Técnicas de Navegación de Robots Basadas en Sistemas de Medición por Láser

Departamento de Informática y Automática
Universidad de Salamanca

Carlos Fernández Caramés

Septiembre, 2007

Don Vidal Moreno Rodilla y Doña Belén Curto Diego, profesores del Departamento de Informática y Automática de la Universidad de Salamanca,

CERTIFICAN:

Que el trabajo titulado "Técnicas de navegación de robots basadas en sistemas de medición por láser" ha sido realizado por Carlos Fernández Caramés, y constituye la memoria para optar al título de Grado por la Universidad de Salamanca.

En Salamanca, a 3 de Septiembre de 2007

D. Vidal Moreno Rodilla	Dª. Belén Curto Diego
Dpto. Informática y Automática Universidad de Salamanca	Dpto Informática y Automática Universidad de Salamanca

Agradecimientos

Quisiera agradecer a mis tutores, Vidal Moreno Rodilla y Belén Curto Diego, su colaboración y ayuda. También deseo agradecer a J. Andrés Vicente Lober su asesoramiento en la construcción del robot móvil. Por último, debo también dar las gracias a todos aquellos que me han apoyado durante la duración de este trabajo.

A Noelia, por estar
siempre a mi lado.

Índice general

Índice de figuras

1 Introducción

A lo largo de los últimos siglos el hombre ha inventado y perfeccionado ingenios como el avión, el automóvil o las computadoras. Una de las máquinas que siempre se ha pensado que el futuro nos tiene reservado es un robot que pueda evitarnos la realización de tareas repetitivas, pesadas o peligrosas. Todavía no se ha alcanzado ese punto, pero se está trabajando para que algún día el futuro que imaginamos sea realidad, y los robots puedan proporcionar cualquier tipo de ayuda a quien lo necesite. Hay innumerables aplicaciones que se pueden pensar para un robot, como por ejemplo las tareas domésticas, la construcción o el trabajo en entornos peligrosos.

En la actualidad es en los entornos industriales donde la robótica ha alcanzado su mayor grado de implantación. Tanto es así, que los brazos robóticos, o *manipuladores*, mueven una industria de miles de millones de euros al año [35]. Fijados a un punto específico en las cadenas de montaje industriales, los brazos robóticos (fig. 1.1) pueden moverse con velocidad y precisión sobrehumanas, y realizar tareas repetitivas sin descanso, como soldar, pintar, taladrar, o ensamblar componentes. Gracias a este notable desarrollo robótico, en la industria ha sido posible automatizar la producción en cadena de automóviles, electrodomésticos, o incluso brazos robóticos, paradójicamente.

Sin embargo, a pesar de su gran éxito en la industria, estos robots comerciales presentan una desventaja fundamental: la falta de movilidad. Los movimientos del manipulador están ligados al punto exacto en que esté fijado dentro de la cadena de montaje. En contraste, un robot móvil sería capaz de

Figura 1.1: *Robots soldadores en la planta de ensamblaje de Chrysler en Windsor, Canadá.*

viajar a través de toda una planta de manufacturación, aplicando sus funciones donde se necesitasen. Además de la movilidad, una de las principales diferencias entre estos dos tipos de robots es el nivel de autonomía e inteligencia. Normalmente un brazo robótico está preprogramado para soldar y ésa es la única tarea que podrá desempeñar. Por el contrario, un robot móvil debe tener, por lo general, un mayor grado de autonomía, al tener capacidad para resolver situaciones desconocidas que pueden suceder mientras se desplaza. Algunos entornos industriales están bastante estructurados, y son bastante estáticos, por lo que los imprevistos para un robot móvil podrían ser mínimos. No obstante, otros entornos como el campus de una facultad, o una vivienda, son mucho más dinámicos, ya que siempre suele haber gente moviéndose de un lado para otro, y los objetos pueden cambiar de lugar sin previo aviso. Esto significa que el robot necesita percibir el entorno para poder reaccionar ante determinadas circunstancias, lo que se traduce en la necesidad de mecanismos de percepción exteroceptivos.

Existen multitud de problemas que necesitan resolverse antes de poder disponer de robots que puedan realizar algunas tareas del mismo modo que lo haría una persona cualquiera. Muchos de estos problemas están relacionados con las capacidades innatas de todo ser humano. Es evidente que para que sea capaz de planchar se debe enseñar al robot dónde se encuentran la plancha y las camisas, o cómo obtener los ingredientes necesarios de la despensa para preparar una receta de cocina. Antes de afrontar problemas tan complejos, hay otros mucho más básicos que se deben solucionar. Dos de

los más importantes son el problema de navegación y dentro de éste, el de localización [20], para poder disponer de movilidad autónoma.

El problema de navegación

Murphy [26] plantea que el problema de navegación se puede resolver respondiendo a cuatro preguntas: "¿a dónde voy?", "¿cuál es el mejor camino para ir?", "¿dónde he estado?" y "¿dónde estoy?". La respuesta a la primera pregunta, es decir, la especificación del destino del robot, normalmente es decidida por un humano o por un subsistema inteligente del robot conocido como *planificador de misiones*. Algunos investigadores no incluyen esta tarea como parte del problema de navegación, al considerarla como una etapa previa. La segunda pregunta versa sobre cómo planear un camino cuyo seguimiento permita alcanzar el destino especificado, y es uno de los subproblemas de navegación que mayor interés ha suscitado. Los métodos de planificación de caminos se pueden dividir a su vez en dos categorías: cualitativos (o topológicos) y cuantitativos (o métricos). La tercera pregunta trata sobre la construcción de mapas. A medida que el robot explora nuevos entornos, puede que parte de su misión sea construir un mapa de las zonas que visita. Incluso si el entorno del robot es siempre el mismo y es bien conocido, cabe la posibilidad de que algunos muebles cambien de sitio, o se construyan nuevas paredes. Si el robot detecta estos cambios en el entorno y los añade a su mapa, puede mejorar significativamente su rendimiento. Finalmente, la última pregunta se refiere a la localización: cómo averiguar la ubicación del robot basándose en las percepciones y en la información adquirida previamente. Conocer la posición es indispensable para construir un mapa o para seguir una trayectoria planificada en algunos ámbitos.

La navegación, hoy en día, es un problema que no tiene secretos para el hombre, y que se utiliza de manera rutinaria en sistemas marítimos y de aviación. En cambio, para los robots móviles sigue siendo un problema complicado. Esto se debe, entre otras cosas, a la dificultad de extraer información fiable del entorno a partir de datos sensoriales, y a la complejidad de establecer una correlación entre los datos obtenidos y el mapa de navegación que haya construido el robot hasta el momento.

El problema de navegación se ha logrado resolver satisfactoriamente si el entorno de trabajo de un robot se prepara con anterioridad. Un procedimiento ampliamente empleado en la actualidad en robots móviles que trabajan en una planta industrial es construir un sistema de navegación utilizando balizas que los robots puedan detectar. Las balizas pueden ser, por ejemplo, códigos de barras ubicados en lugares fijos del entorno, de modo que al detectar un determinado código, el robot lo compare con su base de datos de posiciones,

y automáticamente pase a conocer con exactitud su emplazamiento. Aunque existen multitud de soluciones comerciales ya implantadas en entornos industriales que utilizan este procedimiento, sigue quedando sin resolver el problema de cómo navegar en entornos abiertos, es decir, en entornos que no se hayan preparado mediante balizas artificiales que faciliten la navegación del robot.

Como se ha comentado anteriormente, para poder navegar de manera efectiva los robots necesitan información de su posición y orientación en el mundo (problema de localización). Existen dos principales modos de expresar esta información. Una opción es calcular la ubicación en relación a un sistema global de coordenadas, mientras que la otra es calcular el desplazamiento relativo con respecto a algún objeto, o respecto de sí mismo. Normalmente se necesita una combinación de ambas, puesto que el robot necesitará conocer su posición relativa a un objeto contra el que ha colisionado, por ejemplo, y por otra parte deberá conocer su posición global cuando tenga que planificar cómo ir de un sitio a otro.

Construcción de mapas y extracción de características

La localización es un problema estrechamente relacionado con la construcción de mapas. Tanto es así, que si un robot está explorando un entorno desconocido y desea mantener información actualizada sobre su posición, entonces es imprescindible que vaya construyendo un mapa de su entorno al mismo tiempo que descubre nuevas zonas. Este problema se conoce a menudo como SLAM, siglas en inglés de *Simultaneous Localization and Mapping* (localización y construcción de mapas simultánea).

De todo lo anterior se puede concluir que para poder localizarse y navegar es necesario construir un modelo o mapa del entorno, o disponer de uno *a priori*. Uno de los métodos más habituales para la construcción de mapas es la extracción de características. ¿Qué es una característica? Se puede considerar que, dentro de un determinado entorno, una característica es un objeto físico estático, perceptible, y que pueda tener una descripción abstracta (esquina, puerta, ...) [2]. Si una característica va a resultar útil o no, es difícil establecerlo de antemano, puesto que generalmente depende del contexto en el que se encuentre y del sistema de percepción que el robot utilice. Las características contienen información semántica y métrica. La primera se refiere al tipo de característica, mientras que la información métrica indica la dimensión, color, textura, etc. En la mayoría de entornos del mundo real existen características distinguibles que son muy útiles para la navegación y se pueden extraer de los datos obtenidos por los sensores del robot. Las características que tienen una geometría simple suelen ser una buena elección,

ya que son relativamente fáciles de detectar, y son frecuentes en los entornos artificiales construidos por el hombre.

El uso de características en robótica móvil comenzó en torno a 1990. Uno de los trabajos pioneros al respecto fue la extracción de líneas mediante sensores de ultrasonidos [22]. Los modelos más empleados para describir características suelen ser líneas, segmentos, esquinas, bordes y curvas. Además de los típicos ejemplos de características de entornos interiores, como puertas, esquinas, o paredes, también es posible utilizar características de entornos exteriores. Algunos ejemplos son rocas y salientes extraídos mediante un láser de medición de distancias en 3D y modelados mediante elipsoides [5], o plantas extraídas y clasificadas a partir del perfil de densidad acústica de sensores ultrasónicos [18].

Otra técnica habitual para la construcción de mapas es emplear directamente los datos sin tratamiento previo, es decir, sin extraer ningún tipo de característica. Algunos ejemplos de técnicas que funcionan de este modo son: localización Markov [16], comparación de barridos de escáner láser (*scan matching*) [23] y localización basada en filtros de partículas [10]. La ventaja de este enfoque está en que funciona en entornos que no dispongan de características identificables. Por desgracia, esta ventaja se transforma en inconveniente, puesto que los datos sin tratamiento suelen plantear problemas de eficiencia, al crecer desmesuradamente a medida que aumenta el tamaño del entorno. Por el contrario, el enfoque basado en características proporciona descripciones compactas del mundo, y permite mantener la escalabilidad incluso cuando el entorno tratado sea de dimensiones extensas.

Quedan así planteadas las técnicas de extracción de características como una de las técnicas básicas necesarias para la navegación, y dentro de ésta, para la localización, y la construcción de mapas. No cabe ninguna duda de que las técnicas de extracción de características están ligadas al tipo y número de sensores que se empleen como mecanismo de percepción. Por consiguiente, para comprender con mayor profundidad los problemas de extracción de características, construcción de mapas, localización, o navegación en términos generales, es importante conocer en primer lugar los distintos tipos de sensores que se pueden utilizar para dichas tareas. Por este motivo se ha incluido un capítulo dentro de esta memoria de Grado que resume los principales sensores empleados en la navegación.

Enfoque del trabajo

Entre los múltiples sensores disponibles para la navegación, este trabajo se va a centrar en el uso de un sensor láser de medición de distancias (ver sección 2.2.3), el cual se empleará para la extracción de características de en-

tornos tanto interiores como exteriores. Como se ha comentado brevemente en párrafos anteriores, existen una variedad de características que se pueden detectar, y ante esta diversidad, el foco de este trabajo se dirigirá hacia las técnicas de extracción de líneas. Cabe destacar que de manera previa a la extracción de líneas es habitual dedicar una etapa al *clustering* (o agrupamiento) de puntos de los datos proporcionados por el láser [8], con el fin de facilitar la tarea de los detectores de líneas. De este modo los esfuerzos de investigación de este trabajo se dirigirán tanto hacia los algoritmos de *clustering* como hacia los algoritmos de extracción de líneas.

1.1. Objetivos del trabajo

La idea base de la que parte este trabajo es que las técnicas de extracción de características constituyen los cimientos sobre los que se erige la construcción de mapas, que a su vez es una parte imprescindible en los problemas de localización y navegación. Los objetivos concretos que se persiguen con la realización de este Grado de Salamanca son los siguientes:

- **Análisis bibliográfico:** se propone realizar un estudio de las técnicas existentes para la extracción de características, siempre y cuando los métodos estén basados, al menos en parte, en la utilización de un escáner láser de medición de distancias.
- **Propuesta de nuevas técnicas de extracción de características:** tratar de desarrollar técnicas innovadoras que mejoren aspectos deficientes de los métodos analizados. Adicionalmente se efectuará un análisis comparativo de diferentes aspectos de rendimiento entre los nuevos métodos propuestos y los existentes.
- **Construcción de una plataforma robótica de validación:** diseñar y construir una plataforma que permita explorar las capacidades de las soluciones planteadas. Dicha plataforma incluirá el escáner láser, además de los elementos de comunicación y procesamiento necesarios.
- **Implementación de software de validación:** desarrollar una herramienta gráfica, junto con un conjunto de utilidades para analizar las técnicas existentes y validar los nuevos métodos propuestos. Del mismo modo servirá para analizar la corrección, robustez, y el rendimiento de los métodos, con el fin de establecer una comparativa.

El resto del trabajo está organizado del siguiente modo: en el capítulo 2 se analizarán los sensores más habituales empleados para la navegación,

dedicando un especial interés hacia el sensor láser utilizado en este trabajo. Los capítulos 3 y 4 se refieren al análisis bibliográfico de antecedentes con respecto a la extracción de características. Concretamente, el capítulo 3 revisará algoritmos de *clustering*, que en la mayoría de los casos se considera como una etapa previa a las técnicas de extracción de líneas, cuyo análisis se abordará en el capítulo cuarto. A continuación se dedicará el capítulo quinto para exponer los nuevos métodos desarrollados. El capítulo 6 se reservará para describir la plataforma robótica y el software de validación implementado. Para finalizar, el capítulo séptimo planteará una comparativa de los algoritmos analizados en este trabajo, mientras que en el octavo y último capítulo se presentarán las conclusiones.

2

Sensores para la navegación

Para poder navegar por cualquier tipo de entorno, los robots necesitan sensores que les permitan percibir el medio que los rodea. Al igual que los humanos tenemos diferentes sentidos, los robots se pueden equipar con gran variedad de sensores que hoy en día son capaces de medir estímulos muy variados: color, distancia, fuerza, presión, inclinación, etc.

Los sensores se pueden agrupar de diversas maneras. Una de las clasificaciones más habituales consiste en separarlos en propioceptivos y exteroceptivos. Los del primer tipo sirven para observar el estado interno del robot, como la orientación de un brazo robótico, o el nivel de carga de las baterías. Los sensores exteroceptivos miden estímulos que se originan en el exterior del robot, como distancias a objetos, o la temperatura ambiente, y su función es proporcionar al robot una representación del entorno en el que se encuentra.

Es importante tener un buen conocimiento del modo de funcionamiento de los sensores, ya que de este modo se logrará aprovechar al máximo las prestaciones que ofrecen. A continuación se expondrán brevemente algunos de los sensores más empleados para la navegación.

2.1. Sensores Propioceptivos

En esta sección se presentarán algunos de los sensores propioceptivos más comunes en la navegación, como por ejemplo, los *encoders* rotatorios,

los acelerómetros, y los giróscopos. Mediante los primeros se puede obtener una estimación de la velocidad y la distancia viajada, mientras que los dos últimos proporcionan información sobre aceleración y rotación, y a menudo se utilizan conjuntamente formando los denominados "Sistemas de Navegación Inercial" o INS (*Inertial Navigation Systems*). La principal ventaja de los sensores propioceptivos, en términos generales, es que no dependen de estímulos externos, ya que sólo miden estados internos del robot. Como inconveniente, resulta complicado efectuar estimaciones fiables, ya que los errores se acumulan continuamente, al no disponer de información externa.

2.1.1. *Encoders* rotatorios

En robótica, los *encoders* se utilizan habitualmente para llevar a cabo tareas odométricas. La odometría es una técnica que permite estimar la posición de un robot móvil con respecto a una posición inicial conocida, basándose en el número de revoluciones de cada rueda del robot, y en la dirección en que se mueven las ruedas. Los *encoders* más empleados son los rotatorios (o de eje), que son dispositivos electromecánicos que convierten la posición angular de un eje en un código digital. Dependiendo de si el código que se genera para cada posición del eje es único o no, el *encoder* será absoluto o relativo, respectivamente. Los *encoders* pueden ser magnéticos, mecánicos y de otros tipos, si bien los más comunes son los ópticos.

En la fig. 2.1a se muestra el esquema general de un *encoder* óptico rotatorio, de tipo relativo (o incremental). Está formado por un disco con agujeros (o ranuras) cerca de su borde, y se coloca de modo que el eje del disco coincida con el eje del motor y el de la rueda (normalmente son el mismo). El número de agujeros en el disco determina la precisión del *encoder*. A un lado del disco se coloca un LED infrarrojo, mientras que al otro lado, y enfrente del LED emisor, se coloca un fototransistor receptor de infrarrojos. Cuando el disco gira, los agujeros hacen que el fototransistor reciba luz de manera intermitente, generándose así una secuencia de pulsos como la que se muestra en la fig. 2.1a . Los pulsos detectados por el fototransistor se cuentan, y se convierten en distancia con un cálculo basado en el diámetro de la rueda.

Los *encoders* de tipo absoluto son más apropiados para casos en los que se producen rotaciones más lentas, o poco frecuentes, como por ejemplo averiguar la rotación de un volante en un vehículo automatizado. En la fig. 2.1b se ilustra el aspecto de los discos típicamente empleados en los *encoders* absolutos. El disco está formado por patrones codificados de zonas opacas y zonas transparentes. El funcionamiento es similar a los *encoders* relativos, pero en lugar de utilizarse un único LED y un único fotorreceptor, se utilizan varios. En el caso de la figura se necesitan 8 LEDs a un lado del disco y

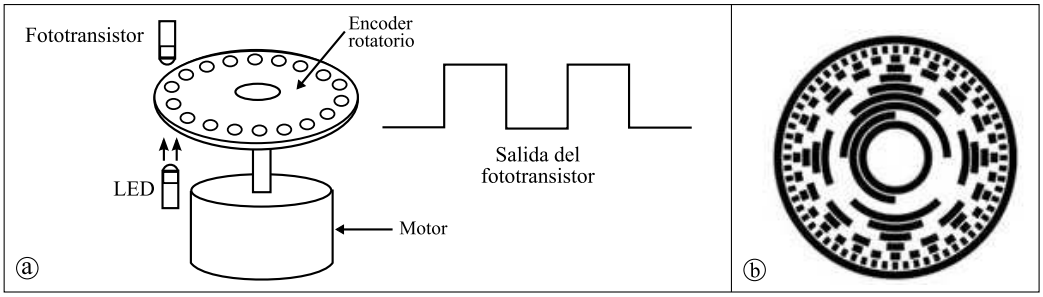


Figura 2.1: *a)Encoder óptico rotatorio unido a un motor. b)Disco de un encoder absoluto.*

8 fotorreceptores al otro lado. La luz sólo atraviesa las zonas transparentes, y así se obtiene la posición absoluta de giro.

2.1.2. Acelerómetros

Los acelerómetros son dispositivos que detectan cambios en la velocidad. La mayoría no están preparados para medir velocidad constante, sino que solamente miden aceleración o deceleración. En un principio estaban restringidos al ámbito científico e industrial, debido a su alto coste. Sin embargo, gracias a su abaratamiento, cada vez se encuentran más presentes en aparatos de uso cotidiano, como ordenadores portátiles (que se suspenden en caso de caídas), mandos de consolas de videojuegos, o incluso teléfonos móviles.

Dentro de la robótica móvil, uno de los principales usos de un acelerómetro es la detección de movimiento. Para realizar esta tarea, es habitual utilizar *encoders* en las ruedas del robot, con el inconveniente de que no se puede detectar movimiento al producirse derrapes, ya que las ruedas no giran. Los acelerómetros presentan la ventaja de que pueden detectar desplazamientos del robot incluso cuando las ruedas del robot están detenidas. Por ello, se pueden emplear en conjunto con los *encoders* para obtener una mayor fiabilidad al calcular el desplazamiento de un robot. Otros posibles usos del acelerómetro son la detección de colisiones (actuando como un *bumper*) o la teleoperación robótica. Un interesante ejemplo de esto último consistiría en colocar acelerómetros en las extremidades inferiores de una persona, para detectar cuando camina, y replicar estos movimientos en un robot bípedo.

A pesar de que todos los acelerómetros tienen el mismo fin, pueden ser muy distintos unos de otros. Dependiendo de la tecnología empleada para su construcción, sus características y precisión serán diferentes. Existe una gran diversidad de acelerómetros: piezo-eléctricos, piezo-resistivos, térmicos, ópticos, capacitivos, basados en inducción magnética, etc. Según la aplicación a la que van a ser destinados y las condiciones en las que han de trabajar, se

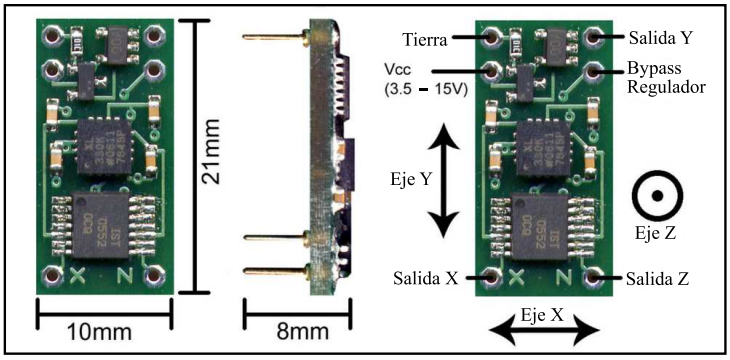


Figura 2.2: *Acelerómetro de 3 ejes (X,Y,Z) de Dimension Engineering, modelo DE-ACCM3D.*

deberá elegir el tipo de acelerómetro adecuado. Por otro lado, también es importante decidir el número de dimensiones en las que se desea medir cambios de velocidad. Los acelerómetros más básicos detectan aceleraciones o deceleraciones en una sola dimensión, mientras que existen otros más complejos en dos o tres dimensiones. Es posible combinar acelerómetros de una sola dimensión para obtener un efecto tridimensional. En la fig. 2.2 se muestra un ejemplo de acelerómetro en 3 dimensiones.

Algunas desventajas de los acelerómetros es que detectan con dificultad las aceleraciones de pequeña magnitud (como por ejemplo al realizar giros muy lentos) y que son muy sensibles a irregularidades en el suelo [6].

2.1.3. Giróscopos

Los giróscopos son dispositivos que sirven para medir o para mantener la orientación en relación a un sistema de referencia fijo. Actualmente es un aparato empleado como estabilizador de embarcaciones, de telescopios espaciales, o como piloto automático de aviones, entre otras muchas utilidades. En robótica se suele usar en conjunción con los acelerómetros, para detectar cambios en la orientación de los robots. De manera general, los giróscopos se pueden clasificar en dos categorías: mecánicos y ópticos.

El primer giróscopo mecánico fue construido por *Johann Bohnenberger* en 1817, pero no recibió su nombre actual hasta 1852, año en que *León Foucault* utilizó el aparato para demostrar el movimiento de rotación de la Tierra sobre su propio eje. Un giróscopo consiste básicamente en un sólido rígido (normalmente un disco) montado sobre un eje principal de inercia, alrededor del cual el disco gira constantemente a gran velocidad gracias a un motor. Al girar rápidamente el disco, su eje de rotación tiende a mantenerse siempre en la misma dirección gracias al principio de conservación del momento an-

gular. Esta propiedad se conoce como inercia giroscópica, y es directamente proporcional al radio, la masa, y la velocidad angular del disco en rotación. Esto se debe a que el momento angular de un sólido rígido, para un objeto de masa fija y que se encuentra rotando alrededor de un eje de simetría, vale:

$$L = I \cdot \omega$$

donde L es el momento angular, I el momento de inercia, y ω la velocidad angular del sólido. Otra importante característica del giróscopo, denominada precesión, consiste en que si se aplica una fuerza sobre el giróscopo que dé lugar a un par de giro, el eje de giro se moverá en una dirección perpendicular a la fuerza aplicada. Esto resulta bastante sorprendente, ya que el eje de giro no se desplaza en la dirección esperada intuitivamente.

Para poder utilizar un giróscopo como indicador de giros se utiliza la propiedad de inercia giroscópica. En ausencia de momentos de fuerza externos, el eje de rotación del giróscopo permanece inmóvil, por lo que para poder aprovechar esta característica, es necesario preparar un sistema que aisle al disco en rotación de momentos externos. El sistema de anillas en suspensión Cardán está diseñado específicamente para este propósito (ver fig. 2.3), ya que los soportes que sujetan el disco tienen un coeficiente de rozamiento muy pequeño, permitiendo al disco conservar su momento angular. La suspensión Cardán consiste en dos o más soportes que pueden rotar en una sola dirección, y cuyos ejes de giro son ortogonales entre ellos. Si el disco está rotando a gran velocidad los soportes podrán rotar sin afectar al eje de rotación del disco. Si se colocan unos sensores en los soportes que midan el ángulo que forma el soporte con respecto a la dirección inicial, entonces se podrá calcular la rotación que ha efectuado el sistema en el que se ha instalado el giróscopo.

Otro tipo de giróscopos, cada vez más populares son los giróscopos ópticos, que aparecieron en torno a 1960, y están basados en el efecto *Sagnac*, descubierto por el científico francés *Georges Sagnac* en 1913. Este efecto se manifiesta cuando se hace que dos rayos de luz sigan una misma trayectoria en direcciones opuestas, al mismo tiempo que la plataforma en la que viajan los haces de luz se encuentra rotando. En la parte derecha de la fig. 2.3 aparece representado un giróscopo óptico muy simplificado, que utiliza el efecto *Sagnac* para calcular rotaciones. El dispositivo está formado por un láser que emite, al mismo tiempo, un pulso de luz en sentido horario, y otro en sentido antihorario. Los pulsos de luz se reflejan mediante espejos, de manera que se dirigen hacia un sensor que mide el tiempo que ha tardado en llegarle la luz. Si la plataforma no rota sobre sí misma, los dos haces de luz llegarán al mismo tiempo. Sin embargo, si la plataforma se encuentra rotando en sentido horario, por ejemplo, el haz que viaja en ese mismo sentido tardará más

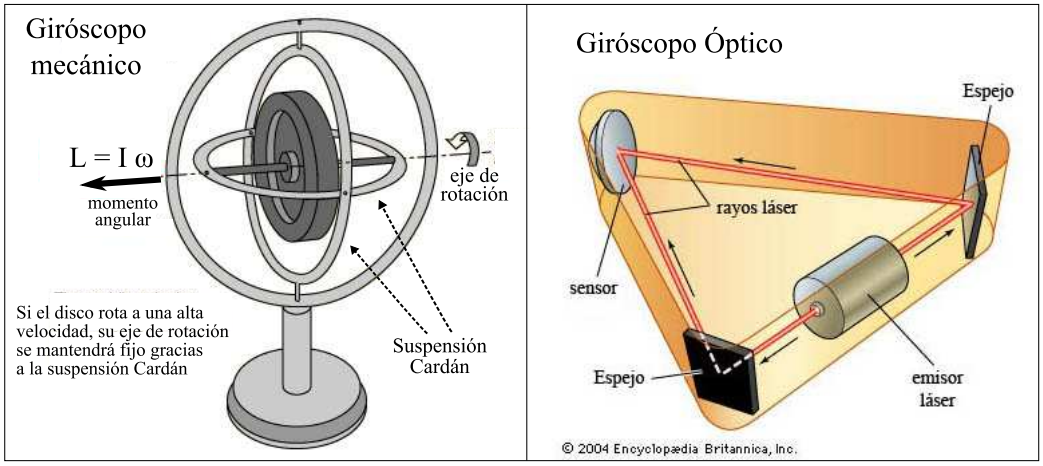


Figura 2.3: *Esquema de un giróscopo mecánico y otro óptico.*

tiempo en llegar a su destino, ya que el sensor se habrá desplazado en sentido horario ligeramente y la luz habrá recorrido más distancia. Del mismo modo, el haz que viaja en sentido antihorario tarda menos tiempo, puesto que la luz recorre una distancia menor. Esta pequeña diferencia de tiempos permite medir la rotación de la plataforma con precisión, y con respecto a los giróscopos mecánicos necesita una tecnología mucho más compleja para su fabricación.

2.2. Sensores Exteroceptivos

Este apartado recoge la descripción de algunos de sensores exteroceptivos más populares en robótica: sensores de visión, el sónar y el láser. En contraposición a los propioceptivos, los sensores exteroceptivos solamente detectan lo que ocurre en el exterior del robot. Las medidas de este tipo de sensores normalmente son interpretadas por el robot para extraer características del entorno y construir un modelo del mismo.

2.2.1. Sensores de visión

Los sensores de visión son muy importantes, debido a que proporcionan una enorme cantidad de información acerca del entorno, probablemente más que ningún otro tipo de sensor ([6]). Mediante la visión, se pueden realizar una gran cantidad de tareas diferentes, como por ejemplo, reconocer patrones, detectar movimiento, estimar la posición, extraer características, etc. En las máquinas, los sensores de visión tratan de emular a la vista humana por

medio de sensores que sean capaces de detectar la luz del mismo modo que nosotros la detectamos. Actualmente existen dos tecnologías diferentes para construir sensores de visión: CCD y CMOS, siendo ambas más limitadas que la visión humana.

Los CCD (*Charge-Coupled Devices*) están formados por una matriz de varios millones de diminutas células fotoeléctricas sensibles a la luz denominadas *píxels* (*picture elements*). Cada píxel se puede entender como un condensador sensible a la luz, con un tamaño de entre 5 y 25 μm. Para medir la luz, en primer lugar los condensadores se cargan y después comienza el periodo de integración. Cuando los fotones de la luz llegan a cada píxel, liberan electrones, que son capturados y retenidos por campos eléctricos. A medida que pasa el tiempo, cada píxel acumula un nivel de carga, basado en el número total de fotones que ha almacenado. Una vez terminada la acumulación de fotones, se procede a leer la carga de cada píxel. Este proceso se efectúa en una esquina del chip CCD. Los píxels de la fila inferior se transportan hasta la esquina y se leen, y a continuación las células de la fila superior se transportan hasta la fila inferior y se repite el proceso. Este proceso es muy complejo, y es crucial que las cargas sean transportadas correctamente a través del chip.

Los sensores de visión CMOS (*Complementary Metal Oxide Semiconductor*), al igual que los CCD, también están compuestos por una matriz de píxels, con la diferencia de que al lado de cada píxel se encuentran varios transistores específicos para ese píxel. Los píxels acumulan carga durante el periodo de integración, como en los CCD, pero a la hora de leer los valores de cada píxel, no hace falta leerlos por filas, sino que se pueden leer todos de una sola vez, gracias a los transistores colocados en cada píxel. Una de las ventajas de los CMOS respecto a los CCD, es que tiene un menor consumo de energía, y su coste es menor.

