RECONSTRUCCIÓN TRIDIMENSIONAL DE AMBIENTES INTERIORES UTILIZANDO UN ROBOT MÓVIL EQUIPADO CON UN TELÉMETRO LÁSER

TESIS

Que para obtener el grado de

MAESTRÍA EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta

J. Jesús Arellano Pimentel

Leonardo Romero Muñoz

Director de Tesis

Universidad Michoacana de San Nicolás de Hidalgo

Agosto 2007

A Dios, por tener la familia que tengo.

A mis padres y hermanos, por su constante e infinito cariño, así como a mis dos pequeños sobrinos por el simple hecho de existir.

A mis maestros. Especialmente al Dr. Leonardo Romero por su valiosa asesoría.

A la Universidad Michoacana de San Nicolás de Hidalgo y al Consejo Nacional de Ciencia y Tecnología, por el apoyo otorgado para realizar estos estudios.

Resumen

Esta tesis trata el problema de la reconstrucción y modelado tridimensional de ambientes interiores, utilizando un robot móvil de plataforma abierta construido localmente, equipado con un telémetro láser 2D montado sobre un sistema de giro inclinación.

Principalmente se abordan dos problemas: (1) la localización local y construcción de mapas 2D de forma simultánea, y (2) la reconstrucción y modelado tridimensional de ambientes reales. El primero trata simultáneamente de estimar la localización del robot y construir un mapa del ambiente. El segundo es el proceso de construir un modelo tridimensional directamente del ambiente real, prácticamente sin la intervención humana.

En la solución del problema de localización local y construcción de mapas 2D simultáneos, se propone un método de localización local robusto utilizando el estimador Lorentziano y el método de optimización no lineal Newton-Levenberg-Marquardt. La localización local se realiza mediante el ajuste de dos rastreos consecutivos del telémetro láser 2D realizados en dos posiciones del robot, relativamente cercanas.

Uno de los principales problemas en la construcción de modelos 3D virtuales, cuando se toman los datos del ambiente real, es desplegar y visualizar las escenas reconstruidas del modelo en tiempo real. Para solucionar el problema se propone utilizar herramientas que interactúen directamente con el hardware especializado en la aceleración gráfica, en nuestro caso la interfaz de graficación OpenGL.

En la reconstrucción y modelado 3D del ambiente real se usa el robot móvil como sistema de adquisición de datos, utilizando el sistema de giro inclinación y el telémetro láser 2D para adquirir los datos de rango 3D. Con la solución del problema de la localización local y construcción de mapas 2D, se tiene la capacidad de realizar reconstrucciones 3D cuando el robot se mueve.

La validación y experimentación de las metodologías propuestas se realizaron satisfactoriamente sobre ambientes interiores reales de tipo estructurado utilizando nuestro robot móvil.

Abstract

This thesis deals with the problem of the three-dimensional modeling and reconstruction in indoor environments, using a mobile robot with an open platform, constructed locally, equipped with a 2D laser rangefinder mounted on a pan-tilt system.

Mainly two related problems are approached: (1) the local localization and 2D map building (SLAM), and (2) the 3D reconstruction and modeling of real environments. First the robot simultaneously tries to estimate its location and to construct a map of the environment. Then it constructs a 3D model of the real environment. Exploration of the environment is guided by a user.

In the solution of the problem of the SLAM, sets out a robust method of local localization using the Lorentzian Estimator and the Newton-Levenberg-Marquardt nonlinear optimization method. The local localization is made by means of the adjustment of two scans of the 2D laser rangefinder taken in two consecutive locations of the robot.

One of the main problems in the construction of 3D virtual models, when data are taken from the real environment, is to unfold and to visualize the reconstructed scenes of the model in real time. In order to solve this the OpenGL interface is used where it takes advantage of special hardware with graphical acceleration.

In the reconstruction and modeling of the real environment the mobile robot senses its environment using the pan-tilt system and the 2D laser rangefinder to collect 3D range data. Once the localization problem is solved, and the robot moves, is is able to make 3D reconstructions.

The ideas to solve the localization and 3D reconstruction problems were validated with experimental tests using a real mobile robot in structured environments (our laboratories).

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos

α	ángulo de barrido horizontal del telémetro láser.
β	ángulo de giro del *pan-tilt*.
$\delta\Phi$	tamaño de paso.
λ	constante para hacer ∇^2 definido positivo.
∇	matriz de primeras derivadas (Jacobiano).
∇^2	matriz de segundas derivadas (Hessiano).
ω	ángulo de inclinación del *pan-tilt*.
Φ	transformación rígida de rotación y traslación.
ρ_{mc}	estimador de mínimos cuadrados.
ρ_σ	estimador Lorentziano.
σ	parámetro para el control de la influencia sobre los errores grandes utilizando el estimador Lorentziano.
θ	orientación (rotación) de un rastreo láser.
d	distancia reportada por el telémetro láser.
$e_{x,i}$	error de distancia en x.
$e_{y,i}$	error de distancia en y.
l_j	j-ésimo rastreo láser.
l_{j+1}	siguiente rastreo láser después del j-ésimo.
l'_{j+1}	siguiente rastreo láser después de aplicar la transformación rígida.
(p_x, p_y)	posición de un rastreo láser.
t_x	traslación con respecto al eje coordenado X.
t_y	traslación con respecto al eje coordenado Y.
t_z	traslación con respecto al eje coordenado Z.
(x_i, y_i)	i-ésimo punto de un rastreo láser.
$(x_{j,i}, y_{j,i})$	i-ésimo punto del j-ésimo rastreo láser.
$(x_{j+1,i}, y_{j+1,i})$	i-ésimo punto del siguiente rastreo láser.
$(x'_{j+1,i}, y'_{j+1,i})$	i-ésimo punto del siguiente rastreo láser después de aplicar la transformación rígida.
$(x_{\psi(i)}, x_{\psi(i)})$	punto más cercano al i-ésimo punto del siguiente rastreo láser.
E	suma de los errores de distancia en x y y.
I	matriz identidad.
Mt_s	matriz de traslación en el sistema de referencia del sensor.
Mr_i	matriz de rotación en el sistema de referencia de inclinación.

Mr_g	matriz de rotación en el sistema de referencia de giro.
Mr_r	matriz de rotación en el sistema de referencia del robot.
Mt_r	matriz de traslación en el sistema de referencia del robot.
P_s	un punto en el sistema de referencia del sensor.
P_r	un punto en el sistema de referencia del robot.
V_i	i-ésimo vértice de coordenadas (x, y, z).

Lista de Publicaciones

Un resumen del capítulo 2 se presentó en el artículo:

> "*Building an Open Platform for Mobile Robotics*", Leonardo Romero Muñoz, J. Jesús Arellano Pimentel. En el *International Symposium on Robotics and Automation* (ISRA 2004), realizado del 25 al 27 de Agosto de 2004 en Querétaro.

Un resumen del capítulo 3 será presentado en el artículo:

> "*Robust Local Localization of a Mobile Robot Using a* 180∘ *2-D Laser Range Finder*", Leonardo Romero Muñoz, J. Jesús Arellano Pimentel. En el *Encuentro Internacional de Ciencias de la Computación 2005* (ENC 2005), a realizarse del 26 al 30 de Septiembre de 2005 en Puebla.

Capítulo 1

Introducción

Esta tesis aborda el problema de la reconstrucción y modelado tridimensional (3D) de ambientes interiores, utilizando un robot móvil de plataforma abierta construido localmente. Los dos problemas principales a los que nos enfrentamos en el proceso de reconstrucción y modelado tridimensional (3D) son: (1) el problema del mapeo y localización simultánea (SLAM del Inglés *Simultaneous Localization And Mapping* [Thrun02]), y (2) el problema de la reconstrucción 3D. El primero, simultáneamente trata de estimar la localización del robot y construir un mapa del ambiente. El segundo es el proceso de construir un modelo 3D virtual del ambiente, lo más preciso posible, a partir de los datos sensados y de la estimación obtenida en el proceso de localización local.

En este capítulo se presenta una introducción a la reconstrucción y modelado 3D, utilizando robots móviles, así como los problemas a los que nos enfrentamos; además de los antecedentes, el trabajo previo directamente relacionado con la presente; el objetivo de la tesis, el alcance, los retos y las contribuciones; el esquema de trabajo para lograr los objetivos; y finalmente, la organización del documento.

1.1. Motivación

En la actualidad la reconstrucción y modelado 3D de objetos, escenas y ambientes, tomados directamente del mundo real, son componentes fundamentales de las aplicaciones gráficas modernas, ya que juegan un papel muy importante en la creación de experiencias virtuales realistas. Desde aplicaciones tales como entrenamiento y simulación, hasta el entretenimiento y efectos especiales en la industria del cine, así como: (1) estudios de fun-

cionalidad en arquitectura, (2) visitas o recorridos virtuales, (3) telepresencia, (4) recorridos remotos, y (5) navegación robótica, entre otras, pueden ser altamente beneficiadas por tal modelado o reconstrucción [Corrêa02].

En el entrenamiento y simulación es de gran ayuda contar con un modelo 3D del ambiente real, más aún cuando la interacción con el ambiente en situaciones de emergencia se torna peligrosa. Por ejemplo, en un incendio sería de gran utilidad para los bomberos contar con un modelo 3D para planear cuál es la mejor forma de operar en ese sitio peligroso. En la industria de los videojuegos podrían ser un gran suplemento, especialmente si la complejidad del modelo es lo suficientemente baja para el desplegado en tiempo real [Hähnel03]. En la industria cinematográfica, en el campo de los efectos especiales, contar con modelos 3D en la computadora lo suficientemente realistas es crucial, debido a que evitan la construcción de un modelo físico del ambiente. Las visitas o recorridos virtuales tienen aplicaciones en el ámbito académico, ya que se pueden visitar museos, sitios arqueológicos, plantas de producción o maquilado, laboratorios de universidades, en fin, cualquier lugar de interés didáctico. En los ambientes de alto riesgo que pueden contener químicos o radiación nociva para el ser humano se utilizan robots móviles o fijos con capacidad de telepresencia, cuando no es posible transmitir en tiempo real toda la información visual requerida del ambiente es de gran ayuda un modelo 3D. En robótica la tendencia es construir robots completamente autónomos; para ello se requiere una representación rica en información y lo suficientemente precisa del ambiente. En estos casos los modelos o mapas 3D son ideales. La navegación en la robótica móvil tradicionalmente emplea representaciones 2D del ambiente, las cuales confinan a los robots a un plano [Thrun03] cuando realmente se mueven en un espacio 3D. Un modelo 3D puede llevar mucha más información útil que un mapa 2D utilizado en la mayoría de las aplicaciones de robótica móvil [Biber04].

Sin embargo, la reconstrucción y modelado 3D de escenas reales no es una tarea trivial y en la actualidad es un campo de investigación muy amplio. Existen diversas aproximaciones para resolver el problema. Algunas se basan en imágenes como la fotogrametría y la visión estéreo, por ejemplo en [Iocchi00]; existen otras como [Biber04, Liu01, Pervölz04, Thrun03] que combinan telémetros láser con imágenes para generar reconstrucciones 3D fotorealistas; y recientemente tecnología especializada como cámaras de rango con la capacidad de capturar color RGB y profundidad por cada píxel [3DV04], sin embargo esta tecnología aún no es muy utilizada en robótica móvil debido a su alto costo. Cualquiera que sea la aproximación, como se menciona en [Miles04], existen tres formas de realizarla:

(1) manual, (2) semiautomática y (3) automática . En el proceso manual la intervención humana es esencial en todas sus etapas. Por semiautomático nos referimos a la intervención humana para determinar el posicionamiento de los sensores como: cámaras CCD (del Inglés *Charged Coupled Device*), cámaras de rango, telémetros láser 2D y 3D, entre otros, en un marco de referencia fijo respecto del objeto o escena a reconstruir. En el proceso automático la intervención humana es prácticamente nula debido a la utilización de robots móviles o fijos con varios grados de libertad.

Este trabajo es una aproximación para reconstruir tridimensionalmente, de forma automática, ambientes interiores reales generando un modelo 3D con capacidad de navegación virtual. El proceso inicia con el muestreo del ambiente, posteriormente se estima la localización del robot (problema de la localización local) y se construye un mapa 2D, el cual es la base para realizar la reconstrucción 3D global (cuando el robot se desplaza) de escenas 3D previamente reconstruidas (con el robot fijo), para finalmente obtener un modelo 3D que posibilita la navegación virtual a través de él. Es decir, el modelo 3D, a través de un tablero de control manual (un *gamepad*) o el teclado, permite desplazarse tal y como lo realizaría el robot, de forma que posicionándose en cualquier punto del espacio, donde pudo haberse encontrado el robot con cierto giro o inclinación del sistema de giro inclinación (*pan-tilt*), se visualiza el ambiente reconstruido. Los dos principales problemas, referidos al inicio del capítulo, se abordan con más detalle a continuación.

1.1.1. El problema de la localización local y construcción de mapas en 2D

Una de las principales habilidades en robótica es el movimiento, por lo que es necesario tener una representación del ambiente para moverse adecuadamente. Los mapas son una representación interna del ambiente y sobre ellos se toman decisiones de movimiento [Uni03]. En la construcción automática de mapas un robot móvil necesita resolver el problema de la localización, conocer cual es su localización, posición y orientación con respecto a un marco de referencia. Desafortunadamente cuando un robot explora un ambiente desconocido no tiene ningún marco de referencia para ubicar su posición, con excepción de características detectadas a través de sus sensores: odómetros, sonares, telémetros láser, etc.

Cuando un robot móvil tiene un mapa, parcial o total, y una estimación inicial de su localización, el problema se conoce como *seguimiento de la posición* o *localización local*

[Dudek00]. Durante la exploración, el robot debe actualizar la estimación de su localización utilizando los datos provistos por los sensores. Esto se logra mediante el empate (*matching*) de los datos con el mapa total o parcial del ambiente [Jensfelt01]. Cuando no existe una estimación a priori de la localización del robot, el problema es referido como *localización global* [Dudek00]. Supóngase el caso en el que la energía del robot falla, al restablecerse la energía, es posible que el robot no se encuentre en la posición donde ocurrió el fallo. En este caso, cuando el robot desconoce su ubicación inicial, el reto para el robot es determinar su ubicación utilizando sus sensores, su capacidad de moverse y el mapa del ambiente [Romero01].

Un robot autónomo debe realizar la localización local y la construcción del mapa [Jensfelt01]. Este problema es como *el del huevo y la gallina*, debido a que la localización local requiere del mapa y la construcción del mapa requiere la localización local. Aquí la respuesta al problema es que ambos deben realizarse simultáneamente. El proceso de construir un mapa 2D al mismo tiempo que se realiza la localización local es conocido como SLAM(*Simultaneous Localization And Mapping*).

Resolver el problema de la localización local de un robot móvil es de particular relevancia durante la construcción del mapa. Si el robot está construyendo el mapa de forma automática necesita una estimación confiable de su localización cada vez que sensa su ambiente, así la nueva información sensada puede ser adicionada al mapa en construcción. La localización local representa un reto, debido a que los sensores de movimiento como los odómetros son imprecisos y generan errores acumulativos. Por lo tanto sensores como sonares, cámaras y telémetros láser son combinados para proporcionar información sobre la localización absoluta del robot y refinar la estimación de su localización.

El segundo problema al que nos enfrentamos es el de la reconstrucción 3D, el cual será abordado a continuación.

1.1.2. El problema de la reconstrucción 3D

El empleo de robots móviles conlleva un mayor reto en la adquisición de los datos de rango que los sistemas estacionarios (por ejemplo los utilizados en [Horstmann02, Scheibe04]). Estos sistemas estacionarios utilizan tripiés o teodolitos que permiten, de forma manual, rotar los sensores de rango montados en ellos. En un sistema de adquisición de datos (muestreo) 3D estacionario se conoce su posición con precisión de algunos milímet-

ros, mientras que cuando se utiliza un robot móvil la incertidumbre de su posición varía dependiendo de la calidad en la información de sus sensores.

Sin embargo, el muestreo del ambiente real es solo el comienzo del proceso de reconstrucción de una escena, para posteriormente poder explorar esos modelos 3D virtuales. En el proceso de creación de tales modelos varios problemas necesitan resolverse: (1) para cubrir todo el ambiente, múltiples conjuntos de datos deben ser adquiridos de diferentes puntos de vista, y necesitan ser integrados; (2) las mediciones producidas generalmente tienen ruido, haciendo necesaria la aplicación de un preprocesamiento de datos; (3) la salida de los pasos previos aún son una nube de puntos, y se requiere una reconstrucción de superficies si se desea desplegar el modelo 3D como una malla poligonal; (4) en la mayoría de los casos no es posible cubrir el ambiente en su totalidad, por problemas de oclusión y limitaciones del robot móvil, en esos casos es deseable reconstruir, siempre que sea posible, la geometría ausente a partir de los datos disponibles; (5) la enorme cantidad de datos que se adquieren durante el muestreado se convierte en un problema a la hora de crear imágenes de forma similar a como las persivimos en en mundo real del ambiente en la computadora. Este proceso de crear imágenes realistas es conocido como (*rendering*).

En la siguiente sección se abordan los antecedentes que sirven como sustento al presente trabajo.

1.2. Antecedentes

Los antecedentes previos, realizados en la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica, de la Universidad Michoacana de San Nicolás de Hidalgo, son básicamente dos: la construcción de un robot móvil de plataforma abierta, y el trabajo de tesis [Vieyra04] presentado por Adrián Núñez Vieyra titulado "Fusión de lecturas de un telémetro láser con un sistema de visión estéreo trinocular para detectar obstáculos en tres dimensiones". A continuación se menciona la particular importancia y contribución de estos trabajos a la presente tesis. El estado del arte referente a la localización local (Capítulo 3), así como a la reconstrucción 3D (Capítulo 4) se abordarán en la introducción de los capítulos correspondientes.

1.2.1. El robot móvil

El robot con el que se cuenta actualmente es el producto de una evolución de múltiples prototipos realizados por el Dr. Leonardo Romero Muñoz, desde su trabajo de tesis doctoral en [Romero01]. Se han tenido tres versiones previas a la actual, en cada nueva versión se añadieron nuevas y mejores capacidades. Dentro de las nuevas capacidades se encuentran el sistema de giro inclinación (*pan-tilt*) construido especialmente para alojar un telémetro láser 2D; ambos, *pan-tilt* y telémetro, son la base para la reconstrucción 3D de escenas. Por escena nos referimos al área cubierta por el barrido horizontal que realiza el telémetro y la operación de inclinación del *pan-tilt*.

El robot móvil es la base para sensar todo el ambiente y generar la reconstrucción 3D global. Se construyó localmente, con sensores y motores comerciales, como una plataforma abierta basada en el sistema operativo Linux para fomentar la generación de tecnología local, reducir costos e impulsar la investigación en el área de la robótica móvil. La idea fue tomada de un pequeño grupo de Japoneses y su proyecto PINO [Williams02] en el mundo de los robots humanoides. El Capítulo 2 describe completamente el nuevo robot.

1.2.2. Trabajo previo

La tesis de maestría titulada "Fusión de lecturas de un telémetro láser con un sistema de visión estéreo trinocular para detectar obstáculos en tres dimensiones", presentada por Adrián Núñez Vieyra [Vieyra04] para obtener el grado de Maestro en Ingeniería Eléctrica, forma parte de los trabajos previos que han utilizado el robot móvil actual. En dicha tesis se abordan dos problemas relacionados con la visión estéreo: el problema de la calibración y el problema de reconstrucción. En el primer problema se trata de encontrar un conjunto de parámetros que participan en la corrección de la distorsión de las imágenes adquiridas por las cámaras. Haciendo uso de estos parámetros, el segundo problema es generar información, a partir de varias imágenes, para poder llevar a cabo la reconstrucción del entorno en tres dimensiones. En ambos casos, se refiere a que un robot con capacidades de movimiento hará uso del sistema de visión estéreo en forma autónoma, con el fin de navegar en su entorno sin colisionar.

El uso del telémetro láser, el robot móvil, el método de optimización no lineal llamado Gauss-Newton-Levenberg-Marquardt (GNLM) [Press86], y la experiencia de la reconstrucción 3D con visión estéreo, son las principales aportaciones.

1.3. Objetivo de la tesis

El objetivo de la tesis es la reconstrucción tridimensional de ambientes interiores reales, de tipo estructurado, utilizando un robot móvil de plataforma abierta, equipado con un telémetro láser 2D montado sobre un sistema de giro-inclinación (*pan-tilt*); para finalmente obtener un modelo 3D con capacidad de navegación virtual a través de él.

La Figura 1.1 representa gráficamente el objetivo. El robot utiliza el sistema *pan-tilt* y el telémetro láser 2D para adquirir imágenes de rango (escenas 3D) del ambiente, el robot se desplaza, girando o avanzando, y comienza a sensar nuevamente. Con los datos adquiridos se genera un modelo virtual 3D del ambiente. El modelo permite la navegación virtual 3D en una computadora por medio de los comandos reconocidos por el robot, simulando así lo que el robot aprecia después de la ejecución de un comando (ver Capítulo 2, secciones 2.4.1 y 2.5 para una descripción de los comandos). La introducción de los comandos se realiza mediante el teclado o un tablero de control manual. De aquí en adelante dicho tablero de control manual será referido como *gamepad*.

Figura 1.1: Diagrama representativo del objetivo de la tesis

Para lograr el objetivo propuesto se aplica el Esquema de la Figura 1.2. El robot móvil provee de los datos de rango y odométricos, sensados a través del telémetro láser y el odómetro respectivamente.

La lectura del telémetro en un plano paralelo al piso, es decir cuando el *pan-tilt* tiene 0° de inclinación, y los datos de posicionamiento del odómetro, son la entrada para resolver el problema de la localización local y construcción del mapa simultáneos (SLAM). Las lecturas realizadas por el láser, durante la inclinación del *pan-tilt*, desde su inclinación inicial hasta su inclinación máxima, son la entrada para generar las escenas 3D. El conjunto

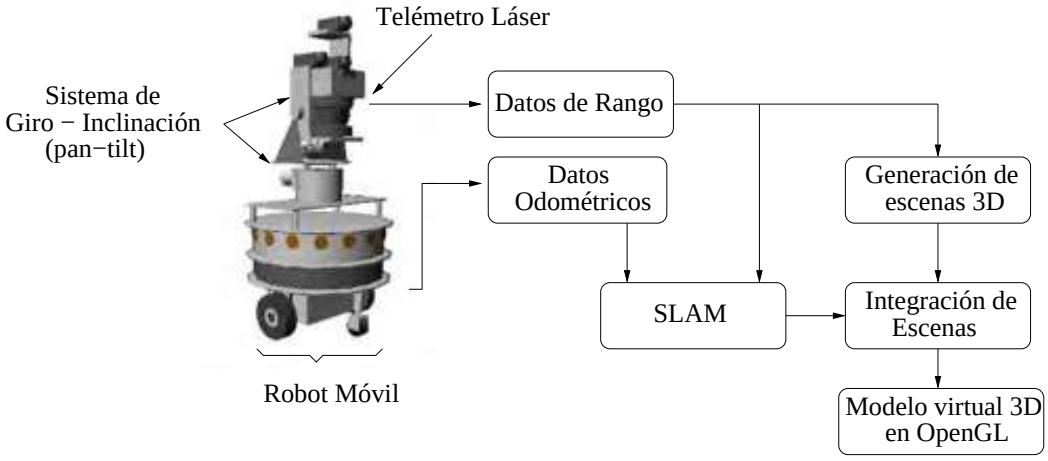


Figura 1.2: Esquema general del trabajo para lograr el objetivo

de escenas 3D son integradas en un marco de referencia global, utilizando la estimación de la localización del robot obtenida del SLAM. Finalmente se construye el modelo virtual 3D del ambiente utilizando una herramienta de graficación 3D llamada OpenGL [Richard00], la cual utiliza la aceleración gráfica 3D por Hardware para el *rendering* del modelo.

1.3.1. Alcance

Las restricciones y consideraciones en el proceso de reconstrucción 3D respecto del ambiente así como del robot son:

- **Ambientes interiores estáticos y estructurados sin marcas artificiales.** El robot móvil se mueve en un ambiente tipo oficina, con un piso y paredes planas, sin marcas colocadas intencionalmente para ayudar a la navegación del robot. Se asume que durante el muestreo, tanto para la construcción del mapa 2D y la localización local, así como para la reconstrucción de escenas 3D, el ambiente permanece sin cambio; esto es, el robot es el único objeto que se mueve.

- **Robot móvil redondo con capacidad de giro en su lugar.** La forma redonda y la capacidad de giro sobre si mismo permiten simplificar la representación del robot y del mapa, así como la planeación de movimientos. Por ejemplo, si el robot llega al

final de un pasillo angosto en forma de 'L' y desea continuar, simplemente puede dar un giro de 90 grados y avanzar. La navegación para un robot con otra forma, por ejemplo uno tipo automóvil, resulta más compleja o inclusive puede no ser factible para el caso del pasillo en forma de 'L' [Romero01].

- **Telémetro Láser 2D.** Cuando se utiliza un telémetro láser, existen diversos factores que influyen en la precisión de los datos sensados (mediciones). Factores internos que tienen que ver con las características específicas del telémetro utilizado como: ángulo máximo de rastreo horizontal, el intervalo de rastreo, y el error promedio de las mediciones. Factores externos como: la reflectividad de las superficies sensadas, condiciones ambientales de temperatura, niebla e interferencia por alguna fuente de radiación [Boehler03]. De manera particular el telémetro utilizado en este trabajo tiene un ángulo máximo de rastreo horizontal de 180°, desde $-90°$ a 90°, con intervalos de rastreo de 0,5°, por lo que se tienen 361 lecturas por rastreo del telémetro, con una precisión de $\pm 2cm$.

- **Robot móvil equipado con sistema pan-tilt.** La lectura horizontal del telémetro láser en un plano paralelo al piso provee la información de rango para efectuar el mapeo y localización local. El sistema *pan-tilt* porta al telémetro láser 2D, convirtiéndose en un sistema de adquisición de datos de rango 3D, capaz de muestrear una escena tan solo con la operación de inclinación (*tilt*); el giro (*pan*) permite la posibilidad de tener el muestreo de múltiples escenas alrededor del robot sin tener que desplazarlo. Así pues el sistema puede cumplir con dos propósitos, la navegación del robot y la reconstrucción 3D.

1.3.2. Retos

La reconstrucción y modelado 3D utilizando robots móviles involucra varios problemas, algunos de estos son:

- **Obtener una buena estimación de la localización del robot.** Se debe resolver el problema de la localización local de forma robusta, es decir que desprecien los datos atípicos (errores de medición inherentes al sensor láser), con el fin de realizar una estimación de la localización local lo más precisa posible, para permitir la integración

global de las distintas imágenes de rango en el modelo 3D final. En resumen, cuanto más precisa la estimación más precisa la integración.

- **Reconstrucción de superficies en las escenas con el robot fijo.** Como ya se mencionó, el muestreo de una escena comprende desde el rastreo de la posición mínima hasta la posición máxima de inclinación del *pan-tilt*, 100 posibles posiciones. Un rastreo contiene 361 puntos, por lo que una escena contiene 36100 puntos sensados, a partir de los cuales se construyen los vértices 3D que forman la malla poligonal.

- **Integración de las diferentes escenas en un modelo 3D.** Las superficies de las escenas muestreadas deben ser mezcladas para generar un modelo 3D global consistente. El modelo debe estar provisto de una fuente de luz para dar mayor realismo a la escena, haciendo más notorias las formas geométricas de las escenas escaneadas.

1.3.3. Contribuciones

En seguida se describen brevemente las principales contribuciones de este trabajo de tesis:

- *Esquema de mapeo y localización local robusta.* Se utiliza un método de optimización no lineal de Newton-Levenberg-Marquardt sobre una función de error basada en un estimador Lorenziano para resolver el problema del mapeo y localización local.

- *Generación del modelo 3D.* En la robótica, contar con un modelo 3D del ambiente es de gran utilidad en tareas como navegación y planeación de trayectorias del robot en un ambiente estructurado, y algunas otras más. Para el modelo 3D virtual se generó el código, en "C" y con OpenGL [Richard00], suficiente para producir el modelo 3D a partir de la localización local y los datos sensados; el modelo hace posible la navegación virtual a través de él, simulando al robot.

- *Código.* Se colaboró en la migración de código ensamblador a código en lenguaje "C" para los microcontroladores, además del desarrollo de nuevo código para el sistema *pan-tilt*. También se modificó la Interfaz del Programa de Aplicación (API por sus siglas en Inglés) del telémetro láser para sensar distancias de $32m$ con precisión de milímetros. Además se programó un *gamepad* (control manual), para facilitar la manipulación del robot en la toma de muestras.

1.4. Organización del documento

El capítulo 2 y el apéndice C describen las herramientas utilizadas en el proceso de reconstrucción y modelado 3D (robot móvil y OpenGL); los capítulos 3 y 4 la metodología empleada para lograr el objetivo (SLAM y reconstrucción 3D); el capítulo 5 los experimentos, y finalmente el capítulo 6 las conclusiones y trabajos futuros.

El capítulo 2 trata sobre el robot utilizado en los experimentos. Se describe de forma general la arquitectura y más específicamente los principales componentes como son: microcontroladores, circuitería de control, sistema *pan-tilt*, actuadores, odómetro, sonares, telémetro láser, cámaras, computadora portátil y dispositivos de interfaz como conexión inalámbrica y *gamepad*.

El capítulo 3 aborda el problema de la localización local y construcción de mapas en 2D. En la introducción del capítulo se da un breve repaso al estado del arte en el problema del SLAM. Después se define el problema, y se presentan los problemas específicos en la adquisición de datos con los sensores actuales. Se describe la metodología robusta utilizando el estimador Lorentziano, así como el mapeo de puntos y el porceso de minimización no lineal Newton-Levenberg-Marquardt.

En el capítulo 4 se trata la reconstrucción 3D. Se aborda el estado del arte de la reconstrucción 3D utilizando robots móviles. Se presentan algunos sistemas para la adquisición de datos de rango. Más específicamente, se describe la adquisición utilizando el robot y el sistema *pan-tilt*. Además se describe el proceso de transformación de los datos de rango en puntos 3D, así como la reconstrucción de superficies, integración de escenas y construcción global del modelo virtual 3D.

Los experimentos realizados son mostrados en el capítulo 5. Además de los experimentos realizados para la localización local y construcción del mapas 2D, se tienen los de la reconstrucción 3D. En uno de los experimentos de reconstrucción el robot permanece fijo y solo se utiliza la inclinación del *pan-tilt*; otro experimento consiste en mantener el robot fijo pero utilizar tanto el giro como la inclinación para el rastreo; por último, y más importante de todos, cuando el robot utiliza su capacidad de desplazarse, además del giro e inclinación del *pan-tilt* para el muestreo de los datos de rango.

Las conclusiones y sugerencias para trabajos futuros se presentan en el capítulo 6. Después se tienen las referencias y finalmente los apéndices, los cuales describen: el control del robot, la modificación al controlador (*driver*) del telémetro láser, el conjunto

de herramientas OpenGL utilizadas para crear el modelo 3D, así como los programas en OpenGL para la construcción del mismo, el ajuste de líneas rectas, y finalmente el cálculo de la localización local usando el estimador Lorentziano.

Capítulo 2

El robot móvil

Una característica importante en robótica móvil es contar con un robot que sea flexible, abierto y poderoso. Sin embargo, hoy en día en México, todos los robots móviles comerciales son importados y la mayoría de ellos no son plataformas abiertas que permitan adicionar nuevos accesorios, y además el mantenimiento es lento y caro [Romero04b]. En éste capítulo se presenta el robot móvil de plataforma abierta.

Es importante mencionar que un resumen de lo que aquí expuesto se publicó como artículo en el *International Symposium on Robotics and Automation (ISRA 2004)*, realizado del 25 al 27 de Agosto del 2004, con el título de *Building an Open Platform for Mobile Robotics* [Romero04b], en el cual el autor de esta tesis colabora como coautor del mismo.

2.1. Introducción

En la actualidad la mayoría de la investigación de robótica móvil en México utiliza robots móviles importados. Sin embargo estos robots tienen varias desventajas: (1) son caros y su costo se incrementa con el pago de impuestos de importación; (2) el mantenimiento es lento y caro, si ocurre alguna avería hay que exportarlo y reimportarlo con todo lo que ello implica; (3) normalmente son sistemas cerrados, no se tiene la posibilidad de añadir nuevas capacidades.

Para superar las desventajas, existe mucha gente interesada en utilizar y desarrollar sobre hardware y software abierto. Un ejemplo de ello es un pequeño grupo de investigadores Japoneses, que esperan emular el éxito del movimiento del código abierto que ha tenido el sistema operativo Linux, en el mundo de los robots humanoides [Williams02]. El proyecto

se llama *Open PINO Platform* y la intención es acelerar la investigación y desarrollo de robots humanoides abriendo al público toda la información técnica de PINO.

Siguiendo esta idea, es que se desarrolla y construye localmente un robot móvil flexible, poderoso y expandible. Está basado en el sistema operativo Linux con sensores y actuadores comerciales. Todo el software requerido por el robot ya forma parte de Linux, o fue desarrollado utilizando herramientas bajo GNU (*General Public License*[Stallman01]). En otras palabras no hay código propietario.

El desarrollo del robot fue realizado de manera conjunta en el transcurso del trabajo de tesis. Inicialmente se contaba físicamente con el robot utilizado, sin embargo aún faltaba migrar de la última versión, que hasta ese momento se tenía, a la versión actual del robot. Dentro de los cambios más significativos están la sustitución del microcontrolador 68hc12 por el 68hcs12 [Arts04], ambos de Motorola. En la Sección 2.3 se verán a detalle las características de este nuevo microcontrolador; otro cambio muy significativo es la adición del sistema de giro inclinación (*pan-tilt*) con su respectivo nivel de circuitería y microcontrolador; además se migró de una codificación en lenguaje ensamblador a una codificación en lenguaje "C" para el microcontrolador.

Con el nuevo microcontrolador se incrementaron las capacidades de procesamiento y almacenamiento de datos del robot. La estructura del sistema *pan-tilt* hizo posible montar nuevos sensores tal como las cámaras CCD [Trucco] que forman el sistema de visión estéreo trinocular (ver Figura 2.1), así como el telémetro láser utilizado para la adquisición de los datos de rango en la localización local y construcción del mapa 2D, y también para la reconstrucción tridimensional. A continuación se describe la arquitectura del robot y sus principales componentes.

2.2. Arquitectura del Robot

El robot tiene una configuración de locomoción diferencial [Jones98], con dos ruedas de tracción y dos ruedas de apoyo, un anillo de 16 sonares, 3 cámaras y un telémetro láser montado sobre un sistema *pan-tilt* (Fig. 2.1). En la estructura física la mayoría de las partes están hechas de aluminio para aligerar el peso, pero la estructura se construyó de acero inoxidable dando una mayor resistencia y tiempo de vida.

En el esquema de la Figura 2.2 se muestra la arquitectura del robot agrupando los principales componentes en módulos: actuadores, circuitería de control, sensores, interfaces

Figura 2.1: El robot móvil.

de los componentes, procesamiento y control, e interfaz con el exterior.

En el módulo de procesamiento y control (ubicado en la parte central de la Figura 2.2) se encuentran los microcontroladores (ver Sección 2.3), uno para controlar la base del robot y otro para controlar el sistema *pan-tilt*. En éste módulo también se tiene una computadora portátil con el sistema operativo Linux Red Hat 9.0 [Inc.04b]. La computadora portátil, además de servir como interfaz entre un operador y los microcontroladores del robot, es donde se desempeña todo el cómputo necesario para la captura, almacenamiento y envío de la información recibida o sensada por los microcontroladores, el telémetro láser o las cámaras, a través de sus dos puertos USB y de su puerto IEEE1394 (*FireWire*). La computadora tiene un procesador Pentium III @ 1 Ghz, 256 MB de RAM, y 1 slot de PCMCIA. Los *drivers* (controladores) para los puertos USB y el PCMCIA más comunes forman parte de Linux [Forge04, Kroah-Hartman04]. El software, controladores y documentación para *FireWire* en Linux se encuentra disponible en [Collins04]. En general, la información referente a los modelos comerciales de computadoras portátiles soportadas en varias distribuciones de Linux se encuentra en [onLaptops04].

El módulo de interfaz con los componentes contiene el concentrador de USB que conecta, a través de un convertidor USB-Serial, tanto a los microcontroladores como al telémetro láser, permitiendo así el monitoreo o control desde la computadora portátil. La mayoría de los convertidores USB-Serial están soportados por Linux [Kroah-Hartman04]. Dentro de este módulo también se encuentra un concentrador de *FireWire* para conectar

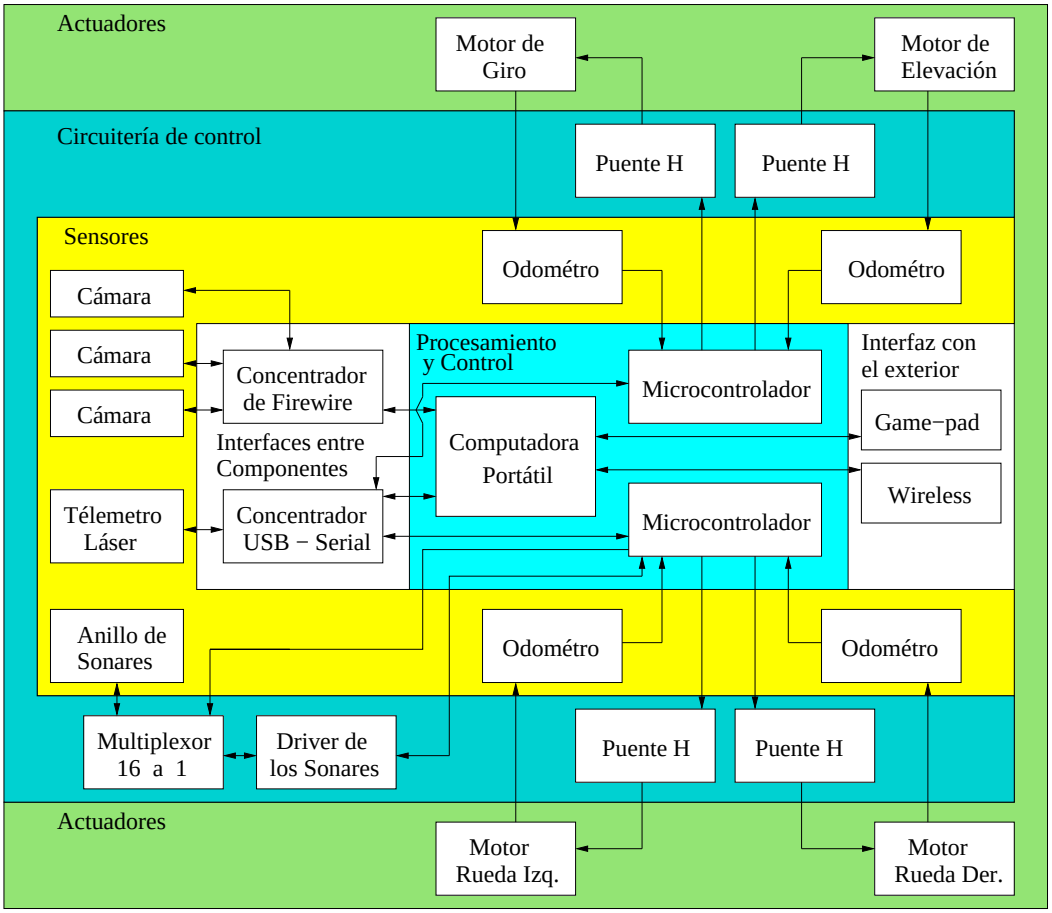


Figura 2.2: Arquitectura de los principales componentes del robot agrupados en módulos

las tres cámaras del sistema trinocular a la computadora.

En el módulo de interfaz con el exterior se provee de dos medios de comunicación e interacción con el robot. (1) Conexión de red inalámbrica por medio de una tarjeta de red PCMCIA Orinoco 802.11b a 11 Mbps alojada en su correspondiente puerto. El controlador de esta tarjeta forma parte de Linux RH9, para éste y otros controladores de tarjetas de red inalámbrica ver [Tourrilhes99]. En el Laboratorio de Sistemas Computacionales de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica de la UMSNH, se cuenta con un *Access Point* Orinoco [Proxim04] para conexiones inalámbricas de 11 Mbps y 55 Mbps con tarjetas PCMCIA 802.11a y 802.11b. (2) *Gamepad* USB *Sidewinder* de *Microsoft* conectado al puerto USB restante de la computadora. El controlador para este *gamepad* y otros *joystick* en Linux están disponibles en [Vojtech99].

Así pues el robot puede ser controlado remotamente a través de la conexión inalámbrica proporcionando, por medio de una terminal remota, los comandos aceptados por el robot (ver Sección 2.4.1 y Sección 2.5), o directamente mediante el *gamepad* conectado al puerto USB de la computadora. La pulsación de los botones del *gamepad* controlan los movimientos de rotación y translación del robot, así como el giro e inclinación del *pan-tilt*.

El módulo de circuitería de control abarca los puentes H (*H-Bridge*) [Semiconductors99] utilizados en el control de los motores (ver Sección 2.4.1), el *driver* y el multiplexor del anillo de los sonares (ver Sección 2.6.2). Los componetes más importantes como los microcontroladores, los actuadores (motores), y sensores son descritos más a detalle en las secciones siguientes.

El robot es alimentado por 4 baterías de plomo-ácido de 12 Volts, 7 Amperes-Hora. Dos de ellas suministran energía a los motores de 24 Volts, y las otras dos suministran energía para el concentrador de *FireWire* (15 V regulados), el telémetro láser (24 V), los sonares (5 V regulados) y los microcontroladores (5 V regulados).

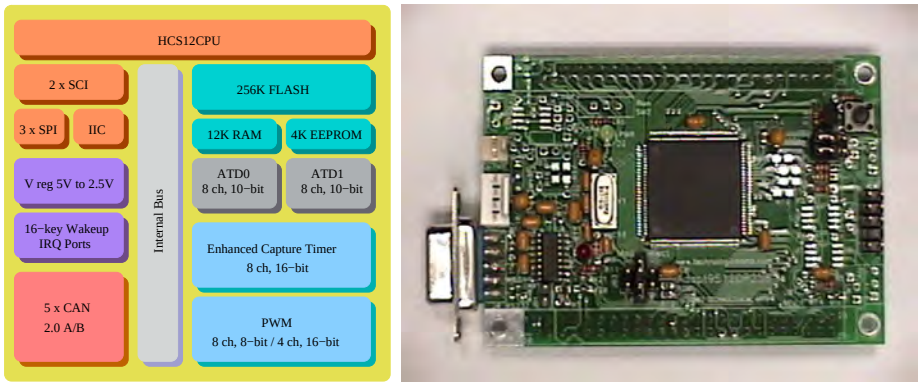
2.3. Microcontroladores

El microcontrolador es el circuito más sofisticado en el robot. Tiene numerosas ventajas en función de versatilidad, consumo de energía, tamaño y fácil uso. Más importante aún, el microcontrolador introduce una herramienta muy significativa en la solución del problema del control del robot: el software [Jones98].

La versión actual del robot utiliza dos módulos Adapt9S12DP256 (Figura 2.3 (b)) con el nuevo microcontrolador 9S12DP256C [Arts04] de Motorola. Uno controla los motores de las ruedas y el anillo de sonares, el otro controla los motores de giro e inclinación del *pan-tilt* (ver Figura 2.2).

El microcontrolador 9S12DP256 es un dispositivo de 16-bits compuesto de periféricos estándar *on-chip*. La Figura 2.3 (a) muestra el diagrama de los principales bloques del módulo Adapt9S12DP256. Una unidad central de procesamiento (HCS12CPU), 256 Kb de *Flash* EEPROM, 12 Kb de RAM, 4 Kb de EEPROM, dos interfaces de comunicación serial asíncrona (SCI), tres interfaces periféricas serie (SPI), un temporizador de captura mejorado de 8 canales, dos convertidores analógicos de 10 bits (ADC), modulación de ancho de pulso (PWM) con 8 canales, un controlador de enlace de datos digitales, 29 canales digitales discretos de I/O (Puerto A, Puerto B, Puerto K y Puerto E), 20 líneas digitales discretas de

I/O con capacidad de interrupción y activación, 5 CAN 2.0 A, B de módulos compatibles de software (MSCAN12), y un Bus Inter-IC. El microcontrolador tiene un bus de datos de 16 bits internamente, sin embargo, el bus externo puede operar a 8 bits. Además utiliza 5 V de corriente regulada y trabaja a una frecuencia máxima de reloj de 24 Mhz. Para una información más detallada del microcontrolador ver [Mot02].



(a) Diagrama de bloques (b) Módulo Adapt9S12DP256

Figura 2.3: Diagrama de los principales bloques e imagen del módulo Adapt9S12DP256 del microcontrolador 68hcs12 [Arts04].

Las capacidades que proporciona el microcontrolador son bastantes, de hecho la versión actual del robot no las utiliza todas. Sin embargo, en versiones posteriores pueden ser de gran utilidad y no se tendría la necesidad de cambiar el microcontrolador.

Programación

La construcción y habilitación de un robot generalmente involucra codificación en lenguaje ensamblador. La programación en lenguaje ensamblador consiste en escribir código de un conjunto especifico de instrucciones diseñado para un microcontrolador específico. Sin embargo para acelerar el proceso de desarrollo y abstraerse un poco de la arquitectura específica del microcontrolador se utiliza un lenguaje de alto nivel como el lenguaje C. Para nuestro robot, la codificación del control se implementó en dicho lenguaje, bajo el proyecto 68hcs12 [Carrez03] de licencia GNU [Stallman01].

2.4. Motores

Los motores son los músculos del robot [McComb00]. Adjuntar motores con ruedas posibilita al robot rodar sobre el piso; de igual forma, los motores proveen la capacidad de giro e inclinación del *pan-tilt*. Existen muchos tipos de motores, sin embargo solo unos cuantos son útiles en robótica, y dentro de estos están los motores de corriente directa (DC) [Jones98].

En el robot la corriente directa, además de ser utilizada como fuente de poder principal para la electrónica a bordo, suministra energía a los motores de la base y *pan-tilt*. En la base (tracción) del robot se utilizan dos motores Pittman DC de 24 Volts serie GM9000 con caja de engranes para reducir la velocidad e incrementar el torque, y un encoder óptico *Hewlett-Packard* (HP) con tres canales y 500 cuentas por revolución (CPR) para conocer la posición, velocidad y dirección del motor (ver Secciones 2.4.1 y 2.6.1). Específicamente el modelo de motor utilizado es el *GM9236S027 Loc-Cog DC Servo Motor*. La Figura 2.4(a) muestra la imagen del motor; el encoder óptico HP se presenta en la Figura 2.4(b). El *pan-tilt* usa un motor GM9236S027 para la realizar la operación de giro; en la operación de inclinación utiliza un motor de la misma serie pero de 12 Volts modelo *GM9234S031 Loc-Cog DC Servo Motor* (Figura 2.4(c)).

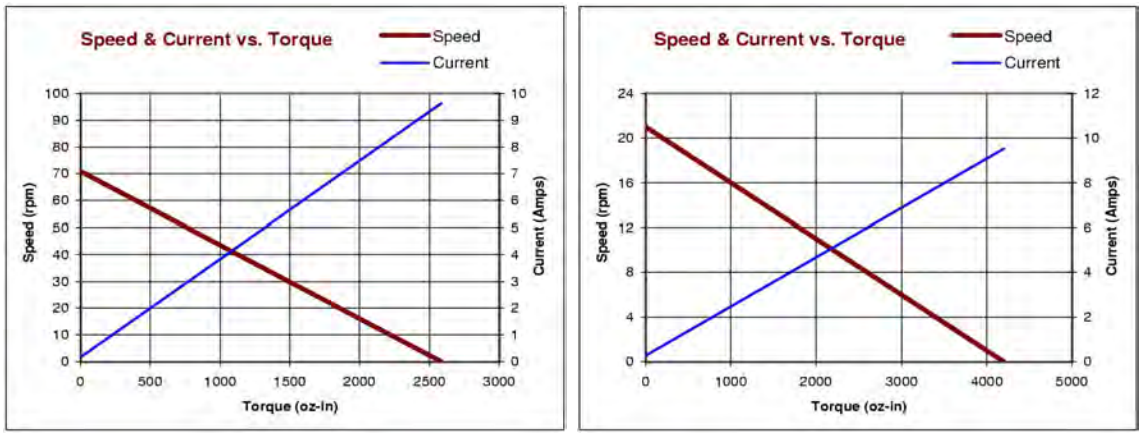
(a) Motor GM9236S027 (b) Encoder HP (c) Motor GM9234S031

Figura 2.4: Motores DC Pittman y Encoder HP utilizados en el robot.

