Antonio Carlos

Facultad de Informática. Universidad de Las Palmas de G.C.

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

Aqui ponemos los agradecimientos

Capítulo 1

Introducción

Los sensores láser de rango se han venido empleando de forma rutinaria en robótica desde hace años en la elaboración de mapas de entorno para la navegación y la evitación de obstáculos. Tradicionalmente, estos sensores, debido a su tamaño y peso, se han instalado en los robots móviles en una posición fija, de manera que el plano barrido por el sensor quedaba fijado en un cierto ángulo respecto a un plano de referencia ligado al robot [2]. Este tipo de instalación hace que estos sensores sólo puedan percibir su entorno en un único plano, lo que puede provocar que el mapa de entorno que se obtiene no incluya detalles importantes. Así, un láser paralelo al suelo, instalado en un robot, no puede detectar ningún obstáculo que esté situado por debajo o por encima de la altura a la que se sitúa el plano de barrido. No podría detectar, por ejemplo, ni el inicio de unas escaleras descendentes ni evitar colisionar con el tablero de una mesa.

Dotar a un sistema robótico con una representación tridimensional de su entorno haría posible el que pudiera desenvolverse en entornos complejos y reales. Para alcanzar esta meta la representación debería ser capaz de manejar la realidad dinámica del mundo. Es reseñable el interés que la comunidad está mostrando en los últimos años por la construcción de tales mapas tridimensionales. Se presentan anualmente distintos artículos así como se dedican distintos *workshops* a este área en revistas y en conferencias como la *IEEE International Conference on Intelligent Robots and Systems* (IROS) y la *IEEE International Conference on Robotics and Automation* (ICRA). Como ejemplo, el *Journal of Fields Robotics* dedica en el año 2009 un número especial a los mapas tridimensionales. Citamos de su propia web [1]:

> Recientemente, representaciones tridimensionales del entrono han ganado un interés substancial entre la comunidad de robotistas debido a que dichos mapas proporcionan un mejor soporte a una gran variedad de tareas entre las que se incluye la navegación, la localización y la percepción. Por ejemplo, los robots que conocen la estructura tridimensional del entorno pueden evitar obstáculos de una forma mejor, pueden localizarse a si mismos con más fiabilidad y pueden detectar objetos de forma más robusta. En consonancia, las representaciones tridimensionales proporcionan beneficios en todas las aplicaciones en las cuales los robots son empleados en escenarios reales.

Dotar al robot de un sistema de adquisición, actualización y procesado de mapas tridimensio-

nales no es un problema trivial y hay que resolver numerosas cuestiones: qué dispositivos hardware usar, cómo integrar los dispositivos entre sí y a su vez con el sistema de procesado, elegir un modelo de representación tridimensional interno versátil, controlar los consumos de memoria y procesador de forma que el sistema sea implementable en un robot real, actualización del modelo en tiempo real o diferido, extraer información de alto nivel semántico de la representación, etc.

Presentaremos en los siguientes capítulos con detenimiento las distintas soluciones que en este proyecto se han dado a los diversos retos expuestos en el párrafo anterior. Veremos como se ha optado por montar un sensor láser de peso ligero sobre un dispositivo apuntador con dos grados de libertad, teniendo que resolverse diversos problemas de sincronización. Se detallará el tipo de representación que se ha elaborado, un tipo de mapa de superficies multinivel. Por último, se abordará la cuestión de cómo realizar abstracciones sobre la representación de forma que se puedan detectar y reconocer estructuras y objetos.

1.1. Estado actual

Repasaremos en esta sección cuáles son las tendencias actuales en el uso de dispositivos para la adquisición de mapas 3D, qué técnicas se vienen empleando en la construcción de tales mapas y cómo puede se sirve un sistema robótico dotado de mapas 3D para resolver el problema de localización y de construcción simultánea de mapas.

Existen, principalmente, tres modos distintos de percivir imágenes tridimensionales. Una forma de estimar la profundidad es mediante triangulación de puntos correspondientes respecto a una línea base. Otra forma es estimar la profundidad en base al principio del «tiempo de vuelo» Un tercer método realiza la estimación a partir de una simple imagen haciendo uso de algún conocimiento del fondo de la misma. Este último método no es usado actualmente en robótica [2].

La medición del «tiempo de vuelo» se basa en medir el lapso de tiempo transcurrido entre la emisión de una señal y su retorno. La distancia recorrida, s, se calcula en función del tiempo medido, t, y de la velocidad v de la señal como:

$$s = v \cdot t$$

Normalmente, el receptor y el emisor son un mismo dispositivo por lo que la distancia del emisor al objeto se estima como

$$D = v\frac{t}{2}$$

Otra forma de estimar la disntancia a partir de una onda contínua, por ejemplo, una onda contínua de luz láser. La amplitud de la portadora es modulada por señales sinusoidales de diferentes frecuencias, obteniéndose así la llamda señal de medición. Se obtiene finalmente el desplazamiento de fase entre la señal emitida y la señal recibida. Este desplazamiento es proporcional

a la distancia D. Siendo $\Delta\varphi$ la diferencia de fase, λ lalongitud de onda de la señal modulada y f la frecuencia correspondiente, la distancia D se obtiene como:

$$D = \frac{\Delta\varphi}{4\pi}] = \frac{\Delta\varphi v}{4\pi f}$$

Como señalamos anteriormente, la distancia también puede ser medida mediante triangulación. El emisor emite una señal y el receptor, que ahora se encuentra a una distancia L del emisor, detecta la señal con una cierta disviación x usando para ello una cámara «pinhole». La distancia se obtiene como:

$$D = f\frac{L}{x}$$

Los sensores láser miden distancias usando como señal un haz láser. Los métodos de escaneo se usan para digitalizar superficies, lo que implica mecanismos para mover o rotar el sistema de medición. Se usan alternativamente, espejos giratorios para cambiar la dirección del rayo. La distancia al objeto medido se determina por cualquiera de los métodos expuestos anteriormente: «tiempo de vuelo», desplazamiento de fase o por triangulación.

Los sensores láser se pueden clasificar en función de la región del espación que son capaces de adquirir sin necesidad de desplazarlos. Los sistemas de medición láser 1D consisten en un sensor que hace uso de un rayo láser para medir distancias. La medida es transmitida directamente a un sistema de compuación que calcula la distancia. Los escáner laser 2D son una extensión de los sistemas 1D. Los escáner 2D, para obtener una línea de datos, rotan bien el láser, bien un espejor, de forma que el rayo láser mide distancias en diferentes direcciones. Los escáner 3D, a su vez, son una extensión de los 2D a los que añaden algún grado de libertad. Estos sistemas o bien añaden un segundo espejo o bien rotan tdo el escáner 2D. Además, existen escáner de proyección 3D. Estos escáner usan el principio de triangulación.

En adición a los escáner láser, también se usa en la adquisición de imágenes 3D cámaras. Estos dispositivos adquieren puntos luminosos usando habitualmente sensores CCD o CMOS. La tecnología CMOS es más compleja siendo, sin embargo, los costes de producción más bajos. Las webcams usan normalmente sensores CMOS, mientras que las cámaras digitales usan un chip CCD. Las sistema estéreo permiten estimar la distancia de un objeto usando dos cámaras que distan una distancia fija. La fusión de ambas imágenes permiten computar la distancia en función de la diferencia entre puntos correspondientes en ambas cámaras. La disparidad en la localización de estos puntos proporciona, por triangulación, la distancia a la que se encuentran. Las cámaras estéreo tienen que lidiar con el problema de correspondencia y el de calibración de la cámara. El primer problema trata de encontrar, dado un punto en una imagen, cuál es el correspondiente en la otra imagen. La calibración de la cámara trata de determinar los parámetros extrínsecos de ambas cámaras. Esto permite obtener la rotación y traslación que existe entre las cámaras. Mediante esos parámetros se pueden rectificar las imágenes y se simplifica el problema de correspondencia.

Por último, una tendencia actual en el empleo de cámaras es el desarrollo de mámaras 3D. Estas cámaras contienen un sensor CMOS y diodos láser que iluminan la escena con luz modulada. El sensor CMOS detecta la luz y calcula el «tiempo de vuelo» basándose en el desplazamiento de fase del rayo láser.

Una aproximación popular de afrontar el problema de crear mapas a partir de robots móviles en entornos abiertos han sido los mapas de elevación [3]. Estos mapas aplican una representación en $2\ \frac{1}{2}$ dimensiones. Un mapa de elevación consta de una malla de dos dimensiones en las que cada celda almacena la altura del terreno. Mientras que este acercamiento conlleva una reducción de memoria sustancial, puede ser problemática cuando un robot tenga que utilizar dichos mapas para navegar o cuando tenga que integrar dos mapas diferentes.

Una mejora que se ha propuesto a la solución anterior, y que se va a usar en este proyecto, es la propuesta en [4]. Básicamente esta estrategia realiza una extensión de los mapas de elevación hacia múltiples superficies. Estos mapas multisuperficie ofrecen la posibilidad de modelar el entorno con más de un nivel transitable. La ventaja de esta solución es que se puede almacenar de forma compacta y, al mismo tiempo, usarse para caracterizar el entorno y dar soporte al problema de asociación durante el alineamiento de distintos mapas.

Conjuntamente a la elaboración del mapa, surge también el problema de ser capaces de extraer información de mayor nivel semántico de la representación del mundo elegida. Mientras que la visualización de nubes de puntos muy detalladas y complejas es actualmente posible, las capacidades de interacción en un cierto nivel semántico son aún muy limitadas. Incluso tareas tan básicas como seleccionar todas las ventanas en un scan de una casa requiere una cantidad desproporcionada de interacción con el usuario. Esto es debido al hecho de que los datos adquiridos durante el scan no proporcionan ninguna información estructural, o en todo caso muy débil, ni semántica.

Sin embargo, el trabajar con nubes de puntos tiene una doble motivación. Por una parte, es la forma natural en la que se adquiere la información por parte de sensores láser o similares. Además, las nubes de puntos son representaciones geométricas extremadamente generales debido a que no contienen representaciones de los datos.

Se ha propuesto un gran número de métodos para la detección de formas primitivas. En la visión por ordenador, las dos metodologías mejor conocidas son el paradigma RANSAC [5] y la transformada de Hough [6]. Ambas han probado con éxito la detección de formas en 2D al igual que en 3D, siendo fiables incluso en presencia de un alto número de puntos ajenos, pero carecen de eficiencia debido al gran consumo de memoria. Se han propuesto distintas técnicas de aceleración. En este proyecto se aplicará el método Efficient RANSAC [7].

1.2. Objetivos

Este proyecto tiene como objetivo principal el estudio y desarrollo de un marco de representación óptimo para mapas de rango tridimensionales obtenidos mediante un sensor láser activo. Para ello

se propone la consecución de una cierta cadena de subobjetivos:

1. Control del sensor láser: adquisición de medidas, control de tiempos, etc.
2. Control preciso del sensor y del sistema de apuntado y su sincronización con el sensor láser. Este aspecto es importante, pues permitirá realizar un barrido tridimensional con el láser de forma continua.
3. Desarrollo de una representación 3D del entorno que sea versátil (precisa, fácil de actualizar con nuevas medidas), pero cuyo consumo de memoria resulte manejable en tiempo real por un equipo que disponga de 1 GB de RAM.
4. Se desarrollarán herramientas de visualización del entorno a partir del mapa disponible en cada momento.
5. Será posible realizar ciertas operaciones básicas sobre la representación 3D del entorno como detección de obstáculos (positivos y negativos), zonas transitables (suelo), zonas no exploradas, planos (paredes, suelos, techos, ...), proyecciones en planos y secciones (cortes)
6. A partir del mapa 3D se podrá sintetizar un mapa 2D de zonas transitables sobre la que se pueda desarrollar la navegación del robot a nivel de evitación de obstáculos y de planificación de caminos.

1.3. Contenido del documento

El presente proyecto se divide en los siguientes capítulos:

Capítulo I: Introducción El presente capítulo presente el proyecto realizado, el estado actual del tema abordado y la estructuración de esta memoria.

Capítulo II: Metodología, recursos y plan de trabajo Se exponen la metodología seguida en el desarrollo del este proyecto, los recursos que han sido necesarios para llevarlo a término y qué plan de trabajo se siguió durante el mismo.

Capítulo III: Estudio de los dispositivos hardware y su integración Se realiza un análisis de los dispositivos hardware empleados en este proyecto, sus características e interfaz de comandos. Se trata también los problemas de integración y de conexión con el ordenador. Por último, se expone la forma de convertir la información proporcionada por los dispositivos en coordenandas del mundo.

Capítulo IV: Estudio de los marcos de representación 3D Se expondrán en este capítulo los principales métodos de creación de mapas 3D. Se repasarán los mapas de rejilla de ocupación, una extensión eficiente de los mapas de elevación y los mapas de superficie multinivel. Al final del capítulo, se expondrá el método de extracción de formas en nubes de puntos «Ransac eficiente».

Capítulo V: Diseño El capítulo está dedicado a presentar la arquitectura del sistema y las clases que se han diseñado.

Capítulo VI: Implementación Se recogen en el sexto capítulo los detalles mas significativos de la implementación final del sistema. Se presentan los parámetros configurables y las distintas soluciones dadas a las distintas capas de nuestra arquitectura.

Capítulo VII: Resultados Este capítulo recoge y presenta los resultados obtenidos en las distintas pruebas realizadas con el sistema.

Capítulo VIII: Conclusiones y Trabajo Futuro Exponemos en este capítulo final nuestras valoraciones después de haber realizado y probado este trabajo y se presentarán cuáles son las líneas de trabajo futuro que se abren a partir de aquí.

Apéndices En los distintos apéndices se presentan resultados de distintos tests y la interfaz de comandos de los dispositivos hardware.

Capítulo 2

Metodología, recursos y plan de trabajo

2.1. Metodología

La Ingeniería del Software nos proporciona la metodología seguida en este proyecto. La Ingeniería del Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software (IEEE, 1993). Esta disciplina proporciona métodos y técnicas para conseguir desarrollar software de calidad.

La estructuración de nuestro software se ha hecho según el modelo de máquina abstracta o modelo de capas. Existen, dentro del diseño arquitectónico, distintos modelos específicos que muestran cómo los subsistemas comparten datos, cómo están distribuidos y cómo se conectan entre ellos [8]. En concreto, el modelo de capas organiza el sistema en una serie de niveles cada uno de los cuales suministra un conjunto de servicios. Este modelo permite el desarrollo incremental del sistema y es, además, una arquitectura fácil de cambiar y portable. Esta estructura se adapta perfectamente a los objetivos de nuestro proyecto en el que, partiendo de un determinado hardware, se deberá ir construyendo una serie de módulos que, sustentándose en el módulo anterior, irán construyendo un modelo cada vez más abstracto del mundo.

Entre los paradigmas de desarrollo ofrecidos por la Ingeniería del Software, el modelo incremental se adapta a los objetivos de este proyecto en los cuales se propone desarrollar una serie de módulos cada uno de los cuales usará los resultados del módulo anterior: control de los dispositivos hardware, actualización de un mapa 3D a partir de la información del entorno, creación de herramientas de visualización del mapa 3D y extracción de primitivas y proyecciones 2D a partir del mapa 3D.

El modelo incremental combina elementos del modelo lineal secuencial con la filosofía interactiva de construcción de prototipos [9]. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia o fase genera un incremento del software. Este proceso se repite hasta que se elabore el producto completo. El modelo incremental es consustancialmente iterativo, como otros enforques evolutivos. A diferencia de la técnica de

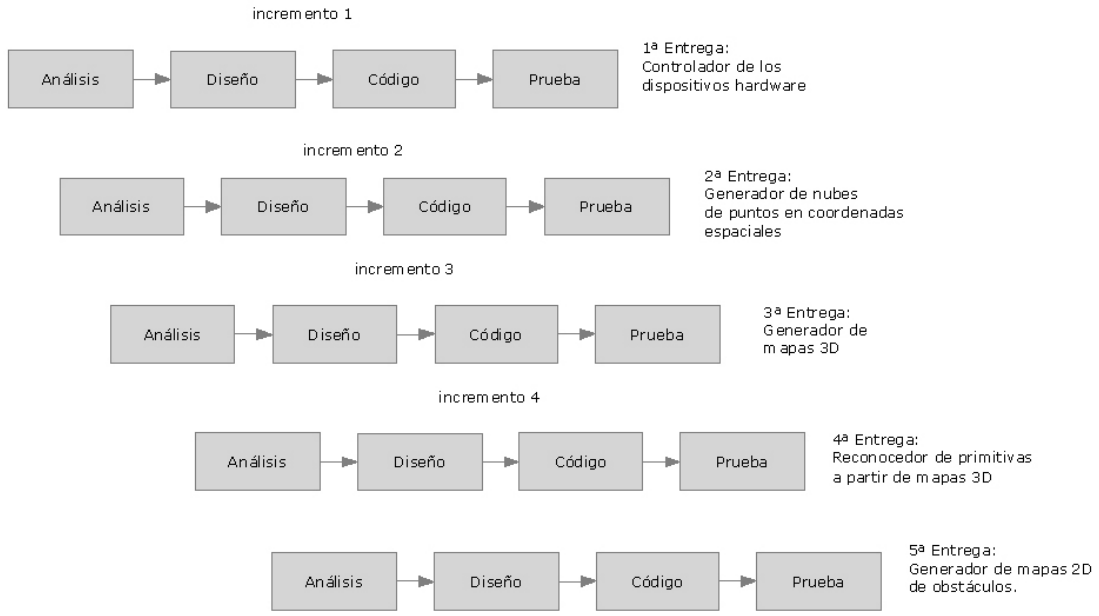


Figura 2.1: Modelo de desarrollo incremental aplicado al proyecto

prototipado, el modelo incremental se centra en la entrega de un producto operacional con cada incremento. Siendo los primeros incrementos versiones incompletas aún, proporcionan una determinada funcionalidad usable y evaluable. Vemos en la figura 2.1 la aplicación de este modelo a nuestro proyecto.

2.2. Recursos

Este proyecto, por su propia naturaleza, se soporta en gran medida en el hardware. Se ha usado un sensor láser de pequeño tamaño para así poder montarlo sobre un dispositivo apuntador. Un ordenador personal (PC) realizará las funciones de controlador y procesador del sistema. Se limita la capacidad de memoria y procesador del PC para poder instalar y ejecutar el sistema en los equipos que vienen empotrados en los robots de desarrollo habituales.

Las demandas de software no son grandes: un compilador C++ y librerías gráficas, además del sistema operativo y los drivers correspondientes.. Además, se ha buscado el poder desarrollar todo el proyecto usando software libre, lo que ha sido posible.

2.2.1. Recursos Hardware

- PC con 2 GB de RAM y puertos serie y/o USB 1.0 o superior
- Conversor de puerto USB a RS232

- Conexión a internet.
- Dispositivo apuntador «Pan-Tilt» de dos grados de libertad de Directed Perception, modelo PTU46–17.5
- Sensor de rango láser, marca Hokuyo, modelo URG-04LX

2.2.2. Recursos Software

- Sistema operativo Linux
- Compilador GNU g++ de lenguaje C++
- Librerías OpenCV
- Librerías OpenGL
- Drivers del dispositivo apuntador y del sensor láser indicados en los recursos hardware desarrollados por el grupo de visión y robótica del Instituto Universitario de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería (IUSIANI) de la Universidad de Las Palmas de Gran Canaria.

2.3. Plan de trabajo y temporización

El plan de trabajo de esta propuesta se articula de forma natural en base a la secuencia de subobjetivos que se plantean en el proyecto.

Etapa A: Estudio de los dispositivos hardware (sensor láser, dispositivo apuntador y robot) y su integración.

Cada uno de los dispositivos dispone de librerías de control, por lo que no ha sido necesario trabajar en el desarrollo de las mismas. Durante esta primera fase se experimentó con las características de operación. Una vez familiarizado con cada uno de los dispositivos, se trabajo en conseguir la plena integración de todos ellos.

Un hito importante a conseguir fue la sincronización del barrido del pan-tilt y el sensor láser. Esto ha supuesto una mejora en las prestaciones del sistema al permitir hacer movimientos suaves y continuos del pan-tilt mientras se conoce con precisión la localización del plano de escaneo.

Tiempo de dedicación: 50 horas.

Etapa B: Estudio de los marcos de representación 3D más habituales

Entre dichos marcos se incluyen: mapas de celdillas [10], mallas de triángulos, mapas de elevación [3] y mapas de superficie multinivel [4] e implementación de un marco de representación basado en los mapas de superficie multinivel.

Se han estudiados múltiples marcos de representación 3D que en el capítulo correspondiente a esta etapa se presentarán de forma concisa. Más en extensión se hablará de los dos métodos seleccionados para su implementación en este proyecto: los mapas multinivel y el algoritmo RANSAC eficiente.

El primero de los métodos permite tratar con el hecho de que en la realidad es frecuente encontrar entornos con superficies a distintos niveles: mesas, puentes, etc. Incluir en la representación del mundo este hecho permite un mejor aprovechamiento del espacio transitable y una mayor capacidad de movimiento a robots en entornos reales y desconocidos.

La implementación del algoritmo RANSAC seleccionado, una mejora del RANSAC original [5], permite detectar distintas formas básicas del entorno: planos, cilindros, esferas, conos, toros, etc. Esto dota a nuestro sistema de un mayor nivel de abstracción que se usará para la localización de estructuras complejas.

Un desafío que se ha tenido que superar es la integración de ambos métodos. De esta forma, el e-RANSAC diseñado para trabajar con nubes de puntos se ha tenido que adaptar a las estructuras de datos usadas en los mapas multinivel. Esto será explicado con detalle más adelante.

Tiempo de dedicación: 120 horas.

Etapa C: Validación experimental del sistema en entornos de interior.

Se pretende que la experimentación se pueda desarrollar, tanto a nivel de simulador (Player-Stage-Gazebo, http://playerstage.sourceforge.net/), como sobre el sistema real. Para facilitar la exploración no guiada del robot, el sistema se ha integrado con un módulo de evitación de obstáculos y planificación de caminos ya desarrollado.

Tiempo de dedicación: 60 horas.

Etapa D: Elaboración de la documentación final.

Aunque etapa final, en paralelo a la ejecución del proyecto se ha venido desarrollando la presente memoria.

Tiempo de dedicación: 70 horas.

La gráfica 2.2 muestra la proporción temporal dedicada a las distintas fases del proyecto.

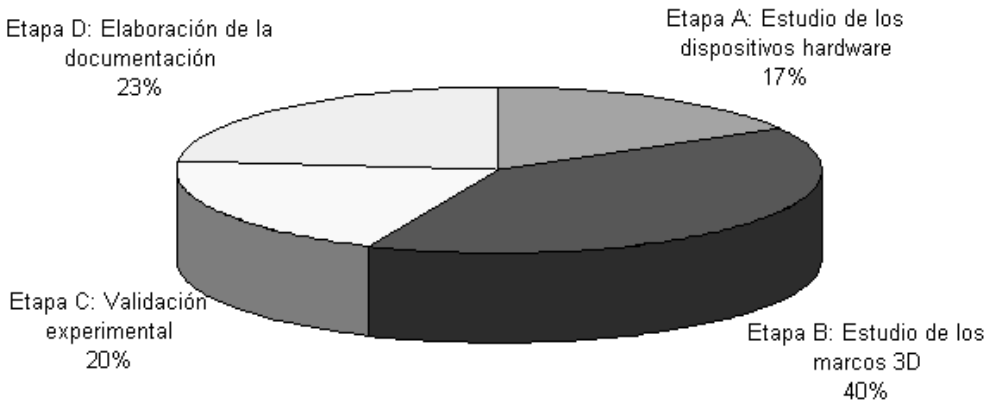


Figura 2.2: Distribución temporal de las distintas fases del proyecto

Capítulo 3

Estudio de los dispositivos hardware y su integración

Este proyecto tiene como objetivo crear mapas tridimensionales del entorno a partir de la información proporcionada por un sensor láser activo. El sensor láser usado es el modelo URG-04LX de la empresa Hokuyo. Este sensor realiza un barrido de su rayo en un único plano. Por tanto, para poder obtener información tridimensional del mundo, se montará el sensor sobre un dispositivo apuntador de Directed Perception, modelo PTU46-17.5 [11]. Este dispositivo posee dos grados de libertad lo que nos permite realizar un barrido del entorno.

Ambos dispositivos se conectan a un PC, el cual controlará el movimiento del dispositivo apuntador, recogerá la información enviada por el sensor e irá construyendo la representación del mundo tal y como se verá en el próximo capítulo.