Ambos sensores proporcionan varios millones de datos, y tanta información es lenta de procesar. Además, extraer características visuales para la navegación no es una tarea fácil, más aún si en lugar de una cámara de visión se utilizan dos (visión estéreo), para emular la visión humana [6]. A pesar de que la visión por computador lleva estudiándose desde hace décadas, existen pocos algoritmos robustos. Los métodos existentes suelen funcionar muy bien bajo determinadas circunstancias, pero si por ejemplo se modifica la iluminación o la textura del fondo, su rendimiento disminuye.

2.2.2. Sónar

El vocablo *sónar* es el acrónimo de "SOund Navigation And Ranging", o "navegación y localización por sonido", en español. Los sensores de tipo sónar

son los más empleados habitualmente en robots móviles de interiores [6]. La razón de su popularidad es que son muy fáciles de obtener, tienen un coste muy bajo, son muy sencillos de controlar, y además existe una gran cantidad de artículos de investigación en los que se hace uso de este sensor, con lo cual disponer de algoritmos de control está al alcance de la mano. Debido a su bajo precio, es común combinar de estos dispositivos para equipar una plataforma robótica con cinturón de sónares, cubriendo 360º.

El principio de funcionamiento del sónar es utilizar energía acústica para efectuar medidas de distancia. Un dispositivo emisor envía un paquete de ondas de presión ultrasónicas, y posteriormente se registra el tiempo que tardan las ondas en reflejarse y volver a un receptor. El tiempo de ida y vuelta se denomina tiempo de vuelo (*time of flight*). La distancia (d) al objeto que causó la reflexión se puede calcular basándose en la velocidad de propagación del sonido (c), y el tiempo de vuelo (t):

$$d = \frac{c \cdot t}{2} \tag{2.1}$$

La velocidad del sonido en el aire (c) viene dada por:

$$c = \sqrt{\gamma RT}$$

donde γ es el coeficiente adiabático, R es la constante universal de los gases, y T es la temperatura absoluta en grados Kelvin. En el aire, en condiciones normales de presión y temperatura, la velocidad del sonido es aproximadamente $c = 343 m/s$.

La figura 2.4 muestra el funcionamiento de envío y recepción de señales de un sensor ultrasónico. Primero, se emiten una serie de pulsos de sonido, conocidos como *paquete de ondas*. Cuando la transmisión de ondas termina, un integrador comienza a incrementar su valor, con el objeto de medir el tiempo desde la transmisión de las ondas hasta la detección de un eco. En el receptor se establece un valor umbral, que en caso de ser superado al recibir una onda de sonido, se considera que la onda es un eco de entrada válido. Normalmente en el sónar, el emisor y el receptor son el mismo dispositivo, pero no al mismo tiempo. La transformación de uno en otro no puede ser instantánea, porque la membrana electrostática que se usa para emitir los pulsos ultrasónicos no puede usarse como receptor hasta que no haya dejado de vibrar por completo. El tiempo que tarda en detenerse la membrana se denomina *blanking time* (tiempo de borrado). Debido a esto, durante la transmisión del pulso de sonido y durante el tiempo de borrado, el umbral se fija con un valor más alto de lo normal para eliminar la posibilidad de que el receptor detecte las vibraciones de la onda emitida. Una vez pasado

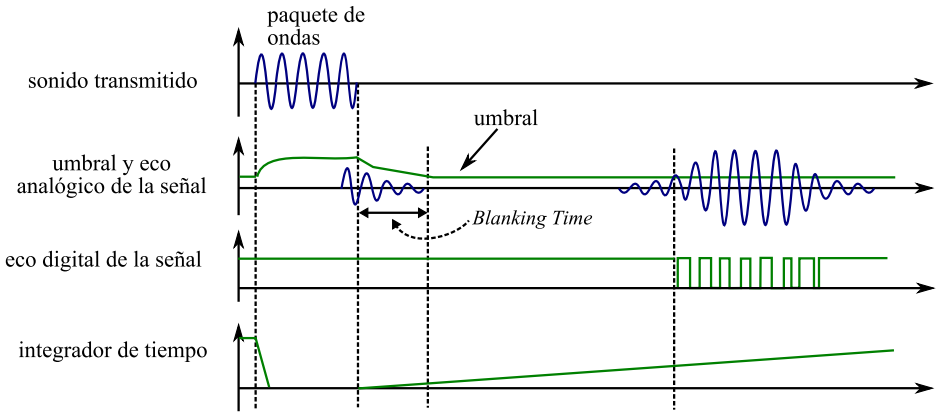


Figura 2.4: *Emisión y recepción de señales en un sensor ultrasónico*

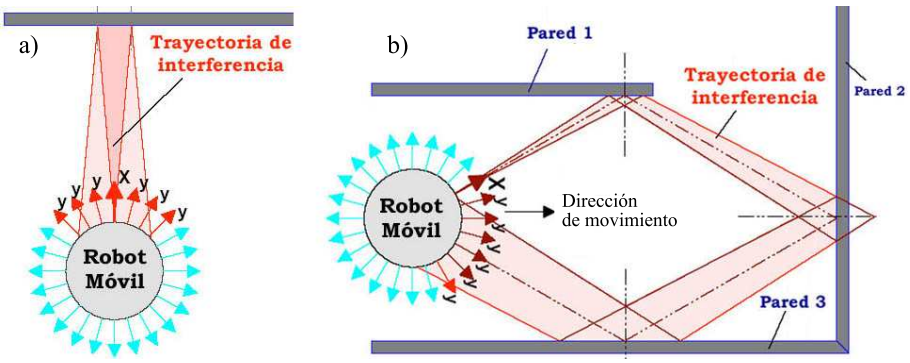


Figura 2.5: *Interferencias (crosstalk) en el sónar: a) directa. b)indirecta.*

el tiempo de borrado, si se recibe una onda que supere el umbral, el sónar producirá una señal digital y calculará la distancia utilizando el valor del integrador.

Uno de los inconvenientes del sónar es que la propagación de la energía acústica se ve afectada por varios factores, como puede ser la reflectividad acústica de los materiales del entorno. Algunos materiales pueden no reflejar la onda con la intensidad necesaria para ser detectada por el sónar. Por ejemplo, la espuma y las prendas de piel, pueden absorber las ondas de sonido. Otro problema, que se produce cuando se tiene un robot móvil con sónares cubriendo los 360º, son las interferencias (*crosstalk*), representadas en la figura 2.5. Las interferencias directas se dan cuando el sónar x emite un pulso de sonido y al reflejarse directamente en una pared u otro objeto, lo recibe el sónar x, junto con los sónares y adyacentes. Las interferencias indirectas se dan cuando el sónar x envía un pulso de sonido y mediante una serie de reflexiones, lo reciben los sónares y, pero no el x. Para eliminar las

interferencias hay que evitar utilizar todos los sónares simultáneamente.

2.2.3. Escáner láser de medición de distancias

La palabra láser responde a las siglas *Light Amplification by Stimulated Emission of Radiation* (amplificación de luz mediante la emisión inducida de radiación. Al igual que el sónar, utiliza el principio de tiempo de vuelo para medir distancias (ver eq. 2.1). Entre sónar y láser existe una diferencia fundamental: la velocidad de propagación. Para el sonido es 0,3 m/ms, mientras que para las señales electromagnéticas es 0,3 m/ns, es decir, 1 millón de veces más rápido. Por ejemplo, en una distancia de 3 m, un sónar tardaría 10 ms, mientras que un láser mediría la distancia en 10 ns. Es evidente que para medir el tiempo de vuelo de señales electromagnéticas se necesita una tecnología más avanzada que para medir el tiempo de vuelo de un sónar. Esto explica que el precio de un láser sea mucho más elevado que el de un sónar. En general, se pueden citar como ventajas de los sensores láser las siguientes:

- En la mayoría de los casos, las medidas pueden ser consideradas como instantáneas. Esto significa que no hace falta tener en cuenta si el robot se desplaza mientras se efectúa la medición, debido a que la velocidad de movimiento del robot es despreciable en comparación con la velocidad de la luz. El sónar, por el contrario, puede requerir compensaciones de movimiento del robot para que la medida de distancia sea más exacta.

- La precisión de un láser es mucho mayor que la de un sónar. Por ejemplo, existen modelos que tienen una desviación estándar de error en la medida de tan sólo 1 mm, la distancia máxima que pueden medir es de 80 m, y la resolución angular es de 0,25º.

- Los datos de un escáner láser pueden interpretarse directamente, puesto que representan distancias a objetos. Esto también se cumple para un sónar, pero no para los sensores de visión, cuyas imágenes son mucho más difíciles de analizar.

- Con respecto a los sensores de visión, el tiempo de procesamiento de los datos de un láser es mucho menor. Una imagen de una cámara de visión está compuesta por millones de píxels, mientras que en un barrido completo, un láser obtiene del orden de 300 medidas de distancia.

Como principales desventajas se pueden contar su elevado precio y el hecho de que algunos materiales como el cristal son invisibles al sensor, debido a que los pulsos de luz lo atraviesan. Otro inconveniente es que normalmente

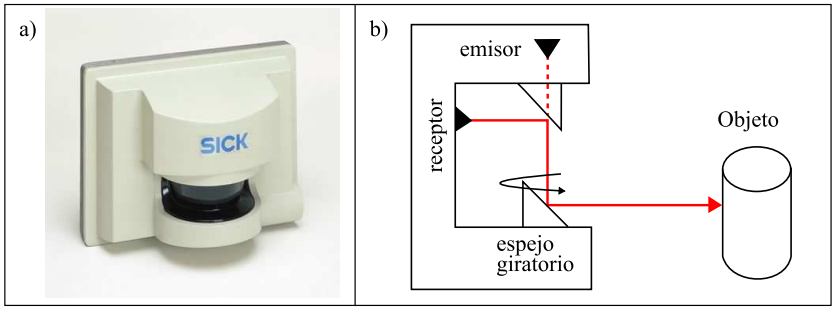


Figura 2.6: *a) Escáner láser lms 221 de sick. b)Sistema de funcionamiento interno*

los datos de un láser se limitan a un plano, si bien esto se puede solventar colocando el sensor en un dispositivo *pan & tilt*, de modo que se puedan obtener datos en 3D balanceando y girando el láser en diferentes planos.

Debido a sus ventajas con respecto al resto de sensores, el uso de escáneres láser de medición de distancia está muy difundido últimamente y es un campo muy activo de investigación hoy en día. Estos motivos han influido en que sea el sensor principal utilizado en este trabajo.

Escáner Láser SICK LMS 221

El modelo utilizado para el desarrollo de este trabajo ha sido un LMS 221 de la casa SICK (fig. 2.6a). Las siglas LMS significan *Laser Measurement System* (sistema de medición basado en láser). Este modelo escanea su entorno en un plano 2D y está especialmente diseñado para poder trabajar tanto en ambientes interiores, como en ambientes exteriores con condiciones extremas (lluvia, nieve, niebla, frío).

El LMS 221 basa su funcionamiento en la medida del tiempo de vuelo. Como se muestra en la fig. 2.6b, para determinar la distancia a la que se encuentra un objeto, el láser emite un pulso de luz infrarroja. Cuando el pulso incide sobre el objeto más cercano, regresa hacia el sensor y se determina el tiempo transcurrido. Conocido el tiempo de ida y vuelta del pulso (tiempo de vuelo), se calcula fácilmente la distancia al objeto detectado. De este modo se puede medir la distancia en una sola dirección, pero gracias a que dispone internamente de un espejo rotatorio, se logra un efecto de barrido en dos dimensiones. La amplitud del barrido en este modelo es siempre de 180º, como se ilustra en la fig. 2.7a. En esta figura también se puede observar que si los pulsos emitidos no indicen sobre ningún objeto cercano, el láser devuelve la distancia máxima, dando lugar a un conjunto de medidas que forman un semicírculo.

Al conjunto de medidas que obtiene el láser se le denomina *barrido láser*

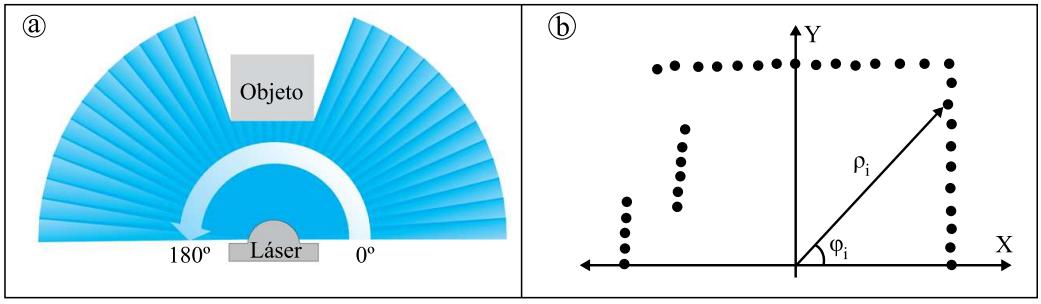


Figura 2.7: *a) Amplitud de barrido del* SICK LMS *221. b) Ejemplo de barrido láser*

(en inglés se emplea *scan* o *laserscan*). El láser proporciona los datos en formato polar, $R = \{(\rho_i, \varphi_i) | i = 1, \ldots, n\}$ (Fig. 1b), donde n puede valer 181 ó 361, y (ρ_i, φ_i) son las coordenadas del i-ésimo punto detectado por el sensor. El término ρ_i indica la distancia al obstáculo detectado al emitir el pulso de luz con orientación φ_i. Los puntos son adquiridos secuencialmente por el escáner, en sentido antihorario, y con una resolución angular $\varphi_{i+1} - \varphi_i = \Delta\phi$ que puede valer 0,5º ó 1º.

El modelo LMS 221 permite configurar los siguientes parámetros:

- Resolución angular del barrido: 1º, 0,5º ó 0,25º. Dependiendo de la selección, la amplitud de barrido y el número de mediciones realizadas es variable.

Resolución angular	**0,25º**	**0,5º**	**1º**
Amplitud de barrido	100º	180º	180º
Número de mediciones	401	361	181

- Tasa de transmisión de datos: 9,6, 19,2, 38,4 ó 500 KBaudios. Los datos obtenidos se transmiten a un PC a través del puerto serie utilizando el estándar RS-232 o el RS-422.

- Máxima distancia de detección de objetos: hasta 8 m, o hasta 80 m. Dependiendo de una u otra selección, la desviación estándar del error en la medida es de 5 mm ó 10 mm, respectivamente.

En la tabla 2.1 se resumen las características técnicas del *sick lms* 221.

La configuración más habitual del láser con respecto a la resolución angular es 0,5º, debido a que ofrece mayor número de datos que con una resolución de 1º. Además el barrido para 0,5º ó 1º, es más rápido que para $0,25º$, dado que para lograr esta resolución se necesita unir dos barridos: uno de 0º a 180º y otro de 0,5º a 179,5º, ambos con una resolución angular de 0,5º. Con

Amplitud de barrido	180º
Resolución angular	de 0,25º a 1º
Tiempo de respuesta	de 13 a 52 ms
Resolución	10 mm
Alcance del láser	80 m
Interfaz de datos	RS-422 o RS-232
Velocidad de transmisión de datos	9.6, 19.2, 38.4 ó 500 Kbaudios
Longitud de onda	Infrarrojo ($\lambda = 905$ nm)
Suministro de voltaje	24 V DC
Consumo de potencia	20 W
Peso	9 kg
Dimensiones	196 x 352 x 266 mm

Cuadro 2.1: *Datos técnicos del láser*

esta resolución, el tiempo de respuesta es de 26,6 ms y por tanto la tasa de muestreo máxima es de 37 Hz, aproximadamente.

Extracción de características mediante láser

En la parte superior de la fig. 2.8 se muestra una fotografía de una sala, y sobre ella se ha dibujado una línea punteada en rojo, correspondiente a una aproximación del plano en el que el láser ha efectuado las medidas. En la parte inferior de la misma figura se muestra una gráfica del barrido láser realizado por el láser en dicha sala.

Observando con detenimiento la figura, hay dos tipos de características que es especialmente importante poder detectar en el barrido láser[3]: los puntos de ruptura y los segmentos. Los primeros indican discontinuidades en el proceso de escaneado, y se producen normalmente debido a la existencia de objetos o superficies que obstaculizan la detección de otros elementos más lejanos. La detección de puntos de ruptura permite clasificar los puntos en grupos denominados *clusters*. Por otra parte, los segmentos están formados por conjuntos de puntos consecutivos que forman entre sí una línea recta. Estos segmentos son el resultado de escanear superficies planas en el entorno, como armarios, mesas o paredes. En la fig. 2.8 se indica un ejemplo de punto de ruptura, y además se pueden observar varios segmentos rectilíneos.

Dentro de la variedad de características detectables por un láser, este trabajo se centra en algoritmos de *clustering* y de extracción de líneas. En los siguientes capítulos (3 y 4) se revisarán métodos existentes en la bibliografía para extraer ambos tipos de características, mientras que en el capítulo 5 se propondrán dos nuevos métodos: uno de *clustering* y otro de extracción de líneas.

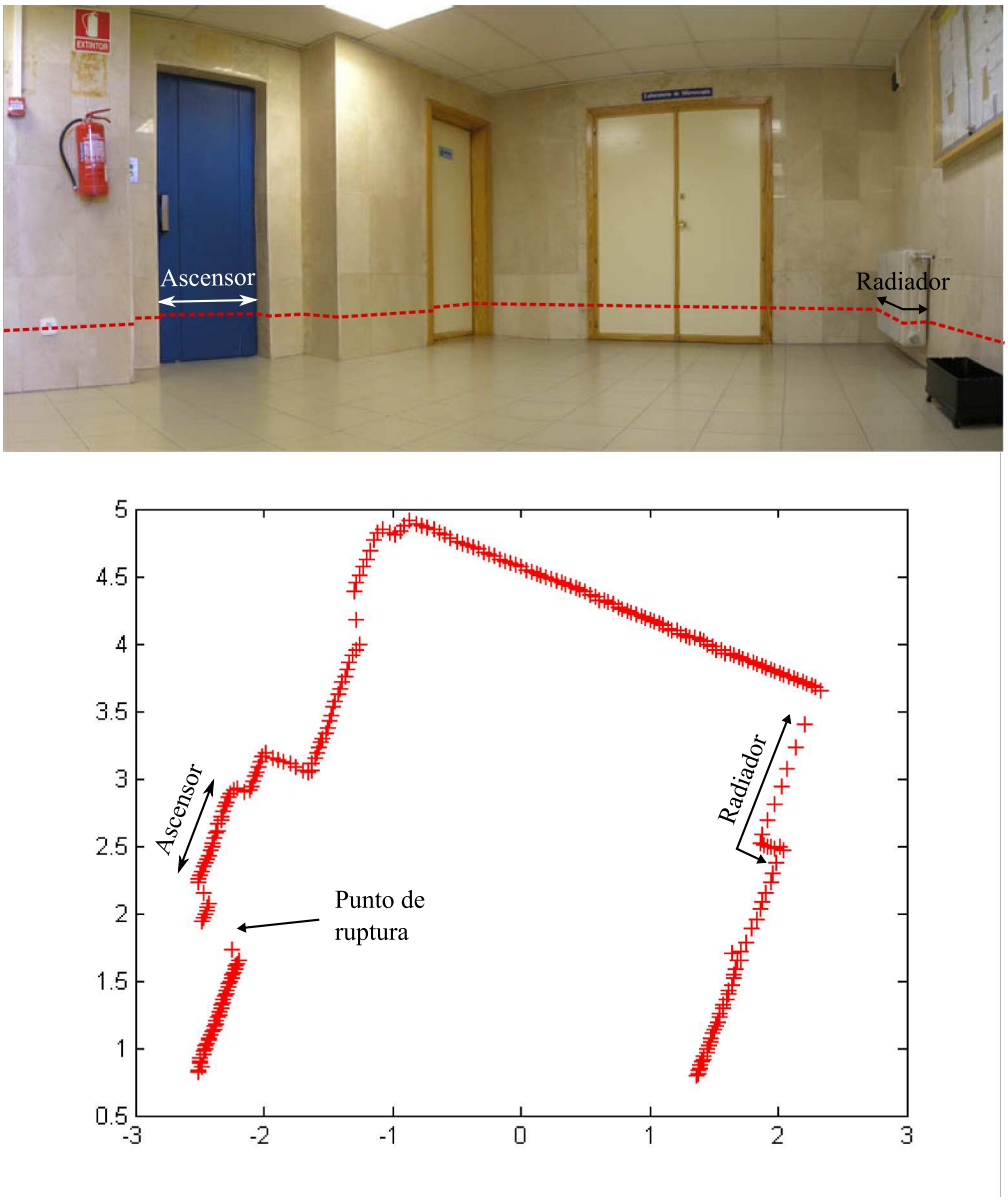


Figura 2.8: *Fotografía de un escenario real junto con su correspondiente barrido láser*

3

Técnicas de *clustering*

De manera general, se entiende por *clustering* la división de un conjunto de datos en subconjuntos (*clusters*), tales que los datos de un mismo subconjunto compartan alguna característica en común. Es una técnica usada en diversos campos, como minería de datos, reconocimiento de patrones, o bioinformática. Este capítulo comienza tratando los aspectos específicos del *clustering* aplicado a los datos obtenidos mediante un escáner láser. A continuación se presenta un algoritmo general de *clustering*, y por último se estudian en detalle los métodos más interesantes encontrados en la bibliografía, clasificándolos en "adaptativos" y "no adaptativos".

3.1. *Clustering* de barridos láser

Cuando se hace referencia al proceso de sub-agrupar los puntos obtenidos por un escáner láser, es común encontrar diferentes términos en la bibliografía. En [8], por ejemplo, se denomina *breakpoint detection* (detección de puntos de ruptura). Quizás sea una expresión acertada, pero lamentablemente no es un término ampliamente usado. En algunas referencias ([11], [30], [33]) aparece el término *segmentation* (segmentación). Este término es algo ambiguo y puede inducir a confusión, ya que en ocasiones se utiliza para referirse al proceso de detección de segmentos rectilíneos y no simplemente a *clusters* de puntos. Finalmente, otros autores ([21], [27], [39]) prefieren em-

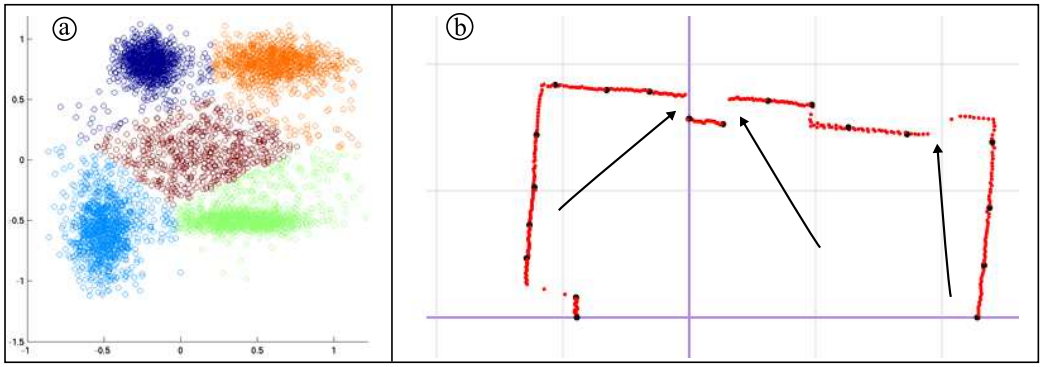


Figura 3.1: *a)Ejemplo de clustering K-means. b)Ejemplo de clustering en un barrido láser.*

plear *clustering* (agrupamiento). Este término, sin su respectiva traducción al español, es muy común en el ámbito científico hispano. A lo largo de este trabajo, ***clustering*** será el término empleado.

Existen algoritmos de *clustering* muy populares en otros campos, pero que no se adaptan bien a los barridos láser. Normalmente, estos métodos están preparados para funcionar con grandes conjuntos de datos, y son bastante lentos, ya que lo que importa es clasificar bien los datos, no el tiempo de ejecución. Un algoritmo muy conocido es el *K-means* (ver fig. 3.1a), que organiza los *clusters* basándose en el cálculo de centroides, siguiendo un proceso iterativo. El uso de centroides no es de mucha utilidad en un barrido láser, porque los datos están ordenados en secuencia y además suelen corresponder a estructuras rectilíneas o curvilíneas. Debido a estas diferencias, existen diversas técnicas de *clustering* especialmente diseñadas para dividir en *clusters* el conjunto de puntos que conforman un barrido láser. Estas técnicas suelen clasificar los *clusters* de una sola vez, sin necesitar varias iteraciones.

La etapa de *clustering*, aplicada a un barrido láser, normalmente lo que busca es identificar los límites de los objetos detectados por el escáner láser. La idea principal consiste en subdividir el conjunto de medidas de distancia en subconjuntos más pequeños, con el fin de detectar discontinuidades de distancia en los datos obtenidos. En la fig. 3.1b se muestra un ejemplo de barrido láser en el que se indican las discontinuidades mediante flechas. Cada flecha indica que un *cluster* de puntos termina y comienza el siguiente. En un barrido láser, se puede considerar un *cluster* como un conjunto de medidas de distancia (puntos) que se encuentran lo suficientemente cerca unos de otros como para poder decir que forman parte del mismo objeto.

La detección de *clusters* es un procedimiento que se suele aplicar en una etapa previa a la extracción de líneas. Esto permite incrementar la eficacia

del algoritmo de extracción de líneas, al entregarle un conjunto de datos compuesto exclusivamente por elementos contiguos. Sin esta fase previa, al intentar extraer líneas se podrían conectar de manera errónea los extremos de dos *clusters* contiguos, incluso aunque existiera una gran distancia de separación entre ellos. Debido a esto, el extractor de líneas probablemente no funcionaría de modo adecuado, y su rendimiento se vería disminuido. De todas maneras, para decidir si se debe aplicar o no una etapa de *clustering* habrá que estudiar el detector de líneas en concreto que se vaya a utilizar, ya que puede darse el caso de que no se logre ninguna mejora.

3.2. Algoritmo general de *clustering*

En las siguientes secciones se analizarán con detalle diversos métodos de *clustering* que basan su funcionamiento en la determinación de la distancia mínima que debe existir entre dos puntos consecutivos para considerar que deben pertenecer a *clusters* diferentes. Los métodos que funcionan bajo esta perspectiva tienen la forma general del algoritmo 1, en el que MAXPUNTOS es la constante correspondiente al número de puntos que constituyen un barrido láser, r_i indica la distancia desde el láser hasta el punto i, y $D(r_i, r_{i+1})$ representa la distancia euclídea entre dos puntos sucesivos.

Algorithm 1 Algoritmo de *clustering* básico

1: **para** $i \leftarrow 1$ **hasta** MAXPUNTOS **hacer**
2: **si** $D(r_i, r_{i+1}) > D_{umbral}$ **entonces**
3: r_{i+1} pertenece a un nuevo *cluster*
4: **si no**
5: r_{i+1} pertenece al *cluster* actual
6: **fin si**
7: **fin para**

La principal dificultad de estos enfoques radica en el cálculo de la distancia umbral D_{umbral}. Este valor suele ser una función de la distancia a la que se encuentran los puntos medidos por el sensor láser, puesto que la resolución angular del sensor es constante. Si el valor D_{umbral} es constante durante todo el algoritmo, entonces el *clustering* será *no adaptativo*. En cambio, si dicho parámetro es variable y toma un valor diferente en cada iteración, el *clustering* será de tipo *adaptativo*.

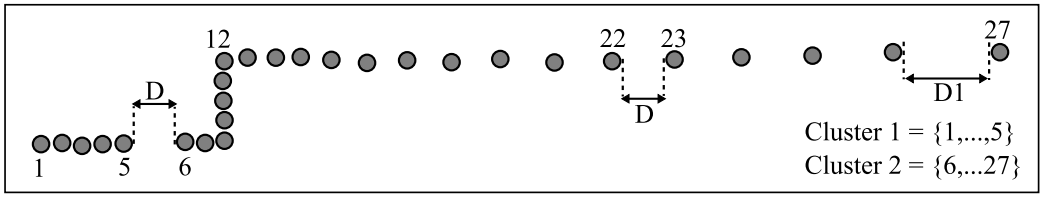


Figura 3.2: *Problemas con clustering no adaptativo*

3.3. *Clustering* no adaptativo

Como indica su nombre, este tipo de *clustering* no depende de la distancia a la que se encuentren los objetos del sensor láser. Siguiendo los pasos del pseudocódigo del algoritmo 1, esta técnica consiste en seleccionar un valor fijo para el parámetro D_{umbral}. Esto significa que es muy importante el valor umbral que se seleccione, ya que de ello va a depender todo el proceso de *clustering*.

Supóngase que se va a aplicar *clustering* no adaptativo a un barrido láser como el de la figura 3.2, formado por 27 puntos. En la figura hay dos *clusters*: uno formado por los puntos del 1 al 5, y otro por los puntos del 6 al 27. A partir del punto 12, las distancias entre cada par de puntos consecutivos aumentan progresivamente, y pertenecen al mismo objeto, por eso no se produce cambio de *cluster*. Esto sucede frecuentemente en entornos del mundo real, cuando se perciben paredes de gran longitud (como pasillos, p. ej.). En este conjunto de puntos, se puede observar que la distancia (D) entre los puntos 5 y 6 es exactamente la misma que entre los puntos 22 y 23. Si se selecciona un valor $D_{umbral} < D$, entonces se detectará correctamente que el espacio entre los puntos 5 y 6 corresponde a un cambio de *cluster*, pero como contrapartida, a partir del punto 22, cada punto se considerará como un *cluster* independiente, cuando en realidad no lo es. Por otra parte, si se escoge un valor $D_{umbral} > D1$, para evitar caer en el error de detectar *clusters* de un solo punto, entonces solamente se detectará un solo *cluster*, con inicio en el punto 1, y fin en el 27. Entre escoger un valor más bien grande, o más bien pequeño para D_{umbral}, sería preferible lo primero. De este modo, puede que no se detecten algunos *clusters* difíciles, pero sí sería posible agrupar puntos bastante separados del resto y filtrarlos si el número de puntos del conjunto es pequeño.

Debido a estos problemas, normalmente se hace uso de técnicas de *clustering* que dependan de lo alejados que estén los puntos detectados. Pese a todo, el *clustering* no adaptativo es un proceso simple y que puede resultar útil en determinados casos. Ejemplos de su aplicación se pueden encontrar

en [25] para la construcción dinámica de mapas de interiores; [27] para comparar el rendimiento de varios algoritmos de extracción de líneas y [21] para la localización de vehículos y construcción de mapas de un aparcamiento.

3.4. *Clustering* adaptativo

Para evitar los problemas mencionados en el apartado anterior, los métodos de *clustering* adaptativos calculan diferentes valores para D_{umbral} en cada iteración del bucle del algoritmo 1. Uno de los métodos adaptativos más básicos que se ha encontrado es el propuesto por Zezhong *et al.* [39]. En dicho trabajo deciden utilizar un valor umbral variable mediante una función lineal, argumentando que la densidad de muestreo del láser es distinta a distancias diferentes. La función lineal consiste en que si se decide que D_{umbral} valga Δ para un punto situado a una distancia D, entonces D_{umbral} pasará a tomar el valor $n\Delta$ para otro punto que esté situado a una distancia de nD.

Al margen de este sencillo ejemplo, a continuación se estudiarán tres diferentes criterios para calcular el valor de D_{umbral}, y que están basados en cálculos adaptativos más elaborados. Puesto que los tres métodos operan siguiendo el pseudocódigo del algoritmo 1, y la única diferencia está en la manera de calcular D_{umbral}, en lo siguiente aparecerán designados como "Criterio de" seguido por el nombre del autor principal del trabajo.