La caja de engranes del motor GM9236S027 tiene un rango de reducción de 65.5:1, con una eficiencia del 0.80 y torque máximo permitido de 500 oz-in. En el motor GM9234S031 se tiene una reducción de 218.4:1, con una eficiencia de 0.73 y torque máximo permitido de 500 oz-in. Los engranes desempeñan dos funciones importantes: primero,

pueden hacer que la velocidad angular de un engrane crezca o decrezca con respecto de otro conectado a él; y segundo, pueden incrementar la fuerza, dependiendo de la relación entre los engranes. Los engranes también pueden simplificar la transferencia de fuerza de un lugar a otro. En la robótica móvil "*altas velocidades*" pueden ocasionar que el robot tenga derrapes y se golpee contra las paredes o personas [McComb00].

En la gráfica de la imagen de la Figura 2.5(a), tomada de [TM01b], se observa que el motor GM9236S027 tiene una velocidad máxima de 71 revoluciones por minuto (rpm), un torque pico de 2585 oz-in, y una corriente pico de 9.64 Ampers. En la gráfica de la Figura 2.5 (b), tomada de [TM01a] puede observarse que el motor GM9234S031 tiene una velocidad máxima de 21 rpm, un torque pico de 4199 oz/in, y una corriente pico de 9.52 Ampers. Como se puede observar, la reducción en la caja de engranes del motor utilizado en la operación de inclinación del *pan-tilt* es mayor respecto a los otros motores, permitiendo con ello un control más fino sobre esta operación.



(a) Motor GM9236S027 (b) Motor GM9234S031

Figura 2.5: Gráficas de velocidad y corriente contra torque de los motores [TM01b, TM01a].

2.4.1. Control de los motores

Para controlar los motores de las ruedas y del *pan-tilt* se utiliza un control proporcional integral (PI). Nuestro control PI se basa en [Jones98]. El Apéndice A contiene la codificación. El control de velocidad aplica un esquema de modulación de ancho de pulso (PWM

por sus siglas en Inglés de *Pulse Width Modulation*), el cual es generado automáticamente por el microcontrolador. También se utiliza un puente H (*H-Bridge*), modelo *LMD18200* de 3A y 55V de *National Semiconductor* [Semiconductors99], para el control de cada motor de CD.

Control proporcional integral

La idea básica de un ciclo de control de velocidad es tomar una referencia de velocidad deseada, enviar el comando a los motores, ver que tan rápido giran los motores, y entonces medir la velocidad y compararla con el comando de velocidad enviado. La diferencia es llamada señal de error y puede ser positiva o negativa [Jones98].

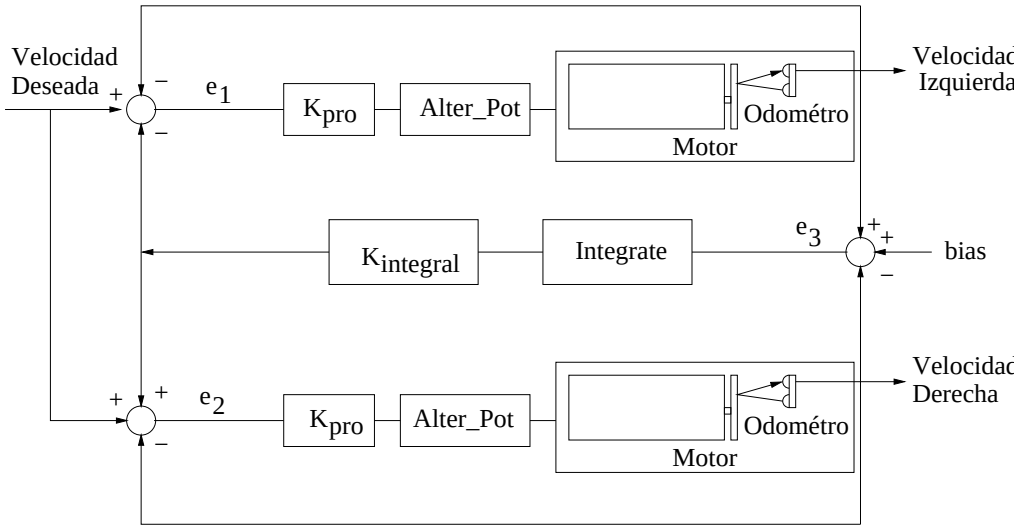


Figura 2.6: Diagrama del ciclo de control proporcional integral

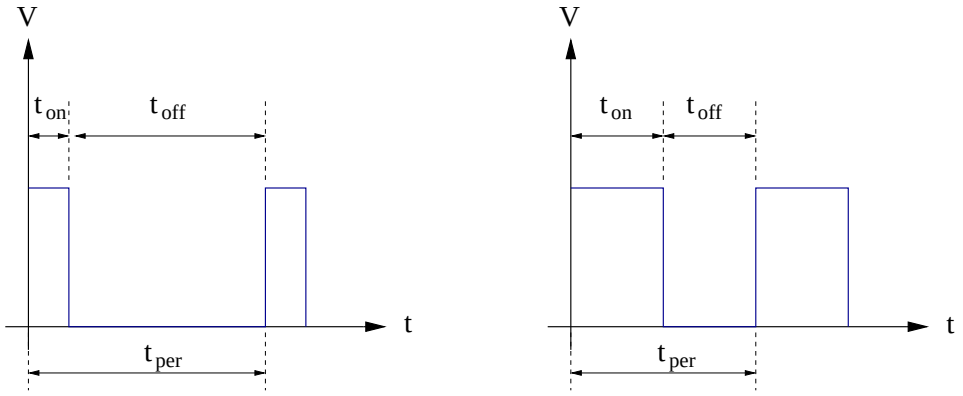
El diagrama de la Figura 2.6 muestra el ciclo de control proporcional integral utilizado. Si la señal de error (e_1, e_2) se incrementa o decrementa de forma constante, entonces se aplica un control proporcional; la velocidad actual es multiplicada por una constante (k_{pro}) y se retroalimenta al motor para ajustar su velocidad. Por otro lado, en la sincronización de los dos motores se aplica un control integral. Este compara las velocidades de ambos motores, la diferencia obtenida proporciona una señal de error (e_3). Un control integral, integra o suma la señal de error sobre el tiempo, multiplica esta suma por una constate ($k_{integral}$), y alimenta el ciclo de control, en nuestro caso, para cada motor.

Donde $e_3 = velocidadizquierda - velocidadderecha + bias$, y el término *bias* es usado para introducir el comando de giro.

Modulación de ancho de pulso (PWM)

La modulación de ancho de pulso es una técnica utilizada para controlar dispositivos, o para proveer un voltaje variable de corriente continua [Jácome04], es decir se dosifica la potencia. Algunas aplicaciones en las que se utiliza PWM son: control de motores, iluminación y temperatura.

La señal generada tendrá frecuencia fija y tiempos de encendido y apagado variables. En otras palabras, el período de la señal se mantendrá constante, pero la cantidad de tiempo que se mantiene en alto y bajo dentro de un período puede variar. El ciclo de trabajo del total del período ($t_{per} = t_{on} + t_{off}$) es t_{on}, es decir el tiempo que se mantiene en alto (Figura 2.7(a)). Por ejemplo, una señal PWM que tiene $10V$ de amplitud y un 50 % de ciclo de trabajo (Figura 2.7(b)), provee $5V$ de salida promedio. Cuando se incrementa o decrementa el ciclo de trabajo de una señal PWM, la salida promedio se incrementa o decrementa respectivamente.



(a) Señal modulada para una velocidad promedio baja

(b) Señal modulada con un ciclo de trabajo del 50 %

Figura 2.7: Esquema de modulación de ancho de pulso (PWM)

Puente H

En un puente H (Figura 2.8), los switchs son abiertos y cerrados de manera tal que al poner un voltaje de una polaridad a través del motor la corriente fluye en una dirección (activando los campos magnéticos y provocando que gire), o un voltaje de polaridad opuesta, causando flujo de corriente a través del motor en dirección opuesta para rotar en el otro sentido.

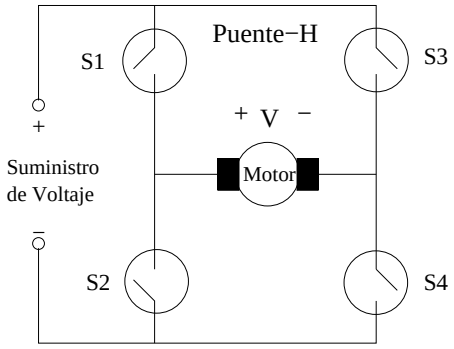


Figura 2.8: Un motor es controlado mediante una topología de circuito conocida como Puente H.

En el control de velocidad de los motores utilizados (24V y 12V), los switchs de los puentes H son abiertos y cerrados a diferentes instantes para aplicar diferentes voltajes a través del motor, es decir se aplica el esquema PWM. Los puentes utilizados tienen como entrada tres señales: dirección, freno, y PWM; y como salida, el voltaje promedio que se aplicará a los motores.

Comandos disponibles para controlar el robot

El microcontrolador de la base del robot recibe los comandos a través de su puerto serie. La interfaz entre un operador y el robot puede ser por medio de una terminal en una sesión remota, o a través del *gamepad* conectado a la computadora portátil abordo. Además de comandos para realizar las operaciones de avanzar, girar a la izquierda, girar a la derecha y retroceder, el microcontrolador permite la ejecución de comandos para configurar la aceleración, así como para reportar información referente al estado del robot y el voltaje de las baterías. La Tabla 2.1 muestra un resumen de los comandos con sus respectivos parámetros y la función que realizan. En el Apéndice A se cuenta con una descripción más

detallada de dichos comandos.

Tabla 2.1: Comandos disponibles para controlar el robot

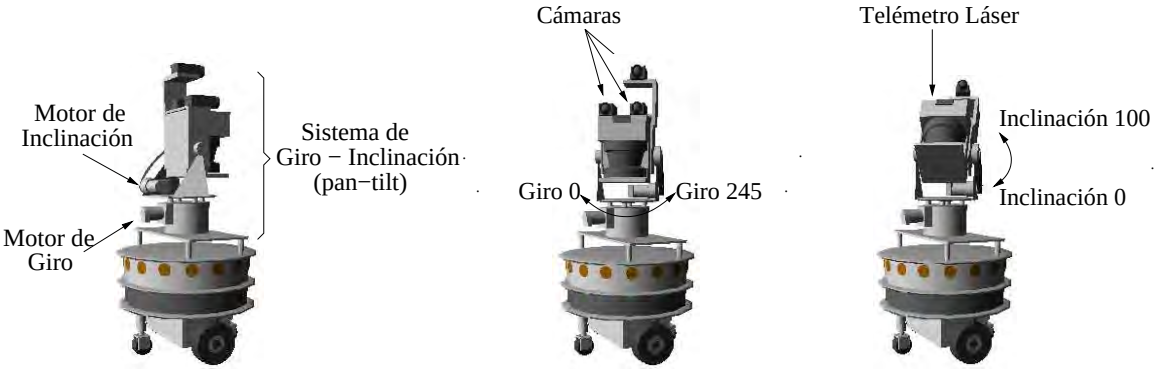
Comando	Parámetros	Función
a	$p1$	Avanza $p1$ unidades
d	$p1$	Gira a la derecha $p1$ unidades
i	$p1$	Gira a la izquierda $p1$ unidades
p		Paro de emergencia de los motores de las ruedas
r	$p1$	Retrocede $p1$ unidades
w	$p1$ 1 1 1	Espera p1 unidades de tiempo por paro de motores de las ruedas y el sonar
k	$pi\ pf\ ms$	Define la aceleración de los motores de las ruedas
l		Reporta el estado del robot
t		Regresa el voltaje de las baterías en milivolts

2.5. Sistema Pan-Tilt

El sistema *Pan-Tilt* (Giro-Inclinación) soporta tres cámaras y el telémetro láser, con un peso total de 5 kilogramos. No es común encontrar un sistema comercial con tales características. Al igual que la base del robot, astá hecho de acero inoxidable y aluminio. Utiliza dos motores Pittman de 24V y 12V para las operaciones de giro y de inclinación respectivamente, en la Figura 2.9(a) se muestra la ubicación de éstos motores en el sistema *pan-tilt*.

La reducción en la caja de engranes del motor de inclinación (ver Sección 2.4) hace posible un control de posicionamiento más fino, permitiendo escanear a intervalos de inclinación de 0,68°. El rango de inclinación va desde $-20°$ a 45°. El rango de giro es de 0° a 360°. El sistema de referencia para el posicionamiento, tanto en el giro como en la inclinación, es absoluto, es decir se parte de una posición inicial. Al energizar el robot se inicializan las posiciones de giro e inclinación en 0° y $-20°$ respectivamente.

Los comandos para el control del *pan-tilt* reciben como parámetro la posición, de 0 a 254 para el giro, y de 0 a 100 para la inclinación (ver Figura 2.9(a) y (b) respectivamente). En la Tabla 2.2 se muestran los comandos disponibles para el control del *pan-tilt*, así como los parámetros que reciben y cuáles son sus funciones. Una descripción más detallada de los mismos se presenta en el Apéndice A.



(a) Sistema de giro inclinación (*Pan-Tilt*)

(b) Operación de Giro

(c) Operación de Inclinación

Figura 2.9: El sistema de giro inclinación soporta las cámaras y el telémetro láser. Tiene un posicionamiento con referencia absoluta, de 0 a 100 posiciones de inclinación y de 0 a 245 posiciones de giro.

Tabla 2.2: Comandos disponibles para el control del *Pan-Tilt*

Comando	Parámetros	Función
e	$p1$	Elevarse a la posición $p1$
g	$p1$	Girar a la posición $p1$
k	$pi\ pf\ ms$	Define la aceleración del motor de giro
K	$pi\ pf\ ms$	Define la aceleración del motor de inclinación
l		Reporta el estado del *pan-tilt*
p		Paro de emergencia de los motores

2.6. Sensores

Los sensores son transductores que convierten algún fenómeno físico en señales eléctricas que el microcontrolador puede leer [Jones98]. Su principal función es trasladar la información del mundo real al mundo abstracto de la unidad lógica de procesamiento. Existe una gran variedad de sensores en robótica, pero los mayormente asociados con la robótica móvil son aquellos que miden la distancia que las ruedas han viajado a lo largo del piso, sensores que miden cambios inerciales, y aquellos que miden la estructura externa en el ambiente. En [Dudek00] se clasifican en dos clases: (1) sensores visuales, los cuales utilizan la luz reflejada por los objetos en el ambiente para razonar sobre su estructura, (2)

sensores no visuales, los cuales utilizan otras modalidades para sensar el ambiente (audio, inercia, y otros). Además se dice que un sensor es activo si para efectos del sensado emite algún tipo de energía, y se dice que es pasivo en caso contrario.

El robot utilizado cuenta con sensores visuales: cámaras; y no visuales: sensores de voltaje, interruptores, odómetros, sonares, y el telémetro láser. Los interruptores son dispositivos simples que reportan un valor binario: cerrado o abierto. En el sistema *pan-tilt* se utilizan tres, uno indica la posición inicial (inferior) de inclinación, los otros dos indican la posición inicial y final (mínima y máxima) de giro. Los convertidores analógico-digitales del microcontrolador son utilizados como sensores de voltaje para medir la carga de las baterías. En las siguientes subsecciones se describen los demás sensores utilizados en el robot.

2.6.1. Odómetro

Un odómetro es un sensor que mide la posición, o rango de rotación, de un eje, en nuestro caso la flecha del motor. La señal entregada por este sensor puede ser de dos tipos: (1) un código que corresponde a una orientación en particular del eje, o (2) un tren de pulsos. En el primero de los casos se le llama odómetro absoluto, en el segundo caso odómetro incremental [Jones98]. En este robot se utilizan odómetros incrementales.

El odómetro incremental es un disco que tiene numerosos hoyos o ranuras a lo largo de su límite exterior, tal como se muestra en la Figura 2.4(b). Un LED infrarrojo es colocado sobre uno de los lados del disco, así que la luz es proyectada a través de los hoyos. Un fototransistor sensitivo al infrarrojo es colocado en posición opuesta al LED, como se muestra en la Figura 2.10, así que cuando el motor y el disco giran, los hoyos pasan la luz intermitentemente. El resultado obtenido, tal como lo ve el fototransistor, es una serie de ráfagas de luz; es decir, un tren de pulsos [McComb00].

La Figura 2.11 muestra las señales generadas por el odómetro. Los odómetros incrementales permiten saber el sentido del giro utilizando dos LEDs y dos fotoransistores colocados en cuadratura. Esto es, uno adelantado 90° del periodo del otro. Así conociendo como cambian las señales (A y B), se sabrá el sentido del giro. Los odómetros HP utilizados por los motores Pittman, además del tren de pulsos en cuadratura (señales A y B), tiene una señal de índice (señal I) que se activa cada vez que se produce una vuelta completa de la flecha del motor.

Así pues, los odómetros permiten hacer una retroalimentación del estado de giro

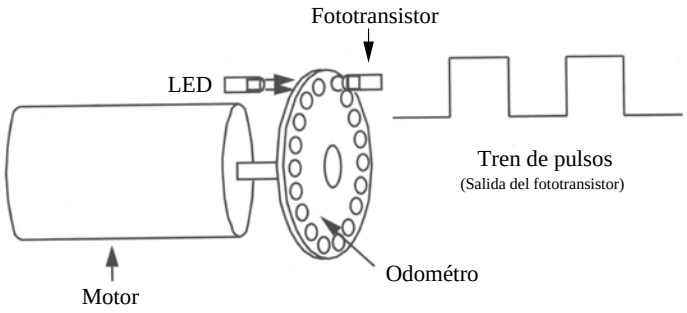


Figura 2.10: Esquema del odómetro incremental adjunto al motor

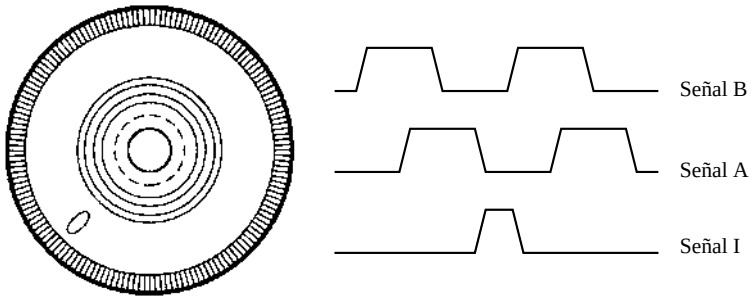


Figura 2.11: Señal en cuadratura generada por el odómetro

de la flecha de los motores, y con ello tener un mayor control sobre la posición, velocidad y aceleración del robot.

En las dos siguientes secciones se presentan los dos sensores con que cuenta el robot para la medición de distancias: los sonares y el telémetro láser.

2.6.2. Sonares

Los sonares son sensores acústicos utilizados ampliamente para la medición de distancias [Borja01]. La forma en que trabajan estos sensores es muy parecida al mecanismo utilizado por los murciélagos para cazar. Un murciélago emite una onda ultrasónica y espera el eco, así determina la ubicación de su presa.

Generalmente un sonar es un dispositivo de dos componentes: una tarjeta controladora (Figura 2.12(a)), y un transductor electrostático (Figura 2.12(b)). El transductor primero funciona como emisor y después como receptor. Inicialmente emite un sonido de

corta duración y alta frecuencia, el sonido viaja y es reflejado de regreso a él por los objetos en su camino, es entonces cuando el transductor funciona como receptor del sonido reflejado y lo convierte en una señal eléctrica.

(a) Tarjeta controladora

(b) Transductor

(c) Anillo de sonares

Figura 2.12: El sonar Polaroid utilizado en el robot consta de una tarjeta controladora y un transductor, el robot contiene un anillo de 16 sonares (Transductores).

En el robot se tiene un anillo de 16 sonares Polaroid (ver Figura 2.12(c)), con un rango de medición efectiva de 4 pulgadas a 40 pies, sobre un cono de aproximadamente 15 grados. Los 16 sonares son operados mediante una tarjeta controladora Polaroid 6500, vía un multiplexor 16-a-1, tal como se ilustra en la Figura 2.13. La tarjeta controladora es conectada al microcontrolador, el cual emite una señal de disparo (*init*) para decirle a la tarjeta controladora que envíe un disparo a través del sonar. Entonces el microcontrolador escucha por el eco del disparo en la tarjeta controladora (*echo*). El multiplexor esta situado entre la tarjeta controladora y los sonares; antes de cada disparo, el microcontrolador elije cual sonar deberá ser conectado para emitir dicho disparo.

La señal que la tarjeta 6500 de Polaroid usa para enviar un disparo a través del sonar es aproximadamente de 400 volts pico-a-pico. Esto provoca un problema, debido a que el nivel de lógica común de los chips multiplexores no puede ser usada para intercambiar ésta señal para la selección apropiada del sonar. Para resolver el problema se tomó la solución presentada en [Musliner95], además se implementó un multiplexor hecho a la medida que utiliza un dispositivo optoaislador llamado *triac coupler* [Inc.04a]. El chip utilizado para la multiplexión es un chip 74LS154 de 4-líneas a 16-líneas, toma 4 líneas de dirección del

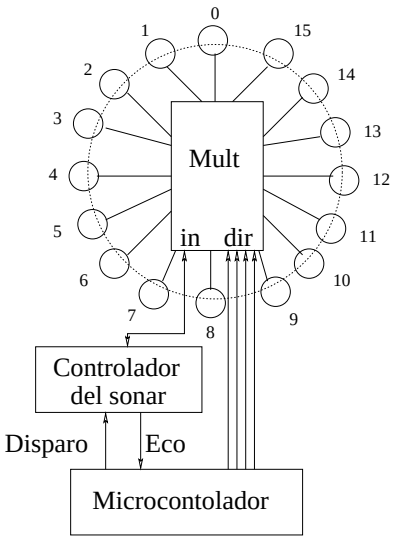


Figura 2.13: Control del anillo de sonares

Tabla 2.3: Comandos disponibles para el control de los sonares

Comando	Parámetro	Función
s0		Deshabilita los disparos del anillo de sonares
s1		Inicia el disparo de los sonares
s2	*te tr*	Inicializa los parámetros de disparo
S	s_0 s_1 ...	Retorna las lecturas de los sonares

microcontrolador y elije una entre 16.

La Tabla 2.3 muestra los comandos relacionados con el anillo de sonares, los parámetros que éstos toman y la función que realizan cada uno de ellos.

Debido a que el anillo de sonares forma parte del robot, se explicó en esta sección; sin embargo, el sensor de mayor importancia en la medición de distancias para este trabajo es el telémetro láser, el cual es abordado en la siguiente sección.

2.6.3. Telémetro láser

Como ya se mencionó, un telémetro láser es utilizado en esta tesis. En [Jensfelt01] se mencionan ventajas de utilizar telémetros lásers, algunas de éstas son:

- Son tan rápidos que, en la mayoría de los casos, se puede considerar que las mediciones

son prácticamente instantáneas. Lo que significa que no es necesario algún tipo de compensación debido al movimiento del robot en la medición, ya que un escaneo solo necesita de aproximadamente 40 ms para efectuarse. Esto contrasta con un sistema de sonares, con el cual se necesita esperar cierto tiempo para recibir el eco.

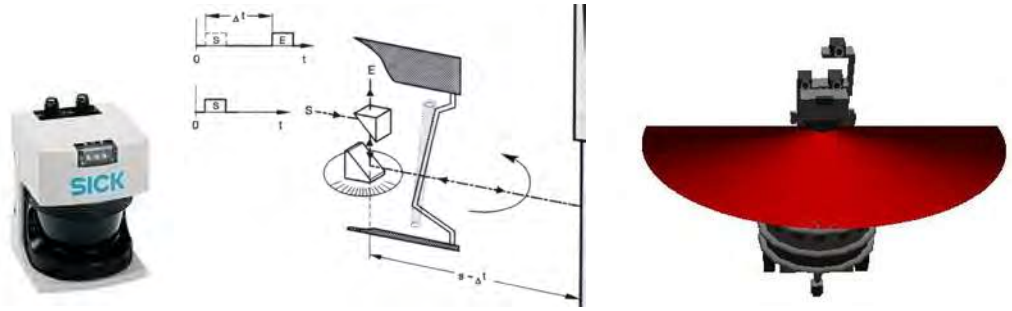
- La precisión en las mediciones es bastante buena. En el telémetro que se utilizó en este trabajo es de ± 2 cm, pero existen otros con una precisión hasta de ± 1 cm.
- La resolución angular puede ser mucho mejor en un telémetro láser, que en un sistema de sonares, dependiendo de la configuración del modo de operación del telémetro.
- Los datos provenientes del telémetro láser pueden ser interpretados directamente como la distancia a un obstáculo u objeto en una cierta dirección.

Para visualizar claramente de donde surgen estas ventajas al utilizar un telémetro láser, como sensor de medición de distancia, enseguida se analiza qué es y cómo opera el telémetro láser.

La Figura 2.14(a) muestra el telémetro láser utilizado, que es un sensor óptico de medición de distancias, el cual rastrea a su alrededor con un barrido de láser infrarrojo [SICb]. El sensor opera sobre el principio del tiempo de reflexión de la luz, también conocido como TOF (del Inglés *Time Of Flight*). En un sistema TOF un pequeño pulso láser es enviado y el tiempo en que tarda en regresar es medido; el tiempo entre el envío y la recepción determina la distancia.

De forma específica, el telémetro utilizado opera de la siguiente forma: se emiten pulsos de luz muy cortos, al mismo tiempo un cronómetro electrónico corre, si la luz encuentra un objeto, ésta es reflejada y regresa al sensor; del tiempo entre el envío y la recepción, el sensor calcula su distancia al objeto. Dentro del sensor existe un espejo que rota uniformemente (Figura 2.14(b)), el cual desvía los pulsos de luz, así forma un barrido de un área semicircular, como el mostrado en la Figura 2.14(c). Determinando el ángulo del espejo, el sensor detecta en que dirección se localiza el objeto reflejado. Además el contorno de dicho objeto puede ser determinado a partir de la secuencia de pulsos recibidos.

El modelo de láser utilizado es el LMS209-S02 de SICK (Figura 2.14(a)). Este es un telémetro láser con un ángulo de rastreo máximo de 180° y una resolución lateral que puede variar entre 0.25° y 1°. La precisión de un simple disparo es de ± 2 cm y una distancia de hasta 32 m. Utilizando la interfaz serial, el rango de transmisión es de un



(a) Telémetro láser (b) Principio de operación (c) Barrido semicircular del láser

Figura 2.14: El telémetro láser mide la distancia haciendo un barrido semicircular, para ello utiliza la técnica de tiempo de vuelo de la luz.

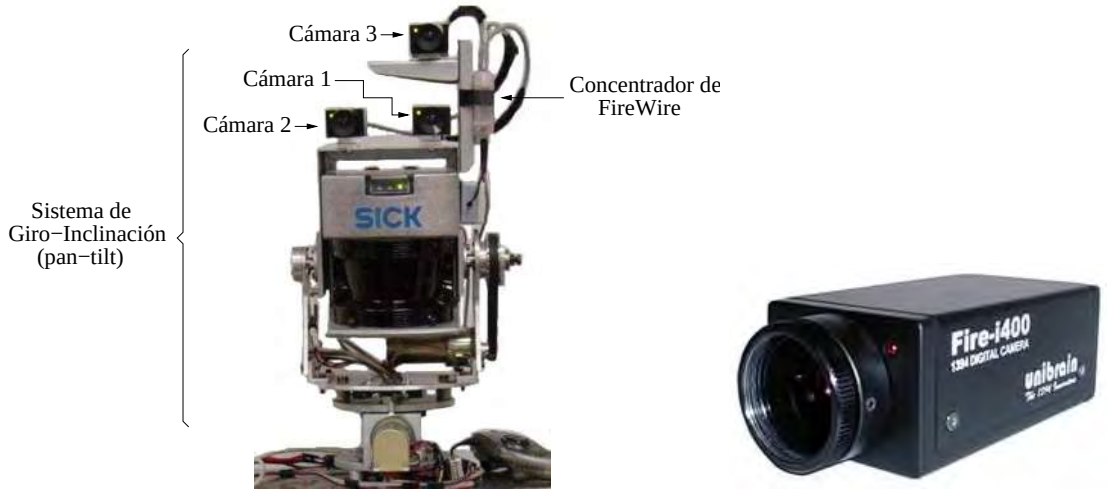
máximo de 500 Kbaudios, con esta velocidad de transferencia es posible transmitir todas las mediciones en tiempo real. El protocolo de comunicación serial entre la computadora huésped y el telémetro se realiza mediante los paquetes (telegramas) definidos en [SICa], los cuales permiten realizar la configuración del telémetro, así como consultar información del mismo.

En particular en este trabajo, el telémetro láser fue configurado para operar a una velocidad de transferencia serial de 9600 bps, con un ángulo máximo de rastreo de 180° a intervalos laterales de 0.5°, y una distancia máxima de 32000 mm, es decir 32 metros reportando la distancia en milímetros.

Además de tener sensores de medición de distancia, como los sonares y el telémetro láser, el robot también cuenta con cámaras, que a continuación se describen.

2.6.4. Cámaras

El robot tiene tres cámaras CCD (del Inglés *Charge-Coupled-Device*) [Trucco]. La posición en la que están colocadas, sobre el sistema de giro-inclinación, forma un sistema trinocular de visión estéreo [Vieyra04] (ver Figura 2.15(a)). Las tres cámaras se conectan al concentrador de *FireWire*, y éste a su vez, a la computadora portátil a bordo.



(a) Sistema trinocular de visión estéreo (b) Cámara Fire-i400

Figura 2.15: Sistema trinocular de visión estéreo montado sobre el sistema de giro-inclinación y tipo de cámaras utilizadas.

El modelo de cámara utilizado se muestra en la Figura 2.15(b), la cual corresponde a una Fire-i400 de *Unibrain*, con un lente de 2.1 mm. Entre las principales características de la cámara destacan:

- Interfaz IEEE-1394a (*FireWire*) de 400 Mbps, dos puertos y 6 pines.
- Resolución de 640 x 480 píxeles.
- Modos de vídeo: YUV (4:1:1, 4:2:2, 4:4:4), RGB-24bits y Monocromático-8bits.
- Rango de captura de 30, 15, 7.5 y 3.75 cuadros por segundo.
- Lentes del tipo C-Mount.

También se han probado otro tipo de cámaras modelo MDCS-C de *Videre Design*. Estas cámaras pueden capturar hasta 60 cuadros por segundo en una resolución de 640x480 píxeles, pero también soportan la resolución de 1260x980 píxeles a rangos de captura menores. En general la información referente a las cámaras *FireWire*, soportadas en Linux se puede encontrar en [Douxchamps04].

2.7. Conclusiones

En este capítulo se describieron las principales características de un robot móvil construido localmente. Se espera que la información aquí descrita sea útil a otros investigadores y fomente el desarrollo de tecnología local, para poder contar con una plataforma abierta de desarrollo e investigación en robótica.

Lo que se propone es utilizar como plataforma de software el sistema operativo Linux, por ser un sistema abierto y gratuito, desarrollando código disponible para todo el público, es decir, sin código propietario.

El capítulo también contiene todas las referencias necesarias para localizar en Web el hardware, los controladores (*drivers*) y el software para Linux, utilizados en el robot. El software desarrollado especialmente para dicho robot también está disponible en Web. Además, en un futuro cercano el robot podría ser comercializado y distribuido por Cervantes Co., una pequeña compañía en Paracho Michoacán, como una plataforma abierta.

El microcontrolador y el tipo de motores utilizados permiten un buen control del robot bajo el esquema PWM. Sin embargo, se preve cambiar estos motores por motores de imanes permanentes sin escobillas, esto para mejorar el control de posicionamiento.

El robot y los sensores con los que está equipado son suficientes para abordar y resolver diversos problemas en robótica, entre éstos están los de la localización local y construcción de mapa simultáneos (SLAM), así como el de la reconstrucción 3D de ambientes interiores, que son de interés principal en esta tesis. Además se podrían abordar otros problemas como: planeación y seguimiento de trayectorias, visión estéreo para reconstrucción 3D, navegación con evasión dinámica de obstáculos, y otros más.

Capítulo 3

Localización local y construcción de mapas 2D

El problema de la localización local y construcción de mapas bidimensionales (2D), tiene gran importancia en el campo de la robótica móvil. Desde hace al menos dos décadas, y hasta la fecha, ha sido una área de investigación muy activa; en la actualidad es considerado como el problema de percepción más difícil en la robótica [Thrun02].

3.1. Introducción

La construcción de mapas 2D involucra el proceso de adquisición, a través de sensores, de modelos espaciales del ambiente físico que rodea al robot móvil [Thrun02]. La localización local involucra determinar la posición y orientación del robot partiendo de una estimación inicial de su localización [Latecki04]. Es decir, se pretende relocalizar al robot después de realizar un movimiento que en la mayoría de las ocasiones es pequeño (por ejemplo, desplazarse un metro). Como ya se ha mencionado, la combinación simultánea de ambos, localización local y construcción de mapas 2D, es conocida como SLAM (*Simultaneous Localization And Mapping*).

Aunque un robot puede realizar ciertas tareas sin contar con una representación interna de su ambiente, muchas otras tareas sí requieren tal representación. La manera más natural en la que un robot representa su ambiente es un mapa [Dudek00]. Cualquiera que sea el método empleado en la solución del problema SLAM, la representación espacial del ambiente físico, es decir el mapa, es una pieza clave. Existen muchos tipos de mapas, los

más comunes cuando se trabaja con robots móviles en ambientes interiores estructurados o semiestructurados son los siguientes [Jensfelt01]:

- *Mapas topológicos*. El mundo es representado como un grafo conectado. Los nodos en el grafo corresponden a lugares de importancia y las aristas son las conexiones entre los nodos involucrados (por ejemplo, un corredor).
- *Mapas de características*. Las características geométricas de los obstáculos son usadas para representar el mundo. Ejemplos comunes son los puntos y líneas.
- *Mapas de rejilla*. En lugar de restringir la representación del mapa a características específicas, el mundo es dividido usando una rejilla. Cada celda de la rejilla representa una pequeña área o volumen de mundo y tiene asociado un cierto valor para expresar que la parte del ambiente que representa está ocupada o libre.
- *Métodos basados en apariencia*. En lugar de tener una representación intermedia de los datos del sensor, ya sea en forma de rejillas o características, los datos del sensor son directamente utilizados para representar el ambiente.

En nuestro caso, la representación del ambiente se realiza mediante mapas de características. En particular se trata de mapas de puntos. Esto se debe principalmente a la metología empleada para la localización local la cual será explicada posteriormente.

En la construcción de mapas, un robot utiliza sus sensores: cámaras, telémetro láser, sonares, etc. Sin embargo, dichos sensores están sujetos a errores, comúnmente ruido de medición. Más importante aún, la mayoría de los sensores están sujetos a estrictas limitaciones físicas. Por ejemplo, la luz emitida por el telémetro láser 2D, o el sonido emitido por los sonares de nuestro robot, no pueden atravesar muros. Estas limitaciones hacen necesaria la navegación del robot a través del ambiente cuando se construye el mapa 2D. Los comandos para el control de movimiento (ver Sección 2.4.1) durante la navegación proveen información importante sobre la localización del robot. Sin embargo, el movimiento del robot también está sujeto a errores, a causa de los derrapes e irregularidades del piso, por lo que la información provista por el control de movimiento es insuficiente para calcular la posición y orientación del robot respecto de su ambiente.

La localización local involucra el alineamiento (*matching*) de un conjunto de observaciones con algún mapa establecido. En [Borenstein96] se describen diferentes aproxi-

maciones para alinear observaciones. Dudek y Jenkin [Dudek00] clasifican los métodos de alineamiento estándar en las siguientes categorías:

- *Alineamiento Datos-Datos*. Directamente se tratan de emparejar los datos de rango actuales y el conjunto de datos previamente almacenados o predichos a partir del mapa.

- *Alineamiento Datos-Modelo*. Se tratan de emparejar los datos observados con modelos de mapa más abstractos. Se asocian los modelos (por ejemplo: líneas, segmentos de línea, polígonos) con los datos.

- *Alineamiento Modelo-Modelo*. Trata de emparejar modelos almacenados en el mapa con modelos generados de las observaciones actuales.

En el caso típico de un robot móvil equipado con un telémetro láser 2D, el alineamiento de observaciones, es llamado alineamiento de rastreo (*scan matching*) [Latecki04], pero el método de alineamiento específico puede ser cualquiera de los tres anteriores.

La mayoría de los mejores métodos de alineamiento de rastreo del tipo datos-datos, están basados en una optimización de mínimos cuadrados [Gutmann96, Latecki04, Lu97]. Sin embargo, la optimización por mínimos cuadrados es notoriamente sensible a los datos atípicos, debido a que tienden a contribuir demasiado sobre la solución final, ya que éstos no siguen una distribución Gausiana. En [Rousseeuw03] se realiza una discusión a fondo de este punto.

En este capítulo se presenta un método de alineamiento de rastreo del tipo datos-datos, basado sobre un estimador robusto en lugar del estimador de mínimos cuadrados. La meta de la estadística robusta es describir la estructura del mejor ajuste al volumen de los datos e identificar los datos atípicos [Hampel86]. En particular se va a utilizar el estimador Lorentziano, el cual es un estimador robusto que ha sido aplicado exitosamente en varias áreas, por ejemplo en visión computacional en la reconstrucción de superficies [Black96].

3.2. Adquisición de datos

Como ya se ha mencionado, el robot utiliza un telémetro láser 2D (ver Sección 2.6.3), cubre un rango de 180° a intervalos de 0,5°, con una distancia máxima de 32000mm.

Así, por cada rastreo del telémetro láser se tiene un conjunto de 361 mediciones. Sin embargo, no siempre el telémetro láser mide la distancia a los obstáculos. En seguida se describen los fenómenos que ocasionan lecturas erroneas del telémetro láser.

3.2.1. Ruido en las mediciones

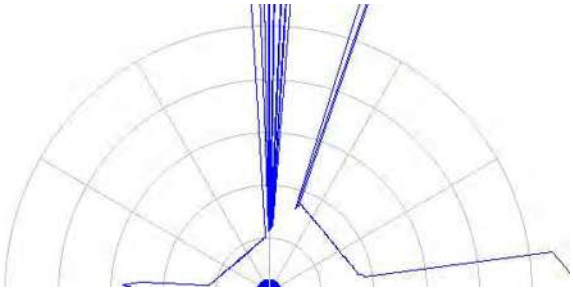
El ambiente que rodea al robot esta formado por diversos objetos: muros, sillas, mesas, puertas, ventanas, macetas, cajas, etc.; y las propiedades reflectivas de los materiales con los que están hechos estos objetos, así como el tipo de superficie geométrica de los mismos, pueden provocar errores de medición. En el manual de operación del láser LMS209-S02 de SICK [SICb] se presentan las propiedades de diversos materiales en términos de porcentaje de reflectividad.

Al sensar, el telémetro láser emite un pulso de luz; el pulso incide en la superficie de los objetos que rodean al robot y se refleja de nuevo al sensor. En base al tiempo de emisión y de recepción, la distancia al objeto es calculada. Sin embargo, no todos los objetos reflejan la luz de igual forma. Puede ocurrir que los pulsos de luz no se reflejen debido al color, material o forma de la superficie; en otras palabras el pulso no regresa al sensor. En otros casos, los objetos pueden actuar como espejos reflejando el pulso en otra dirección antes de retornar al sensor. En la Figura 3.1 se presentan dos casos en los cuales los objetos que detecta el telémetro láser provocan errores de medición.

La Figura 3.1(a) muestra al robot con el telémetro láser orientado en dirección a una puerta de color negro con superficie ligeramente ondulada. La lectura del sensor, para éste caso, se presenta en la Figura 3.1(b). Los puntos consecutivos de las lecuturas reportadas por el telémetro láser se unieron por líneas. Así, cuando se presenta una lectura erronea, la continuidad de los segmentos de línea se ve interumpida, y se traza una línea de longitud máxima. Si el pulso de luz emitido no regresa al sensor, el valor reportado es el máximo valor ($32m$). Sin embargo, cuando el robot está justo frente a la misma puerta, ésta es completamente detectada.

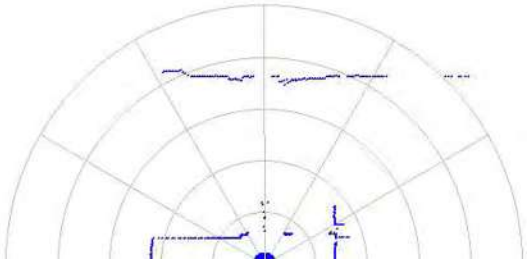
En el segundo caso, presentado en la Figura 3.1(c), algunas mediciones muestran que la puerta de cristal tiene un comportamiento de tipo espejo (ver Figura 3.1(d)). El pulso de luz es reflejado por el cristal en dirección al robot, pero no es detectado por el sensor sino hasta que el robot se refleja en el cristal; esto provoca que aparentemente el robot este del otro lado de la puerta de cristal. Algo similar pasa con las cajas de cartón junto a la

(a) Puerta negra



(b) Rastreo láser

(c) Puerta de cristal



(d) Rastreo Láser

Figura 3.1: Las superficies con reflectividad pobre y objetos transparentes provocan errores de lectura. Las puertas negras no son detectadas, mientras que las puertas de cristal, que en algunas ocasiones tampoco son visibles, algunas veces pueden actuar como espejo.

puerta.

Para visualizar más claramente los fenómenos anteriores, en la Figura 3.2 se presentan los posibles casos que pudieron haber ocurrido durante el sensado sobre la puerta de cristal. Los pulsos 1, 3, 5 y 7, no tienen problema y retornan normalmente al sensor. El pulso 2 es reflejado en dirección a las cajas mostradas en la Figura 3.1(c), a su vez, la superficie de las cajas refleja el pulso de luz sobre el cristal en cierto ángulo que ahora detecta el sensor. La distancia finalmente reportada es la suma de distancia del sensor a la puerta de cristal y de la puerta a las cajas. Algo similar ocurre con el pulso 4. Las líneas punteadas hacen referencia a la distancia finalmente reportada en ambos casos. En los pulsos 6 y 8, el pulso incide sobre una superficie en un cierto ángulo, lo cual provoca que el pulso nunca pueda regresar al sensor, y entonces se reporta la máxima distancia posible que puede sensar el

telémetro láser.

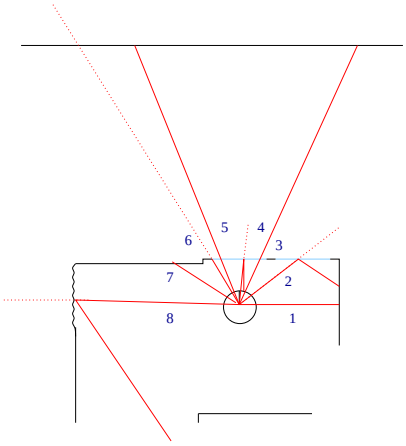


Figura 3.2: Posibles casos de reflexión de los pulsos de luz emitidos por el telémetro láser cuando se sensa un ambiente que contiene una puerta de cristal.

Todos los casos que provocan errores de medición dan como resultado lo que se conoce como *datos atípicos*, los cuales pueden estar cerca de la superficie sensada (pulsos 3 y 4) o visiblemente retirados (pulsos 6 y 8). Tratar con datos atípicos no es fácil. Si se da mucha importancia a estos datos, tal como lo hace mínimos cuadrados, el mejor alineamiento (*match*) arrojado no necesariamente ajusta a la mayoría de los puntos.

Las lecturas del láser necesitan transformarse en puntos 2D de coordenadas cartesianas rectangulares, para poder aplicar el método de alineamiento de rastreo de dos lecturas consecutivas. En seguida se describe como realizar esto.

3.2.2. Transformación de las lecturas en puntos

El telémetro láser realiza una lectura cada $0.5°$ desde $-90°$ hasta $90°$. Por lo que se tiene una medición de distancia para cada uno de los 361 ángulos posibles. Estas distancias están reportadas en coordenadas polares. Sin embargo, los métodos de alineamiento de rastreo empleados en este capítulo trabajan con coordenadas cartesianas rectangulares. Por lo tanto, el primer paso será convertir las coordenadas polares en coordenadas cartesianas rectangulares.

Sea d la distancia reportada por el telémetro láser; sea α su ángulo correspondiente. Si estas coordenadas polares se sobreponen a un plano cartesiano correspondiente a las

lecturas del telémetro láser, haciendo coincidir el polo con el origen (ver Figura 3.3), se obtienen las relaciones que se muestran a continuación:

$$x = dsin\alpha \tag{3.1}$$

$$y = dcos\alpha \tag{3.2}$$

$$d = \sqrt{x^2 + y^2} \tag{3.3}$$

$$\alpha = Tg^{-1}\left(\frac{y}{x}\right) \tag{3.4}$$

Así, para cada lectura reportada en coordenadas polares $P(d, \alpha)$, bastará con aplicarle las Ecuaciones 3.1 y 3.2, para calcular sus correspondientes coordenadas rectangulares en el plano cartesiano. Las ecuaciones 3.3 y 3.4 resolverían el caso inverso.

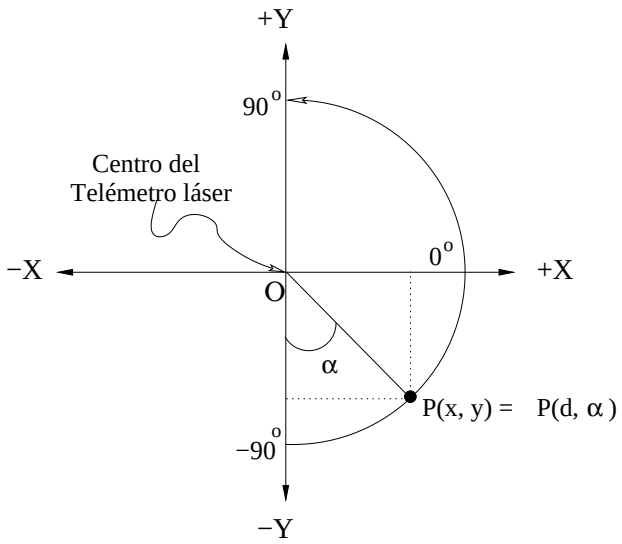


Figura 3.3: Plano polar y plano cartesiano de las lecturas del telémetro láser coincidentes en el polo y origen respectivamente, para transformar un punto $P(d, \alpha)$, de coordenadas polares, en un punto $P(x, y)$ de coordenadas cartesianas rectangulares.

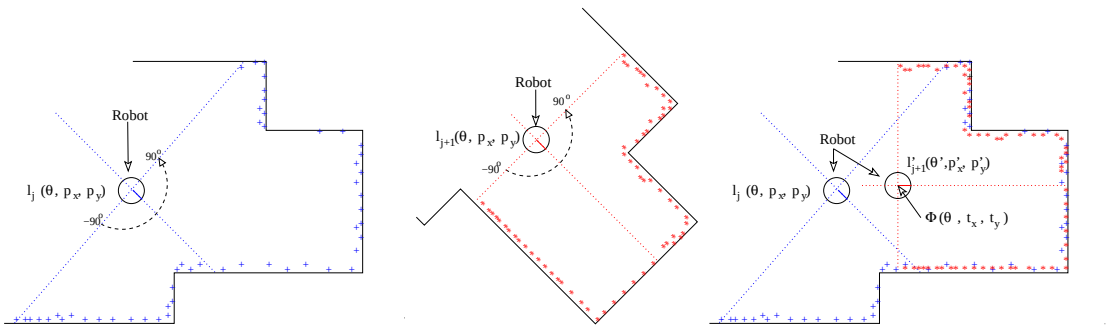
La secuencia de lecturas del telémetro láser transformadas en puntos $P(x_i, y_i)$, para $i = 0, 1, 2, ..., 360$, en cordenadas cartesianas rectangulares, forman lo que llamaremos un *rastreo*.

En seguida se hará la definición del problema de localización local por alineamiento de dos rastreos consecutivos.

3.3. Definición del problema

3.3.1. Estimación de la localización local por alineamiento de rastreos

Sea $l_j(\theta, p_x, p_y)$, un rastreo realizado en la localización l_j con orientación θ y posición (p_x, p_y) dentro de un sistema de referencia global. Suponiendo que el robot realiza un rastreo l_j (ver Figura 3.4(a)), la localización de ese rastreo momentáneamente será tomada como sistema de referencia fijo. Posteriormente el robot se desplaza dentro del ambiente estático y realiza otro rastreo l_{j+1}(ver Figura 3.4(b)). El problema es formulado como sigue: dado un primer rastreo l_j, y un segundo rastreo l_{j+1}, encontrar la transformación rígida Φ (en una rotación θ y traslación (t_x, t_y)) tal que aplicada a los puntos del rastreo l_{j+1} se obtenga l'_{j+1}, de puntos transformados por Φ (x'_i, y'_i) que den el mejor alineamiento con los puntos del rastreo l_j (ver Figura 3.4(c)).



(a) Rastreo l_j (b) Rastreo l_{j+1} (c) Alineamiento de l_{j+1} con respecto a l_j

Figura 3.4: Alineamiento de dos rastreos consecutivos: l_{j+1} con respecto a l_{j+1}.