Cada uno de los dispositivos posee su propio protocolo de comunicaciones. Se usarán en la integración con el ordenador sendos drivers ya desarrollados por el laboratorio xxxx del IUSIANI. Una cuestión importante será tener siempre conocimiento preciso de la localización espacial del punto apuntado por el sensor. La integración del sensor con el dispositivo apuntador hará necesario plantear un problema de cinemática directa para conocer las coordenadas espaciales del punto obtenido por el sensor. Sin perder precisión en la localización del punto, se ha buscado que el movimiento del dispositivo apuntador sea lo más rápido posible, de forma que el barrido del entorno pudiera hacerse en el menor tiempo posible. Sin embargo, esto entraña resolver varios problemas de sincronización entre el barrido del haz láser y el desplazamiento del dispositivo apuntador.

En las siguientes secciones se describirá con detalle el funcionamiento de los dos elementos hardware principales, así como la integración entre ambos y a su vez con el PC.

Figura 3.1: Sensor láser Hokuyo URG-04LX

3.1. Sensor láser activo

En este proyecto el sensor láser que se ha empleado es el modelo URG-04LX de la empresa Hokuyo (ver fig. 3.1). Los motivos principales de la elección de este sensor son sus reducidas dimensiones y su pequeño peso. Estas características lo hacen adecuado para poder montarlo sobre un dispositivo apuntador. Además, su menor coste en comporación con otros sensores láser también lo hacen adecuado para un proyecto final de carrera.

El sensor tiene unas dimensiones de 50 mm de ancho, 50 mm de profundidad y 70 mm de alto, pesando aproximadamente 160g. Otras especificaciones del sensor se muestran en la tabla 3.1.

El haz láser del sensor realiza su barrido en un plano perpendicular a la altura del dispositivo, rotando en sentido anti horario cuando es visto desde arriba. La resolución angular del sensor es de $0{,}35^o$. Esta resolución se obtiene al dividir los 360^o de la circunferencia en 1024 posiciones distintas, llamadas pasos. Sin embargo, el sensor no obtiene información desde los 1024 pasos. Sólo es capaz de proporcionar lecturas válidas entre los pasos 44 y 725, ambos inclusive. Esto proporciona un área de medición de 240^o aproximadamente, con un total de 682 pasos. El resto se denomina zona muerta. Los 0^o se sitúan en la parte frontal del sensor y el rango medible se extiende entre los -120^o y las 120^o. Podemos ver esto representado en la figura 3.2

Es de singular importancia para este proyecto la frecuencia de funcionamiento del sensor láser ya que, al ir montado sobre un dispositivo móvil, habrá que sincronizar el barrido el láser con el desplazamiento del apuntador. De esta forma, podremos estimar a qué punto del espacio corresponde la información suministrada por el sensor. El Hokuyo URG-04LX es capaz de realizar una rotación completa del haz en un tiempo mínimo de 100 ms, siendo por tanto su frecuencia máxima de funcionamiento de 10 Hz. Se ha comprobado la precisión de esta frecuencia mediante un osciloscopio. El sensor proporciona en una de las patillas de su interfaz una señal de sincronía. Cada vez que el haz inicia una nueva rotación, esta señal presenta un flanco de subida. El conocimiento

Tabla 3.1: Especificaciones del Sensor Láser Hokuyo URG-04LX

Fuente de luz	Diodo semiconductor láser(=785nm)
Seguridad láser clase	1
Fuente energía	5V DC ±5 %
Consumo de corriente	500 mA o menos
Distancia de detección	20mm 400mm
Precisión Distancia	20-1000mm: $\pm 10mm$
Distancia	1000 4000mm:±1 % de la medida
Resolución	1 mm
Ángulo de barrido	240º
Resolución angular	Pasos de (360/1024)º
Tiempo de barrido	100 mseg/barrido
Interfaz RS-232C	(19.2, 57.6, 115.2 kbps)
USB	Versión 2.0 FS mode (12 Mbps)
Temperatura Humedad	-10 50ºC 85 % o menos
Chasis	Policarbonato

Figura 3.2: Zona de barrido del sensor Hokuyo URG-04LX

de la velocidad de rotación será tenido en cuenta en la integración con los demás componentes hardware para conseguir una alta velocidad de desplazamiento del conjunto con una alta precisión en el posicionamiento del punto escaneado. Veremos cómo se ha conseguido esta integración con más detalle en el apartado 3.3

Un aspecto sensible en la realización del proyecto es la fiabilidad de la medida. Según la información proporcionada por el fabricante, el sensor tiene un error en la medida de ± 10 mm en objetos situados a menos de un metro y de $\pm 1\,\%$ de la distancia de objetos más distantes. El sensor, tras hacer un barrido completo, devuelve en un buffer las medidas obtenidas en milímetros, correspondientes a cada paso entre los -120^o y los 120^o. Si en algún punto hubiese obtenido una medición incorrecta, informa devolviendo un código entero menor que 20 (20 mm. es la distancia mínima que puede detectar el sensor). La estimación de la incertidumbre, ν, en la medida se ha caracterizado en funciçon de la distancia medida m, conforme a la información del fabricante, de la siguiente forma:

$$\nu = \begin{cases} \frac{10}{m} & si \quad m < 1000 \\ 0{,}01 & sim \quad \geq 1000 \end{cases}$$

Hemos realizado una serie de tests para comprobar el comportamiento del sensor en una situación real. Queremos averiguar qué variación presenta en la medición de un objeto estático y también queremos saber si su sensibilidad es igual en todo el rango de medición. Para ello hemos situado al sensor a diferentes distancias de una pared, formando distintos ángulos y hemos registrado 500 lecturas en cada posición. En el apéndice B se muestran los resultados. Estos tests permitieron detectar un mal funcionamiento en el sensor que se venía empleando. Se realizaron los tests con un segundo sensor que sí presentó un comportamiento acorde con el esperado según el fabricante. En el apéndice se comparan los resultados de ambos sensores.

3.2. Dispositivo apuntador

El dispositivo apuntador va a servir para dotar al sistema de la capacidad de obtener información del entorno tridimensional. Se ha usado en este proyecto un dispositivo apuntador de la empresa de Directed Perception [11], modelo PTU 46-17.5 (figura 3.3)

Este dispositivo apuntador posee dos grados de libertad: uno horizontal (pan) y otro vertical o de azimut (tilt). Sobre la estructura que se ve en la parte superior se montará el sensor. Vemos algunas características del modelo en la tabla 3.2.

La característica esencial de este dispositivo apuntador es que proporciona un control preciso de la velocidad y aceleración de los ejes en base a dos motores paso a paso. Como se puede ver en la figura 3.4, los extremos entre los que puede oscilar una velocidad no estacionaria están determinados por la velocidad límite superior y la velocidad límite inferior. La velocidad base, o velocidad inicial, se refiere a la velocidad a la cual los ejes pueden ser iniciados partiendo de una

Figura 3.3: Dispositivo apuntador PTU 46-17.5 de Directed Perception

Tabla 3.2: Unidad PTU-D46-17.5

Peso	1.38 kg
Dimensiones	7,62cm ancho 13.03cm alto 10.795cm fondo
Velocidad máxima sin carga	300^o por segundo
Resolución	0.013^o
Rango de elevación	$-47^o a 31^o (78^o)$
Rango de azimut	$\pm 159^o$

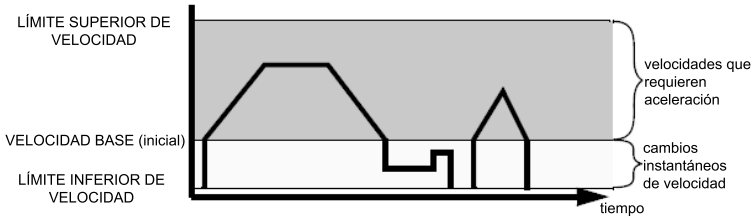


Figura 3.4: Velocidad en los ejes, Velocidades instantáneas, Aceleración trapezoidal y Cambios Instantáneos de velocidad y posición

parada absoluta sin pérdida de sincronía. Debido a los requisitos de la velocidad base y a que el par motor disminuye a medida que se incrementa la velocidad, se necesita aceleración para alcanzar velocidades por encima del nivel base. El controlador de la unidad usa aceleración y frenado trapezoidal para velocidades comprendidas entre la base y el máximo permitido.

La figura 3.4 muestra dos casos en los que se produce aceleración. En el primero, uno de los ejes acelera hasta alcanzar una velocidad constante deseada y, posteriormente, frena. En el segundo caso, vemos la situación en que la unidad no tiene tiempo suficiente para acelerar hasta una velocidad constante antes de que necesite frenar para llegar a la posición deseada. Este esquema se usará para estimar la posición del dispositivo apuntador en un determinado instante de tiempo. De esta forma, podremos localizar con precisión la situación de un objeto señalado por el sensor láser.

3.3. Integración de los dispositivos hardware

Los dos principales elementos hardware han tenido que integrarse cada uno con el ordenador y a su vez entre ellos. La interacción entre el ordenador y cada dispositivo ha tenido que afrontar tareas como la conexión física y el protocolo de comunicación. En ese proyecto se ha usado unos controladores ya desarrollados por el laboratorio xxx del IUSIANI, estando por tanto determinado el formato de los comandos y el de los paquetes de datos.

La interacción de ambos dispositivos entre sí ha generado dos tareas que resolver. En primer lugar, se ha abordado la sincronización entre la frecuencia de barrido y el desplazamiento del apuntador. El movimiento del apuntador tiene que ser tal que, maximizándose su velocidad, no dejemos zonas del espacio sin barrer y no perdamos conocimiento de a dónde apunta cada rayo del sensor. En segundo lugar, debemos poder conocer las coordenadas espaciales del punto señalado en cada instante por el sensor.

3.3.1. Integración de los dispositivos con el ordenador

El sensor láser se conecta al ordenador a través de un puerto usb, sin embargo el dispositivo apuntador viene dotado de una interfaz serie. En nuestro caso hemos optado por conectar un cable

conversor USB a serie, modelo PA088U de la empresa Targus entre el ordenador y el dispositivo apuntador. La elección de este sistema de conexión ha sido el hecho de que los puertos USB son de uso frecuente en los actuales ordenadores mientras que los puertos serie no se incorporan ya a muchos equipos.

El sensor láser acepta siete tipos de comandos predefinidos [12]. Los tipos de comandos son: comandos de petición de información del sensor, comandos de habilitación / deshabilitación de la medición, un comando de ajuste de la comunicación por el puerto RS-232, comandos de adquisición de datos de medidas, un comando de ajuste de la velocidad del motor, un comando de adquisición/ajuste de la hora y un comando de reinicio. En los apéndices los distintos comandos serán presentados, pero ahora haremos un comentario en detalle únicamente de los comandos de adquisición de datos y del comando de ajuste de la velocidad del motor.

El sensor proporciona dos comandos para la adquisición de los datos de una medición. El primero de ellos (comando `GD-GS`) sirve para solicitar los resultados de la última medición efectuada por el sensor. Mientras, el segundo comando (comando `MDMS`) solicita al sensor las medidas tomadas después de que el comando haya sido procesado. Nosotros hemos usado exclusivamente este segundo comando, ya que, para poder estimar con precisión las coordenadas espaciales de las medidas proporcionadas por el láser, necesitamos conocer el instante preciso en que se efectuó el barrido del haz y, de esta forma, poder cotejar ese instante de tiempo con la posición del dispositivo apuntador.

Los dos comandos, por otra parte, usan los mismos parámetros: paso inicial, paso final y tamaño de la agrupación. El formato del comando es:

G	D o S	Paso inicial (4B)	Paso final (4B)	Agrupación (2B)	String	LF

Los pasos inicial y final especifican los límites de la región de la que queremos obtener información. Hay que recordar que cada paso del sensor equivale a $360^o/1024 = 0{,}35^o$ y que los 0^o están situados en la parte frontal del sensor (fig 3.2). El tamaño de la agrupación es una forma de suavizar el desvío que puede haber entre la distancia real y la medida. El tamaño de la agrupación es el número de pasos adyacentes que serán combinados en un único dato. Cuando el tamaño de la agrupación, n, es mayor que uno, se devuelve de cada n pasos aquel cuya distancia medida sea menor. Tras realizar un barrido el sensor devuelve en un array los valores de distancia obtenidos. El sensor usa como unidad el milímetro. Cuando la medida es correcta, devuelve un valor mayor o igual a 20. Sin embargo, si el valor devuelto es menor que 20 se trata de un código de error. Vemos en la tabla 3.3 los posibles códigos de error devueltos por el sensor.

La velocidad de rotación del sensor se puede ajustar en el modelo URG-04LX entre 540 y 600 rpm mediante el comando `CR`. Inicialmente la velocidad está ajustada a 600 rpm, la máxima, pero puede ser conveniente modificarla cuando se usan múltiples sensores para evitar problemas de interferencia. En nuestro proyecto hemos usado un solo sensor y, por tanto, la velocidad se deja ajustada al máximo posible para mejorar la respuesta del sistema. El formato del comando es:

El controlador del dispositivo apuntador proporciona funciones para ajustar la velocidad de desplazamiento y para mover los ejes a una posición deseada. Existen dos formas, absoluta o rela-

Tabla 3.3: Códigos de error en la medida devueltos por el URG-04LX

Código	**Tipo de error**
0	Objeto detectado posiblemente a 22m
1	La luz reflejada tiene poca intensidad
2	La luz reflejada tiene poca intensidad
3	La luz reflejada tiene poca intensidad
4	La luz reflejada tiene poca intensidad
5	La luz reflejada tiene poca intensidad
6	Otros
7	Las distancias de los pasos precedente y posterior tienen errores
8	La intensidad difiere en dos ondas
9	El mismo paso tuvo errores en las das últimas mediciones
10	Otros
11	Otros
12	Otros
13	Otros
14	Otros
15	Otros
16	Otros
17	Otros
18	Error de lectura debido a objetos fuertemente reflectivos
19	Paso no medible

C	R	Velocidad (2B)	String	LF

tiva, de indicar un cambio en la velocidad o en la posición. Cuando se hace un cambio de forma absoluta, el valor pasado como parámetro será la nueva posición o velocidad. Como se veía en la figura 3.4, si la nueva velocidad supera la velocidad base, el cambio de velocidad no será instantáneo cuando se produzca un cambio de posición. Por el contrario, será necesario esperar un tiempo para que se alcance la velocidad deseada. Además, si la posición final está demasiado próxima, es posible que se empiece a frenar antes de haber alcanzado la velocidad especificada. Por otra parte, tanto la velocidad como la posición se pueden especificar de forma relativa, esto es, como una variación respecto al valor actual de uno u otro parámetro.

El dispositivo apuntador nos permite especificar cambios en la velocidad o en la posición «al vuelo». Esto es, mientras se está produciendo un desplazamiento, y sin que se haya completado, podemos especificar una nueva posición o una nueva velocidad. Si alcanzar la nueva posición o velocidad requiere frenar o acelerar, el controlador se encargará de todo ello. Sin embargo, puede ser necesario en ocasiones, el tener que sincronizar el posicionamiento del dispositivo con otras funciones del sistema. Para ello el controlador proporciona una función que no retorna hasta que el posicionamiento actual haya finalizado. Con esta función, podemos asegurar que el dispositivo se encuentra en la posición deseada antes de continuar con otras tareas.

3.3.2. Integración del sensor láser con el dispositivo apuntador

En la figura 3.5 podemos ver una foto del sistema formado por el dispositivo apuntador y el sensor láser. Como se ha comentado anteriormente, se ha buscado el poder mover el dispositivo apuntador a la mayor velocidad que permitiera realizar un barrido exhaustivo del entorno y que no mermara la precisión de la localización espacial de los puntos obtenidos por el sensor. Esto conlleva resolver distintos problemas de sincronización: conocer el instante exacto en que arranca el dispositivo apuntador, conocer el instante exacto en que cada medida ha sido adquirida por el láser y hacer corresponder cada medida con una posición exacta del dispositivo apuntador. El sensor láser usado en este proyecto, el Hokuyo URG-04LX, proporciona por una patilla de su interfaz una señal que presenta un flanco de subida cada vez que el haz inicia una nueva rotación. Sin embargo, esta señal que podría servir de base para la sincronización, no ha podido usarse en este proyecto al carecer el dispositivo apuntador de un mecanismo de control por línea. El dispositivo apuntador posee su propio controlador el cual sólo puede ser gobernado mediante el paso de comandos. Por tanto, los problemas de sincronización se han abordado desde una estrategia software como se mostrará en los siguientes párrafos. Posteriormente se aborda el problema de localización espacial de las medidas proporcionadas por el sensor láser.

La obtención de información espacial del entorno motiva que el sensor láser, el cual realiza barridos en un único plano, haya sido montado sobre un dispositivo apuntador. El sistema irá moviendo progresivamente los motores horizontal y vertical del apuntador para así poder barrer regiones del espacio. Hay que determinar con qué resolución se va a realizar el barrido, esto es, cuántos grados

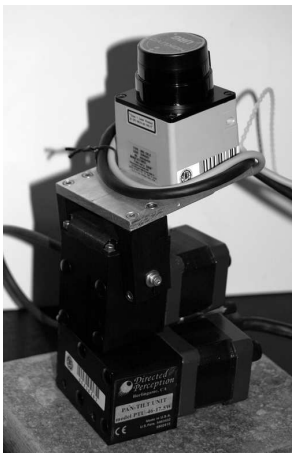


Figura 3.5: Foto del sensor láser montado sobre el dispositivo apuntador

se desplazará el dispositivo apuntador en lo que el sensor láser realiza una rotación completa. Éste será un parámetro configurable del sistema. El movimiento del sensor láser, arrastrado por el dispositivo apuntador, genera la necesidad de conocer dónde está el dispositivo cada vez que el láser realiza una medición. De esta forma podremos localizar espacialmente cada una de las lecturas de láser. El sensor usado posee una frecuencia máxima de barrido de 10 Hz., es decir, tarda 0,1 segundos en hacer una rotación completa del haz láser. Una primera estrategia posible de sincronía sería, en código algorítmico, la que se muestra en el algoritmo 1 (ver fig. 3.6).

```
mover apuntador(posición inicial)
esperar finalización movimiento apuntador
mientras no se haya alcanzado posición final hacer
    obtener medición
    procesar medición
    mover apuntador(siguiente posición)
    esperar finalización movimiento apuntador
fin mientras
```

Figura 3.6: Algoritmo 1: Algoritmo de adquisición de datos que detiene al dispositivo apuntador antes de adquirir datos con el láser

Con este sencillo esquema garantizaríamos, en todo momento, la sincronía entre el barrido del sensor y el movimiento del apuntador y, como consecuencia, podríamos obtener con total precisión la localización espacial de las lecturas del láser. La estrategia consiste simplemente en separar ambos movimientos, el del dispositivo apuntador y el del haz láser. Se puede ver en el código anterior que, tras cada orden de reposicionamiento del apuntador, el algoritmo espera que éste informe de que ha acabado de efectuar su movimiento. Cuando ya el apuntador está quieto en la nueva y conocida posición, se solicita al sensor que realice una medición, se obtiene y se procesa. Sin

embargo, esta sencilla estrategia tiene como contrapartida el hacer muy lento el barrido espacial del entorno al introducir unas latencias entre medición y medición de láser.

Un segundo algoritmo solventa el problema anterior evitando tener que esperar continuamente a que el dispositivo apuntador se reposicione. Hemos visto en la sección 3.2 que es posible ajustar la velocidad de desplazamiento del apuntador. Seleccionando adecuadamente esta velocidad, podremos mover de forma continua el dispositivo apuntador entre la posición inicial y la final al mismo tiempo que el sensor va realizando las medidas en los puntos deseados. El sensor permite una velocidad máxima de barrido de 600 rpm, es decir, una barrido en 0'1 segundos. Podemos, entonces, ajustar la velocidad de desplazamiento del apuntador de forma que entre una posición y la siguiente tarde 0'1 segundos. Esto nos permitirá realizar el desplazamiento del dispositivo apuntador entre la posición inicial y la final de forma suave y continua, sin tener que esperar a que complete un sinfín de pequeños saltos. Veamos un ejemplo. Supongamos que queremos hacer un barrido vertical, dejando parado el motor horizontal, entre los 0º y los 5º con una resolución de una medición cada 0'5º. Con esta resolución, los ángulos en los que queremos que se inicie los distintos barridos del láser son los del siguiente conjunto: $\{0^o, 0'5^o, 1^o, 1'5^o, 2^o, 2'5^o, 3^o, 3'5^o, 4^o, 4'5^o, 5^o\}$. Sabemos que la frecuencia máxima de barrido del sensor láser es de 10Hz., por lo tanto, a esta frecuenca, tenemos que ajustar la velocidad del apuntador a $0{,}5^o$ cada 0,1 segundos (duración del barrido del sensor). En general, la velocidad de desplazamiento del dispositivo apuntador, vd, en grados por segundo para una resolución dada, r, en grados y una frecuencia de barrido del haz láser, f, en hertzios se obtiene como:

$$vd = r \cdot f$$

La velocidad resultante en nuestro ejemplo es $0{,}5^o \cdot 10Hz = 5^o/s$. El algoritmo modificado se puede ver en la figura 3.7.

```
mover apuntador(posición inicial)
esperar finalización movimiento apuntador
modificar velocidad apuntador(resolución deseada)
mover apuntador(posición final)
para i=posición inicial hasta posición final con
incremento=resolución deseadado hacer
    obtener medición
    procesar medición
fin para
```

Figura 3.7: Algoritmo 2: Algoritmo de adquisición de datos que adquiere datos del láser mientras se realiza un movimiento contínuo del dispositivo apuntador

Este nuevo algoritmo permite acortar sustancialmente el tiempo de barrido del entorno para una resolución determinada. Sin embargo, cabe reseñar dos aspectos de este nuevo algoritmo que

podrían provocar errores en la estimación de la posición. En primer lugar, el procesamiento de los datos se realiza de forma secuencial a la obtención de los mismos. Si el tiempo de procesamiento fuera mayor que las latencias mecánicas del láser, la estimación de la posición del dispositivo apuntador no se correspondería con la realidad, con lo que la localización espacial de las medidas del láser serían incorrectas. Es de señalar, sin embargo, que, en la implementación que se ha hecho de este proyecto, los tiempos de procesamiento suponen alrededor de un 10 % del tiempo de latencia como se señala en el capítulo 6. Sin embargo, nada garantiza que una futura ampliación en las funcionalidades del sistema no provoquen un aumento de los tiempos de procesamiento o que una mejorar en el hardware no disminuyan las latencias mecánicas. Por ello, se debería comprobar antes de cada barrido del láser la posición del dispositivo apuntador. Otra solución alternativa sería realizar la recogida de datos del sensor y el procesamiento de la información en dos hilos independientes. De esta forma, el hilo que recoge los datos del sensor los almacenaría en un buffer de tamaña suficiente para hacer frente a cuellos de botella del hilo de procesamiento.

Hay otro aspecto que, al contrario del anterior, sí que causa que en este algoritmo se estime de forma errónea la localización espacial de las medidas obtenidas. En el primer algoritmo, como vimos, tras situar al apuntador en una nueva posición, esperábamos a que se detuviera y luego hacíamos la medición con el láser. En ese caso, todas las medidas de un mismo barrido del láser eran efectuadas con un mismo ángulo de elevación vertical conocido. En el segundo algoritmo, por el contrario, realizamos la medición mientras el apuntador se está moviendo. Por tanto, a medida que el haz láser va girando, el ángulo de elevación vertical está variando. Si quisiéramos estimar con precisión la localización espacial de las medidas, deberíamos estimar cuál es dicho ángulo para todos y cada uno de los pasos del barrido del haz. Incluso con resoluciones pequeñas, en torno a $0{,}5^o$, no tener en cuenta el desplazamiento entre el principio y el fin de la rotación del haz puede suponer, a distancias de 4 metros, errores en la estimación de la posición espacial de 35 milímetros. Se puede solventar este problema calculando la posición del dispositivo apuntador en cada instante de tiempo en que fue obtenida cada una de las medidas del sensor. Habrá que tener en cuenta para ello el esquema de aceleración del dispositivo apuntador que se expuso en 3.2. Pero, aunque seamos capaces de calcular cuánto ha variado el ángulo de elevación vertical a medida que el haz ha rotado, aún no somos capaces de conocer cuál es exactamente ese ángulo en cada uno de los distintos pasos del láser.