3.4.1. Criterio de Dietmayer

En [11], Dietmayer *et al.* proponen un criterio adaptativo para calcular la distancia D_{umbral}, cuyo funcionamiento se puede explicar mediante la ilustración de la figura 3.3. P_a y P_b representan dos puntos consecutivos detectados por el láser, mientras que r_a y r_b son las respectivas distancias de dichos puntos al origen de coordenadas. Dado el triángulo OP_aP_b donde r_a y r_b son conocidos y α es la resolución angular del láser, entonces aplicando el teorema del coseno, se calcula la distancia entre P_a y P_b:

$$r_{ab} = \sqrt{r_a^2 + r_b^2 - 2r_a r_b cos(\alpha)}$$

Como el escáner utilizado por Dietmayer tiene una resolución angular $\alpha = 0{,}25^o$, un valor muy pequeño, propone simplificar el cálculo de r_{ab} suponiendo:

$$r_{ab} \approx |r_a - r_b|$$

El criterio que se utiliza para formar los *clusters* consiste en que si la

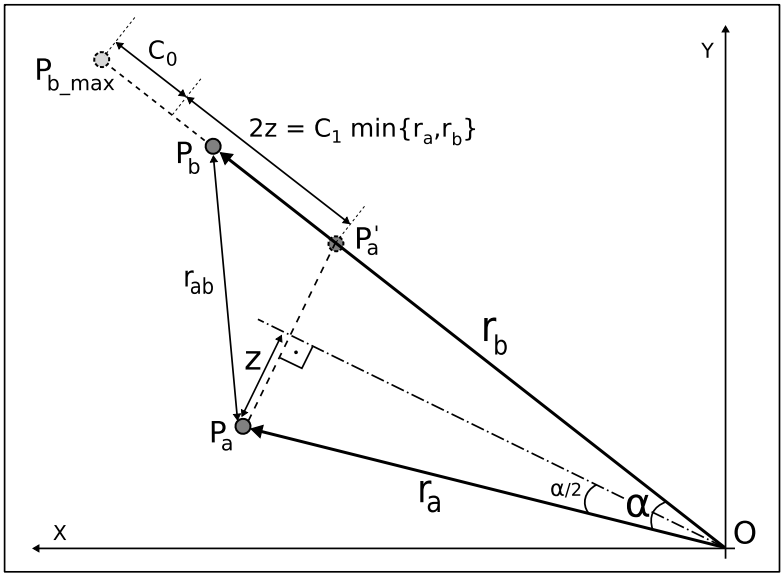


Figura 3.3: *Ilustración del criterio de clustering de Dietmayer*

distancia entre P_a y P_b es menor que

$$r_{ab} \leq C_0 + C_1 \cdot min\{r_a, r_b\},$$

donde $C_1 = \sqrt{2(1 - cos(\alpha))}$, entonces P_b pertenece al mismo *cluster* que P_a. En caso contrario, los puntos P_a y P_b formarán parte de *clusters* diferentes.

La constante C_0 se justifica como un ajuste frente al ruido en las medidas del láser. La otra constante, C_1, toma un valor cuyo cálculo y significado gráfico no es explicado por Dietmayer, pero que se puede justificar del siguiente modo. En la figura 3.3, se puede apreciar que $min\{r_a, r_b\} = r_a$, y por tanto:

$$C_1 \cdot min\{r_a, r_b\} = r_a \cdot \sqrt{2(1 - cos(\alpha))} = 2 \cdot r_a \cdot \sqrt{\frac{1 - cos(\alpha)}{2}}$$

Por otro lado, la variable marcada como z en la figura tomaría el siguiente valor:

$$z = r_a \cdot sin(\frac{\alpha}{2}) = r_a \cdot \sqrt{\frac{1 - cos(\alpha)}{2}}$$

De este modo, finalmente resulta que:

$$2z = C_1 \cdot r_a$$

lo que quiere decir que $C_1 \cdot r_a$ es la distancia entre los puntos P_a y $P_a^{'}$, siendo $P_a^{'}$ el punto situado sobre el segmento $\overline{OB}$ que se encuentra a la misma distancia que P_a del origen de coordenadas. Por otra parte, sabiendo que en la figura se cumple que $r_b > r_a$, entonces se puede obtener:

$$\begin{aligned} r_{ab} &= r_b - r_a \leq C_0 + C_1 \cdot r_a \\ r_b &\leq r_a + C_0 + C_1 \cdot r_a \end{aligned}$$

Esto significa que si al punto $P_a^{'}$ le añadimos las distancias $C_1 \cdot r_a$ y C_0 obtendremos el punto P_{b_max} que se corresponde con la distancia máxima a la que se puede situar el punto P_b para formar parte del mismo *cluster* que el punto P_a. Como se puede observar, es un criterio bastante más elaborado que los presentados anteriormente.

3.4.2. Criterio de Santos

Basándose en el criterio de distancia de Dietmayer, Santos *et al.* [33] critican que el valor calculado depende en gran medida de la distancia entre el láser y el objeto medido. Para contrarrestar esta dependencia incluyen un nuevo parámetro, β, de tal modo que el criterio queda como sigue:

$$|r_a - r_b| \leq C_0 + min\{r_a, r_b\} \cdot \frac{\sqrt{2(1 - cos\alpha)}}{cotg(\beta) \cdot cos(\alpha/2) - sin(\alpha/2)}$$

donde C_0 es una constante usada para compensar el ruido en las medidas del láser. El papel de β se puede apreciar en la figura 3.4. Sean P_a y P_b dos puntos consecutivos medidos por el láser, r_a y r_b sus respectivas distancias al láser y m su bisectriz. Si ahora se traza la perpendicular a la bisectriz por el punto P_a, entonces β es la inclinación máxima absoluta que puede tener la superficie de un objeto para pertenecer al mismo *cluster* que el punto P_a. Por lo tanto el parámetro β determina la distancia s. Veamos cómo demostrar la relación entre ambos:

$$\begin{aligned} a &= s \cdot cos(\phi) = s \cdot cos(\frac{\pi}{2} - \frac{\alpha}{2}) = s \cdot sin(\frac{\alpha}{2}) \\ b &= s \cdot sin(\phi) = s \cdot sin(\frac{\pi}{2} - \frac{\alpha}{2}) = s \cdot cos(\frac{\alpha}{2}) \end{aligned}$$

La distancia $P_a - P_a^{'}$ ha sido calculada en el análisis de la sección anterior, y toma el valor $C_1 \cdot min\{r_a, r_b\}$. Por tanto:

$$cotg(\beta) = \frac{C_1 \cdot min\{r_a, r_b\} + a}{b} = \frac{C_1 \cdot min\{r_a, r_b\} + s \cdot sin(\frac{\alpha}{2})}{s \cdot cos(\frac{\alpha}{2})}$$

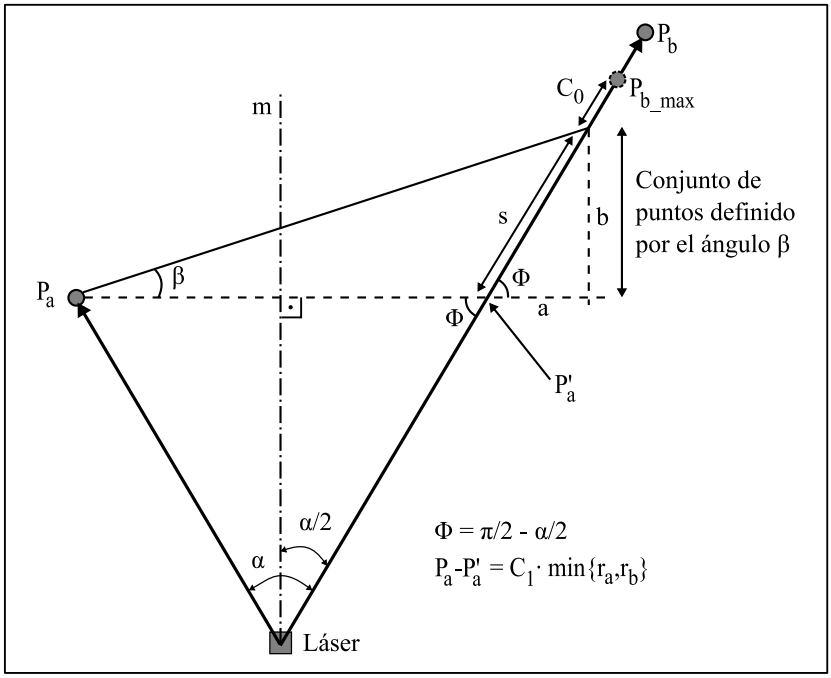


Figura 3.4: *Ilustración del algoritmo de clustering de Santos*

Sacando factor común a s y despejando se obtiene:

$$s = \frac{C_1 \cdot min\{r_a, r_b\}}{cotg(\beta) \cdot cos(\alpha/2) - sin(\alpha/2)}$$

La interpretación gráfica de este algoritmo es bastante similar a la del algoritmo de Dietmayer. El punto P_b formará parte del mismo *cluster* que P_a si no está más lejos que el punto P_{b_max}, el cual se puede calcular añadiendo s y C_0 a P_a'. En el caso de la figura 3.4 P_b está más alejado que P_{b_max}, de modo que no estará en el mismo *cluster* que P_a.

Hay dos aspectos a tener en cuenta con respecto al parámetro β. En primer lugar, si β es pequeño, los límites de algunos objetos no serán detectados correctamente; y si β es muy grande se corre el riesgo de que un punto correspondiente a otro objeto no sea detectado en un *cluster* diferente sino en el mismo. El otro aspecto a meditar es el siguiente: si $min\{r_a, r_b\}$ es pequeño, el conjunto de puntos definido por el ángulo β será pequeño. De igual manera, si $min\{r_a, r_b\}$ es grande, el conjunto de puntos definido por el ángulo β será grande.

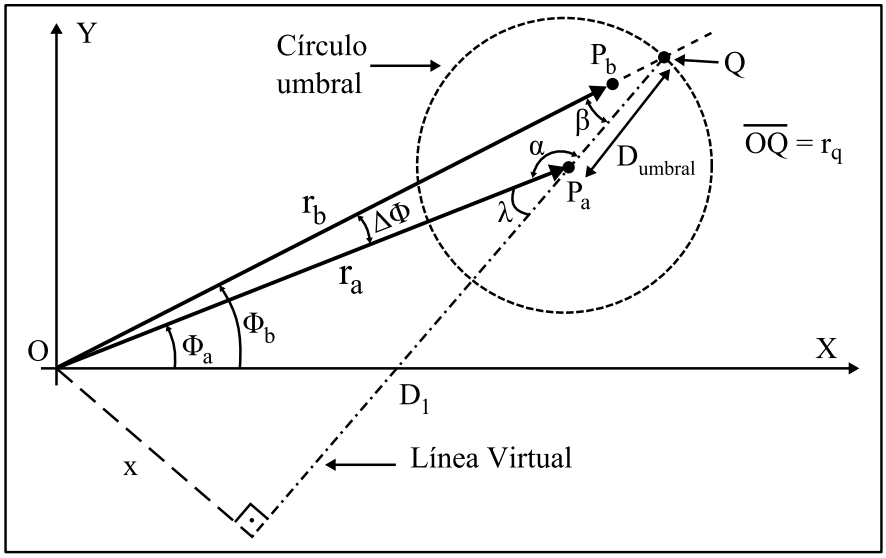


Figura 3.5: *Ilustración del algoritmo de clustering de Borges*

3.4.3. Criterio de Borges

Borges *et al.* [8] proponen otro criterio adaptativo para formar *clusters* con los puntos de un escáner láser, distinto a los dos anteriores. Lo denominan "Detector Adaptativo de Puntos de Ruptura" (*Adaptive Breakpoint Detector*). En la figura 3.5 se ofrece una representación de los conceptos que intervienen en los cálculos. Los puntos P_a y P_b son dos medidas de distancia consecutivas del láser, separados por un ángulo $\Delta\phi$ fijo, mientras que r_a y r_b son las respectivas distancias de los puntos anteriores al origen de coordenadas.

El método consiste básicamente en trazar una *línea virtual* que pase por el punto P_a. Dicha línea forma un ángulo λ con respecto al rayo lanzado por el escáner láser ϕ_a, y representa el peor caso en que una superficie podría ser detectada por el láser. Por ejemplo, si se toma $\lambda = 10^o$, significa que podría detectarse una pared u otro objeto cuya superficie forme un ángulo mínimo de 10^o con respecto a la dirección del rayo emitido por el láser. Además esta línea virtual sirve para averiguar el punto Q, que corresponde a la posición más alejada que podría tomar el punto P_b para formar parte del mismo *cluster* que P_a. El punto Q además determina una distancia D_{umbral} que sirve como radio de un *círculo umbral* centrado en el punto P_a.

El cálculo de D_{umbral} se puede realizar del siguiente modo. En la figura

3.5 se puede observar lo siguiente para x:

$$\begin{aligned} x &= r_a \cdot sin(\lambda) \\ x &= r_q \cdot sin(\beta) \\ r_a \cdot sin(\lambda) &= r_q \cdot sin(\beta) \\ r_q &= \frac{r_a \cdot sin(\lambda)}{sin(\beta)} \end{aligned} \tag{3.1}$$

Igualmente, utilizando D_1 se comprueba que:

$$\begin{aligned} D_1 &= r_a \cdot cos(\lambda) \\ D_1 + D_{umbral} &= r_q \cdot cos(\beta) \\ D_{umbral} &= (D_1 + D_{umbral}) - D_1 = r_q \cdot cos(\beta) - r_a \cdot cos(\lambda) \end{aligned} \tag{3.2}$$

Si se unen las ecuaciones anteriores, sustituyendo la ecuación (3.1) en (3.2), y teniendo en cuenta que $sin(x-y) = sin(x) \cdot cos(y) - cos(x) \cdot sin(y)$, se obtiene:

$$\begin{aligned} D_{umbral} &= \frac{r_a \cdot sin(\lambda)}{sin(\beta)} \cdot cos(\beta) - r_a \cdot cos(\lambda) \\ D_{umbral} &= r_a \cdot \frac{sin(\lambda) \cdot cos(\beta) - cos(\lambda) \cdot sin(\beta)}{sin(\beta)} \\ D_{umbral} &= r_a \cdot \frac{sin(\lambda - \beta)}{sin(\beta)} \end{aligned} \tag{3.3}$$

Finalmente, se puede simplificar (3.3) si se se observa que los ángulos α y λ son suplementarios, por lo que $\alpha = \pi - \lambda$. Además, conocidos α y $\Delta\phi$, se obtiene $\beta = \pi - \Delta\phi - \alpha = \lambda - \Delta\phi$. Sustituyendo, resulta:

$$D_{umbral} = r_a \cdot \frac{sin(\Delta\phi)}{sin(\lambda - \Delta\phi)} \tag{3.4}$$

La distancia D_{umbral} que presenta la ecuación anterior se puede utilizar como distancia umbral para decidir si P_b pertenece al mismo *cluster* que P_a. Sin embargo, es necesario tener en cuenta el ruido que el láser añade al medir el punto P_b, ya que de lo contrario el algoritmo puede fallar, según indica Borges, para objetos cercanos al sensor láser Así, es conveniente modificar (3.4) quedando como:

$$D_{umbral} = r_a \cdot \frac{sin(\Delta\phi)}{sin(\lambda - \Delta\phi)} + 3\sigma_r$$

donde σ_r es el error en la medida del láser, y habitualmente varía entre uno y diez centímetros ($\sigma_r = 0{,}01 \ldots 0{,}10$).

Quedan así expuestos en detalle varios criterios de distancia basados en el algoritmo 1, en los cuales la distancia D_{umbral} es variable en cada iteración del bucle, adaptándose en cada ocasión a la distancia a la que se encuentran los objetos detectados por el láser. Al ser adaptativos, se pueden eliminar en teoría, los problemas indicados en la sección 3.3. Los tres criterios analizados se implementarán y se compararán con el algoritmo de *clustering* propuesto en el capítulo 5 de esta memoria, y los resultados de rendimiento serán presentados en el capítulo séptimo.

4

Extracción de líneas

En el campo de la robótica móvil es importante conocer la ubicación de un robot, tanto si éste navega por entornos conocidos, como si está explorando nuevas zonas. La estimación precisa de la posición es la esencia de los sistemas de localización, la construcción dinámica de mapas o la planificación de caminos. Es bien sabido que no es suficiente utilizar exclusivamente la odometría para calcular la posición, puesto que el error en la estimación puede crecer ilimitadamente [6]. Para cualquiera de estas tareas es necesario obtener datos del entorno, y algunos de los sensores más comúnmente utilizados para ello son el sónar, el láser, los infrarrojos o las cámaras de visión. Como se expuso en la sección 2.2.3, el láser presenta una serie de ventajas sobre el resto de sensores, como por ejemplo, su rápidez en la adquisición de datos y su alta precisión en las medidas, entre otras características.

Una cuestión de suma importancia con respecto a los sensores láser es cómo tratar los puntos que suministran en cada medición. Supóngase, por ejemplo, que o bien se dispone de un mapa *a priori* de un entorno, o bien el robot ha navegado un cierto tiempo por un determinado entorno y ha obtenido una serie de datos. Entonces, el problema de comparar los datos recién recolectados por el láser con los datos almacenados previamente, se puede abordar principalmente de dos maneras [27]:

- Comparando datos punto a punto.
- Comparando características extraídas de los datos.

Uno de los pioneros en el uso del primero de los dos enfoques anteriores fue Cox [19], quien propuso un algoritmo para buscar una equivalencia entre cada punto dado por el láser y alguno de los segmentos de un mapa del entorno obtenido a mano. Otro algoritmo bien conocido dentro de esta categoría es el de Lu y Milios [23], en el que punto a punto, se tratan de alinear los datos de dos *scans*, mejorando iterativamente la alineación entre ambos por medio de un ajuste por mínimos cuadrados.

En lugar de trabajar directamente con los puntos uno a uno, los algoritmos que forman parte del segundo enfoque efectúan análisis de los datos y los transforman en descripciones geométricas del entorno, como pueden ser líneas, esquinas, curvas u otras figuras de interés. A continuación son estas características geométricas las que se comparan entre sí. Una de las ventajas de usar descripciones geométricas en lugar de puntos individuales es que se pueden reducir las necesidades de almacenamiento ([1], [9]). Así, para almacenar la descripción geométrica de un segmento únicamente hacen falta los puntos correspondientes a los extremos y los parámetros de la ecuación de la recta correspondiente.

Por lo tanto, utilizar descripciones geométricas permitirá mantener la escalabilidad aun cuando el entorno tratado sea de dimensiones extensas. En entornos interiores, como universidades, oficinas, o naves industriales, es común poder describir el entorno mediante líneas, dado que los objetos más comunes son paredes, armarios o mesas.

La gran mayoría de los métodos utilizados hoy en día para la detección de rectas mediante láser son adaptaciones de algoritmos consolidados procedentes del campo de visión artificial. De la multitud de algoritmos existentes en la bibliografía para este fin, en las secciones siguientes de este texto se procederá a analizar los más significativos.

4.1. Método de mínimos cuadrados

Una de las técnicas más comunes para ajustar un modelo matemático a un conjunto de datos en dos dimensiones es el método de **mínimos cuadrados** (*least squares*), desarrollado por Carl F. Gauß a principios del siglo XIX. Aunque el modelo matemático utilizado puede ser un polinomio de cualquier orden, en nuestro caso se trata de ajustar una recta a una serie de puntos proporcionados por el escáner láser.

Por lo tanto, el problema consiste en partir de un conjunto de puntos (x_i, y_i), con $i = \{1, 2, \ldots, n\}$, y estimar la recta

$$\widehat{y}_i = ax_i + b$$

que mejor se ajuste a los datos, donde a y b son incógnitas, e $\widehat{y_i}$ es la estimación de y_i. La técnica de mínimos cuadrados se basa en minimizar la suma del cuadrado de los valores residuales (r_i), entendiendo por valor residual la diferencia entre la estimación $\widehat{y_i}$ y el valor real y_i. Es decir, que se desea minimizar:

$$\begin{aligned} R &= \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (\widehat{y_i} - y_i)^2 \\ &= (ax_1 + b - y_1)^2 + (ax_2 + b - y_2)^2 + \cdots + (ax_n + b - y_n)^2 \end{aligned} \tag{4.1}$$

Para minimizar R es necesario igualar a cero sus derivadas parciales con respecto a a y b:

$$\begin{aligned} \frac{\partial R}{\partial a} &= 2(ax_1 + b - y_1)x_1 + \cdots + 2(ax_n + b - y_n)x_n = 0 \\ \frac{\partial R}{\partial b} &= 2(ax_1 + b - y_1) + \cdots + 2(ax_n + b - y_n) = 0 \end{aligned}$$

También se debe comprobar que la segunda derivada parcial de R con respecto a a y b es mayor que cero, para garantizar que se trata de un mínimo y no un máximo. Finalmente, simplificando, resulta el siguiente sistema de ecuaciones lineales:

$$\begin{aligned} a\sum_{i=1}^{n} x_i + n \cdot b &= \sum_{i=1}^{n} y_i \\ a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} x_i \cdot y_i \end{aligned}$$

y resolviéndolo se obtienen los valores de a y b necesarios para determinar la recta.

Una de las principales desventajas del método de mínimos cuadrados es que no es un método robusto, como se explicará más adelante.

4.2. Estimación Robusta vs No Robusta

En estadística se entiende por ***outlier***[1] un dato que se encuentra distante numéricamente con respecto al resto de las observaciones. Supongamos que se está midiendo la temperatura de 10 objetos en una cocina, y la mayoría se

[1]De aquí en adelante se optará por utilizar el término anglosajón *outlier* en lugar de su traducción al español (dato atípico o erróneo).

encuentran entre 20-25 ºC (temperatura ambiente), salvo el horno, que está a 350 ºC. Entonces la media de los datos podría ser cercana a 55 ºC, mientras que la mediana sería unos 23 ºC. De este modo, la mediana reflejaría mejor que la media la temperatura de un objeto tomado al azar. Los 350 ºC en este caso sería un *outlier*. En contraste con lo anterior, el término ***inlier*** sirve para denotar a las observaciones que no están caracterizadas como *outliers*.

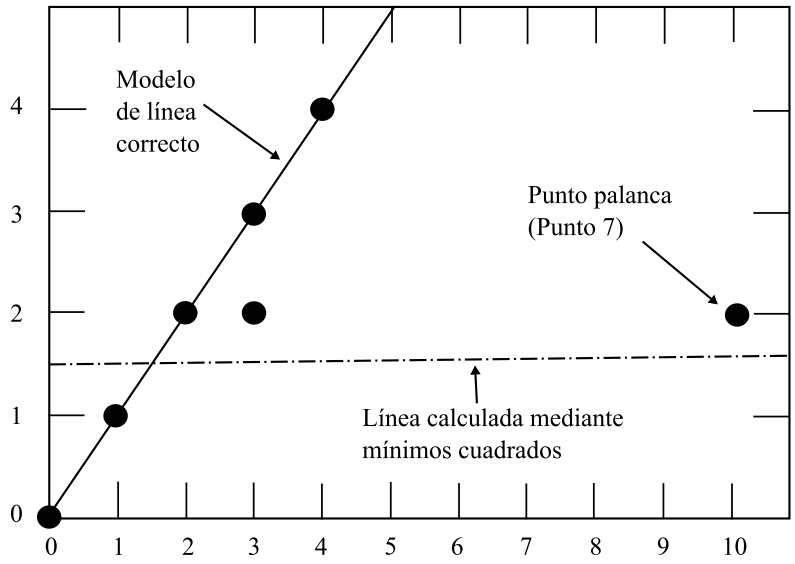
La **tasa de ruptura** (*breakdown point*) se define como el porcentaje de *outliers* que un determinado método de estimación puede tolerar antes de que la estimación se pueda considerar como errónea. En el ejemplo anterior se observa como la media tiene una tasa de ruptura del 0 % puesto que basta un único dato diferente para que el resultado no sea el esperado. Por el contrario la mediana tiene una tasa de ruptura del 50 %, que es la máxima que se puede tener, ya que en caso de estar contaminado más del 50 % de los datos, no sería posible distinguir los datos reales de los *outliers*.

Los métodos de estimación que son capaces de tolerar un determinado porcentaje de *outliers*, es decir, que tienen una tasa de ruptura mayor del 0 %, se denominan **robustos**. Por el contrario se conoce como estimadores **no robustos** a aquellos que en el caso de tratar con *outliers* proporcionarían una estimación poco o nada acertada.

4.3. Algoritmos no robustos

El método de mínimos cuadrados es un ejemplo de algoritmo no robusto. Basta que entre los datos se encuentre un único *outlier* para que la recta estimada sea totalmente incorrecta. Normalmente a estos *outliers* que hacen que la recta estimada se desvíe por completo, se les conoce como puntos palanca (*leverage points*) o puntos envenenados (*poisoned points*). En la fig. 4.1 se muestra un ejemplo de lo explicado. Hay siete puntos representados, de los que seis son *inliers*, y el restante es *outlier*. Para ajustar una recta a todos los puntos, lo correcto sería hacerla pasar por los seis *inliers*, y descartar el *outlier*. Si se realiza un ajuste por mínimos cuadrados, siguiendo la heurística de descartar el peor residual (véase [15]), entonces el resultado del ajuste sería la línea punteada. Es decir, todo se ha echado a perder por un único punto, con lo que se demuestra que la tasa de ruptura del método de mínimos cuadrados es del 0 %.

A continuación se presentarán una serie de algoritmos que, a pesar de no ser robustos, gozan de bastante popularidad. Así pues, se analizarán los métodos denominados *Successive Edge Following* (SEF), *Line Tracking* (LT), *Iterative End Point Fit* (IEPF) y *Split and Merge*. Los dos primeros son procedimientos secuenciales, mientras que los dos últimos son recursivos. Una



Punto	X	Y
1	0	0
2	1	1
3	2	2
4	3	2
5	3	3
6	4	4
7	10	2

Figura 4.1: *Mínimos cuadrados y el punto "palanca".*

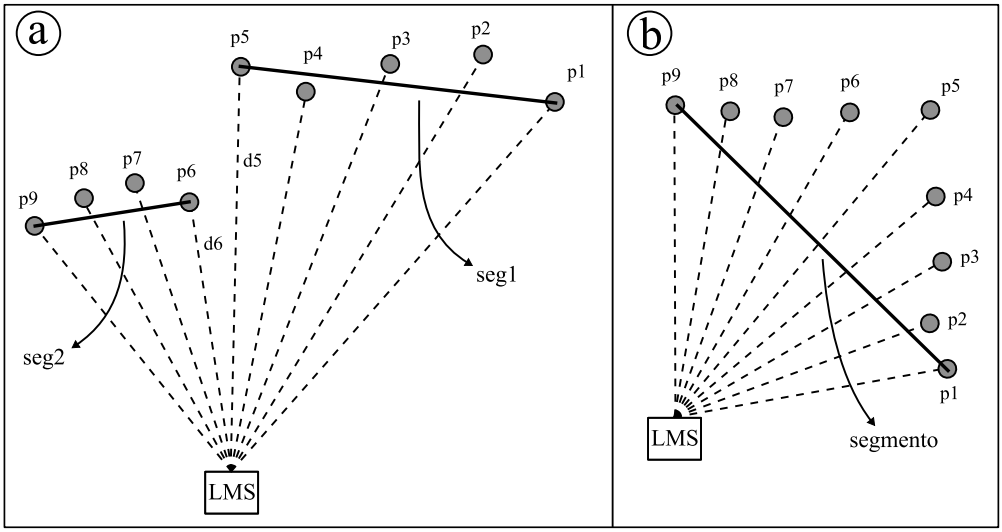


Figura 4.2: *a) SEF detecta dos segmentos porque la diferencia entre dos distancias consecutivas es mayor que el umbral D ($|d6 - d5| > D$). b) SEF falla al analizar un rincón.*

particularidad de estos cuatro algoritmos es que son generalmente muy sensibles a cambios en sus parámetros.

4.3.1. *Successive Edge Following* (SEF)

Este algoritmo ("Seguimiento de Extremos Consecutivos") ([7], [34]) trabaja directamente con las distancias medidas por el escáner láser, que se encuentran en formato polar (ρ, ϕ). Por consiguiente, no se necesita llevar a cabo ninguna transformación a coordenadas cartesianas (x, y). Según este método, se debe considerar que un segmento termina cuando la diferencia entre una distancia y su siguiente ($|\rho_{j+1} - \rho_j|$) excede un determinado valor umbral (T_{max}) fijado con antelación.

Algorithm 2 Successive Edge Following (SEF)

```
1: Entrada: Un conjunto {ρ_1, ..., ρ_n} de medidas de distancia
2: i ← 1
3: para j ← 1 hasta n − 1 hacer
4:     si |ρ_{j+1} − ρ_j| > T_max entonces
5:         S ← {ρ_i, ..., ρ_j}
6:         Almacenar S
7:         i ← j + 1
8:     fin si
9: fin para
```

Al terminar el algoritmo lo único que se obtiene es una serie de conjuntos de distancias, pero éste no es el objetivo. Lo que se pretende es conseguir un conjunto de líneas, y para ello es necesario realizar una etapa de post-procesamiento, para ajustar una línea a cada conjunto mediante algún método de regresión [34]. Aunque es posible efectuar una regresión lineal directamente en coordenadas polares, eso sería mucho más costoso computacionalmente que hacerlo en coordenadas cartesianas [3]. Lo más conveniente es efectuar una transformación de coordenadas para no elevar el tiempo de procesamiento innecesariamente.

Como ventaja principal de este procedimiento destaca su rapidez, mientras que algunos de sus inconvenientes son la dependencia de un umbral no variable y su correspondiente dificultad de ajuste. Además se pueden producir graves errores de detección en determinadas situaciones, como por ejemplo al colocar el láser frente a un rincón. En la fig. 4.2a se observa cómo se detectan correctamente los segmentos porque la distancia entre $p5$ y $p6$ supera el umbral establecido. En cambio, en el caso de la fig. 4.2b, al no superarse

en ningún caso el umbral, se decide erróneamente que el segmento se debe trazar entre $p1$ y $p9$.

4.3.2. *Line Tracking* (LT)

Al igual que el algoritmo anterior, el LT destaca por su simplicidad y rapidez ([7], [34]). Además de *Line Tracking*, también se conoce por el nombre de algoritmo *Incremental* ([24], [27]). Una traducción aproximada al español sería "Algoritmo de Seguimiento de Líneas". El método se basa en calcular la ecuación de la recta ajustada sobre los últimos $n-1$ puntos. A continuación se determina la distancia del punto actual a la recta. Si dicha distancia supera un umbral T_{max} fijado, entonces se comienza una nueva línea. En caso contrario, se recalculan los parámetros de la recta que mejor se ajuste a los últimos n puntos.