3.3.2. Criterio para el alineamiento de rastreos

El método para alinear dos rastreos consecutivos es del tipo datos-datos, para nuestro caso se trata de puntos contra puntos. Los puntos (x_i, y_i), que pertenecen al rastreo l_{j+1}, son asociados (mapeados) con su correspondiente punto más cercano $(x_{\psi(i)}, y_{\psi(i)})$ en el rastreo previo l_j. Así, el criterio para determinar el mejor ajuste, entre los dos rastreos consecutivos l_j y l_{j+1}, se plantea como la minimización de la distancia entre los puntos de

los rastreos, después de aplicar la transformación rígida $\Phi(\theta, t_x, t_y)$ a los puntos del rastreo l_{j+1}. El objetivo es encontar la transformación rígida Φ que logre dicho mínimo.

En seguida se describe a detalle el uso y aplicación del estimador Lorenziano, así como el método empleado en la localización local.

3.4. Localización basada en el estimador Lorentziano

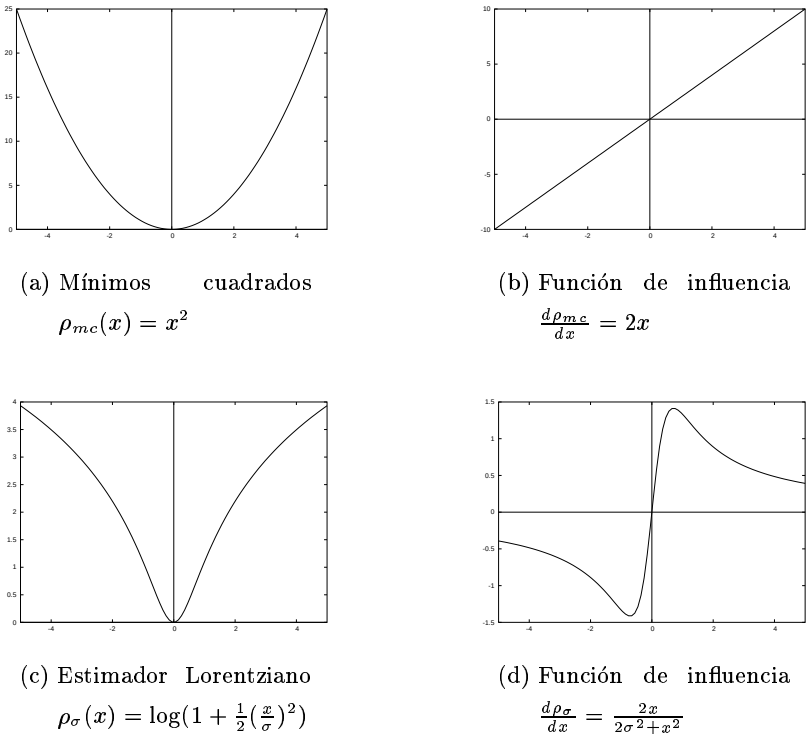
El estimador Lorentziano es un estimador robusto. En general, el término robusto hace referencia a un estimador estadístico, que es insensible a desviaciones de una suposición ideal para la cual el estimador es óptimo [Porikli98]. Cuando las desviaciones tienen una distribución Gaussiana, o se tiene una buena estimación de la varianza, el mejor estimador es el de mínimos cuadrados. Sin embargo, las desviaciones de los datos atípicos no siguen tal distribución [Rousseeuw03] y no es posible estimar la varianza. Un análisis realizado por el Dr. Félix Calderón, deduce que el estimador Lorentziano esta basado en una distribución T [Walpole86] donde la varianza no existe. Estima que para una distribución T studen con dos grados de libertad se obtiene el estimador Lorentziano.

La Figura 3.5 ayuda a visualizar el comportamiento de los estimadores de mínimos cuadrados $\rho_{mc}(x) = x^2$, y el Lorentziano $\rho_\sigma(x) = \log(1 + \frac{1}{2}(\frac{x}{\sigma})^2)$. La función de influencia [Hampel86] del estimador de mínimos cuadrados $\frac{d\rho_{mc}}{dx} = 2x$ (Figura 3.5(b)) muestra que grandes errores, o valores, tienen gran influencia sobre el comportamiento del estimador. Por otro lado, con la función de influencia del estimador Lorentziano $\frac{d\rho_\sigma}{dx} = \frac{2x}{2\sigma^2+x^2}$ (Figura 3.5(d)), los errores pequeños son más significativos que los errores grandes, y el parámetro σ controla la influencia de los errores grandes.

Un valor pequeño de σ ignoraría errores grandes, mientras que valores más grandes de σ les otorgaría una mayor influencia a dichos errores. Este comportamiento es el que permite evitar el problema de los datos atípicos, debido a que éstos normalmente están asociados con errores grandes. En la Figura 3.6 se muestra el comportamiento de la función de influencia del estimador Lorentziano con diferentes valores de σ: 0.5, 1 y 2. Se observa que entre menor es σ mayor es la influencia de los errores pequeños y menor la de los errores grandes; por otro lado, entre mayor es σ menor es la influencia de los errores pequeños y mayor la de los errores grandes.

Para ejemplificar el impacto de los datos atípicos, la Figura 3.7 presenta un conjunto de seis puntos (uno de ellos es un dato atípico), y tres líneas calculadas por tres diferentes

métodos de ajuste: mínimos cuadrados (LS), componentes principales (PC) [Gonzalez87] y con el estimador Lorentziano (Lorentzian). Se puede observar que el dato atípico tiene un gran impacto en la línea estimada por mínimos cuadrados y la estimada por componentes principales, pero es prácticamente descartado en el caso de la estimación con el Lorentziano. En otras palabras, la línea calculada por el estimador Lorentziano considera la mayoría de los puntos y naturalmente tiende a descartar los datos atípicos. En el Apéndice E se presenta el desarrollo de los tres métodos de ajuste utilizados en el ejemplo anterior. Los ajustes de línea por mínimos cuadrados y por el estimador Lorentziano son planteados como un problema de optimización.



(a) Mínimos cuadrados $\rho_{mc}(x) = x^2$

(b) Función de influencia $\frac{d\rho_{mc}}{dx} = 2x$

(c) Estimador Lorentziano $\rho_\sigma(x) = \log(1 + \frac{1}{2}(\frac{x}{\sigma})^2)$

(d) Función de influencia $\frac{d\rho_\sigma}{dx} = \frac{2x}{2\sigma^2 + x^2}$

Figura 3.5: Estimadores: mínimos cuadrados y Lorentziano; con sus respectivas funciones de influencia.

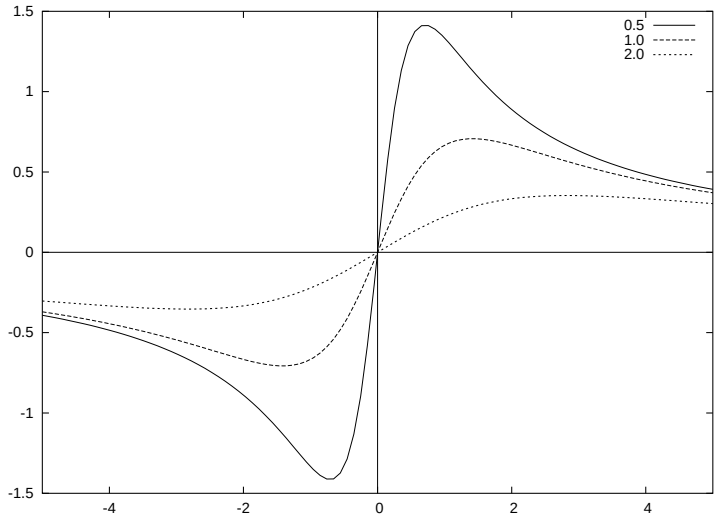


Figura 3.6: Comportamiento de la función de influencia del estimador Lorentziano con diferentes valores para σ (0.5, 1 y 2).

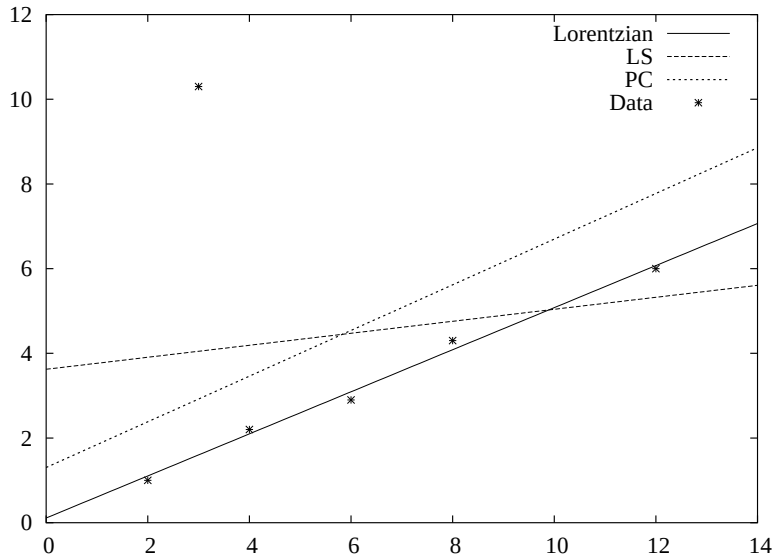


Figura 3.7: Comparación entre tres métodos de ajuste de líneas a un conjunto de puntos con un dato atípico.

La mayoría de los mejores métodos de alineamiento de rastreos del tipo puntos contra puntos [Gutmann96, Latecki04, Lu97], utilizados en el problema de la localización local, están basados en la optimización sobre estimadores de mínimos cuadrados. En este

trabajo se propone utilizar una optimización basada sobre un estimador robusto que descarte de manera natural los datos atípicos, es decir, el estimador Lorentziano.

En seguida se aborda la metodología empleada en el mapeo de puntos para lograr el alineamiento de los dos rastreos consecutivos aplicando el estimador Lorentziano.

3.4.1. Mapeo de puntos

Sean $[(x_{j,1}, y_{j,1}), ..., (x_{j,n}, y_{j,n})]$ para $n = 361$, los puntos del j-ésimo rastreo l_j, donde $(x_{j,i}, y_{j,i})$ representan las coordenadas del i-ésimo punto dado por el telémetro láser con respecto al sistema de referencia del sensor. Si el robot obtiene un rastreo l_j, después se mueve y toma otro rastreo l_{j+1}, tal como se planteó en la descripción del problema, la meta es encontrar la transformación rígida Φ que aplicada a l_{j+1} lo alinee con l_j.

Para que el rastreo l_{j+1} quede alineado con el rastreo l_j la distancia entre sus puntos debe ser mínima. Utilizando el estimador Lorentziano $\rho_\sigma(e) = \log(1 + \frac{1}{2}(\frac{e}{\sigma})^2)$, el problema de la localización local es formulado como la minimización del error total E entre dos rastreos consecutivos como sigue. Sea E la suma de los errores de distancia en x $e_{x,i}$, y en y $e_{y,i}$, para los n puntos, es decir:

$$E = \sum_{i=1}^{n}(\rho_\sigma(e_{x,i})) + \rho_\sigma(e_{y,i})) = \sum_{i=1}^{n} \rho_\sigma(x'_{j+1,i} - x_{j,\psi(i)}) + \rho_\sigma(y'_{j+1,i} - y_{j,\psi(i)}) \tag{3.5}$$

Donde el punto $(x'_{j+1,i}, y'_{j+1,i})$ representa el punto $(x_{j+1,i}, y_{j+1,i})$ después de la transformación rígida Φ, dada por una rotación θ y una traslación t_x y t_y:

$$\begin{bmatrix} x'_{j+1,i} \\ y'_{j+1,i} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{j+1,i} \\ y_{j+1,i} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3.6}$$

Sea l'_{j+1} el nuevo rastreo con su conjunto de puntos después de una transformación rígida Φ. El punto $(x_{j,\psi(i)}, y_{j,\psi(i)})$ representa el punto más cercano en l_j asociado al punto i-ésimo de l'_{j+1}. En otras palabras, el punto en l_j con la mínima distancia euclidiana al i-ésimo punto de l'_{j+1}.

Para encontrar los parámetros deseados de $\Phi = (\theta, t_x, t_y)$ que dan el mínimo valor para el error total E en la Ecuación 3.5, las siguientes condiciones deben aplicarse:

$$\begin{aligned} \frac{\partial E}{\partial \theta} &= 0 \\ \frac{\partial E}{\partial t_x} &= 0 \\ \frac{\partial E}{\partial t_y} &= 0 \end{aligned} \tag{3.7}$$

Sea $f_1 = \frac{\partial E}{\partial \theta}$, $f_2 = \frac{\partial E}{\partial t_x}$ y $f_3 = \frac{\partial E}{\partial t_y}$; el resultado final del cálculo de la localización local usando el estimador Lorentziano (Apéndice F) es un sistema de ecuaciones:

$$2A\ \delta\Phi = -2B \tag{3.8}$$

donde $A = \nabla^2$ (Hessiano), $\delta\Phi = (\delta\theta, \delta t_x, \delta t_y)^t$, y $B = \nabla$ (Jacobiano). Así, los elementos de A y de B son:

$$\begin{aligned}
a_{1,1} &= \frac{\partial f_1}{\partial \theta} = 2\sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}} \frac{\partial^2 x'}{\partial \theta^2} + \frac{\partial x'}{\partial \theta} \frac{D_{x,i} \frac{\partial x'}{\partial \theta} - 2e_{x,i}^2 \frac{\partial x'}{\partial \theta}}{D_{x,i}^2} + \frac{e_{y,i}}{D_{y,i}} \frac{\partial^2 y'}{\partial \theta^2} + \frac{\partial y'}{\partial \theta} \frac{D_{y,i} \frac{\partial y'}{\partial \theta} - 2e_{y,i}^2 \frac{\partial y'}{\partial \theta}}{D_{y,i}^2} \\
a_{1,2} &= \frac{\partial f_1}{\partial t_x} = 2\sum_{i=1}^{n} \frac{\partial x'}{\partial \theta} \frac{D_{x,i} \frac{\partial x'}{\partial t_x} - 2e_{x,i}^2 \frac{\partial x'}{\partial t_x}}{D_{x,i}^2} + \frac{\partial y'}{\partial \theta} \frac{D_{y,i} \frac{\partial y'}{\partial t_x} - 2e_{y,i}^2 \frac{\partial y'}{\partial t_x}}{D_{y,i}^2} \\
a_{1,3} &= \frac{\partial f_1}{\partial t_y} = 2\sum_{i=1}^{n} \frac{\partial x'}{\partial \theta} \frac{D_{x,i} \frac{\partial x'}{\partial t_y} - 2e_{x,i}^2 \frac{\partial x'}{\partial t_y}}{D_{x,i}^2} + \frac{\partial y'}{\partial \theta} \frac{D_{y,i} \frac{\partial y'}{\partial t_y} - 2e_{y,i}^2 \frac{\partial y'}{\partial t_y}}{D_{y,i}^2} \\
a_{2,1} &= \frac{\partial f_2}{\partial \theta} = 2\sum_{i=1}^{n} \frac{D_{x,i} \frac{\partial x'}{\partial \theta} - 2e_{x,i}^2 \frac{\partial x'}{\partial \theta}}{D_{x,i}^2} \\
a_{2,2} &= \frac{\partial f_2}{\partial t_x} = 2\sum_{i=1}^{n} \frac{D_{x,i} \frac{\partial x'}{\partial t_x} - 2e_{x,i}^2 \frac{\partial x'}{\partial t_x}}{D_{x,i}^2} \\
a_{2,3} &= \frac{\partial f_2}{\partial t_y} = 2\sum_{i=1}^{n} \frac{D_{x,i} \frac{\partial x'}{\partial t_y} - 2e_{x,i}^2 \frac{\partial x'}{\partial t_y}}{D_{x,i}^2} \\
a_{3,1} &= \frac{\partial f_3}{\partial \theta} = 2\sum_{i=1}^{n} \frac{D_{y,i} \frac{\partial y'}{\partial \theta} - 2e_{y,i}^2 \frac{\partial y'}{\partial \theta}}{D_{y,i}^2} \\
a_{3,2} &= \frac{\partial f_3}{\partial t_x} = 2\sum_{i=1}^{n} \frac{D_{y,i} \frac{\partial y'}{\partial t_x} - 2e_{y,i}^2 \frac{\partial y'}{\partial t_x}}{D_{y,i}^2} \\
a_{3,3} &= \frac{\partial f_3}{\partial t_y} = 2\sum_{i=1}^{n} \frac{D_{y,i} \frac{\partial y'}{\partial t_y} - 2e_{y,i}^2 \frac{\partial y'}{\partial t_y}}{D_{y,i}^2}
\end{aligned} \tag{3.9}$$

$$\begin{aligned}
b_1 = f_1 &= \frac{\partial E}{\partial \theta} = 2\sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}} \frac{\partial x'}{\partial \theta} + \frac{e_{y,i}}{D_{y,i}} \frac{\partial y'}{\partial \theta} \\
b_2 = f_2 &= \frac{\partial E}{\partial t_x} = 2\sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}} \frac{\partial x'}{\partial t_x} + \frac{e_{y,i}}{D_{y,i}} \frac{\partial y'}{\partial t_x} \\
b_3 = f_3 &= \frac{\partial E}{\partial t_y} = 2\sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}} \frac{\partial x'}{\partial t_y} + \frac{e_{y,i}}{D_{y,i}} \frac{\partial y'}{\partial t_y}
\end{aligned}$$

donde:

$$
\begin{array}{ll}
D_{x,i} = 2 * \sigma^2 + e_{x,i}^2 & D_{y,i} = 2 * \sigma^2 + e_{y,i}^2 \\
\frac{\partial x'}{\partial \theta} = -x\, sin(\theta) - y\, cos(\theta) & \frac{\partial y'}{\partial \theta} = x\, cos(\theta) - y\, sin(\theta) \\
\frac{\partial^2 x'}{\partial \theta^2} = -x\, cos(\theta) + y\, sin(\theta) & \frac{\partial^2 y'}{\partial \theta^2} = -x\, sin(\theta) - y\, cos(\theta) \\
\frac{\partial x'}{\partial t_x} = 1 & \frac{\partial y'}{\partial t_x} = 0 \\
\frac{\partial x'}{\partial t_y} = 0 & \frac{\partial y'}{\partial t_y} = 1
\end{array}
$$

Dado que el factor dos afecta a todos los elementos de A, y del otro lado de la igualdad a todos los elementos de B, se omite. El Apéndice F contiene el desarrollo completo para el cálculo de la localización local usando el estimador Lorentziano. En seguida se presenta el proceso de búsqueda para encontrar la solución al sistema de la Ecuación 3.8, y con ello los parámetros de Φ.

3.4.2. El proceso de optimización NLM

Para encontrar el conjunto de parámetros Φ, que minimizan el error, se aplica el método de Newton-Levenberg-Marquardt (NLM) [Press86]. Este método es una mezcla basada en los métodos de descenso de gradiente y Newton. El algoritmo es el siguiente:

1. Calcular el error total, $E(\Phi)$ (Equación 3.5).
2. Establecer un valor pequeño para λ, por ejemplo λ =0.001.
3. Resolver el sistema de ecuaciones $(A + \lambda I)\delta\Phi = B$, y calcular $E(\Phi + \delta\Phi)$. I es la matriz identidad.
4. Si $E(\Phi+\delta\Phi) > E(\Phi)$, incrementar λ por un factor de 10, y regresar al paso anterior. Si λ se hace muy grande, significa que no existe forma para alcanzar una mejor solución de Φ.
5. Si $E(\Phi + \delta\Phi) < E(\Phi)$, decrementar λ por un factor de 10, remplazar Φ por $\Phi + \delta\Phi$, y regresar al paso 3. Si $\mid E(\Phi + \delta\Phi) - E(\Phi) \mid$ es suficientemente pequeño, entonces Φ es la solución.

Cuando $\lambda = 0$, el método se comporta enteramente como un Newton, y cuando λ tiende a infinito, $\delta\Phi$ convierte al método en descenso de gradiente y el tamaño de paso $\delta\Phi$ tiende a cero.

Sin embargo, la asociación de puntos contra puntos no es lo suficientemente buena en todos los casos, especialmente en ambientes que tienen una estructura regular. A continuación se presenta una mejora al método de asociación de datos para superar éste problema.

3.4.3. Mejora de asociación de datos

En trabajos previos [Gutmann96, Latecki04] se toma ventaja de los ambientes con estructura regular modelando el ambiente con líneas y puntos, cuando les es posible. Para entender el problema con los métodos de la asociación de puntos contra puntos, la Figura 3.8 muestra un ejemplo de dos rastreos del telémetro láser en una ambiente interior.

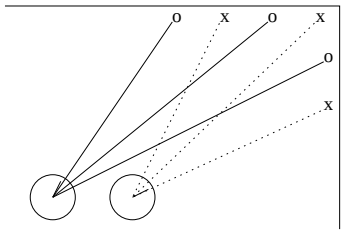


Figura 3.8: Dos rastreos láser en ambientes interiores pueden presentar problemas en la asociación de datos con métodos de puntos contra puntos.

El problema radica en que difícilmente a un punto del rastreo l_{j+1} le corresponde exactamente otro punto en el rastreo previo l_j. La razón es que los puntos sensados representan diferentes partes del ambiente, por ejemplo en las áreas cercanas a las esquinas. Así pues, no es posible encontrar la transformación rígida Φ que alinee completamente los dos rastreos consecutivos.

En nuestro caso, la solución al problema de asociar puntos contra puntos, también es tomar ventaja, cuando sea posible, de la estructura del ambiente. Así, en lugar de utilizar los puntos sensados directamente por el telémetro láser, se utilizan puntos calculados $(x_{j,\psi(i)}, y_{j,\psi(i)})$ de la proyección ortogonal de cada punto $(x_{j+1,i}, y_{j+1,i})$ sobre la línea más cercana contenida en l_j.

Para mejorar el alineamiento de los rastreos, se elige entre un punto sensado y uno calculado mediante la siguiente heurística: se ajusta una línea recta en la vecindad del punto más cercano de l_{j+1} sobre l_j, si el error de la línea ajustada es mayor de cierto umbral, es decir, no es lo suficientemente bueno, se elige el punto sensado, sino, el punto de l_{j+1} es

proyectado sobre la línea ajustada, finalmente el punto sobre la línea ajustada, o en l_j, con la menor distancia euclidiana al punto en l_{j+1} será el que se considere para el alineamiento.

La metodología y heurística para ajustar la línea recta más cercana al punto de l_{j+1} en l_j, así como el cálculo de la proyección ortogonal de los puntos sobre la línea ajustada están contenidos en el Apéndice E Sección E.3.

Al mismo tiempo que la localización local del robot se realilza, mediante el alineamiento de dos rastreos consecutivos, se construye el mapa 2D. En la siguiente sección se presenta más a detalle el procedimiento para la construcción del mapa 2D.

3.5. Construcción del mapa 2D

El procedimiento seguido para la construcción del mapa 2D puede ser resumido mediante la siguiente secuencia de pasos:

1. El primer rastreo l_j realizado por el robot móvil sobre el ambiente es considerado como sistema de referencia global.
2. Los puntos del rastreo l_j son adicionados al mapa 2D.
3. Si no existe un rastreo consecutivo l_{j+1} el procedimiento termina aquí.
4. Se calcula la transformación rígida Φ que alinee el rastreo l_{j+1} con el rastreo l_j, aplicando el proceso de búsqueda NLM (ver Sección 3.4.2).
5. Se proyecta l_{j+1} con Φ (rotación y traslación) para obtener l'_{j+1}.
6. Hacer $l_j = l'_{j+1}$ y regresar al paso 2.

En otras palabras, la construcción del mapa 2D se logra tomando el primer rastreo como sistema de referencia global, y adicionando sus puntos como los primeros del mapa 2D, alienar el segundo rastreo con el primero y adicionar sus puntos al mapa 2D actual, alinear el tercer rastreo con el segundo y adicionar sus puntos con el mapa 2D actual, y así susecivamente hasta que no exista un rastreo siguiente.

Los experimentos realizados, mejorando y no la asociación de datos, así como los correspondientes mapas 2D obtenidos, se presentarán en el Capítulo 5. En seguida se presentan las conclusiones derivadas de éste capítulo.

3.6. Conclusiones

En este capítulo, se describió un método robusto para la localización local y construcción de mapas 2D, utilizando el estimador Lorentziano y el método de optimización no líneal Newton-Levenberg-Marquardt. El estimador Lorentziano es suficientemente robusto con el ruido (datos atípicos) de los rastreos adquiridos por el telémetro láser. Dicho estimador, de forma natural, alinea la mayoría de los puntos de un rastreo láser sobre un segundo rastreo, sin importar que tan grandes sean los errores provocados por los datos atípicos.

Una vez resuelto el problema de la localización local y construcción de mapas 2D, se tiene la capacidad de realizar reconstrucciones 3D cuando el robot se desplaza o gira, ya que se sabrá con una buena precisión la localización (posición y orientación) del robot cada vez que éste se mueve.

Capítulo 4

Reconstrucción 3D

La obtención de modelos 3D a partir de ambientes reales, es uno de los temas de investigación que ha tomado un gran auge en los últimos años. Tradicionalmente el modelado tridimensional se centraba en objetos pequeños o de tamaño mediano. En la actualidad, existe un número modesto de grupos de investigación que se han centrado en el problema de reconstruir tridimensionalmente ambientes interiores o exteriores, de casi cualquier tamaño.

4.1. Introducción

Rusinkiewicz clasifica la adquisición de datos de rango para la obtención de modelos 3D, en tres categorías principales [Rusinkiewicz01]: de contacto, transmisivas y reflectivas. La Figura 4.1 presenta una taxonomía de los métodos de adquisición de datos de rango.

Los métodos de contacto tocan la superficie de los objetos con una sonda y registran la posición. Por ejemplo, las máquinas de medición de coordenadas (CMMs, del Inglés *Coordinate Measuring Machines*), pueden ser muy precisas, pero son lentas, y pueden ser manipuladas torpemente, usualmente requieren de un operador humano, y necesariamente tienen que hacer contacto con el objeto, lo cual puede no desearse con objetos frágiles.

Los métodos transmisivos proyectan ondas de energía sobre un objeto, y registran la energía trasmitida. Por ejemplo, la tomografía computarizada (CT, del Inglés *Computer Tomography*) proyecta rayos-x sobre un objeto, y mide la cantidad de radiación que pasa a través de él. El resultado es un modelo volumétrico de alta definición del objeto. Los

métodos transmisivos pueden capturar detalles no visibles por fuera del objeto, sin embargo, estos métodos son caros, sensibles a la variedad de la densidad del material, y pueden ser potencialmente peligrosos para el ser humano.

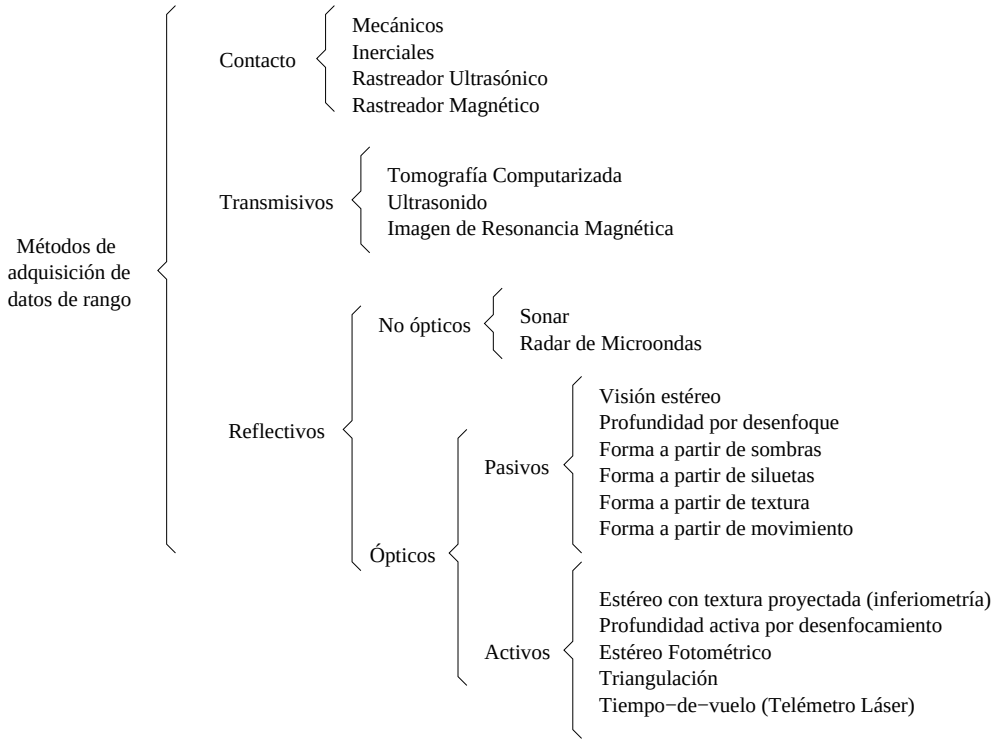


Figura 4.1: Clasificación de los métodos para la adquisición de datos de rango según Rusinkiewicz [Rusinkiewicz01].

Los métodos reflectivos se clasifican a su vez en: no ópticos y ópticos. Los métodos no ópticos incluyen al sonar y al radar de microondas, los cuales miden la distancia a un objeto mediante la emisión de un pulso de sonido, o microonda de energía, en dirección del objeto, y registran el tiempo que la emisión tarda en rebotar del objeto al sensor. Los sonares típicamente no son caros, pero no son muy precisos ni rápidos. Los radares de microondas comúnmente son utilizados para sensar distancias grandes.

Los métodos ópticos son de dos tipos: pasivos y activos. Los métodos pasivos no interactúan con el objeto a rastrear y generalmente requieren de poco hardware especial. Por otro lado, los métodos activos proyectan luz sobre el objeto y requieren de hardware especializado.

Algunas de las ventajas de los métodos ópticos sobre los métodos anteriores son las siguientes: (1) no hacen contacto físico con la superficie de los objetos a reconstruir, (2) son seguros, no emiten radiación nociva al ser humano, (3) usualmente no son tan caros, en comparación a los métodos transmisivos, y (4) usualmente son rápidos.

Por otro lado, algunas de sus desventajas son: (1) son sensibles a objetos transparentes, (2) se ven afectados por la especularidad y la no reflexión, (3) en algunos casos la textura de los objetos es necesariamente requerida.

Los métodos de adquisición de datos de rango más comúnmente encontrados, cuando se utilizan robots móviles en la adquisición, son los reflectivos. Típicamente un robot móvil cuenta con sonares (método reflectivo no óptico), o con sensores infrarrojos (método reflectivo óptico activo por triangulación), sin embargo, estos dos tipos de sensores difícilmente pueden ser adaptados para la adquisición de datos de rango 3D. Las dos metodologías mayormente utilizadas en robótica móvil para adquirir datos de rango 3D son: la fotogrametría y visión estéreo, así como los rastreos láser. Por ejemplo, Iocchi [Iocchi00] realiza reconstrucción 3D de ambientes interiores con visión estéreo (método reflectivo óptico pasivo). Otros trabajos, por ejemplo: Thrun [Hähnel03, Liu01, Thrun00, Thrun03], Hartmut [Nüchter02, Pervölz04, Surmann01, Surmann03], Sequeira [Dias03, Klein00, Sequeira95], y otros más como Biber [Biber04], Fangi [Fangi01], Kaess [Kaess01], utilizan telémetros láser para la adquisición de datos de rango 3D (método reflectivo óptico activo por tiempo-de-vuelo); algunos de estos trabajos también combinan imágenes de cámaras CCD con la información de rango para obtener una apariencia fotorealista.

La tecnología de rastreo láser (telémetro láser) tiene muchas ventajas respecto a la fotogrametría y visión estéreo (cámaras CCD). En [Pervölz04] y [Rusinkiewicz01] se mencionan algunas de las principales ventajas y desventajas entre estos dos métodos de adquisición de datos de rango. En la Tabla 4.1 se resumen las ventajas de la tecnología de rastreo láser respecto a la fotogrametría y visión estéreo. La Tabla 4.2 presenta el caso contrario, las ventajas de la fotogrametría y visión estéreo respecto a la tecnología de rastreo láser.

En este trabajo se utiliza un telémetro láser para la adquisición de datos de rango, es decir, un método de adquisición de datos de rango óptico activo del tipo tiempo-de-vuelo. Además para realizar la adquisición se hace uso del robot móvil descrito anteriormente en el Capítulo 2.

Este capítulo presenta una breve descripción del estado del arte referente a las

Tabla 4.1: Ventajas de la tecnología de rastreo láser (telémetro láser) sobre la fotogrametría y visión estéreo.

Tecnología	Iluminación	Medición 3D	Fiabilidad	Alcance
Telémetro Láser	Independiente	Directamente por el tiempo de vuelo	Altamente confiable	De 8-80 mts
Fotogrametría y Visión Estéreo	Dependiente	Extrae 3D a partir de correspondencia de fotografías	La textura es requerida	Limitada

Tabla 4.2: Ventajas de la fotogrametría y visión estéreo (cámaras CCD) sobre la tecnología de rastreo láser.

Tecnología	Tamaño	Resolución 3D	Costo	Analogía con la visión humana
Telémetro Láser	Dispositivos grandes y pesados	Resolución espacial limitada	Alto	No
Fotogrametría y Visión Estéreo	Cámaras pequeñas	Fotografías de alta resolución	Bajo	Si

plataformas de adquisición de datos de rango, específicamente cuando se utilizan robots móviles para realizar la reconstrucción 3D de ambientes interiores. Se presenta también nuestra plataforma de adquisición, así como la generación del modelo 3D del ambiente y la navegación a través de él utilizando OpenGL. Al final del capítulo se presentan las conclusiones. Los experimentos correspondientes se presentaran en el Capítulo 5.

4.2. Plataforma de adquisición de datos

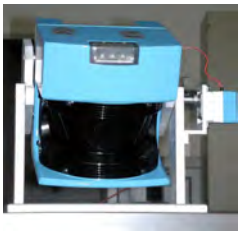
Por plataforma de adquisición de datos se hace referencia al tipo y configuración de *hardware* utilizado para la adquisición. Por ejemplo, en [Hähnel03, Liu01, Thrun03] utilizan un robot móvil modelo *B21R*, o *Pioneer*, equipado con dos telémetros láser 2D, uno girado 90° con respecto del otro (ver Figura 4.2 (a) y (b)). Así, uno es utilizado para localizar al robot, y el otro recopila información sobre la estructura 3D del ambiente mientras el robot se desplaza. Con tal configuración, la exactitud de los datos depende en gran parte de la localización del robot; errores de localización repercuten directamente en errores de adquisición de datos 3D.

(a) B21R

(b) *Pioneer*

Figura 4.2: Robots móviles equipados con dos telémetros láser, uno girado 90° con respecto del otro [Hähnel03, Liu01, Thrun03].

Algunos otros trabajos utilizan plataformas basadas en la extensión de un telémetro láser 2D por medio de un servomotor [Nüchter02, Pervölz04, Surmann01, Surmann03]. Esta configuración comúnmente es conocida como telémetro láser 3D (ver Figura 4.3(a)), algunos robots móviles han sido equipados con tales telémetros láser 3D (ver Figuras 4.3(b) y (c)). Con esta configuración es posible realizar un rastreo 3D del ambiente frente al robot sin necesidad de desplazarlo. Sin embargo, si se desea rastrear alrededor del robot será necesario girarlo.



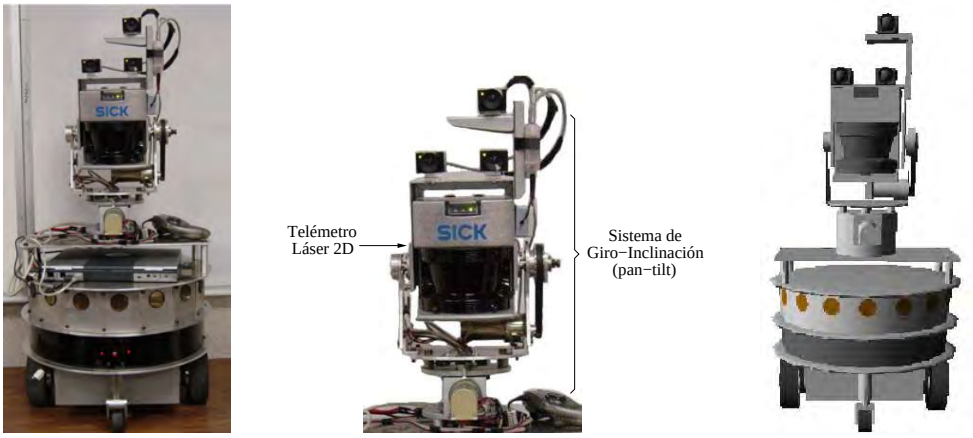
(a) Telémetro Láser 3D

(b) Kurt3D

(c) Ariadne

Figura 4.3: Los robots móviles Kurt3D [Pervölz04] y Ariadne [Surmann01], se equiparon con la extensión del telémetro láser 2D, es decir con un telémetro láser 3D.

La plataforma de adquisición de datos de rango 3D utilizada en este trabajo, nuestro robot móvil, se muestra en la Figura 4.4(a). Dicho robot ya fue descrito en el Capítulo 2. Está dotado de diferentes elementos que le permiten obtener información sobre la estructura 3D de su ambiente.



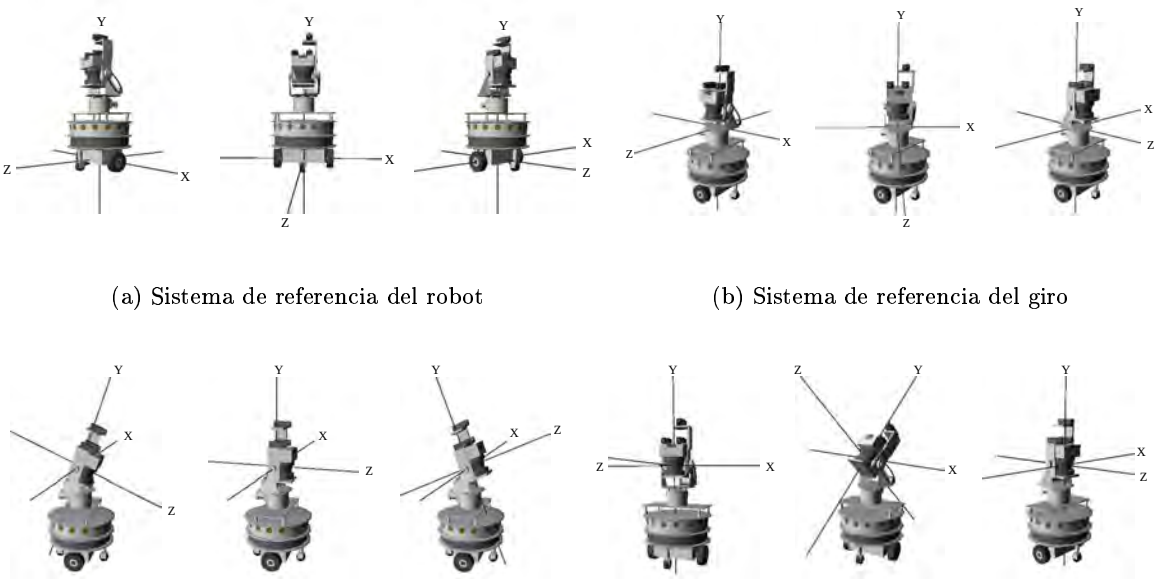
(a) Robot móvil (b) Telémetro láser 3D (c) Modelo 3D

Figura 4.4: Nuestro robot móvil equipado con un sistema de giro inclinación y un telémetro láser 3D.

El telémetro láser 2D, mostrado en la Sección 2.6.3, es el encargado de registrar la geometría del ambiente. Está montado sobre el sistema de giro-inclinación, por lo tanto se trata de un telémetro láser 3D, que a diferencia de las plataformas de adquisición anteriores, permite realizar rastreos 3D completamente alrededor del robot, es decir, cubre 360° sin la necesidad de desplazar o rotar el robot móvil.

Dadas las características de nuestro robot es posible identificar 4 sistemas de referencia: del robot, para el giro, la inclinación y el sensor láser. La Figura 4.5 muestra estos sistemas de referencia. El sistema de referencia del robot está relacionado con la superficie del piso, es decir, del plano XZ en la cual se desplaza, el origen se encuentra en el centro del robot sobre el plano XZ (ver Figura 4.5(a)). El sistema de referencia de giro también tiene su origen en el centro del robot, pero a la altura del eje de giro sobre la base del robot (ver Figura 4.5 (b)). El sistema de referencia de inclinación, al igual que los anteriores, su origen

esta situado en el centro del robot, pero sobre el sistema de giro en el eje de inclinación (ver Figura 4.5 (c)). En lo que respecta al sistema del sensor láser, se encuentra dentro del telémetro, un poco desfasado del origen del sistema de inclinación. En sí, el origen del sistema es la posición del propio sensor láser (ver Figura 4.5(c)).



(a) Sistema de referencia del robot

(b) Sistema de referencia del giro

(c) Sistema de referencia de inclinación

(d) Sistema de referencia del sensor láser

Figura 4.5: Sistemas de referencia: (a) robot móvil, (b) y (c) del sistema giro-inclinación, y (d) del sensor del telémetro láser.

Para la reconstrucción del modelo virtual 3D, a partir del entorno real, es necesario tener un sistema de referencia global. En nuestro caso, éste queda definido por el sistema de referencia de inclinación respecto del primer rastreo; a partir de un primer rastreo, todos los demás se mapean con respecto al sistema de referencia global.

Definidos los distintos sistemas de referencia, a continuación se presenta la obtención de los datos de rango 3D para la construcción del modelo 3D, a partir de escenas 3D sensadas por el robot a través del telémetro láser.

4.3. Obtención de los datos de rango 3D

Durante la adquisición de datos de rango, el telémetro láser toma una "rebanada" (rastreo) del ambiente, en un ángulo de inclinación ω, y un ángulo de giro β del sistema de giro-inclinación, en una localización particular del robot. La información proporcionada por el telémetro láser, 361 datos en coordenadas polares para un rastreo, corresponden al contorno de los objetos inmersos en la escena (ver Figura 4.6). Reconstruir dichos objetos representa transformar las 361 lecturas en puntos en 3D.

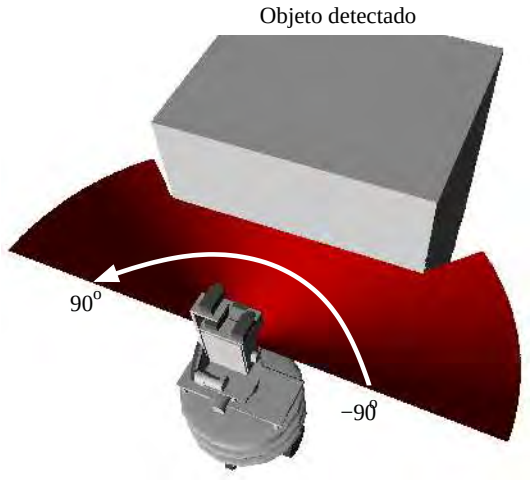


Figura 4.6: Rastreo ("rebanada") del telémetro láser de un plano para detectar objetos en el ambiente (desde $-90°$ hasta $90°$).

Para transformar los datos adquiridos por el láser en puntos 3D, se utilizan matrices de transformaciones en tres dimensiones [Román94], con coordenadas homogéneas [Barrientos97]. Utilizar coordenadas homogéneas simplifica el cálculo de las transformaciones compuestas de rotación y traslación mediante simples premultipicaciones de matrices. Así, el problema se presenta como la transformación de un punto P_s, en el sistema de referencia del sensor láser, en un punto P_r del sistema de referencia del robot, pasando por los sistemas de inclinación y de giro, es decir:

$$P_r = [Mt_r]\,[Mr_r]\,[Mr_g]\,[Mr_i]\,[Mt_s]\,P_s \tag{4.1}$$

Donde un punto P_s corresponde a una de las 361 lecturas (d, α) reportadas por el telémetro láser, en coordenadas rectangulares sobre el plano XZ, es decir:

$$P_s = \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} = \begin{bmatrix} dsin(\alpha) \\ 0 \\ dcos(\alpha) \\ 1 \end{bmatrix} \tag{4.2}$$

La matriz $[Mt_s]$ es una matriz de traslación del sistema de referencia del sensor láser al sistema de referencia de inclinación. La traslación se da sobre los ejes X y Y, por lo que la matriz es la siguiente:

$$Mt_s = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

La matriz $[Mr_i]$ es una matriz de rotación, en el sistema de referencia de inclinación, sobre el eje X a un ángulo ω; los valores de la matriz son los siguientes:

$$Mr_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\omega) & -sin(\omega) & 0 \\ 0 & sin(\omega) & cos(\omega) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

La matriz $[Mr_g]$ es una matriz de rotación, en el sistema de referencia del giro, sobre el eje Y a un ángulo β; los valores de la matriz son los siguientes:

$$Mr_g = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

La matriz $[Mr_r]$ es una matriz de rotación, en el sistema de referencia del robot, sobre el eje Y a un ángulo θ. El ángulo de rotación θ se obtiene de resolver el problema de la localización local mediante dos rastreos consecutivos (ver Capítulo 3). La matriz correspondiente es igual a la matriz de la ecuación 4.5, pero aplicada al ángulo θ.

La matriz $[Mt_r]$ es una matriz de traslación, en el sistema de referencia del robot, sobre el plano XZ. La traslación correspondiente, al igual que el ángulo θ, también es

calculada al resolver el problema de la localización del robot. Los valores de la matriz son los siguientes:

$$Mt_r = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

La reconstrucción 3D se basa en la adquisición de escenas 3D. Una *escena 3D* está conformada por 36100 puntos (P_r) correspondientes a 100 rastreos realizados desde la posición inicial de inclinación, hasta la posición final del sistema de inclinación. La relación de vecindad que guardan éstos puntos entre rastreos consecutivos es invariante al ángulo de giro y a la localización del robot. En la Figura 4.7 se muestran los ocho vecinos del punto $p_{i,j}$ correspondientes al propio rastreo l_j, a un rastreo previo l_{j-1}, y a un rastreo posterior l_{j+1}. La vecindad del punto $p_{i,j}$ se mantienen sin cambio sin importar el ángulo de giro del *pan-tilt* y la localización del robot.

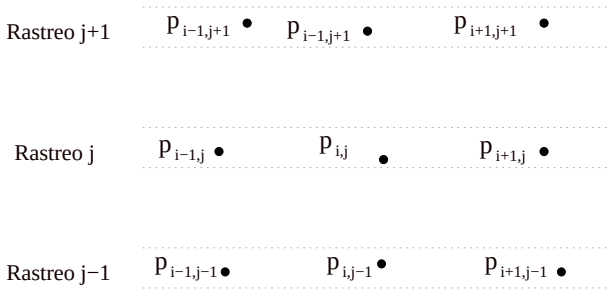


Figura 4.7: Puntos vecinos correspondientes al punto $p_{i,j}$.

Con los puntos P_r correspondientes a una escena 3D, se genera una malla poligonal texturizada para tener una mejor apreciación de los objetos inmersos en el ambiente. La heurística para la generar la malla, así como la texturización de la misma, se presentan en seguida.

4.4. Generación y texturización de mallas

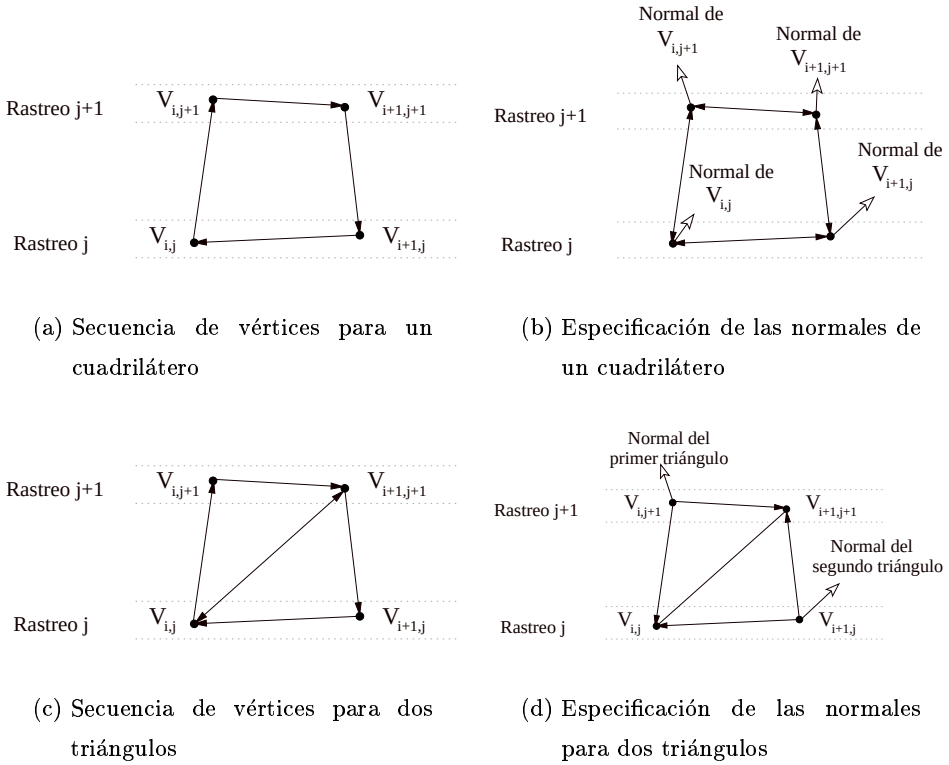
En nuestro caso, una malla poligonal puede estar formada por dos tipos de polígonos: cuadriláteros y triángulos. La elección de éstos dos tipos de poligonos se debe a que el hardware de aceleración está optimizado para el trazo de triángulos o cuadriláteros. En OpenGL los puntos que conforman un polígono son referidos como vértices, por lo que de aquí en adelante los puntos P_r pertenecientes a una escena 3D serán referidos de tal forma.