Un tercer y último algoritmo resuelve todos los problemas planteados aquí: mover el dispositivo apuntador de forma contínua, conocer la posición del dispositivo apuntador al inicio de cada barrido del láser y estimar el ángulo de elevación vertical para cada medida proporcionada por el sensor. Podemos ver el algoritmo en la figura 3.8. En este algoritmo se obtiene en *t_inicial* el instante en que empieza a desplazarse el dispositivo apuntador. Este instante se obtiene promediando los instantes anterior y posterior a la llamada al comando que inicia el movimiento del dispositivo. Este promediado corrige las latencias que se producen entre el instante en que se invoca al comando y el instante en que éste rotorna. Posteriormente, se procede a ir adquiriendo mediciones del sensor láser. El sensor incluye en el array que devuelve tras cada barrido el instante (*timestamp*) en que comenzó el barrido. Como en un barrido el haz láser pasa por 1024 posiciones, para una frecuencia f dada, el instante de tiempo en que se tomó la lectura l_p, siendo p el paso de la lectura, se calcula

como:

$$t(p) = timestamp + \frac{p}{1024 \cdot f}$$

Por tanto, el tiempo transcurrido desde el inicio del desplazamiento del sensor hasta que se tomó la lectura l_p ha sido $t(p) - t\_inicial$. Con este lapso de tiempo se puede calcular la posición del dispositivo apuntador tomando en cuenta el esquema de aceleración trapezoidal que se expuso en 3.2. El procedimiento que calcula la posición se muestra en la figura **??**. El procedimiento discrimina tres posibles casos:

1. La velocidad de desplazamiento es inferior o igual a la velocidad base. En este caso el dispositivo apuntador no requiere aceleración, por lo que la velocidad es constante y la posición se estima mediante una interpolación lineal.

2. La velocidad de desplazamiento es superior a la velocidad base y antes de terminar de acelerar se alcanza la posición media de desplazamiento. En este caso, el dispositivo apuntador sigue un esquema de aceleración triangular. Se calcula si el instante de tiempo para el que se va a calcular la posición es anterior o posterior al vértice del triángulo y se obtiene la posición buscada.

3. La velocidad de desplazamiento es superior a la velocidad base y se termina de acelerar antes de alcanzar la posición media de desplazamiento. Aquí el esquema de aceleración es trapezoidal. Habrá que calcular si el instante de tiempo para el que se quiere obtener la posición está situado sobre la rampa ascendente de aceleración, soble la cima horizontal de velocidad constante o sobre la rampa descendente de frenado. En función de la situación, se calcula la posición del dispositivo apuntador.

Con este último algoritmo se solventan los problemas de sincronía entre la rotación del haz láser y el desplazamiento del dispositivo apuntador al tomar en consideración el instante en que comienza el desplazamiento y el instante en que el sensor láser realiza cada barrido. El problema de localización se solventa calculando la posición del dispositivo apuntador en base al esquema de aceleración proporcionado por el fabricante e incorporando la información sobre el instante de tiempo en que se tomó cada medida. Sin embargo, como se comentó anteriormente, se aborda el problema de sincronía desde una solución software dentro de un sistema que no es de tiempo real. Esto conlleva que la precisión a la hora de determinar los distintos eventos, arranque del dispositivo apuntador y adquisición de las medidas, siempre será aproximado. Estos errores no serán significativos para el uso que se va a hacer aquí de la información proporcionada en esta primera capa del sistema. En primer lugar, el rango de medidas del láser empleado, hasta los 5m., hace que el efecto de los pequeños errores de precisión no sean significativos. Por otra parte, estas mediciones se usarán para crear un mapa discretizado del entorno. La discretización usual en las pruebas ha sido de 5cm. Esta discretización absorverá la mayor parte de las desviaciones de las medidas. Sin embargo, en un sistema que fuera dotado de sensores láser de largo alcance o para aplicaciones sin discretización, sería necesario implementar un sistema de sincronización en tiempo real por hardware entre el dispositivo apuntador y el láser.

```
mover apuntador(posición inicial)
esperar finalización movimiento apuntador
modificar velocidad apuntador(resolución deseada)
t1 := obtener hora
mover apuntador(posición final)
t2 := obtener hora
t_inicial := (t1+t2)/2
repetir
    medidas := obtener siguiente scan del láser
    /*
      Almacenamos en t_scan el instante en que se inició el último barrido del
      sensor. El sensor devuelve este dato en el array de medidas
    */
    t_scan := medidas.timestamp
    para cada lectura en medidas hacer
      tilt_angle := aceleración_trapezoidal(velocidad, velocidad base,
      aceleración, t_scan-t_inicial, ángulo inicial, ángulo final)
      t_scan := t_scan + 1/(frecuencia * 1024)
      procesar(lectura, tilt_angle)
    fin para
    tilt := obtener del dispositivo apuntador el ángulo tilt
hasta que tilt < posición final
```

Figura 3.8: Algortimo 3: Algoritmo de adquisición de datos que adquiere datos del láser mientras se realiza un movimiento contínuo del dispositivo apuntador con estimación precisa de la posición en cada instante. El algoritmo hace uso del esquema de aceleración trapezoidal del dispositivo apuntador

Una vez conocida la posición del dispositivo apuntador para cada una de las medidas suministradas por el láser, queda el problema de convertir estas medidas en coordenadas espaciales del mundo. Esta tarea será resuelta mediante las herramientas de la cinemática robótica. La cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia. Existen dos problemas fundamentales para resolver en la cinemática del robot. El primero de ellos se conoce como el problema cinemático directo, y consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot. El segundo problema, el problema cinemático inverso, resuelve la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas. De ambos problemas, es el primero el que coincide con la situación que tenemos que resolver en nuestro sistema.

Antes de afrontar el problema cinemático directo debemos modelar nuestro sistema como cadena cinemática. Una cadena cinemática es un conjunto de eslabones y articulaciones interconectados de modo que proporcionan un movimiento de salida controlado en respuesta a un movimiento de entrada proporcionado. Los eslabones son una serie de elementos estructurales sólidos. A su vez, cada articulación proporciona, al menos un grado de libertad. Existen dos tipos de articulaciones: prismáticas y rotacionales. La articulación prismática es aquella en la que el eslabón se apoya en un deslizador lineal. La articulación rotacional es tal que el eslabón realiza un giro pivotando sobre la propia articulación. En nuestro sistema sólo dispondremos de articulaciones giratorias. En concreto, nuestro sistema dispone de tres articulaciones. Las dos primeras articulaciones las proporciona el dispositivo apuntador con sus dos motores. La tercera articulación será el motor del sensor considerando al propio haz láser como un eslabón más. Efectivamente, nuestro interés radica

en localizar espacialmente las coordenadas del punto señalado por el sensor. Por ello, aunque el haz láser no sea un eslabón al uso, sólido, lo consideramos como un eslabón más de la cadena el cual, en cada punto detectado, vendrá determinado por el ángulo del haz y la distancia del punto alcanzado.

Denavit y Hartenberg [13] propusieron en 1955 un método sistemático para describir y presentar la geometría espacial de los elementos de una cadena cinemática, y en particular de un robot, con respecto a un sistema de referencia fijo. Este método utiliza una matriz de transformación homogénea para describir la relación espacial entre dos elementos rígidos adyacentes, reduciéndose el problema cinemático directo a encontrar una matriz de transformación homogénea cuatro por cuatro que relacione la localización espacial del extremo del robot con respecto al sistema de coordenadas de su base. Según la representación de Denavit-Hartenberg, escogiendo adecuadamente los sistemas de coordenadas asociados para cada eslabón, será posible pasar de uno al siguiente mediante cuatro transformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.

Presentamos a continuación un algoritmo basado en el método de Denavit y Hartenberg para resolver el problema cinemático directo. El algoritmo se estructura en ocho pasos consecutivos:

1. Se localiza cada uno de los ejes de las articulaciones y se etiquetan como $z_0, \ldots, z_{n-1}$

2. Se establece el sistema de coordenadas base. Se sitúa el origen en cualquier punto del eje z_0. Los ejes x_0 e y_0 son elegidos convenientemente para que el sistema sea dextrógiro. Los pasos 3, 4 y 5 se realizan para $i = 1, ..., n-1$

3. Se ubica el origen de coordenadas O_i en la intersección entre el eje z_i y la normal común a z_i y z_{i-1}. Si z_i y z_{i-1} son paralelos, se ubica O_i en cualquier punto que convenga de z_i

4. Se establece x_i a lo largo de la normal común entre z_{i-1} y z_i a través de O_i, o en la dirección normal al plano z_{i-1}, z_i si z_{i-1} y z_i se intersectan

5. Se establece y_i para que el sistema resultante sea dextrógiro.

6. Se localiza el n-ésimo sistema de coordenadas en el extremo del robot. En nuestro sistema lo localizaremos en el extremo del haz láser.

7. Se crea una tabla de parámetros de cada eslabón $a_i, d_i, \alpha_i, \theta_i$.

   - a_i = distancia a lo largo de x_i desde O_i hasta la intersección de los ejes x_i y z_{i-1}.
   - d_i = distancia a lo largo de z_{i-1} desde O_{i-1} hasta la intersección de los ejes x_i y z_{i-1}.
   - α = ángulo entre z_{i-1} y z_i medido en x_i.
   - θ = ángulo entre x_{i-1} y x_i medido en z_{i-1}.

8. Se forman las matrices de transformación A_i sustituyendo los cuatro parámetros anteriores.

Estas matrices de transformación consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento i con el sistema de referencia del elementos i-1. Las transformaciones en cuestión se componen de dos rotaciones y dos traslaciones en el siguiente orden:

1. Rotación alrededor del eje Z_{i-1} un ángulo θ_i;
2. Traslación a lo largo de Z_{i-1} una distancia d_i;
3. Traslación a lo largo de X_i una distancia a_i;
4. Rotación alrededor del eje X_i un ángulo α_i;

De este modo, la transformación del sistema de referencia i-1 al sistema i viene dado por la composición de las cuatro transformaciones anteriores. Expresadas cada una de las cuatro mediante matrices homogéneas, se tiene:

$$R_{z,\theta_i} = \begin{bmatrix} cos(\theta_i) & -sen(\theta_i) & 0 & 0 \\ sen(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trasl_{z,d_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trasl_{x,a_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{x,\alpha_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sen(\alpha_i) & 0 \\ 0 & sen(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El producto de las cuatro transformaciones nos proporciona la matriz A_i de transformación del sistema i-1 al sistema i.

$$A_i = \begin{bmatrix} cos(\theta_i) & -sen(\theta_i)cos(\alpha_i) & sen(\theta_i)sen(\alpha_i) & a_icos(\theta_i) \\ sen(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sen(\alpha_i) & a_isen(\theta_i) \\ 0 & sen(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tabla 3.4: Parámetros de la cadena cinética dispositivo apuntador - sensor láser

Eslabón	θ_i	d_i	a_i	α_i
Eslabón 1	Ángulo horizontal	h_0 mm	0mm	$-\pi/2$
Eslabón 2	-Ángulo vertical	0mm	-50mm	$\pi/2$
Eslabón 3	Ángulo del haz láser	100mm	Rango del haz (mm)	0^o

Finalmente, el producto de las distintas matrices de transformación correspondientes a cada eslabón nos permitirán expresar la localización espacial del extremo de la cadena cinemática en coordenadas referidas al sistema de referencia base.

Como ya comentamos, nuestra cadena cinemática está formada por el dispositivo apuntador y el haz láser. Dispondremos por tanto de tres eslabones, dos «físicos», proporcionados por la estructura del apuntador, y un tercer eslabón «lumínico», el propio haz láser. Las dos primeras articulaciones las forman los motores horizontal y vertical del apuntador y la tercera articulación, el motor del sensor láser (ver fig. 3.10). Para poder caracterizar nuestro sistema, debemos indicar los valores de los cuatro parámetros a_i, d_i, α_i y θ_i. El origen del sistema de coordenadas base lo situaremos, por conveniencia, en el suelo. La distancia h_0 desde este origen de coordenadas O_0 hasta O_1 se ajustará según dónde se monte el dispositivo apuntador sobre el robot. De esta forma, la distancia entre ambos sistemas de referencia será la altura del primer eslabón del dispositivo apuntador sobre el suelo. El origen del sistema de coordenadas O_1 está situado en el eje de la articulación vertical. El siguiente origen de coordenadas, O_2 se sitúa debajo del sensor, en la intersección del eje del motor del sensor con la normal al eje vertical. Finalmente, el origen último sistema de coordenadas O_3 se halla en el extremo del haz láser, hasta dondequiera que éste se extienda. Para poder ajustar los parámetros a_i y d_i se recurre a los datos de tamaño proporcionados por los fabricantes (ver fig. 3.11 y fig. 3.12)

De esta forma, sea H el ángulo formado por el motor horizontal, V el ángulo formado por el motor vertical, L el ángulo del haz láser, R la longitud del haz y h_0 la altura del primer eslabón sobre el suelo, la matrices de transformación A_i correspondientes a cada uno de los eslabones de nuestro sistema quedan de la siguiente forma:

$$A_1 = \begin{bmatrix} cos(H) & 0 & -sen(H) & 0 \\ sen(H) & 0 & cos(H) & 0 \\ 0 & -1 & 0 & h_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} cos(V) & 0 & -sen(V) & -50 \cdot cos(V) \\ -sen(V) & 0 & -cos(V) & 50 \cdot sen(V) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} cos(L) & -sen(L) & 0 & R \cdot cos(L) \\ sen(L) & cos(L) & 0 & R \cdot sen(L) \\ 0 & 0 & 1 & 100 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
entrada:
vd - velocidad programada
vb - velocidad base
a - aceleración
t - instante para el que se desea calcular el ángulo de elevación
tilt0 - ángulo de elevación inicial
tiltf - ángulo de elevación final
salida:
tilt - ángulo de elevación en el instante t
si vd<vb entonces
     tilt=tilt0+vd*t
     retornar
si no
     ta=(vd-vb)/a
     tilta=tilt0+vb*ta+a*(ta^2)/2
     si tilta>(tilt0+tiltf)/2 entonces
          tm=(-vb+raiz(vb^2-2*a*(tilt0-tiltf)/2))/a
          si t<tm entonces
               tilt=tilt0+vb*t+a*t^2/2
               retornar
          si no
               tiltm=tilt0+vb*tm+a*tm^2/2
               vm=vb+a*tm
               tilt=tiltm+vm*(t-tm)-a*(t-tm)^2/2
               retornar
          fin si
     si no
          si t<ta entonces
               tilt=tilt0+vb*t*a*t^2/2
               retornar
          si no
               tb=ta+(tiltf-2*tilta)/vd
               si t<tb entonces
                    tilt=tilta+vd*(t-ta)
                    retornar
               si no
                    tiltb=tilta+vd*(tb-ta)
                    tilt=tiltb+vd*(t-tb)-a*(t-tb)^2/2
                    retornar
               fin si
          fin si
     fin si
fin si
```

Figura 3.9: Algoritmo que proporciona el ángulo de inclinación vertical en función del tiempo transcurrido desde el comienzo del desplazamiento del dispositivo apuntador.

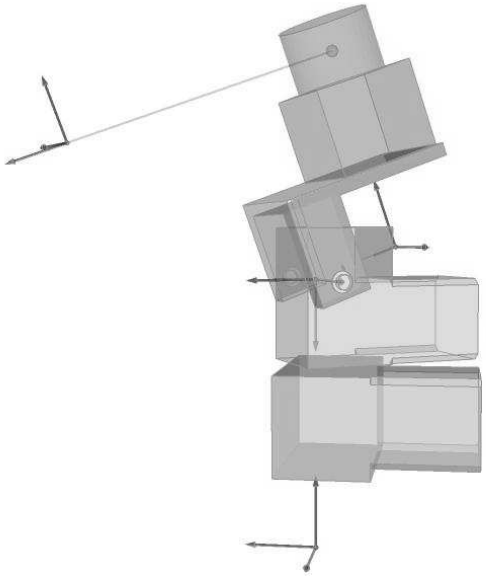


Figura 3.10: Representación de los ejes de referencia del sistema como cadena cinemática

Figura 3.11: Medidas del dispositivo apuntador

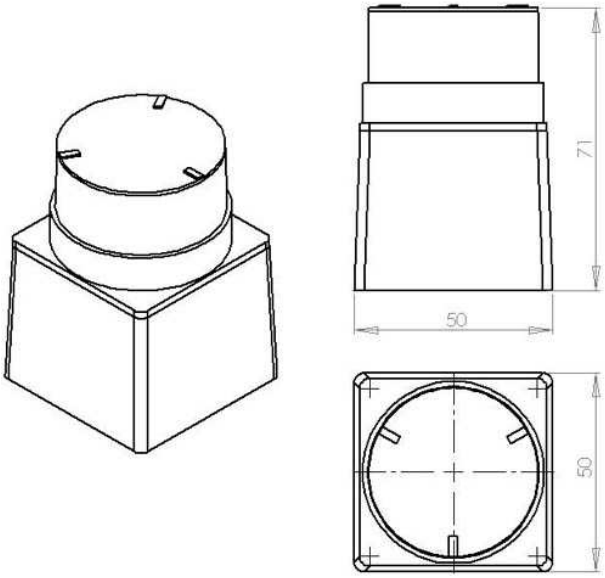


Figura 3.12: Medidas del sensor láser

Capítulo 4

Estudio de los marcos de representación 3D

En robótica se necesitan buenas representaciones del entorno para poder planificar y ejecutar desplazamientos de forma adecuada. Estas representaciones se recogen en lo que se denominan mapas del entorno. El entorno posee múltiples propiedades con distinto nivel semántico que, según el problema que vaya a resolver el robot, se deberán incluir en último término en el mapa: obstáculos, superficies transitables, color, texturas, etc. Una de las utilidades más frecuentes de los mapas es permitir la navegación autónoma a través del entorno. Los mapas pueden ser proporcionados a priori desde un ente externo (humano, fuente de geolocalización, etc). Sin embargo, en aras de la autonomía del robot, es deseable la construcción automática de tales mapas por parte de los propios robots.

Los robots están típicamente dotados de sensores que podrán ser usados para construir el mapa del entorno. Sin embargo la información proporcionada por los sensores difícilmente puede ser usada directamente. Los sensores tienen ruido e incertidumbre inherentes al método de adquisición de información y dan información parcial al no poder percibir todo el entorno. Además, los sensores poseen un alcance limitado y no tienen memoria. Por ello es necesario crear una representación interna que filtre el ruido. Esta representación interna permitirá integrar la información parcial que vaya proporcionando el sensor a lo largo del tiempo y desde diferentes localizaciones y servirá como mapa coherente del entorno.

Existen distintos tipos de mapas. Una primera clasificación divide a los mapas entre métricos y topológicos. Los mapas métricos extraen la propiedades geométricas del entorno mientras que los topológicos describen cuál es la conectividad existente entre los diferentes lugares. Entre los primeros métodos, encontramos los mapas de malla de ocupación que, mediante mallas de grano fino, representan el espacio ocupado y el libre del entorno. Los mapas topológicos representan el entorno como una lista de sitios significativos que se conectan mediante arcos. Los arcos se etiquetan con información acerca de cómo navegar desde un lugar a otro.

En los últimos tiempos [14], las técnicas probabilísticas para la construcción de mapas están

siendo dominantes. Existe un potente marco estadístico para resolver simultáneamente el problema de la construcción de los mapas y el de localización del robot. Dentro de los mapas probabilísticos encontramos distintas aproximaciones como se recoge en [14]. Una primera usa filtros de Kalman para estimar el mapa y la localización. Los mapas resultantes de este método describen la localización de ciertas marcas o de características significativas del entorno. Una familia alternativa de algoritmos están basados en el algoritmo de Dempster de maximización de la expectativa. Esta solución se enfoca específicamente al problema de correspondencia, el cual es el problema de determinar si la medida del sensor registrada en diferentes momentos corresponden a la misma entidad física del mundo real. Una tercera familia de técnicas probabilísticas buscan identificar objetos en el entorno, que pueden corresponder a techos, paredes, puertas abiertas o cerradas, mobiliario u otros objetos que incluso se mueven.

4.1. Mapas de rejilla de ocupación

Los mapas de rejilla de ocupación fueron desarrollados por Elfes y Moravec a mitad de los años ochenta del pasado siglo [15] y han disfrutado de mucha popularidad. Este algoritmo se ha usado en numerosos robots autónomos, en muchos casos en combinación con algún otro método probabilístico.

Los mapas de rejilla de ocupación se enfocan en el problema de generar un mapa métrico consistente a partir de datos ruidosos o incompletos obtenidos de uno o varios sensores. En ocasiones es difícil determinar si un lugar del entorno está ocupado o no debido a ambigüedades en los datos aportados por el sensor. Las aplicaciones más estudiadas de aplicación de estos mapas son robots dotados de «sensores de rango», como sónar y láser. Los mapas de rejilla de ocupación son útiles incluso para integrar información de varios sensores aún siendo estos de distinta naturaleza. Estos sensores se caracterizan por el ruido que poseen. Además, los sonares cubren una región del espacio cónica, y, a partir de una única medición del sónar, es imposible decir en qué parte del cono se encuentra el objeto. El sónar, por otra parte, es sensible a las condiciones de reflexión del haz ultrasónico con la superficie del objeto, al ángulo de incidencia del frente de onda y a las propiedades reflectivas de la superficie del objeto (problemas de absorción y de dispersión).

Los mapas de rejilla de ocupación representan el entorno mediante rejillas normalmente bidimensionales aunque también podrían ser tridimensionales. Cada rejilla se corresponde con un área del entorno que puede variar, según la implementación, de 5 a 50 cm. El algoritmo de generación de mapas de rejilla de ocupación es una versión de los filtros de Bayes [10]. Los filtros de Bayes se usan para calcular la probablidad a posterior de la ocupación de cada celda de la rejilla. Cada celda (x, y) tienen asociado un valor de ocupación. Este valor de ocupación es una variable binaria: celda ocupada o celda libre. El problema es, por tanto, calcular la probabilidad a posterior de un conjunto de variables binarias, cada una de las cuales es una simple probabilidad numérica.

En los mapas de rejilla de ocupación cada celda puede estar únicamente en uno de los siguientes estados: ocupada o vacía. El estado se estima a partir de las observaciones acumuladas obtenidas

por los sensores. El conocimiento que se posee en el instante t acerca del estado de la celda c_{ij} se obtiene a partir de la probabilidad de que la celda esté en alguno de los dos estados condicionada a las observaciones anteriores al momento t. Esto se expresa mediante la siguiente ecuación en la que $datos_{t-1}$ expresa los datos acumulados antes del instante t y obs_t la observación obtenida en el instante t.

$$p_{ocupada}(c_{ij}, t) = p(ocupada/obs_t, datos_{t-1})$$

4.2. Extensión eficiente de los mapas de elevación

Los mapas de elevación son estructuras populares para representar el entorno de un robot móvil que opera en entornos exteriores o en terrenos con superficies que no son planas. Los mapas de elevación almacenan en cada celda de una malla discreta la altura de la superficie del sitio correspondiente del entorno [3]. El uso de esta representación de $2\frac{1}{2}$ dimensiones es, no obstante, inadecuada cuando se usa para construir mapas con robots móviles que operen desde el suelo, debido a que objetos verticales o colgantes no se pueden representar adecuadamente. Tales objetos pueden provocar errores cuando se intenta emparejar dos mapas de elevación. En el artículo [3] proponen una solución que permite a un robot móvil tratar con objetos verticales o colgantes en un mapa de elevación.

Los mapas de elevación son representaciones del entorno de $2\frac{1}{2}$ dimensiones. Estos mapas poseen una malla de dos dimensiones y en cada celda de la malla almacenan una estimación de la altura del terreno en el punto correspondiente. Se parte de la suposición de que se conoce el ángulo de inclinación lateral(roll) y longitudinal (tilt) del sistema y de esta forma reflejar correctamente la inclinación real del terreno.

Se debe tener en cuenta, cuando se adapta la información correspondiente a una celda, que la incertidumbre en la medida aumenta con la distancia. En [3] se aplica un filtro de Kalman para estimar los parámetros $\mu_{1:t}$, elevación del terreno sobre una celda, y $\sigma_{1:t}$, su desviación estándar. Una nueva medida z_t con una desviación estándar σ_t en el instante t se incorpora con las siguientes ecuaciones:

$$\mu_{1:t} = \frac{\sigma_t^2 \mu_{1:t-1} + \sigma_{1:t-1}^2 z_t}{\sigma_{1:t-1}^2 + \sigma_t^2} \tag{4.1}$$

$$\sigma_{1:t}^2 = \frac{\sigma_{1:t-1}^2 \sigma_t^2}{\sigma_{1:t-1}^2 + \sigma_t^2} \tag{4.2}$$

La aplicación del filtro de Kalman permite incorporar la incertidumbre acerca de la medida. En [3] se aplica un modelo en el que la varianza en la altura de una medida se incrementa linealmente con la distancia del rayo correspondiente.

Por otra parte, es necesario identificar qué celdas del mapa de elevación corresponden a estructuras verticales y cuáles contienen huecos. Para determinar la clase de una celda, primero se

debe considerar la varianza de la altura de todas las medidas que caen en dicha celda. Si este valor sobrepasa un cierto umbral, es identificado como un punto que no ha sido observado desde arriba. Entonces se comprueba si el conjunto de puntos correspondiente a una celda contiene huecos que excedan la altura del robot. Esta comprobación se puede realizar manteniendo un conjunto de intervalos para cada celda de la malla, los cuales son calculados y adaptados cada vez que el sensor proporciona datos. En [3] se unen los intervalos con una separación menor de 10 cm. En consecuencia, es posible que la clasificación de una celda en particular necesite ser cambiada y pase de ser etiquetada como «celda vertical» o «celda que se observó desde arriba» a la etiqueta de «celda hueco». Además, puede suceder en caso de oclusión, que una celda cambie del estado «celda hueco» al estado «celda vertical». Cuando un hueco es identificado, se determina la elevación mínima que puede atravesar dicho conjunto de puntos.

4.3. Mapas de superficies multinivel

Como se comenta en la anterior sección, los mapas de elevación han sido una solución popular al problema de construir mapas con robots móviles. Sin embargo, incluso las extensiones de los mapas de elevación que permiten tratar con objetos verticales o colgantes sólo se pueden aplicar en entornos con superficies sencillas. Por ejemplo, un robot que use mapas de elevación extendidos no puede planificar al mismo tiempo un camino por debajo y sobre un puente.

En [4] se propone una extensión de los mapas de elevación hacia las múltiples superficies, los llamados mapas de superficies multinivel, que ofrece la oportunidad de modelar el entorno con más de un nivel transitable. El conocimiento acerca de las superficies horizontales es adecuado para dar soporte al análisis de transitabilidad y planificación de trayectos. Sin embargo, únicamente proporciona un débil soporte a la localización de vehículos o al registro de diferentes mapas. Modelar únicamente las superficies significa que las estructuras verticales, percibidas frecuentemente por vehículos anclados en el suelo, no pueden ser usadas para apoyar la localización y el registro. Para solventar este problema, en [4] se evita este problema representando los intervalos correspondientes a objetos verticales del entorno. Esta solución presenta la ventaja de poder almacenarse de forma compacta y al mismo tiempo puede ser usada para dar soporte al problema de la asociación de datos durante la alineación de mapas.

Un mapa de superficies multinivel consiste en una rejilla de dos dimensiones de tamaño variable en la que cada celda $c_{i,j}$ almacena una lista de parches de superficie $p_{ij}^1, \cdots, p_{ij}^k$. Un parche de superficie se representa como la media y la varianza de la alturas medidas en la posición de la celda $c_{i,j}$ en el mapa y refleja la posibilidad de atravesar el entorno 3D a la altura dada por la media μ_{ij}^k, mientras que la incertidumbre de esta altura se representa mediante la varianza σ_{ij}^k. En [4] se asume que el error en la altura sigue una distribución gausiana por lo que usan indistintamente «Gausiano en una celda» y «parche en una celda».

Además de la media y la varianza, se almacena un valor de profundidad en cada parche de superficie. Este valor de superficie refleja el hecho de que un parche de superficie puede estar en

Figura 4.1: Ejemplo de entorno con múltiples superficies: el suelo el asiento de las sillas y la mesa

la cima de un objeto vertical, como pudiera ser un edificio, un puente o una rampa. En esos casos, la profundidad se define como la diferencia de la altura h_{ij}^k de la superficie y la altura $h_{ij}'^k$ de la medida más baja perteneciente al objeto vertical. Un objeto plano, como puede ser el suelo, tendrá profundidad 0.

Se puede generar un mapa de superficies multinivel de dos formas distintas, a partir de un conjunto de mediciones 3D, como puede ser una nube de puntos con sus varianzas, o bien uniendo dos mapas de superficies multinivel en uno solo. Ambas formas son equivalentes, esto es, si se crea el mapa m_1 a partir de la nube de puntos C_1 y el mapa m_2 a partir de la nube C_2, entonces el mapa m_3 que resulte de la unión de los dos mapas anteriores es idéntico al mapa generado a partir de la nube $C_3 = C_1 \cup C_2$. Dado una nube de puntos C con varianzas $\sigma_1, \cdots, \sigma_n$ el mapa correspondiente se crea de la siguiente forma:

- Cada celda del mapa $c_{i,j}$ recogerá todos los puntos $p = (x, y, z)$ tal que $s \cdot i \leq x \leq s \cdot (i + 1)$ y $s \cdot j \leq y \leq s \cdot (j + 1)$ donde s denota el tamaño del lado de una celda del mapa.

- En cada celda se calcula un conjunto de intervalos en altura a partir de la altura de los puntos. Si dos alturas distan menos que un determinado valor de «hueco», de tamaño γ, se las considera pertenecientes al mismo intervalo. Este tamaño de «hueco» se elige en [4] de forma que el sistema completo (robot más sensores) pueda pasar a través del hueco.

- Los intervalos se clasifican como estructuras horizontales o verticales. La diferencia entre ambas estructuras radica en la altura del intervalo. Si esta altura es mayor que un valor de «grosor» τ se considera al intervalo vertical, en caso contrario se considera horizontal.

- En los intervalos verticales se almacenan la media y la varianza de la medida más alta junto con la altura. El valor de la altura se usará para emparejar juntos dos mapas.

- En los intervalos horizontales se calcula la media y la varianza de todas las medidas. Para