Algorithm 3 *Line Tracking* (LT)

```
1: Entrada: un conjunto {p_1, ..., p_n} de puntos
2: i ← 1
3: j ← 1
4: mientras j < n − 2 hacer
5:     R ← recta que se ajusta a los puntos {p_i, ..., p_{j+1}}
6:     T ← distancia euclídea de p_{j+2} a R
7:     si T > T_max entonces
8:         Almacenar R
9:         i ← j + 2
10:        j ← i
11:    si no
12:        j ← j + 1
13:    fin si
14: fin mientras
```

Un posible pseudocódigo de este método es el mostrado en el algoritmo 3. Al comenzar el bucle, se ajusta una recta en base a dos puntos consecutivos. Como es posible que en algunos casos los puntos del *scan* estén demasiado juntos, los parámetros de la recta pueden no ser los esperados. Así pues, quizás sería más aconsejable comenzar el ajuste tomando entre 5 y 10 puntos consecutivos en lugar de dos, de modo que la orientación de la recta sea más fiable. Por otra parte, como se puede apreciar en la quinta línea, se deja libertad en la elección del procedimiento para el ajuste de la recta. Algunas sugerencias de autores que han usado el LT son: ajuste por mínimos

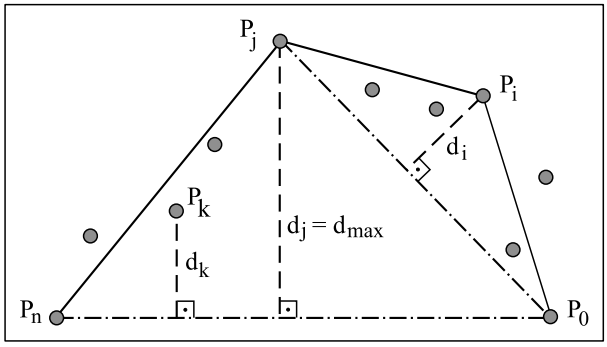


Figura 4.3: *Algoritmo Iterative End Point Fit (IEPF)*

cuadrados totales [27], regresión lineal [38] o técnicas de filtrado de Kalman [37].

Al igual que el SEF, el LT presenta el inconveniente de que es difícil seleccionar un valor adecuado para el umbral T_{max}. Además, se necesita convertir las coordenadas de polares a cartesianas al trabajar con puntos en lugar de con medidas de distancia. Sin embargo, una ventaja del LT es que tiene capacidad para solventar satisfactoriamente el caso reflejado en la figura 4.2b, a diferencia del SEF.

4.3.3. Iterative End Point Fit (IEPF)

Este método, cuya traducción al castellano podría ser "Ajuste Iterativo por Puntos Extremos", tiene su origen en el campo de la visión por computador, al igual que los dos algoritmos anteriores. Aparece descrito por primera vez por Duda y Hart en su conocido libro *Pattern Classification and Scene Analysis* [13].

El modo de funcionamiento del IEPF ([8], [34]) aparece ilustrado en la fig. 4.3. En primer lugar se define una línea tomando los puntos inicial (P_0) y final (P_n) del *scan*. A continuación se busca el punto más alejado (P_j) con respecto a dicha línea. Si el punto encontrado está a una distancia mayor que un determinado valor umbral, entonces se divide el *scan* en dos intervalos. Después de esto el algoritmo comienza recursivamente con cada uno de los segmentos restantes; $\overline{P_0P_j}$ y $\overline{P_jP_n}$ en el caso de la figura. El procedimiento termina una vez que la distancia al punto más alejado sea menor que el umbral fijado, siempre y cuando esta circunstancia se cumpla para todos y cada uno de los segmentos en los que ha sido dividido el *scan*.

Una de las dificultades en el uso del algoritmo es encontrar un valor umbral adecuado con el cual la extracción de líneas se efectúe de manera correcta. Si el umbral es grande, el proceso acabará demasiado pronto, ex-

Algorithm 4 *Iterative End Point Fit* (IEPF)

1: Entrada: $\mathcal{P} \leftarrow \{p_1, \cdots, p_n\}$
2: Poner $\mathcal{P}$ en la lista $\mathcal{L}$.
3: $R \leftarrow$ recta que pasa por los puntos extremos de $\mathcal{P}$
4: Detectar punto p_t con mayor distancia T a la línea R
5: **si** $T > T_{max}$ **entonces**
6: $\quad \mathcal{P}' \leftarrow \{p_1, \cdots, p_t\}$
7: $\quad \mathcal{P}'' \leftarrow \{p_t, \cdots, p_n\}$
8: $\quad \mathcal{L} \xleftarrow{añadir}$ ITERATIVE END POINT FIT($\mathcal{P}'$)
9: $\quad \mathcal{L} \xleftarrow{añadir}$ ITERATIVE END POINT FIT($\mathcal{P}''$)
10: **fin si**
11: **devolver** $\mathcal{L}$

trayendo pocas líneas. Por el contrario, si el umbral es pequeño, se harán demasiadas divisiones. Con respecto a las ventajas, cabe destacar que debido a su buen rendimiento y a que no es computacionalmente costoso, el IEPF y sus derivados son usados habitualmente para la extracción de líneas en los datos proporcionados por sensores láser [8].

4.3.4. Split and Merge

En 1974, tan sólo un año después de la publicación del IEPF, Pavlidis y Horowitz [29] presentaban un método similar con el nombre de *Split and Merge* (Dividir y Unir). Este método de extracción de líneas es probablemente uno de los más populares y ha sido estudiado y utilizado en multitud de trabajos. (Véanse por ejemplo las referencias [14], [21], [25], [32]).

El algoritmo está compuesto principalmente por dos partes. La primera fase es recursiva y consiste en dividir (*split*) los segmentos disponibles en otros más pequeños, mientras que la segunda sirve para fusionar (*merge*) los segmentos que sean prácticamente colineales. La primera fase es casi idéntica al algoritmo IEPF, salvo por una pequeña diferencia. En el IEPF las líneas se construyen simplemente conectando el primer y último punto de un conjunto de puntos. A diferencia de éste, en el *Split and Merge* las líneas se construyen ajustándose a un conjunto de puntos (ya sea por mínimos cuadrados, u otros métodos de ajuste). Por lo tanto se puede considerar que el *Split and Merge* es una modificación ampliada del algoritmo 4.

En la fig. 4.4 se muestra un ejemplo práctico del algoritmo.

Algorithm 5 *Split and Merge*

función SPLIT($\mathcal{P}$)
  Entrada: $\mathcal{P} \leftarrow \{p_1, \cdots, p_n\}$
  Poner $\mathcal{P}$ en la lista $\mathcal{L}$.
  $R \leftarrow$ recta ajustada a los puntos de $\mathcal{P}$
  $p_t \leftarrow$ punto con mayor distancia T a la recta R
  si $T > T_{max}$ **entonces**
    $\mathcal{P}' \leftarrow \{p_1, \cdots, p_t\}$
    $\mathcal{P}'' \leftarrow \{p_t, \cdots, p_n\}$
    $\mathcal{L} \xleftarrow{añadir}$ SPLIT($\mathcal{P}'$)
    $\mathcal{L} \xleftarrow{añadir}$ SPLIT($\mathcal{P}''$)
  fin si
  devolver $\mathcal{L}$
fin función

función MERGE($\mathcal{L}$)
  Entrada: $\mathcal{L} \leftarrow$ lista de conjuntos $\{\mathcal{P}_i, \cdots, \mathcal{P}_n\}$
  para $i \leftarrow 1$ **hasta** $n-1$ **hacer**
    si $\mathcal{P}_i$ y $\mathcal{P}_{i+1}$ son colineales dentro de un margen de error **entonces**
      $\mathcal{Q} \leftarrow \mathcal{P}_i \bigcup \mathcal{P}_{i+1}$
      $R \leftarrow$ recta que se ajusta a los puntos de $\mathcal{Q}$
      $q_t \leftarrow$ punto $\in \mathcal{Q}$ con mayor distancia T a la recta R
      si $T \leq T_{max}$ **entonces**
        $\mathcal{L} \xleftarrow{eliminar} \mathcal{P}_i$ y $\mathcal{P}_{i+1}$
        $\mathcal{L} \xleftarrow{añadir} \mathcal{Q}$
      fin si
    fin si
  fin para
  devolver $\mathcal{L}$
fin función

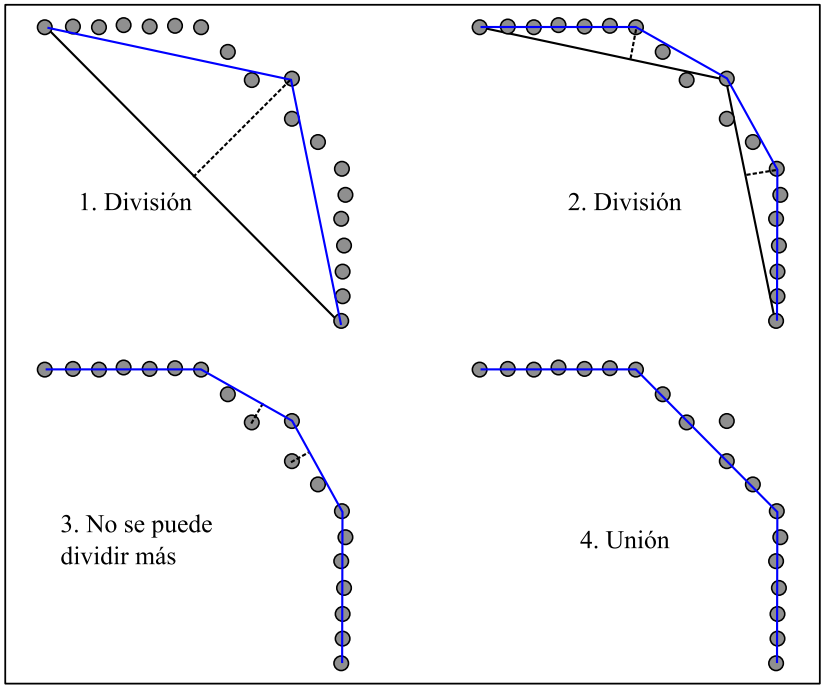


Figura 4.4: *Algoritmo Split and Merge*

4.4. Algoritmos robustos

Existen una serie de algoritmos que, al contrario que los del apartado anterior, son capaces de ofrecer una estimación correcta aunque entre los datos aparezcan *outliers*. Es decir, tienen la particularidad de ser robustos ante fallos o datos que no tienen similitud con el resto. Dentro de esta categoría, se analizarán a continuación el método de mínima mediana, el algoritmo RANSAC, y la Transformada de Hough.

4.4.1. Mínima mediana de cuadrados

Como se expuso en las secciones 4.1 y 4.2, el de mínimos cuadrados es un método no robusto con una tasa de ruptura del 0 %. Con el fin de conferirle una mayor robustez, en 1887 *Francis Y. Edgeworth* propuso una de las primeras variantes. Argumentando que los *outliers* tenían demasiado peso debido a que los residuales estaban elevados al cuadrado (eq. 4.1), propuso sustituir los cuadrados por el valor absoluto de los residuales. De esto modo, se obtiene el *least absolute values* (mínimos valores absolutos, o regresión L_1),

que consiste en minimizar la expresión:

$$R = \sum_{i=1}^{n} |r_i| = \sum_{i=1}^{n} |\widehat{y}_i - y_i|$$

Con esto se consigue solamente eliminar el efecto de los *outliers* en el eje Y, y es por tanto preferible al método de mínimos cuadrados. Sin embargo si los *outliers* aparecen en el eje X, no logra evitar que la estimación sea errónea.

Posteriormente se propusieron diferentes estimadores que trataban de resolver el problema de ajustar una línea a unos datos cuando hay errores graves tanto en el eje X como en el Y (ver [31], pag. 13), pero ninguno de ellos llegaba a alcanzar tan siquiera una tasa de ruptura del 30 %. Este hecho llevó a *Peter J. Rousseeuw* a pensar si sería posible obtener un estimador con una tasa de ruptura del 50 % (la máxima posible). Inspirado en otros autores, finalmente *Rousseeuw* dio con la solución. Dado que la mediana es muy robusta se le ocurrió plantear una variante del mínimos cuadrados, sustituyendo el sumatorio (ver eq. 4.1) por la mediana. De este modo *Rousseeuw* bautizó el método como *least median of squares* (mínima mediana de cuadrados), consistente en minimizar la mediana de los residuales al cuadrado [31]:

$$R = mediana_i\,(r_i^2) = mediana_i\,(\widehat{y}_i - y_i)^2$$

Utilizar la mínima mediana de los residuales tiene varias ventajas. Una de ellas es que el método tiene una tasa de ruptura del 50 %. Esto quiere decir que incluso si la mitad de los datos son *outliers* la estimación será la correcta. Otra ventaja es que admite que los *outliers* aparezcan tanto en el eje X como en el Y.

Sin embargo, no todo son ventajas. Uno de los principales inconvenientes es que no es posible obtener una forma cerrada para calcular la solución final, debido a que la mediana es un estadístico de orden [4]. Para resolverlo es necesario en primer lugar obtener las variables a y b en la estimación $\widehat{y}_i = ax_i + b$. A continuación se calculan los residuales (r_i), y por último la mediana de los residuales. El problema está en que hay muchas posibles combinaciones de a y b y para cada una hay que buscar la mediana, de modo que el espacio de búsqueda para el método de la mínima mediana es enorme. Por consiguiente, es un procedimiento muy costoso computacionalmente, y existe una gran variedad de métodos que intentan optimizar su cálculo abordando el problema desde diversos puntos de vista. La mayoría de estos métodos optan por dar soluciones aproximadas, siendo el algoritmo PROGRESS [31] uno de los que gozan de más popularidad y difusión [28].

La idea básica del algoritmo PROGRESS de *Rosseeuw* consiste en tomar submuestras de p observaciones, siendo p el número de parámetros a estimar

(en nuestro caso $p = 2$). Para cada muestra de 2 puntos se deben calcular los parámetros que determinan la recta en forma punto-pendiente. Después se calcula la mediana de los residuales, y por último se comprueba si es menor la nueva mediana que la última mínima mediana que se tenía almacenada. La dificultad reside en cómo decidir cuántos pares de puntos se deben probar de entre todas las combinaciones posibles. Si se hace una búsqueda completa, resultará que el número total de posibilidades será las combinaciones de n elementos tomadas de p en p:

$$C_p^n = \frac{n!}{(n-p)! \cdot p!}$$

En algunos casos esta búsqueda resultará inviable. En tales casos PROGRESS disminuye el tamaño del espacio de búsqueda realizando una serie de m selecciones aleatorias del siguiente modo (en el caso $p = 2$):

Opción "extensa"	**Opción "rápida"**
Si $n \leq 50$ entonces $m = C_2^n$	Si $n \leq 25$ entonces $m = C_2^n$
Si $n > 50$ entonces $m = 1000$	Si $n > 25$ entonces $m = 300$

donde n es el número total de puntos del conjunto de datos.

4.4.2. Algoritmo RANSAC

RANSAC es el acrónimo correspondiente a RAN*dom* SA*mple* C*onsensus* (Consenso por Muestreo Aleatorio), un algoritmo diseñado por *Fischler* y *Bolles* en 1981 [15]. El propósito general del método consiste en ajustar un modelo matemático (recta, círculo, ...) a un conjunto de datos experimentales. El ajuste además es robusto, puesto que admite la presencia de *outliers* entre los datos.

Algunos métodos de ajuste utilizan todos los datos disponibles para obtener una solución inicial, y después tratan de eliminar los puntos que no son parte del modelo. El procedimiento RANSAC funciona de manera opuesta: utiliza un conjunto de datos tan pequeño como sea posible para realizar la estimación inicial, y posteriormente trata de ampliar el conjunto añadiendo únicamente datos consistentes con el modelo. Por ejemplo, si se va a estimar una recta, se tomarán dos puntos para iniciar el modelo, mientras que para un círculo serían tres puntos los necesarios.

El algoritmo 6 muestra el pseudocódigo correspondiente al algoritmo RANSAC en el caso de una recta. Se parte de un conjunto de puntos, y se ajusta una recta a dos puntos seleccionados al azar. Luego se crea el conjunto de consenso, del que formarán parte los puntos que se pueda considerar que

están próximos a la recta. Si el número de puntos del conjunto de consenso ($\#C$) es mayor que un determinado umbral T, entonces se reajusta la recta al conjunto de consenso (usando mínimos cuadrados, p. ej.). En caso contrario se repite el algoritmo hasta que se alcance un número máximo de iteraciones fijado de antemano.

Algorithm 6 RANSAC

1: Inicio: $P = \{p_1, \ldots, p_n\}$, un conjunto de puntos
2: $S \leftarrow$ muestra aleatoria de 2 puntos de P
3: **repetir**
4: $\quad \mathcal{R} \leftarrow$ modelo de recta usando S
5: $\quad C \leftarrow$ subconjunto de puntos de P que se ajustan al modelo $\mathcal{R}$ dentro de un margen de error (conjunto de *Consenso*)
6: $\quad$ **si** $\#C \geq T$ **entonces**
7: $\quad\quad \mathcal{R}^* \leftarrow$ nuevo modelo de recta ajustada a C
8: $\quad$ **si no**
9: $\quad\quad S \leftarrow$ muestra aleatoria de 2 puntos de P
10: $\quad$ **fin si**
11: **hasta** máximo número de iteraciones o consenso encontrado

En el algoritmo RANSAC quedan tres parámetros sin especificar:

1. El margen de error para decidir si un punto se ajusta o no al modelo.
2. El número máximo de iteraciones.
3. El umbral T, que es el número de puntos que el conjunto de consenso debe tener para considerar que se ha encontrado el modelo correcto.

El margen de error se puede determinar de manera experimental. Por ejemplo se puede hacer una prueba preliminar perturbando los datos, calculando el error medio y después estableciendo el margen de error a una o dos desviaciones típicas con respecto al error medio.

Por otro lado, el número máximo de iteraciones se puede considerar como el número de intentos k que se deben efectuar para lograr obtener un conjunto de n datos que pertenezcan al modelo correcto con una probabilidad z. Si se define w como la probabilidad *a priori* de que un punto se encuentre dentro del margen de error del modelo, k se obtiene como:

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

Por poner un ejemplo, sea $n = 2$ en el caso de que el modelo sea una recta. Si la probabilidad de que un punto cualquiera esté de acuerdo con el modelo es

$w = 0{,}5$ y queremos obtener unas posibilidades de éxito del 98 %, entonces, $k = log(1 - 0{,}98)/log(1 - 0{,}5^2) = 13{,}56$. Es decir tomando unas 14 muestras aleatorias de 2 puntos, tendremos una certeza del 98 % de que el modelo decidido por RANSAC será el correcto.

Finalmente para determinar el número de puntos del conjunto de consenso, se puede hacer del siguiente modo. Sea y la probabilidad de que un punto esté dentro del margen de error de un modelo incorrecto. Se pretende que y^{T-n} sea muy pequeño. Aunque es muy difícil determinar y con precisión, es razonable asumir que será menor que w. Suponiendo que $y < 0{,}5$, un valor $T - n = 5$ proporcionará una probabilidad mayor del 95 % de que los puntos elegidos no pertenezcan a un modelo incorrecto. Dicho de otro modo, si $y < 0{,}5$ y $n = 2$, entonces bastará que el conjunto de consenso tenga $T = 7$ puntos para considerarse que se ha encontrado el modelo correcto con una probabilidad de al menos un 95 %.

4.4.3. Transformada de Hough

Del campo de procesamiento digital de imágenes procede la Transformada de Hough, una de las técnicas más populares de extracción de características. La transformada original únicamente es capaz de extraer líneas en una imagen dada, aunque se puede generalizar con el fin de identificar formas más complejas, como curvas, por ejemplo.

Paul Hough patentó su transformada en 1962 (bajo la patente estadounidense nº 3,069,654) con el título *"Method and Means for Recognizing Complex Patterns"*. El inconveniente de este original método es que utilizaba la definición de la recta en la forma punto-pendiente, $y = mx + n$, con lo que el espacio de transformación era ilimitado, puesto que la pendiente m puede tomar un valor infinito. Debido a esta inconveniencia, la transformada tal y como se usa hoy en día es ligeramente diferente a la patentada por Hough, y fue descrita por Richard Duda y Peter Hart en 1972 [12]. Esta variante utiliza la definición de la recta en forma normal (fig. 4.5), $x \cdot cos(\theta) + y \cdot sin(\theta) = \rho$, y gracias a esto el espacio de transformación deja de ser infinito.

Descripción de la transformada

El objetivo de la Transformada de Hough es detectar un conjunto de píxeles que formen una línea recta. Los píxeles se obtienen de una imagen binaria en la que el color negro identifica obstáculos y el color blanco espacios vacíos, o viceversa. Por lo tanto, se parte de un conjunto de n puntos $\mathcal{P} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ que representan los obstáculos en la imagen, y se desea encontrar el conjunto de líneas que se ajustan a $\mathcal{P}$. Para ello hay que

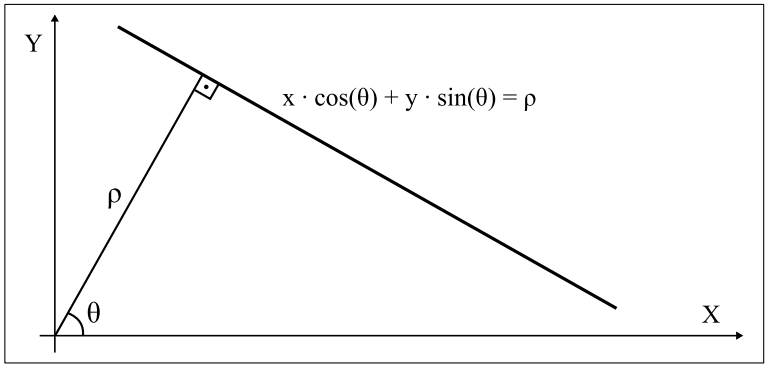


Figura 4.5: *Recta en forma normal*

transformar cada punto (x_i, y_i) de $\mathcal{P}$, ubicado en el plano cartesiano XY, en una curva sinusoidal en el plano ΘP. ¿Por qué se transforma un punto en una sinusoide, y no un punto en otro punto? Si se utiliza la ecuación de la recta en forma normal:

$$\rho = x_i \cdot cos(\theta) + y_i \cdot sin(\theta)$$

y se sustituyen en ella los valores fijos x_i e y_i, entonces resulta que, cumpliendo la ecuación, existen infinitas combinaciones de valores para el par (θ, ρ). Si se hace un gráfico con los valores obtenidos para θ y ρ, se puede comprobar que el resultado es una curva sinusoidal.

La peculiaridad de esta transformación está en que si los puntos analizados en el plano XY forman una recta, entonces las curvas sinusoidales generadas por la transformación se cortarán en un único punto. Esto tiene mucho sentido, pues si en el espacio de transformación ΘP se toma un único punto fijo (θ_0, ρ_0), y se sustituye en la ecuación:

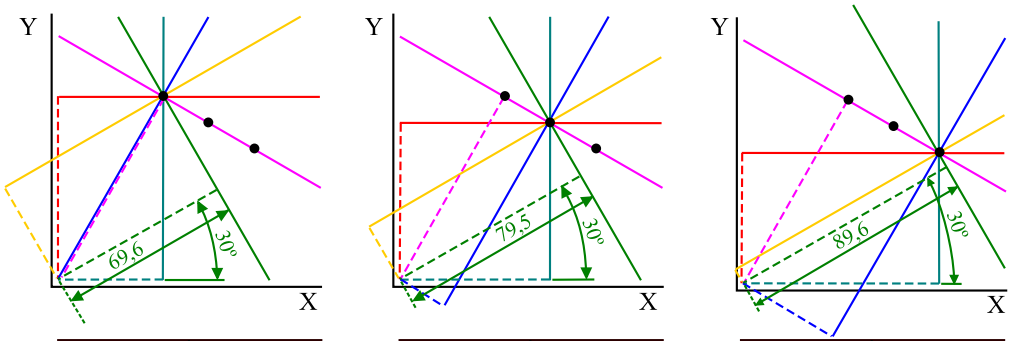
$$\rho_0 = x \cdot cos(\theta_0) + y \cdot sin(\theta_0)$$

entonces resultará que hay infinitas combinaciones de valores (x, y), con la característica de que estarán situados sobre una misma recta.

Se pueden resumir todas estas interesantes propiedades de la Transformada de Hough del siguiente modo:

1. Un punto en el plano XY corresponde a una curva sinusoidal en el plano ΘP.

2. Un punto en el plano ΘP corresponde a una línea recta en el plano XY.



Ángulo	Distancia
0	40
30	69,6
60	81,2
90	70
120	40,6
150	0,4

Ángulo	Distancia
0	57,1
30	79,5
60	80,5
90	60
120	23,4
150	-19,5

Ángulo	Distancia
0	74,6
30	89,6
60	80,6
90	50
120	6,0
150	-39,6

Figura 4.6: *Cálculo de los parámetros de las rectas en forma normal.*

3. Puntos de una misma recta en el plano XY corresponden a curvas sinusoidales que se cortan en un único punto en el plano ΘP.

4. Puntos de una misma curva en el plano ΘP corresponden a rectas que se cortan en un único punto en el plano XY.

Ejemplo práctico

Supóngase que se dispone de tres puntos en coordenadas cartesianas, tal y como se ilustra en la fig. 4.6. En dicha figura, el proceso de cálculo de la Transformada de Hough consiste en trazar una serie de líneas continuas a través de cada punto. Para cada una de estas líneas, se traza una línea perpendicular (normal) en trazo discontinuo, de modo que pase por el origen de coordenadas. Finalmente se miden el ángulo y la longitud de las normales (líneas discontinuas) y se anotan en una tabla. A continuación se crea una gráfica de ángulos *vs.* distancias (espacio de Hough), cuyo resultado se muestra en la fig. 4.7. En esta imagen, aparecen tres curvas sinusoidales, y cada una corresponde a uno de los puntos de la fig. 4.6. Hay un punto en el que se produce una intersección de las tres curvas, y sus coordenadas indican la distancia y el ángulo de la recta que pasa por los tres puntos de la fig. 4.6.

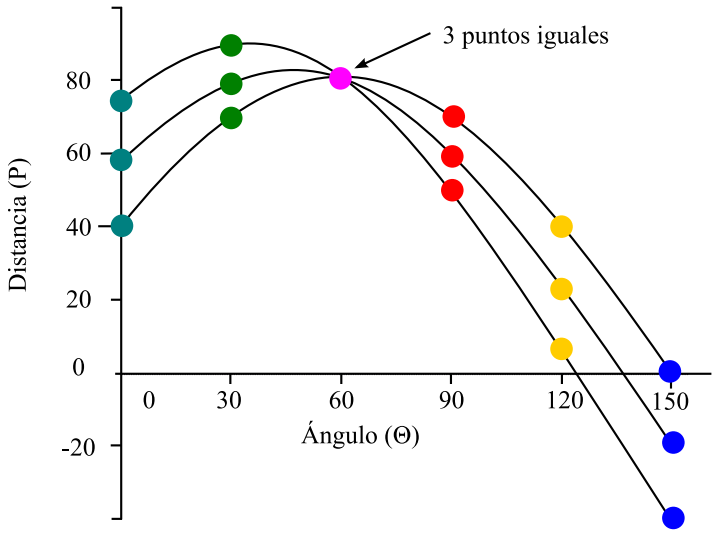


Figura 4.7: *Curvas sinusoidales en el espacio de Hough.*

Implementación

Como se ha visto, la Transformada de Hough nos proporciona los medios necesarios para detectar puntos colineales, o lo que es lo mismo, rectas. Para lograrlo, en primer lugar se necesita obtener una representación discreta del espacio de la Transformada de Hough (el plano ΘP). Se necesita decidir la resolución angular con la que se va a hacer el muestreo. Por ejemplo, 0.5º de resolución pueden ser suficientes, en cuyo caso tendremos $360º/0.5º = 720$ valores discretos para θ. De manera similar, se debe muestrear ρ decidiendo la máxima distancia desde el origen a la que se espera detectar una recta, por ejemplo, 500 cm, y la resolución de distancia, 1cm por ejemplo. El espacio de Hough (ΘP), ahora es un array bidimensional de dimensiones 500x720, donde cada elemento del array corresponde a un valor particular de ρ y θ. A este array se le llama *acumulador*, puesto que va a ser utilizado para acumular la evidencia de curvas correspondientes a la transformación de ciertos puntos (x, y) en la imagen original. Para cada punto (x_i, y_i) en el plano cartesiano, se incrementan en una unidad todas las celdas del acumulador de modo que las coordenadas de la celda (ρ, θ) satisfagan la ecuación $x_i \cdot cos(\theta) + y_i \cdot sin(\theta) = \rho$.

Cuando se ha repetido este proceso para todos los puntos (x_i, y_i), entonces se buscan en el acumulador las celdas con mayor valor, puesto que dichos valores corresponderán a rectas en la imagen. Por lo tanto la tarea de detección de rectas se ha transformado en la tarea de encontrar máximos locales en el acumulador (ρ, θ) del espacio de Hough. El pseudocódigo correspondiente a la Transformada de Hough se refleja en el algoritmo 7.

Algorithm 7 Algoritmo para la Transformada de Hough

1: Discretizar el espacio de la Transformada de Hough: identificar los valores máximos y mínimos de ρ y θ, y la resolución del acumulador para ρ y θ.
2: Crear un array acumulador $A(\rho, \theta)$ e iniciar todos sus valores a cero.
3: **para cada** punto (x_i, y_i) **hacer**
4: **para cada** angulo θ **hacer**
5: Calcular ρ en $x_i \cdot cos(\theta) + y_i \cdot sin(\theta) = \rho$
6: Incrementar el acumulador $A(\rho, \theta)$
7: **fin para**
8: **fin para**

Transformada de Hough para un barrido láser

La Trasformada de Hough está pensada originalmente para imágenes en las que un píxel negro indica un obstáculo y uno blanco indica vacío, o viceversa. Puesto que un barrido láser consiste en un conjunto de puntos que representan los obstáculos encontrados en el entorno, aplicar la Transformada de Hough a un barrido láser es muy similar. La única diferencia es que en una imagen, hay que revisar punto por punto y hacer la Transformada, mientras que en un barrido láser hay que transformar cada uno de los puntos que lo componen (361 normalmente), con lo cual es un proceso mucho menos costoso computacionalmente. Los pasos que hay que seguir para transformar un barrido láser son los mismos del algoritmo 7.

5

Métodos propuestos

En este capítulo se propondrán dos nuevos métodos destinados a la extracción de características. El primero es un método de *clustering* que permite dividir en subgrupos los datos de cada barrido del escáner láser. El segundo método que se propone, modifica en cierta medida la Transformada de Hough para extraer líneas más rápidamente que la transformada original, al tiempo que está inspirado en el algoritmo *Line Tracking*.

5.1. *Clustering* por Convolución de Distancias

Los métodos de *clustering* analizados en el capítulo 3 se podían englobar bajo el esquema general del algoritmo 1, en el cual la principal dificultad consistía en determinar el valor de una distancia umbral. Dicho umbral indicaba la distancia euclídea máxima que podía existir entre dos puntos consecutivos para que ambos pertenecieran al mismo *cluster*. Si la distancia excediera el umbral fijado, entonces se consideraría que los puntos no deberían formar parte del mismo *cluster*.

Uno de los problemas más importantes que surgen a la hora de establecer dónde termina un *cluster* y comienza el siguiente, es el que se presenta en la figura 5.1, en la cual se ilustra un fragmento de los datos de un barrido

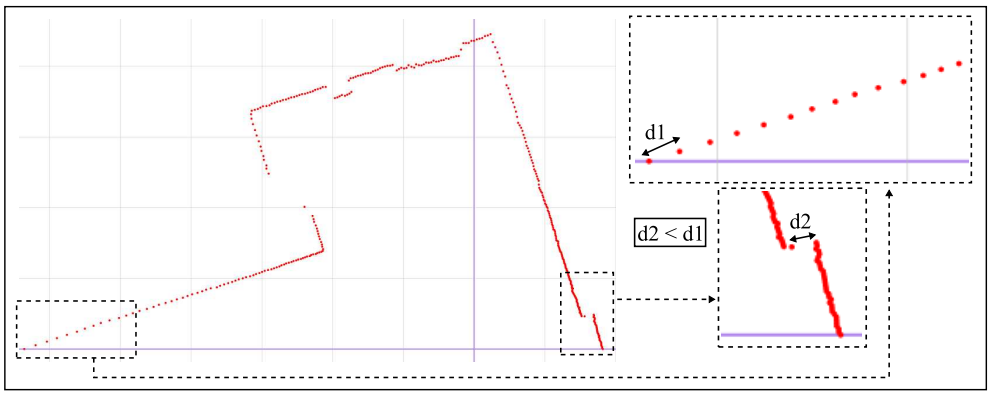


Figura 5.1: *Problema de distancias en las técnicas de clustering*

realizado por el láser en un escenario real. En la parte derecha del gráfico se encuentran dos ampliaciones de las porciones clave del barrido. Se puede observar cómo la distancia $d2$ es menor que la distancia $d1$. Si se utilizara un valor fijo como distancia umbral, se podría dar el caso de que si se considera que $d2$ representa un cambio de *cluster*, entonces $d1$ también debería serlo, puesto que es mayor que $d2$. Esta decisión implicaría el grave error de que el *cluster* calculado en base a $d1$ sería un falso positivo, y esto nunca es deseable. Por este motivo se suelen emplear técnicas en las que la distancia umbral se calcula de manera adaptativa. En el capítulo 3 se discutieron tres métodos adaptativos, cada uno de los cuales dependía de una serie de constantes y parámetros que debían ser estudiados a fondo con el fin de obtener el mejor rendimiento posible.