La heurística para la generación de la malla poligonal está basada en la distancia euclidiana entre los vértices que conforman un polígono, con ello se ignora a la mayoría de los datos atípicos. En una malla poligonal formada por cuadriláteros o triángulos, para que uno de éstos polígonos sea válido, la distancia del primer vértice al resto de los vértices, no debe exceder cierto umbral Δ. Trazar polígonos sin ésta restricción generaría una malla demasiado irregular, con polígonos que se extenderían hasta la máxima distancia reportada por el telémetro láser. También podría ser que huecos en el ambiente se trazaran como polígonos sólidos.

Aprovechando que la vecindad entre los puntos de una escena 3D es invariante, para generar una malla poligonal de cuadriláteros o triángulos, se toman dos vértices consecutivos de un rastreo y otros dos del rastreo siguiente. Es decir, dos vértices $v_{i,j}$ y $v_{i+1,j}$, donde i representa el número de vértice y j el número de rastreo, y otros dos vértices $v_{i,j+1}$, $v_{i+1,j+1}$ del siguiente rastreo. Como el orden en el que se especifican los vértices que forman un polígono es muy importante en OpenGL (ver Sección C.2.3), la secuencia de vértices para definir un cuadrilátero de la malla se da en sentido de las manecillas del reloj de la siguiente forma: $v_{i,j}$, $v_{i,j+1}$, $v_{i+1,j+1}$ y $v_{i+1,j}$ (ver Figura 4.8(a)).

Para el caso de una malla poligonal de triángulos, se construyen dos triángulos con los cuatro vértices, el primer triángulo está definido por la siguiente secuencia de vértices: $v_{i,j}$, $v_{i,j+1}$ y $v_{i+1,j+1}$; el segundo triángulo está dado por la siguiente secuencia de vértices: $v_{i+1,j+1}$, $v_{i+1,j}$ y $v_{i,j}$ (ver Figura 4.8(c)).

Para apreciar realmente la forma de los objetos inmersos en el ambiente sensado es necesario texturizar la escena 3D. Esto se logra adicionando efectos de luz, por lo que una fuente de luz ambiental y difusa es adicionada a la escena 3D. Sin embargo, la fuente de luz por si sola no es suficiente, se debe especificar tanto el color como las propiedades reflectivas de los polígonos, además se debe especificar en que dirección han de reflejar éstos la luz, es decir, se debe especificar una normal para el polígono.



(a) Secuencia de vértices para un cuadrilátero

(b) Especificación de las normales de un cuadrilátero

(c) Secuencia de vértices para dos triángulos

(d) Especificación de las normales para dos triángulos

Figura 4.8: Construcción de la malla poligonal con cuadriláteros o triángulos y definición de las normales correspondientes.

Cuando se trata de una malla poligonal de triángulos, se especifica una normal para cada triángulo (ver Figura 4.8(d)). Tomando un vértice del triángulo como referencia, se obtienen dos vectores de dicho vértice a cada uno de los vértices restantes, y se realiza el producto cruz entre estos dos vectores para calcular la normal de la superficie del triángulo, con la cual queda definida la dirección en que habrá de reflejarse la luz para ese triángulo en particular.

En una malla poligonal de cuadriláteros se especifica una normal para cada vértice (ver Figura 4.8 (b)), con el propósito de dar un efecto de suavizado de color entre las distintas profundidades de los vértices que forman el cuadrilátero. Para calcular cada una de las normales correspondientes a cada uno de los vértices, además de tomar el vértice en cuestión, se toman los siguientes dos vértices para calcular los dos vectores cuyo producto cruz habrá de definir la normal correspondiente para cada vértice del cuadrilátero.

El conjunto de escenas 3D, presentadas como mallas poligonales texturizadas, forman el modelo 3D reconstruido del ambiente, el cual permite la navegación a través de él. En seguida se presenta cómo es que se logra tal efecto.

4.5. Navegación a través del modelo 3D reconstruido

La navegación a través del modelo 3D reconstruido se realiza mediante la manipulación del espacio 3D por transformación de coordenadas (ver Sección C.4). Se simulan los sistemas de referencia del robot, de giro y de inclinación. La posición inicial dentro del modelo queda definida por el sistema de referencia global, es decir la localización del robot en el primer rastreo.

Para simular el desplazamiento de robot, basta con aplicar la matriz de traslación de la Ecuación 4.6, con un cierto valor tanto para t_x como para t_z. De manera similar, la rotación del robot y el giro del sistema de giro-inclinación, se simulan mediante la matriz de rotación de la Ecuación 4.5, aplicada a un ángulo θ o β, de rotación o giro respectivamente. La inclinación del sistema de giro-inclinación también es simulada de igual forma, pero con la matriz de rotación de la Ecuación 4.4. El efecto final es que se visualiza en el monitor una imagen (*frame*) similar a la que sería capturada por una cámara monocromática infraroja montada sobre el robot real en el ambiente real.

El resultado de utilizar la heurística y texturización de la malla poligonal, ya sea de cuadriláteros o triángulos se presenta en el Capítulo 5.

4.6. Conclusiones

Este capítulo describe el uso del robot móvil como la plataforma de adquisición de datos utilizada en la obtención de escenas 3D, para la generación del modelo 3D como una malla poligonal texturizada, teniendo así una mejor apreciación de las formas inmersas en el ambiente reconstruido.

El sistema de giro-inclinación permite realizar la adquisición de datos de rango con el telémetro láser, de tal forma, que sin necesidad de girar el robot, es posible reconstruir todos los objetos que sean alcanzados por el sensor láser alrededor del robot. La capacidad de navegación del robot móvil y la solución del problema de localización permitirá sensar y reconstruir la mayor parte del ambiente.

Representar los datos de rango 3D en coordenadas homogéneas, así como utilizar matrices de transformación para relacionar los diferentes sistemas del robot, facilitan el cálculo del conjunto de puntos 3D (vértices) que conforman una escena 3D.

OpenGL permite adicionar luz, color y material, a los polígonos que representan una escena 3D a través de una malla triángulos o cuadriláteros que reflejan la luz en cierta dirección, con ello se hace posible distinguir visualmente los objetos inmersos en las escenas 3D reconstruidas. Además, OpenGL facilita las operaciones de navegación a través del modelo 3D reconstruido mediante la aplicación de las mismas matrices de transformación utilizadas en la reconstrucción.

Capítulo 5

Experimentos

El propósito de este capítulo es mostrar los experimentos y resultados obtenidos al aplicar las metodologías descritas en los dos capítulos anteriores: Localización local y construcción de mapas 2D, y Reconstrucción 3D. Utilizando para ello nuestro robot móvil equipado con un telémetro láser, el cual fue descrito en el capítulo 2.

5.1. Introducción

Como ya se ha mencionado, el sensor principal en la adquisición de datos para el presente trabajo es el telémetro láser (ver Sección 2.6.3). Sin embargo, en el problema de la localización local también se cuenta con otro sensor importante: el odómetro (ver Sección 2.6.1). Con la lectura odométrica se tiene una estimación inicial de la localización del robot una vez que realiza un movimiento relativamente pequeño.

Los experimentos se realizaron en los laboratorios de Sistemas Computacionales (LSC) y de Instrumentación y Control (LIC), ambos localizados en el edificio de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo. Además, también se utilizó el simulador del robot hecho por el Dr. Leonardo Romero Muñoz [Romero01], con el objetivo de comparar estadísticamente, con ruido simulado, los métodos de mínimos cuadrados y el estimador Lorentziano.

En seguida se muestran los resultados de los experimentos realizados, dividiéndolos en localización local y construcción de mapas 2D, así como en reconstrucción 3D.

5.2. Localización local

El método empleado para resolver el problema de la localización local y construcción de mapas 2D fue probado sobre un ambiente simulado y uno real. En la prueba sobre el simulador del robot móvil, se construyó un ambiente con muros verticales y horizontales (ambiente estructurado ortogonal). Se simularon datos atípicos con una distribución uniforme con 2 % y 20 % de ruido.

La Tabla 5.1 presenta una comparación, entre los métodos de Mínimos Cuadrados y el Estimador Lorentziano aplicados a la solución del problema. Las columnas $\overline{e}_\theta$, s_{e_θ} y $e_{\theta max}$, se refieren al promedio de las diferencias entre el ángulo correcto y el estimado, la desviación estándar de esos errores y el error máximo, respectivamente. De una manera similar las columnas $\overline{e}_t$, s_{e_t} y $e_{t\,max}$ representan el promedio de las diferencias entre la traslación correcta y la estimada, la desviación estándar de los errores y el error máximo de traslación. Tal como se puede apreciar en los valores presentados, en todos los casos, el alineamiento con el Estimador Lorentziano fue mejor que el alineamiento por Mínimos Cuadrados.

Tabla 5.1: Resultados utilizando el simulador con 2 % y 20 % de datos atípicos.

Método	2 % ruido					
	$\overline{e}_\theta$	s_{e_θ}	$e_{\theta max}$	$\overline{e}_t$	s_{e_t}	$e_{t\,max}$
Mínimos cuadrados	8.402	7.568	38.61	27.13	22.43	110.367
Lorentziano	0.346	0.706	3.357	0.609	1.054	5.4759

	20 % ruido					
	$\overline{e}_\theta$	s_{e_θ}	$e_{\theta max}$	$\overline{e}_t$	s_{e_t}	$e_{t\,max}$
Mínimos Cuadrados	5.093	5.076	24.66	10.69	11.14	109.7
Lorentziano	1.353	2.456	10.76	1.772	3.333	18.52

En lo que respecta a la prueba sobre un ambiente real, la Figura 5.1 presenta parte del del Laboratorio de Sistemas Computacionales, nuestro ambiente real. Como se puede apreciar, existe una gran cantidad de mesas y sillas negras, además de puertas de cristal. Los objetos inmersos en la escena, así como las propiedades reflectivas de los mismos, dan como resultado un ambiente muy ruidoso, es decir, con una gran cantidad de datos atípicos que no siguen alguna distribución en particular. Tales ambientes no son fáciles de simular y representan todo un reto para la localización local y construcción de mapas 2D.

Figura 5.1: El robot móvil en el ambiente real: Laboratorio de Sistemas Computacionales.

La toma de muestras (rastreos) se realizó haciendo desplazamientos hacia adelante o hacia atrás de $50cm$ y rotaciones a la izquierda y a la derecha de $25°$, para un total de 210 movimientos. La Figura 5.2 presenta el plano arquitectónico del ambos laboratorios (LSC y LIC) donde se realizó la toma de muestras. Además de los muros, los únicos objetos que aparecen en dicho plano son puertas, algunas de ellas (las más grandes) son de cristal con marcos de metal, las sillas y las mesas no aparecen. Así, el perímetro de los mapas 2D obtenidos, ya sea mediante la utilización únicamente del odómetro, con el método de Mínimos Cuadrados o con el Estimador Lorentziano, deben ser lo mayormente semejantes al plano arquitectónico.

La Figura 5.3 presenta las lecturas 0, 23, 46, 69, 92, 112, 162, 182 y 203 del telémetro láser, así como el mapa 2D correspondiente, utilizando únicamente el odómetro para localizar el robot y construir el mapa 2D. Es evidente que las irregularidades en el piso afectan directamente las lecturas odométricas, lo que ocasiona que el robot no se localice de manera correcta, y por consiguiente el mapa 2D obtenido finalmente, presentado en la Figura 5.4, no sea útil ya que ni siquiera es semejante al plano arquitectónico mostrado en la Figura 5.2.

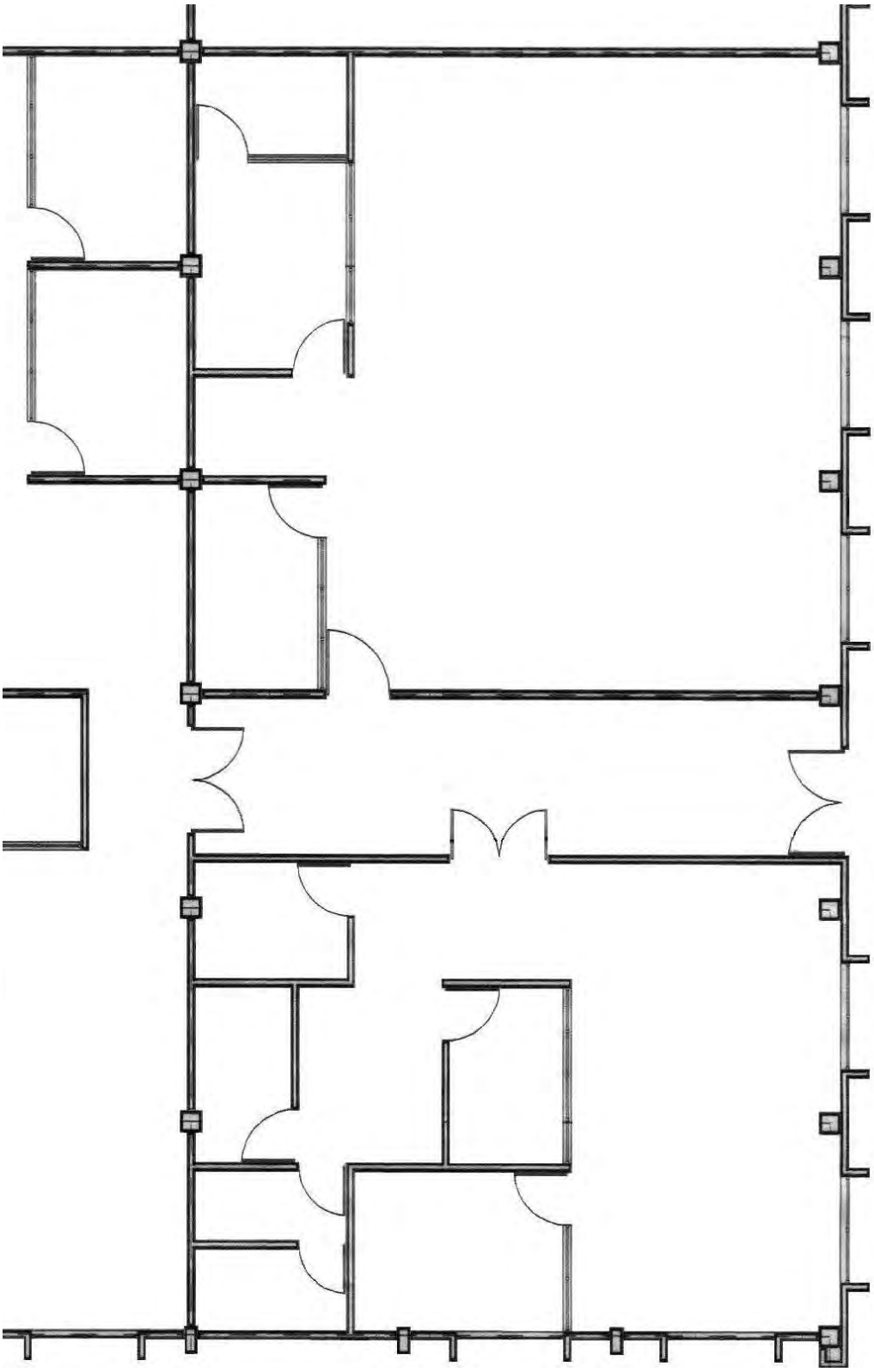
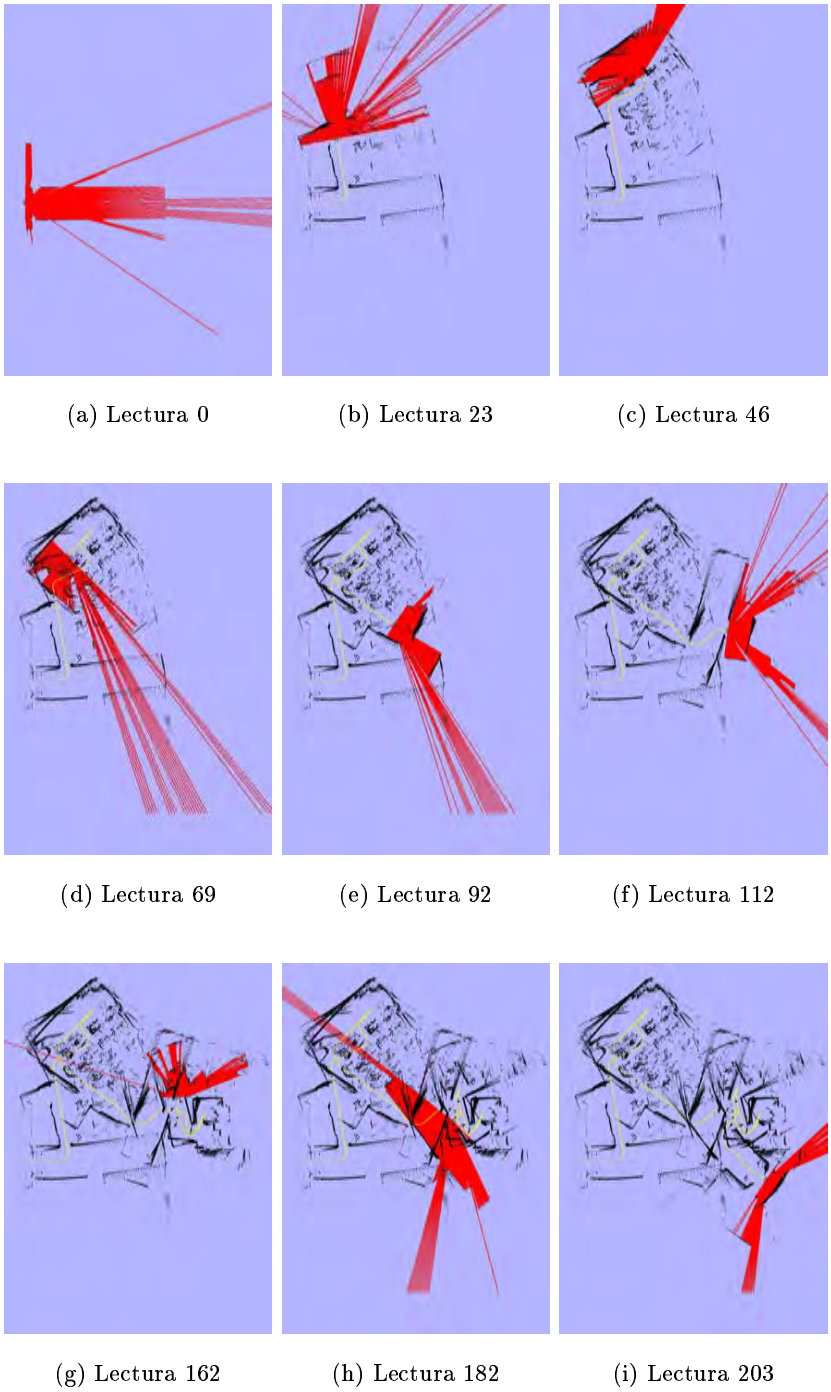


Figura 5.2: Plano arquitectónico de LSC y LIC sin mesas ni sillas.



(a) Lectura 0 (b) Lectura 23 (c) Lectura 46

(d) Lectura 69 (e) Lectura 92 (f) Lectura 112

(g) Lectura 162 (h) Lectura 182 (i) Lectura 203

Figura 5.3: Lecturas del telémetro láser y el mapa 2D correspondiente utilizando solo las lecturas del odómetro.

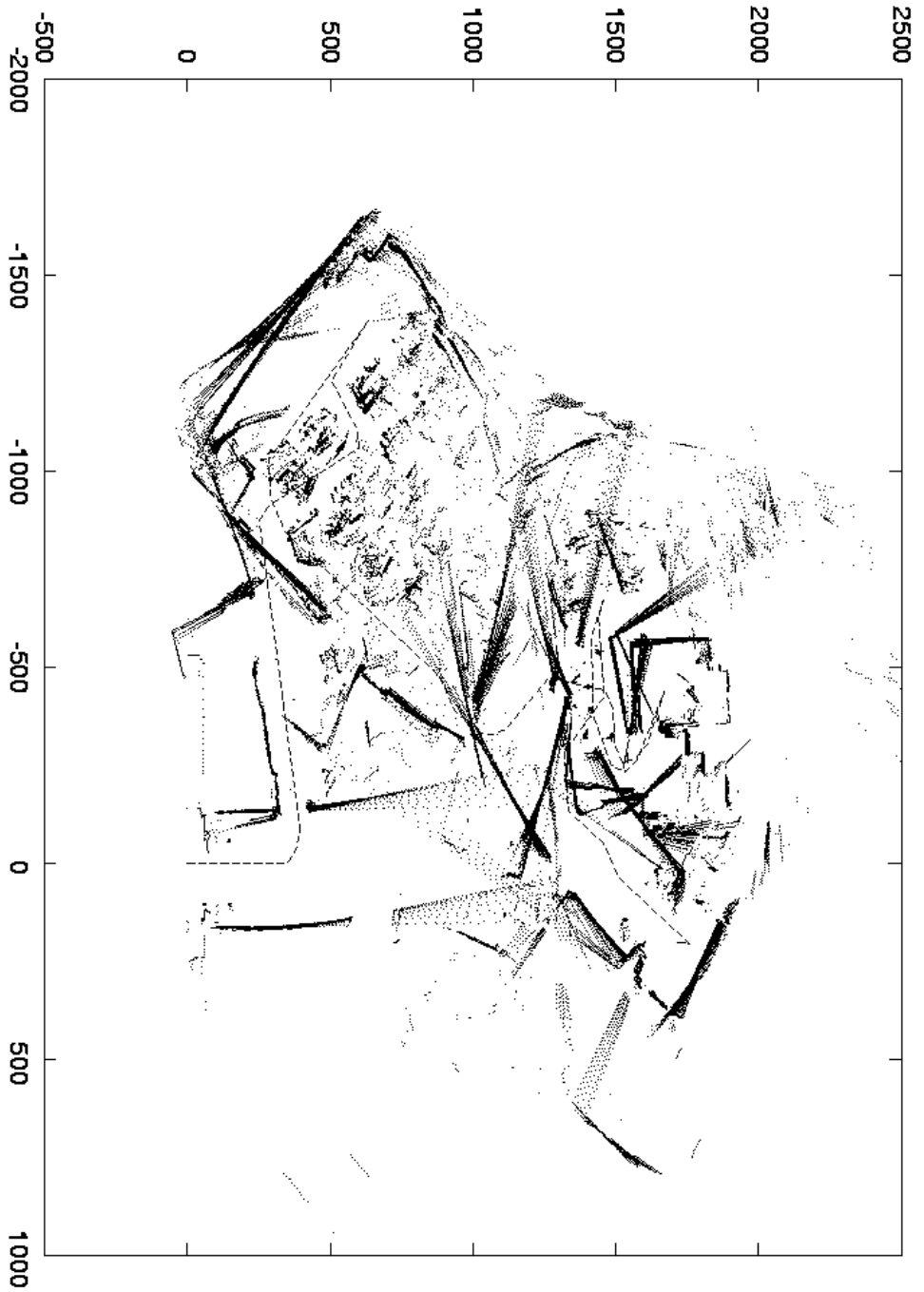


Figura 5.4: Mapa 2D construido directamente con las lecturas del odómetro.

En todos los mapas la trayectoria seguida por el robot se traza usando líneas y la escala mostrada esta en milímetros. Además, en los mapas parciales el robot es dibujado como un semicirculo del cual emergen los rayos proyectados por el telémetro láser para sensar el ambiente.

La localización del robot utilizando el método de Mínimos Cuadrados en nuestro ambiente real no obtiene resultados satisfactorios. La Figura 5.5 muestra diversas lecturas (0, 23, 46, 69, 92, 112, 162, 182 y 203) y el alineamiento de las mismas, es decir, el mapa 2D construido hasta ese momento. Claramente se puede observar que el método tienen serios problemas con la gran cantidad de datos atípicos. En otras palabras, el método de Mínimos Cuadrados para ponderar el error en un alineamiento de puntos contra puntos, no puede encontrar la transformación rígida Φ de rotación θ y traslación (t_x, t_y) que alinee correctamente los rastreos tomados del ambiente real. El mapa final obtenido se muestra en la Figura 5.6, el cual difiere bastante del plano arquitectónico de los laboratorios, esto se debe a la gran cantidad de errores de localización acumulados, por lo cual no es un mapa 2D útil.

Tanto en el método de Mínimos Cuadrados, como en el método del Estimador Lorentziano, los datos del odómetro son utilizados como valores iniciales para aplicar la metodología de NLM. Sin embargo, al utilizar el método propuesto, utilizando el Estimador Lorentziano para ponderar el error al alinear dos rastreos consecutivos da mejores resultados, obteniendo un mejor mapa 2D que en los casos anteriores. Esto se debe a que el Estimador es más robusto a la gran cantidad de datos atípicos. La Figura 5.7 muestra las mismas lecturas que las Figuras 5.3 y 5.5, se puede observar que el método si encuentra la transformación rígida Φ que alinea correctamente los rastreos, y el error de localización acumulado es mucho menor que en los otros casos. La semejanza entre el perímetro del mapa 2D construido con el método propuesto, que se presenta en la Figura 5.8, y el plano arquitectónico, mostrado en la Figura 5.2, es evidente.

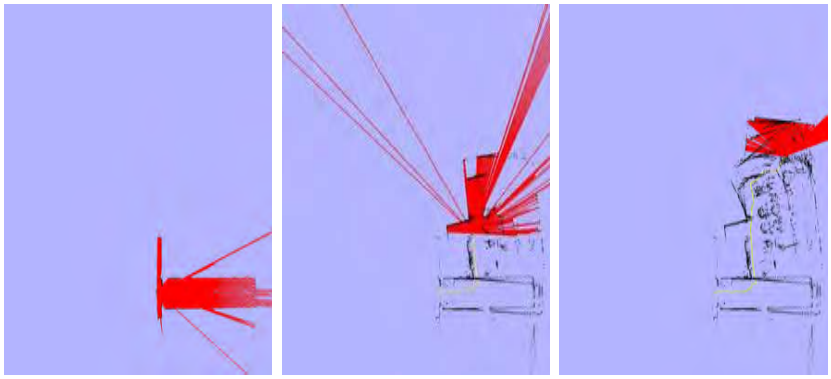
Experimentalmente se encontró que aplicar 3 veces el método NLM, con diferentes valores para σ, en el caso del Estimador Lorentziano, se obtienen un mejor alineamiento de dos rastreos consecutivos. Así los valores utilizados que mejor alinean los rastreos son $\sigma = 2$ la primera vez, $\sigma = 5$ la segunda vez y $\sigma = 0{,}5$ la tercera y última vez. Los valores de la transformación rígida calculados con la primera σ son los valores iniciales al aplicar por

segunda vez el método NLM, de igual forma los segundos valores calculados son los valores iniciales al aplicar el método por tercera vez.

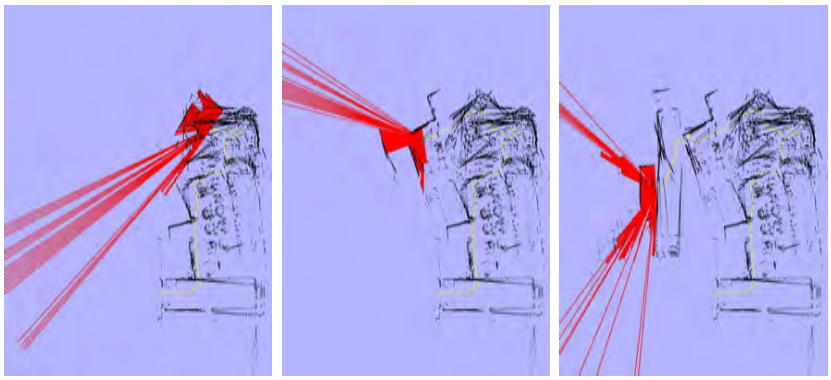
El método de alineamiento utilizando el estimador Lorentziano y NLM, también fue probado utilizando la historia de todos los rastreos previos, no solamente el anterior. Sin embargo ésto repercute notablemente en el desempeño computacional y la mejora que se obtiene es mínima.

Un caso típico del alineamiento de dos rastreos con muchos datos atípicos, donde el estimador Lorentziano da mejores resultados que el estimador de mínimos cuadrados, se presenta en la Figura 5.10. Los datos atípicos afectan la estimación de la localización del robot al usar el estimador de mínimos cuadrados y por lo tanto los dos rastreos consecutivos no se alinean.

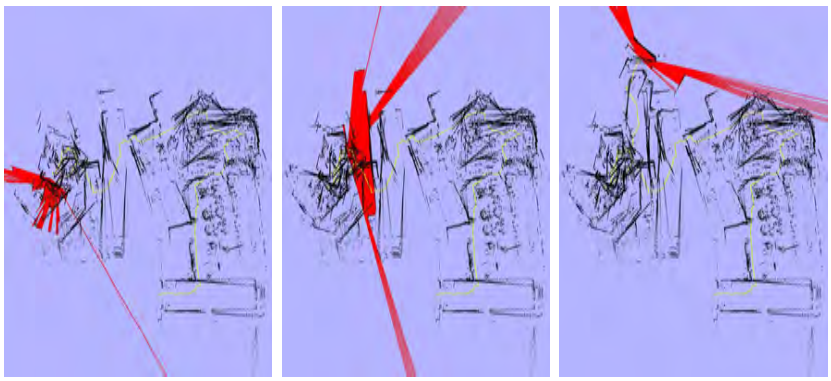
Sin embargo, como se vio en la Sección 3.4.3, la asociación puntos contra puntos no siempre es lo suficientemente buena, especialmente en ambientes que tienen una estructura regular, tal como nuestro ambiente real. Así pues, al aplicar la mejora para la asociación de datos, descrita en la Sección 3.4.3, en la cual se elige entre el punto más cercano sensado por el telémetro láser y un punto calculado sobre la línea más cercana, el mapa que se obtiene (ver Figura 5.9) es visiblemente mejor que el mapa de la Figura 5.8, en el que no se aplica el criterio de formación de líneas para mejorar la asociación de datos. La Figura 5.11 presenta una vista ampliada de los mapas que permite comparar la diferencia de construir el mapa 2D sin tomar ventaja de la estructura del ambiente, y tomado ventaja de la estructura del ambiente formando líneas a partir de él.



(a) Lectura 0 (b) Lectura 23 (c) Lectura 46



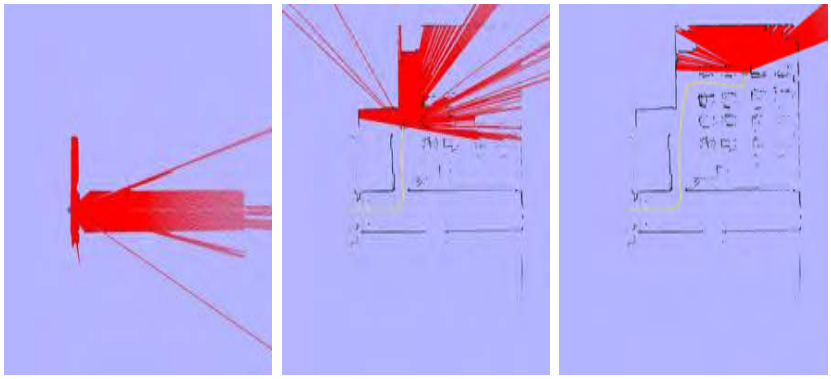
(d) Lectura 69 (e) Lectura 92 (f) Lectura 112



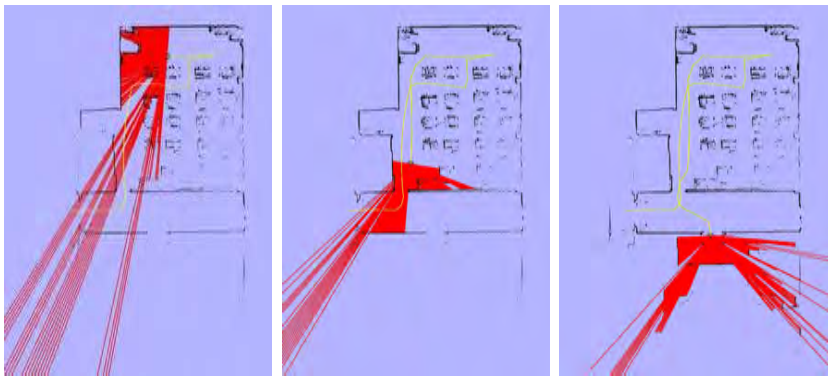
(g) Lectura 162 (h) Lectura 182 (i) Lectura 203

Figura 5.5: Lecturas del telémetro láser y el mapa 2D correspondiente utilizando el método de mínimos cuadrados.

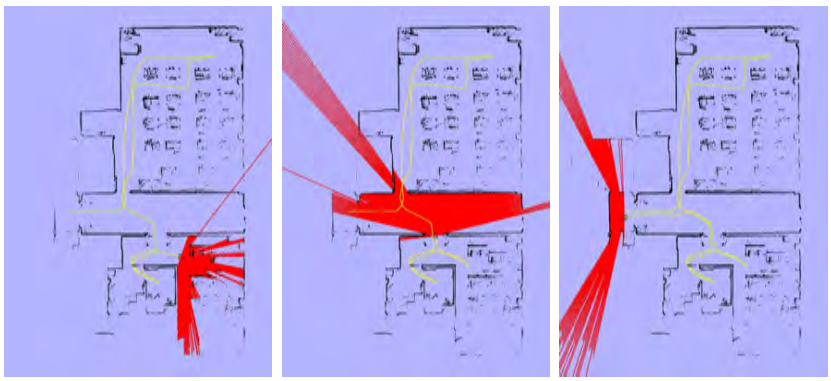
Figura 5.6: Mapa 2D construido aplicando el alineamiento con el estimador de mínimos cuadrados.



(a) Lectura 0 (b) Lectura 23 (c) Lectura 46



(d) Lectura 69 (e) Lectura 92 (f) Lectura 112



(g) Lectura 162 (h) Lectura 182 (i) Lectura 203

Figura 5.7: Lecturas del telémetro láser y el mapa 2D correspondiente utilizando el método del estimador Lorentziano.

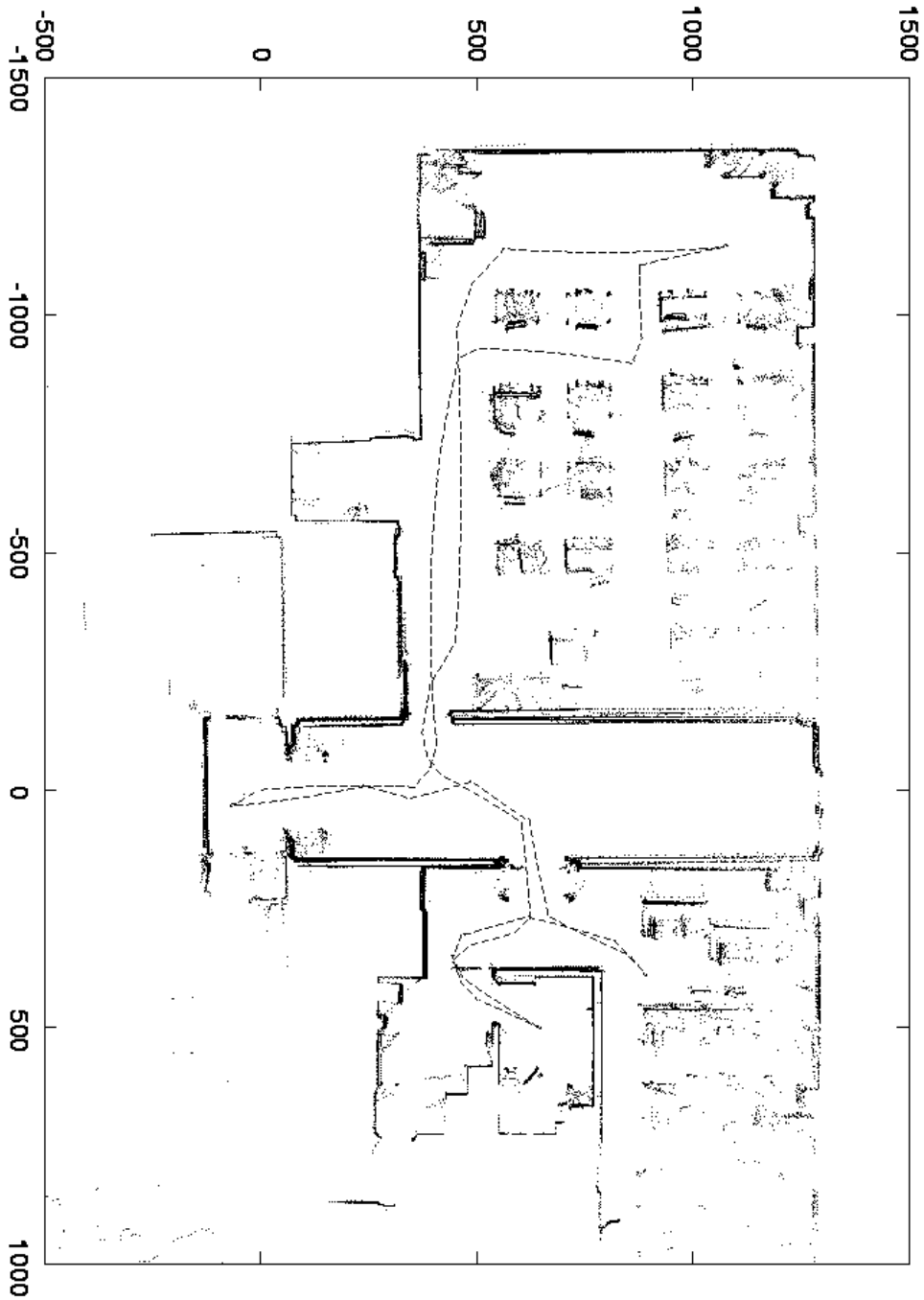


Figura 5.8: Mapa 2D construido aplicando el alineamiento con el estimador Lorentziano y el método NLM.

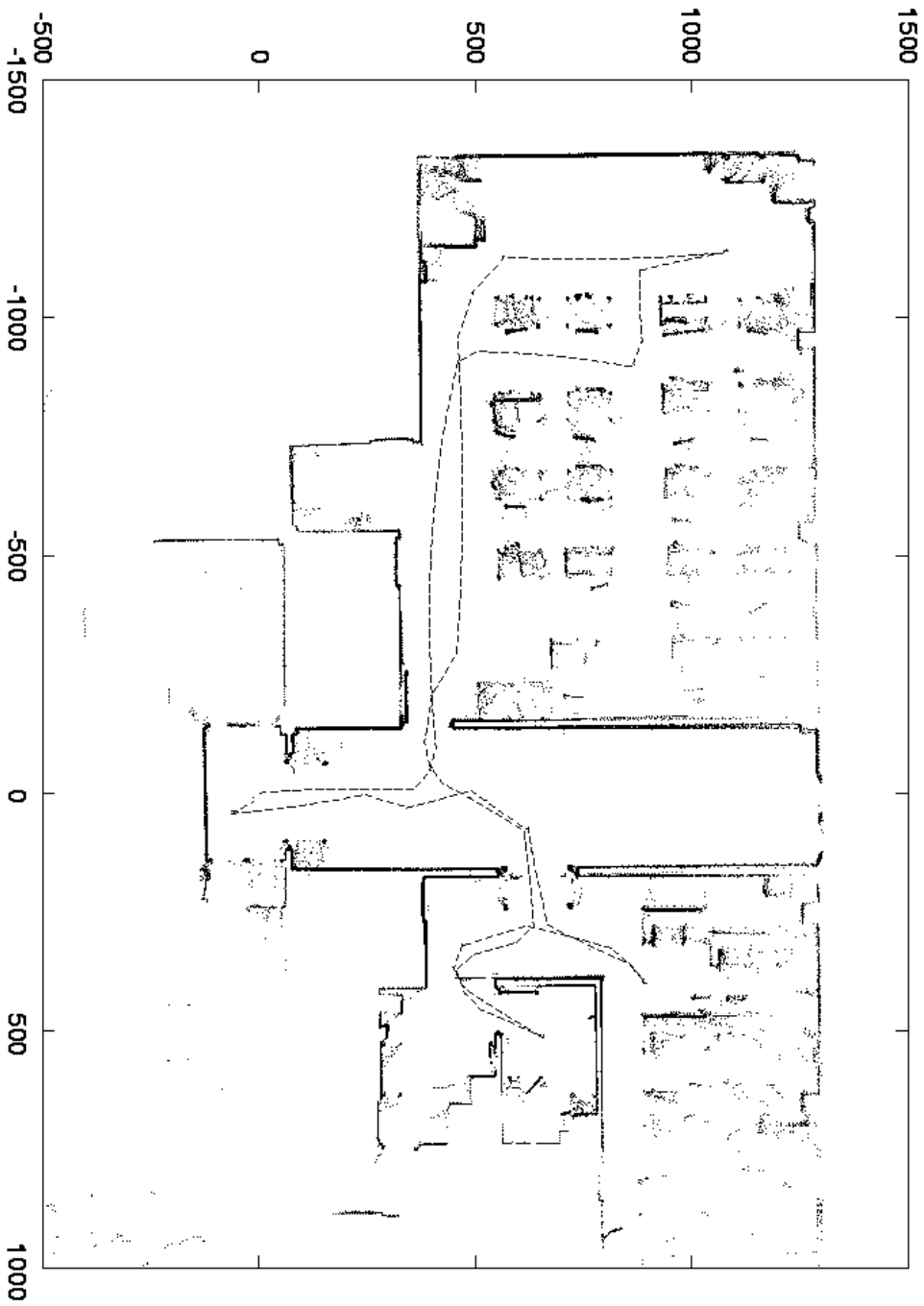
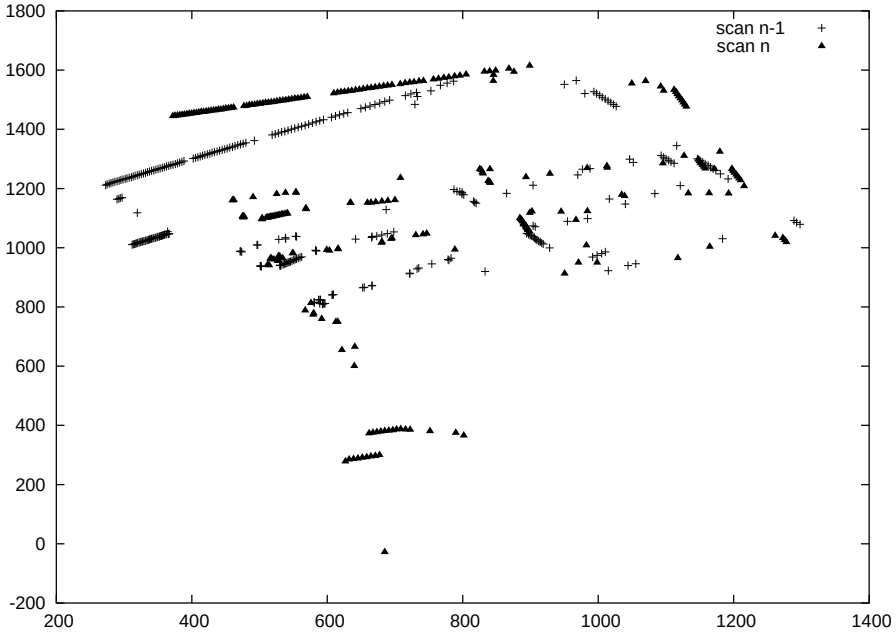
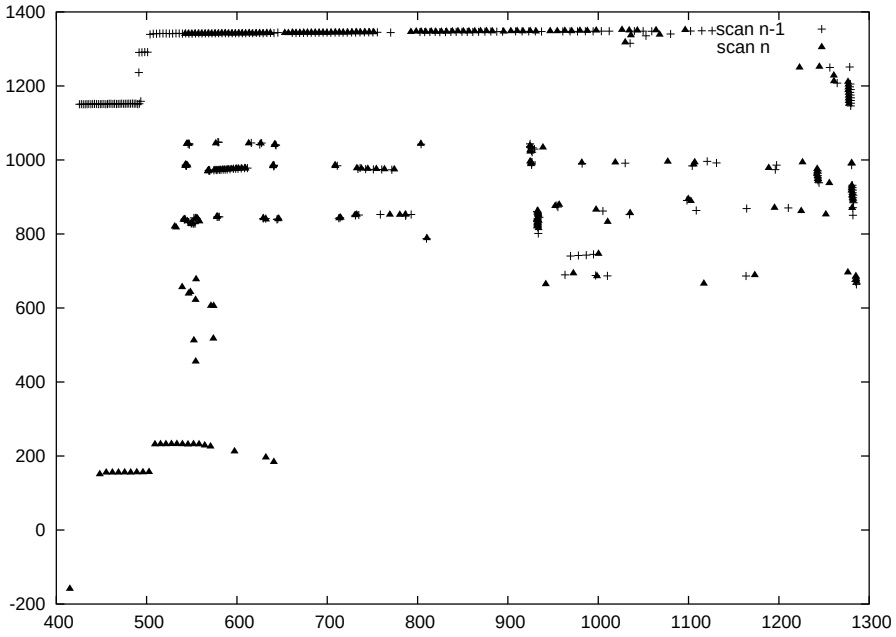


Figura 5.9: Mapa 2D construido aplicando el alineamiento con el método propuesto, y además tomando ventaja de líneas calculadas del rastreo previo.

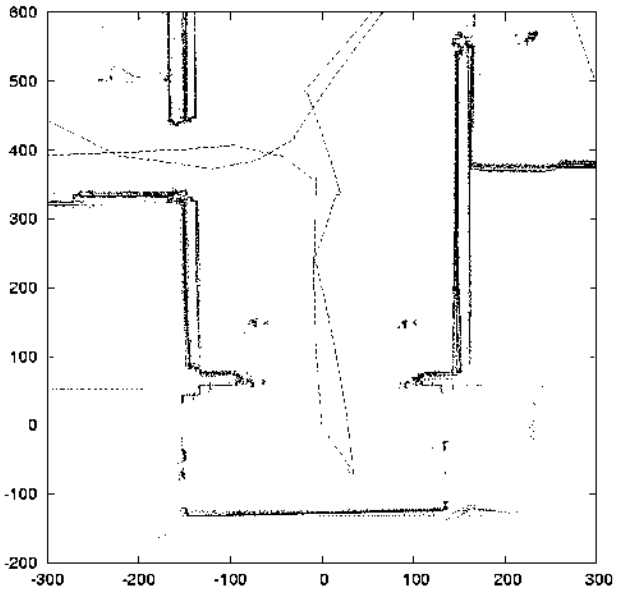


(a) Mínimos cuadrados

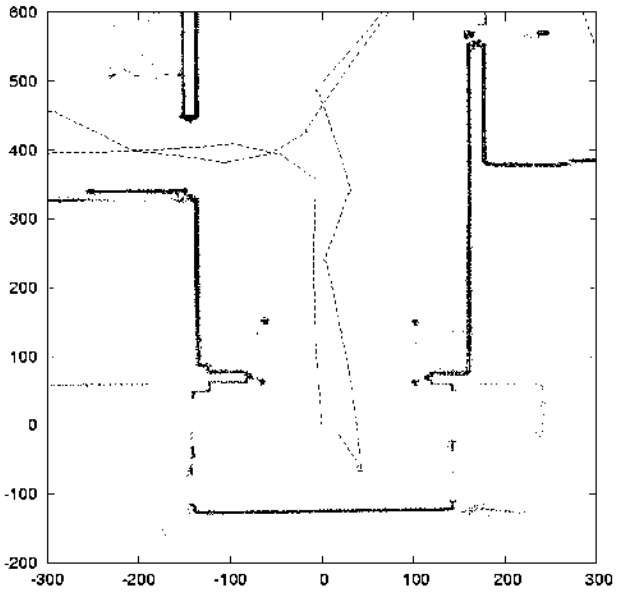


(b) Estimador Lorentziano

Figura 5.10: Comparación entre los métodos de mínimos cuadrados y estimador lorenziano para un emparejamiento de rango del tipo datos-datos.



(a) Alineamiento punto sensado contra punto sensado



(b) Alineamiento punto *calculado* contra punto sensado

Figura 5.11: Vistas amplificadas de los mapas utilizando diferentes criterios en la asociación de datos.

5.3. Reconstrucción 3D

El equipo de cómputo utilizado en la reconstrucción es una máquina *WorkStation Dell Precision*, con un procesador *Xeon* @ 3.2 Ghz, 1 MB de memoria RAM, y una tarjeta de aceleración Grafica *NVidia Quadro pro*, la cual está optimizada para trazar cuadriláteros.