```
añadir_nueva_medida
entrada:
px, py, pz - coordenadas espaciales de la medida
σ - varianza de la medida
s - longitud de la celda
i := entero (px/s)
j := entero(py/s)
G := gausiano_más_próximo(cij,pz)
si G es nulo entonces
          añadir_gausiano(cij,px,py,pz,σ)
          retornar
si no
          si distancia(G.z,pz)<3·σ entonces
                    actualizar_gausiano(G,px,py,pz,σ)
                    retornar
          si no
                    si gausiano_coincidente(cij,pz) es nulo
entonces

          añadir_nuevo_gausiano(cij,px,py,pz,σ)
                              retornar
                    fin si
          fin si
fin si
```

Figura 4.2: Algoritmo para añadir una nueva medida a un mapa de superficies multinivel.

hacer esto se utiliza la regla de adaptación de Kalman aplicada a todas las medidas. En estos intervalos la profundidad se establece como cero.

Una vez calculadas las medias, varianzas y alturas de cada parche de superficie, la nube de puntos se borra y todo cálculo posterior se lleva a cabo únicamente con los datos del mapa. Esto reducirá notablemente la cantidad de memoria necesaria.

El procedimiento que se sigue cuando se añade una nueva medida se muestra en la figura **??** expresado en pseudocódigo. Cuando se añade una nueva medida $z = (p, \sigma)$ a un mapa de superficies multinivel, lo primero que se necesita saber es si la medida pertenece a algún objeto que ya está representado en el mapa o si corresponde a algún nuevo objeto. Con este fin, primero se determina cuál es la celda c_{ij} en la cual la nueva medida cae en función de las coordenadas (p_x, p_y). Después se busca en c_{ij} el Gausiano cuya media esté más cerca de la altura de z. Si el Gausiano está lo suficientemente cerca, se adapta el Gausiano con esta nueva medida usando de nuevo la regla de adaptación de Kalman. En [4] se establece que un Gausiano está suficientemente próximo a la medida z si la altura de z dista menos de 3σ del Gausiano. Vemos las tres situaciones

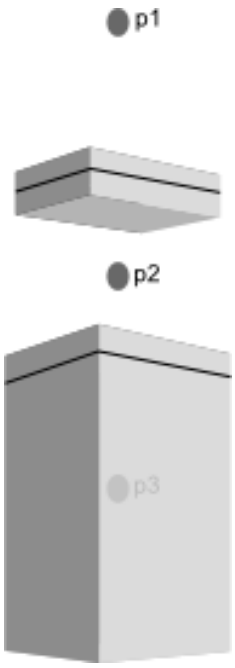


Figura 4.3: Distintas situaciones a la hora de insertar una nueva medida en un mapa multinivel. El punto p1 está alejado de cualquier gausiano por lo que se creará uno nuevo para él. El punto p2 se encuentra próximo a una estructura horizontal, por lo que procederá a actualizar la media y la varianza de esta mediante filtros de Kalman. Finalmente, el punto p3 está en el interior de una estructura vertical por lo que no es necesario realizar ninguna actualización.

posibles en la figura 4.3

Si z está lejos del Gausiano más cercano, puede aún pertenecer a un objeto vertical. Esto se puede determinar comprobando si z está dentro de la altura ocupada por uno de los Gausianos en cuyo caso z es descartado. Finalmente, si z no perteneciera a ningún Gausiano se crea con él un nuevo Gausiano.

Además de la distinción entre estructuras horizontales y verticales, las estructuras horizontales también pueden ser clasificadas como transitables o no transitables. El concepto refleja el hecho de que ciertas estructuras horizontales no pueden ser alcanzadas por el robot debido a que se encuentran aisladas. En [4] se determina que un parche de superficie sólo puede ser transitable si al menos 5 de las 8 celdas vecinas existen y si la diferencia de alturas entre el parche y cada uno de sus vecinos es menor de 10cm.

4.4. Detección de formas en nubes de puntos

La digitalización de geometrías es una tarea común en distintas áreas:

- Construcción de representaciones digitalizadas de objetos.

- Estudios de propiedades de conjuntos digitales.
- Transformaciones de representaciones digitales de objetos.
- Reconstrucción de objetos reales, o de sus propiedades, a partir de imágenes digitales.
- Estudio de curvas y superficies digitales.
- Diseño de algoritmos de rastreo para objetos digitales.

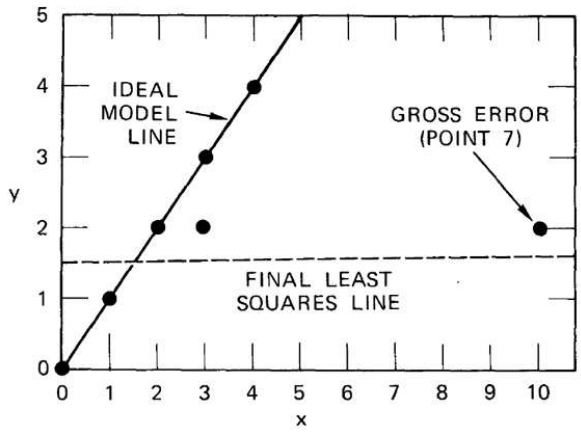
La digitalización se realiza mediante sensores de distintos tipos como sensores láser, sonares, etc. Estos sensores habitualmente retornan un vector de coordenadas tridimensionales que se incorporan a una nube de puntos. Una buena digitalización requiere obtener muchos puntos del objeto o entorno a tratar. Esto conlleva que exista una demanda creciente de abstracciones sobre los datos. En aras de poder hacer un uso más efectivo de la información obtenida, se busca enriquecer los datos con información semántica y con abstracciones de mayor nivel. La detección de formas en nubes de puntos, por tanto, permite pasar de los datos puros a interpretaciones de la escena. Estas interpretaciones dotarán al sistema de conocimiento acerca, por ejemplo, de los límites del recinto en que se encuentra, permitirán saber cuándo se pasa de un espacio a otro (se pasa de una habitación a la siguiente), harán posible separar un objeto del resto del entorno, etc.

En este proyecto se ha implementado una versión del algoritmo «Efficient RANSAC» desarrollada por Schnabel, Wahl y Klein [7]. Los anteriores autores han enfocado su trabajo en desarrollar un método eficiente de detectar formas dentro de una nube de puntos y han tomado como punto de partida el método RANSAC.

4.4.1. El algoritmo RANSAC

El algoritmo RANSAC (RANdom SAmple Consensus) es un método iterativo para estimar parámetros de un modelo matemático a partir de un conjunto de datos experimentales en los que existen valores atípicos (outliers). RANSAC es un algoritmo no determinista ya que únicamente genera un resultado razonable dentro de una cierta probabilidad. Esta probabilidad se incrementa con el número de iteraciones que se ejecuten. Este algoritmo fue publicado originalmente por Fischler y Bolles en 1981 [5]. Éstos desarrollaron un método para mejorar un problema de las técnicas clásicas de estimación de parámetros, como la de mínimos cuadrados. Estas técnicas clásicas optimizan la adecuación de una descripción funcional, un modelo, a todos los datos presentes. El problema surge porque esas técnicas no tienen mecanismos internos para detectar y rechazar datos que pueden distorsionar gravemente el ajuste del modelo a los datos (ver figura 4.4). Estos datos atípicos son relativamente frecuentes en los dispositivos de escaneado de objetos reales como sensores láser y sonares. Los métodos clásicos se basan en técnicas de promediado que confían en que la desviación máxima esperada de cualquier dato respecto al modelo es función directa del tamaño del conjunto de datos y, por tanto, independiente del tamaño de tal conjunto habrá siempre suficiente valores buenos para compensar cualquier desviación.

El algoritmo RANSAC, por su parte, puede producir un modelo que sea calculado únicamente a partir de los valores internos (inliers) basándose en que la probabilidad de seleccionar dicho tipo de



POINT	x	y
1	0	0
2	1	1
3	2	2
4	3	2
5	3	3
6	4	4
7	10	2

Figura 4.4: Problema del método de mínimos cuadrados cuando trata con datos erróneos

valores es suficientemente alta. Evidentemente, al ser una cuestión probabilística, no hay garantía de que esto sea así, y se debe seleccionar con cuidado cierto número de parámetros del algoritmo para mantener la probabilidad suficientemente alta.

El algoritmo RANSAC toma como entrada un conjunto de valores observados, un modelo parametrizado que, a priori, se puede ajustar a las observaciones y un conjunto de parámetros de confianza. RANSAC, para alcanzar su meta, va seleccionando iterativamente un subconjunto de los datos originales. Se parte de la hipótesis de que dichos datos son internos y posteriormente se verifica la hipótesis. El contraste se realiza de la siguiente forma:

1. Se ajustan los parámetros del modelo al subconjunto de datos seleccionados.

2. Se comprueban el resto de los datos frente al modelo generado en el punto anterior. Si un dato se ajusta al modelo así obtenido, se le considera considera también un dato interno.

3. Se considera que el modelo estimado es razonablemente bueno si un número suficiente de puntos se han clasificado como hipotéticos datos internos.

4. El modelo es ahora reestimado a partir de todos los datos hipotéticamente internos. Esto se lleva a cabo ya que el modelo fue estimado sólo a partir de los datos iniciales.

5. Finalmente, el modelo se evalúa para estimar el error de los datos internos frente al modelo.

El procedimiento se repite un número determinado de veces. En cada iteración se obtiene bien un modelo que es rechazado por haber clasificado muy pocos puntos como internos o bien un modelo refinado junto con una medida del correspondiente error. En este último caso, seleccionamos el modelo si el error es menor que el del mejor modelo que teníamos hasta el momento.

El algoritmo, que se muestra en la figura 4.5, posee tres parámetros que pueden ser ajustados:

```
entrada:
datos - un conjunto de observaciones
model - un modelo que se puede ajustar a los datos
n - el número mínimo de datos necesarios para ajustar el modelo
k - el nímero máximo de iteraciones permitidas en el algoritmo
t - valor umbral para determinar cuándo un dato se ajusta al modelo
d - el número de datos cercanos necesario para evaluar que un modelo se
ajusta bien a los datos

salida:
mejor_modelo - parámetros del modelo que mejor se ajusta a los datos (o nulo
si no se encuentra ninguno)
mejor_conjunto_consenso - puntos a partir de los cuales se ha estimado el
modelo
mejor_error - el error del mejor_modelo respecto a sus datos

iteraciones := 0
mejor_modelo := nulo
best_conjunto_consenso := nulo
mejor_error := infinito
mientras iteraciones < k hacer
        posibles_internos := n valores seleccionados aleatoriamente de los
        datos
        posible_modelo := parámetros del modelo ajustados a posibles_internos
        conjunto_consenso := posibles_internos
        para cada punto de los datos que no esté en posibles_internos hacer
                si el punto ajusta a posible_modelo con un error menor que t
                entonces
                                añadir el punto al conjunto_consenso
        si el número de elementos en el conjunto_consenso > d entonces
        (esto implica que se ha encontrado un buen modelo, ahora se comprueba
        cuán bueno es)
                mejor_modelo := parámetros del modelo ajustados a todos los
                puntos del conjunto_consenso
                este_error := una medida de lo bien que se ajusta el modelo a
                esos puntos
                si este_error < mejor_error entonces
                (se ha encontrado un modelo que es mejor que los anteriores)
                        mejor_modelo := mejor_modelo
                        mejor_conjunto_consenso := conjunto_consenso
                        mejor_error := este_error
        incrementar iteraciones
devolver mejor_modelo, best_conjunto_consenso, mejor_error
```

Figura 4.5: Algoritmo RANSAC

1. La tolerancia a errores usada para determinar si un punto es compatible o no con un modelo. La desviación de un dato respecto a un modelo es función del error asociado con el dato y el error asociado con el modelo. Generalmente la tolerancia a errores se determina en función del problema.

2. El número de subconjuntos a probar. Esto se traducirá en el número máximo de iteraciones a ejecutar. La decisión de parar de seleccionar nuevos subconjutnos se puede estimar basándonos en el número esperado de pruebas, k, necesarias para seleccionar un subconjunto de n puntos internos. Siendo w la probabilidad de que un punto seleccionado esté dentro de la tolerancia al error del modelo, el número k de iteraciones máximas será:

$$k = \frac{log(1-z)}{log(1-w^n)}$$

Figura 4.6: Problema del método de mínimos cuadrados cuando trata con datos erróneos

3. El mínimo número de puntos compatibles necesarios para implicar que se ha encontrado un modelo correcto. Este parámetro debe ser elegido de forma que satisfaga dos objetivos: que se ha encontrado el modelo correcto y que se ha encontrado un número suficiente de puntos mútuamente consistentes para satisfacer las necesidades del procedimiento de reajuste final.

4.4.2. RANSAC eficiente (eRANSAC)

Como hemos comentado anteriormente, en este proyecto hemos implementado una versión del algoritmo eRANSAC [7]. Este algoritmo tiene como objetivo detectar automáticamente formas (planos, esferas, cilindros, etc.) en una nube de puntos. El algoritmo descompone la nube de puntos en una colección de formas concisas y un conjunto de puntos sobrantes. Cada forma detectada representa un conjunto de puntos de la nube original.

Este método es adecuado en situaciones donde se generan automáticamente datos geométricos. Está basado en el algoritmo RANSAC anteriormente expuesto, el cual posee las siguientes buenas propiedades:

- Es conceptualmente simple, lo que lo hace fácilmente extensible y sencillo de implementar.
- Es muy general, permitiéndole ser aplicado en un gran número de problemas.
- Puede tratar de forma robusta con datos que contengan más de un 50 % de valores atípicos.

Sin embargo, su mayor deficiencia es su gran demanda computacional si no se aplica algún tipo de optimización. El método eRANSAC busca precisamente aplicar el método de una forma eficaz computacionalmente. Expondremos en los siguientes párrafos el funcionamiento de este algoritmo.

Partiendo de una nube de puntos, cada uno con una vector normal asociado, la salida del algoritmo proporciona un conjunto de formas primitivas con sus correspondientes conjuntos de puntos disjuntos y un conjunto de puntos sobrantes. La estructura global del problema expresada en pseudocódigo se muestra en la figura 4.7

En cada iteración, se busca la primitiva que alcance una mayor puntuación usando el algoritmo RANSAC. Las nuevas formas candidatas será generadas tomando muestras de tamaño mínimo de