Uno de los aspectos más deseables a la hora de diseñar un algoritmo es que no se establezca una dependencia con respecto a parámetros configurables, de modo que el algoritmo funcione a la perfección sin necesidad de afinar o ajustar ninguna variable. Un diseño de este tipo supondría un gran logro, puesto que evitaría horas de pruebas para lograr el rendimiento óptimo. En este sentido, el algoritmo de *clustering* que se propone en esta sección da un paso adelante en la eliminación de los parámetros relacionados con la distancia, tal y como se verá posteriormente.

En lugar de establecer una distancia fija o adaptativa para decidir cuándo comenzar un nuevo *cluster*, se pueden plantear otras alternativas. Por ejemplo, si se calculan las distancias euclídeas entre cada par de puntos consecutivos y se representan en una gráfica, se obtiene el resultado de la fig. 5.2. Se puede apreciar que los puntos marcados con los números del 1 al 5 corresponden a puntos en los que se debe terminar un *cluster* y comenzar el siguiente. Cuando las distancias son homogéneas, en la gráfica aparecen rec-

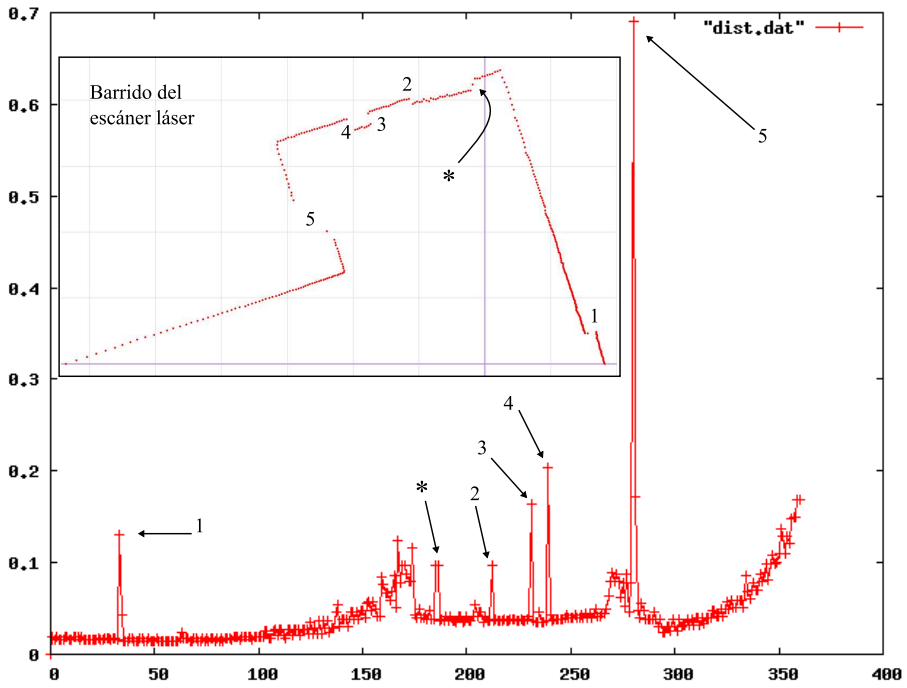


Figura 5.2: *Distancia entre puntos consecutivos junto con el barrido láser original.*

tas, como por ejemplo en las inmediaciones del punto marcado como 1. En cambio, cuando las distancias crecen progresivamente, como ocurre a partir del punto etiquetado como 5 hasta el final, en la gráfica de distancias aparecen curvas ascendentes. Mención especial merece el punto marcado con un asterisco, puesto que realmente es difícil decidir si en ese caso los *clusters* se deben separar o no.

Por lo tanto está claro que para obtener los *clusters* hay que ser capaces de detectar los puntos que están muy separados con respecto al resto de los puntos de la gráfica de distancias. Éste es un problema complejo, y una de las posibilidades para resolverlo es utilizar un filtro pasa alta, ya que permiten pasar las altas frecuencias mientras que atenúan las bajas. Una de las opciones para implementar filtros en una computadora es hacer uso de la convolución (o suma de convolución), cuyo uso es muy habitual en el campo de procesamiento digital de imágenes y en el campo de procesamiento digital de señales [36]. Las aplicaciones de la convolución son muy variadas, y abarcan desde filtrado lineal de imágenes, hasta la detección de bordes, extracción de características, reconocimiento de patrones, etc.

La convolución es un operador matemático, que transforma dos funciones f y g en una tercera función que en un cierto sentido representa la magnitud

en la que se superponen f y una versión trasladada de g. La convolución de f y g se denota como $f*g$. Tomando como modelo la ecuación de la convolución discreta en una dimensión aplicada a una señal digital ([36], p. 120), se puede efectuar una adaptación a los datos que proporciona un escáner láser. Si $r = [1, \ldots, n]$ es un vector de n elementos, y $k = [-(m-1)/2, \ldots, (m-1)/2]$ es otro vector de m elementos, con m impar, entonces la convolución discreta de ambos vectores será otro vector $y = r * k$, de n elementos, donde cada elemento vendrá dado por:

$$y[i] = \sum_{j=-\frac{m-1}{2}}^{\frac{m-1}{2}} r(i-j) \cdot k(j)$$

El láser utilizado en este trabajo, normalmente proporciona 361 distancias en cada barrido, las cuales se deberán colocar en el vector r. El vector k se conoce por diferentes nombres, normalmente filtro, máscara, o kernel de convolución. Dependiendo de los valores que se almacenen en el kernel, el proceso de filtrado ofrecerá diferentes resultados. En los filtros pasa alta el valor central del kernel de convolución suele ser positivo, y el resto de valores a la derecha e izquierda son negativos. De este modo, al aplicar la convolución se potenciarán los valores del vector r que estén aislados del resto. De entre todos los kernels que se han probado con datos experimentales, hay varios que han proporcionado muy buenos resultados, concretamente los vectores: $k1 = [-3, -3, 5, -3, -3]$ y $k2 = [-1, -2, -3, 5, -3, -2, -1]$.

En la fig. 5.3 se muestra el resultado de aplicar el kernel de convolución $k1$ a los datos de la figura anterior. Se puede observar que la mayoría de los elementos de la gráfica toman el valor cero, a excepción de los cinco picos que aparecen, y que coinciden exactamente con los analizados manualmente en la figura anterior.

Por lo tanto da la impresión de que una vez calculada la convolución basta con seleccionar los elementos que tengan un valor distinto de cero. Según lo descrito hasta el momento, este procedimiento no tendría en cuenta el ruido asociado a las medidas del sensor. Se ha comprobado de manera experimental que cuando se detectan objetos muy próximos al sensor, los puntos están tan extremadamente juntos que en ocasiones aparecen picos correspondientes a distancias de separación extremadamente pequeñas. Para solucionar este pequeño inconveniente basta con obtener la desviación estándar del error en la medida de los datos del láser, y seleccionar solamente los picos que superen este valor. Como los valores que aparecen la gráfica de la convolución van multiplicados por el valor central del kernel de convolución, se deberá multiplicar la desviación estándar del láser por el valor central del kernel de convolución

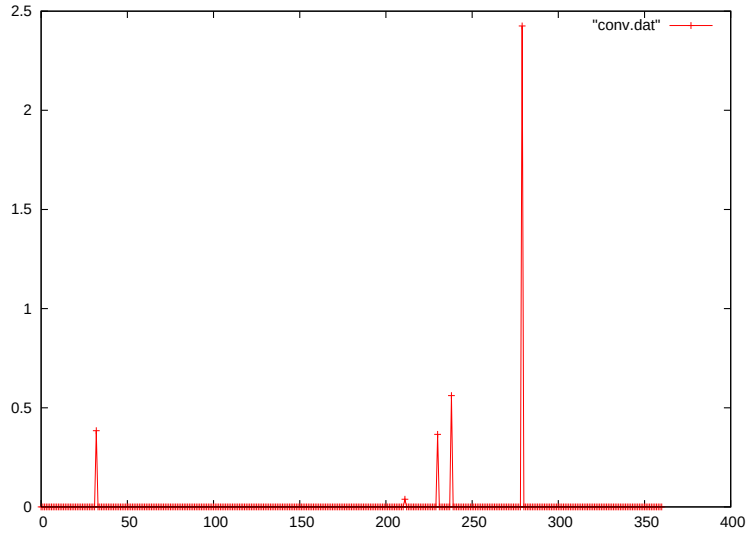


Figura 5.3: *Resultado de la convolución aplicada a los datos de la fig. 5.2.*

Algorithm 8 *Clustering* por Convolución de Distancias (CCD)

$P \leftarrow \{P_1, \ldots, P_n\}$ ▷ Medidas del láser
$K \leftarrow \{K_1, \ldots, K_m\}$ ▷ Kernel de Convolución
$\sigma \leftarrow$ desviación estándar del láser
para $i \leftarrow 1$ **hasta** $n - 1$ **hacer**
    $D_i \leftarrow distancia\_euclidea(P_i, P_{i+1})$
fin para
para $i \leftarrow 1$ **hasta** n **hacer**
    $suma \leftarrow 0$
    para $j \leftarrow -[(m-1)/2]$ **hasta** $[(m-1)/2]$ **hacer**
        $suma \leftarrow suma + D_{i+j} \cdot K_{j+[(m+1)/2]}$
    fin para
    $C_i \leftarrow suma$ ▷ Suma de Convolución
    si $C_i > \sigma K_{(m+1)/2}$ **entonces**
        P_i pertenece a un nuevo *cluster*
    si no
        P_i pertenece al *cluster* actual
    fin si
fin para

utilizado, para garantizar un cierto margen de seguridad. En nuestro caso particular la desviación estándar del error en la medida del láser es $\sigma = 1cm$, y el valor central del kernel es de 5 unidades. De este modo, una vez calculada la convolución, se seleccionarán únicamente aquellos puntos que tengan un valor mayor o igual que $5cm$.

5.2. Extracción de líneas REHOLT

El método de extracción de líneas que aquí se presenta está pensado para ser utilizado en cada uno de los *clusters* que proporcionaría el algoritmo de la sección anterior (sección 5.1), o alguna de las técnicas de *clustering* del capítulo 3. En un principio, se pensó denominar el método como "Seguimiento de Líneas basado en la Transformada de Hough reducida", pero debido a la dificultad de encontrar un acrónimo adecuado, se pensó en traducir el título al inglés, obteniendo REHOLT: RE*duced* HO*ugh* T*ransform* L*ine* T*racking*.

Como indica su nombre, REHOLT está inspirado por un lado en el algoritmo *line tracking* (secc. 4.3) y por otro lado en una versión modificada de la Transformada de Hough (secc. 4.4), que se ha optado por llamar "Transformada de Hough reducida". La figura 5.4 junto con el algoritmo 9 servirán para explicar con detalle el funcionamiento esencial de REHOLT.

La idea inicial del algoritmo consiste en tomar como referencia un punto inicial, p_{base}, y otro punto, p_j, que esté alejado de p_{base} una distancia mayor o igual que un valor $D1$ fijado de antemano. Una vez hecho esto, se obtiene la recta $\mathcal{R}$ que pasa por los puntos p_{base} y p_j (ver fig. 5.4a). Después se calcula la distancia d del punto actual a la recta $\mathcal{R}$, hasta que en un determinado momento d sea mayor que una determinada distancia umbral $D2$ (ver fig. 5.4b). Acto seguido se calcula la Transformada de Hough reducida (basándose en la recta $\mathcal{R}$) para el conjunto de puntos $(p_{base}, \ldots, p_j)$, obteniendo un nuevo modelo de recta $\mathcal{R}'$. Para finalizar, se busca un nuevo punto base, desde el cual repetir el algoritmo. Este punto se calculará como el punto más cercano a la proyección de p_j en $\mathcal{R}'$ (ver fig. 5.4c). Una vez terminada la detección de rectas, se deberán buscar segmentos consecutivos que sean colineales, agruparlos y calcular la Transformada de Hough reducida para el nuevo grupo de puntos.

5.2.1. Transformada de Hough reducida

Es bien sabido que el efectuar la Transformada de Hough para un conjunto de puntos es un proceso muy eficaz y robusto, pero a la vez muy costoso computacionalmente. El alto coste computacional se debe a que es necesario

Algorithm 9 Algoritmo REHOLT

1: Entrada: un conjunto $\{p_1, \ldots, p_n\}$ de puntos
2: $base \leftarrow 1$
3: $j \leftarrow base$
4: **repetir**
5: $\quad j \leftarrow j + 1$
6: $\quad d \leftarrow$ distancia_euclidea$(p_{base}, p_j) >= D1$
7: **hasta** $d >= D1$
8: $\mathcal{R} \leftarrow$ recta que pasa por p_{base} y p_j
9: **mientras** $j < n$ **hacer**
10: $\quad d \leftarrow$ distancia_euclidea$(p_{j+1}, \mathcal{R})$
11: $\quad$ **si** $d > D2$ **entonces**
12: $\quad\quad \mathcal{R}' \leftarrow$ Transformada de Hough de los puntos $(p_i, \ldots, p_j)$ reducida según $\mathcal{R}$
13: $\quad\quad p_{base} \leftarrow$ punto más cercano a la proyección de p_j en R'
14: $\quad\quad$ Almacenar la recta R y repetir el algoritmo desde la línea 3
15: $\quad$ **si no**
16: $\quad\quad j \leftarrow j + 1$
17: $\quad$ **fin si**
18: **fin mientras**
19: Fusionar los segmentos de recta que sean colineales, aplicando la Transformada de Hough reducida

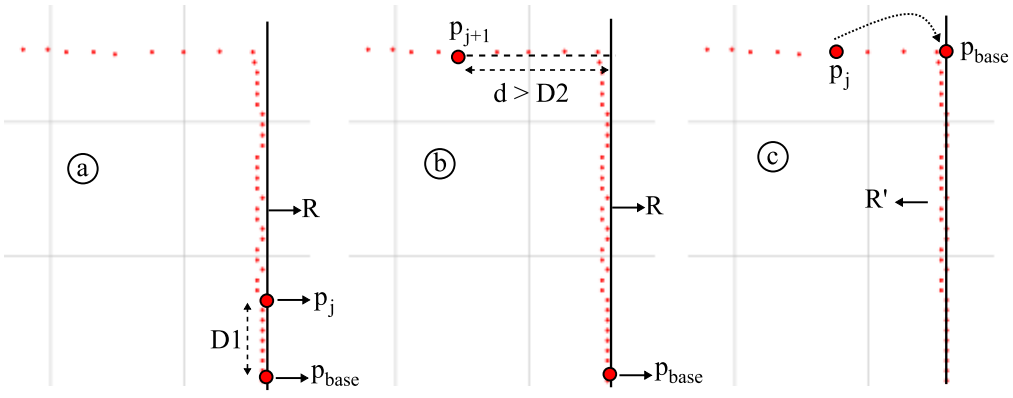


Figura 5.4: *Representación gráfica del algoritmo 9*

convertir el espacio de Hough en un acumulador. Supóngase que el tamaño de las celdas del acumulador es de 0.1º para ϕ, y 1 cm, para ρ. Entonces, si se quieren detectar rectas que se encuentren, por ejemplo a una distancia de entre 0 y 15 metros, y con una posible orientación desde 0º hasta 360º, se tendrá que el tamaño del array acumulador correspondiente al espacio discretizado será de $1500 \times 3600 = 5400000$. En este acumulador de 5 millones de celdas habrá que realizar procesos de búsqueda para detectar las celdas que correspondan a máximos relativos, la cuales serán las rectas detectadas en el conjunto de puntos inicial.

La "Transformada de Hough reducida" que en este texto se propone, consigue disminuir el tiempo de cómputo reduciendo el espacio de búsqueda en el acumulador del espacio de Hough, basándose en la estructura de REHOLT. En este algoritmo, en un determinado momento, se consigue obtener un conjunto de puntos al que se aplica la transformada, y se puede aprovechar el hecho de que la recta $\mathcal{R}$ representa una aproximación de la dirección y orientación que tendrá la recta calculada por la transformada. Puesto que la Transformada de Hough trabaja con la recta en forma normal, con parámetros (ρ, ϕ), habrá que obtener estos mismos parámetros para la recta $\mathcal{R}$. La reducción del espacio de Hough se logra disminuyendo las dimensiones del acumulador a los parámetros (rho, phi) de $\mathcal{R}$ más un cierto margen de seguridad.

Por ejemplo, supóngase que la orientación y distancia estimadas de la recta que se va a calcular son $\rho = 8metros$ y $\phi = 90º$. Entonces tomando los mismos tamaños de celda que en el párrafo anterior (1 cm para ρ y 0.1º para ϕ), y poniendo un márgen de seguridad de $\pm 1m$ y $\pm 20º$, resultará que las dimensiones del acumulador serán: $200 \times 400 = 80000$. Es decir, la búsqueda de la recta se realizará entre 6 y 8 metros, y entre 70º y 110º, y además solamente habrá que detectar una única recta, con lo que simplemente bastará con buscar el máximo absoluto del acumulador, y se evitará tener que buscar máximos relativos.

6

Robot y *Software* de Validación

Con el propósito de validar y evaluar el rendimiento, tanto de los algoritmos analizados en los capítulos 3 y 4, como de los métodos propuestos en el capítulo 5, se ha desarrollado una herramienta *software* en la que se incluyen todas las funcionalidades para lograr tal fin. Además, se ha construido un robot móvil que incluye un sensor láser, de modo que se pueden obtener datos del entorno en diferentes lugares, tanto en el interior como en el exterior de la Facultad de Ciencias de la Universidad de Salamanca.

6.1. Construcción del robot

Antes de emprender la construcción de un robot es necesario estudiar, planear y decidir todos los elementos que lo constituirán, así como el emplazamiento físico de cada uno de ellos. A continuación se van a describir los principales subsistemas por los que está compuesto el robot:

- Una plataforma física, cuyo criterio de diseño dependerá del entorno, el peso y el número de componentes que formen parte del robot.
- Sistema de alimentación: baterías, fuentes de alimentación, conversores de potencia...
- Mecanismos de actuación: motores, servos, ruedas, ...

Figura 6.1: *Aspecto final del robot*

- Mecanismos de percepción: sensores de luz, calor, movimiento, ultrasonidos, presión, inclinación, ...
- Sistema de procesamiento: ordenadores de propósito general, sistemas empotrados, procesadores digitales de señal ...

6.1.1. Construcción de la plataforma

En la figura 6.1 se muestra el robot terminado. En la mitad delantera de la plataforma se aprecia la disposición del sensor láser, mientras que en la mitad trasera se encuentran un mini-pc, y dos conversores DC-DC, uno de 12V a 24V y otro de 12V a 5V. Encima de la pieza de madera trasera se reserva sitio para colocar el sistema de alimentación. En la parte inferior de la plataforma se colocan las ruedas, en disposición diferencial, y el resto de componentes electrónicos.

6.1.2. Sistema de alimentación

Como sistema electrónico que es, el robot necesita alimentación eléctrica. El esquema de alimentación eléctrica del robot aparece ilustrado en la fig. 6.2.

Para lograr independencia de cables, la alimentación eléctrica se proporciona a través de baterías de plomo/calcio de 12V y 7A/h. Además, se necesitan convertidores de potencia para poder proporcionar el voltaje adecuado a cada componente. En el robot hay tres componentes que requieren un determinado voltaje específico para poder funcionar: un mini-pc, un escáner

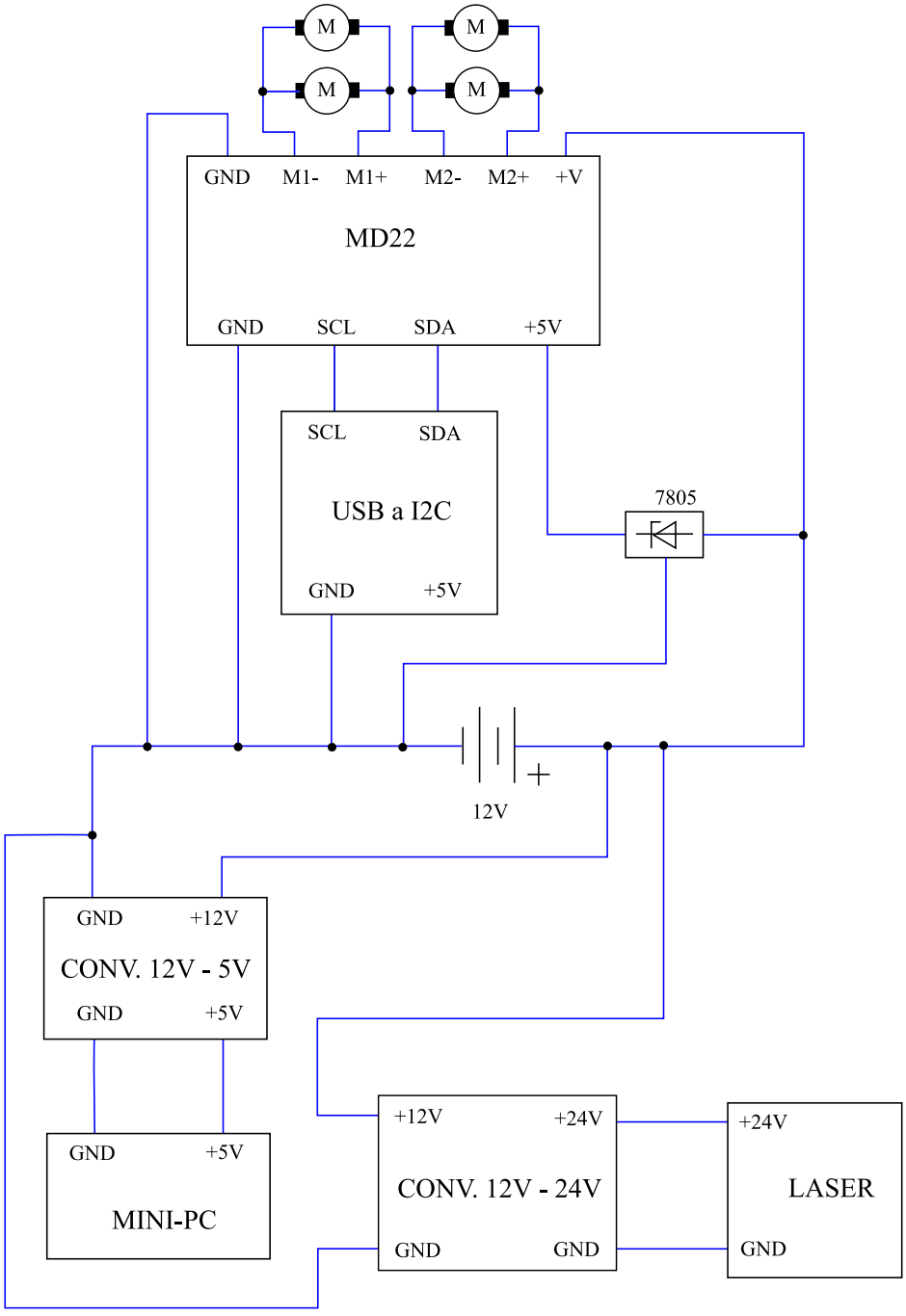


Figura 6.2: *Esquema eléctrico general del robot*

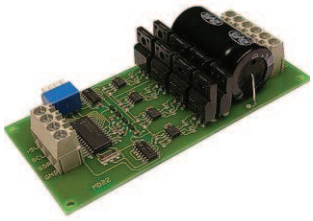


Figura 6.3: *Controlador de motores MD22*

láser SICK, y cuatro motores DC. El mini-pc necesita 5V, y se ha colocado un convertidor DC-DC conectado a las baterías que transforma 12V en 5V. El láser necesita una alimentación de 24V, y se ha instalado otro convertidor DC-DC que permite transformar los 12 V de la batería en los 24V que necesita este sensor.

Los motores, por su parte, se han conectado de tal manera que los cuatro motores actúan como si sólo fueran dos. Esto es, en la parte izquierda del robot se han colocado dos motores conectados en paralelo, de modo que al enviarle un determinado voltaje a un motor para que gire, el motor contiguo gire en el mismo sentido. En la parte derecha del robot se han colocado otros dos motores con la misma configuración. Para poder controlar los motores correctamente es imprescindible utilizar un *driver* de potencia. En este caso se ha elegido el módulo controlador MD22 (fig. 6.3), que permite controlar dos motores de corriente continua de mediana potencia, y está diseñado para proporcionar más potencia que los controladores basados en un único circuito integrado.

El MD22 genera mediante una bomba de carga los 15V de la tensión de control del MOSFET, por lo que sólo se requieren 5V a 50 mA para la alimentación del circuito, además de la alimentación del motor, que en este caso son 12V a 6mA. Para proporcionar la tensión de 5V al módulo controlador MD22, se utiliza un regulador de tensión 7805, que acepta tensiones de entrada de entre 7 y 25 voltios. El MD22 puede controlarse de 5 formas diferentes:

1. Modo bus I2C. Hasta ocho módulos MD22 con direcciones seleccionables mediante micro interruptores y 4 modos de funcionamiento incluyendo el control de dirección (modo diferencial).

2. Dos entradas analógicas independientes de 0V - 2.5V - 5V. Un sentido es 0V; 2,5V es parado y 5V es la otra dirección.

3. Una entrada analógica 0V-2,5V-5V para el control de la velocidad y la otra para el control de dirección.

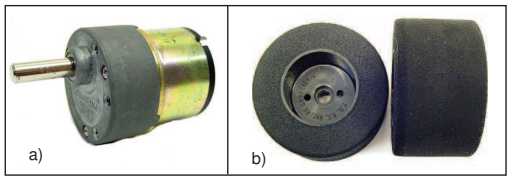


Figura 6.4: ***a)*** *Motores del robot.* ***b)*** *Ruedas del robot.*

4. Modo RC de canales independientes. Para controlarlo directamente desde un receptor de radio control estándar. Cada canal controla un motor de forma independiente.

5. Modo RC con control de dirección. Un stick controla la velocidad, mientras que el otro controla la dirección.

De entre todos estos modos solamente nos interesa el primero, porque utiliza el bus I^2C para controlar los motores, y en el mini-pc se puede disponer de un bus I^2C a través de un conversor USB a I^2C.

6.1.3. Mecanismos de actuación

Para la tracción del robot se han elegido cuatro motores DC como el de la fig. 6.4a. Es un motor que necesita una alimentación de 12 V, tiene un consumo de 6mA y una fuerza de 8,8 Kg/cm. Como máximo puede alcanzar 120 revoluciones por minuto, pero también cabe la posibilidad de activar una reductora con una proporción de 50:1. Gracias a esto, por cada 50 revoluciones internas, el eje externo del motor solamente da una vuelta y se consigue una potencia de tracción mucho mayor. Para transferir el movimiento rotatorio del motor al suelo se han empleado ruedas de gran adherencia, como las de la figura 6.4b.

6.1.4. Mecanismos de percepción

La interacción de un robot con su entorno es esencial para que éste alcance sus objetivos. Si un robot no puede percibir un evento, tampoco podrá reaccionar ante él. De ahí que cada vez exista una mayor demanda de tecnología para percibir cualquier tipo de estímulo. Algunos ejemplos de sensores pueden ser los de presión, temperatura, distancia, posición, elevación, fuerza, visión, etc. En el segundo capítulo, se expuso una comparación entre diferentes sensores. El sensor láser ha sido el sensor principal utilizado en este trabajo, debido a que posee determinadas ventajas con respecto a sus principales competidores: el sónar y la cámara de visión.

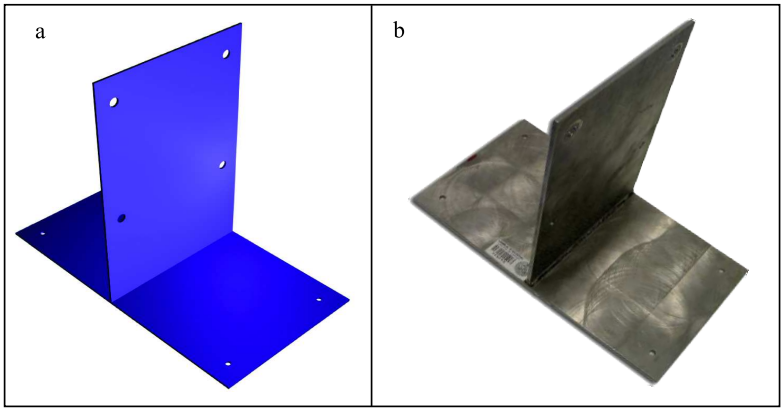


Figura 6.5: *a) Diseño 3D en Autocad. b) Pieza construida por los talleres mecánicos.*

La colocación del sensor láser en la plataforma requiere una especial atención. Por un lado se necesita que el láser se disponga de tal manera que tome siempre los datos en el plano horizontal, ya que de lo contrario cada conjunto de datos obtenido pertenecería a un plano diferente y los datos no serían consistentes. Debido a este requisito y al elevado peso del sensor (9 Kg.), para poder integrarlo en la plataforma fue necesario encargar la construcción de una pieza metálica al taller mecánico salmantino "Rectificados del Oeste". En la figura 6.5 se muestra una comparación entre el diseño 3D de la pieza realizado en AutoCAD (a) y el acabado final (b).

6.1.5. Sistema de procesamiento

Todo robot necesita algún tipo de computador a bordo para poder lograr sus objetivos. En nuestro robot, el elemento principal de procesamiento es un mini-pc que coordina los sensores y los actuadores a través del servidor de dispositivos robóticos *Player*[17]. Sus reducidas dimensiones de $10\times15\times4$ cm, y su bajo consumo (5V) permiten integrarlo a la perfección en la plataforma robótica. Dispone de una placa base ICOP-6075, un disco duro de 40 GB, 128 MB de RAM y una CPU Vortex a 166 MHz.

El mini-pc se comunica con el escáner láser de manera bidireccional mediante el puerto serie, gracias a un conjunto de *driver* e *interfaz* incluidos en el servidor *Player*. Antes de iniciar la recepción de datos, se envían una serie de comandos (según el estándar RS-232) para indicar la resolución angular, la distancia máxima de detección de objetos y la velocidad de comunicación del láser. Una vez configurado el láser, el mini-pc comienza a recibir periódicamente los datos que el láser le proporciona.

La comunicación con los motores es unidireccional. Sin embargo los mo-

tores no se pueden controlar directamente por medio del puerto serie como ocurría con el láser, si no que para acceder a ellos es necesario hacerlo a través de un *driver* de potencia. En este caso se ha elegido el módulo controlador MD22, que permite controlar de manera independiente dos motores, haciendo uso del bus *Inter-Integrated Circuit* (I^2C). Para acceder al MD22 se ha adquirido un circuito interfaz que transforma un puerto USB en un puerto I^2C. Como principal inconveniente cabe destacar que no existe un conjunto *driver-interfaz* disponible en *Player* para comunicarse directamente con los motores, y por lo tanto ha sido necesario desarrollar un *driver* de control en *Player* para el circuito MD22 por medio del bus I^2C.