Todos los experimentos de reconstrucción 3D se realizaron sobre los ambientes reales anteriormente señalados, el Laboratorio de Instrumentación y Control, y el Laboratorio de Sistemas Computacionales. La adquisición de los datos de rango se realizó utilizando las posibles combinaciones que nuestro robot móvil y el sistema de giro-inclinación (*pan-tilt*) permiten. Estas son: robot fijo e inclinación del *pan-tilt*, robot fijo pero con inclinación y giro del *pan-tilt*, y finalmente, robot desplazado con inclinación y giro del *pan-tilt*. En seguida se presentan los resultados experimentales de cada una de éstas combinaciones.

5.3.1. Reconstrucción 3D con robot fijo e inclinación del pan-tilt

En un primer experimento se realizó la reconstrucción 3D de una escena real con el robot fijo y la inclinación del *pan-tilt*. El escenario fue una parte del Laboratorio de Instrumentación y Control, el cual se muestra en la Figura 5.12. Como se puede observar, existen diversos objetos en la escena: cajas, sillas, puertas, muros, una mesa, y algunos otros objetos más.

Para reconstruir una escena 3D, basta con utilizar la inclinación del *pan-tilt*, desde su posición inicial hasta su posición final de inclinación. Para nuestro sistema de adquisición de datos de rango en particular (ver Sección 4.2), en 100 diferentes posiciones de inclinación y 361 datos por rastreo láser, un total de 36100 datos son adquiridos por escena 3D. La Figura 5.13 muestra los puntos adquiridos, transformados en vértices de OpenGL, de la escena mostrada en la Figura 5.12. Se puede observar que apenas son perceptibles algunos de los objetos y detalles de la escena.

Con el conjunto de vértices que forman una escena 3D se construye una malla poligonal. Los polígonos de la malla pueden ser cuadriláteros o triángulos. Sin embargo, en nuestro caso particular, para que se trate de un polígono válido, la distancia del primer vértice al resto de los vértices no debe ser mayor a $200mm$, así se descartan la mayoría de los datos atípicos y se evita construir polígonos no deseados.

Figura 5.12: Imagen parcial del Laboratorio de Instrumentación y Control, correspondiente al ambiente real a reconstruir.

Figura 5.13: Escena reconstruida con vértices 3D en OpenGL.

La escena 3D reconstruida, y en general todo el modelo 3D reconstruido en OpenGL, tiene una fuente de luz que es una combinación de luz ambiental y luz difusa, ambas con una intensidad de RGB del 30 % para los tres canales. La localización de la fuente de luz está en el centro del sistema de referencia global, a la altura del sensor láser.

El color asociado a los polígonos de la malla tiene 95 % de intensidad de RGB, en los tres canales, y el material responde a los tipos de luz ambiental y difusa. La dirección en la cual la luz es reflejada por un cuadrado o triángulo, está en función de sus vértices (ver Sección 4.4, Figura 4.8 (b) y (c)).

La Figura 5.14 presenta la escena 3D reconstruida con una malla poligonal de triángulos, mientras que en la Figura 5.15 se presenta el caso de una malla poligonal de cuadriláteros. A simple vista no existe mucha diferencia entre ambas mallas. Sin embargo, la malla poligonal de triángulos tiene 59924 polígonos válidos, mientras que la malla poligonal de cuadriláteros tiene 29785 polígonos válidos, casi la mitad que la cantidad de triángulos.

Además de la cantidad de polígonos válidos, otra diferencia es la forma en que los polígonos reflejan la luz, que además de definir la forma de los objetos y superficies en la escena 3D, nos permite observar el error del sensor láser cuando rastrea superficies "planas" como las paredes. La Figura 5.16 contiene un acercamiento de ambas mallas poligonales, donde se puede apreciar la diferencia entre ellas.

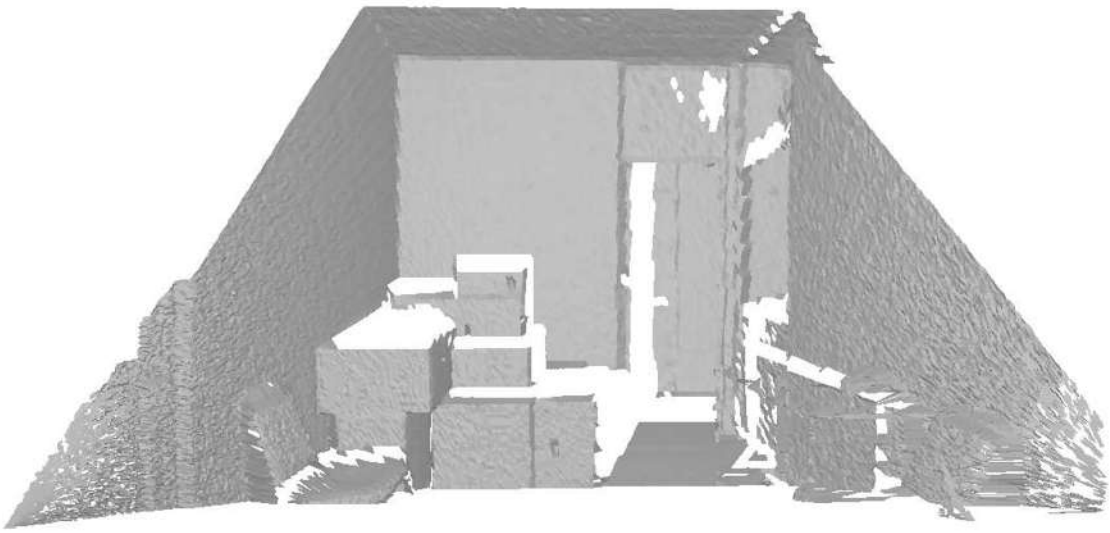


Figura 5.14: Escena 3D reconstruida con una malla poligonal de triángulos en OpenGL.

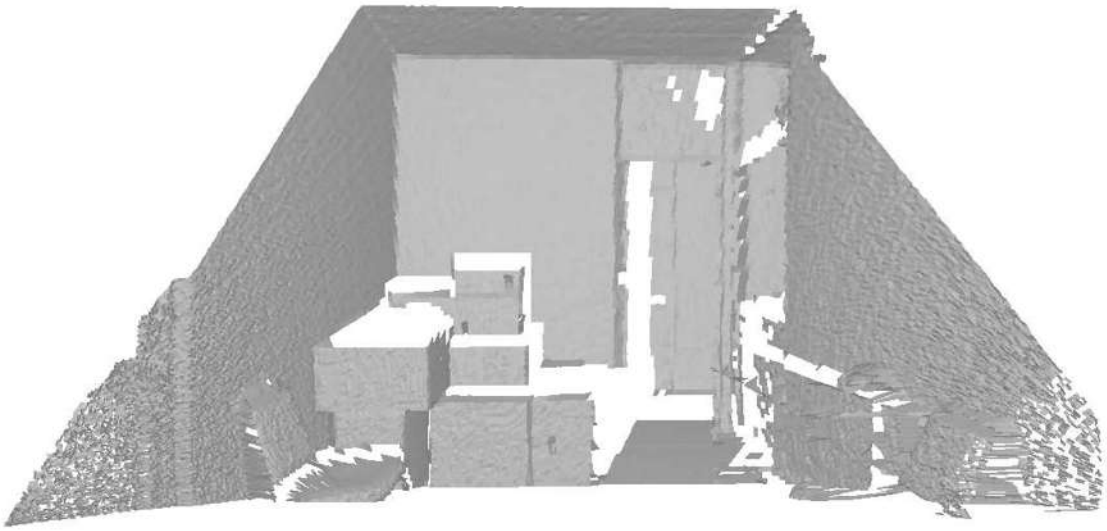
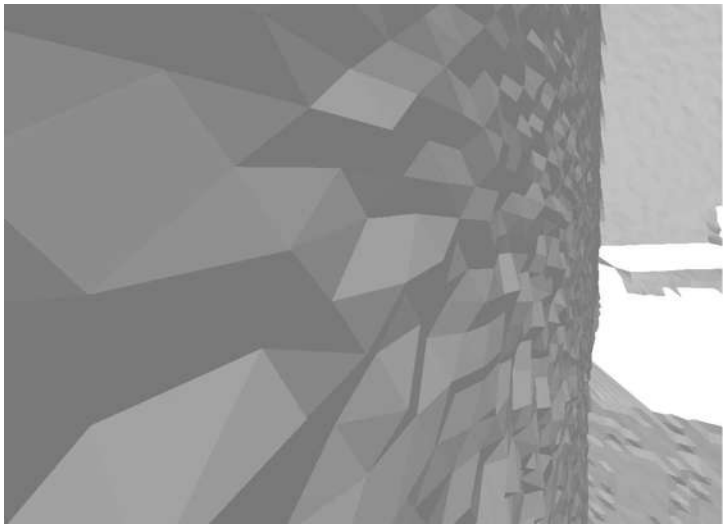
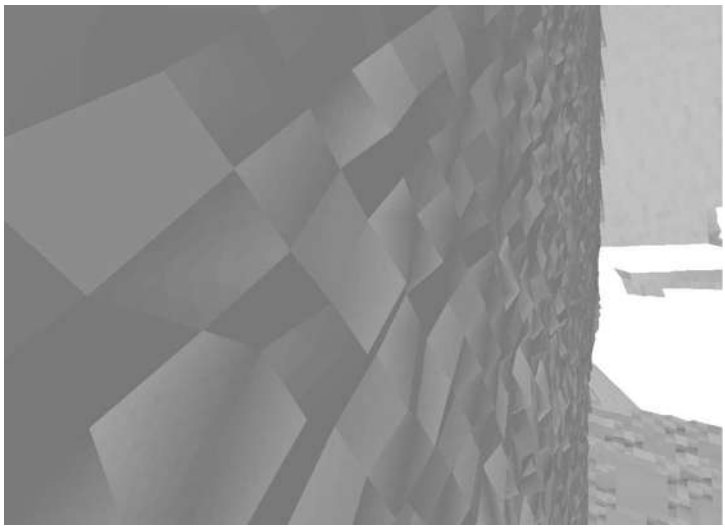


Figura 5.15: Escena 3D reconstruida con una malla poligonal de cuadriláteros en OpenGL.



(a) Malla poligonal de triángulos



(b) Malla poligonal de cuadriláteros

Figura 5.16: Acercamiento a las mallas poligonales de triángulos (a) y cuadriláteros (b).

5.3.2. Reconstrucción 3D con robot fijo y giro inclinación del pan-tilt

El segundo experimento se realizó con el robot fijo, utilizando las operaciones de inclinación y giro del *pan-tilt*. Como en el primer experimento, la operación de inclinación es utilizada para adquirir y reconstruir una escena 3D. La operación de giro es utilizada para adquirir y reconstruir 360° alrededor del robot. Prácticamente, con tres escenas 3D, una cada 120°, es posible cubrir los 360°. Sin embargo, con cuatro escenas 3D, una cada 90° se cubre una mayor área vertical del ambiente. Esto ocurre porque en los límites del conjunto de rastreos, cerca de $-90°$ o de 90°, el área vertical cubierta es poca, debido a que la inclinación del *pan-tilt* se realiza sobre un eje fijo.

La Figura 5.17 muestra una vista de la reconstrucción 3D con cuatro escenas 3D, una cada 90°, orientada en dirección de la primer escena 3D reconstruida.

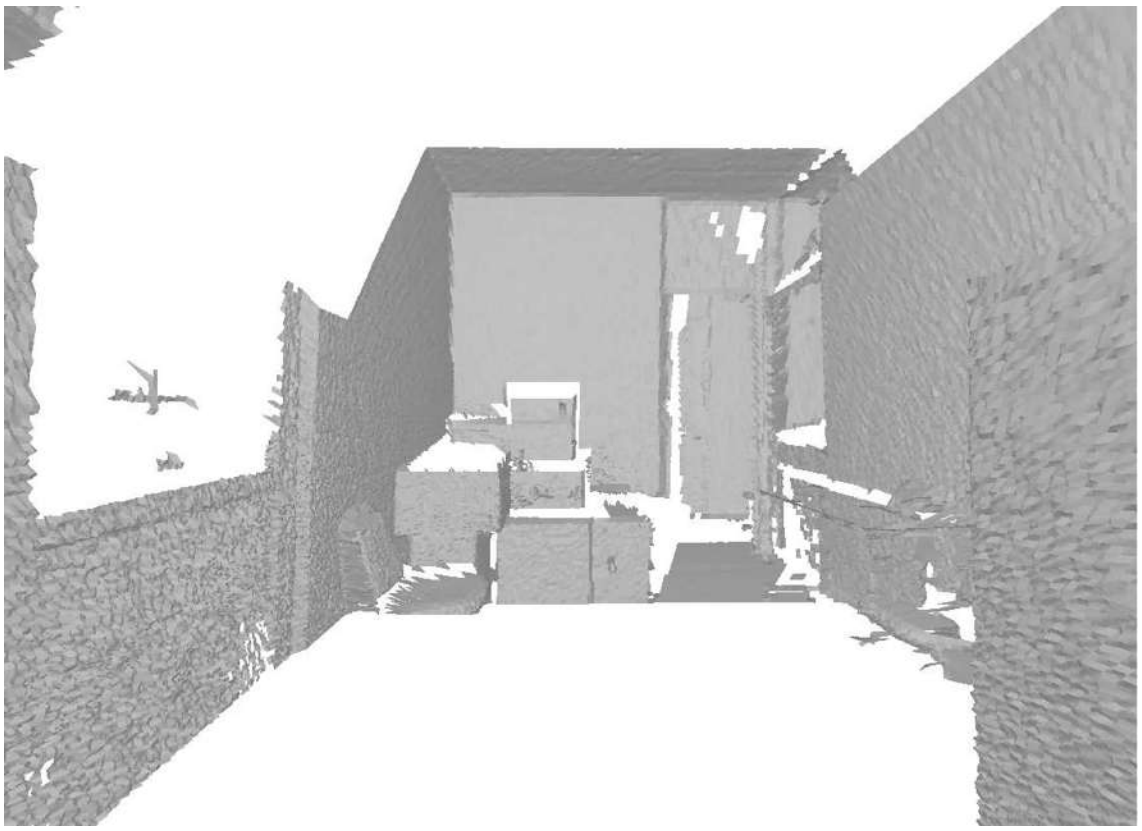


Figura 5.17: Reconstrucción 3D con cuatro escenas 3D, una cada 90°.

5.3.3. Reconstrucción 3D desplazando el robot con giro e inclinación del pan-tilt

El modelo 3D final está formado por las escenas 3D reconstruidas del ambiente. Dichas escenas 3D son capturadas de cuatro en cuatro, antes de cada movimiento (rotación o desplazamiento) del robot. Este experimento toma ventaja de la localización local del robot para ubicar la posición en el espacio de las escenas 3D reconstruidas una vez que el robot se mueve. Las siguientes figuras muestran el resultado del experimento. En la Figura 5.18 se presenta el escenario real (LSC) a reconstruir con más de una posición del robot. Como se puede observar, en el escenario pueden apreciar diversos objetos como: mesas, computadoras, sillas, puertas, ventanas, lámparas, etc.

Figura 5.18: Escenario a reconstruir tridimensionalmente (LSC) con más de una posición del robot.

La Figura 5.19 presenta una reconstrucción 3D en una primera posición del robot dentro del ambiente. La Figura 5.20 muestra la reconstrucción 3D de una segunda posición del robot, es decir, una vez que el robot se movió de la primera posición a una segunda posición. Se puede observar que algunos de los objetos aparecen más detallados en una reconstrucción que en otra, además algunos otros aparecen en una sí y en otra no.

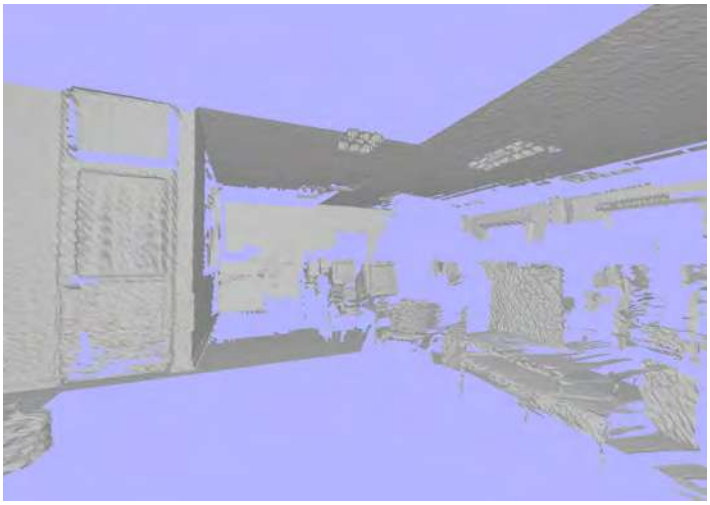


Figura 5.19: Reconstrucción 3D de LSC en una primera posición del robot.

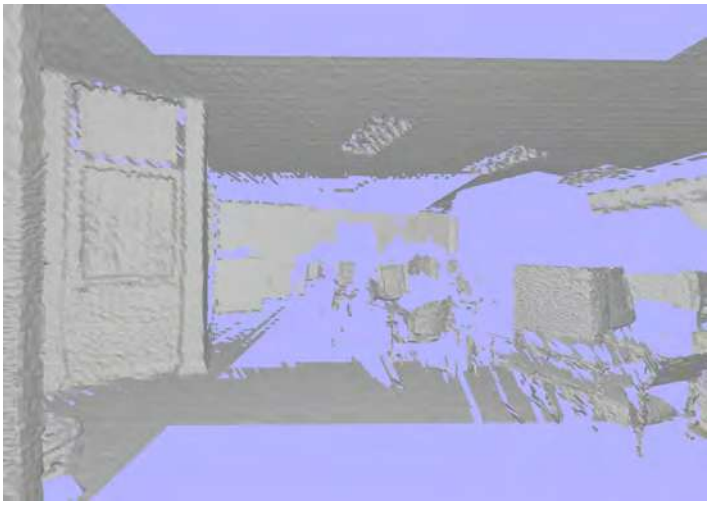


Figura 5.20: Reconstrucción 3D de LSC en una segunda posición del robot.

En la Figura 5.21 se presenta la integración de las reconstrucciones mostradas en las Figuras 5.19 y 5.20. Para ello se calculó la localización local del robot mediante la metodología propuesta en el capítulo 3, es decir, fue posible conocer la traslación y rotación que realizó el robot de la primera posición a la segunda y así integrar ambas reconstrucciones 3D en una sola.

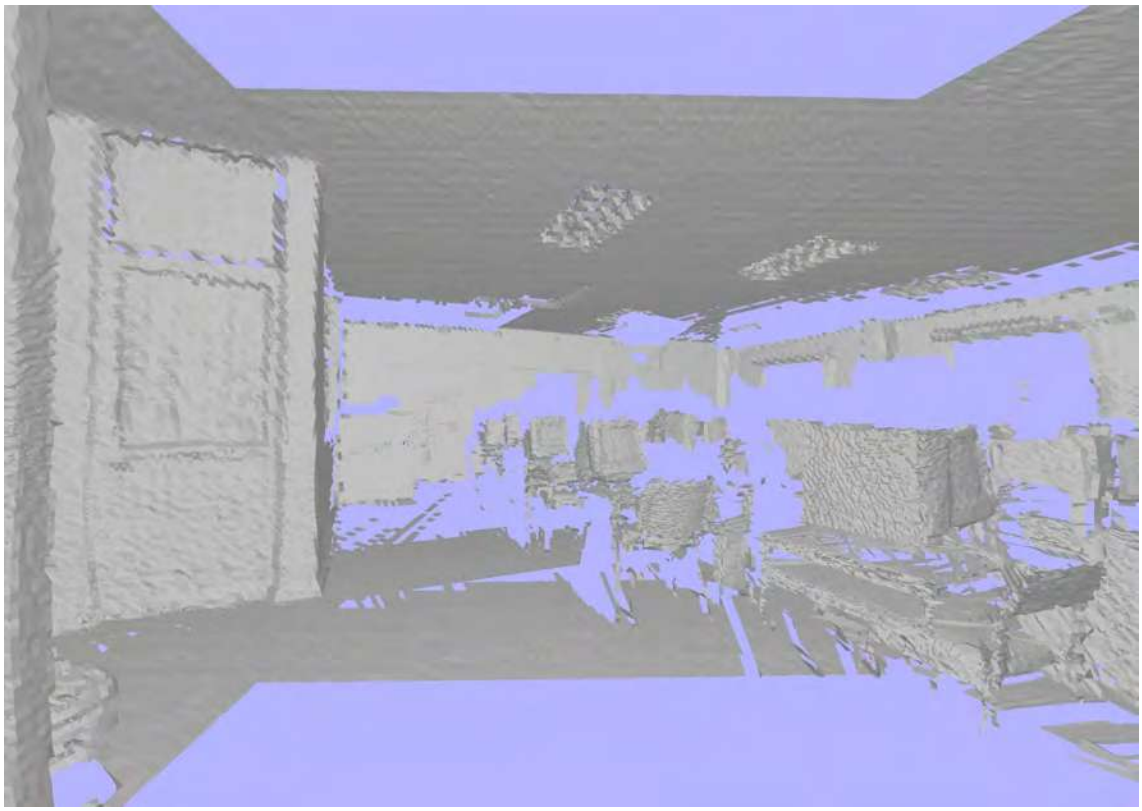


Figura 5.21: Integración de dos reconstrucciones 3D realizadas en dos distintas posiciones del robot.

5.4. Conclusiones

En éste capítulo se presentaron los resultados experimentales sobre la localización local y construcción de mapas 2D, así como los referentes a la reconstrucción 3D.

En lo que respecta a la localización local, los experimentos demostraron que las lecturas del odómetro son insuficientes para resolver el problema de la localización local, debido a las irregularidades del piso. Sin embargo, son útiles como valores iniciales para los métodos de optimización empleados en el alineamiento de dos rastreos consecutivos.

Cuando se aplica un estimador de mínimos cuadrados en el alineamiento, los datos atípicos lo hacen fallar, es decir, no encuentra los valores de rotación y traslación correctos. Por otro lado, al aplicar el estimador Lorentziano, en el ambiente real o en el simulado, resultó ser bastante robusto a los datos atípicos, sin importar que tan grandes sean éstos.

El tiempo de cómputo para el alineamiento de dos rastreos consecutivos, aplicando el estimador Lorentziano (aproximadamente 0.38 segundos, en promedio, sobre una computadora Pentium III @ 1.0 Ghz), es suficientemente bueno para localizar el robot en tiempo real.

Tomar ventaja de la estructura del ambiente, con el ajuste de segmentos de línea en los rastreos, reduce visiblemente el error en el alineamiento, por lo que el mapa, y en sí la localización del robot, es mejor. Sin embargo, existe una pequeña penalización de tiempo.

En lo que se refiere a la reconstrucción 3D, los vértices generados a partir de los datos sensados por el telémetro láser, forman las mallas poligonales correspondientes a las escenas 3D reconstruidas. La iluminación permite visualizar la forma de los objetos inmersos en el ambiente reconstruido. Estos objetos se forman visualmente por un conjunto de pequeñas superficies, es decir, polígonos de la malla: cuadriláteros o triángulos.

Cuando la malla poligonal está formada por cuadriláteros, se requiere una cantidad menor de polígonos. Si el *hardware* de aceleración gráfica está optimizado para trazar cuadriláteros, evidentemente el *rendering* de la malla tendría un mejor desempeño que si se tratara de una malla poligonal de triángulos.

Una malla poligonal formada por triángulos contiene pequeñas superficies que una malla poligonal formada por cuadros no cubre, ésto es más notorio en los bordes de los objetos reconstruidos. Lo anterior se debe a que un triángulo solo requiere de tres vértices, a diferencia del cuadrilátero que requiere de cuatro. Además, si el *hardware* de aceleración gráfica está optimizado para trazar triángulos, entonces el *rendering* de la malla poligonal de triángulos tendría mejor desempeño, a pesar de tener una mayor cantidad de polígonos.

Capítulo 6

Conclusiones y trabajos futuros

Esta tesis ha descrito el problema de la reconstrucción y modelado tridimensional de ambientes interiores, por lo cual se abordaron dos problemas principales: la localización local y construcción de mapas 2D, así como la reconstrucción tridimensional de ambientes reales. En este capítulo se recopilan las principales conclusiones y sugerencias para trabajos futuros derivados de esta de tesis.

6.1. Conclusiones

Este trabajo abordó los problemas de localización local y construcción de mapas 2D, así como la reconstrucción tridimensional de ambientes reales, utilizando un robot móvil equipado con un telémetro láser montado sobre un sistema de giro inclinación, también conocido como *pan-tilt*.

Se espera que la información provista en el Capítulo 2 sea útil para otros investigadores, y con ello fomentar el desarrollo e investigación en robots de plataforma abierta, es decir, sin códigos propietarios. Las ligas relacionadas y el software desarrollado en este trabajo de tesis están disponibles en Internet, en la dirección http://faraday.fie.umich.mx/ jjap/robot.

La utilización de OpenGL permitió: (1) distintas representaciones gráficas de los datos sensados y del modelo virtual 3D, (2) uso directo del hardware de aceleración gráfica, y (3) abstracción de la geometría proyectiva tridimensional y su mapeo final a la ventana de desplegado.

En lo que se refiere a la localización local y construcción de mapas 2D, el método propuesto utilizando el estimador Lorentziano y el método de optimización no líneal Newton-

Levenberg-Marquardt proporciona suficiente robustez sobre los datos atípicos. Además, también resuelve el problema de oclusión en dos rastreos consecutivos al tratar dicha oclusión como datos atípicos, ya que el método descarta de forma natural dichos datos.

Nuestro robot móvil está lo suficientemente equipado para ser utilizado como plataforma de adquisición de datos en la obtención de escenas 3D, para generar el modelo tridimensional como una malla poligonal texturizada. El sistema de giro-inclinación del robot, permite realizar la adquisición de datos de rango con el telémetro láser, de tal forma, que sin necesidad de girar el robot, es posible reconstruir todos los objetos que sean alcanzados por el sensor láser alrededor de robot. La representación de los datos de rango 3D en coordenadas homogéneas y las de matrices de transformación para relacionar los diferentes sistemas de referencia del robot, facilitan el cálculo del conjunto de puntos 3D (vértices) que conforman una escena 3D.

La parte experimental se presentó en el capítulo 5, de dichos experimentos se desprenden las siguientes conclusiones:

- El robot y los sensores con los que está equipado fueron suficientes para abordar y resolver los dos problemas anteriormente mencionados. Además se podrían abordar otros problemas de robótica como: planeación y seguimiento de trayectorias, visión estéreo para reconstrucción 3D, navegación con evasión dinámica de obstáculos, y otros más.

- Los experimentos demostraron que las lecturas del odómetro no son suficientes para resolver el problema de la localización local, debido a las irregularidades del piso. Sin embargo, si son útiles en la inicialización de los métodos de optimización empleados para el alineamiento de dos rastreos consecutivos.

- También se demostró que un estimador de mínimos cuadrados falla cuando se presentan datos atípicos. Es decir, no encuentra los valores de rotación y traslación correctos. Por otro lado, el estimador Lorentziano en el ambiente real, o en el simulado, resultó ser bastante robusto a los datos atípicos, sin importar que tan grandes sean éstos.

- El tiempo de cómputo para el alineamiento de dos rastreos consecutivos, aplicando el estimador Lorentziano es de aproximadamente 0.38 segundos en promedio sobre la computadora a bordo del robot móvil, tiempo razonable para localizar el robot en tiempo real.

- El ajuste de segmentos de línea en los rastreos permite tomar ventaja de la estructura del ambiente. Los experimentos mostraron que el error se reduce notablemente, por lo que el mapa, y en sí la localización del robot, es mejor cuando se realiza el ajuste. Sin embargo, existe una pequeña penalización de tiempo.

- La capacidad de localización del robot después de realizar cualquier movimiento con él, provee de la posibilidad de realizar reconstrucciones 3D prácticamente completas del ambiente en el cual se desplaza el robot móvil.

- En la construcción del modelo 3D, el manejo del color, la luz y las propiedades del material con OpenGL proporcionaron a las escenas trazadas con mallas poligonales un mayor realismo, permitiendo identificar visualmente la forma de la mayoría de los objetos reconstruidos del ambiente real.

- Finalmente el modelo 3D generado a partir de la reconstrucción del ambiente real puede ser trazado por dos tipos de polígonos: triángulos o cuadriláteros. Cuando la malla poligonal está formada por cuadriláteros, se requiere una cantidad menor de polígonos, aproximadamente la mitad que si se tratara de una malla poligonal de triángulos. Sin embargo una malla poligonal formada por triángulos contiene superficies más pequeñas que una malla poligonal formada por cuadriláteros. La utilización de una malla poligonal formada por triángulos o cuadriláteros depende en gran medida del tipo de hardware de aceleración grafica que se tenga.

6.2. Sugerencias para trabajos futuros

Algunas de las sugerencias para trabajos futuros que pueden ampliar o complementar el trabajo realizado en esta tesis son las siguientes:

- Adaptación de motores sin escobillas. Los motores actuales del robot son motores con escobillas por lo cual se dificulta un poco el control. Es recomendable pues utilizar motores de imanes permanentes sin escobillas tanto en las ruedas de tracción como en el sistema *pan-tilt*. Sobre todo en éste último donde el control de posicionamiento incide directamente sobre la fiabilidad de los datos obtenidos. Con estos nuevos motores la velocidad y precisión de posicionamiento para adquirir una escena 3D tendría un mejor desempeño.

- Aplicar un método robusto de ajuste de segmentos de líneas rectas a los rastreos, con el fin de mejorar la asociación de datos (puntos calculados contra puntos sensados), además de simplificar la representación espacial de un mapa global 2D del ambiente.
- Considerar la incertidumbre de las mediciones producidas por el láser, tanto para el problema de la localización local y construcción de mapas 2D, como para la reconstrucción y modelado 3D.
- Realizar una simplificación poligonal que reduzca el número de vértices, ya que cuando el robot se mueve adquiere más y más puntos, lo que provoca una gran cantidad de vértices que al graficarse disminuyen el desempeño del modelado 3D.
- Texturización con imágenes reales. La integración de imágenes reales, tomadas por una cámara, al modelo 3D reconstruido, y posiblemente simplificado, le daría una apariencia fotorealista, con lo cual el modelo 3D tendría una información visualmente más rica del ambiente real reconstruido.
- Con el modelo 3D reconstruido, planear trayectorias para realizar una navegación capaz de evitar obstáculos en el espacio tridimensional.
- Desarrollar una estrategia automática de exploración para reconstruir el ambiente con el mínimo de movimientos.

Apéndice A

Control del robot

Este apéndice complementa el documento del protocolo de comunicación serial con el robot [Romero04a], que trata del encendido y apagado del robot, así como los comandos empleados para la comunicación con él. Este apéndice también contiene los comandos empleados para comunicarse con el sistema de giro-inclinación (*pan-tilt*),así como el código correspondiente al control tanto de la base del robot como del *pan-tilt*.

A.1. Encendido y apagado del robot

En la parte inferior del robot, visto por atrás, se encuentran dos interruptores de palanca que controlan el suministro de energía del robot:

- El interruptor del lado derecho (visto el robot por atrás), controla la alimentación a las cargas constantes del robot: los dos microcontroladores, la tarjeta de control de disparo del anillo de sonares, la alimentación externa de 15v regulados (para alimentar un *Hub Firewire* para cámaras digitales) y la alimentación externa de 24v (para alimentar un telémetro láser). En adelante se llamará a este interruptor como el *interruptor de cargas constantes*.

- El interruptor del lado izquierdo (visto el robot por atrás), controla la alimentación a los motores que están conectados a las dos ruedas de tracción del robot, así como los motores del *pan-tilt*. En adelante se llamará este interruptor como el *interruptor de los motores*.

Cuando se activa el interruptor de cargas constantes (jalando hacia afuera del robot), se enciende un led rojo que está situado arriba del interruptor, mostrando que el robot está listo para recibir órdenes a través de los puertos seriales correspondientes a cada uno de los dos microcontroladores.

Cuando se activa el interruptor de los motores (jalando hacia afuera del robot), se enciende un led rojo que está situado arriba del interruptor mostrando que el robot esta listo para moverse. La activación de este interruptor también inicializa la posición inicial del *pan-tilt*.

La posición inicial de giro del *pan-tilt* se alcanza girando a la derecha hasta presionar un switch, el cual señala la posición inicial de giro, una vez presionado se gira un poco a la izquierda para liberarlo, con ello se indica que la posición inicial de giro ha sido alcanzada. La posición inical de inclinación se alcanza de manera similar, descendiendo el sistema de inclinación hasta presionar el switch que señala la posición inicial de inclinación, una vez presionado se eleva un poco el *pan-tilt* para liberarlo, una vez liberado, la posición inicial de inclinación ha sido alcanzada.

Entre los dos leds anteriormente mencionados se localiza un interruptor de botón llamado *interruptor de reset* de los microcontroladores. Cuando se oprime se genera un reset del microcontrolador, reiniciando la ejecución del programa almacenado en cada uno de los microcontroladores. Este botón también funciona como un interruptor de emergencia: al ser oprimido, se para la actividad del robot; si estaba en movimiento, los motores se paran; si el anillo de sonares estaba efectuando disparos, entonces se detine los disparos.

A.2. Comunicación con el robot

El robot obedece órdenes dadas a través del puerto serial siguiendo la sintaxis:

$< c >< p1 >< sp >< p2 >< sp >< p3 > ... < pn >< \backslash n >$

Donde: $< c >$ indica una letra que identifica la orden o comando dada al robot.

$< pn >$ indica el n-ésimo argumento numérico no negativo de la orden.

El argumento debe ser un número en el rango de 0 a 65535.

$< sp >$ indica un caracter de espacio.

$< \backslash n >$ indica un caracter de salto de línea.

El puerto serial debe configurarse de la siguiente forma: 9600bps, 8bits, sin paridad y sin control de flujo. A continuación se describen los comandos que obedece el robot.

A.2.1. Comandos del robot

Comando de control del zumbador

Comando:	$z < F >< \backslash n >$ Donde F puede ser 0 ó 1.
Respuesta:	$z < \backslash n >$
Descripción:	Si $F = 1$, se activa un zumbador localizado en la parte inferior al frente del robot. Si $F = 0$, se apaga el zumbador. El robot responde a este comando regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea.
Ejemplos:	$z0 < \backslash n >$ $z1 < \backslash n >$

Comando para avanzar en línea recta

Comando:	$a < c >< \backslash n >$ Donde c es un número entre 0 y 65535.
Respuesta:	$a < \backslash n >$
Descripción:	El robot avanza en línea recta una distancia especificada por el argumento c. Las unidades de c están dadas en décimas de vuelta del motor acoplado a las ruedas. El robot responde regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea, e inmediatamente se inicia el movimiento. La respuesta del robot no se espera a que termine el movimiento.
Ejemplos:	$a10 < \backslash n >$

Comando para retroceder en línea recta

Comando:	$r < c >< \backslash n >$ Donde c es un número entre 0 y 65535.
Respuesta:	$c < \backslash n >$
Descripción:	El robot retrocede en línea recta una distancia especificada por el argumento c. Las unidades de c están dadas en décimas de vuelta del motor acoplado a las ruedas. El robot responde regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea, e inmediatamente se inicia el movimiento. La respuesta del robot no se espera a que termine el movimiento.
Ejemplos:	$r10 < \backslash n >$

Comando para girar a la izquierda

Comando:	$i < c >< \backslash n >$ Donde c es un número entre 0 y 65535.
Respuesta:	$i < \backslash n >$
Descripción:	El robot gira (en el mismo lugar) hacia la izquierda una distancia especificada por el argumento c. La rueda izquierda (visto el robot por atrás) gira hacia atrás, al mismo tiempo que la rueda derecha gira hacia adelante. El argumento c mide la distancia que debe recorrer cada rueda. Las unidades de c están dadas en décimas de vuelta del motor acoplado a las ruedas. El robot responde regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea, e inmediatamente se inicia el movimiento. La respuesta del robot no se espera a que termine el movimiento.
Ejemplos:	$i10 < \backslash n >$

Comando para girar a la derecha

Comando:	$d < c >< \backslash n >$ Donde c es un número entre 0 y 65535.
Respuesta:	$d < \backslash n >$
Descripción:	El robot gira (en el mismo lugar) hacia la derecha una distancia especificada por el argumento c. La rueda izquierda (visto el robot por atrás) gira hacia adelante, al mismo tiempo que la rueda derecha gira hacia atrás. El argumento c mide la distancia que debe recorrer cada rueda. Las unidades de c están dadas en décimas de vuelta del motor acoplado a las ruedas. El robot responde regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea, e inmediatamente se inicia el movimiento. La respuesta del robot no se espera a que termine el movimiento.
Ejemplos:	$d10 < \backslash n >$

Comando para parar los motores de las ruedas de tracción

Comando:	$p < \backslash n >$
Respuesta:	$p < \backslash n >$
Descripción:	Para el movimiento de las ruedas del robot.
Ejemplos:	$p < \backslash n >$

Comando para reportar el estado del robot

Comando:	$l < \backslash n >$
Respuesta:	$l < Ci >< sp >< Cd >< sp >< Ei >< sp >< Ed >< sp >$ $< Es >< \backslash n >$
	Donde: Ci, Cd son números en el rango [0,0, 65535,99]
	Ei, Ed y Es pueden tomar valor 0 ó 1
Descripción:	El robot responde a este comando informando los giros efectuados por las ruedas, tomando como referencia la posición del robot cuando recibió el último comando de movimiento: avanzar (comando a), retroceder (comando r), girar a la izquierda (comando i) o girar a la derecha (comando d).
	Ci y Cd se refieren al contador de giro de la rueda izquierda y derecha del robot (visto el robot desde atrás) respectivamente. Teniendo en cuenta que la resolución de los contadores de giro de los motores acoplados a las ruedas es de 1000 unidades por cada vuelta del rotor del motor, se reportan los giros efectuados con una precisión de centésimas de las unidades dadas a los comandos de movimientos: a, r ,i, d.
	Ei y Ed se refieren al estado actual de los motores de las ruedas de tracción izquierda y derecha respectivamente. Un valor de 1 indica que el correspondiente motor recibe energía de las baterias (y por lo tanto la rueda asociada está en movimiento) y un valor de 0 indica que el motor no recibe alimentación de la batería.
	Es indica el estado de operación del anillo de sonares. Un valor de 1 indica que la secuencia de disparos del anillo de sonares está en ejecución y un valor de 0 indica que los sonares no están disparando.
Ejemplos:	$l < \backslash n >$

Comando para esperar por el paro de los motores o nuevas lecturas del anillo de sonares

Comando:	$w < ms >< sp >< Fi >< sp >< Fd >< sp >< Fs >< \backslash n >$ Donde: ms es un número entre 0 y 65535. Fi, Fd y Fs pueden tomar valor 0 ó 1 Nota: El comando $w < ms >< \backslash n >$ es una forma compacta del comando $w < ms >< sp > 0 < sp > 0 < sp > 0 < \backslash n >$
Respuesta:	$w < \backslash n >$
Descripción:	El robot responde a este comando (enviando $w < \backslash n >$ por el puerto serial) sólo cuando estado de los motores de las ruedas corresponde a las indicaciones definidas por Fi (para el motor izquierdo) y Fd (para el motor derecho), así como a la indicación definida por Fs para el estado del anillo de sonares. Un valor de 1 para Fi o Fd corresponde a que el robot debe esperar hasta el motor correspondiente esté apagado, antes de enviar su respuesta por el puerto serial. Un valor de 0 indica que no se tome en cuenta el estado del motor correspondiente. Adicionalmente, el argumento $< ms >$ indica el tiempo máximo, en milisegundos, que debe esperar el robot antes de contestar. Es decir, transcurrido ese tiempo el robot enviará su respuesta por el puerto serial, aún cuando los motores de las ruedas sigan activos.
Ejemplos:	Espera a que paren ambos motores con un tiempo de espera máximo de 10 segundos: $w10000\ 1\ 1 < \backslash n >$. Espera por un período de 1 segundo: $w1000 < \backslash n >$.

22

Comando para definir la aceleración de los motores de las ruedas de tracción

Comando:	$k < Pi >< sp >< Pf >< sp >< Tc > \backslash n >$ Donde: Pi y Pf son números enteros en el rango $[0, 90]$ Tc es un número en el rango $[0, 65535]$
Respuesta:	$k < \backslash n >$
Descripción:	Este comando define los parámetros de aceleración de los motores que están acoplados a las ruedas de tracción, de forma de lograr un movimiento suave al inicio y final del recorrido del robot. Estos parámetros se consideran en la ejecución del siguiente comando de movimiento: avanzar (comando a), retroceder (comando r), girar a la izquierda (comando i) o girar a la derecha (comando d). Tomando en consideración que por cada vuelta del rotor del motor se generan 1000 unidades del contador de giro acoplado al rotor, experimentalmente se encontró que la velocidad del motor puede variar de 0 a 90 unidades de giro por milisegundo. Un valor de 0 representa el motor parado y un valor de 90 indica máxima velocidad. El argumento $< Pi >$ indica la velocidad deseada al inicio de ejecución del comando de movimiento (a,r,i ó d). La velocidad deseada del motor se incrementa una unidad cada $< Pi >$ milisegundos, hasta llegar a la velocidad máxima definida por $< Pf >$. De manera similar a este incremento de velocidad al inicio del movimiento, se efectúa una disminución de la velocidad al final del movimiento. El control de los motores de las ruedas se efectúa mediante un controlador proporcional integral fijando como metas las velocidades deseadas dadas por el control de aceleración descrito en el párrafo anterior.
Valor por omisión:	Al encender el robot, los valores por omisión son: $Pi = 5$, $Pf = 30$ y $Tc = 50$.
Ejemplos:	Arranque y frenado suave con velocidad pequeña de desplazamiento $k1\ 5\ 50 < \backslash n >$. Arranque y frenado rápido alcanzando velocidad máxima de los motores $k5\ 90\ 25 < \backslash n >$.

Comandos de definición de secuencia de disparo del anillo de sonares

Comando:	$S < d0 >< sp >< d1 >< sp > ... < d15 >< \backslash n >$ Donde: $d\#$ puede ser un número entero en el rango $[0, 15]$
Respuesta:	$S < \backslash n >$
Descripción:	Define la secuencia de disparos del anillo de sonares. Los sonares están numerados en sentido contrario a las manecillas del reloj (visto el robot por arriba). El sonar número 0 es el que está situado justo al frente del robot. $< d0 >$ indica el primer sonar que se dispara, $< d1 >$ el segundo sonar que se dispara y así sucesivamente hasta el último disparo definido. Se pueden definir una secuencia de hasta 16 disparos.
Valor por omisión:	Al encender el robot, la secuencia predefinida de disparos sonar es la siguiente: $S0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15 < \backslash n >$
Ejemplos:	Habilita el disparo solamente de tres sonares del frente del robot $S15\ 0\ 1 < \backslash n >$.

Comando de activación/desactivación del anillo de sonares

Comando:	$s < F >< \backslash n >$ Donde $< F >$ puede ser 0 o 1
Respuesta:	$s < \backslash n >$
Descripción:	Si $F = 1$ se activa en forma continua la secuencia de disparos del anillo de sonares establecida por el comando de la sección anterior. Si $F = 0$, se desactiva la secuencia de disparos de los sonares.
Ejemplos:	Activa los sonares $s1 < \backslash n >$.

Comando de definición de parámetros de control de disparos de sonares

Comando:	$s2 < Te >< sp >< Tr >< \backslash n >$ Donde: Te es un entero en el rango $[10, 65535]$ Tr es un entero en el rango $[10, 65535]$
Respuesta:	$s < \backslash n >$
Descripción:	Este comando fija la operación de los disparos de los sonares. Cuando un sonar dispara, emite 16 pulsos sonoros de una frecuencia de 49.4 Khz. Después del disparo, el sonar se pone en modo receptor para escuchar el eco. Internamente, el sonar amplifica las señales sonoras percibidas buscando un posible eco. La ganancia de amplificación de las señales va creciendo conforme el tiempo transcurre, permitiendo así que los ecos débiles puedan ser detectados por el sonar. Este aumento de ganancia es necesario si se toma en cuenta que la energía sonora disminuye rápidamente a medida que se aleja de la fuente transmisora. El argumento Te define el tiempo máximo en milisegundos, después del disparo del sonar, que se permite al sonar ponerse en modo receptor para escuchar un posible eco. Después de que se alcanza el tiempo Te, el sonar espera un tiempo Tr, antes de iniciar el siguiente disparo. Si el parámetro Te es pequeño sólo se permite al sonar escuchar ecos fuertes, ocasionados por obstáculos cercanos al robot. Con ello, el alcance de los sonares se disminuye, pero la posibilidad de escuchar ecos provenientes de sonares distintos al último que disparo, también disminuye. El tiempo Tr tiene por objeto habilitar al sonar para el siguiente disparo, así como esperar a que disipe la energía acústica del medio, evitando detectar falsos ecos. Con estos dos parámetros se puede controlar tanto la frecuencia de disparo de los sonares, el alcance permitido a los sonares y tratar de evitar el problema de cruzamiento de los disparos de los sonares.
Valor por omisión:	Al iniciar el robot, se tienen los siguientes parámetros por omisión: $s2\ 50\ 20 < \backslash n >$
Ejemplos:	Define una secuencia lenta de disparos que requiere 0.1 segundos por disparo: $s2\ 50\ 50 < \backslash n >$.

Comando de informe de lecturas de sonares

Comando:	$s < \backslash n >$
Respuesta:	$s < t0 >< sp >< t1 >< sp > ... < t15 >< \backslash n >$
Descripción:	Este comando informa de las mediciones realizadas por los sonares. La medición $t0$ corresponde al tiempo de recepción del eco, en unidades de 5.33 microsegundos, que midió el primer sonar de la secuencia de disparos definida por el comando S. La medición $t1$ corresponde al segundo sonar, y así sucesivamente para todos los sonares de la secuencia de disparo.
Ejemplos:	Para una secuencia de 16 sonares: $s < \backslash n >$. s1634 1779 417 1157 360 756 859 256 831 762 1048 607 261 651 230 1163 $< \backslash n >$

Comando de informe de estado de las baterías

Comando:	$t < \backslash n >$
Respuesta:	$t < Bat12 >< sp >< Bat24 >$ $< Bat12M >< sp >< Bat24M >< \backslash n >$
Descripción:	Este comando informa del voltaje suministrado por las baterías del robot en milivolts. $< Bat12 >$ y $< Bat24 >$ son los voltajes en milivolts suministrados por el par de baterías que alimentan los microcontroladores, los sonares y la tarjeta de control. $< Bat12M >$ y $< Bat24M >$ son los voltajes en milivolts suministrados por el par de baterías que alimentan los motores. El voltaje nominal de las baterías es de 12V, pero puede llegar hasta 13.5V. Un valor menor o igual a 11200mv (o 22400mv) indica baterías completamente descargadas. Operar el robot abajo de este límite, puede dañar las baterías.
Ejemplos:	$t < \backslash n >$. t12716 25394 12859 25837 $< \backslash n >$

A.2.2. Comandos para el sistema de giro inclinación (*pan-tilt*)

Comando para elevar el *pan-tilt*

Comando:	$e < c >< \backslash n >$
	Donde c es un número entero entre 0 y 100.
Respuesta:	$e < \backslash n >$
Descripción:	El *pan-tilt* se coloca en la posición de inclinación especificada por el argumento c. La posición es absoluta respecto a la posición inicial del *pan-tilt*. La posición inicial está localizada en 0 y es incializada cuando se enciende o resetea el robot. El robot responde regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea, e inmediatamente se inicia el movimiento. La respuesta del robot no se espera a que termine el movimiento.
Ejemplos:	$e10 < \backslash n >$

Comando para girar el *pan-tilt*

Comando:	$g < c >< \backslash n >$
	Donde c es un número entero entre 0 y 254.
Respuesta:	$g < \backslash n >$
Descripción:	El *pan-tilt* se coloca en la posición de giro especificada por el argumento c. La posición es absoluta respecto a la posición inicial del *pan-tilt*. La posición inicial está localizada en 0 y es incializada cuando se enciende o resetea el robot. El robot responde regresando por el puerto serial la letra del comando que recibió, seguida por el caracter de nueva línea, e inmediatamente se inicia el movimiento. La respuesta del robot no se espera a que termine el movimiento.
Ejemplos:	$g10 < \backslash n >$

Comando para parar los motores del *pan-tilt*

Comando:	$p < \backslash n >$
Respuesta:	$p < \backslash n >$
Descripción:	Para el movimiento de los motores de giro e inclinación del *pan-tilt*.
Ejemplos:	$p < \backslash n >$

Comando para esperar por el paro de los motores del *pan-tilt*

Comando:	$w < ms >< sp >< Fg >< sp >< Fe >< \backslash n >$
	Donde: ms es un número entre 0 y 65535.
	Fg, Fe y Fs pueden tomar valor 0 ó 1
	Nota: El comando $w < ms >< \backslash n >$ es una forma compacta del comando $w < ms >< sp > 0 < sp > 0 < \backslash n >$
Respuesta:	$w < \backslash n >$
Descripción:	El robot responde a este comando (enviando $w < \backslash n >$ por el puerto serial) sólo cuando estado de los motores del *pan-tilt* corresponde a las indicaciones definidas por Fg (para el motor de giro) y Fe (para el motor de inclinación). Un valor de 1 para Fg o Fe corresponde a que el robot debe esperar hasta el motor correspondiente esté apagado, antes de enviar su respuesta por el puerto serial. Un valor de 0 indica que no se tome en cuenta el estado del motor correspondiente.
	Adicionalmente, el argumento $< ms >$ indica el tiempo máximo, en milisegundos, que debe esperar el robot antes de contestar. Es decir, transcurrido ese tiempo el robot enviará su respuesta por el puerto serial, aún cuando los motores sigan activos.
Ejemplos:	Espera a que paren ambos motores con un tiempo de espera máximo de 10 segundos:
	$w10000\ 1\ 1 < \backslash n >$.
	Espera por un período de 1 segundo:
	$w1000 < \backslash n >$.