```
Ψ←0  (formas extraídas)
C←0  (formas candidatas)
repetir
     C←C U nuevosCandidatos()
     m←mejorCandidato(C)
     si P(|m|,|C|)>p_t entonces
          P ←P/P_m  (elimina los puntos)
          Ψ← Ψ U m
          C ←C/C_m  (elimina los candidatos
          inválidos)
     fin si
hasta P(σ,|C|)> p_t
devuelve Ψ
```

Figura 4.7: Algoritmo RANSAC eficiente

la nube de puntos. En el eRANSAC se presenta una nueva forma de muestrear. Para cada muestra se generan, siempre que sea posible, candidatas de todas las formas consideradas. Posteriormente, se calcula cuál de las formas candidatas generadas alcanza una mayor puntuación. Para ello, se usa un esquema de evaluación poco costoso. La mejor candidata se acepta únicamente si, dado el tamaño en número de puntos de la candidata $|m|$, y el número de candidatas generadas, $|C|$, la probabilidad $P(|m|, |C|)$ de que ninguna candidata mejor haya sido pasada por alto durante el proceso de muestreo sea lo suficientemente alto. Si una forma candidata es aceptada, los puntos correspondientes a esta forma, P_m, son eliminados de la nube y las candidatas C_m con puntos en P_m son borradas de C. El algoritmo termina en cuanto la probabilidad $P(\tau, |C|)$, dado un tamaño τ definido por el usuario, sea lo suficientemente alta.

La función que evalúa la puntuación de una forma candidata cuenta el número de puntos en la nube compatibles con ella. La función tiene dos parámetros libres: ϵ, que especifica la distancia máxima desde un punto compatible a la forma, y α, que especifica la máxima desviación de la normal de un punto respecto a la normal a la forma candidata. Las formas que se estudian en [7] son planos, esferas, cilindros, toros y conos. Se necesitan entre tres y siete parámetros para caracterizar cada una de estas formas. Cada punto de la nube determina un único parámetro de la forma. Con el objetivo de reducir el número de puntos necesarios, se calcula una normal aproximada, n_i, para cada punto. De esta forma, con sólo uno o dos puntos es posible estimar una de las formas candidatas. En cualquier caso, siempre es conveniente usar algún punto adicional, ya que el parámetro extra puede ser usado para verificar una candidata y de esta forma eliminar la necesidad de evaluar muchas formas que tendrían una baja puntuación.

La determinación de las normales en cada punto servirá, como hemos visto, para reducir el

número de parámetros necesarios para caracterizar cada forma candidata. Esto además conlleva que el número máximo de iteraciones necesario también disminuya. Sin embargo, calcular estas normales no es un problema trivial. En [16] se expone un método para calcular el plano tangente a cada punto de la nube. El plano tangente $T_p(x_i)$ asociado al punto x_i se representa mediante un punto o_i llamado el centro, junto con un vector unitario normal $\hat{n_i}$. El centro y la normal se determinan a partir de los k puntos de la nube más cercanos a x_i. Este conjunto es denominado la k-vecindad, $Nbhd(x_i)$, de x_i. El centro y la normal son calculados de forma que el plano resultante sea el que mejor se ajuste a $Nbhd(x_i)$ por mínimos cuadrados. Esto es, el centro o_i es el centroide de $Nbhd(x_i)$, y la normal $\hat{n_i}$ es determinada usando el análisis de componentes principales (ACP).

El ACP es una técnica estadística la cual es aplicada en diversos campos como reconocimiento de caras, compresión de imágenes y reconocimiento de patrones en datos de gran tamaño. Estas técnicas fueron inicialmente desarrolladas por Pearson a finales del siglo XIX y posteriormente fueron estudiadas por Hotelling en los años 30 del siglo XX. Sin embargo, hasta la aparición de los ordenadores no se empezaron a popularizar. Podemos encontrar una guía al ACP en [17].

Para calcular la normal a cada punto $\hat{n_i}$ se forma la matriz de covarianzas de $Nbhd(x_i)$:

$$CV = \sum_{y \in Nbhd(x_i)} (y - o_i) \otimes (y - o_i)$$

Siendo $\lambda_i^1 \leq \lambda_i^2 \leq \lambda_i^3$ los autovalores de la matriz de covarianza asociados respectivamente con los autovectores $\hat{v}_i^1, \hat{v}_i^2, \hat{v}_i^3$, elegimos como normal $\hat{n_i}$ al vector $\hat{v}_i^3$ o al $-\hat{v}_i^3$. La elección de un vector u otro como normal determina la orientación del plano tangente, y debe ser hecha de forma que el plano esté consistentemente orientado.

Las distintas formas candidatas que se buscan en [7] se estiman de la siguiente forma:

Plano: Un conjunto mínimo para determinar un plano serán tres puntos cualesquiera p_1, p_2, p_3 si no se tiene en cuenta las normales. Si se considera las normales, un único punto y su normal bastarían para determinar un plano pero conviene seleccionar un mayor número de puntos para no generar formas candidatas con baja puntuación.

Esfera: Una esfera queda completamente definida con dos puntos, p_1 y p_2, con sus correspondientes normales. El punto medio del segmento más corto entre las dos líneas formadas por los puntos y sus normales definen el centro de la esfera c. El radio de la esfera se obtiene como $r = \frac{||p_1 - c|| + ||p_2 - c||}{2}$. La esfera se acepta como forma candidata sólo si hay tres puntos dentro de la distancia ϵ de la esfera y sus normales no se desvían más de α grados.

Cilindro: Se genera un cilindro a partir de dos puntos con sus normales. Primero se establece la dirección del eje con $a = n_1 \times n_2$. Se proyecta entonces las dos líneas paramétricas $p_1 + t \cdot n_1$ y $p_2 + t \cdot n_2$ a lo largo del eje sobre el plano $a \cdot x = 0$ y tomamos su intersección como el centro c. Establecemos como radio la distancia entre c y p_1 en dicho plano. De nuevo, el cilindro ser verifica aplicando el umbral ϵ y α para la distancia y la desviación de la normal de las muestras.

Cono: Aunque el cono puede ser también definido a partir de dos puntos con sus correspondientes normales, por simplicidad se usarán tres puntos y sus normales. Para determinar el vértice, se intersectan los tres planos definidos por los puntos y los pares de normales. La dirección del eje a es dada por la normal del plano definido por los tres puntos $\left\{c + \frac{p_1 - c}{||p_1 - c||}, \cdots, c + \frac{p_3 - c}{||p_3 - c||}\right\}$. El ángulo de apertura del cono, ω, es dado como $\omega = \frac{\sum_i arccos((p_i - c) \cdot a)}{3}$. Finalmente, como en las anteriores formas, el cono se verifica antes de convertirlo en una forma candidata.

Toro: Como en el cono, se usará un punto más del teóricamente necesario para facilitar los cálculos necesarios para la estimación, en este caso, cuatro puntos y sus normales. El eje rotacional del toro se encuentra como una de las hasta dos líneas que intersectan las cuatro líneas $p_i + \lambda \cdot n_i$. Para elegir entre los dos ejes posibles, se estima un toro para cada una de las dos posibilidades y se selecciona aquel que genera menos error respecto a los cuatro puntos. El radio menor se encuentra reuniendo los puntos en un plano que es rotado alrededor del eje. Entonces se calcula un círculo usando tres puntos de este plano. El radio mayor se obtiene como la distancia del círculo central al eje.

La complejidad del eRANSAC está dominada por dos factores principales: el mínimo número de conjuntos que son generados y el coste de evaluar la puntuación para cada forma candidata. Debido a que se desea extraer la forma que alcanza la puntuación más alta posible, el número de candidatas que se tienen que considerar está gobernado por la probabilidad de que la mejor forma posible sea detectada de hecho, esto es, de que un conjunto mínimo sea seleccionado el cual defina a esta forma.

Para estimar los factores anteriores consideremos una nube de puntos P de tamaño N y una forma ψ que consta de n puntos. Sea k el tamaño del conjunto mínimo necesario para definir una forma candidata. Debido a que cualesquiera k puntos de la forma llevarían a generar también una forma candidata, entonces la probabilidad de detectar ψ en un único paso es:

$$P(n) = \binom{n}{k} / \binom{N}{k} \approx \left(\frac{n}{N}\right)^k$$

La probabilidad de tener éxito en una detección, $P(n, s)$, después de que s candidatas hayan sido generadas es igual al complementario de s fallos consecutivos:

$$P(n, s) = 1 - (1 - P(n))^s$$

Así, de la ecuación anterior, resolviendo s obtenemos el número de candidatos T necesario para detectar formas de un tamaño n con una probabilidad $P(n, T) \geq p_t$:

$$T \geq \frac{ln(1 - p_t)}{ln(1 - P(n))}$$

Sea C el coste de la función de evaluación que puntuará a cada candidata. Entonces, la complejidad asintótica del método es $O(TC) = O(\frac{1}{P(n)}C)$. Se puede observar que la complejidad del tiempo de ejecución está ligado a la proporción exitosa de encontrar buenos conjuntos de muestra para generar las candidatas. Debido a que una forma es un fenómeno local, no se encuentra

«dispersa» por toda la nube, la probabilidad a priori de que dos puntos pertenezcan a la misma forma aumenta a medida que es menor la distancia entre los puntos. La probablidad de seleccionar un conjunto de puntos internos aumenta significativamente mediante un muestreo no uniforme basado en localidad. Así, a partir de una bola de un radio determinado alrededor de una muestra tomada al azar se seleccionan la muestras restantes para obtener un conjunto mínimo. Esto requiere seleccionar por adelantado un radio que se establecerá en función del conocimiento que se tenga de la densidad de datos atípicos en la nube.

Anteriormente se proporcionó la fórmula para el número de candidatas necesarias para detectar una forma de tamaño n con una determinada probabilidad. No obstante, el tamaño n no se conoce por adelantado. Es más, si la candidata de tamaño más grande ya ha sido generada se debería poder detectar y extraer la forma antes de procesar otras candidatas precomputadas. Por otra parte, se deberían generar candidatas adicionales si aún no es seguro que se haya detectado la mejor candidata. Por esto, en vez de determinar el número de candidatas, se analizan repetidamente pequeños números t de candidatas adicionales y se considera la mejor ψ_m generada cada vez. Para minimizar la probabilidad de que una forma que no es realmente la de máxima puntuación sea extraída, se evalúa la probabilidad $P(|\psi_m|, s)$ con la que se habría encontrado otra forma del mismo tamaño que ψ_m. Cuando esta probabilidad es mayor que un determinado umbral p_t se concluye que la posibilidad de que se haya pasado por alto una candidata mayor es muy baja y por tanto se extrae ψ_m. El algoritmo termina en cuanto $P(\tau, s) > p_t$.

La función de puntuación σ_p se encarga de medir la calidad de una forma candidata. La función de puntuación usa los siguientes aspectos:

- El cálculo de la puntuación de una forma candidata usa el número de puntos que caen dentro de una banda de radio ϵ en torno a la forma.
- Sólo se contarán los puntos que, cayendo dentro de la banda, tengan una normal que no se desvía de la normal de la forma más de un ángulo α. De esta forma se asegura que los puntos siguen aproximadamente el patrón de curvatura de la forma candidata.
- De entre todos los puntos que superan las dos condiciones anteriores sólo se seleccionan los que forman el mayor conjunto conexo.

El algoritmo eRANSAC presenta una segunda mejora de rendimiento frente al RANSAC original. Esta mejora se centra en la función de evaluación. Inicialmente, la función debería examinar todos los puntos junto con sus normales para cada forma candidata. Esto tiene un coste alto en cuanto crece el número de datos en la nube de puntos. Debido a que sólo se está interesado en la candidata que obtiene la mayor puntuación, no es necesario usar la nube de puntos entera P para calcular $\sigma_P(\psi)$. Se reduce significativamente el número de puntos que tienen que ser considerados para evaluar $\sigma_P(\psi)$ dividiendo la nube de puntos en un conjunto de subconjuntos aleatorios $P = \{S_1 \cdots S_r\}$. Después de que una forma candidata haya sido generada y verificada con éxito, la candidata se evalúa contra el primer subconjunto S_1. Partiendo de la puntuación obtenida en el primer subconjunto $\sigma_{S_1}(\psi)$ se calcula $\hat{\sigma}_P(\psi)$, una estimación de $\sigma_P(\psi)$. Esta estimación se puede

obtener por extrapolación usando la inducción de la inferencia estadística.

Finalmente, cuando una forma candidata ψ se selecciona para su extracción se ejecuta un paso de reajuste antes de que la forma sea finalmente aceptada. Al igual que en el RANSAC original, se usa una aproximación basada en mínimos cuadrados usando todos los puntos incluidos finalmente en la forma. Esto disminuye el error geométrico de la forma candidata.

Capítulo 5

Diseño

5.1. Arquitectura del sistema

Como se comentó en 2.1, este proyecto ha seguido un modelo de capas en su diseño arquitectónico. El modelo aporta una descomposición del sistema la cual, en su concepción de un conjunto de módulos que suministran una serie de servicios al módulo siguiente, se adapta al proyecto aquí presentado.

Esquemáticamente, el sistema desarrollado en el presente proyecto, puede verse en la figura 5.1. Se inicia el proceso cuando se ordena al sistema hardware integrado, láser y dispositivo apuntador, realizar una medición de los objetos en su entorno. El dispositivo apuntador inicia un barrido durante el cual el sensor láser, de forma sincronizada, va suministrando en forma de array las distancias de los objetos que rodean al sensor. Cada array de datos es procesado para convertir las medidas en coordenadas tridimensionales del mundo. Para poder realizar esta conversión se necesita conocer la odometría del robot. La siguiente estación en el sistema consiste en modificar el mapa 3D multisuperficie a partir de las coordenadas espaciales. Cada localización representa un impacto del haz láser con algún objeto del mundo. Estos impactos son procesadas por el método de integración de impactos que o bien creará una nueva estructura en el mapa o bien integrará el impacto en alguna ya creada. Por último, tenemos dos procesos que operan sobre el mapa multisuperficie: el método de obtención de planos y el método de proyección de mapa multisuperficies en un mapa de obstáculos 2D. El algoritmo que extrae planos es una modificación del algoritmo efficient-Ransac [7] que en vez de operar sobre una nube de puntos, trabaja sobre Gausianos. La proyección de la representación 3D sobre un mapa 2D proporciona la localización de obstáculos a nivel del suelo que va a permitir navegar al robot. Se necesita conocer el tamaño del robot para poder decidir si un objeto del mundo, por ejemplo el tablero de una mesa, constituye un obstáculo a la navegación o no.

Se puede ver el sistema desde un punto de vista de flujo de datos como una cadena de transformación de información. Se muestra esta cadena en la figura 5.2. Partiendo de las medidas proporcionadas por el sensor láser, estas medidas son convertidas en coordenadas espaciales y posteriormente en estructuras del mapa de superficies multinivel. A partir del mapa 3D, un proceso extrae planos y otro proyecta en un mapa 2D los distintos obstáculos para la navegación.

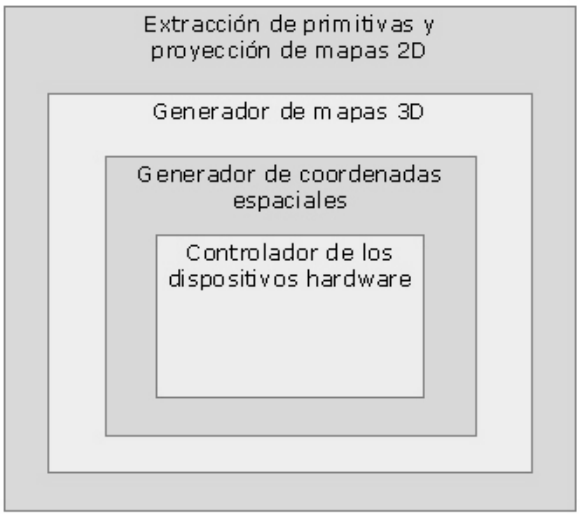


Figura 5.1: Arquitectura del software

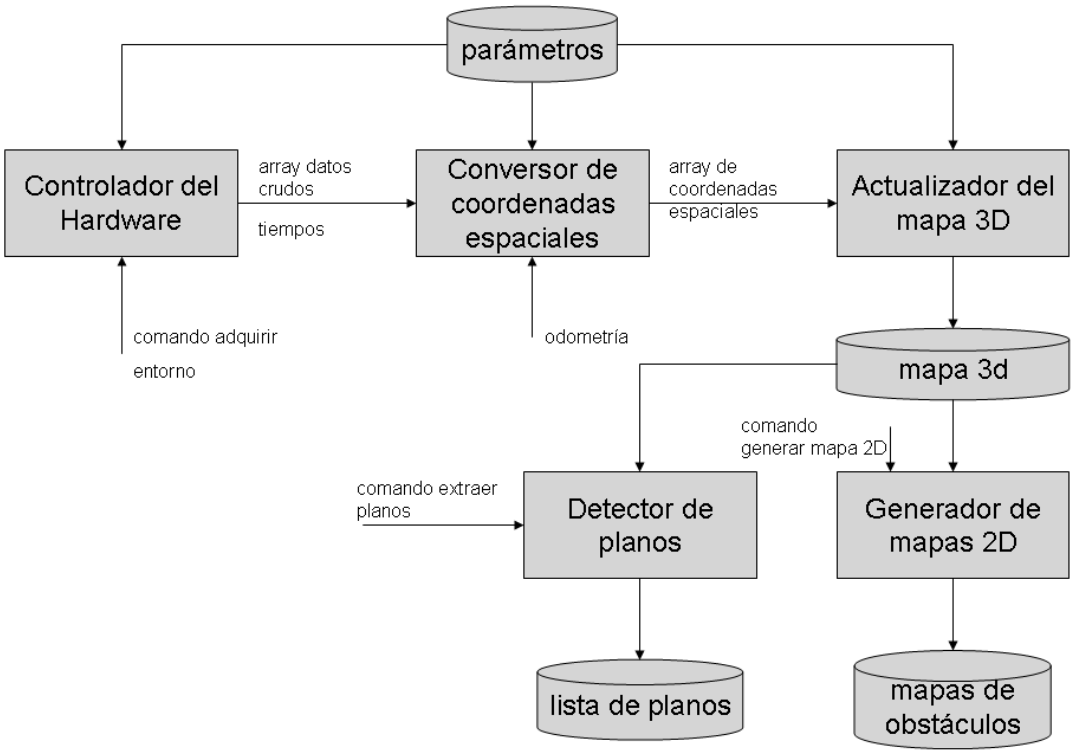


Figura 5.2: Flujo de datos del sistema

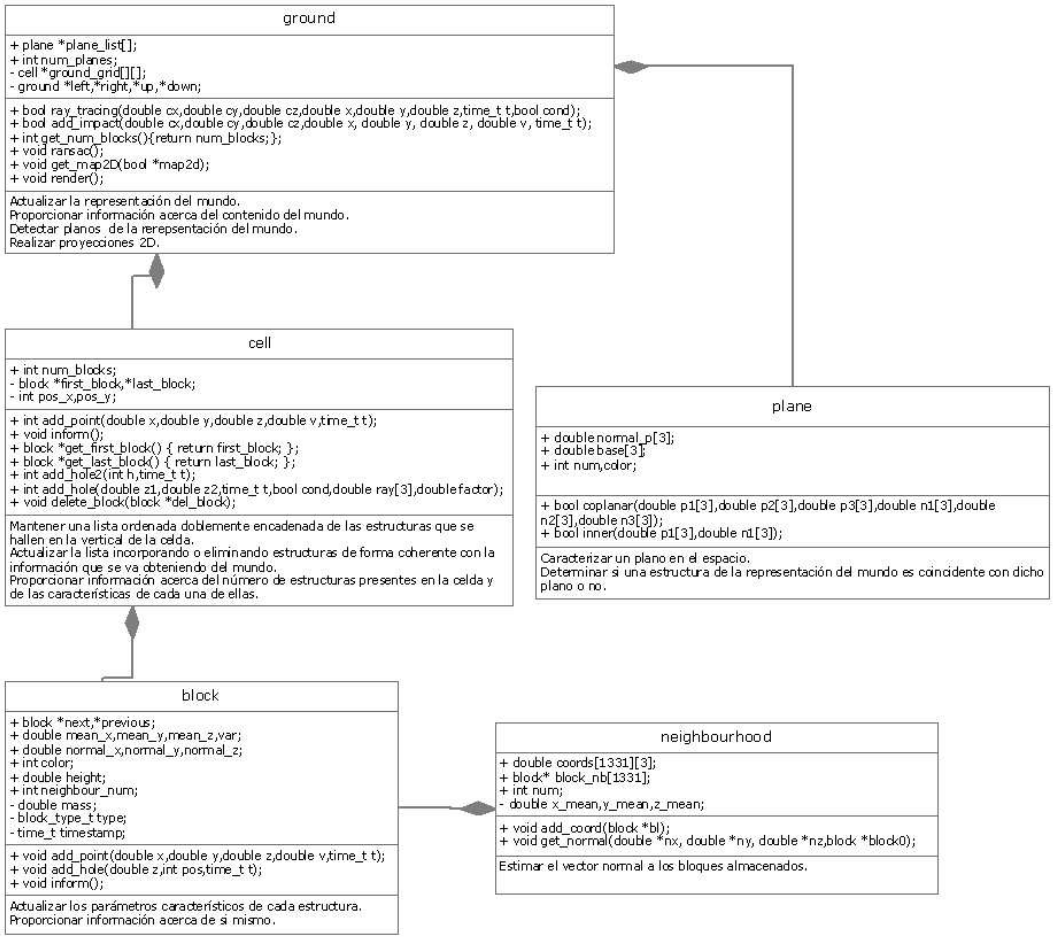


Figura 5.3: Diagrama de clases del sistema implementado

5.2. Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Describimos en el resto de la sección las clases desarrolladas en este proyecto. Vemos su esquema en la figura 5.3.

5.2.1. Clase ground

Esta clase representa el mundo conocido por le sistema. El modelo de representación elegido en este proyecto es un mapa multinivel. En tales mapas, el suelo se ha digitalizado en celdas o

casillas de una ancho y largo fijos. La clase ground mantiene una matriz de objetos cell. Serán responsabilidades de la clase ground:

- Actualizar la representación del mundo coherentemente con la información suministrada por el SLDA. Esto consistirá en añadir los nuevos objetos que se vayan detectando y eliminar los que vayan desapareciendo.
- Proporcionar información acerca del contenido del mundo. Esta información será tanto en modo gráfico como en formato texto.
- Detectar estructuras planas (suelos, paredes, techos, mesas, etc.) de la representación del mundo.
- Realizar proyecciones 2D.

5.2.2. Clase cell

La clase cell representa cada uno de los cuadros en que se ha dividido el suelo. Las responsabilidades de la clase cell son:

- Mantener una lista ordenada doblemente encadenada de las estructuras que se hallen en la vertical de la celda.
- Actualizar la lista incorporando o eliminando estructuras de forma coherente con la información que se va obteniendo del mundo.
- Proporcionar información acerca del número de estructuras presentes en la celda y de las características de cada una de ellas.

5.2.3. Clase block

Esta clase representa a las estructuras que se encuentren en la vertical de una celda. Podemos definir estructura como una cantidad de materia los suficientemente conexa y con la misma proyección sobre una determinada celda. El concepto de «suficientemente conexa» será determinado como un parámetro del sistema, como veremos en el siguiente capítulo. Las responsabilidades de la clase son:

- Actualizar los parámetros característicos de cada estructura (se explican más adelante) de forma coherente con la información obtenida.
- Proporcionar información acerca de si mismo.

5.2.4. Clase neighbourhood

La clase *neighbourhood* tiene un papel auxiliar en el sistema. Sirve para encapsular a los bloque vecinos a un bloque dado. Se entiende por vecinos de un bloque aquellos que no distan más de una cierta distancia D. Este conjunto de vecinos se usará para estimar el vector normal a un bloque o para estimar si un bloque es transitable o no. Es responsabilidad de la clase:

- Estimar el vector normal a los bloques almacenados.

5.2.5. Clase plane

Uno de los objetivos del proyecto (ver sección 1.2) es extraer información de la representación del mundo. Se implementa un algoritmo que detecta planos en el mundo. Estos planos serán de gran utilidad en interiores para caracterizar suelos, paredes, techos, etc. Cada plano es representado por un objeto de la clase plane. Las responsabilidades son:

- Caracterizar un plano en el espacio mediante un punto del plano y un vector normal al mismo.
- Determinar si una estructura de la representación del mundo es coincidente con dicho plano o no.

Capítulo 6

Implementación

Se ha desarrollado en este proyecto, en C++, un sistema software que satisface los objetivos propuestos. Se ha implementado el sistema de forma que su funcionamiento sea flexible y el usuario pueda ajustar el comportamiento a través de un conjunto de parámetros. Este software posee las siguientes capacidades:

- Capacidad de controlar la posición del dispositivo apuntador y de adquirir mediciones y marcas de tiempo del sensor láser.
- Capacidad de sincronizar por software al dispositivo apuntador y al sensor láser. Esto hará que el sistema sea capaz de localizar espacialmente con precisión las medidas suministradas por el sensor láser.
- Crear un mapa tridimensional del entorno que representa con fiabilidad los objetos, el suelo, paredes y techos que rodean al sistema. Esta representación es rápida de actualizar y compacta.
- El sistema genera una vista 3D del entorno en una ventana en la que se puede cambiar el punto de vista así como hacer zoom.
- Se ha dotado al sistema con la capacidad de detectar planos en el mapa 3D. También es capaz de generar un mapa 2D de obstáculos y zonas transitables.

En el resto del capítulo se muestran detalles acerca de cómo se ha realizado la implementación de los distintos módulos.

6.1. Parámetros del sistema

El comportamiento del sistema puede ser ajustado o regulado mediante una serie de parámetros. Para facilitar la tarea al usuario, estos parámetros se recogen desde un fichero de texto llamado «*datos.dat*» en la misma carpeta que el ejecutable. La estructura de este fichero es sencilla, en el que el valor de cada parámetro se escribe en un línea concreta. Se puede escribir un comentario

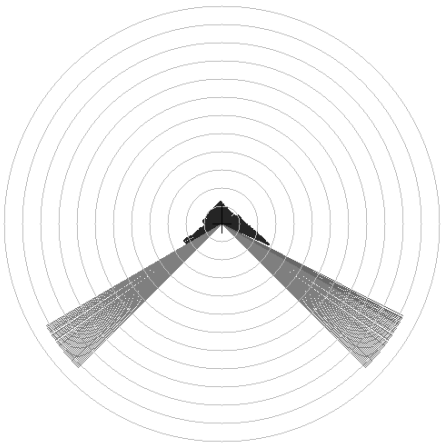


Figura 6.1: Visualización de la información devuelta por el sensor láser en un barrido del haz

antes del valor. El comentario debe acabar con un punto y coma. En la tabla 6.1 encontramos una descripción de cada uno de los parámetros. En las secciones siguientes de este capítulo se irán explicando cada uno de los parámetros en su contexto.

6.2. Generación de coordenadas espaciales

El objetivo de esta fase es obtener coordenadas espaciales de las superficies de los objetos que hay en el entorno tridimensional del robot. En esta primera fase se necesita el concurso del sistema hardware formado por el sensor láser y el dispositivo apuntador. También se necesita conocer la odometría del robot sobre el que va montado el sistema sensor láser-dispositivo apuntador (SLDA). El sistema comanda al SLDA para que obtenga medidas de los objetos presentes en su entorno tridimensional. Para ello, el SLDA se orienta con el ángulo de inclinación lateral (pan_init_angle) y realiza un barrido entre los ángulos de inclinación vertical inicial (tilt_init_angle) y final (tilt_end_angle).

Con la frecuencia programada, el sensor proporciona un array con las mediciones obtenidas en el último barrido. Este es, un array de enteros positivos en el que cada elemento representa en milímetros la distancia medida por el haz. Sólo se consideran medidas válidas las que tengan un valor mayor 20, ya que 20 milímetros es la distancia mínima que puede medir el sensor. Valores menores que 20 indican que se ha producido algún tipo de error (ver 3.3).

Se ha sometido al sensor a pruebas en entornos controlados. Mediante un software de visualización de la lectura del sensor, *urg-demo*, se contrasta la información obtenida por el sensor con el mundo real. El sensor láser realiza mediciones en un único plano (ver fig. 3.1). En la figura 7.1 se muestra la salida de la aplicación *urg-demo* correspondiente al entorno mostrado en la figura 7.2.