6.2. *Software* de control del robot

En este apartado se estudiarán los elementos necesarios para controlar el robot móvil a través de un portátil o un ordenador de sobremesa. Uno de los elementos esenciales en el proceso de control del robot es el servidor *Player*.

6.2.1. Proyecto *Player/Stage*

El proyecto *Player/Stage* [17] fue fundado por Brian Gerkey (Stanford), Richard Vaughan (SFU) y Andrew Howard (NASA JPL) en el año 2000, basándose en *software* que habían desarrollado previamente con Kasper Stoy y otros en los Laboratorios de Investigación Robótica de la Universidad de California del Sur. La versión actual de *Player* es la 2.0.4, y la de *Stage* es la 2.0.3 (Julio de 2007). El nombre del proyecto fue tomado de una célebre frase de *W. Shakespeare*: *"All the world's a stage, And all the men and women merely players"*, que traducida al español, significa "El mundo entero es un escenario, y los hombres y mujeres son simplemente actores".

Este proyecto está compuesto por tres componentes bien diferenciados:

- *Player*, un servidor de dispositivos robóticos.
- *Stage*, un simulador multi-robot en 2D.
- *Gazebo*, un simulador multi-robot en 3D.

A continuación se comentarán las características más importantes de los componentes *Player* y *Stage* únicamente, puesto que el componente Gazebo no ha sido utilizado en el desarrollo del proyecto.

Servidor de dispositivos robóticos *Player*

Player es un servidor en red multihilo para el control de robots. Ejecutándose en un robot, *Player* proporciona una interfaz simple y clara para comunicarse con los sensores y actuadores del robot a través del protocopo TCP/IP. El programa cliente habla con *Player* mediante un *socket* TCP, leyendo datos de los sensores, escribiendo órdenes en los actuadores, y configurando dispositivos al vuelo.

Player soporta una amplia variedad de *hardware* robótico. La plataforma original de *Player* es la familia ActivMedia Pioneer 2, pero muchos otros robots y muchos sensores comunes son soportados actualmente. La arquitectura modular de *Player* hace que sea muy fácil añadir soporte para nuevo *hardware*, además de que existe una comunidad de usuarios y desarrolladores que contribuyen con nuevos *drivers*.

Player se puede ejecutar en la mayoría de las plataformas compatibles con POSIX, incluyendo sistemas empotrados. Los requisitos que necesita *Player* son:

- Entorno de desarrollo POSIX, con *threads* (`pthreads`).
- Pila de protocolos TCP/IP.
- Una versión reciente de GNU gcc, con soporte para C y C++.

Trabajos anteriores a *Player* en el área de programación de interfaces para robots se habían centrado principalmente en proporcionar un entorno de desarrollo que utilizara una determinada filosofía de control. A pesar de que esto es bastante útil, los desarrolladores de *Player* creyeron que la implementación a bajo nivel impondría restricciones innecesarias al programador, el cual debería tener la oportunidad de construir cualquier clase de sistema de control a la vez que se mantiene la abstracción y el encapsulamiento.

Por lo tanto *Player* hace una clara distinción entre la interfaz de programación y la estructura de control, optando por una interfaz de programación general, con la creencia de que los usuarios desarrollarán sus propias herramientas para construir sistemas de control. Además, muchas interfaces de programación de robots permiten al programador utilizar un único lenguaje de programación, proporcionando una biblioteca (normalmente que no es de código abierto) con la cual el usuario debe enlazar sus programas. En contraste, la abstracción de *Player* mediante *sockets* TCP permite el uso potencial de cualquier lenguaje de programación. De este modo, es mucho menos restringido que otras interfaces de programación de robots.

Por tanto, *Player* está diseñado para ser independiente del lenguaje de programación y de la plataforma. Un programa cliente puede ejecutarse en

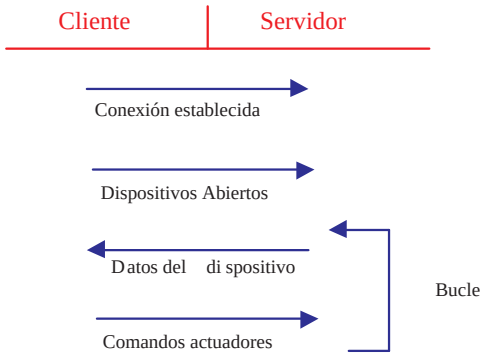


Figura 6.6: *Interacción entre cliente y servidor enPlayer*

cualquier máquina que tenga una conexión TCP con el robot en el que se ejecuta *Player*, y puede ser escrito en cualquier lenguaje que soporte *sockets* TCP. Actualmente, existen clientes en C++, Tcl, Java y Python. Además, *Player* no impone restricciones en cuanto a cómo estructurar los programas de control del robot. Si se desea que el cliente sea multi-hilo y altamente concurrente, se puede hacer así. Si se desea que sea un simple bucle lectura-actuación, también se puede hacer así.

Otra característica muy interesante es que tiene capacidad para soportar un gran número de clientes (dependiendo de las capacidades de la máquina en la que se ejecute) y permite que varios clientes accedan simultáneamente a dispositivos comunes de uno o varios robots. Esto hace que la creación de clientes multi-robot sea muy sencilla.

Por último, pero no menos importante, es que *Player* es código fuente abierto, liberado bajo la licencia pública GNU. Un ejemplo sencillo del funcionamiento de *Player* es el mostrado en la figura 6.6. El cliente establece una conexión con el servidor (*Player*) por medio de un *socket* TCP, y envía un conjunto de mensajes al servidor para abrir los dispositivos a los que desea acceder. Tras este proceso, *Player* enviará continuamente datos desde los dispositivos al cliente y éste, por su parte, realizará el control sobre el robot, enviado las órdenes apropiadas al servidor *Player*.

Simulador de robots *Stage*

Stage simula una población de robots móviles, sensores y objetos en un entorno bidimensional. Está diseñado para soportar investigación en sistemas autónomos multi-agente, de modo que proporciona modelos computacionalmente simples de gran cantidad de dispositivos en lugar de intentar emular

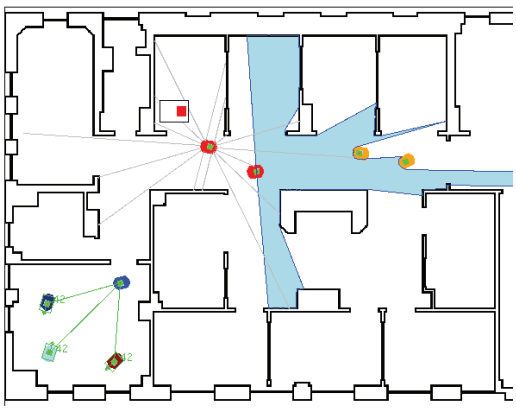


Figura 6.7: *Simulador de robots Stage*

cualquier dispositivo pero con alta fidelidad. Los desarrolladores consideran que este enfoque es de gran utilidad.

Stage proporciona poblaciones de dispositivos virtuales para *Player*. Los usuarios escriben programas clientes para el servidor *Player*. Normalmente los clientes no pueden notar la diferencia entre robots reales y sus equivalentes simulados en *Stage*. La experiencia ha demostrado que los clientes desarrollados utilizando *Stage* funcionan con ninguna o escasa modificación con los robots reales y viceversa. Por lo tanto *Stage* permite desarrollar prototipos rápidos de controladores destinados para robots reales, así como experimentar con dispositivos reales que el usuario no tiene en su posesión, o no dispone de ellos en un momento dado. Además proporciona la simulación de una amplia variedad de sensores y actuadores, tales como sónar, medidores de distancia por láser, dispositivos de detección de color por visión, odometría, etc...

En la fig. 6.7 se ofrece una imagen de la última versión de *Stage*, simulando distintos tipos de dispositivos, entre ellos, láser y sónar.

6.2.2. Integración del robot en Player

Para poder integrar un robot físico en *Player* es necesario conocer con detalle el significado de tres términos muy importantes: *dispositivos*, *interfaces* y *drivers*. El concepto de *dispositivo* se usa para referirse a una entidad abstracta que proporciona una interfaz estándar con algún servicio, de modo que es posible leer datos de dispositivos y también escribir en ellos. Un dispositivo puede ser, por ejemplo, un sónar o un láser. *Player* no implementa dispositivos de acceso individual, sino que varios clientes pueden acceder de forma concurrente a un dispositivo dado, de modo que el sistema es más flexible.

Por otra parte, *Player* establece una distinción entre la *interfaz* de un dispositivo y el *driver* del mismo. La *interfaz* de un dispositivo especifica el formato de los datos, órdenes y posibilidades de configuración que el dispositivo soporta. El *driver* de un dispositivo implementa el control a bajo nivel sobre el dispositivo. Según esto más de un *driver* puede soportar una única interfaz de dispositivo, pero no al contrario.

Todas las interacciones entre los clientes y el servidor (*Player*) se realizan a través de las *interfaces*, sin hacer referencia al *driver* que se está ejecutando por debajo. La asociación entre los dispositivos que va a controlar *Player* y el *driver* que va a utilizar con cada uno de ellos se hace en un fichero de configuración que *Player* lee al arrancar. Por ejemplo, podríamos configurar *Player* para usarse con un dispositivo `position`, al que asociamos el índice 0 y el *driver* `p2os_position`, y otro dispositivo del mismo tipo al que asociamos el índice 1 y el *driver* `rwi_position`. De esta forma un programa cliente podría controlar la posición de dos robots distintos mediante una única *interfaz* sin preocuparse del *driver* asociado al dispositivo de cada robot, simplemente debe decidir si accede al dispositivo con índice 0 ó 1.

En *Player* se encuentran implementadas *interfaces* para un gran número de dispositivos. Puesto que nuestro robot necesita actuar sobre motores, y leer datos de un láser, nos interesan únicamente las siguientes interfaces:

- **position:** en caso de que el robot disponga de un sistema de odometría, esta interfaz sirve para controlar la posición de robot, siendo posible obtener las coordenadas y el ángulo del robot, su velocidad angular y tangencial. Además, tenga o no sistema odometría, se emplea para indicar nuevas velocidades a los motores.

- **laser:** controla el láser SICK LMS 200, obteniendo las lecturas del mismo. Puede ser configurado para cambiar la apertura de lectura y activar o desactivar la lectura de los valores de reflexión. No puede escribirse.

Es posible implementar nuevas interfaces para otros dispositivos, o múltiples *interfaces* para un dispositivo, aunque esto último no es aconsejable ya que es preferible modificar una interfaz existente ampliando su funcionalidad que definir varias, cada una para realizar ciertas tareas sobre un mismo dispositivo. Es importante tratar de mantener una única *interfaz* para cada tipo de dispositivo, ya que esto permite a los usuarios el manejo de distintos modelos de dispositivos de un mismo modo. En este caso las interfaces `position` y `laser` se han podido utilizar sin ningún tipo de problema para controlar nuestro robot.

Con respecto a los *drivers*, es más frecuente tener problemas en el uso de un *driver* de *Player*, que en el uso de una interfaz, como sucede en este caso.

Para controlar la interfaz `laser` existe un *driver* para un láser SICK LMS 200, que se ha podido utilizar sin ningún tipo de problema para el sensor de nuestro robot, un láser SICK LMS 221. Sin embargo, se ha presentado el problema de que no existe ningún *driver* para la interfaz `position` que permita controlar motores a través del bus I^2C. Para poder controlar desde *Player* los dispositivos de un robot que no se encuentre en la lista de los implementados, es necesario crear los *drivers* para el manejo de los mismos, adaptándolos a las *interfaces* existentes. Por este motivo se ha desarrollado un *driver* de control de motores a través del bus I^2C según los requisitos de un módulo de control de motores MD-22, y se ha adaptado a la interfaz `position` de *Player*.

6.3. Control del robot

Para controlar el robot a través del servidor *Player* es necesario desarrollar un programa cliente que interactúe con el servidor. Una de las ventajas de utilizar *Player* es que el servidor se puede lanzar en un ordenador que tenga conexión a una red TCP/IP, y en otro ordenador diferente y que esté conectado a la misma red TCP/IP, se puede ejecutar un cliente. Este modo de trabajar de *Player* se ha aprovechado para colocar un mini-pc en el robot, conectado a la red mediante un punto de acceso inalámbrico. De este modo, como se muestra en la fig. 6.8, en el mini-pc se ejecuta el servidor *Player*, mientras que en un PC de sobremesa o un ordenador portátil se puede lanzar un cliente que se comunique con el servidor en el mini-pc, y todo sin necesidad de cables, gracias a las comunicaciones inalámbricas por TCP/IP.

Dentro del robot, el servidor *Player* se comunica con el sensor láser a través del puerto serie (RS-232) por medio de la interfaz `laser` y el *driver* `sick lms 220`. Por otro lado, para actuar en los motores a través del bus I^2C y el controlador MD22, se hace a través de la interfaz `position` y el *driver* propio desarrollado en este trabajo y denominado `i2c_a_md22`.

Para facilitar el manejo del robot se ha desarrollado una interfaz gráfica de usuario para Linux basada en GNOME/GTK, que integra el código de la aplicación cliente de *Player* en la que se incluye todo el código de control necesario para manejar los motores del robot y leer datos del láser. En la fig. 6.9 se muestra una captura de la aplicación. El menú "robot" permite indicar la dirección IP en la que se encuentra el servidor *Player*. El menú "laser" ofrece la opción de conectarse al láser para recibir datos de manera periódica, y mostrar los datos recibidos en el área de dibujo mediante puntos, líneas o una combinación de ambos. El menú "position" permite conectarse al dispositivo "position" del robot móvil, y habilitar de este modo la barra de

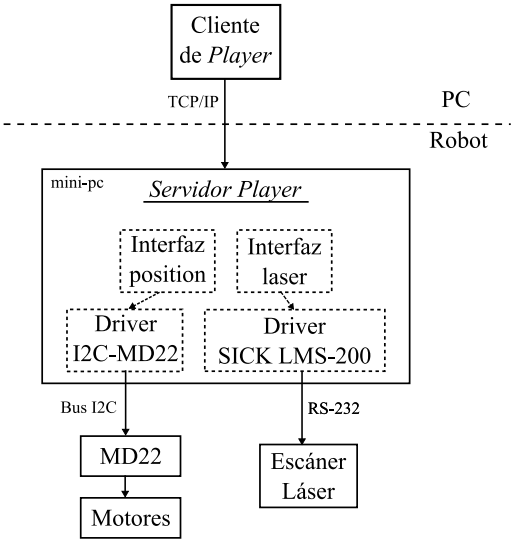


Figura 6.8: *Diagrama del sistema de comunicaciones utilizado.*

herramientas de manejo del robot que se muestra en la parte derecha de la ventana. Mediante la barra con deslizador se puede controlar la velocidad del robot. Con los botones con flechas el usuario puede controlar el sentido de la marcha y el giro del robot.

6.4. *Software* de extracción de características

La aplicación gráfica desarrollada comentada en el apartado anterior, además de contener los clientes de *Player* para controlar el robot, también incluye varios elementos relacionados con la extracción de características. Concretamente se encuentra implementada una utilidad de visualización de barridos láser almacenados en disco, así como otras relacionadas con el *clustering* y la extracción de líneas, como se detallará a continuación.

6.4.1. Visualización de ficheros

Una de las funciones más importantes del robot es la de explorar diferentes entornos con el fin de almacenar los datos que proporciona el láser. Para automatizar la toma de datos se ha incluido una utilidad que permite almacenar en disco los datos láser de manera periódica cada t segundos. Una vez almacenados es necesario volver a visualizar los datos, y ejecutar

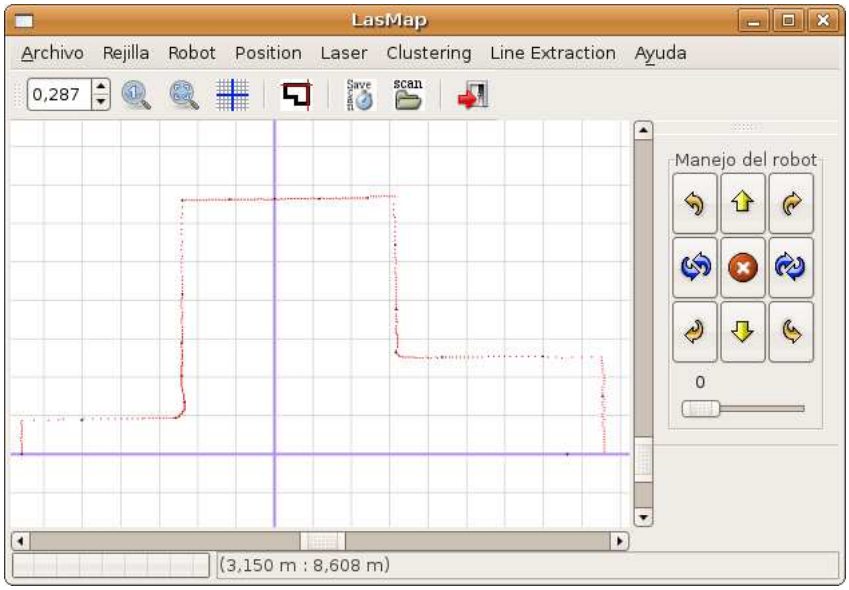


Figura 6.9: *Aspecto general de la interfaz gráfica desarrollada.*

los algoritmos programados para comprobar su rendimiento. En lugar de implementar un cuadro de diálogo estándar en el que se pueda seleccionar un nombre de fichero, se ha optado por incrustar un navegador de ficheros en la aplicación, tal y como se muestra en la fig. 6.10. De este modo, se permite cambiar el fichero visualizado en el área de dibujo mediante un sólo click, o utilizando las flechas del teclado, aumentando así la usabilidad y la velocidad para cambiar de fichero.

6.4.2. Algoritmos de *Clustering*

Los algoritmos de *clustering* adaptativo del capítulo 3, junto con el método (CCD) presentado en el capítulo 5 han sido implementados e incluidos entre las funcionalidades del *software* desarrollado. De este modo, dentro el menú "Clustering" de la aplicación se encuentran varias opciones que permiten seleccionar diferentes algoritmos de *clustering*, y comprobar el resultado de su ejecución de manera visual en el área de dibujo. Una vez ejecutado el algoritmo seleccionado, el comienzo y fin de un *cluster* se marcan mediante un cuadrado amarillo y uno azul, respectivamente, para cada uno de los *clusters* detectados, como se muestra en la fig. 6.11.

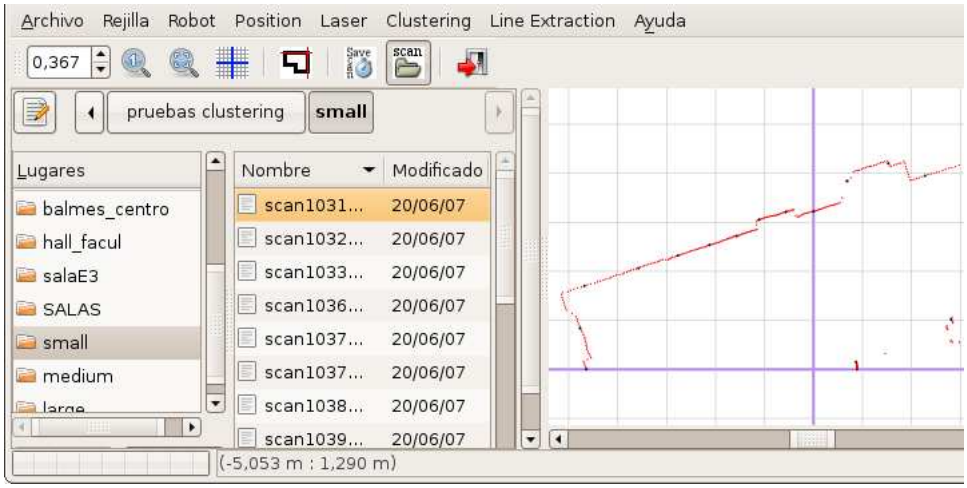


Figura 6.10: *Navegación de ficheros*

6.4.3. Algoritmos de Extracción de Líneas

Los algoritmos de extracción de líneas del capítulo 4, y el algoritmo REHOLT presentado en el capítulo quinto, se han implementado y se han integrado dentro del *software* desarrollado. Se pueden encontrar en el menú "Line Extraction" de la aplicación, donde se puede seleccionar una opción entre las presentadas, y a continuación comprobar el resultado de su ejecución de manera visual en el área de dibujo. Una vez ejecutado el algoritmo seleccionado, las líneas detectadas se muestran en azul, indicando el principio y fin de línea con un cuadrado verde y otro violeta, respectivamente. En la fig. 6.12 se muestra una captura de pantalla de la aplicación tras la ejecución de uno de los métodos de extracción de líneas implementados.

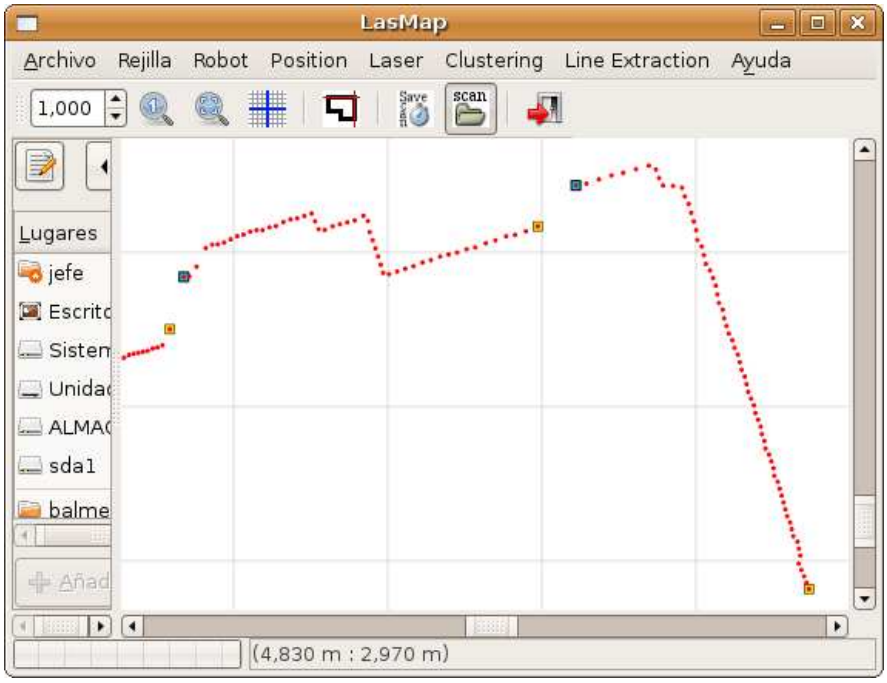


Figura 6.11: *Identificación de clusters*

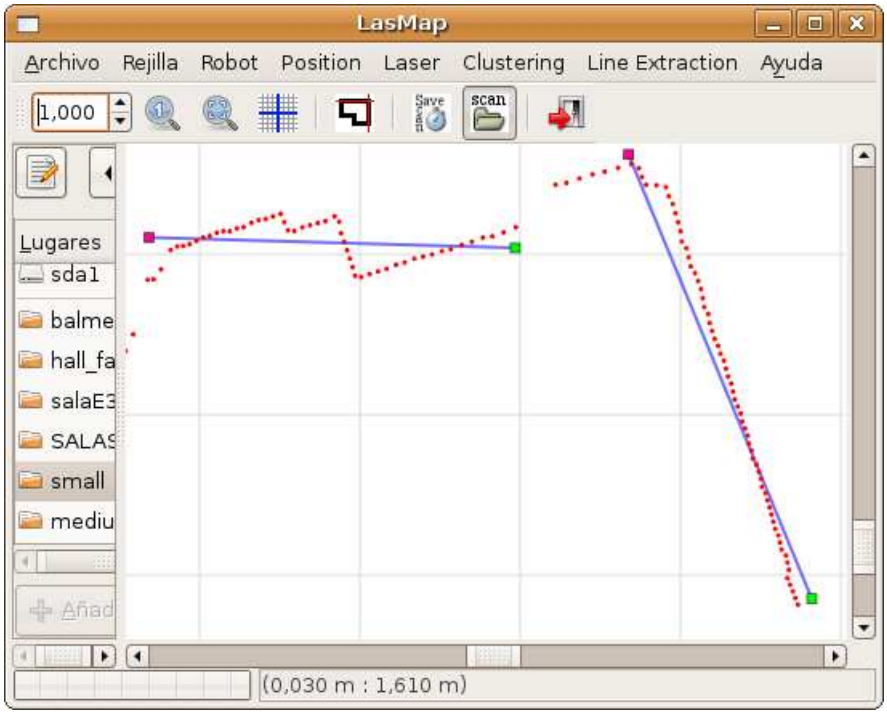


Figura 6.12: *Identificación de líneas*

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Brian W. Kernighan

7 Resultados

En este capítulo se expondrá una comparativa de los resultados prácticos obtenidos mediante la implementación de las técnicas de extracción de características descritas en los capítulos 3, 4 y 5. A continuación se comentará la plataforma de experimentación utilizada, seguido por una descripción de los escenarios de pruebas seleccionados. Después, se dedicará un apartado para analizar los resultados de los algoritmos de *clustering*, mientras que en el último apartado se estudiarán los resultados de los métodos de extracción de líneas.

7.1. Plataforma de experimentación

Para tomar los datos de los experimentos se ha utilizado la plataforma robótica analizada en el capítulo sexto, equipada con un sensor láser SICK LMS 221, y un mini-pc con una CPU a 166 MHz y 128 MB de memoria RAM. En este último está instalado el sistema operativo *Linux Ubuntu*, y sobre él se ejecuta continuamente el servidor de dispositivos robóticos *Player* ([17]), el cual permite operar sobre los actuadores y los efectores del robot.

El láser se ha configurado con una distancia máxima de detección de objetos de 80 metros. Según esta configuración, el error en cada medida tiene una desviación estándar de $10mm$, según datos del fabricante. El sensor obtiene datos efectuando barridos con una amplitud fija que abarca desde los

0^o hasta los 180^o. La resolución angular del barrido se ha establecido en $0,5^o$ y la tasa de muestreo a $10Hz$.

Durante la totalidad de la recogida de datos el robot fue teleoperado, controlando su dirección y velocidad, y automatizando el almacenamiento de barridos láser a disco cada 5 segundos. De este modo se han obtenido más de 400 barridos láser, de entre los cuales se han seleccionado manualmente aproximadamente el 10 % del total. Algunos barridos fueron escogidos al azar, mientras que otros lo fueron por ser de bastante interés para efectuar la comparativa, al presentar algunas características especialmente complejas.

Cada uno de los algoritmos ha sido programado en lenguaje C, y las pruebas de ejecución se han realizado en una estación de trabajo *Sun Ultra 20 M2*, con un procesador AMD Opteron 1218 de doble núcleo a 2,6 GHz y 2 GB de memoria RAM.

7.2. Escenarios de experimentación

Para obtener el conjunto de datos de pruebas se han utilizado dos diferentes escenarios en el interior de la Facultad de Ciencias, junto con un tercer escenario en el exterior. En adelante, para hacer referencia a estos entornos, se hará en función de una clasificación por su superficie: entorno pequeño, mediano y grande.

Uno de los escenarios, el pequeño, ha sido el hall de la planta E3 de la Facultad. En la fig. 7.1 se muestra una fotografía panorámica de dicho escenario. La fotografía abarca 180^o más o menos, igual que la amplitud de barrido del láser, y la pared central aparece muy curvada. Esto se debe a que la panorámica es una composición de varias fotos tomadas con una cámara normal de $35mm$, y al componerlas en formato panorámico, se producen grandes distorsiones cuando los objetos se encuentran cercanos al objetivo, como en este caso. No obstante, la panorámica permite hacerse una idea de lo que percibiría el láser en este entorno. El tamaño aproximado de la sala es de unos $10m \times 5m$, y existen multitud de pequeños recovecos a causa de los radiadores, marcos de las puertas y el espacio cercano a la puerta (azul) del ascensor. En esta sala, el sensor láser representó los datos de manera muy fiel, pues no se encontraron personas desplazándose durante la recogida de datos, y además en distancias cortas, el láser dispone de una mayor precisión.

Otro de los entornos ha sido el vestíbulo de entrada a la Facultad. En las pruebas se ha considerado como una sala de mediano tamaño, al tener unas dimensiones aproximadas de $22m \times 11m$. En la fig. 7.2 aparece retratado este entorno mediante una composición panorámica. En este escenario es en el que más ruido y dificultad de análisis existe, pues es un lugar muy

Figura 7.1: *Panorámica del entorno del hall de la planta E3*

Figura 7.2: *Panorámica del entorno del hall de la Facultad*

concurrido, y durante la toma de datos se encontraba gente continuamente entrando y saliendo de la facultad. Además, en la parte derecha de la fotografía, se observan varias mesas, así como bancos ocultos detrás de ellas, con la peculiaridad de que las patas de estos muebles están justo a la altura del sensor láser, produciendo numerosos *clusters* de pocos elementos.

El último escenario seleccionado se encuentra en el exterior de la Facultad, en el centro de la calle Balmes. Dicho entorno se ha considerado de tamaño grande, al tener unas dimensiones aproximadas de $70m \times 40m$. En este lugar las medidas del sensor fueron bastante limpias, aunque ocasionalmente también se detectaron personas paseando. En la fotografía panorámica correspondiente (fig. 7.3), se observa que a la altura del sensor se encuentran paredes de gran longitud, que darán lugar a posibles problemas (como el indicado en la fig. 7.4) en el proceso de *clustering*.

Figura 7.3: *Panorámica del entorno de la calle Balmes*

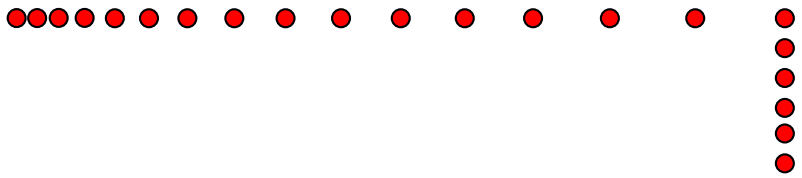


Figura 7.4: *Separación progresiva entre puntos*

7.3. Pruebas de *Clustering*

En esta sección se comparan los algoritmos de *clustering* del capítulo tercero, concretamente el criterio de Dietmayer [11], el criterio de Santos [33], y el de Borges (algoritmo de detección adaptativa de puntos de ruptura) [8]. Además se incluye en la comparativa el método original propuesto en este trabajo: *Clustering* por Convolución de Distancias (CCD).

En cada uno de los 40 barridos láser que componen el conjunto de datos de pruebas se han inspeccionado visualmente los *clusters*, anotando para cada barrido el número total de *clusters* que debían ser detectados. A continuación se han ejecutado los cuatro algoritmos y se ha apuntado el número de *clusters* detectados, y el número de aciertos. Se ha establecido como restricción principal que cada *cluster* estuviera formado por un mínimo de 5 puntos, y durante la comprobación de corrección se han contado como válidos los *clusters* detectados con una variación de $\pm$ 2 puntos o $10cm$ en su defecto.

Una de las principales dificultades que se presenta en la detección de *clusters*, es cuando la separación entre los puntos detectados va creciendo progresivamente, como se muestra en la fig. 7.4. Éste es un caso muy habitual en escenarios de gran tamaño, en el que son habituales paredes extremadamente largas, como en el caso del entorno de la calle Balmes (fig. 7.3).