Comando de reporte de estado del *pan-tilt*

Comando:	$l < \backslash n >$
Respuesta:	$l < Cg >< sp >< Ce >< sp >< Eg >< sp >< Ee >< sp >< \backslash n >$ Donde: Cg, Ce son números en el rango $[0, 65535]$ Eg y Ee pueden tomar valor 0 ó 1
Descripción:	El robot responde a este comando informando los giros efectuados por los motores de giro e inclinación, tomando como referencia la posición del *pan-tilt* cuando recibió el último comando de movimiento: elevar (comando e), o girar (comando g). Cg y Ce se refieren al contador del giro e inclinación del *pan-tilt*, respectivamente. Teniendo en cuenta que la resolución de los contadores de giro de los motores acoplados a las ruedas es de 1000 unidades por cada vuelta del rotor del motor, se reportan los giros efectuados con una precisión de centésimas de las unidades dadas a los comandos de movimientos: g, y e. Eg y Ee se refieren al estado actual de los motores de giro y inclinación respectivamente. Un valor de 1 indica que el motor correspondiente recibe energía de las baterias (y por lo tanto el motr está en movimiento) y un valor de 0 indica que el motor no recibe alimentación de la batería.
Ejemplos:	$l < \backslash n >$

Comando para definir la aceleración de los motores del *pan-tilt*

Comandos:	$k < Pi >< sp >< Pf >< sp >< Tc > \backslash n >$ $K < Pi >< sp >< Pf >< sp >< Tc > \backslash n >$ Donde: Pi y Pf son números enteros en el rango $[0, 90]$ Tc es un número en el rango $[0, 65535]$
Respuestas:	$k < \backslash n >$ $K < \backslash n >$
Descripción:	El comando k define los parámetros de aceleración del motor de giro, el comando K define los parámetros de aceleración del motor de inclinación. Los parámetros se consideran en la ejecución del siguiente comando de movimiento: girar (comando g), o elevar (comando e). Tomando en consideración que por cada vuelta del rotor del motor se generan 1000 unidades del contador de giro acoplado al rotor, experimentalmente se encontró que la velocidad del motor puede variar de 0 a 90 unidades de giro por milisegundo. Un valor de 0 representa el motor parado y un valor de 90 indica máxima velocidad. El argumento $< Pi >$ indica la velocidad deseada al inicio de ejecución del comando de movimiento (g o e). La velocidad deseada del motor se incrementa una unidad cada $< Pi >$ milisegundos, hasta llegar a la velocidad máxima definida por $< Pf >$. De manera similar a este incremento de velocidad al inicio del movimiento, se efectúa una disminución de la velocidad al final del movimiento. El control de los motores del *pan-tilt* se efectúa mediante un controlador proporcional integral fijando como metas las velocidades deseadas dadas por el control de aceleración descrito en el párrafo anterior.
Valor por omisión:	Al encender el robot, los valores por omisión son: $Pi = 5$, $Pf = 30$ y $Tc = 50$.
Ejemplos:	Arranque y frenado suave con velocidad pequeña de desplazamiento $k1\ 5\ 50 < \backslash n >$. Arranque y frenado rápido alcanzando velocidad máxima de los motores $k5\ 90\ 25 < \backslash n >$.

A.3. Código para el control

La codificación del control para el robot y el *pan-tilt* se realizó en lenguaje "C" para el microcontrolador 68hcs12 bajo el proyecto 68hcs12 [Carrez03] de licencia GNU.

El robot tiene dos microcontroladores, uno controla los movimientos de desplazamiento y giro del robot, el otro controla el sistema de giro inclinación. Ambos microcontroladores fueron programados por separado, sin embargo, el código para el módulo de comunicación serial, así como el archivo que contiene el mapa de registros *ioRegsHcs12DP256.h* y el archivo de utilerias *util.h* son comúnes para los dos microcontroladores. En seguida se muestra el listado del código fuente en común para la comunicación serial.

```
/*
   serial.c
*/

#include "ioRegsHcs12DP256.h"

/* This assumes a 8 MHz clock. If not, you'll need to change the
   Baud rate enum.
 */

#define scimask  b(0,0,1,0,1,1,0,0) //RIE - SCI Interrupt enable ;RE - Receiver Enable

void ser0_init (unsigned short baud)
{
   SCI0BD = baud;
   /* enable transmitter and receiver and receiver interrupt */
   SCI0CR1 = 0;
   SCI0CR2 = scimask;
   SCI0SR1; //read register to clear flag RDRF
   SCI0DRL; //dummy read to flush receive buffer
}

#define TDRE 0x80
void ser0_putchar(char c)
{
   while ((SCI0SR1 & TDRE) == 0)
      ;
   SCI0DRL = c;
}

void ser0_puts(char s[])
{
   while(*s){
      ser0_putchar(*s);
      s++;
   }
}

void ser0_putnl()
{
  ser0_putchar('\n');
}

void ser0_putspace()
{
```

```
  ser0_putchar(' ');
}

#define NDIGITS 5
void ser0_putushort(unsigned short x)
{
   static byte d[NDIGITS];
   static int i;

   i = NDIGITS -1;
   do {
      d[i--] = x % 10;
      x /= 10;
   } while (x);
   for(i++; i < NDIGITS; i++)
      ser0_putchar(d[i] + '0');
}

volatile short g_command_pending;
volatile char g_command_line[256];
volatile int  g_command_len=0;

void  __attribute__ ((interrupt)) ISR_SerialInput0(void)
{
   asm("pshc\n pshd\n pshx\n pshy");
   static char c, new_command=0, i=0;

   SCI0SR1;      //read register to clear flag RDRF
   c = SCI0DRL;       //read receive buffer

   g_command_line[g_command_len++]=c;
   if(c=='\n'){
      g_command_line[g_command_len++]=0;
      g_command_pending=1;
   }

   asm("pulc\n puld\n pulx\n puly");
}
```

A continuación se listará el codigo correspondiente para cada microcontrolador.

A.3.1. Código para el control del robot

El código para el control del movimiento del robot está organizado en 5 archivos fuente: *control.c*, *command.c*, *isr.c*, *vectors.c* y *main.c*. En seguida se muestra el listado de cada uno de ellos.

```
/// control.c ///

#include "ioRegsHcs12DP256.h"
#include "serial.h"

//       PORT ASSIGNMENTS for this program for adapt9S12Dp256 ROCA-MOBILE-2

//                                              pata            Conector
//       PT0     Echo Sonar      ;I             13              (H1)    4.7kohms
//       PT1     MIzqI           ;I             12  4.7Kohms  //Verde
```

```
//       PT2     MIzqA           ;I             11                      4.7kohms //Amarillo
//       PT3     MDerA           ;I             10                      4.7kohms //Amarillo
//       PT4     MDerI           ;I              9                      4.7kohms //Verde
//       PT5     MIzqB           ;I              8  4.7kohms //Azul
//       PT6     MderB           ;I              7  4.7kohms //Azul
//               (red led on the board)
//       PT7     Beep            ;O              6 4.7kohms + zumbador

//       PP0     MIzqDir         ;O             21              (H1)
//       PP1     MIzqBrake       ;O             20
//       PP2     MDerDir         ;O             19
//       PP3     MDerBrake       ;O             18
//       PP4     InitSO          ;O             17
//       PP5     BinhSO          ;O             16
//       PP6     MIzqPWM         ;O             15
//       PP7     MDerPWM         ;O             14

//       PH0     Line A Sonar    ;O             42              (H1)
//       PH1     Line B Sonar    ;O             41
//       PH2     Line C Sonar    ;O             40
//       PH3     Line D Sonar    ;O             39
//       PH4     Spare           ;I             38
//       PH5     Spare           ;I             37
//       PH6     Spare           ;I             36
//       PH7     Spare           ;I             35

// Masks
#define mT_EchoSonar    b(0,0,0,0,0,0,0,1) // PORTT
#define mT_MIzqI        b(0,0,0,0,0,0,1,0)
#define mT_MIzqA        b(0,0,0,0,0,1,0,0)
#define mT_MDerA        b(0,0,0,0,1,0,0,0)
#define mT_MDerI        b(0,0,0,1,0,0,0,0)
#define mT_MIzqB        b(0,0,1,0,0,0,0,0)
#define mT_MDerB b(0,1,0,0,0,0,0,0)
#define mT_Beep b(1,0,0,0,0,0,0,0)

#define mP_MIzqDir      b(0,0,0,0,0,0,0,1) // PORTP
#define mP_MIzqBrake    b(0,0,0,0,0,0,1,0)
#define mP_MDerDir      b(0,0,0,0,0,1,0,0)
#define mP_MDerBrake    b(0,0,0,0,1,0,0,0)
#define mP_InitSO       b(0,0,0,1,0,0,0,0)
#define mP_BinhSO       b(0,0,1,0,0,0,0,0)
#define mP_MIzqPWM b(0,1,0,0,0,0,0,0)
#define mP_MDerPWM b(1,0,0,0,0,0,0,0)

#define mH_SonarDirA    b(0,0,0,0,0,0,0,1)       // PORTH
#define mH_SonarDirB    b(0,0,0,0,0,0,1,0)
#define mH_SonarDirC    b(0,0,0,0,0,1,0,0)
#define mH_SonarDirD    b(0,0,0,0,1,0,0,0)

volatile extern unsigned char g_left_vel, g_right_vel;
volatile extern unsigned char real_time_flag;
volatile unsigned char  g_left_vel, g_right_vel;
volatile static short des_left_vel_clicks;  // Desired vel, clicks/ms
volatile static short des_right_vel_clicks;  // Desired vel, clicks/ms
volatile static short des_bias_clicks;  // Desired bias, clicks/ms
volatile static short max_left_vel_clicks;  // Initial desired bias, clicks/ms
volatile static short max_right_vel_clicks;  // Initial desired bias, clicks/ms
volatile static short max_bias_clicks; // Initial desired bias, clicks/ms
volatile static unsigned char left_pwm, right_pwm;  //Power to motors
volatile static short  integral=0; // Integral of velocity difference
volatile long k_integral = 1; // Integral error gain
volatile long k_pro = 50; // Proportional gain
```

```
volatile short edo_left_motor=0;
volatile short edo_right_motor=0;
volatile short sonar_wait;
volatile short k_vel_ini = 5;
volatile short k_vel_end = 30;
volatile short k_ms_inc  = 50;

volatile unsigned long clicks_stop, clicks_break, total_left_clicks, total_right_clicks;
volatile static unsigned short acelerando, frenando;
volatile static short timer_ac;

static void inline set_left_pwm()
{
   PWMDTY6 = left_pwm;
}

static void inline set_right_pwm()
{
   PWMDTY7 = right_pwm;
}

static void m_izq_forward()
{
   PORTP |= mP_MIzqDir;
   PORTP &= ~mP_MIzqBrake;
}

static void m_izq_backward()
{
   PORTP &= ~mP_MIzqDir;
   PORTP &= ~mP_MIzqBrake;
}

static void m_izq_stop()
{
   PORTP |= mP_MIzqDir;
   PORTP |= mP_MIzqBrake;
   left_pwm=255U;
   set_left_pwm();
}

static void m_der_forward()
{
   PORTP &= ~mP_MDerDir;
   PORTP &= ~mP_MDerBrake;
}

static void m_der_backward()
{
   PORTP |=  mP_MDerDir;
   PORTP &= ~mP_MDerBrake;
}

static void m_der_stop()
{
   PORTP |= mP_MDerDir;
   PORTP |= mP_MDerBrake;
   right_pwm=255U;
   set_right_pwm();
}

void set_velocity()
{
```

```
   if(!max_left_vel_clicks && !max_right_vel_clicks && !max_bias_clicks){
      m_izq_stop();
      m_der_stop();
      edo_left_motor=edo_right_motor=0;
      acelerando = 0;
      frenando   = 0;
      return;
   }
   left_pwm=right_pwm=0U;          // Initial power to motors
   edo_left_motor=edo_right_motor=1;
   integral=0;
   set_left_pwm();
   set_right_pwm();
   des_left_vel_clicks = abs(max_left_vel_clicks);
   des_right_vel_clicks = abs(max_right_vel_clicks);
   des_bias_clicks = max_bias_clicks;
   if(max_left_vel_clicks > 0){
      m_izq_forward();
   } else {
      m_izq_backward();
   }
   if(max_right_vel_clicks > 0) {
      m_der_forward();
   } else {
      m_der_backward();
   }
}

void m_stop()
{
   edo_left_motor = edo_right_motor=0;
   max_left_vel_clicks = max_right_vel_clicks = max_bias_clicks = 0;
   set_velocity();
}

void stop()
{
   ser0_puts("p\n");
   m_stop();
}

/* *** SET_VEL COMMAND *** */
void set_vel(int num_param, unsigned short param[])
{
   ser0_puts("v\n");
   if(num_param >=2) {
      max_left_vel_clicks  = param[0] - 100;
      max_right_vel_clicks  = param[1] - 100;
   }
   if(num_param == 3)
      max_bias_clicks = param[2] - 100;
   else
      max_bias_clicks = 0;
   set_velocity();
   clicks_stop = 0;
   acelerando = 0;
   frenando  = 0;
}
/* *** END SET_VEL *** */

void set_ac(short num_params, unsigned short param[])
{
   ser0_puts("k\n");
```

```
   k_vel_ini = param[0];
   k_vel_end = param[1];
   k_ms_inc  = param[2];
}

void set_params(short num_params, unsigned short param[])
{
   if(num_params == 0)
      clicks_stop = 0;
   else
      clicks_stop = (long) param[0] * (long) 100;
   total_left_clicks = total_right_clicks =0;
   clicks_break =  clicks_stop/2;
   timer_ac = k_ms_inc;
   max_bias_clicks = 0;
   acelerando = 1;
   frenando = 0;
}

void forward(short num_params, unsigned short param[])
{
   ser0_puts("a\n");
   set_params(num_params, param);
   max_left_vel_clicks = max_right_vel_clicks = k_vel_ini;
   set_velocity();
}

void backward(short num_params, unsigned short param[])
{
   ser0_puts("r\n");
   set_params(num_params, param);
   max_left_vel_clicks = max_right_vel_clicks = - k_vel_ini;
   set_velocity();
}

void turn_left(short num_params, unsigned short param[])
{
   ser0_puts("i\n");
   set_params(num_params, param);
   max_left_vel_clicks = -k_vel_ini;
   max_right_vel_clicks = k_vel_ini;
   set_velocity();
}

void turn_right(short num_params, unsigned short param[])
{
   ser0_puts("d\n");
   set_params(num_params, param);
   max_left_vel_clicks =   k_vel_ini;
   max_right_vel_clicks = -k_vel_ini;
   set_velocity();
}

/* SPEED_CONTROL */

static unsigned char limit_range(unsigned char pwm, long error)
{
   static long new_pwm;

   new_pwm = pwm + error;
   if(new_pwm > 255)
      new_pwm = 255;
   else if(new_pwm < 0)
```

```
      new_pwm = 0;
   return (unsigned char)new_pwm;
}

void alter_left_power(long error )
{
   left_pwm = limit_range(left_pwm,error);
   set_left_pwm();
}

void alter_right_power(long error )
{
   right_pwm = limit_range(right_pwm,error);
   set_right_pwm();
}

void speed_control()
{
   static long int left_vel, right_vel, integral_error, left_error, right_error,
               vel_clicks, bias_clicks;

   if(!edo_left_motor && !edo_right_motor)
      return;
   left_vel  = g_left_vel;
   right_vel = g_right_vel;

   total_left_clicks += left_vel; //Update total clicks
   total_right_clicks += right_vel;

   integral +=  left_vel + des_bias_clicks - right_vel;
   integral_error  = k_integral * integral / 100;
   left_error  = k_pro*(des_left_vel_clicks - left_vel  - integral_error)/100;
   right_error = k_pro*(des_right_vel_clicks - right_vel + integral_error)/100;
   alter_left_power (left_error );
   alter_right_power(right_error);
}

void ac_control()
{
   if(timer_ac){
      timer_ac--;
      return;
   }
   if(acelerando){
      if(g_left_vel + 3 >= k_vel_end){ // reach the max velocity
          clicks_break = clicks_stop - total_left_clicks;
          acelerando = 0;
      }
      if(des_left_vel_clicks < k_vel_end) {
         des_left_vel_clicks++;
 des_right_vel_clicks++;
      }
   }
   if (frenando && des_left_vel_clicks) {
         des_left_vel_clicks--;
         des_right_vel_clicks--;
   }
   timer_ac = k_ms_inc;
}

void check_stop_motors(void)
{
   if(clicks_stop && total_left_clicks >= clicks_stop && total_right_clicks >= clicks_stop)
```

```
      m_stop();
   if(edo_left_motor && !frenando && total_left_clicks >= clicks_break) {
      acelerando = 0;
      frenando   = 1;
   }
}

#define MAX_SONARS 16
volatile static unsigned char sonar_sec[MAX_SONARS]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
volatile static unsigned short sonar[MAX_SONARS];
volatile static short sonar_num = MAX_SONARS;
volatile static short sonar_edo = 0; // 1 means sonars are firing
volatile static short sonar_index = 0;
volatile static short sonar_waiting_init=0;
volatile static short sonar_waiting_echo=0;
volatile static short sonar_waiting_relax=0;
volatile static short sonar_ready2fire=0;
volatile unsigned short sonar_time_start, sonar_time_end, sonar_time;
volatile static unsigned short sonar_cycles = 0;
volatile static unsigned short sonar_timeout_init = 800;   //initial timeout in microseg to
//avoid false echos
volatile static unsigned short sonar_max_time_echo = 50;     //timeout in ms to wait for
//an echo
volatile static unsigned short sonar_time_relax = 20;       //time in ms to start another measurement

static void inline sonar_init_off()
{
   PORTP &= ~mP_InitS0;
   PORTP &= ~mP_BinhS0;
}

void sonar_report()
{
   static short i;
   for(i=0; i < sonar_num; i++){
      ser0_putushort(sonar[i]);
      ser0_putnl();
   }
   sonar_wait = sonar_num + 1;
}

void sonar_stop()
{
  sonar_edo=0;
  ser0_putnl();
}

void sonar_start()
{
   static short i;

   sonar_cycles=0;
   sonar_index=0;
   sonar_waiting_echo = 0;
   sonar_waiting_relax = 0;
   for(i=0; i<sonar_num; i++)
      sonar[i] = 65535u;
   sonar_init_off();
   sonar_ready2fire=1;
   sonar_edo=1;
   sonar_wait = sonar_num;
   ser0_putnl();
}
```

```
void sonar_def_param(int num_params, unsigned short param[])
{
   sonar_timeout_init  = 1;      //initial timeout in microseg to avoid false echos
   sonar_max_time_echo = param[1];   //timeout in microseg to wait for an echo
   sonar_time_relax    = param[2];      //time to start another measurement
   ser0_putnl();
}

void sonar_report_cycles()
{
   ser0_putushort(sonar_cycles);
   ser0_putnl();
}

void sonar_cmd(int num_params, unsigned short param[])
{
   ser0_putchar('s');
   if(!num_params)
      sonar_report();
   else if (param[0] == 0)
      sonar_stop();
   else if (param[0] == 1)
      sonar_start();
   else if (param[0] == 2)
      sonar_def_param(num_params,param);
   else
      sonar_report_cycles();
}

void sonar_def_sec(int num_params, unsigned short param[])
{
   static short i;
   ser0_puts("S\n");
   sonar_num = num_params;
   for(i=0; i < num_params; i++)
      sonar_sec[i] = param[i];
}

static void sonar_fire(unsigned char num_sonar)
{
   PORTH = num_sonar; //set the sonar address
   sonar_time_start = sonar_time_end = TCNT;
   PORTP |= mP_InitS0;
   PORTP &= ~mP_BinhS0;
}

static void inline sonar_binh_on()
{
   PORTP |= mP_BinhS0;
}

static void inline sonar_result()
{
   sonar_time =sonar_time_end - sonar_time_start;
   if(sonar_time)
      sonar[sonar_index] = sonar_time;
   else
      sonar[sonar_index]=65535u;
}

void sonar_demon()
{
```

```
   static unsigned char sonar_active;

   if(sonar_edo){
      if(sonar_ready2fire){
         sonar_active = sonar_sec[sonar_index];
         sonar_fire(sonar_active);
         sonar_ready2fire=0;
         sonar_waiting_init=1;
      } else if(sonar_waiting_init) {
         sonar_waiting_init=0;
         sonar_waiting_echo=sonar_max_time_echo;
         sonar_binh_on();
      } else if(sonar_waiting_echo){
         sonar_waiting_echo--;
         if(!sonar_waiting_echo){
            sonar_init_off();
            sonar_result();
            sonar_waiting_relax = sonar_time_relax;
         }
      } else if(sonar_waiting_relax)
         sonar_waiting_relax--;
      else {
         sonar_ready2fire=1;
         if(++sonar_index == sonar_num){
             sonar_index = 0;
     sonar_cycles++;
         }
         if(sonar_wait)
            sonar_wait--;
      }
   }
}

static void report_clicks(long total_clicks)
{
   static long turns, clicks, cien=100;
   turns  = total_clicks / cien;
   clicks = total_clicks - turns * cien;
   ser0_putushort((unsigned short) turns);
   ser0_putchar('.');
   if(clicks < 10)
     ser0_putchar('0');
   ser0_putushort((unsigned short) clicks);
}

void report_status(void)
{
   ser0_putchar('l');
   report_clicks(total_left_clicks);
   ser0_putspace();
   report_clicks(total_right_clicks);
   ser0_putspace();
   ser0_putushort(edo_left_motor);
   ser0_putspace();
   ser0_putushort(edo_right_motor);
   ser0_putspace();
   ser0_putushort(sonar_edo);
   ser0_putnl();

}

void beep_on()
{
```

```
   PORTT |= mT_Beep;
}

void beep_off()
{
   PORTT &= ~mT_Beep;
}

void beep(int num_params, short param[])
{
   ser0_puts("z\n");
   if(param[0])
      beep_on();
   else
      beep_off();
}

static void inline get_vel()
{

#define ICLAT b(0,0,0,1,0,0,0,0)

   MCCTL |= ICLAT;      //Force the contents of 8-bit accumulators to be latched into their
                        //associated holding registers
   g_left_vel=PA2H;
   g_right_vel=PA3H;
}

void check_rtf(void)
{
   if(real_time_flag) {
      get_vel();
      speed_control();
      ac_control();
      sonar_demon();
      real_time_flag = 0;
   }
}

void init_control()
{
   DDRP = 0xff;         //all are output
   PORTP = 0;

   DDRH = b(0,0,0,0,1,1,1,1);
   PORTH = 0;
   DDRT = b(1,0,0,0,0,0,0,0);
   PORTT = 0;

   //PWM

   PWME |= mP_MIzqPWM | mP_MDerPWM;      // Pulse Width Channel 6 & 7 Enable (use Clock B)
   PWMPRCLK = b(0,0,1,0,0,0,0,0);       //Clock B=Bus clock/4
                                        //1/(1/24e6 * 4 *256))=23437Hz
   PWMPOL = 0xff; // PWM outputs are high at the beginning of the period, then goes
// low when the duty count is reached

   //Initialize Analog To Digital
   ATD0CTL2 = 0x80; //enable ATD
   ATD0CTL3 = 0x40;
   ATD0CTL4 = 0x60; //Select Sample rate
   ATD0CTL5 = 0xb0;   //Select 8 channel mode, Continuous scan
```

```
   //ECT

   ICPAR |= b(0,0,0,0,1,1,0,0); //Pulse Acumulator 3 and 2 are enabled
   DLYCT |= b(0,0,0,0,0,0,0,1); //Delay Counter = 256 bus clock cycles
   TIOS  |= b(1,0,0,0,0,0,0,0); //All channels are input capture but 7
   TCTL4 |= b(1,1,1,1,0,0,0,1); //Input Capture Edge Control
//0,1=captue on rising edges only (sonar echo)
//1,1=Capture on any edge, rising or falling (encoders) T2 &T3
   ICSYS |= b(0,0,0,0,0,1,1,1); //PACMX=1 When the 8-bit pulse accumulator has reached the value
//$FF, it will no be incremented further.
//BUFEN=1 Input Capture and pulse accumulators registers are enabled
//LATQ=1 Latch Mode of Input Capture is enabled

   TIE   =  b(0,0,0,0,0,0,0,1); //Timer Interrupt Enable Register (Sonar Interrupt)
   TSCR2 =  b(0,0,0,0,0,1,1,1); //Timer Prescaler = Bus clock /128 =5.33 microseg
   TSCR1 =  b(1,0,0,0,0,0,0,0); //TEN=1 => TimerENable = 1
   TFLG1 |= b(0,0,0,0,0,0,0,1); //Clear Sonar Interrupt Flag
   MCCNT = 0;

   max_left_vel_clicks = max_right_vel_clicks = max_bias_clicks=0;
   set_velocity();
}

/*************/
/* command.c */
/*************/

#include  <stdio.h>
#include  <string.h>
#include  "ioRegsHcs12DP256.h"
#include  "serial.h"
#include  "control.h"

volatile extern unsigned short g_timer;
volatile extern unsigned short g_ad2;
volatile extern unsigned short g_ad3;
volatile extern unsigned short g_ad4;
volatile extern unsigned short g_ad5;
volatile extern unsigned char  g_left_vel;
volatile extern unsigned char  g_right_vel;

volatile extern unsigned long  k_pro;
volatile extern unsigned long  k_integral;

/* *** WAIT operation **** */
static unsigned short edo_wait=0;
static short wait_for_left_motor, wait_for_right_motor, wait_for_sonar;
extern short edo_left_motor, edo_right_motor, sonar_wait;

static void inline wait(short num_params, unsigned short param[])
{
   g_timer = param[0];
   if(num_params == 4) {
      wait_for_left_motor = param[1];
      wait_for_right_motor = param[2];
      wait_for_sonar = param[3];
   } else {
      wait_for_left_motor  = 0;
      wait_for_right_motor = 0;
      wait_for_sonar = 0;
   }

   edo_wait=1;
```

```
}

static void inline check_end_wait()
{
  if(edo_wait){
     edo_wait=0;
     if(wait_for_left_motor  && edo_left_motor)
        edo_wait=1;
     if(wait_for_right_motor && edo_right_motor)
        edo_wait=1;
     if(wait_for_sonar       && sonar_wait)
        edo_wait=1;
     if((!wait_for_left_motor && !wait_for_right_motor && !wait_for_sonar) || edo_wait)
         edo_wait = g_timer;
     if(!edo_wait)
        ser0_puts("w\n");
  }
}
/* *** End WAIT operation **** */

/* *** QUERY operation **** */
void query(void)
{
   ser0_puts("sonar_wait=");
   ser0_putushort((unsigned short) sonar_wait);
   ser0_putnl();
}
/* *** End QUERY operation **** */

/* *** bateries operation **** */
void bateries(void)
{
   static unsigned short bat12,bat24,bat12M,bat24M;

#define ROBOT2 1
#ifdef ROBOT1
   bat12  = (g_ad2 * (127200L/639)) / 10;
   bat24  = (g_ad3 * (259000L/691)) / 10;
   bat12M = (g_ad4 * (130100L/691)) / 10;
   bat24M = (g_ad5 * (261000L/696)) / 10;
#else
   bat12  = (g_ad2 * (119000L/639)) / 10;
   bat24  = (g_ad3 * (247000L/632)) / 10;
   bat12M = (g_ad4 * (126000L/691)) / 10;
   bat24M = (g_ad5 * (254000L/688)) / 10;
#endif
   ser0_putchar('t');
   ser0_putushort(bat12);
   ser0_putspace();
   ser0_putushort(bat24);
   ser0_putspace();
   ser0_putushort(bat12M);
   ser0_putspace();
   ser0_putushort(bat24M);
   ser0_putnl();
}
/* *** End QUERY operation **** */

void check_end_commands(void)
{
   check_end_wait();
   check_stop_motors();
}
```

```
void delay()
{
   static unsigned short i;
   for(i=0; i < 30000; i++)
      ;
}

void print_help()
{
   ser0_puts("Comandos disponibles:\n\n");

   ser0_puts("? Imprime este mensaje de ayuda\n"); delay();
   ser0_puts("* Regresa un asterisco\n");delay();
   ser0_puts("a<p1> Avanza p1 unidades\n");delay();
   ser0_puts("d<p1> Gira a la derecha p1 unidades\n");delay();
   ser0_puts("i<p1> Gira a la izquierda p1 unidades\n");delay();
   ser0_puts("k<pi> <pf> <ms> Define la aceleracion de los motores de las ruedas\n");delay();
   ser0_puts("l Reporta el estado del robot\n");delay();
   ser0_puts("p Paro de emergencia de los motores de las ruedas\n");delay();
   ser0_puts("r<p1> Retrocede p1 unidades\n");delay();
   ser0_puts("s0 Apaga el anillo de sonares\n");delay();
   ser0_puts("s1 Inicia los disparos del anillo de sonares\n");delay();
   ser0_puts("s2 <te> <tr> Fija parametros de disparo del anillo de sonares\n");delay();
   ser0_puts("s3 Regresa el numero de ciclos que han disparado los sonares\n");delay();
   ser0_puts("S <s0> <s1> ... Define la secuencia de disparo del anillo de sonares\n");delay();
   ser0_puts("t Regresa el voltaje de las baterias en milivolts\n");delay();
   ser0_puts("w Espera por paro de motores de las ruedas y el sonar\n");delay();
   ser0_puts("z0 Desactiva el zumbador al frente del robot\n");delay();
   ser0_puts("z1 Activa el zumbador al frente del robot\n");delay();
}

void do_command(char command, int num_params, unsigned short param[])
{
   switch (command) {
      case 'w': wait(num_params, param);  break;
      case 'q': query();  break;
      case 'v': set_vel(num_params,param);  break;
      case 'p': stop();  break;
      case 't': bateries(); break;
      case 'k': set_ac(num_params,param); break;
      case 'a': forward(num_params,param); break;
      case 'r': backward(num_params,param); break;
      case 'i': turn_left(num_params,param); break;
      case 'd': turn_right(num_params,param); break;
      case 's': sonar_cmd(num_params,param); break;
      case 'S': sonar_def_sec(num_params,param);break;
      case 'l': report_status(); break;
      case 'z': beep(num_params,param); break;
      case '*': ser0_puts("*\n"); break;
      case '?': print_help(); break;
   }
   check_end_commands();
}


/*************/
 *  isr.c
/*************/

#include "ioRegsHcs12DP256.h"

#include  "isr.h"
```

```
#include  "control.h"
#include  "serial.h"

void  __attribute__ ((interrupt)) ISR_Empty(void)
{
}

volatile unsigned short g_timer=0;
volatile unsigned short g_ad2=0;
volatile unsigned short g_ad3=0;
volatile unsigned short g_ad4=0;
volatile unsigned short g_ad5=0;
volatile unsigned char  real_time_flag=0;

#define RTIF            b(1,0,0,0,0,0,0,0)
#define SCFflag         b(1,0,0,0,0,0,0,0)       //SCF - Sequence Complete flag

/* Real Time Interrupt is running each milisecond */
void  __attribute__ ((interrupt)) ISR_RealTimeInt(void)
{
   if(g_timer)
      g_timer--;

   while(!(ATD0STATH & SCFflag));       //Loop here until SCF of ATD is set
   g_ad2=ATD0DR2H; //save ATD
   g_ad3=ATD0DR3H;
   g_ad4=ATD0DR4H;
   g_ad5=ATD0DR5H;

   real_time_flag = 1;

   CRGFLG |= RTIF;      //clear flag

}

extern unsigned short sonar_time_end;

/* Interrupt from the sonar echo */
void  __attribute__ ((interrupt)) ISR_SonarEcho(void)
{
   asm volatile ("pshc\n pshd\n pshx\n pshy");
   TFLG1 |= b(0,0,0,0,0,0,0,1);      //clear sonar flag
   sonar_time_end = TCNT;

   asm volatile ("pulc\n puld\n pulx\n puly");
}


/*************/
   main.c
/*************/

#include  <stdio.h>
#include  <string.h>
#include  "ioRegsHcs12DP256.h"
#include  "serial.h"
#include  "command.h"
#include  "control.h"

int                main(void);

extern char             _start_of_data;
extern char             _end_of_data;
```

```
extern const char       _start_of_init;

unsigned short int        start = 0x0000;
unsigned short int        end   = 0xffff;

#define BAUD57600  26      //(baud) 19200*3 baud with MCLK=24Mhz
#define BAUD28800  52      //(baud) 9600*3 baud with MCLK=24Mhz
#define BAUD14400 104      //(baud) 4800*3 baud with MCLK=24Mhz
#define BAUD9600 156  //(baud) 156 = 208 * 7200/9600 with MCLK=24Mhz
#define BAUD7200 208      //(baud) 2400*3 baud with MCLK=24Mhz

static void inline disableCOP(void)
{
   COPCTL = 0x00; //Disable COP
   COPCTL = 0x00;
}

#define OscClk 16000000 // Crystal or oscillator frequency
#define Eclock   24000000 // final E-clock frequency (PLL)
#define RefClock  8000000 // frequency used by the PLL to generate E-clock
#define REFDVVal (OscClk/RefClock)-1  //value for REFDV Register
#define SYNRVal (Eclock/RefClock)-1  //value for SYNR Register
static void inline rise24Mhz(void)
{
   REFDV = REFDVVal; //PLLCLK = 24MHz
   SYNR  = SYNRVal;
   asm("nop\n" "nop\n" "nop\n" "nop\n"); //nops required for bug
#define LOCK 0x08
   while(!(CRGFLG & LOCK)) //wait here till the PLL is locked
      ;
#define PLLSEL 0x80
   CLKSEL |= PLLSEL;  //System clocks are derived from PLLCLK

}

#define rtimask         b(0,1,0,0,0,0,0,1) //(1/16Mhz)2*2^13=1.024 msec interrupt
#define RTIF            b(1,0,0,0,0,0,0,0)
#define RTIE            b(1,0,0,0,0,0,0,0)
static void inline set_rti()
{
   RTICTL = rtimask;  //Initialize Real Time Interrupt rate
   CRGFLG |= RTIF; //clear flag
   CRGINT |= RTIE;  //Enable RTI
}

volatile extern short g_command_pending;
volatile extern short g_command_len;
volatile extern char g_command_line[];
volatile extern unsigned short g_timer;

static char command;
static short num_params;
static unsigned short param[20];

static decode_command()
{
   static int inicio_num, i;
   static char c;

   command=g_command_line[0];
   num_params=0;
   param[0]=0;
   inicio_num=0;
```

```
   for(i=1; (c=g_command_line[i]); i++) {
      if(c=='\r')
         continue;
      if(c==' ' || c == '\n') {
         param[++num_params]=0;
      } else {
         inicio_num=1;
         param[num_params] *= 10;
 param[num_params] += c - '0';
      }
   }
   if(!inicio_num)
      num_params--;
   g_command_len=0;
}

int  main()
{
   static int i;
   asm("sei"); //Disable Any interrupts

   disableCOP(); //Disable COP
   rise24Mhz(); //PLL Clock to 24Mhz
   memcpy(&_start_of_data, &_start_of_init, &_end_of_data - &_start_of_data);
   ser0_init(BAUD9600); //Baudrate is 9600
   set_rti();
   init_control();

   asm("cli");                  //unmask interrupts

   ser0_puts("Robot ROCA-MOBILE-2 v1.0 listo\n");

   while(1){
     if(g_command_pending){
       g_command_pending=0;
       decode_command();
       do_command(command,num_params,param);
       g_command_pending=0;
     }
     check_rtf();
     check_end_commands();
  }

   return  0;
}

/*************/
 *  vectors.c      vector table for the 68hc12
/*************/

#include  "isr.h"

extern void  ISR_Empty(void);
extern void  ISR_RTI(void);
extern void  ISR_SerialInput0(void);
extern void  ISR_RealTimeInt(void);
extern void  ISR_SonarEcho(void);
extern void  _start(void);

/*
 *  This is not a routine but a table that the linker can use to assign
 *  vectors to the various interrupt service routines (ISRs).  This
 *  table must ultimately appear in the 68hc12's vector area starting
```

```
 *  at 0xef8c.
 */

void  ( * const vector[])()  =  {
ISR_Empty,  // 0xEF8C
ISR_Empty,// 0xEF8E
ISR_Empty,// 0xEF90
ISR_Empty,// 0xEF92
ISR_Empty,// 0xEF94
ISR_Empty,// 0xEF96
ISR_Empty,// 0xEF98
ISR_Empty,// 0xEF9A
ISR_Empty,// 0xEF9C
ISR_Empty,// 0xEF9E
ISR_Empty,// 0xEFA0
ISR_Empty,// 0xEFA2
ISR_Empty,// 0xEFA4
ISR_Empty,// 0xEFA6
ISR_Empty,// 0xEFA8
ISR_Empty,// 0xEFAA
ISR_Empty,// 0xEFAC
ISR_Empty,// 0xEFAE
ISR_Empty,// 0xEFB0
ISR_Empty,// 0xEFB2
ISR_Empty,// 0xEFB4
ISR_Empty,// 0xEFB6
ISR_Empty,// 0xEFB8
ISR_Empty,// 0xEFBA
ISR_Empty,// 0xEFBC
ISR_Empty,// 0xEFBE
ISR_Empty,// 0xEFC0
ISR_Empty,// 0xEFC2
ISR_Empty,// 0xEFC4
ISR_Empty,// 0xEFC6
ISR_Empty,// 0xEFC8 Pulse Accumulator B Overflow
ISR_Empty,// 0xEFCA
ISR_Empty,// 0xEFCC Port H
ISR_Empty,// 0xEFCE Port J
ISR_Empty,// 0xEFD0 ATD1
ISR_Empty,// 0xEFD2 ATD0
ISR_Empty,// 0xEFD4 SCI1
ISR_SerialInput0,// 0xEFD6 SerInputInt0 (SCI0)
ISR_Empty,// 0xEFD8
ISR_Empty,// 0xEFDA Pulse accumulator input edge
ISR_Empty,// 0xEFDC Pulse accumulator A overflow
ISR_Empty,// 0xEFDE Enhanced Capture Timer overflow
ISR_Empty,// 0xEFE0 Enhanced Capture Timer channel 7
ISR_Empty,// 0xEFE2 Enhanced Capture Timer channel 6
ISR_Empty,// 0xEFE4 Enhanced Capture Timer channel 5
ISR_Empty,// 0xEFE6 Enhanced Capture Timer channel 4
ISR_Empty,// 0xEFE8 Enhanced Capture Timer channel 3
ISR_Empty,// 0xEFEA Enhanced Capture Timer channel 2
ISR_Empty,// 0xEFEC Enhanced Capture Timer channel 1
ISR_SonarEcho,// 0xEFEE Enhanced Capture Timer channel 0
ISR_RealTimeInt,// 0xEFF0 Real Time Interrupt
ISR_Empty,// 0xEFF2
ISR_Empty,// 0xEFF4
ISR_Empty,// 0xEFF6
ISR_Empty,// 0xEFF8
ISR_Empty,// 0xEFFA
ISR_Empty,// 0xEFFC
_start // 0xEFFE Reset
};
```