La velocidad angular de rotación vertical del dispositivo apuntador deber ser ajustada de forma

Tabla 6.1: Parámetros del sistema

PANTILT PORT	Línea 1	Especifica la ruta al driver del dispositivo apuntador
LASER PORT	Línea 2	Especifica la ruta al driver del sensor láser
MAX_ANGLE	Línea 3	Se usa en la detección de planos. Determina en grados sexagesimales el valor absoluto de la máxima diferencia entre el vector normal a un bloque y el vector normal a un plano para considerar que dicho bloque pertenece al plano
EPSILON	Línea 4	Se usa en la detección de planos. Determina en milímetros la distancia máxima de un bloque a un plano para considerar que dicho bloque pertenece al plano
PROBABILITY	Línea 5	Especifica la probabilidad de corte que un plano candidato debe superar para admitirlo como un plano del mundo
LENGTH OF GRID	Línea 6	Especifica el largo, en número de celdas, que tendrá el mundo
WIDTH OF GRID	Línea 7	Especifica el ancho, en número de celdas, que tendrá el mundo
CELL_SIZE	Línea 8	Determina el ancho y largo en milímetros de cada celda
RADIUS	Línea 9	Especifica la distancia máxima, en número de celdas, de lo que se considera vecindad de un bloque
ROBOT_LASER_HEIGHT	Línea 9	Altura en milímetros a la que se encuentra el sistema formado por el dispositivo apuntador y el sensor láser
ROBOT_POS_X	Línea 10	Coordenada x, en milímetros, de la posición inicial del robot
ROBOT_POS_Y	Línea 11	Coordenada y, en milímetros, de la posición inicial del robot
ROBOT_SIZE	Línea 12	Altura en milímetros del robot. Se usará en detecar que zonas son intransitables
ROBOT_ANGLE	Línea 13	Ángulo inicial del robot respecto al eje z en grados sexagesimales
pan_init_angle	Línea 14	Ángulo lateral del dispositivo apuntador en grados sexagesimales.
tilt_init_angle	Línea 15	Ángulo vertical inicial, en grados sexagesimales, en que el dispositivo apuntador inicia su barrido
tilt_end_angle	Línea 16	Ángulo vertical final, en grados sexagesimales, en que el dispositivo apuntador finaliza su barrido
tilt_inc_angle	Línea 17	Resolución, en grados sexagesimales, del barrido vertical

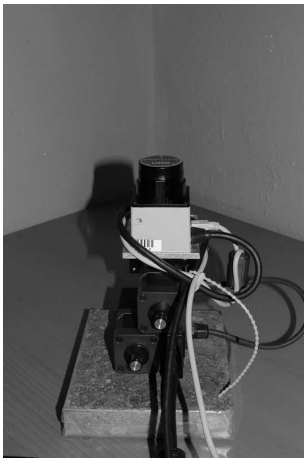


Figura 6.2: Esquina formada por una mesa y dos paredes

que, en el tiempo en que el láser realiza un barrido, el ángulo de elevación se haya desplazado tantos grados como indique el parámetro (tilt_inc_angle). Este parámetro permite ajustar el incremento del ángulo de elevación. La velocidad de desplazamiento, v, en grados por segundo, se obtiene como el producto de la resolución deseada por la frecuencia del haz:

$$v = tilt\_inc\_angle \times frecuencia \ (^{o}/s) \tag{6.1}$$

Se han diseñado tres algoritmos para realizar el barrido del entorno y obtener medidas del mismo. El primer algoritmo sitúa al dispositivo apuntador en un ángulo de inclinación vertical, lo detiene y toma un vector de medidas del sensor láser (ver fig. 3.6). Este algoritmo permite conocer con precisión la orientación del láser en cada medida, sin embargo resulta demasiado lento al tener que situar y detener contínuamente el dispositivo apuntador. El segundo y el tercer algoritmo desarrollados (ver fig. 3.7 y fig. 3.8), permiten obtener medidas del láser sin dejar de mover el dispositivo apuntador. Este último esquema de trabajo permite una mayor velocidad de funcionamiento al realizar el dispositivo apuntador el desplazamiento vertical, entre tilt_init_angle y tilt_end_angle, de forma contínua, como se indicó en 3.3.

El segundo algoritmo, por simplicidad, asigna el mismo ángulo de inclinación vertical a todas las medidas obtenidas por el láser en una misma rotación. Esto, sin embargo, introduce un error en la localización del haz láser ya que, en lo que se realiza el barrido, el dispositivo apuntador ha seguido desplazándose. Recordamos que el sensor láser es capaz de hacer lecturas válidas en un rango de 240º en torno a su eje (ver fig. 3.2), lo que representa 2/3 de una rotación completa. Pues bien, en 2/3 del tiempo mínimo que el haz tarda en hacer un giro, $100ms.$, siendo λ el ángulo de elevación, el dispositivo apuntador se habrá desplazado:

$$\Delta\lambda = v \times \frac{2 \times 0{,}100}{3} = \frac{2 \times tilt\_inc\_angle}{3} \tag{6.2}$$

Se ha comprobado experimentalmente que una resolución de 0,5º de desplazamiento vertical por cada rotación completa del haz es adecuada para entornos de interior. Con dicha resolución, el error en la posición del haz láser entre el comienzo y el final del área de 240º es de $\frac{2\times0,5}{3} = 0,33º$. Evidentemente, el efecto del error se multiplica con la distancia del objeto detectado. Se ha realizado un estudio del error introducido por este esquema de funcionamiento en objetos que se encuentren a 4,5 m., distancia próxima al límite de detección del láser. Para ello se ha calculado cuál sería la posición correcta de cada una de las 682 lecturas que devuelve el sensor en el rango de los 240º. Los resultados arrojan que el error máximo, para la distancia indicada, entre la estimación realizada y la localización exacta sería de 38.84 mm. El mapa 3D de representación del mundo implementado discretiza el mundo. Por ello, este error podría en parte absorbido por dicha discretización. En las pruebas de laboratorio efectuadas, el tamaño de celda habitual ha sido de 50 mm. Por otra parte, también se ha estimado el número de lecturas que serían ubicadas en una celda del suelo errónea debido a este error. Entre las 682 mediciones, 12 de ellas serían ubicadas en celdas erróneas, lo que representa el 1,75 %. Estos resultados muestran que existe un error introducido por realizar un barrido contínuo del dispositivo sin calcular el ángulo de inclinación en cada medida. Según el uso que se haga de la aplicación y la necesidad de precisión este error podría ser grave. Se muestra en la figura xxx el error absoluto en milímetros generado en cada paso de un barrido del láser. El error ha sido calculado para objetos que se encuentren a 4.5 m. del sensor.

En la versión final de este proyecto, se ha usado el tercer algoritmo (ver fig. 3.8), el cual sí estima el ángulo del apuntador en el instante en que se toma cada medida. Vemos en la figura xxx una escena que ha sido muestreada usando el segundo y el tercer algoritmo.

Se ha tratado el problema de obtener las coordenadas espaciales como un problema cinemático directo. El modelo Denavit-Hartenberg realizado para nuestro sistema ha respondido con precisión a los problemas de obtención de las coordenadas espaciales del entorno. El procedimiento que convierte las lecturas del sensor en coordenadas espaciales del mundo, toma como entrada el vector de enteros proporcionado por el sensor, los parámetros de velocidad y aceleración del dispositivo apuntador, las lecturas de tiempo del instante de inicio del barrido del dispositivo apuntador y de la rotación del láser y la odometría del robot. El procedimiento, de forma iterativa, obtiene las coordenadas espaciales correspondientes a cada una de las medidas proporcionadas por el sensor láser. Dentro del bucle, se calcula en primer lugar, para cada medida proporcionada por el láser, la posición del dispositivo apuntador en el instante que se obtuvo dicha medida. A continuación, genera las matrices de transformación m_1, m_2 y m_3 de la cadena cinemática formada por los dos motores del dispositivo apuntador y el haz láser. La multiplicación de las matrices da como resultado la matriz de salida s que proporciona las coordenadas espaciales. Vemos en los siguientes parráfos el procedimiento con detalle.

Sea t_0 el instante en que el dispositivo apuntador comienzo su desplazamiento entre $tilt\_init\_angle$ y $tilt\_end\_angle$ y sea t_t el instante en que el haz láser inició la rotación en la que se obtuvo el vector de medidias $L = \{l_0, l_1, l_2, \cdots, l_{681}\}$

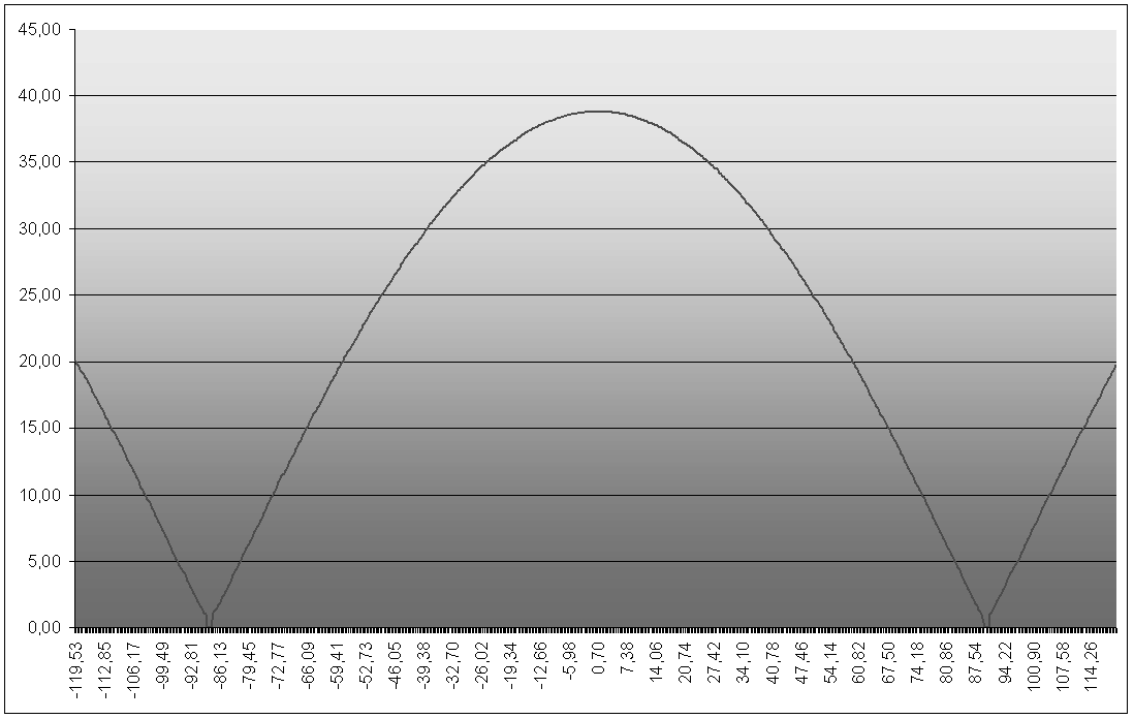


Figura 6.3: Error absoluto en la localización generado por el algoritmo que supone el mismo ángulo de inclinación vertical para todas las medidas obtenidas en una rotación del haz.

Teniendo en cuenta que el haz proporciona la primera lectura válida, l_0, en el paso 44 (ver 3.2), se puede estimar el instante t_i en que se tomó cada medida l_i para una frecuencia f del haz, como:

$$t_i = t_t + \frac{44 + i}{1024 \cdot f} \tag{6.3}$$

El lapso de tiempo tl_i transcurrido desde el instante en que comenzó el desplazamiento, t_0, hasta el instante t_i en que se toma la muestra l_i es igual a:

$$tl_i = t_i - t_0 = t_t + \frac{44 + i}{1024 \cdot f} - t_0 \tag{6.4}$$

El ángulo de inclinación vertical, $tilt$, en el instante t_i se obtiene a partir del esquema de aceleración del dispositivo apuntador. Se presentan tres posibles casos (ver fig. 6.4). Si la velocidad v programada obtenida en la ecuación 6.1 es inferior a la velocidad base del dispositivo apuntador, v_b no se producirá aceleración, la velocidad será constante en todo el desplazamiento y el ángulo de inclinación vertical $tilt_i$ correspondiente a la medida l_i se obtendrá como:

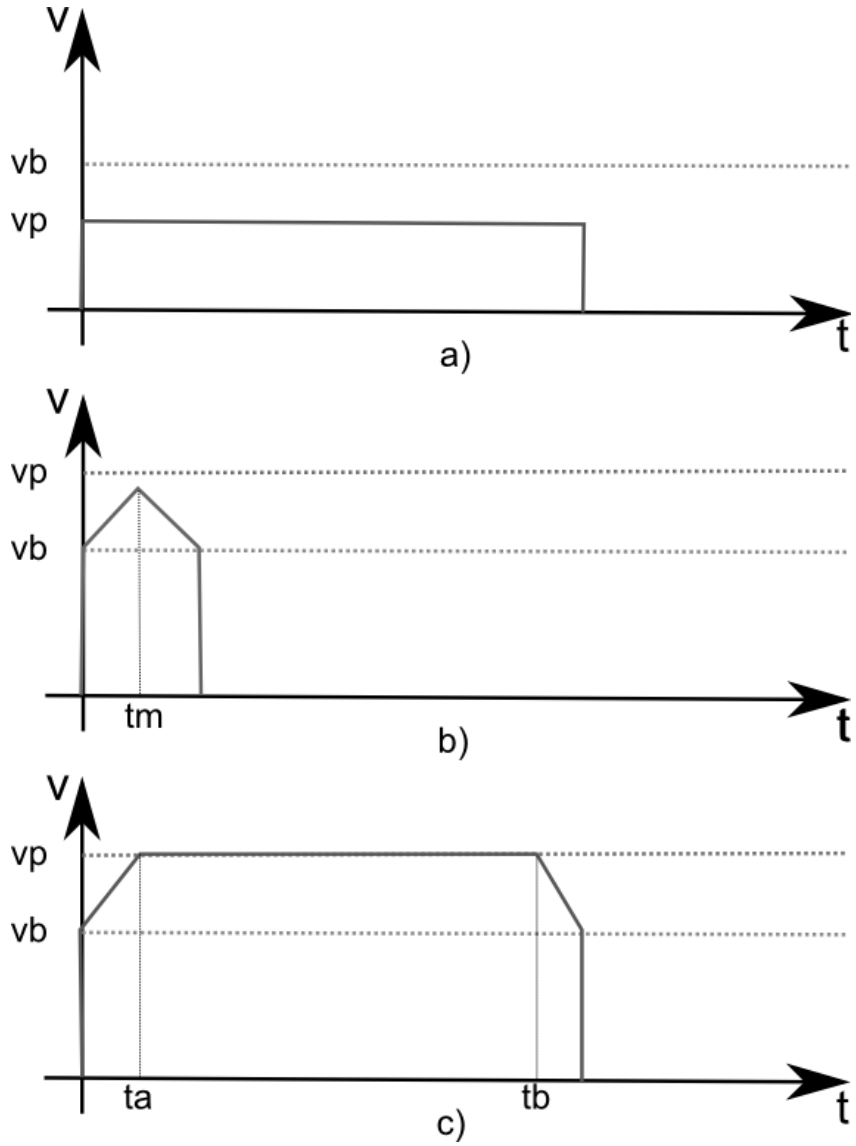


Figura 6.4: Tres posibles esquemas de aceleración del dispositivo apuntador. En a) se presenta el caso en que la velocidad programada es inferior a la velocidad base, por lo que todo el desplazamiento se realizará con velocidad constante, sin aceleración. En b) vemos la situación en que se llega a la mitad del recorrido antes de alcanzar la velocidad programada. Esto da lugar a un esquema de aceleración triangular. Finalmente, en c) la velocidad programada es superior a la velocidad base y se alcanza la velocidad programada antes de llegar a la posición intermedia. Se sigue en dicho caso un esquema de aceleración trapezoidal.

$$tilt_i = tilt\_init\_angle + v \cdot tl_i \tag{6.5}$$

Si v es mayor que la velocidad base del dispositivo apuntador, vb, tendremos un desplazamiento con aceleración a. Si el tiempo que se tarda en alcanzar la posición media entre tit_init_angle y tit_end_angle es menor que el tiempo que se tarda en llegar a la velocidad v, el dispositivo empezará a frenar siguiendo un esquema de aceleración triangular. Si por el contrario se alcanza la velocidad v antes de llegar a la posición media, tendremos un esquema de aceleración trapezoidal. Veamos como se obtiene el ángulo de inclinación vertical $tilt_i$ en cada uno de estos casos. Para ello, primero calcularemos ciertas posiciones y tiempos intermedios.

Sea t_a, el tiempo que se tarda en alcanzar la velocidad v para una aceleración a, igual a:

$$t_a = \frac{v - vb}{a} \tag{6.6}$$

Sea $tilt_a$, el ángulo de inclinación vertical cuando se la alcanza la velocidad v, igual a:

$$tilt_a = tilt\_init\_angle + v_b \cdot t_a + \frac{a \cdot t_a^2}{2} \tag{6.7}$$

Sea t_m, el tiempo que tarda el dispositivo apuntador en alcanzar la mitad de su recorrido, igual a:

$$t_m = \frac{-vb + \sqrt{vb^2 - a \cdot (tilt\_end\_angle - tilt\_init\_angle)}}{a} \tag{6.8}$$

Sea $tilt_m$, la posición media del recorrido del dispositivo apuntador, igual a:

$$tilt_m = \frac{tilt\_end\_angle + tilt\_init\_angle}{2} \tag{6.9}$$

Sea t_b, el tiempo que transcurre antes de empezar a frenar en el esquema trapezoidal, igual a:

$$t_b = t_a + \frac{tilt\_end\_angle - 2 \cdot tilt_a}{2} \tag{6.10}$$

Sea $tilt_b$, el ángulo de inclinación vertical en el instante en que el dispositivo comienza a frenar en un esquema trapezoidal, igual a:

$$tilt_b = tilt_a + \upsilon(t_b - t_a) \tag{6.11}$$

Entonces, si $t_m \leq t_a$, el ángulo de inclinación vertical $tilt_i$ en que se tomó la medida l_i es igual a:

$$tilt_i = \begin{cases} tilt\_init\_angle + v_b \cdot tl_i + \frac{a \cdot tl_i^2}{2} & \text{si} \quad tl_i \leq t_m \\ tilt_m + v_m \cdot (tl_i - t_m) - \frac{a \cdot (tl_i - t_m)^2}{2} & \text{si} \quad tl_i > t_m \end{cases} \tag{6.12}$$

Si por contra, si $t_m > t_a$, el ángulo de inclinación vertical $tilt_i$ en que se tomó la medida l_i es igual a:

$$tilt_i = \begin{cases} tilt\_init\_angle + v_b \cdot tl_i + \frac{a \cdot tl_i^2}{2} & \text{si} \quad tl_i \leq t_a \\ tilt_a + \upsilon \cdot (tl_i - t_a) & \text{si} \quad t_a < tl_i \leq t_b \\ tilt_b + \upsilon \cdot (tl_i - t_b) - \frac{a \cdot (tl_i - t_b)^2}{2} & \text{si} \quad tl_i > t_b \end{cases} \tag{6.13}$$

Una vez conocido el ángulo de inclinación vertical de cada medida, se generan las matrices de transformación correspondientes a la cadena cinemática. Sea XY el plano correspondiente al suelo. Sea (r_x, r_y) las coordenadas del robot en dicho plano y sea r_θ el ángulo que expresa la orientación del robot. Sea pan_angle el parámetro que indica el ángulo de inclinación lateral con que se tomarán las medidas y $tilt_i$ el ángulo de inclinación vertical obtenido mediante una de las ecuaciones 6.5, 6.12 o 6.13. Sea h_0 la altura sobre el suelo del eje del motor de inclinación vertical. Sea $angle_i$ el ángulo del haz con que se obtuvo la medida l_i medido respecto al eje del sensor láser. De la figura 3.2 se deduce que $angle_i$ es igual a:

$$angle_i = -135 + \frac{(i + 44) \cdot 360}{1024} \tag{6.14}$$

Las matrices de transformación correspondientes a cada uno de los eslabones de la cadena cinemática se obtienen mediante las siguientes funciones m_1, m_2 y m_3:

$$m_1(pan\_angle, r_\theta, h_0) = \begin{bmatrix} cos(pan\_angle + r_\theta) & 0 & -sen(pan\_angle + r_\theta) & 0 \\ sen(pan\_angle + r_\theta) & 0 & cos(pan\_angle + r_\theta) & 0 \\ 0 & -1 & 0 & h_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.15}$$

$$m_2(tilt_i) = \begin{bmatrix} cos(tilt_i) & 0 & -sen(tilt_i) & -50 \cdot cos(tilt_i) \\ -sen(tilt_i) & 0 & -cos(tilt_i) & 50 \cdot sen(tilt_i) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.16}$$

$$m_3(angle_i, l_i) = \begin{bmatrix} cos(angle_i) & -sen(angle_i) & 0 & l_i \cdot cos(angle_i) \\ sen(angle_i) & cos(angle_i) & 0 & l_i \cdot sen(angle_i) \\ 0 & 0 & 1 & 100 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.17}$$

El producto de las tres matrices dadas por las funciones anteriores genera la matriz S_i igual a:

$$S_i = m_1(pan\_angle, r_\theta, h_0) \cdot m_2(tilt_i) \cdot m_3(angle_i, l_i) \tag{6.18}$$

Los elementos $S_i^{1,4}$, $S_i^{2,4}$ y $S_i^{3,4}$ son las coordenadas de la medida l_i respecto a la posición del robot. Para obtener las coordenadas, C_i, respecto al sistema de referencia del mundo habrá que sumar las coordenadas del robot:

$$C_i = (x_i, y_i, z_i) = (S_i^{1,4} + r_x, S_i^{2,4} + r_y, S_i^{3,4}) \tag{6.19}$$

Estas coordenadas se almacenan en un array del tipo double del C estándar.

Los procesos de conversión de coordenadas espaciales y de actualización del mapa multisuperficies se deben ejecutar con una frecuencia superior a los 10 Hz. con que el sensor láser suministra el array de medidas. Si esta conversión fuera inferior a los 10 Hz. el efecto que produciría sería diferente en los tres algoritmos anteriormente mencionados. En el primer algoritmo esto no provocaría ningún efecto, más allá de una mayor tardanza, al esperar en este caso que se hayan acabado de procesar las medidas antes de mover el dispositivo apuntador a la nueva posición. El segundo algoritmo sí mostraría una malfunción ya que, en este caso, se realiza la estimación de la posición del dispositivo apuntador a ciegas, sin considerar marcas de tiempo ni de posición de ningún tipo. El tercer algoritmo no mostraría malfunción ya que, como hemos visto anteriormente, el ángulo de elevación vertical correspondiente a cada medida se calcula en función de las marcas de tiempo. El efecto que produciría en el tercer algoritmo una frecuencia de procesamiento superior a los 10Hz. de funcionamiento del sensor láser es que, ciertos barridos del sensor láser, serían descartados al estar aún procesándose el barrido anterior. Como resultado, la resolución efectiva sería menor que la resolución programada.

Tabla 6.2: Resumen de especificaciones del proceso de conversión en coordenadas espaciales

Parámetros de entrada:	Ángulo de inclinación lateral Ángulo de inclinación vertical Odometría del robot Marcas de tiempo
Datos de salida:	682 coordenadas espaciales (x,y,z) en formato double
Tamaño de la salida:	$16368 = 3 \times 682 \times 8$ bytes
Tiempo promedio de procesamiento:	$0,613$ ms.
Desviación estándar del tiempo de procesamiento:	$0,0566$

El tipo double en C++ ocupa 8 bytes. El array de coordenadas espaciales requiere $t = 682 \times 3 \times 8 = 16368$ bytes. Por otra parte, se ha obtenido una estimación del tiempo que consume el proceso de conversión en coordenadas espaciales. Para ello, se ha medido 500 veces el tiempo necesario para procesar un array de 682 elementos y posteriormente se ha estimado su promedio. El tiempo promedio obtenido es de $0,613$ ms. por array, con una desviación estándar de $0,0566$ ms. En la tabla 6.2 se resumen las especificaciones del proceso de conversión de las medidas del láser en coordenadas espaciales.

6.3. Actualización del mapa de superficies multinivel

Al tiempo que el sistema va proporcionando coordenadas espaciales de los objetos en su entorno, se construye una representación del mundo mediante un mapa de superficies multinivel. Este mapa permite detectar obstáculos (objetos que pueden colisionar con el robot) y superficies superpuestas a distintos niveles. Para esta tarea se hace uso de tras clases: *ground*, *cell* y *block* (ver fig. 6.5). Las responsabilidades de cada una de esas clases se expuso en la sección 5.2

La gestión del mapa multinivel debe tratar dos tipos de eventos. Por una parte, la incorporación de coordenadas de las superficies de los objetos del entorno. El sistema SLDA proporciona esta información como hemos visto en la sección anterior. Las coordenadas se incorporan al contenido actual del mapa. Además de esto, y para que el sistema sea capaz de funcionar en un mundo dinámico, el sistema tiene que ser capaz de detectar la desaparición de objetos que anteriormente se habían incorporado al mapa. Esta segunda tarea implica que, cada vez que se añade una nueva lectura del láser, hay que recorrer todo el trayecto del haz para comprobar si su trayectoria corta alguna estructura del mapa. En caso afirmativo se tendrá que abrir un «agujero» en la estructura.

El mundo será representado por el objeto *floor*, instancia de la clase *ground*. Describiremos a continuación cómo se realizan las dos tareas expuestas en el párrafo anterior.

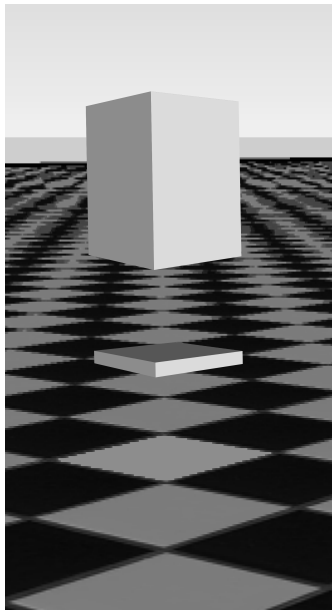


Figura 6.5: Representación gráfica de las tres principales: ground, cell y block. Ground es una malla bidimensional que contiene toda la información del mapa. Cell es cada una de las celdas de ground y que contiene una lista enlazada de objetos tipo block. Block es cada una de las estructuras sólidas, bloques, que se encuentran en la vertical de una celda. Puede haber bloques planos, superficies, y bloques con altura, estructuras verticales.

La clase *ground* contiene el conocimiento que el sistema posee en un instante determinado del mundo, tal y como se representa en los mapas de superficies multinivel (ver 4.3). Recordamos que tales mapas consisten en una matriz de celdas, cada una de las cuales almacena una lista de las estructuras sólidas que se encuentren en su vertical. Cada celda discretiza una región del suelo con un ancho y un largo fijos.

La clase *ground* posee una matriz, *ground_grid*, de punteros a objetos de la clase *cell*. El tamaño de esta matriz, (W, L), es un par de parámetros del sistema. El largo y ancho de las celdas cuadradas se fija a través del parámetro *CELL_SIZE* en milímetros. Cuando se crea una nueva instancia de la clase *ground* se crean en total $W \times L$ instancias de *cell*.

Cada vez que el proceso de conversión a coordenadas espaciales tiene un array listo, invoca al método *add_impact* de la clase *ground* pasando como parámetros las coordenadas espaciales del sensor láser, las coordenadas espaciales del punto a ser añadido y la varianza asociada a dicha medida. La función correspondiente comprueba en primer lugar si, en la trayectoria del haz láser, éste atraviesa alguna estructura que ya estuviera en el mapa invocando para ello al método *ray-tracing* de la clase *ground*. Posteriormente procesa el nuevo punto a añadir invocando al método *add_point* de la clase *ground*. Este método calcula la celda correspondiente y le pasa las nuevas coordenadas a la celda para que actualice las estructuras adecuadas o añada una nueva.Mostramos el pseudocódigo en la figura **??**

Durante el proyecto se han desarrollado dos algoritmos para explorar el trazado de los haces láser. Uno se ha basado en el algoritmo de Bresenham para el trazado eficiente de segmentos en sistema digitales. Posteriormente se desarrolló otro algoritmo basado en la ecuación real de la recta que, siendo más costoso computacionalmente, mostraba una mayor precisión en el trazado del haz y, por tanto, servía para modelar mejor el comportamiento del mundo. El algoritmo calcula cada celda por la que pasa el rayo e invoca al método *add_hole* de la clase *cell*. Este método comprueba si el rayo intersecta a alguno de los bloque que hay en su lista. Si es así, se abre un «agujero» en el bloque (ver fig. 6.7). Este agujero puede ser en su parte superior o inferior, en cuyo caso simplemente se modificaría los atributos *mean_z* y *height* del bloque. Si el agujero se abriera en medio, el bloque se escindiría en dos cada uno con los atributos adecuados. El agujero que el rayo abre en el bloque tiene un diámetro de $\frac{CELL\_SIZE}{4}$ mm., de esta forma nos aseguramos que no queden residuos (estructuras de objetos que han desaparecido del mundo o se han movido) alrededor del rayo. Si al finalizar la operación de agujerear un bloque vertical, el tamaño de este se quedara reducido a menos de CELL_SIZE mm., el bloque se eliminaría completamente.

La clase *cell* encapsula la información y el comportamiento de cada celda de la matriz del mundo. Básicamente, su responsabilidad consistirá en mantener una lista doblemente encadenada y ordenada de las estructuras sólidas que se encuentren en la vertical de la celda. La ordenación se hace según la altura de cada estructura.

El método *add_point* de la clase *cell* comprueba si existe ya alguna estructura que diste menos de CELL_SIZE mm. del punto de impacto. Si es así, se pasa a la estructura en cuestión las nuevas