Otro problema habitual durante la inspección visual es cómo decidir si algunos grupos de puntos se pueden considerar como un *cluster* o no. Por ejemplo, en la fig. 7.5a, se podría pensar erróneamente que el grupo marcado con línea punteada junto a los símbolos de interrogación es un auténtico *cluster*, puesto que da la impresión que los puntos señalados más abajo van antes en orden. Para resolver este problema, en el *software* desarrollado para el análisis visual, se ha decidido añadir al dibujo de los puntos una línea que conecte puntos consecutivos (ver fig. 7.5b). Así, los *clusters* se pueden detectar visualmente con mayor facilidad y fiabilidad.

Con respecto a los factores incluidos en la comparativa, el tiempo de ejecución no ha sido uno de ellos, debido a que todos los algoritmos han requerido un tiempo extremadamente bajo, de aproximadamente entre 50 y 100 μs. Como medida de comparación principal se ha utilizado la tasa

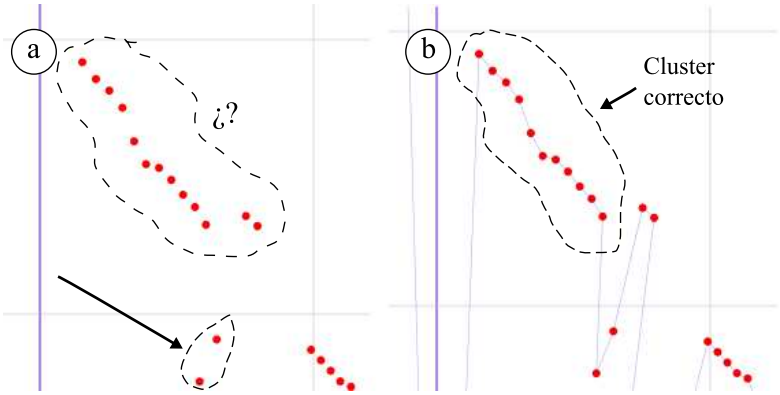


Figura 7.5: *Problema de inspección visual de clusters*

de verdaderos positivos y la tasa de falsos positivos. La tasa de verdaderos positivos se puede definir como la cantidad de *clusters* que se han detectado correctamente, y se calcula mediante la ecuación:

$$V.Pos = \frac{Num.Aciertos}{Num.Clus.Existentes}$$

La tasa de falsos positivos se puede definir como la proporción de *clusters* incorrectos que fueron marcados erróneamente como correctos, y se ha calculado mediante la siguiente ecuación:

$$F.Pos = \frac{Num.Clus.Detectados - Num.Aciertos}{Num.Clus.Detectados}$$

Lo más deseable en cualquier caso, es obtener la tasa más alta posible de verdaderos positivos, y la tasa más baja posible de falsos positivos.

En la siguiente tabla se muestran los resultados conjuntos de los 40 barridos analizados. El número total de *clusters* que se detectó por inspección visual fue de 366. Tras un laborioso trabajo experimental, los parámetros variables de los algoritmos se han configurado de la siguiente manera, al ofrecer el mejor rendimiento conjunto en los tres entornos:

- Dietmayer: $C0 = 0{,}1m$
- Santos: $C0 = 0{,}1m$; $\beta = 65^o$
- Borges: $\lambda = 10^o$
- CCD: kernel de convolución $= [-3, -3, 5, -3, -3]$

Algoritmo	**N. Clus. Detectados**	**V. Pos.** [%]	**F. Pos.** [%]
Dietmayer	395	66,6	38,2
Santos	380	72,3	29,7
Borges	336	71,6	22,0
CCD	390	83,6	21,5

En la fig. 7.6 se encuentra la gráfica que acompaña a la tabla anterior. Los resultados obtenidos mediante el criterio de *Santos* son mejores que los obtenidos con el de Dietmayer: tiene más aciertos correctos, y el porcentaje de *clusters* calificados erróneamente como buenos es menor. Con respecto a los resultados del algoritmo de *Borges*, cabe destacar como curiosidad, que el número de *clusters* detectados (336), es menor que el máximo de *clusters* correctos que podía haber identificado (366), a distinción del resto de métodos, en los que se detectaron más *clusters* de los posibles. Los resultados de Borges son en general mejores que los de Santos, puesto que a pesar de que la tasa de verdaderos positivos es ligeramente menor, sí que existe una reducción en cuanto a falsos positivos. Finalmente el método CCD, propuesto por el autor de este trabajo, no es capaz de reducir significativamente los falsos positivos con respecto a Borges, pero en cambio sí logra aumentar sustancialmente los verdaderos positivos.

Estos resultados son muy interesantes, puesto que muestran el rendimiento general de los cuatro métodos utilizando una configuración única, en tres distintos entornos de distintos tamaños. No obstante, si se estudia el rendimiento de cada algoritmo teniendo en cuenta el tamaño del entorno, algunos de los resultados varían drásticamente, como se explicará en las siguientes subsecciones. En las gráficas 7.7 y 7.8 se muestran los positivos verdaderos y falsos, clasificados según el tamaño del entorno en que se tomaron los datos: pequeño, mediano o grande.

7.3.1. Entorno pequeño

En el entorno de tamaño pequeño, el algoritmo de Dietmayer ofrece una tasa de aciertos casi del 75 %, siendo el número de aciertos muy superior a los criterios de Santos y Borges, viéndose superado únicamente por el método CCD. Esto indica que para mejorar los resultados de Borges y Santos deberían afinarse los parámetros de sus respectivos algoritmos de manera específica para este tipo de entornos. Respecto a los falsos positivos, Dietmayer (29 %) y CCD (23 %) también obtienen un rendimiento superior al resto, al ser sus tasas las menores. Los falsos positivos de los criterios de Santos y Borges son demasiado elevados, en especial este último, ya que la mitad de los *clusters* detectados fueron incorrectos.

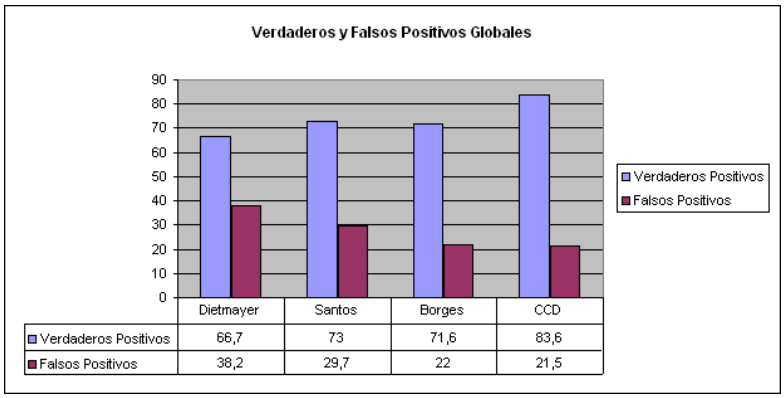


Figura 7.6: *Comparación de los cuatro algoritmos de clustering*

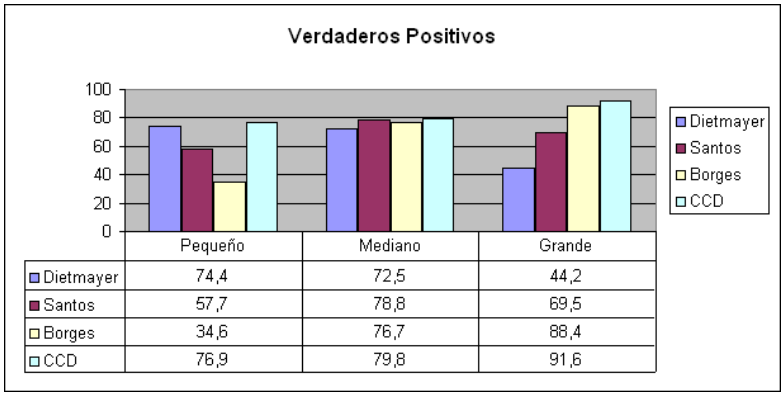


Figura 7.7: *Verdaderos positivos clasificados por entorno y algoritmo*

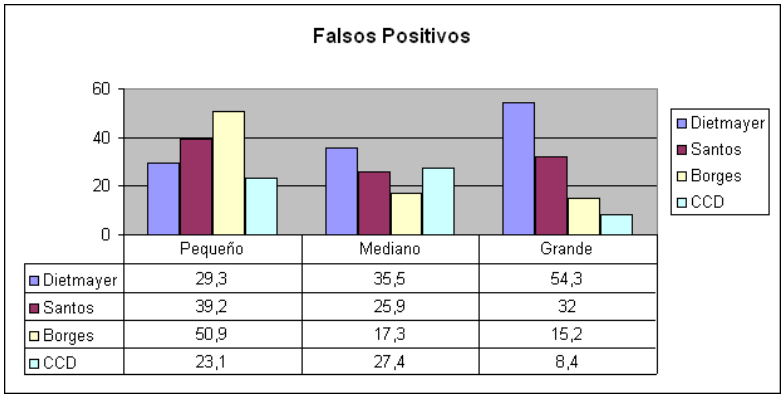


Figura 7.8: *Falsos positivos clasificados por entorno y algoritmo*

En la fig. 7.9 se muestra un ejemplo de cómo responden los cuatro algoritmos ante un determinado fragmento del entorno. El cuadrado amarillo indica el comienzo de un *cluster* y el azul el final. Se ha seleccionado tan sólo un trozo del barrido láser completo, debido al gran tamaño del barrido total. Mediante Dietmayer se detectan correctamente los *clusters*, y además prácticamente del mismo modo que el método CCD, salvo el comienzo del *cluster* inferior, en el que se detecta de diferente manera por tan sólo un punto. Se han considerado aceptables los errores en una diferencia de dos puntos. Las figuras 7.9b y 7.9c muestran cómo el criterio de Santos comete un error, mientras que el de Borges falla en dos ocasiones.

7.3.2. Entorno mediano

En el entorno de tamaño mediano, los algoritmos CCD, Santos, y Borges, obtienen un porcentaje de aciertos muy similar, entre el 76 % y el 79 %, mientras que Dietmayer consigue un porcentaje ligeramente menor (72 %). De los tres entornos, éste es en el que más ruido se reflejó en las medidas, debido al tránsito constante de alumnos y profesores. Las piernas de una persona suelen dar lugar a dos *clusters* bien diferenciados de entre 5 y 10 puntos, y cualquiera de los algoritmos detecta correctamente estos *clusters*. Debido al gran número de pequeños *clusters*, el criterio de Dietmayer ha mantenido su porcentaje de aciertos, a pesar de que detecta muy mal los *clusters* en los que las distancias entre puntos crecen progresivamente. Esto se refleja en su alta tasa de falsos positivos.

Puesto que los cuatro algoritmos presentan un porcentaje de aciertos similar, es conveniente atender a la tasa de falsos positivos para decidir cuál de los cuatro se comporta mejor. El menor valor lo consigue el criterio de Borges con un 17 %, seguido por Santos y CCD con un porcentaje ligeramente mayor y finalmente Dietmayer con un 35 %.

En la fig. 7.10 se muestra un ejemplo de cómo responden los cuatro algoritmos ante un determinado fragmento del entorno mediano. Tal y como se comentaba en el análisis anterior, el criterio de Dietmayer es el que más falsos positivos presenta. El resto de algoritmos responden mejor, aunque presentan ligeras diferencias en la detección del principio y final de los *clusters* establecidos.

7.3.3. Entorno grande

En el exterior de la facultad, en el centro de la calle Balmes, se sitúa el escenario de mayor tamaño. En este caso el criterio de Dietmayer ofrece

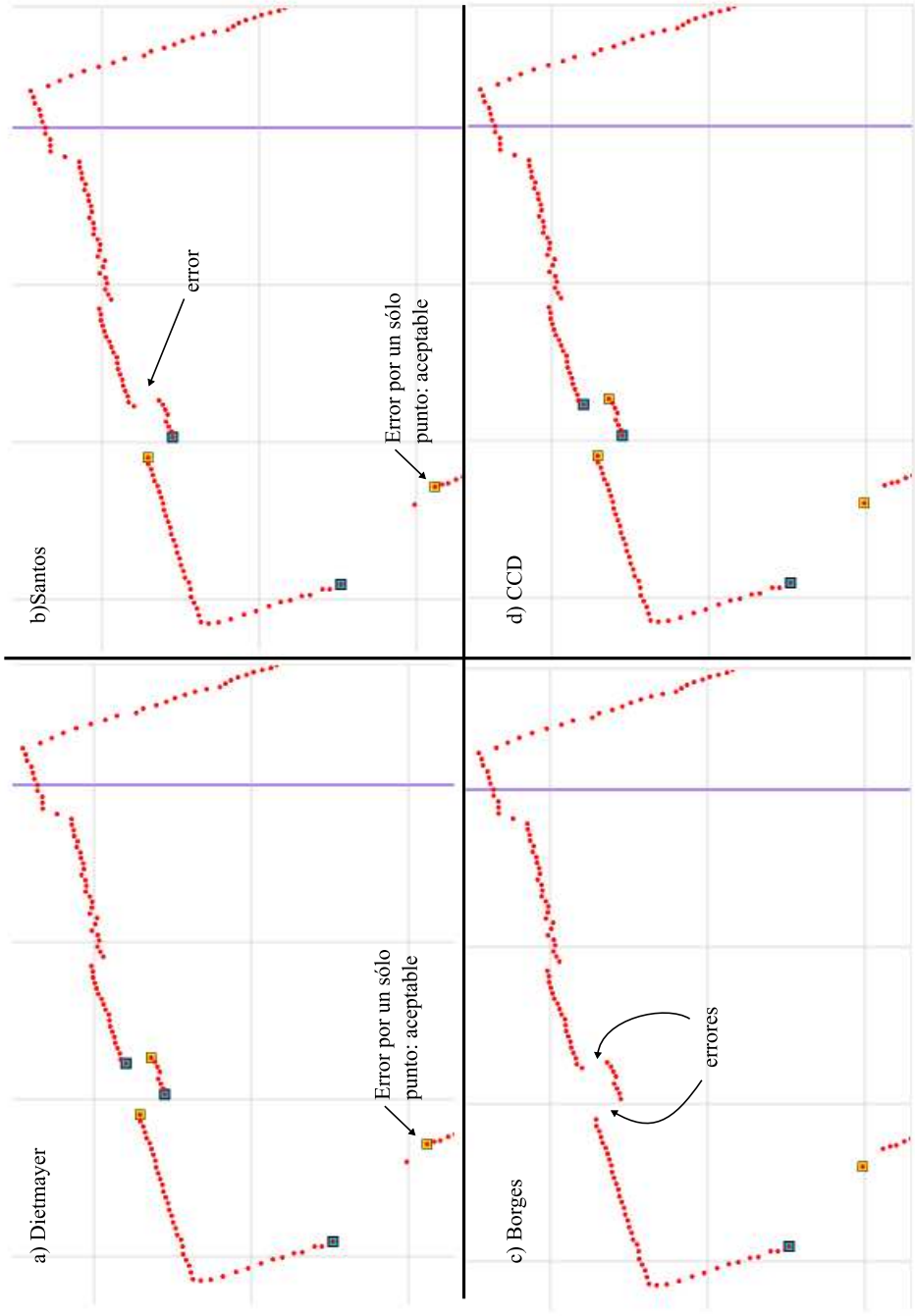


Figura 7.9: *Ejemplos de clustering en el entorno pequeño*

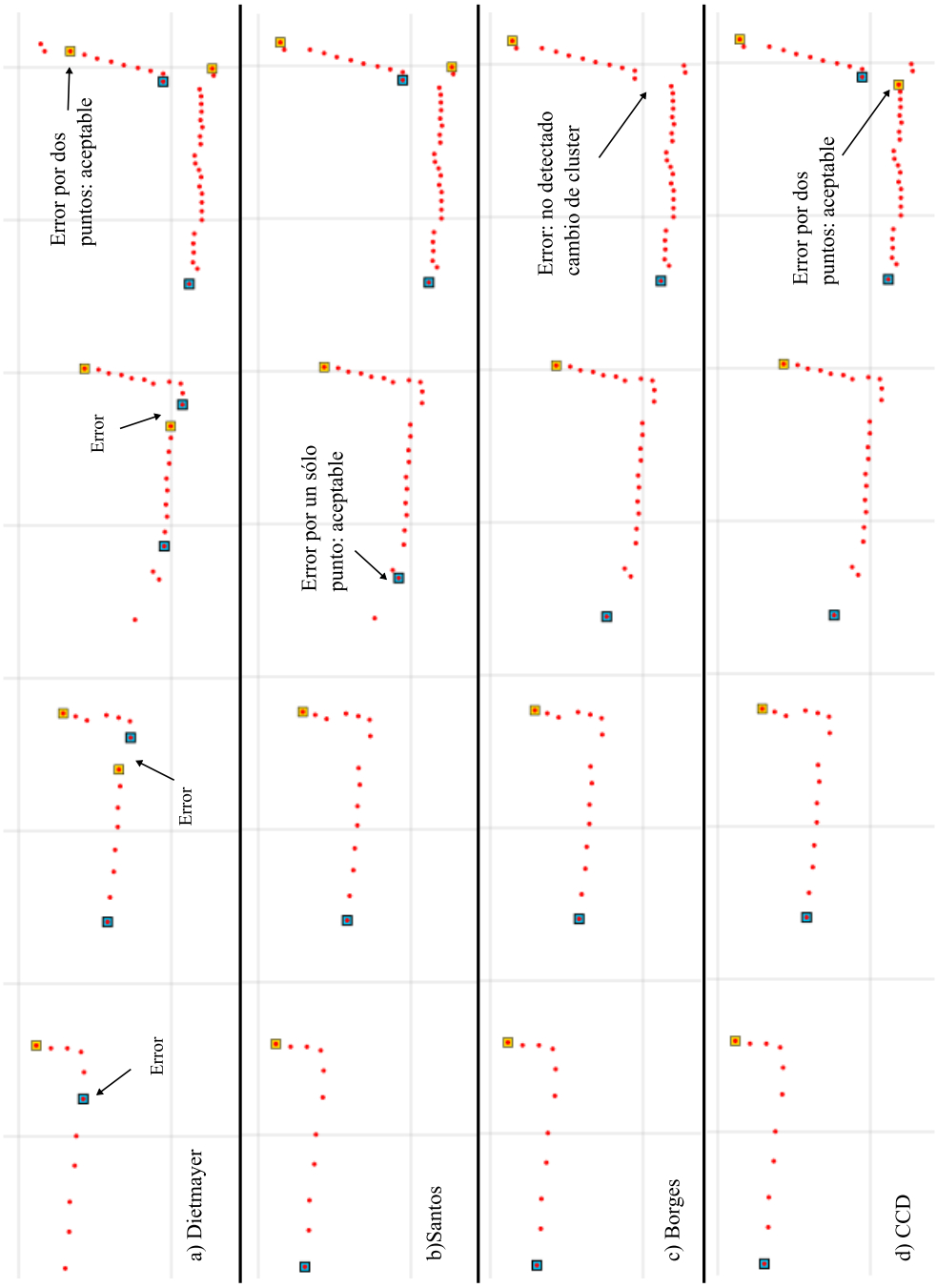


Figura 7.10: *Ejemplos de clustering en el entorno mediano*

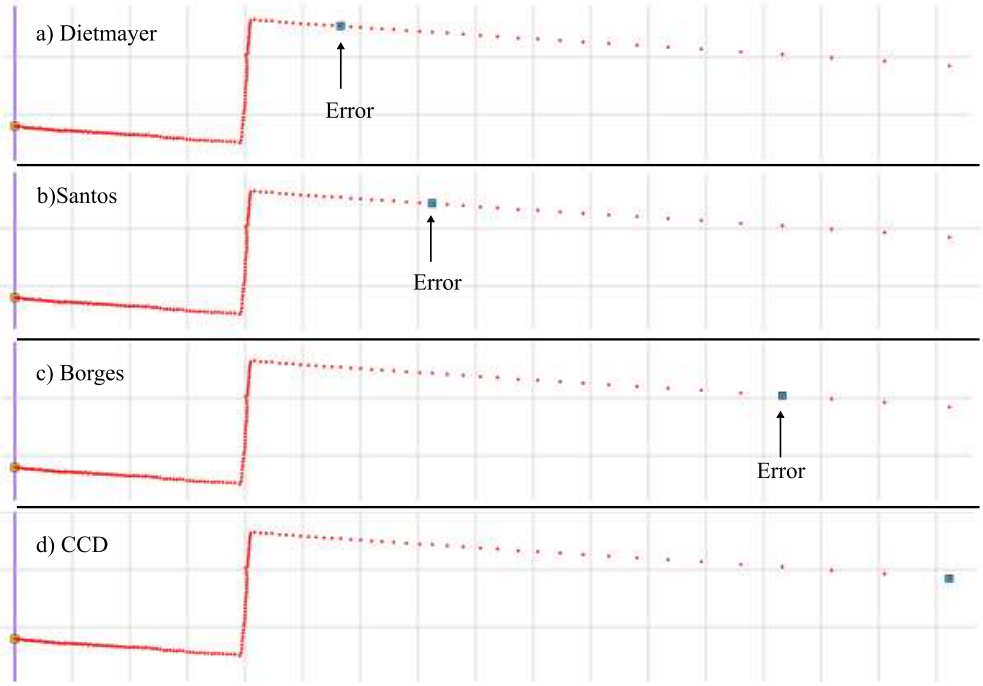


Figura 7.11: *Ejemplos de clustering en el entorno grande*

los peores resultados, con tan sólo un 44 % de *clusters* detectados correctamente y más del 50 % de *clusters* detectados erróneamente como correctos. El segundo que peor comportamiento ofrece es el criterio de Santos, con un porcentaje mayor de éxitos (65 %) pero todavía una alta proporción de falsos positivos (30 %). Finalmente ofrecen un rendimiento excepcional el criterio de Borges así como el algoritmo CCD en los cuales la detección de *clusters* correctos es del 90 % aproximadamente, mientras que los falsos positivos son muy escasos.

El problema de los falsos positivos para los criterios de Dietmayer y Santos, está en que cuando se detectan paredes de gran longitud, la separación entre los puntos crece progresivamente, y es una circunstancia muy difícil de detectar para ambos métodos. En la fig. 7.11 se puede apreciar este hecho con datos reales. Los puntos de más a la izquierda al estar muy cerca del sensor, puede que tengan una separación de 1cm o menos, mientras que, por ejemplo la separación de los puntos de más a la derecha, es de más de 1 metro. El único algoritmo que detecta correctamente este tipo de *cluster* es el CCD. Como se puede ver, el resto de algoritmos marcan el final del *cluster* antes de lo correcto.

Como resumen general se puede decir que el método original propuesto en

este trabajo (CCD), ofrece los mejores rendimientos en el entorno de tamaño pequeño y en el grande, y ocupa la segunda mejor posición en el entorno de tamaño mediano. La detección se podría mejorar ligeramente variando el kernel de convolución utilizado, de modo que se disminuyera la tasa de falsos positivos. Además es el método que mejor comportamiento ofrece en términos generales en caso de que se no se tenga en cuenta el tipo de entorno. El resto de algoritmos podrían mejorar su eficacia, pero sus parámetros tienen una fuerte dependencia con el entorno, y para ello habría que modificar sus configuraciones en cuanto se cambiara de entorno.

7.4. Pruebas de extracción de líneas

En esta sección se comparan los algoritmos de extracción de líneas del capítulo cuarto, junto con el método REHOLT propuesto en el capítulo quinto. Los algoritmos implementados han sido los siguientes:

- *Successive Edge Following* (SEF) (sección 4.3.1)
- *Line Tracking* (LT) (sección 4.3.2)
- *Iterative End Point Fit* (IEPF) (sección 4.3.3)
- *Split and Merge* (S&M) (sección 4.3.4)
- *Random Sample Consensus* (RANSAC) (sección 4.4.2)
- *Reduced Hough transform Line Tracking* (REHOLT)

En las pruebas previas a la comparativa, se observó que el algoritmo LT detectaba demasiadas líneas de manera errónea, y la mayoría correspondían a zonas donde no aparecían prácticamente puntos. Tras revisar la implementación para comprobar posibles errores, se llegó a la conclusión de que los falsos positivos detectados se debían a que el algoritmo unía el principio y el fin de *clusters* de puntos muy separados. Para subsanar el problema, se decidió aplicar una fase previa de *clustering* para que la tasa de falsos positivos de este método no fuese desproporcionada con respecto al resto de métodos implementados. El algoritmo de *clustering* seleccionado ha sido el detector adaptativo de puntos de ruptura de Borges (ver apartado 3.4.3), con un valor de 5^o para el parámetro λ. Se ha escogido el criterio de Borges, porque con respecto a Dietmayer y Santos era el único que detectaba correctamente los *clusters* en el entorno grande. Al elegir $\lambda = 5^o$ nos aseguramos disminuir la tasa de falsos positivos, al hacer que D_{umbral} (ver algoritmo

1) sea suficientemente grande. En las gráficas comparativas, este método se identificará mediante "LT+Clus.Borg".

Algo similar a lo descrito en el párrafo anterior sucedía con los métodos IEPF y S&M. A pesar de que en el conjunto de datos hubiera *clusters* muy separados, ambos algoritmos unían el comienzo y el final de los *clusters* consecutivos. Para solucionarlo se ha optado por eliminar los segmentos en los que la distancia entre puntos consecutivos fuera de más de 1 metro, lo cual ha solventado satisfactoriamente el problema en ambos casos.

Por último, cabe destacar que el algoritmo REHOLT se ha utilizado en conjunto con el método de *clustering* propuesto en este trabajo, *Clustering* por Convolución de Distancias (CCD). Por ello, en las gráficas comparativas el método en conjunto se denominará REHOLT+CCD.

7.4.1. Especificaciones de pruebas y parámetros

Como conjunto de datos de prueba se han seleccionado los mismos 40 barridos láser que se utilizaron en la comparativa de *clustering* del apartado anterior. Cada barrido se ha inspeccionado visualmente, anotando el total de líneas que debían ser detectadas. Se ha considerado que las líneas debían tener las siguientes características para poder ser detectadas:

- Mínimo número de puntos: 6
- Mínima longitud: $30cm$

Al especificar $30cm$ como mínima longitud de un segmento, se consigue eliminar los grupos de puntos correspondientes a personas, y patas de mesas o bancos. Además, un mínimo de seis puntos en $30cm$ garantiza al menos una media de 5 puntos por centímetro, lo que evita detectar segmentos con poca densidad de puntos por centímetro.

Respecto a los parámetros configurables de los algoritmos, hay un parámetro común a todos ellos: la distancia máxima de un punto a una línea para que se considere que el punto debe pertenecer a la línea. En general, en el pseudocódigo de los algoritmos del capítulo de extracción de líneas, esta distancia se denomina T_{max}. Con objeto de que las pruebas fueran lo más homogéneas posibles, se decidió definir esta distancia como $T_{max} = 10cm$ para todos los métodos.

A excepción del algoritmo REHOLT, que hace uso de la "Transformada de Hough reducida", en el resto de métodos se utiliza el método de mínimos cuadrados (secc. 4.1) para ajustar una línea a un conjunto de puntos, debido a su rapidez y a que es una opción de uso extendido en la bibliografía [27]. Teniendo todos estos aspectos en cuenta, los resultados de la comparación

probablemente resalten mejor las virtudes y defectos de cada método, al haber menos posibles diferencias debido a aspectos de la implementación.

Volviendo a los datos de prueba, tal y como se mencionó al principio de este capítulo, los datos están tomados en tres entornos diferentes. Los entornos se clasifican como pequeño, mediano y grande, atendiendo a sus dimensiones. Sin efectuar modificaciones en los parámetros de los algoritmos, se anota para cada método el número de líneas extraídas, así como el número de líneas acertadas en cada barrido láser del conjunto de prueba. En los siguientes apartados se analizará, por un lado, el rendimiento global de los algoritmos para el conjunto total de barridos láser y, por otro lado, se estudiará el rendimiento teniendo en cuenta el tamaño del entorno. Esto permitirá decidir si los métodos dependen o no de las dimensiones y características del entorno.

7.4.2. Resultados generales

De igual manera que en las pruebas de *clustering*, se han utilizado los verdaderos positivos y los falsos positivos como principal medida de comparación. En este caso, la tasa de verdaderos positivos se define como la cantidad de líneas correctamente detectadas:

$$V.Pos. = \frac{Num.Aciertos}{Num.Lin.Existentes}$$

La tasa de falsos positivos corresponde a la proporción de líneas que fueron detectadas pero no eran correctas:

$$F.Pos. = \frac{Num.Lin.Detectadas - Num.Aciertos}{Num.Lin.Detectadas}$$

Atendiendo a estos dos indicadores, el rendimiento será mejor cuanto más alta sea la tasa de verdaderos positivos, y más baja la tasa de falsos positivos.

En la siguiente tabla se muestran los datos generales para los 40 barridos láser analizados, sin separar los datos en función del tipo de entorno. Mediante inspección visual, atendiendo al número de puntos y longitud mínima de segmento, se han extraído un total de 541 segmentos. Los datos que se recogen en la tabla para cada algoritmo son: el tiempo en milisegundos, el número de líneas extraídas, el número de aciertos, la tasa de aciertos (V.Pos.) y la tasa de falsos positivos (F. Pos.).

Algoritmo	**Tiempo (ms)**	**Num. Líneas detectadas**	**V. Pos.** [%]	**F. Pos.** [%]
SEF	0,5 – 1,5	365	46,4	31,2
LT+Clus.Borg	4 – 7	550	79,1	22,2
IEPF	1 – 7	570	77,6	26,3
S&M	2 – 15	545	84,7	16,0
RANSAC	3 – 1000+	373	44,9	34,9
REHOLT+CCD	15 – 35	550	90,4	11,1

Los tiempos requeridos para la extracción de líneas son variables, dependiendo del número de características extraídas y de otros factores. En la tabla se indica aproximadamente la cantidad de tiempo invertida en los casos que menos y más se tardó en analizar un determinado barrido láser. El más rápido, con diferencia es SEF, seguido en un segundo nivel por IEPF y LT. A continuación se sitúa S&M, que alcanza un peor tiempo de 15 ms, el cual viene dado probablemente por la fase de fusión (*merge*). El método REHOLT obtiene unos tiempos bastante razonables, teniendo en cuenta que hace uso de la Transformada de Hough. Está claro que al reducir la transformada mediante una estimación inicial, el tiempo es bastante aceptable, en comparación con el tiempo que tardaría en ejecutarse la Transformada de Hough sin reducción (más de 1 segundo). El algoritmo que puede llegar a ser más lento ha sido con diferencia el RANSAC, debido a su naturaleza no determinista. Dependiendo del resultado de las combinaciones aleatorias podrá tardar más o menos tiempo, desde 3 ms, hasta más de 1 segundo. El mejor caso se da cuando las rectas al azar seleccionadas por ransac coinciden rápidamente con las rectas a detectar. El peor caso viene determinado por el número de iteraciones permitidas, en este caso 10000.

Los verdaderos y falsos positivos de la tabla anterior se muestran gráficamente en la fig. 7.12. Un dato que resulta interesante a primera vista es que, a pesar de la sencillez del algoritmo SEF, éste obtiene unos resultados ligeramente mejores que el RANSAC. Esto puede deberse a que el láser es un sistema de medida de alta precisión y esta cualidad es aprovechada por el SEF. Por otro lado, la alta tasa de falsos positivos de RANSAC se puede explicar por el hecho de que no aprovecha el orden en secuencia de los puntos del barrido láser, de modo que a menudo trata de ajustar líneas entre puntos alejados. La baja tasa de aciertos de RANSAC se explica porque se ha fijado $T_{max} = 10cm$, lo que provoca que sea difícil detectar líneas al azar cuyos puntos no se alejen de la recta más de 10cm.