A.3.2. Código para el control del *pan-tilt*

La codificación para el control del *pan-tilt* también está organizado en 5 archivos. En seguida se muestra el listado correspondiente.

```
/*************/
/* control.c */
/*************/

#include "ioRegsHcs12DP256.h"
#include "serial.h"
#include "control.h"
#include "main.h"
#include "util.h"

#define DEBUG 1

//       PORT ASSIGNMENTS for this program for adapt9S12Dp256   (robot_pant_tilt)

//                                                 pata            Conector
//       PT0     MGiroI          ;I                13              (H1)
//       PT1     MElevI          ;I                12 4.7kohms
//       PT2     MGiroA          ;I                11                      4.7kohms
//       PT3     MElevA          ;I                10                      4.7kohms
//       PT4     MGiroB          ;I                 9                      4.7kohms
//       PT5     MElevB          ;I                 8 4.7kohms
//       PT6     MGiroTopeDer    ;I                 7 4.7kohms
//       PT7     MGiroTopeIzq    ;I                 6

//       PP0     MGiroDir        ;O                21              (H1)
//       PP1     MGiroBrake      ;O                20
//       PP2     MElevDir        ;O                19
//       PP3     MElevBrake      ;O                18
//       PP4     -               ;O                17
//       PP5     MGiroPWM        ;O                16
//       PP6     -          ;O              15
//       PP7     MElevPWM        ;O                14

//       PH0     MElevTopeInf    ;I                42              (H1)
//       PH1     Spare           ;I                41
//       PH2     Spare           ;I                40
//       PH3     Spare           ;I                39
//       PH4     Spare           ;I                38
//       PH5     Spare           ;I                37
//       PH6     Spare           ;I                36
//       PH7     Spare           ;I                35

// Masks
//PORTT
#define mP_MGiroDir     b(0,0,0,0,0,0,0,1) // PORTP
#define mP_MGiroBrake   b(0,0,0,0,0,0,1,0)
#define mP_MElevDir     b(0,0,0,0,0,1,0,0)
#define mP_MElevBrake   b(0,0,0,0,1,0,0,0)
#define mP_MGiroPWM b(0,0,1,0,0,0,0,0)
#define mP_MElevPWM b(1,0,0,0,0,0,0,0)

#define mH_MElevTopeInf b(0,0,0,0,0,0,0,1)       // PORTH
```

```
#define ROBOT1 1

#ifdef ROBOT0
#define MAX_ABS_GIRO 250 //pant-tilt sobre robot 0
#define MAX_ABS_ELEV 100
#endif
#ifdef ROBOT1
#define MAX_ABS_GIRO 248 //pant-tilt sobre robot 1
#define MAX_ABS_ELEV 100
#endif
#ifdef ROBOT2 //pant-tilt solo
#define MAX_ABS_GIRO 244
#define MAX_ABS_ELEV 100
#endif

volatile extern unsigned char   g_giro_vel, g_elev_vel, real_time_flag, g_offset_bypass;
volatile extern unsigned short  g_giro_index, g_elev_index;
volatile extern short  g_giro_clicks, g_elev_clicks;
volatile short  g_dir_giro, g_dir_elev;

volatile short abs_giro, abs_elev;
volatile unsigned short giro_pos_control=0, elev_pos_control=0;
volatile static short des_giro_vel_clicks;  // Desired vel, clicks/ms
volatile static short des_elev_vel_clicks;  // Desired vel, clicks/ms
volatile short max_giro_vel_clicks;  // Initial desired bias, clicks/ms
volatile short max_elev_vel_clicks;  // Initial desired bias, clicks/ms
volatile static unsigned char giro_pwm, elev_pwm;  //Power to motors
volatile long k_pro_giro = 100; // Proportional gain
volatile long k_pro_elev = 100; // Proportional gain
volatile short edo_giro_motor=0;
volatile short edo_elev_motor=0;
volatile short k_vel_ini_giro = 3,  k_vel_ini_elev=3;
volatile short k_vel_end_giro = 50, k_vel_end_elev=50;
volatile short k_ms_inc_giro  = 30, k_ms_inc_elev =30;

volatile unsigned long clicks_stop_giro, clicks_stop_elev,
                       clicks_break_giro,clicks_break_elev,
                       total_giro_clicks, total_elev_clicks;
volatile unsigned short turns_stop_giro, turns_stop_elev;

volatile static unsigned short acelerando_giro, acelerando_elev, frenando_giro, frenando_elev;
volatile static short  integral_giro=0, integral_elev=0;
volatile static short timer_ac_giro, timer_ac_elev;

static void inline set_giro_pwm()
{
   PWMDTY5 = giro_pwm;
}

static void inline set_elev_pwm()
{
   PWMDTY7 = elev_pwm;
}

static void m_giro_izq()
{
   //ser0_puts("GIRO IZQ\n");
   PORTP |= mP_MGiroDir;
   PORTP &= ~mP_MGiroBrake;
}

static void m_giro_der()
{
```

```
   //ser0_puts("GIRO DER\n");
   PORTP &= ~mP_MGiroDir;
   PORTP &= ~mP_MGiroBrake;
}

void m_giro_stop()
{
   //ser0_puts("GIRO STOP\n");
   PORTP |= mP_MGiroDir;
   PORTP |= mP_MGiroBrake;
   giro_pwm=255U;
   set_giro_pwm();
   edo_giro_motor  = 0;
   acelerando_giro = 0;
   frenando_giro   = 0;
}

static void m_elev_bajar()
{
   PORTP &= ~mP_MElevDir;
   PORTP &= ~mP_MElevBrake;
}

static void m_elev_subir()
{
   PORTP |=  mP_MElevDir;
   PORTP &= ~mP_MElevBrake;
}

void m_elev_stop()
{
   PORTP |= mP_MElevDir;
   PORTP |= mP_MElevBrake;
   elev_pwm=255U;
   set_elev_pwm();
   edo_elev_motor  =0;
   acelerando_elev = 0;
   frenando_elev   = 0;
}

void set_velocity_giro()
{
   if(!max_giro_vel_clicks){
      m_giro_stop();
      return;
   }
   giro_pwm=0;          // Initial power to motors
   edo_giro_motor=1;
   giro_pos_control = 0;
   integral_giro=0;
   set_giro_pwm();
   des_giro_vel_clicks = ABS(max_giro_vel_clicks);
   if(max_giro_vel_clicks > 0){
      m_giro_izq();
   } else {
      m_giro_der();
   }
}

void set_velocity_elev()
{
   if(!max_elev_vel_clicks){
      m_elev_stop();
```

```
      return;
   }
   elev_pwm=0U;          // Initial power to motors
   edo_elev_motor=1;
   elev_pos_control = 0;
   integral_elev=0;
   set_elev_pwm();
   des_elev_vel_clicks = ABS(max_elev_vel_clicks);
   if(max_elev_vel_clicks > 0){
      m_elev_subir();
   } else {
      m_elev_bajar();
   }
}

void m_stop_giro()
{
   edo_giro_motor =0;
   max_giro_vel_clicks = 0;
   set_velocity_giro();
}
void m_stop_elev()
{
   edo_elev_motor=0;
   max_elev_vel_clicks = 0;;
   set_velocity_elev();
}

void stop_giro()
{
   ser0_puts("p\n");
   m_stop_giro();
}

void stop_elev()
{
   ser0_puts("P\n");
   m_stop_elev();
}

void set_vel_giro_aux(short vel_clicks)
{
   max_giro_vel_clicks = vel_clicks;
   set_velocity_giro();
   turns_stop_giro = 0;
   clicks_stop_giro = 0;
   acelerando_giro  = 0;
   frenando_giro    = 0;
}

/* *** SET_VEL COMMAND *** */
void set_vel_giro(short num_param, unsigned short param[])
{
   ser0_puts("v\n");
   set_vel_giro_aux (param[0] - 100);
}
/* *** END SET_VEL *** */

void set_vel_elev_aux(short vel_clicks)
{
   max_elev_vel_clicks  = vel_clicks ;
   set_velocity_elev();
   turns_stop_elev = 0;
```

```
   clicks_stop_elev = 0;
   acelerando_elev  = 0;
   frenando_elev    = 0;
}

/* *** SET_VEL COMMAND *** */
void set_vel_elev(short num_param, unsigned short param[])
{
   ser0_puts("V\n");
   set_vel_elev_aux(param[0] - 100);
}

/* *** END SET_VEL *** */

void set_ac_elev(short num_params, unsigned short param[])
{
   ser0_puts("K\n");
   k_vel_ini_elev = param[0];
   k_vel_end_elev = param[1];
   k_ms_inc_elev  = param[2];
}

void set_ac_giro(short num_params, unsigned short param[])
{
   ser0_puts("k\n");
   k_vel_ini_giro = param[0];
   k_vel_end_giro = param[1];
   k_ms_inc_giro  = param[2];
}

void set_params_giro(unsigned short p)
{
   turns_stop_giro = p;
   clicks_stop_giro = (long) turns_stop_giro * (long) 1000;
   total_giro_clicks  = 0;
   g_giro_clicks = 0;
   g_giro_index = 0;
   clicks_break_giro =  clicks_stop_giro/2;
   timer_ac_giro = k_ms_inc_giro;
   acelerando_giro = 1;
   frenando_giro = 0;
}

void set_params_elev(unsigned short p)
{
   turns_stop_elev = p;
   clicks_stop_elev = (long) turns_stop_elev * (long) 1000;
   total_elev_clicks  = 0;
   g_elev_clicks = 0;
   g_elev_index = 0;
   clicks_break_elev =  clicks_stop_elev/2;
   timer_ac_elev = k_ms_inc_elev;
   acelerando_elev = 1;
   frenando_elev = 0;
}

// Returns 0 if switch is pressed
volatile static unsigned short get_switchH(unsigned char mask)
{
   if(PORTH & mask)
      return 0;
   else
      return 1;
```

```
}

// Returns 0 if switch is pressed
volatile static unsigned short get_switchT(unsigned char mask)
{
   if(PORTT & mask)
      return 0;
   else
      return 1;
}

void turn_left_aux(unsigned short p)
{
   if(!get_switchT(mT_MGiroTopeIzq)){
      set_params_giro(p);
      g_dir_giro = 1;
      max_giro_vel_clicks = k_vel_ini_giro;
      set_velocity_giro();
   }
}

void turn_left(short num_params, unsigned short param[])
{
   ser0_puts("i\n");
   turn_left_aux(param[0]);
}

void turn_right_aux(unsigned short p)
{
   if(!get_switchT(mT_MGiroTopeDer)) {
      set_params_giro(p);
      g_dir_giro = 0;
      max_giro_vel_clicks = - k_vel_ini_giro;
      set_velocity_giro();
   }
}

void turn_right(short num_params, unsigned short param[])
{
   ser0_puts("d\n");
   turn_right_aux(param[0]);
}

void go_up_aux(unsigned short p)
{
   set_params_elev(p);
   max_elev_vel_clicks = k_vel_ini_elev;
   set_velocity_elev();
   g_dir_elev = 1;
}

void go_up(short num_params, unsigned short param[])
{
   ser0_puts("s\n");
   go_up_aux(param[0]);
}

void go_down_aux(unsigned short p)
{
   if(!get_switchH(mH_MElevTopeInf)) {
      set_params_elev(p);
      max_elev_vel_clicks = - k_vel_ini_elev;
      set_velocity_elev();
```

```
      g_dir_elev =0;
   }
}

void go_down(short num_params, unsigned short param[])
{
   ser0_puts("b\n");
   go_down_aux(param[0]);
}

void report_status(void)
{
   ser0_putchar('l');
   ser0_putshort(abs_giro); ser0_putspace();
   ser0_putshort(g_giro_clicks); ser0_putspace();
   ser0_putshort(abs_elev);  ser0_putspace();
   ser0_putshort(g_elev_clicks); ser0_putspace();
   ser0_putushort(get_switchT(mT_MGiroTopeIzq)); ser0_putspace();
   ser0_putushort(get_switchT(mT_MGiroTopeDer)); ser0_putspace();
   ser0_putushort(get_switchH(mH_MElevTopeInf)); ser0_putspace();
   ser0_putushort(edo_giro_motor); ser0_putspace();
   ser0_putushort(edo_elev_motor);
   ser0_putnl();
}

/* SPEED_CONTROL */

#define MAXPWM 255
static unsigned char limit_range(unsigned char pwm, long error)
{
   static long new_pwm;

   new_pwm = pwm + error;
   if(new_pwm > MAXPWM)
      new_pwm = MAXPWM;
   else if(new_pwm < 0)
      new_pwm = 0;
   return (unsigned char)new_pwm;
}

void alter_giro_power(long error )
{
   giro_pwm = limit_range(giro_pwm,error);
   set_giro_pwm();
}

void alter_elev_power(long error )
{
   elev_pwm = limit_range(elev_pwm,error);
   set_elev_pwm();
}

void speed_control_giro()
{
   static long int giro_vel, giro_error;

   if(!edo_giro_motor)
      return;
   giro_vel  = g_giro_vel;
   total_giro_clicks += giro_vel ; //Update total clicks
   giro_error  = k_pro_giro * (des_giro_vel_clicks - giro_vel)/100;
   alter_giro_power (giro_error );
}
```

```
void speed_control_elev()
{
   static long int elev_vel, elev_error;

   if(!edo_elev_motor)
      return;
   elev_vel  = g_elev_vel;
   total_elev_clicks += elev_vel; //Update total clicks
   elev_error  = k_pro_elev * (des_elev_vel_clicks - elev_vel)/100;
   alter_elev_power (elev_error);
}

void ac_control_giro()
{
   if(!turns_stop_giro)
      return;
   if(timer_ac_giro){
      timer_ac_giro--;
      return;
   }
   if(acelerando_giro){
      if(g_giro_vel + 3 >= k_vel_end_giro){ // reach the max velocity
          clicks_break_giro = clicks_stop_giro - total_giro_clicks;
          acelerando_giro = 0;
      }
      if(des_giro_vel_clicks < k_vel_end_giro) {
         des_giro_vel_clicks++;
      }
   }
   if (frenando_giro && des_giro_vel_clicks > k_vel_ini_giro) {
         des_giro_vel_clicks--;
   }
   timer_ac_giro = k_ms_inc_giro;
}

void ac_control_elev()
{
   if(!turns_stop_elev)
      return;
   if(timer_ac_elev){
      timer_ac_elev--;
      return;
   }
   if(acelerando_elev){
      if(g_elev_vel + 3 >= k_vel_end_elev){ // reach the max velocity
          clicks_break_elev = clicks_stop_elev - total_elev_clicks;
          acelerando_elev = 0;
      }
      if(des_elev_vel_clicks < k_vel_end_elev) {
         des_elev_vel_clicks++;
      }
   }
   if (frenando_elev && des_elev_vel_clicks > k_vel_ini_elev) {
         des_elev_vel_clicks--;
   }
   timer_ac_elev = k_ms_inc_elev;
}

void check_stop_motor_giro(void)
{
   if(edo_giro_motor && !frenando_giro && total_giro_clicks >= clicks_break_giro) {
      acelerando_giro = 0;
```

```
      frenando_giro   = 1;
   }
}

void check_stop_motor_elev(void)
{
   if(max_elev_vel_clicks < 0  && get_switchH(mH_MElevTopeInf)){
      m_stop_elev();
   }
   if(edo_elev_motor && !frenando_elev && total_elev_clicks >= clicks_break_elev) {
      acelerando_elev = 0;
      frenando_elev   = 1;
   }
}
void do_abs_giro(short num_param,unsigned short param[])
{
   static unsigned short new_abs_giro;
   static short  dif_giro;

   ser0_puts("g\n");
   if(num_param > 0) {
      new_abs_giro = param[0];
      if(new_abs_giro > MAX_ABS_GIRO)
         new_abs_giro = MAX_ABS_GIRO;
      dif_giro = new_abs_giro - abs_giro;
      if(dif_giro > 0)
         turn_left_aux((unsigned short) dif_giro);
      else if(dif_giro < 0){
         dif_giro = - dif_giro;
         turn_right_aux((unsigned short) dif_giro);
      }
   }
}
void do_abs_elev(short num_params,unsigned short param[])
{
  static unsigned short new_abs_elev;
  static short  dif_elev;

   ser0_puts("e\n");
   if(num_params > 0) {
      new_abs_elev = param[0];
      if(new_abs_elev > MAX_ABS_ELEV)
         new_abs_elev = MAX_ABS_ELEV;
      dif_elev = new_abs_elev - abs_elev;
      if(dif_elev > 0)
         go_up_aux((unsigned short) dif_elev);
      else if(dif_elev < 0){
         dif_elev = - dif_elev;
         go_down_aux((unsigned short) dif_elev);
      }
   }

}

int init_secuence(int s)
{
   static short result;

   result = 0;
   if(s==I_GIRO_DER_START){
      if(!get_switchT(mT_MGiroTopeDer)){
          g_dir_giro = 0;
    set_vel_giro_aux(-10);
```

```
#ifdef DEBUG
  ser0_puts("Girando a la derecha\n");
#endif
      }
      result = 1;
   } else if (s == I_GIRO_DER_END) {
      if (get_switchT(mT_MGiroTopeDer)){
          result = 1;
#ifdef DEBUG
  ser0_puts("Se alcanzó el tope derecho\n");
#endif
      }
   } else if(s==I_GIRO_IZQ_START){
      g_dir_giro = 1;
      set_vel_giro_aux(2);
      result = 1;
#ifdef DEBUG
      ser0_puts("Girando a la izquierda\n");
#endif
   } else if (s == I_GIRO_IZQ_END) {
      if (!get_switchT(mT_MGiroTopeDer)){
          result = 1;
          set_vel_giro_aux(0);
#ifdef DEBUG
  ser0_puts("Se liberó el tope derecho\n");
#endif
      }
   } else if (s == I_GIRO_1_IZQ_START){
      g_offset_bypass = 1;
      abs_giro = -1;
      turn_left_aux(1);
      result = 1;
#ifdef DEBUG
      ser0_puts("Giro de 1 unidad\n");
#endif
   } else if (s == I_GIRO_1_IZQ_END){
      if(!edo_giro_motor){
         result = 1;
         g_offset_bypass = 0;
#ifdef DEBUG
         ser0_puts("Motor de giro apagado\n");
#endif
      }
   } else if (s == I_ELEV_ABAJO_START){
      if(!get_switchH(mH_MElevTopeInf)){
          g_dir_elev = 0;
          set_vel_elev_aux(-10);
#ifdef DEBUG
          ser0_puts("Hacia abajo\n");
#endif
      }
      result = 1;
   } else if (s == I_ELEV_ABAJO_END) {
       if(get_switchH(mH_MElevTopeInf)){
          result = 1;
          set_vel_elev_aux(0);
#ifdef DEBUG
          ser0_puts("Se alcanzó el tope inferior\n");
#endif
       }
   } else if (s == I_ELEV_ARRIBA_START){
      g_dir_elev = 1;
      set_vel_elev_aux(2);
```

```
      result = 1;
#ifdef DEBUG
      ser0_puts("Hacia arriba\n");
#endif
   } else if (s == I_ELEV_ARRIBA_END){
      if(!get_switchH(mH_MElevTopeInf)){
          result = 1;
          set_vel_elev_aux(0);
#ifdef DEBUG
          ser0_puts("Se liberó el tope inferior\n");
#endif
      }
   } else if (s == I_ELEV_1_ARRIBA_START){
      g_offset_bypass = 1;
      abs_elev = -1;
      go_up_aux(1);
      result = 1;
#ifdef DEBUG
      ser0_puts("Hacia arriba 1 unidad\n");
#endif
   } else if (s == I_ELEV_1_ARRIBA_END) {
      if(!edo_elev_motor){
         result = 1;
         g_offset_bypass = 0;
#ifdef DEBUG
         ser0_puts("Motor de inclinación apagado\n");
#endif
      }
   } else if (s == I_READY){
      result = 1;
      ser0_puts("Listo\n");
   }
   return result;
}

void check_rtf(void)
{
   if(real_time_flag) {
      real_time_flag = 0;
      speed_control_giro();
      speed_control_elev();
      ac_control_giro();
      ac_control_elev();
   }
}

void init_control()
{
   DDRP = b(1,1,0,0,1,1,1,1);   //all are input but PP4 and PP5
   PORTP = 0;

   DDRH = b(0,0,0,0,0,0,0,0); //all are input
   DDRT = b(0,0,0,0,0,0,0,0);

   //PWM

   PWME |= mP_MGiroPWM | mP_MElevPWM;      // Pulse Width Channel 5 & 7 Enable (use Clock B)
   PWMPRCLK = b(0,0,1,0,0,0,0,0);       //Clock B=Bus clock/4
                                        //1/(1/24e6 * 4 *256))=23437Hz

   //ECT

   ICPAR |= b(0,0,0,0,1,1,0,0); //Pulse Acumulator 3, 2, are enabled
```

```
//   DLYCT |= b(0,0,0,0,0,0,0,1); //Delay Counter = 256 bus clock cycles
   DLYCT |= b(0,0,0,0,0,0,0,0); //Delay Counter disabled
   TIOS  |= b(0,0,0,0,0,0,0,0); //All channels are input capture
   TCTL4 |= b(1,1,1,1,0,1,0,1); //Input Capture Edge Control
//1,1=Capture on any edge, rising or falling (encoders A) T2 &T3
   TCTL3 |= b(1,0,1,0,0,1,0,1); //Input Capture Edge Control
//0,1=Capture on rising only (index MGiroI, MElevI, MGiroTopeIzq, MGiroTopeDer)

   ICSYS |= b(0,0,0,0,0,1,1,1); //PACMX=1 When the 8-bit pulse accumulator has reached the value
//$FF, it will no be incremented further.
//BUFEN=1 Input Capture and pulse accumulators registers are enabled
//LATQ=1 Latch Mode of Input Capture is enabled

   TFLG1 |= 0xff;

   TIE   =  b(1,1,1,1,0,0,1,1); //Timer Interrupt Enable Register
   TSCR2 =  b(0,0,0,0,0,1,1,1); //Timer Prescaler = Bus clock /128 =5.33 microseg
   TSCR1 =  b(1,0,0,0,0,0,0,0); //TEN=1 => TimerENable = 1
   MCCNT = 0;

   PWMPOL = 0xff; // PWM outputs are high at the beginning of the period, then goes
// low when the duty count is reached
   max_giro_vel_clicks = 0;
   max_elev_vel_clicks = 0;
   set_velocity_giro();
   set_velocity_elev();
}


/*************/
/* command.c */
/*************/

#include  <stdio.h>
#include  <string.h>
#include  "ioRegsHcs12DP256.h"
#include  "serial.h"
#include  "control.h"

volatile extern unsigned short g_timer;
volatile extern unsigned short g_ad2;
volatile extern unsigned short g_ad3;
volatile extern unsigned short g_ad4;
volatile extern unsigned short g_ad5;
volatile extern unsigned char  g_giro_vel;
volatile extern unsigned char  g_elev_vel;

volatile extern unsigned long  k_pro;
volatile extern unsigned long  k_integral;

//////////////////////
/* *** WAIT operation **** */
static unsigned short edo_wait=0;
static short wait_for_motor_giro, wait_for_motor_elev;
extern short edo_giro_motor, edo_elev_motor;

static void inline wait(short num_params, unsigned short param[])
{
   g_timer = param[0];
   if(num_params == 3) {
      wait_for_motor_giro = param[1];
      wait_for_motor_elev = param[2];
   } else {
```

```
      wait_for_motor_giro = 0;
      wait_for_motor_elev = 0;
   }

   edo_wait=1;
}

static void inline check_end_wait()
{
  if(edo_wait){
     edo_wait=0;
     if(wait_for_motor_giro && edo_giro_motor)
        edo_wait=1;
     if(wait_for_motor_elev && edo_elev_motor)
        edo_wait=1;
     if((!wait_for_motor_giro && !wait_for_motor_elev) || edo_wait)
         edo_wait = g_timer;
     if(!edo_wait)
        ser0_puts("w\n");
  }
}

/* *** QUERY operation **** */
extern short cg[];
extern short ce[];

void query(void)
{
   static short i;
   for(i=0; i < 251; i++) {
      ser0_putshort(ce[i]);
      ser0_putspace();
   }
   ser0_putnl();
}
/* *** End QUERY operation **** */

void check_end_commands(void)
{
   check_end_wait();
   check_stop_motor_giro();
   check_stop_motor_elev();
}

/*
void print_help()
{
   ser0_puts("Comandos disponibles:\n\n");
   ser0_puts("? Imprime este mensaje de ayuda\n");
   ser0_puts("* Regresa un asterísco\n");
   ser0_puts("e<p1>  Eleva la torreta p1 unidades con referencia al tope inferior\n");
   ser0_puts("g<p1>  Gira  la torreta p1 unidades con referencia al tope derecho\n");
   ser0_puts("k<pi> <pf> <ms> Define la aceleración del giro de la torreta\n");
   ser0_puts("K<pi> <pf> <ms> Define la aceleración de inclinación de la torreta\n");
   ser0_puts("l Reporta el estado de la torreta\n");
   ser0_puts("p Paro de emergencia del motor de giro\n");
   ser0_puts("P Paro de emergencia del motor de inclinación\n");
   ser0_puts("w<p1> 1 1 Espera por paro de motores de la torreta un maximo de p1 unidades de tiempo\n");
}
*/
void do_command(char command, int num_params, unsigned short param[])
{
   switch (command) {
```

```
      case 'w': wait(num_params, param);  break;
      case 'q': query();  break;
      case 'k': set_ac_giro(num_params,param); break;
      case 'K': set_ac_elev(num_params,param); break;
      case 'g': do_abs_giro(num_params,param); break;
      case 'e': do_abs_elev(num_params,param); break;
      case 'l': report_status(); break;
//      case '?': print_help(); break;
      case '*': ser0_puts("*\n"); break;
   }
   check_end_commands();
}

/*************/
/*  isr.c    */
/*************/

#include "ioRegsHcs12DP256.h"

#include  "isr.h"
#include  "control.h"
#include  "serial.h"
#include  "util.h"

#define OFFSET 250

void  __attribute__ ((interrupt)) ISR_Empty(void)
{
}

volatile unsigned char  g_offset_bypass = 0;
volatile unsigned short g_timer=0;
volatile unsigned char  g_giro_vel;
volatile unsigned char  g_elev_vel;
volatile unsigned short g_giro_index;
volatile unsigned short g_elev_index;
volatile short  g_giro_clicks=0, g_elev_clicks=0;
volatile unsigned char  real_time_flag=0;
extern volatile short abs_giro, abs_elev;
extern volatile short g_dir_giro, g_dir_elev;

volatile extern unsigned short turns_stop_giro, turns_stop_elev, giro_pos_control, elev_pos_control;
extern void speed_control();

static unsigned char giro_vel=0, elev_vel=0;
static unsigned char clicks_giro, clicks_elev;

short cg[260];
short ce[260];

static void inline get_vel()
{

#define ICLAT b(0,0,0,1,0,0,0,0)

   MCCTL |= ICLAT;      //Force the contents of 8-bit accumulators to be latched into their
                        //associated holding registers
   g_giro_vel = PA2H;
   g_elev_vel = PA3H;
}

#define RTIF            b(1,0,0,0,0,0,0,0)
#define SCFflag         b(1,0,0,0,0,0,0,0)       //SCF - Sequence Complete flag
```

```
/* Real Time Interrupt is running each milisecond */
void  __attribute__ ((interrupt)) ISR_RealTimeInt(void)
{
   if(g_timer)
      g_timer--;

   get_vel();
   real_time_flag=1;
   CRGFLG |= RTIF;      //clear flag
}

void  __attribute__ ((interrupt)) ISR_MGiroI(void)
{
   static unsigned short clicks;
   asm("sei");                  //Disable Any interrupts
   clicks = ABS(g_giro_clicks);
   if(g_offset_bypass || clicks >= OFFSET){

      g_giro_index++;
      if(g_dir_giro)
         abs_giro++;
      else
         abs_giro--;
      if(turns_stop_giro && g_giro_index >= turns_stop_giro){
         m_stop_giro();
         giro_pos_control = 1;
      }
      g_giro_clicks=0;
   }
   TFLG1 |= mT_MGiroI;
   asm("cli"); //Enable Any interrupts
}

void  __attribute__ ((interrupt)) ISR_MElevI(void)
{
   static unsigned short clicks;

   asm("sei");                  //Disable Any interrupts
   clicks = ABS(g_elev_clicks);
   if(g_offset_bypass || clicks > OFFSET) {

      g_elev_index++;
      if(g_dir_elev)
         abs_elev++;
      else
         abs_elev--;
      if(turns_stop_elev && g_elev_index >= turns_stop_elev){
         m_stop_elev();
         elev_pos_control = 1;
      }
      g_elev_clicks=0;
   }
   TFLG1 |= mT_MElevI;
   asm("cli"); //Enable Any interrupts
}

void  __attribute__ ((interrupt)) ISR_MGiroTopeDer(void)
{
   if(!g_dir_giro){
      m_stop_giro();
   }
   TFLG1 |= mT_MGiroTopeDer;
```

```
}

void  __attribute__ ((interrupt)) ISR_MGiroTopeIzq(void)
{
   if(g_dir_giro){
      m_stop_giro();
   }
   TFLG1 |= mT_MGiroTopeIzq;
}

void  __attribute__ ((interrupt)) ISR_MGiroB(void)
{
   asm("sei");
   if(PORTT & mT_MGiroA)
      g_giro_clicks -= 2;
   else
      g_giro_clicks += 2;
   TFLG1 |= mT_MGiroB;
   asm("cli");
}

void  __attribute__ ((interrupt)) ISR_MElevB(void)
{
   asm("sei");
   if(PORTT & mT_MElevA)
      g_elev_clicks -= 2;
   else
      g_elev_clicks += 2;
   TFLG1 |= mT_MElevB;
   asm("cli");
}


/*************/
   main.c
/*************/

#include  <stdio.h>
#include  <string.h>
#include  "ioRegsHcs12DP256.h"
#include  "serial.h"
#include  "command.h"
#include  "control.h"
#include  "main.h"

int                main(void);

extern char             _start_of_data;
extern char             _end_of_data;
extern const char       _start_of_init;

unsigned short int        start = 0x0000;
unsigned short int        end   = 0xffff;

#define BAUD57600  26      //(baud) 19200*3 baud with MCLK=24Mhz
#define BAUD28800  52      //(baud) 9600*3 baud with MCLK=24Mhz
#define BAUD14400 104      //(baud) 4800*3 baud with MCLK=24Mhz
#define BAUD9600 156  //(baud) 156 = 208 * 7200/9600 with MCLK=24Mhz
#define BAUD7200 208      //(baud) 2400*3 baud with MCLK=24Mhz

static void inline disableCOP(void)
{
   COPCTL = 0x00; //Disable COP
```

```
   COPCTL = 0x00;
}

#define OscClk 16000000 // Crystal or oscillator frequency
#define Eclock   24000000 // final E-clock frequency (PLL)
#define RefClock  8000000 // frequency used by the PLL to generate E-clock
#define REFDVVal (OscClk/RefClock)-1  //value for REFDV Register
#define SYNRVal (Eclock/RefClock)-1  //value for SYNR Register
static void inline rise24Mhz(void)
{
   REFDV = REFDVVal; //PLLCLK = 24MHz
   SYNR  = SYNRVal;
   asm("nop\n" "nop\n" "nop\n" "nop\n"); //nops required for bug
#define LOCK 0x08
   while(!(CRGFLG & LOCK)) //wait here till the PLL is locked
      ;
#define PLLSEL 0x80
   CLKSEL |= PLLSEL;  //System clocks are derived from PLLCLK

}

#define rtimask         b(0,1,0,0,0,0,0,1) //(1/16Mhz)2*2^13=1.024 msec interrupt
#define RTIF            b(1,0,0,0,0,0,0,0)
#define RTIE            b(1,0,0,0,0,0,0,0)
static void inline set_rti()
{
   RTICTL = rtimask;  //Initialize Real Time Interrupt rate
   CRGFLG |= RTIF; //clear flag
   CRGINT |= RTIE;  //Enable RTI
}

volatile extern short g_command_pending;
volatile extern short g_command_len;
volatile extern char g_command_line[];
volatile extern unsigned short g_timer;

static char command;
static short num_params;
static unsigned short param[20];

static decode_command()
{
   static int inicio_num, i;
   static char c;

   command=g_command_line[0];
   num_params=0;
   param[0]=0;
   inicio_num=0;
   for(i=1; (c=g_command_line[i]); i++) {
      if(c=='\r')
         continue;
      if(c==' ' || c == '\n') {
         param[++num_params]=0;
      } else {
         inicio_num=1;
         param[num_params] *= 10;
 param[num_params] += c - '0';
      }
   }
   if(!inicio_num)
      num_params--;
   g_command_len=0;
```

```
}

enum STEP_INIT step;

int  main()
{
   static int i;
   asm("sei"); //Disable Any interrupts

   disableCOP(); //Disable COP
   rise24Mhz(); //PLL Clock to 24Mhz
   memcpy(&_start_of_data, &_start_of_init, &_end_of_data - &_start_of_data);
   ser0_init(BAUD9600); //Baudrate is 9600
   set_rti();
   init_control();

   ser0_puts("Sistema ROCA-PANT-TILT v1.0\n");

   asm("cli");                  //unmask interrupts

   step=I_GIRO_DER_START;
   while(1){
     if(g_command_pending){
       g_command_pending=0;
       decode_command();
       do_command(command,num_params,param);
       g_command_pending=0;
     }
     check_rtf();
     check_end_commands();
     if(step <= I_READY){
        if(init_secuence(step))
            step++;
     }
  }

   return  0;
}
```

Apéndice B

Modificación del controlador del telémetro láser

El modelo de telémetro láser utilizado es el LMS209-S02 de SICK [SICb]. En la sección 2.6.3 del Capítulo 2 se describen las características y el principio de operación de este sensor.

En el sistema operativo Linux originalmente el controlador (*driver*) de este telémetro láser está configurado para reportar distancias con una resolución en centímetros y una distancia máxima de 8 metros. Sin embargo, reportar 8 metros como distancia máxima resulta insuficiente para ambientes interiores de dimensiones más grandes a 8 metros, tal es el caso de nuestro ambiente real con distanicas de hasta aproximadamete 15 metros.

El telemetro láser tiene la capacidad de sensar a una distancia máxima de 32 metros con una resolución en milímetros. Para realizar dicha configuración es necesario aplicar dos modificaciones al código del *driver*. El archivo fuente a modificar es *sensor.c*, y específicamente la función *SENS_ValuesRequest*. El primero de los cambios tiene que ver con el enmascaramiento de los datos recibidos. Originalmene la mascara es *0x1f* sobre el byte más significativo, por lo que el valor máximo es 8191 milimetros. La modidificación consiste en cambiar la mascara por *0x7ff*, así el valor máximo ahora es de 32767 milímetros.

La segunda modificación consiste en eliminar el factor de escala de la medición, así el valor reportado siempre estará en milímetros. El siguiente segmento de código presenta el "parche" que realiza las dos modificaciones al código del archivo *sensor.c*:

464,465c464,465

```
<       answer[ii+5] &= 0x1f;
<       d = (*((unsigned short int *)(&answer[ii+4])))*scalar;
---
>       answer[ii+5] &= 0x7f; // lrm jjap
>       d = (*((unsigned short int *)(&answer[ii+4]))); //jjap
```

Apéndice C

OpenGL

Uno de los principales problemas en la construcción de modelos 3D virtuales es el *rendering*, es decir, desplegar y visualizar las escenas como imágenes realistas en la computadora del modelo en tiempo real. La solución al problema está en utilizar herramientas que interactúen directamente con el hardware especializado en la aceleración gráfica. Entre estas herramientas está la interfaz de aplicación OpenGL [Richard00], que prácticamente se ha convertido en un estándar debido a su compatibilidad con casi cualquier plataforma de hardware, así como con varios lenguajes de programación (C, C++, Java, etc.).

C.1. Introducción

OpenGL estrictamente se define como: "*una interfaz de software para hardware gráfico*" [Shreiner00]. Por interfaz se refiere a comandos y funciones de biblioteca para modelado y graficación 3D. La biblioteca de OpenGL es extremadamente portable y rápida para una gran variedad de hardware de aceleración gráfica.

OpenGL, más que una interfaz de aplicación gráfica descriptiva, es procedural [Richard00]. Es decir, en lugar de describir cómo debería aparecer una escena, se deben especificar los pasos necesarios para lograr una cierta apariencia o efecto. Estos "pasos" no son otra cosa que funciones o comandos de la biblioteca de OpenGL, utilizados para trazar gráficos primitivos tales como puntos, líneas y polígonos en tres dimensiones. Además OpenGL soporta iluminación y sombreado, mapeo de textura, mezclado, transparencia, animación, y muchos otros efectos y capacidades especiales.

Las funciones y comandos de OpenGL están integrados en las tarjetas de acel-

eración gráfica 3D, o en su defecto se simulan por software. Es claro que para el propósito específico del *rendering* del modelo virtual 3D, obtenido de la reconstrucción, no se debe simular por software, se debe tener acceso directo al hardware. La Figura C.1 presenta una aplicación típica utilizando OpenGL y aceleración gráfica por hardware; se puede observar que las llamadas a la interfaz de aplicación OpenGL pasan directamente al hardware de aceleración y no pasan a través de la interfaz de dispositivo gráfico (GDI, por sus siglas en Inglés), que sería el caso si se tratara de simulación por software.

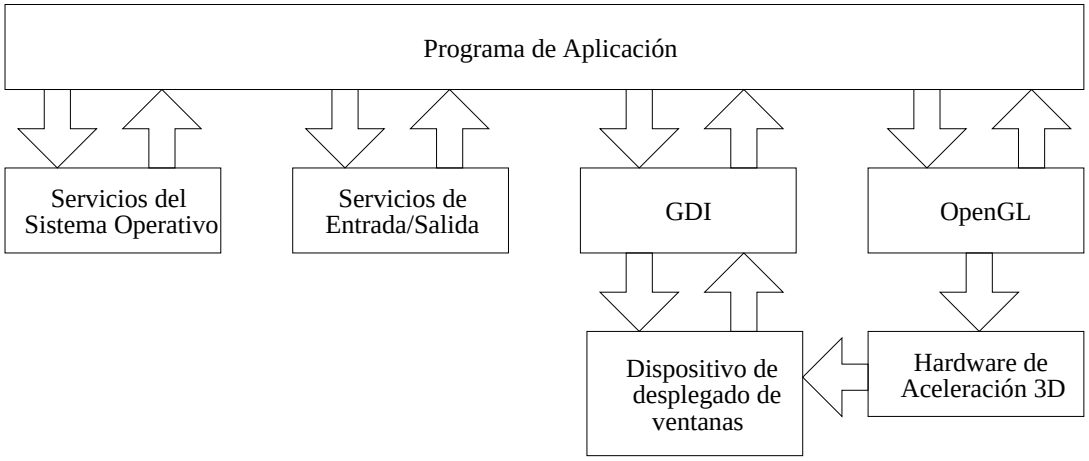


Figura C.1: Aplicación típica utlizando OpenGL y aceleración gráfica por hardware [Richard00].

Es importante resaltar que OpenGL es software público, además de ser independiente de la plataforma. Es decir, la biblioteca se incluye en prácticamente todos los sistemas operativos (Linux, Unix, MacOS, Windows).

En este capítulo se explican las primitivas básicas de dibujo, el color, la luz y materiales de textura necesarios para trazar una escena 3D, así como las transformaciones de coordenadas para el movimiento. Además se describe el conjunto de herramientas GLUT (*OpenGL Utility Toolkit*) para el manejo de ventanas y sus eventos. Se hace la aclaración que el presente capítulo no pretende ser un manual de OpenGL, sino una descripción del conjunto de funciones y comandos necesarios para generar y manipular el modelo 3D virtual, que se obtendrá de la reconstrucción 3D utilizando el robot móvil descrito en el capítulo 2.

C.2. Trazo de primitivas básicas

En esta sección se explica cómo describe OpenGL las primitivas geométricas utilizadas para trazar puntos, líneas, triángulos y otras primitivas en la pantalla (monitor). Todas las primitivas geométricas se encuentran descritas en términos de coordenadas tridimensionales de los vértices que definen sus propios puntos [Shreiner00]. OpenGL se encarga de mapear esos vértices 3D al plano de la pantalla de visualización, es decir al monitor de la PC.

La Figura C.2 muestra el sistema de referencia, llamado *volumen visible*, que se utilizará para explicar las primitivas de dibujo básicas y la proyección ortogonal. Este sistema de referencia se utilizará para realizar la visualización ortogonal del modelo 3D reconstruido. Se puede observar que el área encerrada por este volumen es un espacio cartesiano en el rango de -100 a +100 sobre los tres ejes (X, Y, Z).

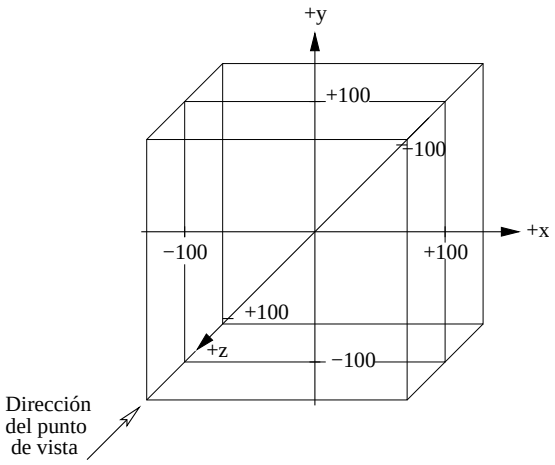


Figura C.2: Volumen visible cartesiano de 200x200x200

Para establecer el volumen visible, se debe llamar a la función de OpenGL *glOrtho*, que inicializa o modifica la extensión del volumen. Al utilizar esta función, si se tiene una proyección ortogonal, los objetos trazados lejos del punto de vista no aparecen más pequeños; como ocurriría en una proyección en perspectiva. La proyección en perspectiva se verá más a detalle en la Sección C.4. En seguida se verá la primera primitiva de OpenGL, el punto 3D.

C.2.1. Trazo de puntos en 3D

Un punto en el espacio es representado por un conjunto de 3 números reales de tipo flotante llamado *vértice*. OpenGL trabaja con coordenadas homogéneas de la geometría proyectiva tridimensional [Semple98], así para los cálculos internos, todos los vértices son representados por cuatro coordenadas en punto flotante (x, y, z, w). Si w es diferente de cero, las coordenadas corresponden a un punto en el espacio tridimensional Euclidiano ($\frac{x}{w}$, $\frac{y}{w}$, $\frac{z}{w}$).

Para especificar el trazo de un punto, en el volumen visible, se utiliza la función de OpenGL *glVertex*. Esta función es el mínimo común denominador de todas las primitivas de OpenGL: un simple punto en el espacio [Richard00]. Puede tomar de 2 a 4 parámetros de cualquier tipo númerico. OpenGL tiene una convención para nombrar sus funciones, así *glVertex2f*, *glVertex3f* y *glVertex4f*, señalan que tomarán 2, 3 y 4 parámetros de punto flotante respectivamente (la última letra indica el tipo de parámetro). Según la convención anterior tenemos tres formas de representar geométricamente el mismo punto en un plano, y en los espacios euclidiano y proyectivo. Por ejemplo: *glVertex2f(50.0, 20.0)*, *glVertex3f(50.0, 20.0, 0.0)* y *glVertex4f(50.0, 20.0, 0.0, 1.0)*, representan el mismo punto en el plano, o en su espacio correspondiente.

Una primitiva, en OpenGL, es simplemente la interpretación de un conjunto, o lista de vértices, dentro de una forma trazada sobre la pantalla. Existen diez primitivas desde un simple punto trazado en el espacio, hasta un polígono cerrado de n lados. Para especificar una lista o conjunto de vértices, que definirán una primitiva en particular, se emplea el comando *glBegin* para indicar el inicio del conjunto, obviamente se usará el comando *glEnd* para señalar el final del conjunto de vértices. El siguiente segmento de código muestra un ejemplo:

```
glBegin(GL_POINTS);              //Selección de puntos como la primitiva
  glVertex3f(0.0, 0.0, 0.0);     //Especificación de un punto (V0)
  glVertex3f(50.0, 50.0, 0.0);   //Especificación de otro punto (V1)
glEnd();                         //Hecho, puntos trazados
```

El argumento de *glBegin*, *GL_POINTS*, le dice a OpenGL que los siguientes vértices deben ser interpretados y trazados como puntos, de igual forma ocurre con las

otras nueve primitivas. En seguida se verá cuáles primitivas permiten trazar líneas.

C.2.2. Trazo de líneas en 3D

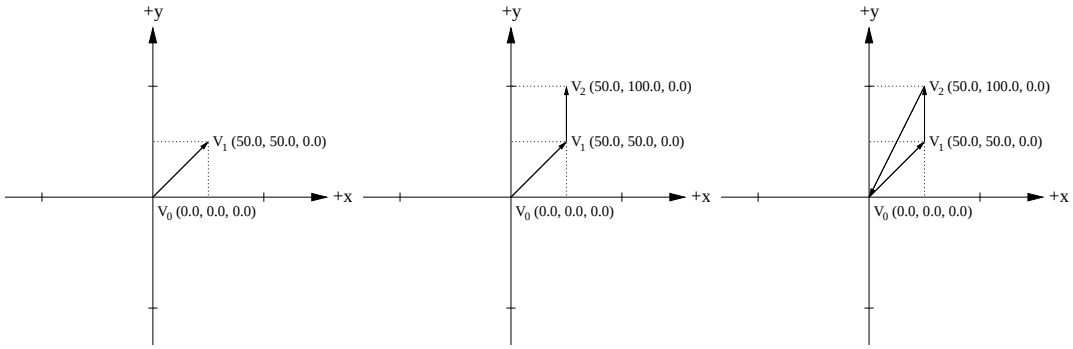
En OpenGL el término línea se refiere a un segmento de línea definido entre dos puntos (vértices) [Shreiner00]. Retomando el ejemplo anterior y cambiando la primitiva *GL_POINTS* por *GL_LINES*, el resultado sería un segmento de línea entre los dos vértices, tal como se muestra en la Figura C.3(a). Ahora los dos vértices (V_0, V_1) especifican una primitiva simple, es decir, por cada dos vértices se traza una línea. Se pueden trazar tantos segmentos de línea como se desee, sin embargo, si se especifica un número impar de vértices, el último vértice será ignorado, debido a que la línea está definida por dos vértices.

También es posible especificar una serie de segmentos de líneas conectadas. Cuando se utiliza la primitiva *GL_LINE_STRIP*, se traza un segmento de línea de V_0 a V_1, después de V_1 a V_2, y así sucesivamente, al final se traza un segmento de línea de V_{n-2} a V_{n-1}. Así un total de $n-1$ segmentos de línea son trazados (notese que el índice de los vértices comienza en 0). A diferencia de *GL_LINES* no hay restricciones en el número de vértices. El siguiente segmento de código traza dos líneas especificadas por tres vértices:

```
glBegin(GL_LINE_STRIP);
  glVertex3f(0.0, 0.0, 0.0);        //(V0)
  glVertex3f(50.0, 50.0, 0.0);      //(V1)
  glVertex3f(50.0, 100.0, 0.0);     //(V2)
glEnd();
```

Si en el ejemplo anterior se sustituye la primitiva *GL_LINE_STRIP* por *GL_LINE_LOOP*, el resultado es similar, pero con la diferencia que al final se traza un segmento de línea de V_{n-1} a V_0, completando o cerrando el ciclo. Las Figuras C.3(b) y C.3(c), muestran el resultado de aplicar las primitivas *GL_LINE_STRIP* y *GL_LINE_LOOP*, respectivamente.

Cuando se trazan segmentos de línea, o una secuencia de ellos, las líneas pueden intersectarse arbitrariamente, sin embargo, existen otras primitivas, como los polígonos, para las cuales la intersección esta restringida. A continuación se verá el primer polígono, el triángulo.



(a) Trazo de segmentos de líneas.

(b) Trazo de segmentos de líneas conectadas.

(c) Trazo de segmentos líneas conectadas en ciclo.

Figura C.3: Trazo de las primitivas básicas correspondientes a segmentos de líneas de tipo: (a) GL_LINES, (b) GL_LINES_STRIP, (c) GL_LINES_LOOP.

C.2.3. Trazo de triángulos en 3D

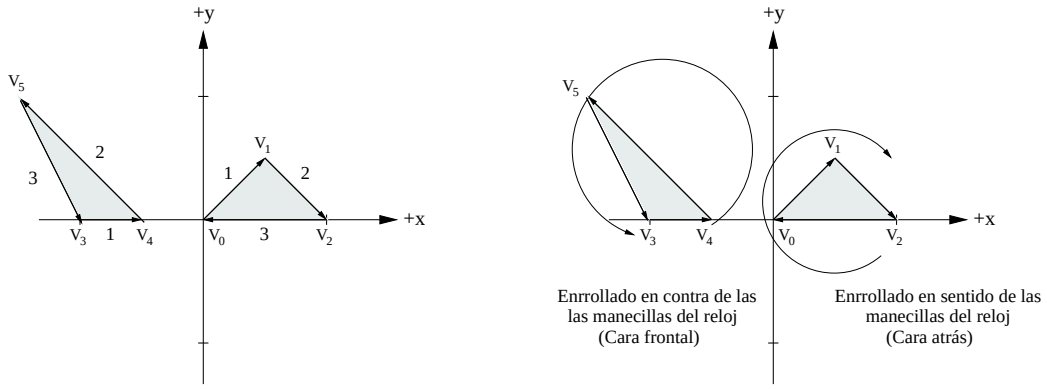
Con solo tres lados, el más simple de los polígonos es el triángulo [Richard00]. La primitiva *GL_TRIANGLES* traza un triángulo conectando tres vértices consecutivos. El siguiente segmento de código traza los triángulos mostrados en la Figura C.4(a):

```
glBegin(GL_TRIANGLES);
   glVertex3f(0.0, 0.0, 0.0);          //(V0)
   glVertex3f(50.0, 50.0, 0.0);        //(V1)
   glVertex3f(100.0, 0.0, 0.0);        //(V2)

   glVertex3f(-100.0, 0.0, 0.0);       //(V3)
   glVertex3f(-50.0, 0.0, 0.0);        //(V4)
   glVertex3f(-150.0, 100.0, 0.0);     //(V5)
glEnd();
```

La elección de esta primitiva, para generar cualquier superficie poligonal, mejora el desempeño. De hecho la mayoría del hardware de aceleración 3D está optimizado para el trazo de triángulos y la referencia para el desempeño se mide en triángulos por segundo

[Richard00].



(a) Trazo de triángulos con GL_TRIANGLES.

(b) Enrrollamiento (*windigns*).

Figura C.4: El trazo de triángulos, y en general de cualquier polígono, puede tener dos tipos de enrrollamiento (*windings*), en sentido de las manecillas del reloj y en contra.

La Figura C.4(b) ilustra una de las características más importantes de las primitivas para el trazo de polígonos, conocida como *windings* (enrrollamiento). En dicha figura las flechas denotan el sentido de conexión de los vértices, que puede ser en sentido de las manecillas del reloj o en contra. Por omisión, OpenGL considera que los polígonos que tienen un enrrollamiento en contra de las manecillas del reloj presentan la cara frontal. Por ejemplo, el triángulo izquierdo de la Figura C.4(b) se muestra de frente, y el del lado derecho se muestra por detrás.

El comportamiento por omisión de OpenGL puede cambiarse mediante la función *glFrontFace*, con los parámetros *GL_CW* y *GL_CCW*, determina si el enrrollamiento en sentido de las manecillas del reloj debe ser considerado como la parte frontal, o si lo debe ser el enrrollamiento en contra, respectivamente.

El enrrollamiento es muy importante ya que permite determinar diferentes características para las dos caras de un polígono. Se puede ocultar la parte de atrás de un polígono, o simplemente darle un color o textura diferente. En el ejemplo, mostrado en la Figura C.4(b), el área delimitada por los triángulos se ha sombreado, sin embargo la utilización de color se verá en la Sección C.3.

La función *glPolygonMode* permite que un polígono sea visualizado con relleno

sólido, con líneas, o simplemente con puntos. Además, se puede aplicar tal visualización a ambas caras del polígono o solo a una de ellas. En el modelo 3D virtual, obtenido de la reconstrucción, los polígonos construidos tienen un color sólido en su cara frontal, mientras que se muestran como estructura de alambre (líneas) por detrás.

Con la primitiva *GL_TRIANGLE_STRIP* es posible trazar una serie de triángulos conectados. Aquí lo importante es preservar el enrrollamiento en contra de las manecillas del reloj, no así el orden en el que se especificaron los vértices. Otra primitiva que también permite trazar una serie de triángulos conectados es *GL_TRIANGLE_FAN*, pero a diferencia de la primitiva anterior, aquí lo importante es que todos los triángulos se conectan a un vértice central.

Los triángulos son la primitiva preferida ya que la mayoría del hardware para OpenGL específicamente acelera triángulos, pero ésta no es la única primitiva disponible.

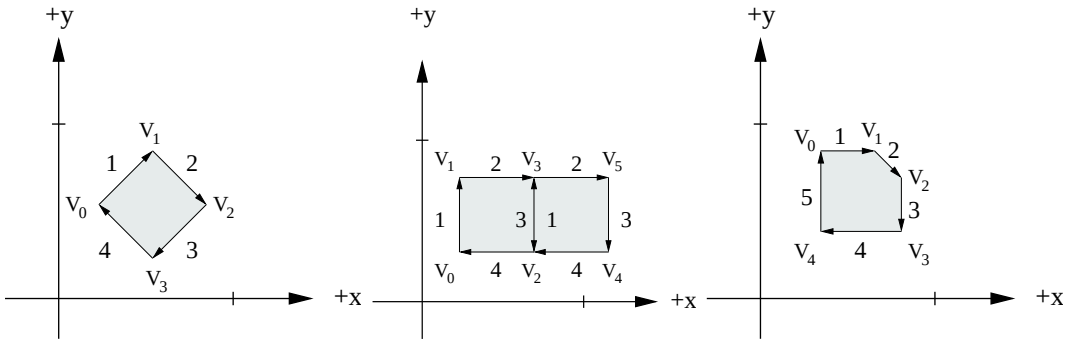
C.2.4. Otras primitivas

No es una regla general que todo hardware de aceleración 3D solo acelere triángulos, existe hardware que provee aceleración para otro tipo de polígonos. El resto de las primitivas de OpenGL dan una rápida especificación de un cuadrilátero, o un conjunto de cuadriláteros conectados, así como también polígonos de propósito general. El tipo de polígonos que el hardware acelera es el tipo de primitivas que se deben utilizar en el código OpenGL.

La primitiva de OpenGL *GL_QUADS*, traza un polígono de cuatro lados (cuadrilátero). La Figura C.5(a) muestra el trazado de un cuadrilátero con enrrollamiento en sentido de las manecillas del reloj. Con la primitiva *GL_QUAD_STRIP* se traza una secuencia de cuadriláteros conectados (ver Figura C.5(b)), al igual que con la primitiva anterior se conserva el enrrollamiento en sentido de las manecillas del reloj.

Finalmente la última primitiva de OpenGL es *GL_POLYGON*, que puede ser utilizada para trazar un polígono con cualquier número de lados. En la Figura C.5(c) se muestra el trazo de un polígono formado por cinco vértices.

Cuando se construyen superficies complejas utilizando polígonos se deben tener en cuenta dos reglas: (1) todos los polígonos deben ser planos, es decir, todos los vértices de los polígonos deben caer dentro de un plano, los polígonos no deben torcerse o mezclarse en el espacio; (2) los límites de los polígonos no deben intersectarse, y los polígonos deben



(a) Trazo con GL_QUADS. (b) Trazo con GL_QUAD_STRIP. (c) Trazo con GL_POLYGON.

Figura C.5: Trazo de polígonos con las primitivas: GL_QUADS, GL_QUAD_STRIP y GL_POLYGON.

ser convexos.

Las primitivas descritas en esta sección nos permiten trazar una escena, pero para darle un mayor realismo a dicha escena es necesario proporcionarle color, luz y propiedades de material.

C.3. Color, luz y materiales

Como ya se mencionó, el color, la luz y las propiedades del material proporcionan a una escena, trazada en OpenGL, un mayor realismo. Evidentemente en el mundo real existen más que simples objetos con color, estos también pueden aparecer brillando, o en color mate, o pueden ser ellos mismos una fuente de luz. En esta sección solo nos ocuparemos de aquellos aspectos que permiten crear el modelo tridimensional virtual como una malla poligonal construida a partir de las lecturas del telémetro láser, con el mayor realismo posible, sin tratar a los objetos inmersos en dicho modelo de forma individual, de hecho el modelo tridimensional completo es considerado como el único objeto en la escena.

C.3.1. Color en OpenGL

OpenGL específica exactamente un color como tres componentes de intensidad: rojo, verde y azul. Tal como la mayoría del hardware gráfico lo hace. El color en una

pantalla de computadora resulta de la emisión de diferentes cantidades luz roja, verde y azul. Ese paquete de colores comúnmente es llamado RGB (del Inglés *Red*, *Green* y *Blue*). Algunas veces dicho paquete también contiene un cuarto valor llamado alfa, entonces éste es referido como RGBA.

Existe una gran variación, entre diferentes plataformas de hardware gráfico, con respecto al tamaño del arreglo de píxeles (resolución, las más comunes: 640x480, 800x600 y 1024x768) y al número de colores que pueden ser desplegados por píxel. En cualquier sistema gráfico cada píxel tiene la misma cantidad de memoria para almacenar su color, y la memoria para todos los píxeles es llamada *buffer de color*. El tamaño del *buffer* es usualmente medido en *bits* (dígitos), así un *buffer* de 8 dígitos, almacena 8 dígitos de datos (256 posibles colores) para cada píxel. El tamaño del *buffer* varía en función del hardware.

La información de color para cada píxel puede ser almacenada en modo RGBA, en el cual los valores R, G, B y posiblemente un A son mantenidos para cada píxel, o en un modo de índice de color, en el cual un simple número es almacenado para cada píxel. Cada índice de color señala una entrada en una tabla que define un conjunto de valores RGBA en particular, tal tabla es llamada *mapa de colores*.