```
método ground.add_impact
entrada:
rx,ry,rz - coordenadas donde se localiza el robot
x,y,z - nuevas coordenadas a incorporar al mapa 3D
v - varianza de la medida

/* se invoca al método que detecta todas las estructuras en el trazado del rayo y
las modifica para reflejar que han desaparecido */
raytracing(rx,ry,rz,x,y,z)
/* se invoca al método que procesa las nuevas coordenadas */
add_point(x,y,z,v)

método ground.add_point
entrada:
x,y,z - nuevas coordenadas a incorporar al mapa 3D

v - varianza de la medida
i <- 200 + entero(x/CELL_SIZE)
j <- 200 + entero(y/CELL_SIZE)
/* se pasa las nuevas coordenadas a la medida correspondiente */
celda(i,j)->add_point(x,y,z,v)
```

Figura 6.6: Métodos para añadir un nuevo impacto en el mapa 3D

coordenadas. Si ninguna estructura dista menos de CELL_SIZE mm., el método crea una nueva instancia de la clase *block* y la encadena en el lugar correspondiente de la lista. Es también posible que un nuevo impacto caiga entre dos estructuras consecutivas y que, gracias a este nuevo impacto, se puedan fusionar en una sola. Para que se produzca tal fusión los objetos deben encontrarse a menos de $2 \times CELL\_SIZE$ mm. En tal caso, se elimina la estructura que se encuentrara más bajo y se actualiza el tamaño del objeto más alto, de forma que a partir de tal instante abarque la extensión ocupada anteriormente por los dos bloques. Se muestra el pseudocódigo correspondiente en la figura 6.8.

La clase *block* representa cada una de las estructuras sólidas que se encuentran en la vertical de una celda $c_{i,j}$. Cada objeto de esta clase tendrá unos atributos que definirán a la estructura. Los atributos *mean_x, mean_y, mean_z* representan el valor promedio de las coordenadas x, y, z de todos los impactos que han caido dentro de la estructura. Estos atributos se actualizan cada vez que llega un nuevo impacto mediante la aplicación de un filtro de Kalman. Para estimar el filtro de Kalman, los objetos de esta clase tienen un atribuo llamado *var* que almacena la varianza de las coordenadas medidas en la posición celda $c_{i,j}$. El parámetro *mean_z* representa la posibilidad de atravesar por dicha altura el entorno 3D, mientras que la varianza representa la incertidumbre

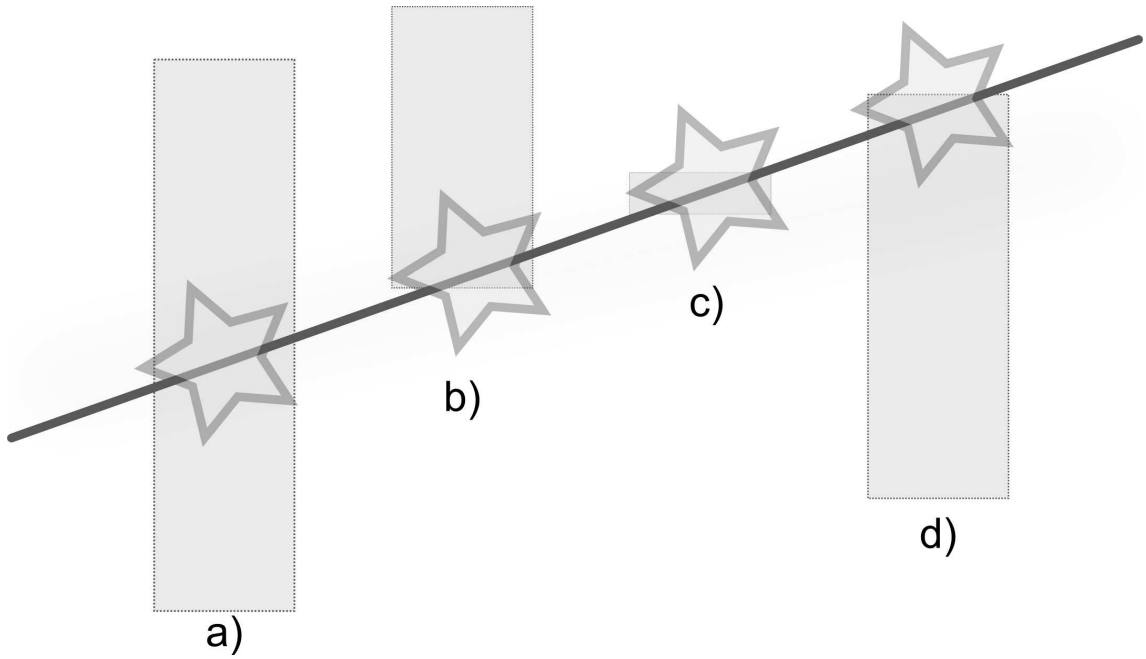


Figura 6.7: Cuando se incorpora una nueva medida del láser hay que comprobar si el rayo en su trayectoria ha atrevasado alguna estructura que estuviera presente el mapa. Eso indica que esa estructura se ha movido o ha desaparecido y hay que reflejar este cambio en el mapa. Surgen 4 posibles escenarios. a) El rayo atraviesa el interior de una estructura: la estructura se divide en dos. b)El rayo intercepta la parte inferior de una estructura: se disminuye la altura. c) El rayo atraviesa una estructura plana: se elimina completamente. d) El rayo intercepta la parte superior de una estructura: se modifica el valor promedio de altura (mean_z) y se disminuye su altura

```
método cell.add_point
entrada:
x,y,z - nuevas coordenadas a incorporar al mapa 3D
v - varianza de la medida

si la lista de bloques de la celda está vacía entonces
          añadir un nuevo bloque con las coordenadas (x,y,z) y la varianza v
          retornar
si el bloque cuya altura sea la más cercana a z dista menos de CELL_SIZE de z entonces
          actualizar el bloque con las nuevas coordenadas
          retornar
si las nuevas coordenadas caen entre dos bloques que distan menos de 2*CELL_SIZE
entonces
          fusionar los dos bloques
          retornar
si no es ninguno de los anteriores casos entonces
          se crea un nuevo bloque con las nuevas coordenadas
          retornar
```

Figura 6.8: Método add_point de la clase cell

de poder hacerlo. Mostramos en la figura **??** la implementación del filtro de Kalman que se lleva a cabo en el método *add_point* de la clase *block*.

En nuestro modelo de mapa multinivel tendremos dos tipos de bloque, aquellos que representan estructuras verticales (una sección de una pared, armario, etc.) y los bloque planos, sin altura, que representan superficies planas (suelos, techos, mesas, etc.). Los objetos planos tienen el atributo *height* con un valor igual a 0,0, mientras que en las estructuras verticales este atributo contiene la diferencia de altura entre el impacto más alto y el más bajo que se han producido en la estructura. Un objeto plano se puede convertir en una estructura vertical si se produce algún impacto a menos de CELL_SIZE mm. de *mean_z* y a más de tres veces la varianza del bloque.

El consumo de memoria por objeto se especifica en la tabla 6.3. En esta tabla podemos encontrar una estimación del uso de memoria por parte de un mapa 3D con una matriz de $400x400$ celdas y 3000 bloques. Las pruebas realizadas muestran que una matriz de 400x400 celdas de 50 mm. de lado permiten generar un mapa útil del entorno. Por otra parte, distintos escenarios probados se representan en el mapa con menos de 3000 bloques. La necesidad total de memoria es, en este caso, de aproximadamente 7 Mb. Este consumo de memoria permite superar con creces el objetivo de que el sistema desarrollado pudiera ejecutarse en un equipo con 1Gb. de memoria RAM.

Por otra parte, es crítico el tiempo de procesamiento pues la actualización del mundo se realiza en paralelo con la adquisición de nuevas mediciones. Como ya sabemos, la frecuencia de sensor láser es de 100 ms. El tiempo de actualización del mundo por impacto no es constante. Depende de la