Los métodos LT, IEPF y S&M, obtienen unos resultados muy superiores a los dos métodos anteriores: la tasa de aciertos es bastante alta, entorno al 80 %, mientras que el porcentaje de falsos positivos se acerca al 20 %. De

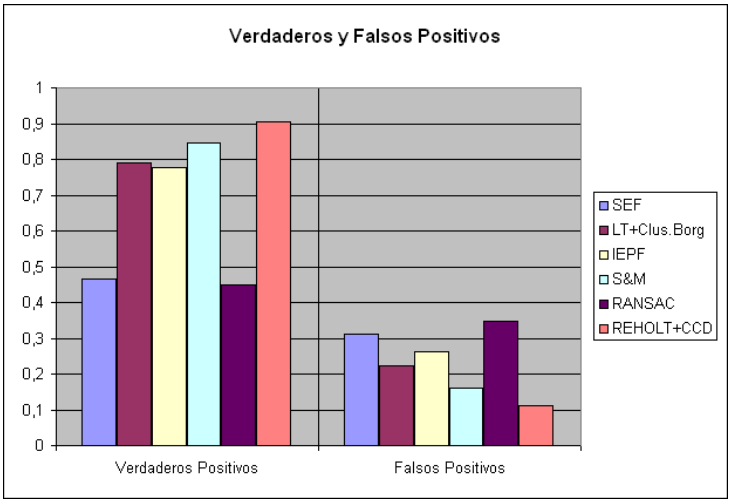


Figura 7.12: *Comparación general de los algoritmos de extracción de líneas*

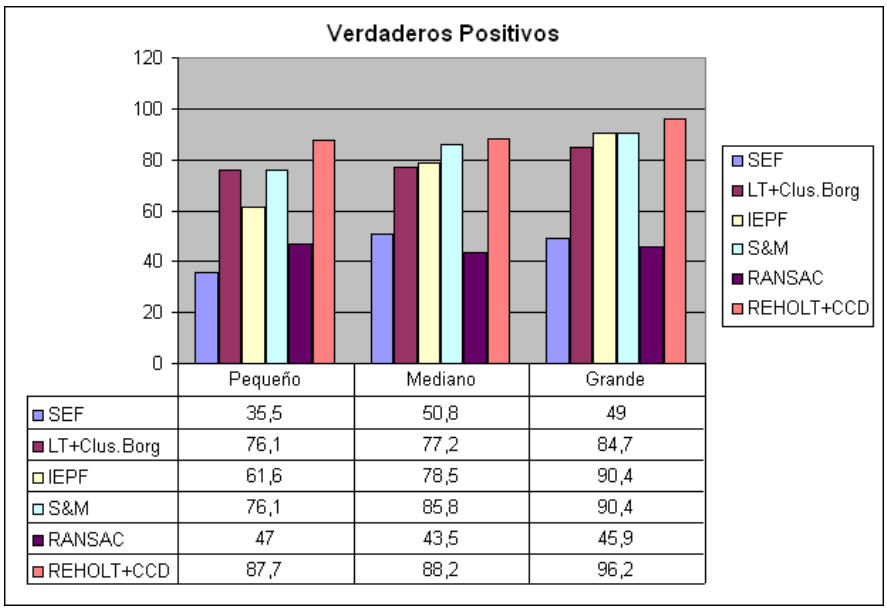
entre estos tres, destaca el S&M como el mejor.

Finalmente, el algoritmo REHOLT propuesto en este trabajo se alza con los mejores resultados, con una proporción de aciertos del 90 % y una tasa de falsos positivos de aproximadamente el 11 %. El principal inconveniente que presenta la técnica desarrollada es que es algo más lenta que el resto de las analizadas, a excepción del RANSAC, claro está.

A continuación se analizarán los resultados de nuevo, pero esta vez teniendo en cuenta el entorno en el que se tomaron los datos. De este modo se podrá comprobar si el comportamiento de los algoritmos es el mismo aproximadamente en cualquier entorno o si por el contrario la detección de rectas es mejor en un entorno específico. En las gráficas de las figuras 7.13 y 7.14 se muestra una comparativa de los verdaderos positivos y falsos positivos, clasificados según el tamaño del entorno.

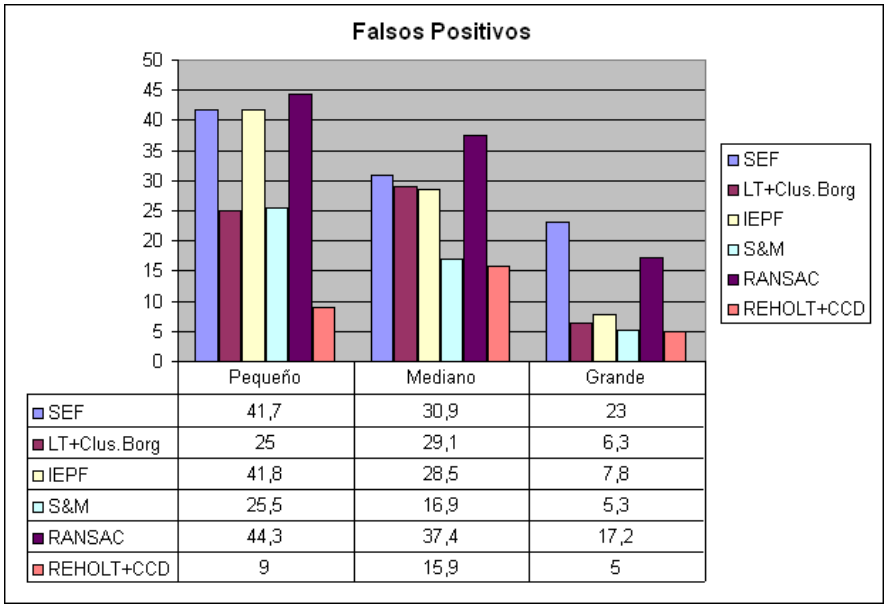
7.4.3. Entorno pequeño

En este entorno, el algoritmo SEF tiene una tasa de aciertos muy reducida (35 %), y una tasa de falsos positivos muy elevada (41 %). De los tres entornos, éste es en el que peor rendimiento ha obtenido el SEF, debido a la elevada cantidad de esquinas y rincones existentes, y a la imposibilidad de este algoritmo para detectarlas, como se expuso en la sección 4.3.1. En la fig. 7.15a, se demuestra cómo SEF traza un promedio del conjunto de puntos mediante el método de mínimos cuadrados, y no detecta ningún segmento correctamente.



	Pequeño	Mediano	Grande
SEF	35,5	50,8	49
LT+Clus.Borg	76,1	77,2	84,7
IEPF	61,6	78,5	90,4
S&M	76,1	85,8	90,4
RANSAC	47	43,5	45,9
REHOLT+CCD	87,7	88,2	96,2

Figura 7.13: *Verdaderos positivos clasificados por entorno y algoritmo*



	Pequeño	Mediano	Grande
SEF	41,7	30,9	23
LT+Clus.Borg	25	29,1	6,3
IEPF	41,8	28,5	7,8
S&M	25,5	16,9	5,3
RANSAC	44,3	37,4	17,2
REHOLT+CCD	9	15,9	5

Figura 7.14: *Falsos positivos clasificados por entorno y algoritmo*

El algoritmo RANSAC, tiene un porcentaje de aciertos del 46 %, algo mayor que el anterior método, pero su tasa de falsos positivos también es superior (44 %). En la fig. 7.15e se observa como detecta tan sólo dos de los seis segmentos detectables, debido a la gran cantidad de esquinas presentes. No obstante, debido a su aleatoriedad, ha sido capaz de detectar el segmento superior izquierdo, de gran dificultad para el resto de algoritmos.

Los métodos LT y S&M curiosamente coinciden en el porcentaje de verdaderos (76 %) y falsos (25 %) positivos. De ahí que visualmente sus resultados sean muy similares, como se puede apreciar en las figuras 7.15b y 7.15d. El IEPF también es muy similar al S&M, si bien este último tiene la ventaja de que puede fusionar (*merge*) segmentos contiguos, como ocurre en el caso de los dos segmentos de arriba a la izquierda de la fig. 7.15c, que corresponden al segmento fusionado de la fig. 7.15d.

Finalmente los mejores resultados los obtiene el algoritmo REHOLT, con una tasa de aciertos del 87 % y un 9 % de falsos positivos. Gran parte de los errores han sido debidos a fallos en el algoritmo de *clustering*. En la figura 7.15e, se demuestra la superioridad del algoritmo con respecto al resto en el análisis del fragmento de barrido láser analizado, ya que detecta correctamente los seis segmentos.

7.4.4. Entorno mediano

Como se ha comentado anteriormente, en este entorno, correspondiente al hall de la facultad, durante la toma de datos se encontraron continuamente personas moviéndose de un lado para otro. Además existen columnas, puertas abiertas, y otros elementos de pequeño tamaño que hacen que el número de segmentos a detectar sea elevado y el número de esquinas o rincones (más difíciles de detectar) sea menor. Debido a esto, y a que los segmentos cortos y bien separados del resto de elementos son fáciles de detectar para la mayoría de algoritmos, hay varios métodos que han mejorado su rendimiento significativamente con respecto al entorno pequeño:

- SEF: aumenta la tasa de aciertos del 36 % al 51 %, y disminuye la tasa de falsos positivos del 42 % al 31 %.

- IEPF: aumenta su porcentaje de aciertos de un 62 % a un 78 %, y reduce la proporción de falsos positivos del 42 % al 29 %.

- S&M: incrementa la proporción de verdaderos positivos del 76 % al 86 %, mientras que el porcentaje de falsos positivos se ve reducido del 26 % al 17 %.

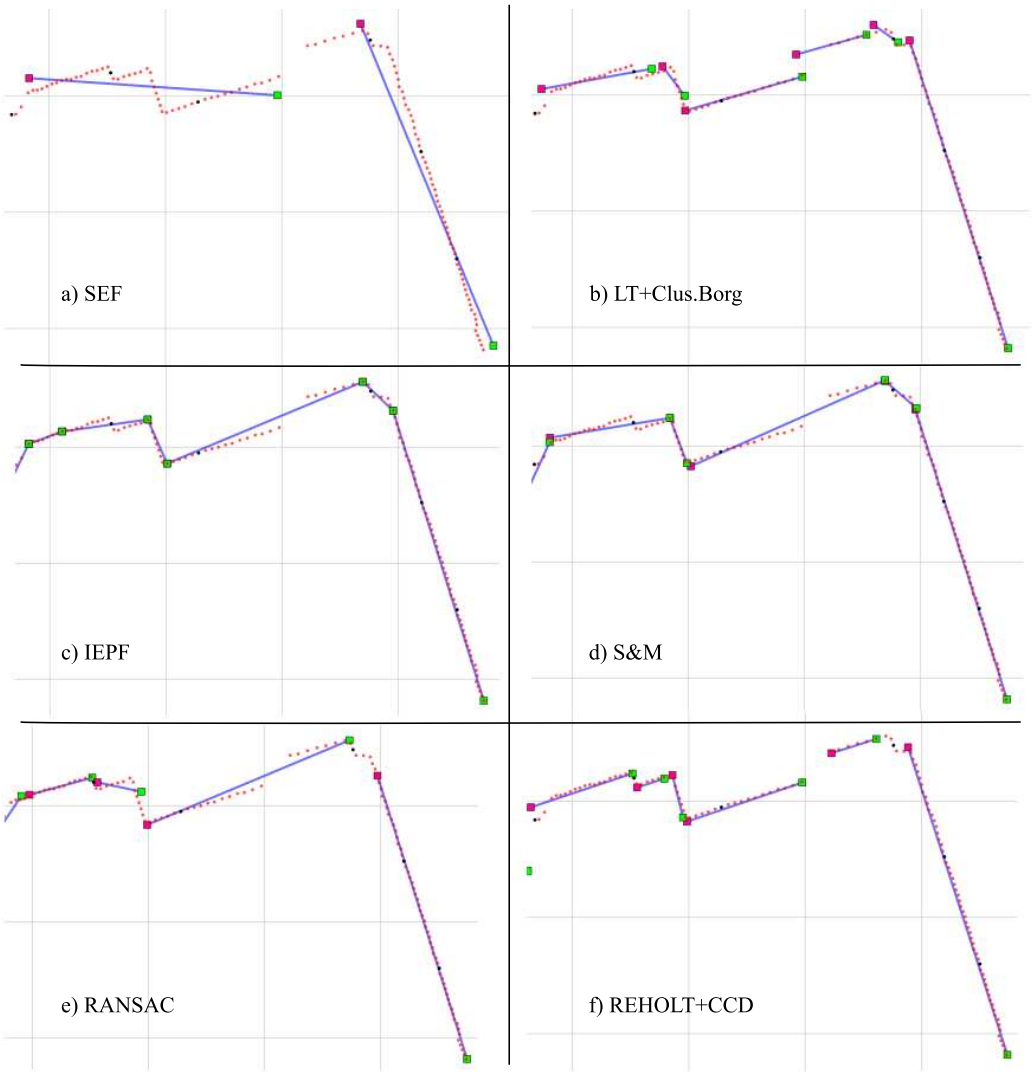


Figura 7.15: *Ejemplos de extracción de líneas en el entorno pequeño*

El resto de algoritmos (LT, RANSAC y REHOLT) mantienen la proporción de verdaderos y falsos positivos con bastante similitud respecto al entorno pequeño.

En este tipo de entorno, el algoritmo REHOLT sigue obteniendo los mejores resultados, seguido muy de cerca por SPLIT & MERGE. En un segundo plano quedarían relegados el LT y el IEPF. En la fig. 7.16 se recogen los diferentes segmentos extraídos por cada uno de los seis algoritmos en un fragmento del hall de especial dificultad. En las figuras 7.16b, 7.16c y 7.16d, se nota la necesidad de una pre-etapa de *clustering* para sendos algoritmos, de modo que se evite la unión con el *cluster* de puntos superior. Además se puede observar que las líneas que mejor se ajustan a los puntos son las extraídas por REHOLT, puesto que se utiliza la Transformada de Hough. En cambio, el resto de algoritmos utilizan el método de mínimos cuadrados, y de ahí que el ajuste de las rectas sea menos preciso. Finalmente, se puede destacar cómo RANSAC ajusta mal la recta inferior debido a que no tiene en cuenta que el láser proporciona los puntos en secuencia ordenada.

7.4.5. Entorno grande

En este entorno del exterior de la Facultad, al ser el más extenso, los segmentos a detectar no han sido complicados de identificar para la mayoría de los algoritmos, excepto para RANSAC y SEF, con menos de un 50 % de aciertos cada uno. Sin embargo en esta ocasión la tasa de falsos positivos, respecto de los entornos mediano y pequeño, es significativamente más baja también en ambos, con un 17 % y un 23 % respectivamente.

Por otro lado, el resto de métodos, presentan más de un 85 % de aciertos y menos de un 8 % de falsos positivos. El algoritmo REHOLT es el que mejor rendimiento ha obtenido con un 96 % de aciertos y solamente un 5 % de falsos positivos.

La información de las gráficas destaca que los resultados para todos los algoritmos son mejores en este entorno que en ningún otro. Especialmente el número de segmentos clasificados erróneamente como correctos es muy reducido. Esto tiene fácil explicación: las medidas han sido muy limpias, prácticamente sin ningún tipo de ruido más que el error en la medida del láser o algún paseante ocasional.

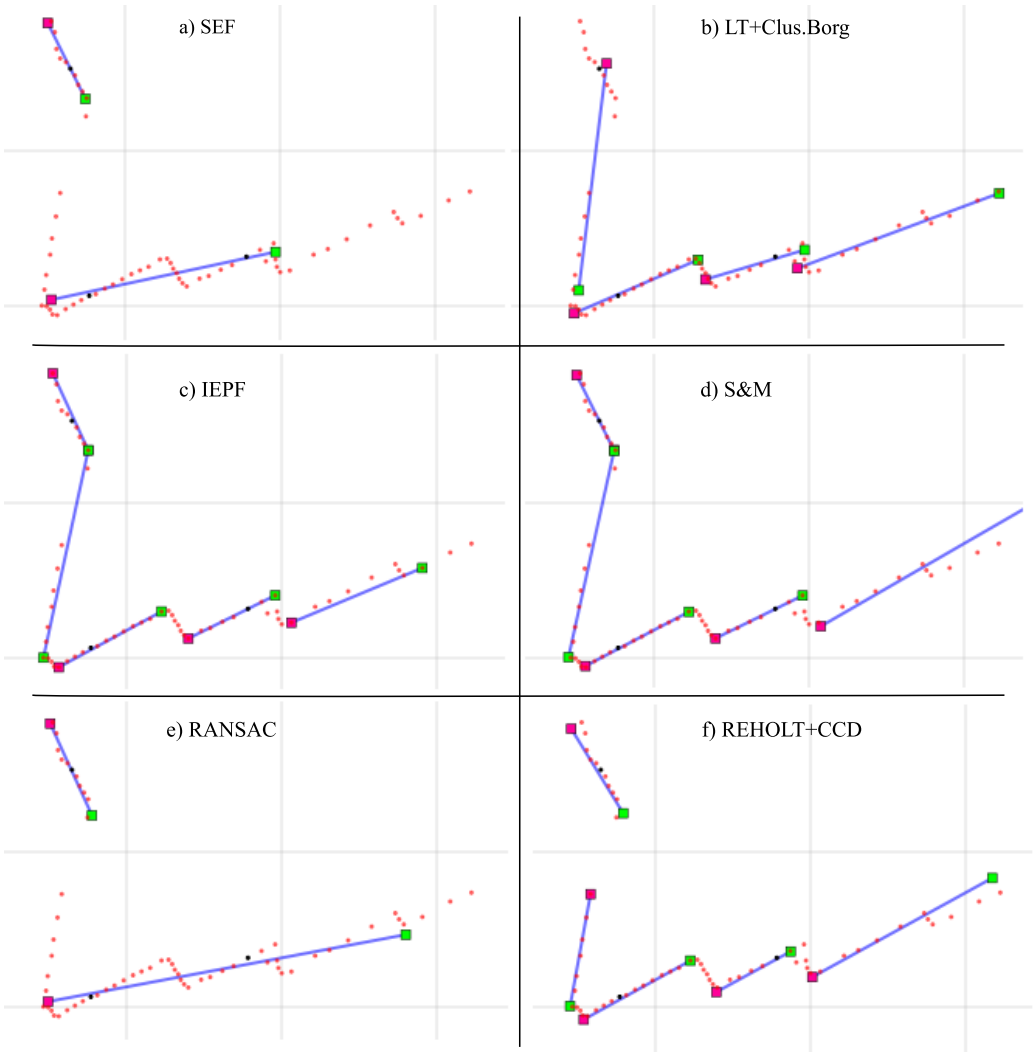


Figura 7.16: *Ejemplos de extracción de líneas en el entorno mediano*

8 Conclusiones

En este trabajo se ha presentado una perspectiva general de los sensores más utilizados para la navegación, algunos de tipo propioceptivo, como *encoders*, acelerómetros y giróscopos, y otros de tipo exteroceptivo, como el sónar, el láser, y los sensores de visión. Se ha concluido que para los propósitos de esta investigación, la mejor opción la constituía el láser, de modo que se ha centrado el foco del trabajo en las técnicas de navegación por láser.

Dentro del problema general de navegación, se encuentran otros problemas, como el de localización, la construcción de mapas o la extracción de características, y se ha identificado este último como uno de los cimientos de las técnicas de navegación. Aun así, las técnicas de extracción de características constituyen un problema muy amplio, puesto que dependen del tipo de sensor utilizado o el tipo de característica a extraer (esquinas, líneas, curvas, etc.). Teniendo esto en consideración se dirigieron los esfuerzos de esta investigación hacia las técnicas de extracción de características que estuvieran basadas en la utilización de un sensor láser, y más concretamente, en algoritmos de *clustering* y de extracción de líneas.

Una vez identificados los métodos a investigar, se ha realizado un estudio

pormenorizado de los algoritmos existentes en la bibliografía que, basándose en un láser de medición de distancias, tengan como finalidad la detección de *clusters* de puntos, así como la detección de líneas en los datos. Se ha llegado a la conclusión de que se necesitan nuevas técnicas que mejoren algunos aspectos deficientes de los métodos ya existentes.

En lo referente al *clustering* se ha propuesto un novedoso método denominado "*Clustering* por Convolución de Distancias" (CCD). Su fundamento reside en calcular las distancias entre cada par de puntos consecutivos, y a continuación aplicar una suma de convolución sobre dichas distancias, utilizando un filtro de convolución que permita seleccionar únicamente las altas frecuencias (filtro pasa alta). Se ha realizado una batería de pruebas en tres diferentes entornos de la Facultad de Ciencias, y los resultados obtenidos han sido muy satisfactorios ya que se detectan más *clusters* correctamente que con el resto de algoritmos comparados. La tasa de falsos positivos obtenidos también es, en términos generales, inferior a la conseguida mediante el resto de métodos, logrando por tanto reducir el número de *clusters* marcados erróneamente como correctos.

Con respecto a la extracción de líneas, se ha desarrollado un nuevo método (REHOLT) basado en una combinación del algoritmo *line tracking* y de la *Transformada de Hough*. Uno de los problemas de la Transformada de Hough es que requiere una cantidad desmedida (varios segundos) de tiempo de procesamiento en comparación con el resto de técnicas analizadas (varios milisegundos). Por ello se ha efectuado una modificación de la técnica, denominada "Transformada de Hough Reducida", que permite disminuir la cantidad de tiempo necesaria para el procesamiento de los datos. La reducción se logra disminuyendo las dimensiones del espacio de Hough en base a una estimación previa, logrando tiempos de procesamiento del orden de decenas de milisegundos. De este modo se obtienen los beneficios de utilizar una técnica robusta como la transformada de Hough, sin tener que sacrificar para ello el coste computacional. Utilizando el mismo conjunto de pruebas que para los algoritmos de *clustering*, se ha comparado el rendimiento de va-

rios algoritmos de extracción de líneas junto con método propuesto (REHOLT) desarrollado en este trabajo. Los resultados de la comparativa han demostrado la superioridad de REHOLT en los tres tipos de entorno, logrando la mayor cantidad de aciertos y la mínima proporción de falsos positivos. Además los segmentos detectados se han caracterizado por una gran precisión, al no estar calculados mediante el método de mínimos cuadrados, sino mediante la transformada de Hough. El mayor inconveniente de REHOLT es que su tiempo de procesamiento ha sido ligeramente superior al resto de métodos, a excepción del algoritmo RANSAC.

Con objeto de validar los métodos antecedentes y las propuestas originales de este trabajo, se ha desarrollado una plataforma robótica, así como una plataforma *software* que posibilita la interacción con el servidor *Player*, y que permite efectuar tareas tanto de control como de validación de resultados:

- El robot construido permite transportar fácilmente el sensor láser por los distintos entornos de la facultad, así como la teleoperación a distancia sin necesidad de ningún tipo de cables. Para ello se ha dotado al robot de baterías y de un punto de acceso inalámbrico.

- Para el control del robot móvil, se han desarrollado las funcionalidades *software* necesarias para controlar tanto el láser como los motores del robot. Además, se ha implementado una utilidad para almacenar datos del láser mediante un temporizador, de modo que la toma de datos se puede realizar de modo automático mientras el robot es teleoperado. También cabe destacar que se puede seleccionar entre manejar el robot móvil físico, o por el contrario un robot virtual simulado por *Stage*.

- Visualización de datos: una vez que el usuario se conecta al dispositivo láser, se inicia la comunicación en tiempo real con él y se muestra la representación gráfica de los datos recibidos sobre un área de dibujo, ofreciendo diversas opciones de visualización.

- Con respecto a la validación de resultados, se han implementado e integrado en la aplicación una serie de algoritmos de *clustering* y de

extracción de líneas. Al ejecutar cada uno de los métodos implementados sobre la base de datos de pruebas, se ofrece información visual en el área de dibujo, de modo que se puede comparar la corrección y efectividad de cada algoritmo con respecto al resto.

Finalmente, cabe mencionar que este trabajo constituirá la base de futuras investigaciones dirigidas hacia otras técnicas de navegación, como la construcción de mapas, la localización, o la planificación de trayectorias, con la finalidad de comenzar un trabajo de Tesis Doctoral.

Bibliografía

[1] AMIGONI, F.; GASPARINI, S.; GINI, M. "Building segment-based maps without pose information". En: *Proceedings of the IEEE*. July 2006. Vol. 94, pp. 1340–1359.

[2] ARRAS, K. O. *Feature-Based Robot Navigation in Known and Unknown Environments*. PhD thesis, Swiss Federal Institute of Technology Lausanne (EPFL), Thèse No. 2765, June 2003.

[3] ARRAS, K. O.; SIEGWART, R. Y. "Feature extraction and scene interpretation for map-based navigation and map building". En: *Proceedings of the Symposium on Intelligent Systems and Advanced Manufacturing*. October 1997.

[4] BARRETO, H.; MAHARRY, D. "Least median of squares and regression through the origin". *Computer Statistics & Data Analysis*. March 2006, Vol. 50, núm. 6, p. 1391–1397.

[5] BETGé-BREZETZ, S.; CHATILA, R.; DEVY, M. "Object-based modelling and localization in natural environments". En: *IEEE 1995 International Conference on Robotics and Automation*. 1995. pp. 2920–2927.

[6] BORENSTEIN, J.; EVERETT, H. R.; FENG, L. "Where am i? sensors and methods for mobile robot positioning". Technical report, The University of Michigan, April 1996.

[7] BORGES, G. A.; ALDON, M.-J. "A split-and-merge segmentation algorithm for line extraction in 2d range images". En: *Proceedings of the 15th International Conference on Pattern recognition*. 2000. Vol. 1, pp. 441–444.

[8] BORGES, G. A.; ALDON, M.-J. "Line extraction in 2d range images for mobile robotics". *Journal of Intelligent and Robotic Systems*. July 2004, Vol. 40, núm. 3, p. 267–297.

[9] BRUNSKILL, E.; ROY, N. "Slam using incremental probabilistic pca and dimensionality reduction". *Proceedings of the 2005 IEEE Conference on Robotics and Automation*. April 2005, pp. 342–347.

[10] DELLAERT, F. et al. "Monte carlo localization for mobile robots". En: *IEEE International Conference on Robotics and Automation (ICRA99)*. May 1999.

[11] DIETMAYER, K. C. J.; SPARBERT, J.; STRELLER, D. "Model based object classification and object tracking in traffic scenes from range images". En: *Proceedings of the IV IEEE Intelligent Vehicles Symposium*. May 2001. pp. 25–30.

[12] DUDA, R. O.; HART, P. E. "Use of the hough transformation to detect lines and curves in pictures". *Communications of the ACM*. January 1972, Vol. 15, núm. 1, p. 11–15.

[13] DUDA, R. O.; HART, P. E. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[14] EINSELE, T. "Real-time self-localization in unknown indoor environments using a panorama laser range finder". En: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. September 1997. Vol. 2, pp. 697–702.

[15] FISCHLER, M. A.; BOLLES, R. C. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". *Communications of the ACM*. June 1981, Vol. 24, núm. 6, p. 381–395.

[16] FOX, D. et al. "A hybrid collision avoidance method for mobile robots". En: *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*. May 1998. pp. 1238–1243.

[17] GERKEY, B.; VAUGHAN, R. T.; HOWARD, A. "The player/stage project: Tools for multi-robot and distributed sensor systems". En: *Proceedings of the 11th International Conference on Advanced Robotics*. 2003. pp. 317–323.

[18] HARPER, N.; MCKERROW, P. "Recognising plants with ultrasonic sensing for mobile robotnavigation". En: *Third European Workshop on Advanced Mobile Robots*. 1999. pp. 105–112.

[19] J. COX, I. "Blanche - an experiment in guidance and navigation of an autonomous robot vehicle". *IEEE Transactions on robotics and Automation.* April 1991, Vol. 7, núm. 2, p. 193–204.

[20] JENSFELT, P. "Localization using laser scanning and minimalistic environmental models". Licentiate thesis, Automatic Control, Royal Institute of Technology, SE-100 44 Stockholm, Sweden, Abril 1999.

[21] KEAT, C. T. M.; PRADALIER, C.; LAUGIER, C. "Vehicle detection and car park mapping using laser scanner". En: *Proceedings of the International Conference of Intelligent Robots and Systems.* August 2005. pp. 2054–2060.

[22] L. CROWLEY, J. "World modeling and position estimation for a mobile robot usingultrasonic ranging". En: *Proceedings of the 1989 IEEE International Conference on Robotics and Automation.* May 1989. Vol. 2, pp. 674–680.

[23] LU, F.; MILIOS, E. "Robot pose estimation in unknown environments by matching 2d range scans". *Journal of Intelligent and Robotic Systems.* March 1997, Vol. 18, núm. 3, p. 249–275.

[24] MARTINEZ-CANTIN, R. et al. "Adaptive scale robust segmentation for 2d laser scanner". En: *International Conference on Intelligent Robots and Systems.* IEEE Press, October 2006. pp. 796–801.

[25] MASTROGIOVANNI, F.; SGORBISSA, A.; ZACCARIA, R. "On the tips of one's toes: Self-lozalization in a dynamic environment". En: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation.* June 2005. pp. 341–346.

[26] MURPHY, R. *Introduction to AI Robotics.* MIT Press, 2000.

[27] NGUYEN, V. et al. "A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics". En: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems.* August 2005. pp. 1929–1934.

[28] OLSON, C. F. "An approximation algorithm for least median of squares regression". *Information Processing Letters.* September 1997, Vol. 63, núm. 5, p. 237–241.

[29] PAVLIDIS, T.; HOROWITZ, S. L. "Segmentation of plane curves". *IEEE Transactions on Computers.* August 1974, Vol. C-23, núm. 8, p. 860–870.

[30] PREMEBIDA, C.; NUNES, U. "Segmentation and geometric primitives extraction from 2d laser range data for mobile robot applications". En: *Proceedings of Robotica 2005 - Scientific Meeting of the 5th Robotics National Festival*. April 2005.

[31] ROUSSEEUW, P. J.; LEROY, A. M. *Robust Regression adn Outlier Detection*. John Wiley & Sons, 1987.

[32] SACK, D.; BURGARD, W. "A comparison of methods for line extraction from range data". En: *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*. July 2004.

[33] SANTOS, S. et al. "Tracking of multi-obstacles with laser range data for autonomous vehicles". En: *Proceedings of ROBOTICA 2003 - Scientific Meeting of the 3rd Robotics National Festival*. May 2003. pp. 59–65.

[34] SIADAT, A. et al. "An optimized segmentation method for a 2d laser-scanner applied to mobile robot navigation". En: *3rd IFAC Symposium on Intelligent Components and Instruments for Control Applications*. June 1997. pp. 153–158.

[35] SIEGWART, R.; NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[36] SMITH, S. W. *The Scientist & Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[37] TAYLOR, R. M.; PROBERT, P. J. "Range finding and feature extraction by segmentation of images formobile robot navigation". En: *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*. April 1996. Vol. 1, pp. 95–100.

[38] VANDORPE, J.; VAN BRUSSEL, H.; XU, H. "Exact dynamic map building for a mobile robot using geometrical primitives produced by a 2d range finder". En: *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*. April 1996. Vol. 1, pp. 901–908.

[39] ZEZHONG, X. et al. "Map building for indoor environment with laser range scanner". En: *Proceedings of the IEEE 5th International Conference on Intelligent Transportation Systems*. September 2002. pp. 136–140.