```
método block.add_point

entrada:
x,y,z - coordenadas de la nueva medida
v - varianza de la medida

/*
mean_x, mean_y, mean_z son la estimación actual de las coordenadas de los impactos que
han caido sobre el bloque (estructura). Esta estimación se actualiza mediante un
filtro de Kalman
var es la estimación actual de la varianza
*/
k=var/(var+v);
mean_x = (1.0-k)*mean_x+k*x;
mean_y = (1.0-k)*mean_y+k*y;
mean_z = (1.0-k)*mean_z+k*z;
var = (1.0-k)*var;
```

Figura 6.9: Implementación del filtro de Kalman en el método add_point de la clase block

Tabla 6.3: Consumo de memoria del mapa 3D

Clase ground	1280464B
Clase cell	32B
Clase block	120B
Clase plane	56B
Matriz de 400 x 400 celdas	5120000B
Mundo de 3000 bloques	360000B
Mapa 3D de 400 x 400 celdas y 3000 bloques	6760464B

distancia del objeto puesto que cuanto más distante, más área habrá que rastrear para detectar si hay bloques que «agujerear». De la misma forma, no es lo mismo un mapa vacío en el que cualquier impacto causa que se tenga que instanciar un nuevo objeto, que un mapa ya densamente construido en el que la mayoría de los impactos sólo causarán la actualización de algún contador. Por ello los tiempos que se muestran aquí se han obtenido a partir de los tiempos de 93092 impactos producidos en distintas configuraciones del mapa. El tiempo promedio por impacto fué de $t_p = 0{,}013$ms. Siendo 682 el número de valores devuelto por el sensor láser en cada rotación, el tiempo promedio empleado en la actualización del mapa multinivel para un vector de datos devuelto por el sensor láser es de $t_t = 0{,}013 \times 682 = 8{,}866$ms. Si a este tiempo le sumamos el que se emplea en convertir las medidas del sensor en coordenadas espaciales, nos sale un tiempo de procesamiento total por rotación del haz láser de $T = 8{,}866 + 0{,}613 = 9{,}479$ms. Este tiempo es muy inferior a los 100 ms. del sensor láser. Esto determina que sólo se usa el procesador un 9,479 % del tiempo, restando un 90,521 % de tiempo que se podría usar en realizar otras tareas en paralelo.

Para poder visualizar los resultados en tres dimensiones, este proyecto ha usado las librerías OpenGL[1]. OpenGL es una especificación estándar que define una API multilenguaje y multiplata-

[1]http://www.opengl.org/

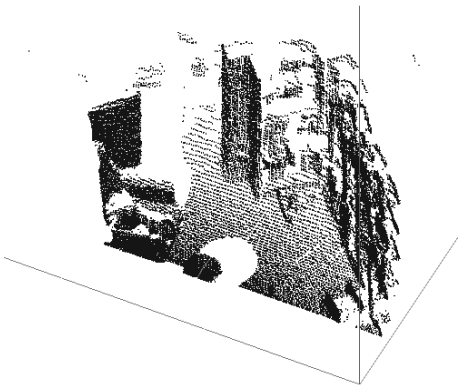


Figura 6.10: Vista tridimensional del mapa creado a partir de un escenario real(1)

forma para escribir aplicaciones que produzcan gráficos 2D y 3D. Consta de más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por Silicon Graphics Inc. en 1992 y es usada ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo (Wikipedia).

Vemos en la figura 7.7 una representación del escenario mostrado en la figura 7.5. Se ha representado en color rojo las estructuras verticales, esto es, elementos del entorno que no son atravesables por pertenecer a una estructura mayor (p.e. parte de una pared). En color azul se han representado las superficies, las partes superiores de los distintos objetos que hay en el entorno. Para crear este mapa se ha utilizado un tamaño de discretización del mundo de 25 milímetros cuadrados.

Vemos en las figuras 7.3 y 7.4 dos construcciones realizadas en los entornos reales que vemos en las figuras 7.5 y7.6. Las dos imágenes se han obtenido haciendo un barrido de 60º en la componente vertical, entre los -45º y los 15º, correspondiendo los 0º con la línea horizontal. La resolución que se ha utilizado ha sido de una medición del sensor cada 0'5º. Como se ha justificado en el capítulo anterior, la velocidad máxima de barrido del dispositivo viene limitada por la frecuencia de barrido del haz láser que es de 10Hz. Por tanto, en barrer los 60º con una resolución seleccionada se tarda 12 segundos. Simultáneamente a la recolección de los datos suministrados por el sensor se va construyendo el mapa tridimensional, habiéndose conseguido unos tiempos de latencia en la construcción del mapa despreciables frente a los 100 milisegundos que tarda el sensor en hacer una rotación completa del haz, es decir, en adquirir un vector de medidas.

6.4. Extracción de primitivas

En este proyecto, hemos implementado una variante del algoritmo *efficient-Ransac* [7]. El algoritmo *efficient-Ransac* opera sobre nubes de puntos. En nuestro algoritmo operaremos sobre

Figura 6.11: Vista tridimensional del mapa creado a partir de un escenario real(2)

Figura 6.12: Escenario real para el mapa(1)

Figura 6.13: Escenario real para el mapa(2)

Gaussianos de los mapas multinivel. Además, hemos restringido la búsqueda de formas a planos por ser estos de gran importancia en las tareas de localización en interiores pues, en estos escenarios, abundan este tipo de formas: paredes, techos, mesas, etc.

El algoritmo opera sobre todo el mapa de superficies multinivel. Esto evita tener que almacenar todas las coordenadas que se van generando por parte del sensor láser, lo que multiplicaría varias veces las necesidades de memoria. Por otra parte, al ser más compacta la representación sobre la que opera el algoritmo, el proceso de descubrimiento de planos en el entorno se acelera en gran medida. Además, se ha introducido una modificación en el algoritmo eRANSAC lo que permite una convergencia más rápida en el descubrimiento de planos como se explicará más adelante.

De forma iterativa se irán generando planos candidatos. Estos planos candidatos se generan a partir de los bloque presentes en el mapa multinivel. Antes de iniciar el bucle de generación y testeo de planos candidatos, hay que calcular los vectores normales a cada bloque. Este vector normal se estima mediante el análisis de componentes principales (ACP). Este método, ideado para encontrar la dimensión o dimensiones en donde se encuentra la mayor variación de un conjunto de datos, también puede ser usado para estimar la normal a una superficie que se desconoce pero de la que se poseen algunos puntos. Para cada bloque b_i se aplica el ACP al conjunto formado por él mismo y los bloques b_j próximos. El método se aplica de la siguiente forma:

1. Se genera la matriz de covarianza de los elementos b_i y b_j.
2. Se calculan los autovectores, v_k, y autovalores, a_k, de la matriz de covarianza.
3. El autovector correspondiente al menor autovalor será justamente la estimación del vector normal a la superficie que uniría todos los puntos, en nuestro caso, bloques.

Los pasos que seguimos en nuestro algoritmo de detección de planos son:

1. Se selecciona al azar un bloque del mundo. Este bloque puede ser un bloque del tipo plano o superficie, esto es, su altura es 0, o puede ser del tipo estructura vertical, con altura distinta de cero. Según el tipo de este primer bloque el resto de la iteración opera sólo con bloques del mismo tipo. El sentido de esto es que en el mundo, las superficies planas abarcan estructuras del mismo tipo. Una pared, por ejemplo, está toda ella formada por bloques verticales. Sin embargo, el tablero de una mesa está formado por bloques planos por completo.
2. Se seleccionan al azar dos bloques de la vecindad del bloque seleccionado en el primer paso. Estos dos bloques deberán tener vectores normales que no difieran del vector normal del bloque anterior más de MAX_ANGLE grados, siendo éste un parámetro del sistema. El radio de esta vecindad vendrá determinado por el parámetro del sistema $RADIUS$. Este radio se especifica en número de celdas. Por tanto, el alcance real de este radio, expresado en milímetros será $RADIUS \times CELL_SIZE$. Para determinar la distancia entre dos bloques se usa la distancia d_∞ por ser menos costosa computacionalmente. Este radio tendrá una determinación en el comportamiento del algoritmo. Cuanto más pequeño sea, más probabilidad

hay de que los bloques seleccionados pertenezcan al mismo plano. Sin embargo, cuanto más próximos sean los bloques, el plano candidato que se genere tendrá más probabilidades de estar influido por anomalías locales como puede ser una irregularidad en una pared o suelo.

3. Se genera un plano a partir de los tres bloques seleccionados en los puntos anteriores. El plano se caracteriza mediante un punto y un vector normal. Como punto se elige el baricentro de los tres bloques y como vector normal se elige el vector promedio de los vectores normales a los tres bloques. Esta es una modificación respecto a *efficient-Ransac* en donde la normal al plano se tomaba como la normal al primer punto seleccionado. Entendemos, sin embargo, que utilizar como normal el promedio de los tres bloques seleccionados minimiza el error que un único elemento aislado de una superficie no perfecta puede tener. La convergencia aumenta al introducir este cambio.

4. Se asigna una puntuación al plano candidato. Esta puntuación es el número de bloques del mundo que se concideran coincidentes con el plano candidato. El concepto coincidente implica que su distancia al plano es menor que el parámetro *EPSILON* y su normal no difiere más de *MAX_ANGLE* grados de la normal del plano.

5. Entre todos los planos candidatos que poseamos, se elige al de mayor puntuación y sólo se admite como válido si la probabilidad de elegir al azar dicho plano es inferior a $1 - PROBABILITY$, siendo $PROBABILITY$ un parámetro del sistema. Como se comenta en 4.4.2, en el algoritmo *efficient-Ransac* el cálculo de esta probabilidad se hace tomando en cuenta todas las combinaciones entre todos los puntos de la nube. Esto implica que, aún teniendo un plano candidato que abarque a un tercio de todos los puntos desde el principio, haría falta haber generado otros 122 planos para que el plano supere el test de probabilidad si queremos una probabilidad de 0,99. Sin duda el método es riguroso, pero se puede acelerar si tenemos en cuenta que los planos candidatos son generados a partir sólo de puntos, bloques en nuestro caso, que tienen unas normales que no difieren entre si más de *MAX_ANGLE* grados. Por tanto, en vez de testear al plano contra todos los bloques del mundo, lo hacemos sólo contra los bloques con una normal que no difiera más de *MAX_ANGLE* grados. Esto acelera enormemente el número de iteraciones necesarias para descubrir planos válidos no generando por ello planos incorrectos.

6. En caso de haber encontrado un plano que supere el test de probabilidad, se almacena en una lista y se eliminan los planos candidatos que tuvieran bloques coincidentes con el plano ganador. Los bloques coincidentes con el plano ganador ya no vuelven a ser usados para generar nuevos planos ni para puntuar a los futuros candidatos.

A cada plano encontrado se le asigna un color que servirá posteriormente para etiquetar cada uno de los bloques coincidentes con él y para representarlos gráficamente.

Capítulo 7

Resultados

El presente proyecto ha sido capaz de abordar con éxito las tres tareas principales que se había propuesto:

1. Adquirir información del entorno a través de un sensor láser activo.
2. Crear una representación tridimensional del mundo de forma rápida, flexible y versátil.
3. Extraer información semántica de la representación del mundo.

Como resultado, este proyecto es capaz de dotar a un robot de una representación tridimensional de su entorno. Mediante tal representación, es posible evitar obstáculos de forma más eficaz, al no estar restringida la detección a un solo plano. Además, es capaz de ubicarse a si mismo con más fiabilidad y puede detectar objetos de forma más robusta. Esto reporta beneficios en cualquier aplicación en la que los robots son empleados en escenarios reales. Además, los modelos tridimensionales del entorno son útilies en un gran rango de aplicaciones, más allá de la robótica, como la arquitectura, planificación en casos de emergencia, interacción y visualización de escenarios.

La elaboración del proyecto ha necesitado la aplicación de técnicas y conocimientos de Ingeniería del Software, Robótica, Sistemas Multimedia y Cálculo y Estadística aplicada a la Ingeniería. La primera dificultad que se venció fue el control y la integración de los dispositivos hardware, en concreto el sensor láser y el dispositivo apuntador. Se ha conseguido una gran estabilidad y precisión en la medida desplazando el dispositivo apuntador a la máxima velocidad que las características del sensor permiten. Ha sido necesario calibrar el sensor y caracterizar su precisión. El modelo final de desplazamiento-lectura del sensor se logró después de probar distintos esquemas. La solución obtenida permite realizar desplazamientos continuos del dispositivo apuntador al mismo tiempo que se sigue obteniendo medidas del sensor. Este método sacrifica algo de precisión en la localización espacial pero, como demostraremos, no resulta significativa incluso con grandes resoluciones y, a cambio, permite incrementar la velocidad de funcionamiento del sistema.

El proyecto se basa en distintos trabajos desarrollados por diferentes autores como se recoge en la bibliografía. La lectura de toda la documentación recopilada junto con la reflexión posterior

ha conseguido integrar distintas líneas, en principio independientes, y adaptar las metodologías ya propuestas para obtener una mejor respuesta en el entorno de este proyecto. De esta forma, se ha logrado aplicar las técnicas de detección de formas en nubes de puntos, [7], con la representación de múltiples superficies en un mapa de $2\frac{1}{2}$ dimensiones [4]. Para ello, se ha modificado el paradigma de extracción de formas en una nube de puntos a la extracción en una estructura de $2\frac{1}{2}$ dimensiones en la que los elementos base ya no son puntos, sino Gaussianos que representan superficies a distinto nivel.

Esquemáticamente, el sistema aquí desarrollado, puede verse en la figura **??**. Se inicia el proceso cuando se ordena al sistema hardware integrado, láser y dispositivo apuntador, realizar una medición de los objetos en su entorno. El dispositivo apuntador inicia un barrido durante el cual el sensor láser ,de forma sincronizada, va suministrando en forma de array las distancias de los objetos que rodean al sensor. Cada array de datos es procesado para convertir las medidas en coordenadas tridimensionales del mundo. Para poder realizar esta conversión se necesita conocer la odometría del robot. La siguiente estación en el sistema consiste en modificar el mapa 3D multisuperficie a partir de las coordenadas espaciales. Cada localización representa un impacto del haz láser con algún objeto del mundo. Estos impactos son procesadas por el método de integración de impactos que o bien creará una nueva estructura en el mapa o bien integrará el impacto en alguna ya creada. Por último, tenemos dos procesos que operan sobre el mapa multisuperficie: el método de obtención de planos y el método de proyección de mapa multisuperficies en un mapa de obstáculos 2D. El algoritmo que extrae planos es una modificación del algoritmo efficient-Ransac [7] que en vez de operar sobre una nube de puntos, trabaja sobre Gausianos. La proyección sobre el mapa 2D crea un mapa de obstáculos a nivel del suelo que va a permitir navegar al robot. Se necesita conocer el tamaño del robot para poder decidir si un objeto del mundo, por ejemplo el tablero de una mesa, constituye un obstáculo a la navegación o no.

7.1. Generación de coordenadas espaciales

En esta primera fase se necesita el concurso del sistema hardware formado por el sensor láser y el dispositivo apuntador. También se necesita conocer la odometría del robot sobre el que va montado el sistema sensor láser-dispositivo apuntador. Se pasa al sistema el ángulo de inclinación lateral,los ángulos de inclinación vertical inicial y final y la resolución de barrido deseada. El sistema, teniendo en cuenta que un barrido del láser tarda 100 ms., programa la velocidad de desplazamiento vertical del dispositivo apuntador.

El sensor proporciona cada 100 ms. un array con las mediciones obtenidas en el último barrido. Este es un array de enteros positivos en el que cada elemento representa en milímetros la distancia medida por el haz. Sólo se consideran medidas válidas las que estén a más de 20 milímetros que es la distancia mínima que puede medir el sensor. Valores menores que 20 indican que se ha producido algún tipo de error (ver **??**).

Se ha sometido al sensor a pruebas en entornos controlados. Mediante un software de visua-

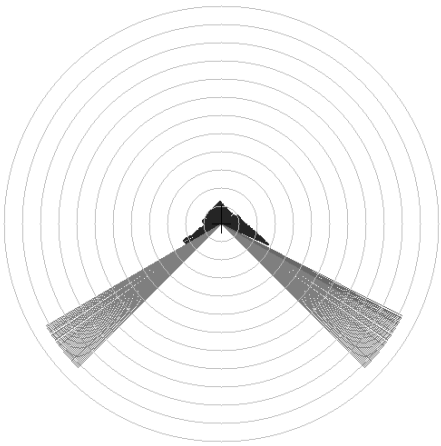


Figura 7.1: Visualización de la información devuelta por el sensor láser en un barrido del haz

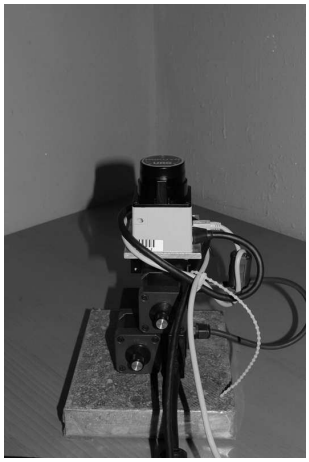


Figura 7.2: Esquina formada por una mesa y dos paredes

lización de la lectura del sensor, *urg-demo*, se contrasta la información obtenida por el sensor con el mundo real. El sensor láser realiza mediciones en un único plano (ver 3.1). En la figura 7.1 se muestra la salida de la aplicación *urg-demo* correspondiente al entorno mostrado en la figura 7.2.

Se ha enfocado el problema de obtener las coordenadas espaciales como un problema cinemático director. El modelo Denavit-Hartenberg realizado para nuestro sistema ha respondido con precisión a los problemas de obtención de las coordenadas espaciales del entorno. Para poder visualizar los resultados en tres dimensiones, este proyecto ha usado las librerías Open-GL. Vemos en las figuras 7.3 y 7.4 dos construcciones realizadas en los entornos reales que vemos en las figuras 7.5 y7.6. Las dos imágenes se han obtenido haciendo un barrido de 60º en la componente vertical, entre los -45º y los 15º, correspondiendo los 0º con la línea horizontal. La resolución que se ha utilizado ha sido de una medición del sensor cada 0'5º. Como se ha justificado en el capítulo anterior, la velocidad máxima de barrido del dispositivo viene limitada por la frecuencia de barrido del haz láser que

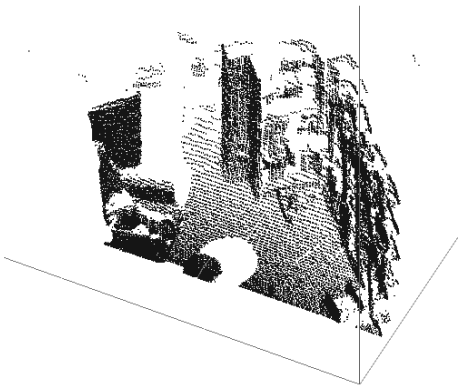


Figura 7.3: Vista tridimensional del mapa creado a partir de un escenario real(1)

Figura 7.4: Vista tridimensional del mapa creado a partir de un escenario real(2)

es de 10Hz. Por tanto, en barrer los 60º con una resolución seleccionada se tarda 12 segundos. Simultáneamente a la recolección de los datos suministrados por el sensor se va construyendo el mapa tridimensional, habiéndose conseguido unos tiempos de latencia en la construcción del mapa despreciables frente a los 100 milisegundos que tarda el sensor en hacer una rotación completa del haz, es decir, en adquirir un vector de medidas.

7.2. Resultados de la construcción de mapas 3D

Al mismo tiempo que se adquiere la información suministrada por el sensor, se construye la representación del mundo mediante un mapa de superficies multinivel. Este mapa permite detectar obstáculos (estructuras verticales) y superficies superpuestas a distintos niveles. Vemos en la figura 7.7 una representación del escenario mostrado en la figura 7.5. Se ha representado en color rojo las estructuras verticales, esto es, elementos del entorno que no son atravesables por pertenecer a

Figura 7.5: Escenario real para el mapa(1)

Figura 7.6: Escenario real para el mapa(2)

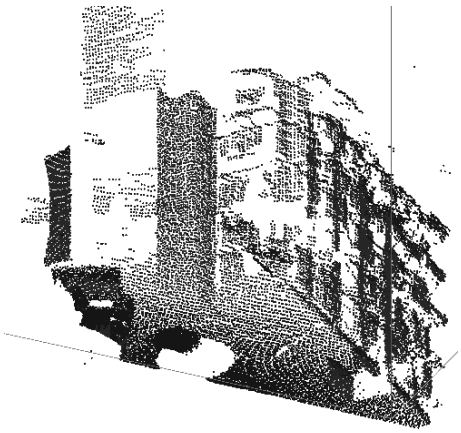


Figura 7.7: Visualización de un mapa multisuperficies. En color rojo las estructuras verticales y en azul las distintas superficies

una estructura mayor (p.e. parte de una pared). En color azul se han representado las superficies, las partes superiores de los distintos objetos que hay en el entorno. Para crear este mapa se ha utilizado un tamaño de discretización del mundo de 25 milímetros cuadrados.

Por último, este proyecto ha sido capaz de detectar estructuras de mayor nivel del mapa 3D. Para ello ha adaptado el algoritmo «Efficient-RANSAC», [7], al mapa multisuperfices implementado. Vemos en la figura 7.8 el resultado de aplicar el algoritmo de detección de formas al escenaria de la figura 7.2. El algoritmo busca en el mapa multisuperfices creado planos aplicando el método «Efficient-RANSAC» adaptado a tal tipo de mapas. Comprobamos en el escenario de la figura, formado por tres planos perpendiculares, que absolutamente todos los elementos del mapa son reconocidos como tres planos diferentes. Canda uno de estos planos se representa por un punto y un vector normal, lo que permite almacenar y procesar el escenario con gran economía de espacio.

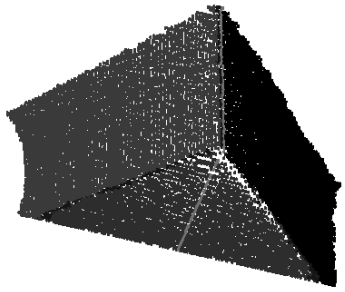


Figura 7.8: Estracción de distintos planos en un mapa multisuperficies. Cada plano detectado se representa en un color distinto.

Capítulo 8

Conclusiones y Trabajo Futuro

8.1. Conclusiones

En el presente proyecto hemos alcanzado los objetivos inicialmente propuestos con las restricciones de hardware impuestas. Los principales aportaciones del mismo son:

- Hemos conseguido un sistema capaz de generar mapas tridimensionales de su entorno a partir de un sensor láser plano y de un dispositivo apuntador con dos grados de libertad.

- En estre proyecto se consigue una velocidad máxima de funcionamiento del támdem sensor láser - dispositivo apuntador. El esquema de accionamiento del hardware, basado en el desplazamiento contínuo del dispositivo apuntador mientras se siguen recolectando datos del láser, introduce un error en la estimación de la posición del láser que ha sido acotado y que se demuestra no sersignificativo en la construcción del mapa que representa al mundo. A cambio, se consigue un tiempo de escaneado mínimo para una resolución dada.

- La conversión de las medidas proporcionadas con el sensor láser en coordenadas espaciales del mundo se conseguido con éxito, aplicando el algoritmo de Denavit y Hartenberg. Este algoritmo, pensado para resolver el problema cinemático directo, se ha aplicado a nuestro problema considerando al haz láser como un eslabón más de la cadena cinemática.

- La construcción y actualización del mapa 3D se ha hecho empleando una representación, los mapas multinivel, compacta y versátil. Los algoritmos que aquí hemos desarrollado para llevar a cabo las tareas de construcción y actualización del mapa tienen tiempos de procesamiento muy inferiores a la latencia del láser. Sumados los tiempos de obtención de coordenadas espaciales con los de actualización del mapa 3D, solamente representan en promedio un 9,5 % del tiempo de rotación del haz láser. Esto deja un gran margen para que futuras ampliaciones en el procesamiento del mapa tengan lugar en paralelo con la adquisición de información. Por otra parte, la representación es lo suficientemente compacta para que equipos dotados con 1 Gb. de memoria RAM puedan procesar de forma eficiente los algoritmos.

- Se han diseñado distintos algoritmos para extraer información del mapa 3D. Actualmente el sistema posee la capacidad de generar mapas 2D de obstáculos, en función de la altura del robot o dispositivo en que esté colocado el SLDA. //

  Nuestro sistema también es capaz de detectar planos en el mapa 3D. Para ello se ha creado un algoritmo capaz de detectar planos en un mapa multinivel en el cual los objetos contenidos en el mapa son Gaussianos que representan superficies. El algoritmo desarrollado se ha basado en el algoritmo *efficient RANSAC*, diseñado originalmente para detectar formas en nubes de puntos. Además de adaptar el algoritmo para que en vez de trabajar en nubes de puntos trabaje en mapas de Gaussianos, se han introducido una variante en el algoritmo que disminuye notablemente el tiempo de procesamiento necesario para detectar planos. El algoritmo *efficient RANSAC*, cada vez que se genera una nueva forma candidata, testea la probabilidad de que haber obtenido la superficie de forma azaroza sea inferior a una probabilidad dada. Para ello calcula la probabilidad frente a todos los puntos que existen en el mapa, por lo que requiere que se haya generado un alto número de superficies candidatas antes de dar a una de ellas por válida. Sin embargo, en nuestro algoritmo sacamos provecho de que un plano candidato sólo se genera a partir de bloques con una orientación similar. Por tanto, cuando testeamos la probabilidad de que dicho plano no se haya generado por azar, lo hacemos respecto al total de bloques con una orientación similar al plano. Esto permite extraer más rápidamente y en consecuencia una mayor cantidad de planos que el algoritmo *efficient RANSAC*.

8.2. Trabajo Futuro

El presente proyecto no se acaba en si mismo, sino que permite continuar ampliando funcionalidades. Consideramos que los aspectos más interesantes a desarrollar son:

- Adquirir información con el sistema en movimiento. Actualmente, el sistema está diseñado de forma que el robot o dispositivo en el que vaya integrado se encuentre en reposo durante la toma de información por parte del dispositivo apuntador y el sensor láser. El diseñar un sistema que permita obtener medidas en movimiento lo haría más eficiente. Puede servir de referencia trabajos como el que se muestra en [18]

- Desarrollar algoritmos para tratar la realidad dinámica de forma eficiente. Los algoritmos desarrollados siguen el trazado de cada haz del láser para eliminar los objetos que anteriormente se encontraran en dicha trayectoria. Este método, sin embargo, deja «residuos» o bien, debido a la discretización del mundo, elimina información de objetos que no se han movido.

- Integración de un sistema de generación de mapas 3d a partir de un sensor láser activo en un robot real. En un futuro se deberán resolver los problemas de SLAM y la forma de integrar en un único mapa coherente la información que se vaya obteniendo desde distintas posiciones y orientaciones.

- Uso de «CoolBot» como marco de integración. El uso de un marco de desarrollo como «CoolBot», diseñado en el seno del SIANI, permitirá resolver de forma sencilla la integración

de nuestro generador de mapas 3D con distintos sistemas robóticos. Esto permitirá usar la proyección 2D del mapa de superficies multinivel para proveer de datos a un algoritmo de evitación de obstáculos.

- Generación de información con alto nivel semántico: reconocimiento y detección de paredes, techos, suelos, mesas, etc. A partir de la información que ya proporciona este proyecto, se podrá ir extrayendo estructuras de mayor nivel semántico que ayuden en el reconocimiento de objetos y en la tarea de localización.

- Segmentación coherente del espacio en módulos (habitaciones, pasillos, etc.) conectados topológicamente. La navegabilidad se mejorará con la identificación de estructuras de segmentanción del espacio. Es de evidente ayuda en la navegación en interiores de edificios.

- Identificación y seguimiento a lo largo del tiempo de zonas de especial interés para la navegación: esquinas, escaleras, rampas, etc.

Apéndice A

Detalles técnicos sobre la implementación del proyecto

En las secciones anteriores del proyecto, no se debe entrar en demasiados detalles técnicos sobre cuestiones de implementación del proyecto pues dificultaría su lectura y comprensión. Los detalles técnicos sobre la implementación del proyecto se incluyen preferentemente en apéndices al final de la memoria.

Apéndice B

Tests realizados al hardware

Se ha querido comprobar el funcionamiento del sensor láser. Para ello se creó un simple test para caracterizar el funcionamiento de este dispositivo. Este test permitión detectar un malfuncionamiento en el primer sensor laser que se probó en este proyecto. Un segundo sensor fue sometido al mismo test arrojando unos resultados correctos. Describimos a continuación el test.

Colocamos al sensor a una distancia de 25 cm. frente a una pared en tres posiciones diferentes. En la primera, enfrentamos el ángulo 0º del sensor a la pared y medimos la respuesta del sensor en el rango de -40º a 40º. En segundo lugar orientamos la línea de -90º frente a la pared y medimos la respuesta del sensor en la zona desde los -120º y los -40º. Finalmente, enfrentamos la línea de 90º del sensor con la pared y medimos la respuesta en la zona desde los 40º a los 120º.

En cada posición se realizan quinientas lecturas del sensor. Hemos estudiado en cada uno de los tres casos el número de lecturas correctas en cada paso y la desviación estándar de las medidas efectuadas en cada paso. La desviación estándar se ha obtenido entre las mediciones correctas correspondientes a un determinado ángulo. Si el número de mediciones correctas en un determinado ángulo es cero, en la gráfica de desviación estándar correspondiente aparece un cero aunque esto no significa realmente homogeneidad en la respuesta del sensor, sino ausencia de datos válidos. Podemos ver en las figuras B.1 y B.2 los resultados obtenidos con el primer sensor láser.

El análisis de los datos permite extraer dos conclusiones prácticas. La primera es que el rango de medida útil del sensor no va desde los -120º a los 120º, como especifican las referencias del fabricante, sino desde los -80º a los 80º. Fuera de esta zona el número de mediciones erróneas crece rápidamente hasta ser de prácticamente el 100 % por debajo de los -90º y por encima de los 90º.

Una segunda conclusión afecta al error en la medida. El fabricante indica que objetos a menos de 1000 mm tienen un error esperado en la medida de ± 10 mm. Como vemos en la figura VII.5, en la gráfica central, la desviación estándar es bastante menor de 10 mm. Concretamente, la media de las desviaciones estándar para el rango de -40º a 40º fue de 2.43 mm. Es decir, en la práctica, el promedio de error esperado es de ±2.43 mm. en dicho rango.

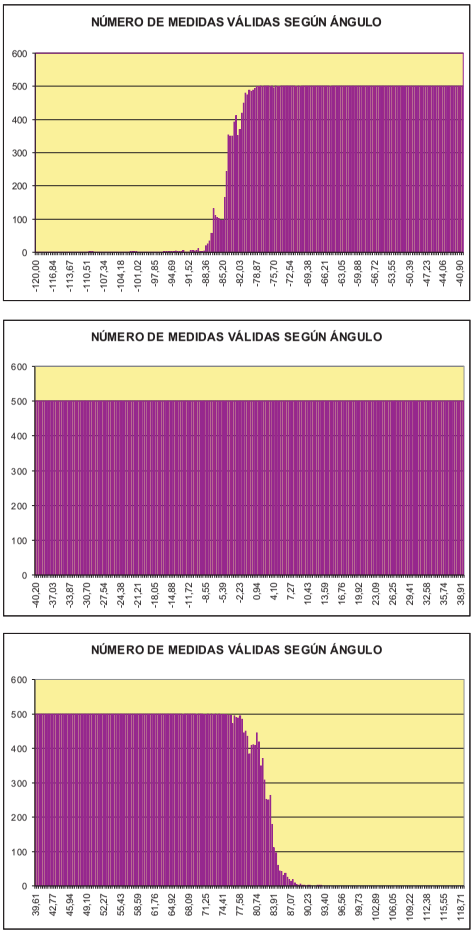


Figura B.1: Sensor nº 1. Número de medidas correctas en la zona 1: -120º a -40º, zona 2: -40º a 40º y zona3: 40º a 120º

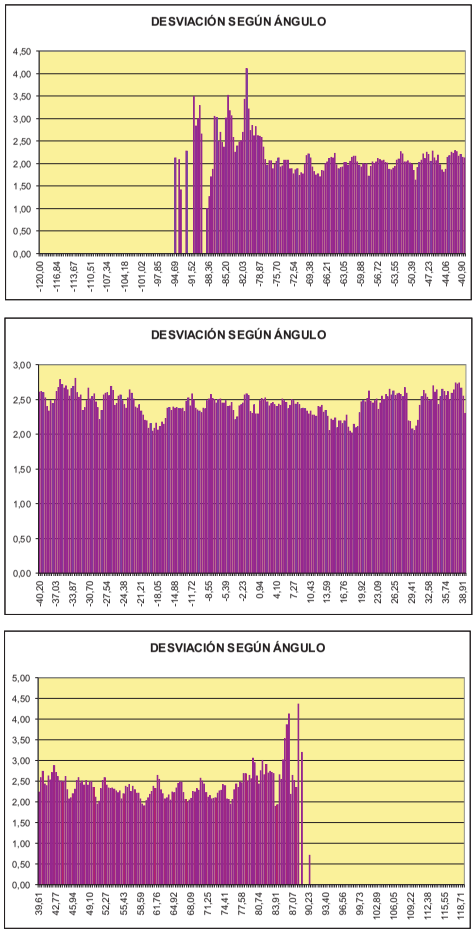


Figura B.2: Sensor nº 1. Desviación estándar de 500 mediciones por ángulo

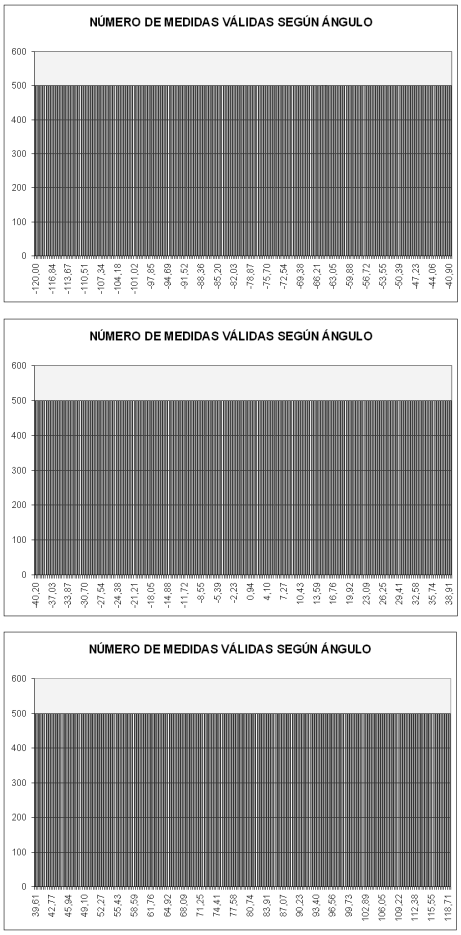


Figura B.3: Sensor nº 2. Número de medidas correctas en la zona 1: -120º a -40º, zona 2: -40º a 40º y zona3: 40º a 120º

Los resultados obtenidos con el segundo sensor se muestran en las figuras B.3 y B.4. Observamos un comportamiento correcto en todo el rango del sensor. El promedio de error es similar al caso anterior.

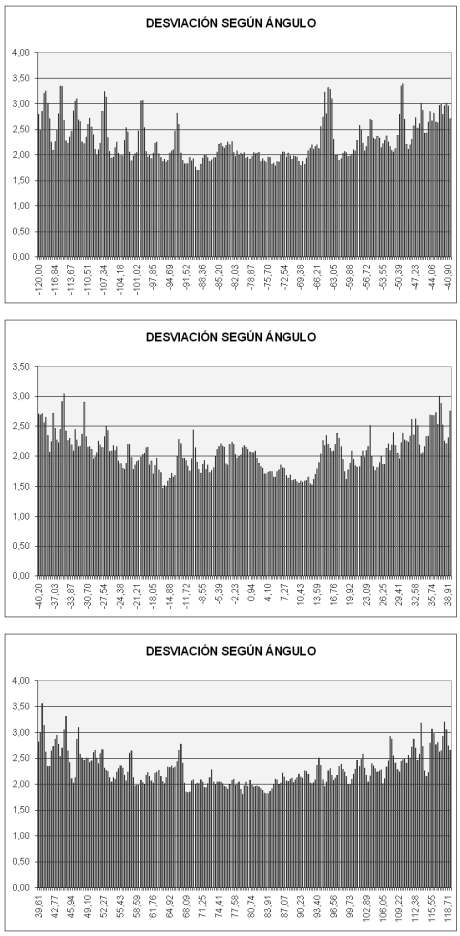


Figura B.4: Sensor nº 2. Desviación estándar de 500 mediciones por ángulo

Bibliografía

[1] Journal of field robotics. http://www.journalfieldrobotics.org/.

[2] Andreas Nüchter. *3D Robotic Mapping*. Springer, 2009.

[3] P. Pfaff; R. Triebel; W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 2007.

[4] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[5] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[6] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972.

[7] Klein R. Schnabel R., Wahl R. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26:214–226, 2007.

[8] Grupo de Ingeniería del software y del conocimiento. Unidades temáticas de Ingeniería del Software.

[9] Roger S. Pressman. *Ingeniería del Sfotware. Un enfoque práctico*. Mc Graw Hill, 2002.

[10] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Mag.*, 9(2):61–74, 1988.

[11] Directed Perception. *Specifications of the PTU-D46*.

[12] Directed Perception. *Pan-Tilt Unit, Command Reference Manual*.

[13] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans ASME J. Appl. Mech*, 23:215–221, 1955.

[14] Sebastian Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[15] H. Moravec and A. E. Elfes. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, pages 116–121, March 1985.

[16] Hugues Hoppe, Tony Derose, Tom Duchamp, John Mcdonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26:71–82, 1992.

[17] Lindsay I. Smith. A tutorial on principal components analysis, 2002.

[18] Alastair Harrison & Paul Newman. High quality 3d laser ranging under general vehicle motion.