Especificación de color en el modo RGBA

Los valores R, G y B, van del rango de 0.0 a 1.0, es decir, de intensidad nula a la máxima intensidad. Por ejemplo $R = 0.0$, $G = 0.0$ y $B = 1.0$ representan la máxima brillantés para el canal azul.

En el modo RGBA se utiliza el comando *glColor* para seleccionar el color actual. Este comando puede tener un máximo de tres sufijos, los cuales van a diferenciar las variaciones de los parámetros aceptados. El prototipo de este comando es el siguiente:

void glColor<x><t>[v](rojo, verde, azul, alfa);

En el nombre del comando, <*x*> representa el número de argumentos, 3 para tres argumentos, el rojo, verde y azul, o 4 para incluir el componente alfa. Por omisión OpenGL asigna un valor de 1.0 para el componente alfa cuando solo se reciben 3 argumentos. En el modelo 3D a crear, son suficientes solo los tres canales RGB. La <*t*> en el nombre de la función especifica el tipo de argumento y puede ser *b*, *d*, *f*, *i*, *s*, *ub*, *ui*, o *us*, para *byte*, doble, flotante, entero, entero corto, *byte* sin signo, entero sin signo, y entero corto sin signo, respectivamente. La *[v]* es opcional, en caso de tenerla en el nombre de la función, señala

que el argumento es un puntero a un arreglo de los valores especificados por los sufijos anteriores.

Las líneas y los polígonos con relleno pueden ser trazados con un color simple: *sombreado plano*, o con diferentes colores: *sombreado suave*. La técnica de sombreado deseada puede especificarse mediante la función *glShadeModel*, que puede recibir uno de los siguientes parámetros: (1) *GL_FLAT*, o (2) *GL_SMOOTH*. Con el primero, el color de un vértice en particular es duplicado en todos los otros vértices de la primitiva para renderizarse. Con el segundo, el color en cada vértice es tratado de forma individual; para una línea, el color a lo largo de la línea es interpolado entre el color de los vértices. De igual forma ocurre con un polígono con relleno, de tal forma que el color del centro será la interpolación de sus vértices.

Sin embargo, los objetos en la realidad no aparecen en colores sólidos, o solamente en color sombreado sobre valores RGB. A continuación se presenta cómo es el color y los materiales de los objetos en el mundo real.

C.3.2. Color en el mundo real

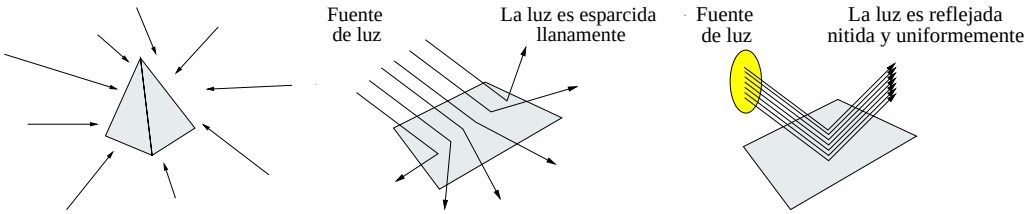
OpenGL realiza una aproximación razonablemente buena del mundo real en términos de condiciones de iluminación. A menos que un objeto emita su propia luz, éste es iluminado por tres diferentes formas de iluminación: ambiental, difusa y especular (semejante a un espejo).

Iluminación ambiental

La iluminación ambiental no proviene de ninguna dirección en particular. Existe una fuente de luz, pero los rayos de luz están rebotando por todos lados en la escena y llegan sin dirección específica. Los objetos, en una escena con luz ambiental, son iluminados igualmente sobre toda la superficie en todas direcciones. Es decir están igualmente coloreados o sombreados independientemente del punto de vista. La Figura C.6(a) muestra un objeto puramente iluminado con luz ambiental.

Iluminación difusa

La iluminación difusa, tiene una fuente de luz que viene de una dirección en particular, y es reflejada con suavidad fuera de la superficie. Aún si la luz es reflejada suavemente,



(a) Objeto con iluminación ambiental.

(b) Objeto con iluminación difusa.

(c) Objeto con iluminación especular.

Figura C.6: En el mundo real los objetos pueden estar iluminados por tres formas distintas de iluminación: ambiental, difusa y especular.

la superficie del objeto está iluminada si la fuente de luz se apunta directamente a la superficie, más que si solo roza la superficie en cierto ángulo. La Figura C.6(b) muestra un ejemplo de luz difusa.

Iluminación especular

Al igual que la iluminación difusa, la iluminación especular también viene de una dirección en particular, emitida por una fuente de luz (ver Figura C.6(c)). Pero a diferencia de la iluminación difusa, la luz es reflejada en una dirección en particular. Una iluminación altamente especular tiende a causar manchas brillantes sobre la superficie.

La iluminación es solo parte de la ecuación. En el mundo real, los objetos tienen su propio color, en función del material que están hechos. En seguida se verá éste aspecto.

C.3.3. Materiales en el mundo real

En ésta sección se describirá el color de un objeto definiendo su reflexión a la longitud de onda de la luz. Por ejemplo, una superficie de color azul refleja mayoritariamente fotones azules y absorbe la mayoría de los otros. Esto asume que la luz brillante sobre la superficie azul tiene fotones azules, que serán reflejados y detectados por el observador.

Generalmente, la mayoría de las escenas en el mundo real están iluminadas por una luz blanca, que contiene una mezcla uniforme de todos los colores. Bajo la luz blanca, la mayoría de los objetos aparecen en sus propios colores. Sin embargo, esto no siempre es así; si la superficie azul se encuentra en un cuarto obscuro con solamente una fuente de luz

amarilla, la superficie aparece negra, esto se debe a que la luz amarilla es absorbida y no existe luz azul para ser reflejada.

Propiedades de los Materiales

Cuando se usa iluminación, realmente no se describen polígonos que tienen un color en particular, no obstante son constituidos de materiales que tienen ciertas propiedades reflectivas. En lugar de decir que un polígono es rojo, se dice que el polígono esta hecho de material que mayoritariamente refleja la luz roja. Con ésto se estará diciendo que la superficie es roja, pero se deberán especificar las propiedades reflectivas del material para las fuentes de luz ambiental, difusa y especular.

Por ejemplo, un material que es especular, refleja la luz especular muy bien, mientras que absorbe la mayor parte de la luz ambiental y la difusa. Recíprocamente, un objeto con color plano debería absorber la luz especular y no brillar bajo ninguna circunstancia. Otra propiedad que puede ser especificada es la emisión de luz por los propios objetos, tal como una lámpara en la obscuridad.

Iluminación sobre los materiales

Cuando se traza un objeto, OpenGL decide cual color usar para cada píxel en el objeto. Este objeto tiene reflexión (color), y existe una fuente de luz que tiene su propio color. En cada vértice de las primitivas se asigna un color, en valores RGB, que es el resultado del producto del color de la luz por las propiedades del material. Por medio de un sombreado suave entre los vértices, la ilusión de iluminación se logra.

Cálculo de los efectos de la iluminación ambiental

Para una iluminación ambiental de mediana intensidad los componentes rojo, verde y azul, en valores RGB de la fuente son (0.5, 0.5, 0.5). Si dicha luz ambiental ilumina un objeto con propiedades de reflexión especificadas en términos RGB de (0.5, 1.0, 0.5), entonces el componente de color neto de la luz ambiental es: (0.5 * 0.5, 0.5 * 1.0, 0.5 * 0.5) = (0.25, 0.5, 0.25), que, evidentemente, es el resultado de multiplicar cada uno de los términos de la luz ambiental por los de las propiedades del material. Así, los componentes del color del material, determinan el porcentaje de incidencia de luz reflejada (ver Figura C.7).

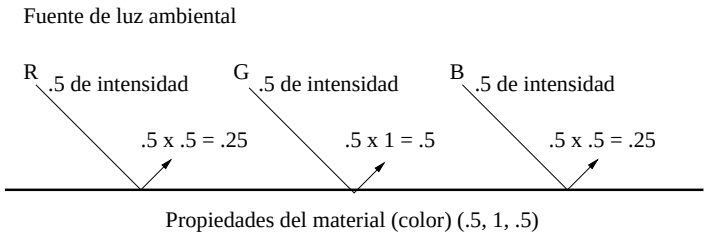


Figura C.7: Cálculo de los componentes del efecto de la luz ambiental sobre el color de un objeto.

Efecto de iluminación difusa e iluminación especular

La iluminación difusa también tiene intensidades de RGB que interactúan de la misma manera con las propiedades del material. Sin embargo, la luz difusa tiene dirección, y la intensidad sobre la superficie del objeto varía dependiendo del ángulo entre la superficie y la fuente de luz, la distancia de la fuente de luz es un factor de atenuación.

De manera similar pasa con las fuentes e intensidad de luz especular. El efecto neto, en términos de valores RGB, es de la misma forma que para la iluminación ambiental. La intensidad de la fuente de luz es ajustada por el ángulo de incidencia, y se multiplica por la reflectancia de material. Finalmente los tres valores RGB, para cada tipo de iluminación, son sumados para producir el color final del objeto.

Generalmente, los efectos de las fuentes de iluminación ambiental e iluminación difusa, sobre materiales con propiedades de iluminación ambiental y difusa, son semejantes y tienen el mayor efecto en la determinación del color del objeto. La iluminación especular y materiales con tal propiedad, tienden a ser grises o blancos [Richard00]. El componente de brillantez depende significativamente del ángulo de incidencia, y la máxima brillantez sobre un objeto usualmente es blanca.

En la siguiente sección se pondrá en código de OpenGL lo visto en ésta sección para adicionar luz a una escena.

C.3.4. Iluminación de escenas

Para decirle a OpenGL que utilice cálculos de iluminación se debe invocar el comando *glEnable*, con el parámetro *GL_LIGHTING* (*glEnable(GL_LIGHTING);*). Lo anterior solamente indica que se utilizará las propiedades del material y los parámetros de

iluminación para determinar el color de cada vértice en la escena. Sin embargo, si no se han especificado las propiedades del material o los parámetros de iluminación, el objeto permanecerá obscuro.

Después de habilitar los cálculos de iluminación, la primer cosa que se debe hacer es inicializar el modelo de iluminación. Los parámetros que afectan el modelo de iluminación son asignados mediante la función *glLightModel*. El siguiente segmento de código presenta un ejemplo:

```
//Brillo de luz blanca
GLfloat luzambiental[] = {1.0f, 1.0f, 1.0f, 1.0f};
//Habilitar iluminación
glEnable(GL_LIGHTING);
//Inicializar el modelo de luz ambiental usando el brillo de luz blanca
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, luzambiental);
```

Las variaciones de *glLightModel* son cuatro: (1) *glLightModelf*, (2) *glLightModeli*, (3) *glLightModelfv* y (4) *glLightModeliv*. En el ejemplo del código anterior se utilizó la tercera variación. El primer parámetro determina el modelo de iluminación, el que se eligió fue el modelo de luz ambiental. Existen otros modelos de luz que permiten determinar si ambas caras de los polígonos deben ser iluminadas, y cómo el cálculo de los ángulos de la iluminación especular se realizan. El segundo parámetro es un arreglo de valores RGBA para indicar los componentes de la luz ambiental.

El siguiente paso para iluminar la escena es especificar las propiedades del material, así los polígonos reflejarán la luz y la escena estará visible. Existen dos formas de asignar las propiedades del material. La primera es utilizando la función *glMaterial* antes de especificar cada polígono o conjunto de ellos. El primer parámetro que recibe la función determina que parte o partes del polígono tomarán las propiedades especificadas (al frente, detrás, o ambos). El segundo parámetro señala que propiedades deben ser asignadas. El tercer parámetro contiene los valores RGBA de dichas propiedades. Todas las primitivas especificadas después de invocada la función serán afectadas, a menos que se realice otra llamada a dicha función.

La segunda forma de asignar las propiedades del material es llamada: *color tracking* (seguimiento de color). Esta forma es la más utilizada, con ella se le dice a OpenGL

que las propiedades del material se asignarán solamente llamando a la función *glColor*. Pero para habilitar el seguimiento de color previamente habrá de invocarse al comando *glEnable* con el parámetro *GL_COLOR_MATERIAL*. Entonces la función *glColorMaterial* especificará los parámetros del material que seguirán los valores asignados por *glColor*. El siguiente segmento de código presenta un ejemplo:

```
//Habilitar seguimiento de color
glEnable(GL_COLOR_MATERIAL);
//Asignación de las propiedades del material
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
//...
glColor3f(0.50, 0.30, 1.0);
glBegin(GL_TRIANGLES);
  //...
glEnd();
```

Los parámetros que recibe la función *glColorMaterial* corresponden a los dos primeros parámetros de la función *glLightModelfv*.

Además del color, la luz tiene otras propiedades como orientación y dirección. La siguiente sección explica el control de dichas propiedades.

C.3.5. Utilizar una fuente de luz

OpenGL soporta ocho o más fuentes de luz independientes localizadas en cualquier lugar de la escena o del volumen visible. Cuando se especifica una fuente de luz, se le dice a OpenGL dónde y en qué dirección esta la brillantez. Muchas veces la fuente de luz ilumina en todas direcciones, o podría ser en alguna dirección en particular. Cualquiera que sea el caso, para cualquier objeto trazado, los rayos de luz, de cualquier fuente, golpean la superficie de los polígonos del objeto en un determinado ángulo. Para calcular el efecto de sombra a través de la superficie de los polígonos, OpenGL debe ser capaz de calcular dicho ángulo.

La Figura C.8 muestra un polígono cuadrado incidido por un rayo de luz proveniente de una fuente. El rayo hace un ángulo A con el plano en el punto que toca la superficie. Entonces la luz es reflejada a un ángulo B en dirección del punto de vista del observador.

Estos ángulos son usados, en conjunción con las propiedades de la luz y los materiales, para calcular que tan claro es el color en esa localidad. Calculando la claridad del color para cada vértice y haciendo un sombreado suave entre ellos, se logra la ilusión de la iluminación con OpenGL.

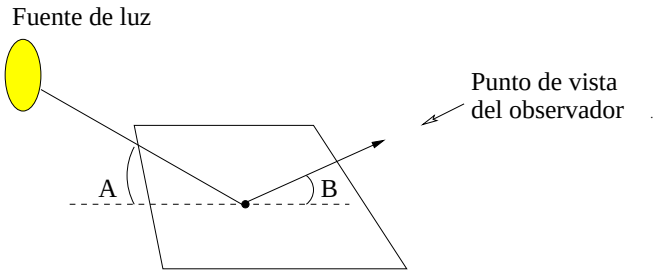


Figura C.8: La luz es reflejada al incidir con un objeto en un ángulo específico.

Cada polígono en una escena está formado por vértices, los cuales no son otra cosa que puntos en el espacio 3D. Cada vértice es incidido por un rayo de luz en cierto ángulo. Aquí surge un problema: calcular el ángulo entre una línea recta (rayo de luz), y un punto (vértice). Geométricamente es imposible ya que existe una infinidad de posibles soluciones. Por lo tanto, se debe asociar con cada vértice alguna pieza de información que denote la dirección hacia arriba del vértice y lejos de la superficie de la primitiva.

Superficie normal

Una línea perpendicular a una superficie real o imaginaria, que bien podría ser un polígono, es llamada *vector normal*. Un vector es una línea orientada en alguna dirección, y la palabra "normal" no es otra cosa que una manera de decir perpendicular. Por lo tanto, un vector normal es una línea orientada en una dirección a un ángulo de 90° de la superficie de un polígono.

Especificación de una normal

La Figura C.9 visualiza gráficamente cómo se especifica una normal para un vértice que pertenece a un polígono. Se puede observar que la línea a través del vértice (1, 1, 0) es perpendicular al plano. Si se selecciona cualquier punto sobre dicha línea, por ejemplo (1, 10, 0), entonces el segmento de línea entre los puntos (1, 1, 0) y (1, 10, 0) es un vector normal.

El segundo punto especifica la dirección hacia arriba del vértice. Lo anterior también es utilizado para indicar la parte de enfrente y la parte detrás de los polígonos.

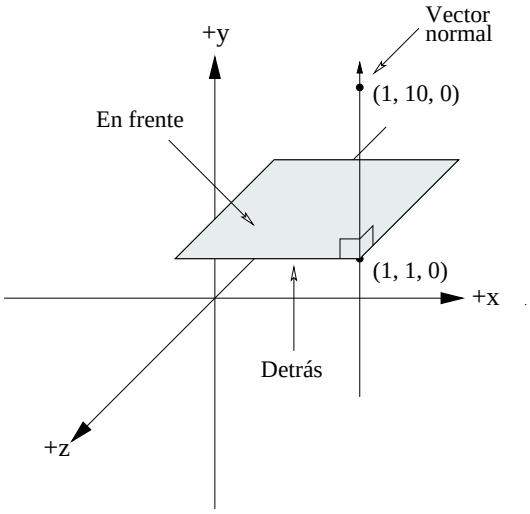


Figura C.9: Un vector normal es perpendicular a una superficie.

Se puede apreciar que el segundo punto es un número de unidades en dirección x, y y z para algún punto sobre el vector normal lejos del vértice. Sin embargo más que especificar dos puntos para cada vector normal, lo que se pretende es tener una simple coordenada que indique que tan lejos se está del vértice. Lo anterior se logra restando ambos puntos $(1, 10, 0) - (1, 1, 0) = (0, 9, 0)$. Este vector resultante es una cuantificación direccional que le dice a OpenGL en que dirección está la cara de los vértices o polígono.

En OpenGL la función *glNormal3f* toma una tripleta de coordenadas para especificar un vector normal orientado en la dirección perpendicular a la superficie de un polígono, que comúnmente es un triángulo. El siguiente segmento de código muestra un ejemplo de dicha función.

```
glBegin(GL_TRIANGLES);
  glNormal3f(0.0, 0.0, -1.0);
  glVertex3f(0.0, 0.0, 0.0);
  glNormal3f(0.0, 0.0, -1.0);
  glVertex3f(50.0, 50.0, 0.0);
  glNormal3f(0.0, 0.0, -1.0);
```

```
    glVertex3f(100.0, 0.0, 0.0);
glEnd();
```

En el segmento de código anterior se puede observar que se especifica una normal para cada vértice, esto no es siempre necesario, de hecho pudo haberse especificado el vector normal, una sola vez, antes de la primitiva correspondiente al triángulo y se tendría el mismo efecto. Especificar una normal diferente para cada vértice de un polígono puede dar el efecto de una curva suave sobre la superficie de dicho polígono [Richard00].

Cuando se especifica el vector normal de un polígono, se debe tener en cuenta el enrollamiento de los vértices. Por ejemplo, en el segmento de código anterior el enrollamiento es en sentido de las manecillas del reloj (ver Figura C.4(b) triángulo derecho), y se trata de un triángulo en el plano $X\ Y$, si el punto de vista es en dirección a $-Z$, entonces su parte frontal está en esa dirección y su parte detrás está viendo hacia $+Z$.

Unidades normales

Para los cálculos de iluminación, eventualmente todas las superficies normales deberían ser convertidas a unidades normales. Una unidad normal es un vector de longitud 1. En la Figura C.9 la normal tiene longitud 9. Para convertir esa normal a unidades normales, se puede dejar el trabajo a OpenGL utilizando el comando *glEnable* con los parámetros *GL_NORMALIZE* y *GL_RESCALE_NORMALS*. Sin embargo, es recomendable cálcular la longitud de la normal implementando en el código la siguiente ecuación: $long = \sqrt{x^2 + y^2 + z^2}$; donde x , y y z son las componentes de la normal. Después habrá que dividir cada una de las componentes entre la longitud para obtener un vector en exactamente la misma dirección pero de longitud unitaria $(\frac{x}{long}, \frac{y}{long}, \frac{z}{long})$. El proceso anterior es llamado *normalización*. Aplicando la normalización al vector normal $(0, 9, 0)$, de la Figura C.9, se obtiene el nuevo vector normal de magnitud unitaria $(0, 1, 0)$.

En el ejemplo de la Figura C.9 el polígono se encuentra en un plano definido por dos ejes (X y Y), el cálculo de la normal en tal caso es trivial, sin embargo la mayoría de las veces no es así.

Cálculo de la normal

La Figura C.10 muestra un polígono que no cae en ninguno de los planos formados por los ejes. Se requiere de una manera lo más sencillamente posible para calcular la normal de un polígono en coordenadas 3D arbitrarias.

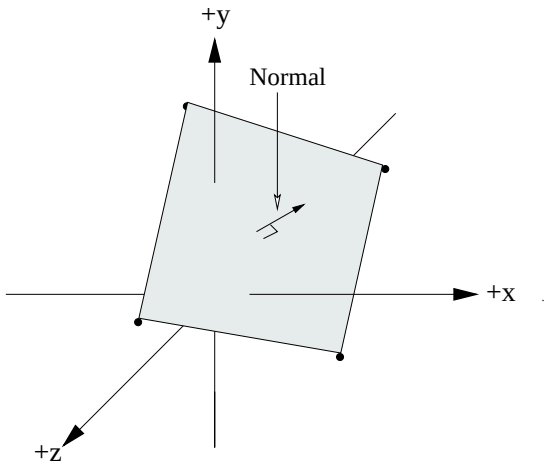


Figura C.10: Problema de la normal no trivial.

Es posible calcular la normal para cualquier polígono que consista de al menos tres puntos que estén dentro de un plano simple (un polígono plano). La Figura C.11 muestra tres puntos, P_1, P_2 y P_3, que son utilizados para formar dos vectores, V_1 de P_1 a P_2, y V_2 de P_1 a P_3. Matemáticamente, dos vectores en el espacio tridimensional definen un plano. Si se toma el producto cruz entre esos dos vectores (V_1 x V_2), el vector resultante es perpendicular al plano, es decir la normal al plano que se está buscando.

Ahora que se han establecido los requerimientos para que los polígonos reciban e interactúen con una fuente de luz, se verá como asignar una fuente de luz a la escena.

Creación de una fuente de luz

La función utilizada para asignar todas las propiedades de la fuente de luz es *glLight*; toma tres argumentos: el primero para indicar la fuente de luz a la que le serán asignadas las propiedades, el segundo para indicar la propiedad, y el tercero para decidir el valor de la propiedad.

El primer parámetro puede ser una, de las al menos ocho, fuentes de luz soportadas

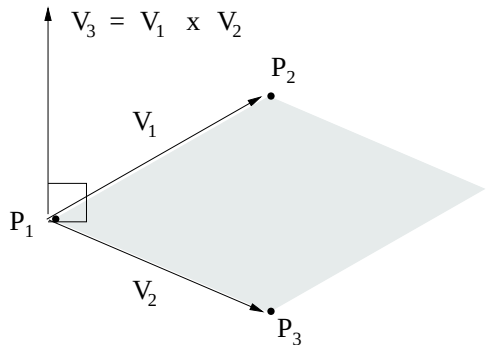


Figura C.11: El producto cruz de dos vectores, definidos por tres puntos, es utilizado para calcular la normal al plano formado por dichos vectores.

por OpenGL, comúnmente se utiliza *GL_LIGHT0*. El segundo parámetro puede ser una de las propiedades especificadas en la Tabla C.1. Dicha tabla solo contiene las propiedades más comunes.

Tabla C.1: Propiedades de la luz y valores por omisión

Propiedad	Valor por omisión	Significado
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	Intensidad RGBA ambiental de luz
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	Intensidad RGBA difusa de luz
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	Intensidad RGBA especular de luz
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	Posición (x, y, z, w) de la luz

El tercer parámetro asigna un valor específico a cada propiedad. Por ejemplo, para la intensidad RGBA de la luz ambiental se podría tener el siguiente segmento de código:

```
//Intensidad RGBA de la luz ambiental
GLfloat luzambiental[] = {1.0f, 1.0f, 1.0f, 1.0f};
//Asignar la intensidad RGBA a la luz ambiental
glLightfv(GL_LIGHT0, GL_AMBIENT, luzambiental);
```

De igual manera podría ser para los otros tipos de luz, y de manera similar para especificar la posición de la fuente de luz. El siguiente segmento de código determina la posición de la fuente de luz:

```
//Posición de la luz
GLfloat pos_luz[] = {-50.0f, 50.0f, 100.0f, 1.0f};
//Asignar la posición de la luz
glLightfv(GL_LIGHT0, GL_POSITION, pos_luz);
```

En el segmento de código anterior, el vector *pos_luz* contiene la posición de la luz. El último valor del arreglo es 1.0, lo cual quiere decir que la fuente de luz se localiza en las coordenadas especificadas por el vector. Si el último valor del arreglo es 0.0 se estaría diciendo que la fuente de luz se encuentra localizada a una distancia infinita del vector especificado por el arreglo.

En la siguiente sección se verá como moverse dentro de la escena trazada con las primitivas vistas en la sección anterior, e iluminadas con las funciones y comandos de ésta sección.

C.4. Manipulación del espacio 3D por transformación de coordenadas

La manipulación del espacio 3D por transformación de coordenadas permite rotar, trasladar y escalar objetos mediante matrices de transformación. Sin embargo, lo que realmente interesa de la manipulación del espacio 3D, es la capacidad de establecer una posición y una perspectiva dentro de la escena trazada. Esto posibilitaría la navegación virtual dentro del modelo 3D.

Las transformaciones hacen posible proyectar las coordenadas en tres dimensiones, del modelo 3D reconstruido, en las coordenadas bidimensionales correspondientes al monitor de la PC. En seguida se verán éstas transformaciones de forma conceptual.

C.4.1. Transformaciones

Además de mapear las coordenadas 3D en coordenadas 2D, las transformaciones permiten rotar, mover, estirar, encoger y deformar objetos. Pero más que modificar directamente los objetos, una transformación modifica el sistema de coordenadas. Por ejemplo, una vez que se aplica una transformación de rotación, el objeto aparece rotado cuando se traza.

Entre el tiempo en el que se especifican los vértices de una primitiva, y el tiempo en que éstos tardan en aparecer en el monitor, pueden ocurrir tres tipos de transformación: vista, modelado y proyección [Richard00]. En esta sección se examinarán los principios de cada tipo de transformación, los cuales se encuentran resumidos en la Tabla C.2.

Tabla C.2: Resumen de las transformaciones de OpenGL

Transformación	Uso
Viewing	Especifica la localización del punto de vista (cámara)
Modeling	Mueve objetos al rededor de la escena
Modelview	Describe la dualidad de las transformaciones de vista y modelo
Projection	Corta y ajusta el volumen visible
Viewport	Escala la salida final sobre la ventana

Además de las transformaciones, existe un concepto importante conocido como *coordenadas del ojo*, el cual puede verse como el sistema de coordenadas absoluto de la pantalla. A continuación se describe con más detalle éste concepto.

Coordenadas del ojo

Las coordenadas del ojo no son coordenadas reales, pero sirven como sistema de referencia fijo en un marco de referencia común. OpenGL utiliza el sistema de coordenadas cartesiano, pero cuando no existe ninguna transformación, el sistema es idéntico al sistema de coordenadas del ojo. Esto se debe a que todas las transformaciones se describen en términos de sus efectos relativos sobre el sistema de coordenadas del ojo. La Figura C.12 presenta un ejemplo de un sistema de coordenadas rotado 45°, en sentido de las manecillas del reloj, con respecto a las coordenadas del ojo. El cuadro es trazado sobre el sistema rotado, así que aparece trazado con una rotación de 45°.

Ya que se ha descrito el concepto de coordenadas del ojo, en seguida se describirá como se reflejan cada una de las transformaciones sobre estas coordenadas.

Transformación *Viewing*

La transformación *viewing*, es decir sobre el punto de vista, es la primera que se aplica en la escena. Es utilizada para determinar el punto de desvanecimiento de la escena. Por omisión, el punto de vista esta localizado en el origen del sistema de coordenadas (0,

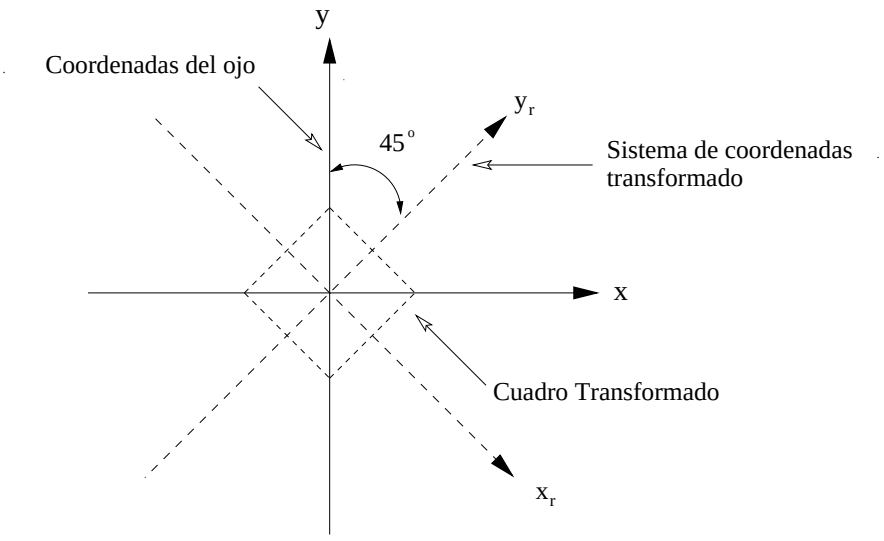


Figura C.12: Sistema de coordenadas rotado 45° con respecto al las coordenadas del ojo.

0, 0) mirando en dirección del eje $-Z$, por lo que los objetos trazados sobre $+Z$ estarán detrás del observador.

La transformación sobre la vista permite colocarse en cualquier punto de observación dentro de la escena, así como en cualquier dirección. Determinar la transformación de vista es como colocar y orientar una cámara en la escena.

Transformaciones *Modeling*

Las transformaciones *modeling*, generalmente son utilizadas para manipular los objetos inmersos en la escena; permiten mover, rotar y escalar objetos. En nuestro caso, se utilizan para simular los movimientos del robot y del sistema de giro inclinación, visualizando la porción del modelo que el robot vería en una posición en particular. El modelo representativo del robot es un ejemplo de la aplicación, a objetos individuales, de tales transformaciones (ver Figuras: 1.1, 1.2, 2.9 y 2.14(a)).

La apariencia final de un objeto, después de aplicar una serie de transformaciones *modeling*, depende del orden en que éstas sean aplicadas. Por ejemplo, el resultado final de aplicar una traslación seguida de una rotación no es igual si se aplica primero la rotación y después la traslación. Esto se debe a que internamente OpenGL realiza un producto matricial para realizar cualquiera de las transformaciones, y el producto de matrices no es

conmutativo. En la Sección C.4.2 se abordarán más a detalle las matrices de transformación.

Transformación *Modelview*

Las transformaciones de *viewing* y *modeling* realmente son la misma, en términos de sus efectos internos, así como de sus efectos sobre la apariencia final de la escena. La distinción entre las dos generalmente es realizada por los programadores [Richard00]. En nuestro caso se optó por realizar la distinción, ya que es más natural pensar en la navegación como el desplazamiento del punto de vista dentro de la escena, que como el desplazamiento del sistema de coordenadas.

Entonces, el término *modelview* solo indica la forma de ver las transformaciones de rotación, traslación y escalamiento, como transformación *viewing* o transformación *modeling*, aunque no exista una diferencia real entre ellas.

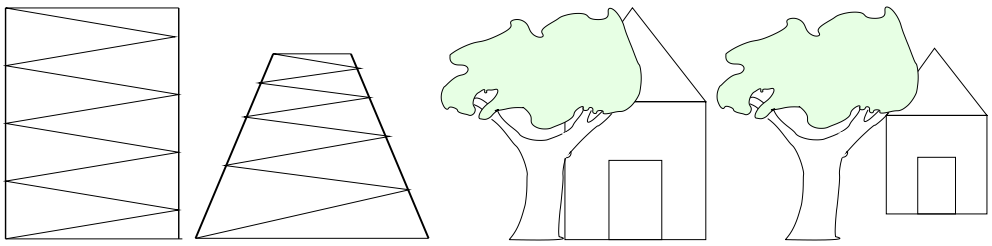
Transformación *projection*

La transformación *projection* se aplica sobre la orientación del punto de vista del modelo final. Esta "proyección" define el volumen visible y establece los planos de corte. Más específicamente, la transformación *projection* determina cómo una escena, ya finalizada, es traducida a la imagen proyectada sobre la pantalla. En OpenGL existen dos tipos de proyecciones: ortogonal y perspectiva.

En una proyección ortogonal, todos los polígonos son trazados sobre la pantalla con exactamente las mismas dimensiones relativas. Un ejemplo de una proyección ortogonal es el trazo de gráficas en dos dimensiones. Mientras que en una proyección en perspectiva, los objetos y las escenas se muestran más similares a como aparecen en el mundo real. Objetos definidos con el mismo tamaño pueden aparecer de diferente dimensión si está uno más cerca que el otro del punto de vista. El ejemplo más claro para diferenciar ambas proyecciones son las líneas paralelas en el espacio 3D, en una proyección ortogonal realmente aparecen paralelas (Figura C.13(a)), pero en una proyección en perspectiva parecen converger en un punto distante (Figura C.13(b)).

Transformación *Viewport*

Cuando todo se ha descrito y realizado, el paso final es mapear la escena 3D a la ventana desplegada en la pantalla o monitor. Este mapeo de la escena a las coordenadas



(a) Ortogonal. (b) Perspectiva. (c) Ortogonal. (d) Perspectiva.

Figura C.13: Ejemplos de proyecciones ortogonal contra proyectiva.

físicas de la pantalla es llamado transformación *viewport*. La transformación ocurre cuando se crea por primera vez la ventana, así como cuando se modifican las dimensiones físicas de ella.

La función de OpenGL encargada de definir el *viewport*, la porción visible de la escena, es: *glViewport*. Dicha función recibe 4 parámetros, los dos primeros definen las coordenadas x, y de la esquina superior izquierda de la ventana, los otros dos parámetros definen el ancho y alto de la ventana respectivamente. La dimensión de los parámetros esta dada en píxeles.

Establecidas de manera conceptual las transformaciones anteriores, en seguida se verán las matrices de transformación que emplea OpenGL para aplicarlas.

C.4.2. Matrices de transformación

Principalmente dos son las matrices que se deben modificar para navegar en el modelo 3D reconstruido: matriz *modelview* y matriz *projection*. OpenGL proporciona funciones de alto nivel que pueden ser invocadas para realizar las transformaciones.

Existe una especie de "flujo" de transformaciones para llegar de un simple vértice en el espacio 3D, a las coordenadas de un píxel sobre la ventana desplegada en la pantalla (ver Figura C.14).

El flujo de transformaciones comienza con la representación, en una matriz de 1 por 4, del vértice en el espacio 3D, donde los primeros 3 valores representan las coordenadas x, y y z. Como OpenGL trabaja con coordenadas homogéneas, el cuarto valor w es el factor de escala. Entonces, el vértice es multiplicado por la matriz *modelview* para transformarlo

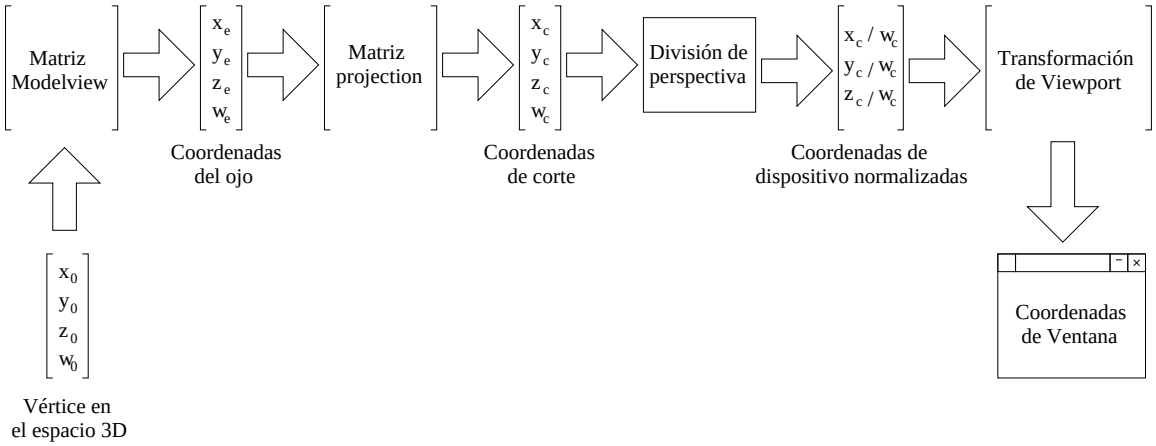


Figura C.14: Flujo de transformación de un vértice en coordenadas del espacio 3D a coordenadas de la ventana [Richard00].

en coordenadas del ojo. Las coordenadas del ojo son multiplicadas por la matriz *projection* para obtener las coordenadas de corte. Lo anterior elimina de la escena todo aquello que este fuera del volumen visible, es decir, lo que quede fuera del volumen visible no será trazado. Después se realiza la división de perspectiva, que no es otra cosa que dividir las coordenadas x, y y z entre la coordenada w, para generar las coordenadas de dispositivo normalizadas. Finalmente, las tres coordenadas se mapean sobre el plano 2D de la ventana por medio de la transformación *viewport*. OpenGL realiza esta última transformación internamente, en función de los valores especificados en *glViewport* y el tamaño actual de la ventana.

La matriz *modelview*, es una matriz de 4 por 4 que representa la transformación del sistema de coordenadas usado para colocar y orientar los objetos en la escena. Como se mencionó anteriormente, en nuestro caso particular, la transformación *modelview* es considerada como una transformación del cambio de punto de vista (*viewing*), es decir, el modelo 3D reconstruido se considera como el único objeto en la escena.

El vértice en el espacio 3D es tomado como matriz columna y multiplicado por la matriz *modelview*, ésto produce las nuevas coordenadas transformadas en relación al sistema de coordenadas del ojo. OpenGL permite especificar una matriz de 4 por 4 para aplicar cualquier tipo de transformación que se desee. Pero también cuenta con funciones de alto nivel para trasladar, rotar y escalar objetos.

Para aplicar una transformación de traslación, se utiliza la función *glTranslatef*. Esta función toma como parámetros la cantidad a trasladarse a lo largo de la dirección de

x, y y de z. Cuando se invoca esta función se construye la matriz apropiada para realizar la multiplicación, que internamente OpenGL ejecuta.

La función encargada de la rotación se llama *glRotatef*. Al invocar *glRotatef* también se crea la matriz apropiada para la multiplicación. Los parámetros que recibe son cuatro, el primero determina el ángulo de rotación especificado en grados, y en contra del sentido de las manecillas del reloj. Los tres parámetros restantes especifican un eje de rotación arbitrario como un vector de coordenadas (x,y,z); en el más simple de los casos el vector se encuentra sobre alguno de los ejes.

El escalamiento incrementa o decrementa el tamaño de un objeto expandiendo todos sus vértices a lo largo de los tres ejes en función de un factor [Richard00]. La función para escalar es: *glScale*. Los parámetros que recibe son los factores de escalación en x, y y z.

Cuando se aplica una transformación sus efectos son acumulativos, esto es, cada vez que se usa una función de transformación, se crea la matriz apropiada y se multiplica por la matriz *modelview* actual. La nueva matriz actual inicia con la matriz *modelview* actual, la cual es ahora multiplicada por la matriz de la siguiente transformación y así sucesivamente. La forma de reinicializar las transformaciones es cargando la matriz *modelview* con la matriz identidad. Con ello se especifica que no ha ocurrido ninguna transformación. Cuando se tracen nuevos vértices aparecerán en las coordenadas especificadas, esto se debe a que ahora la matriz *modelview* actual es la identidad, y el producto de la matriz columna, asociada cualquier vértice, por la matriz identidad es igual a la misma matriz columna. La función encargada de actualizar la matriz *modelview* con la matriz identidad es: *glLoadIdentity();*.

Otro concepto importante, cuando se trabaja con matrices de transformación en OpenGL, es *la pila de matrices*. No siempre es deseable reinicializar la matriz *modelview* con la matriz identidad después de aplicar cualquier número de transformaciones. Por ejemplo, se podría desear que la transformación actual fuera guardada, después aplicar nuevas transformaciones, para posteriormente restablecer la transformación guardada. Para lograr ésto, OpenGL mantiene una pila de matrices de transformación, tanto para las matrices *modelview* como para las matrices *projection*. La pila de matrices trabaja tal como lo haría un programa de pila ordinario [Richard00]. La pila de OpenGL tiene dos funciones para meter y sacar elementos de la pila: *glPushMatrix* y *glPopMatrix*, respectivamente. La matriz en el tope de la pila es tomada como la matriz *modelview* acutal, o como la matriz *projection* según sea el caso. Se debe tener cuidado con la profundidad de las pilas, ya que es limitada.

Como ya se mencionó anteriormente, existen dos tipos de proyecciones: ortogonal y perspectiva. La proyección ortogonal se ha mencionado en secciones anteriores, por lo que en seguida se abordará más a detalle la proyección en perspectiva.

C.4.3. Proyección en perspectiva

Una proyección en perspectiva ejecuta una división en perspectiva para acortar y ajustar los objetos que se encuentran a una cierta distancia del punto de vista. El ancho atrás del volumen visible no tiene las mismas medidas que el ancho de la parte de enfrente. Así, un objeto que tiene las mismas medidas, aparecerá más grande, cuando es trazado, si está en la parte de enfrente del volumen visible que si está en la parte de atrás del volumen. En OpenGL esto se logra mediante una forma geométrica llamada *frustum*.

Un *frustum* (ver Figura C.15) es una sección de una pirámide vista desde la punta hasta la parte ancha [Richard00]. En esta figura, también es posible observar la diferencia entre el tamaño de la parte de enfrente y la parte de atrás de la proyección en perspectiva, la cual está definida por el *frustum*. La parte de atrás se aprecia de mayor dimensión que la parte de enfrente, esto es por que lejos del punto de vista es posible ver una mayor área de la escena, contrario a si se estuviera cerca del punto de vista.

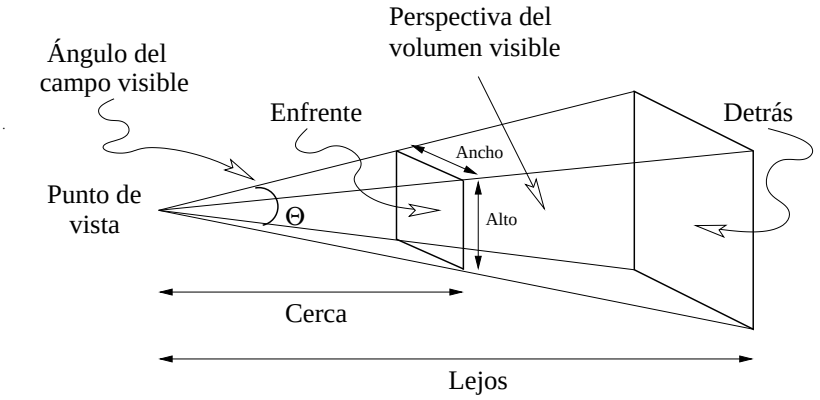


Figura C.15: Proyección en perspectiva definida por un *frustum*.

OpenGL permite definir el *frustum* mediante la función *glFrustum*. Sin embargo, la forma más usual de definir la proyección en perspectiva mediante un *frustum* es utilizando la función llamada: *gluPerspective*. Los parámetros para esta función son: el ángulo del campo visible en la dirección vertical, la razón de aspecto entre lo alto y ancho, y las distancias

cercana y lejos de los planos de corte de enfrente y detrás con respecto al punto de vista (ver Figura C.15).

Por último, en la siguiente sección se comenta, de manera breve, cómo interactuar con las ventanas utilizando GLUT.

C.5. GLUT

Las utilerias del conjunto de herramientas GLUT (*OpenGL Utility Toolkit*) son una interfaz de programación vinculadas con ANSIC y FORTRAN para escribir programas en OpenGL independientes del sistema de ventanas [Kilgard96]. El conjunto de herramientas (biblioteca) se sirve de funciones OpenGL, añadiendo funcionalidades como: inicialización y creación de ventanas, control de eventos de entrada, creación de menús, manejo de procesos en segundo plano, ejecución del programa, entre otras.

Antes de abrir una ventana, se deben especificar sus características: simple o doble *buffer*, manejo de color como RGBA o tabla de color, en que posición debe aparecer. Las funciones presentadas en la Tabla C.3 permiten dar tales especificaciones.

Tabla C.3: Funciones para inicializar y crear ventanas

Función	Parámetros	Descripción
glutInit	*int argc, char **argv*	Procesa la línea de comandos
glutInitDisplayMode	*unsigned int modo*	Determina el modo de desplegado
glutInitWindowSize	*int ancho, int alto*	Fija el ancho y alto de la ventana
glutInitWindowPosition	*int x, int y*	Localiza la ventana en la posición x, y
glutCreateWindow	*char *nombre*	Abre una ventana con el conjunto de características ya señaladas

La función *glutInit* debe ser llamada antes que cualquier otra función de GLUT, ya que es la encargada de inicializar la biblioteca GLUT. Los parámetros que recibe permiten especificar si la ventana debe mostrarse como ícono o maximizada, según lo permita el gestor de ventanas utilizado.

Por medio de una mascara de bits, *glutDisplayMode* determina el modo de desplegado de la ventana. Es aquí donde se determinan características como si se utilizará doble *buffer*, modo RGBA, o tabla de color. Con *glutWindowSize* se establece el ancho y alto de la ventana, y *glutWindowPosition* localiza la esquina superior izquierda de la ventana en

cierta posición. Finalmente *glutCreateWindow* abre y muestra la ventana con el conjunto de características especificadas por las funciones anteriores.

Cuando se utiliza GLUT, las aplicaciones se estructuran para manejar sus eventos utilizando funciones de *callback* [Shreiner00]. Estas funciones son ejecutadas cada que ocurre un evento dentro de un ciclo principal, cuando la ejecución de dicha función termina, el control retorna al ciclo. Después de que una ventana es creada, pero antes del ciclo principal, se deben registrar las funciones de *callback* utilizando rutinas como las presentadas en la Tabla C.4.

Tabla C.4: Rutinas para manejar ventanas y eventos

Rutina	Parámetros	Descripción
glutDisplayFunc	*void*(func)(void)*	Especifica la función que será llamada si el contenido de la ventana necesita ser trazado nuevamente.
glutReshapeFunc	*void*(func)(int, int)*	Especifica la función que será llamada si la ventana fue redimensionada o movida
glutKeyboardFunc	*void*(func)(unsigned int, int, int ,int)*	Especifica la función que será llamada si se presiona una tecla de valor ASCII
glutMouseFunc	*void*(func)(int, int,) int, int)*	Especifica la función que será llamada si se presiona un botón del ratón
glutPostRedisplay	*void*	Marca la ventana actual para ser trazada nuevamente

Los parámetros que recibe la función *glutReshapeFunc*, además de especificar la función que será llamada, permiten conocer el ancho y alto de las nuevas dimensiones de la ventana. Con los parámetros de tipo *int* recibidos en la función *glutKeyboardFunc* se especifica la tecla de valor ASCCI presionada, así como la posición x, y del ratón en el momento de la pulsación. Además de la función anterior existe otra para manejar las pulsaciones de las teclas especiales: *gluSpecialFunc*; recibe el mismo tipo de parámetros a excepción de la tecla de valor ASCCI.

Después de que toda la inicialización esta completa, las aplicaciones GLUT entran en un ciclo principal de procesamiento de eventos. La función empleada para realizar ésta tarea es: *glutMainLoop*.

Apéndice D

Programas de OpenGL

El código realizado para lograr la reconstrucción 3D se puede ver como los siguientes módulos:

1. Programación del control de la base del robot y sistema de giro inclinación (*pan-tilt*)
2. Programación de la metodología de localización local y construcción de mapas 2D mediante el alineamiento de dos rastreos consecutivos
3. Programación en OpenGL de la metodología para construir el modelo 3D a partir de los datos sensados con el telémetro láser del ambiente real.

En este apéndice se presentará la codificación de este último módulo. Que a su véz consta de dos módulos principales: (1) transformación de los datos de rango en vértices 3D, y (2) generación, texturización y visualización de mallas poligonales. Además existe otro módulo para generar archivos en formato postscript encapsulado, el cual fue tomado de [Geuzaine04]. En seguida se muestran los códigos correspondientes a dichos módulos principales.

D.1. Transformación de los datos de rango en vértices 3D

El código para transformar los datos de rango en vértices 3D esta organizado en tres archivos. El primer archivo contiene las macros necesarias para realizar las operaciones de construcción, multiplicación y premultiplicación de matrices en coordenadas homogéneas. El código es el siguiente:

```
/*
Archivo: transgehom.c
Autor: JJAP
Proposito: Contiene las macros para realizar transformaciones geometricas en coordenadas homogeneas.
/

#define TAMH3 4

#define NUEVA_MATRIZ_TRASLACION3D(nmt,t){
   nmt[0][0]=1; nmt[0][1]=0; nmt[0][2]=0; nmt[0][3]=t[0];
   nmt[1][0]=0; nmt[1][1]=1; nmt[1][2]=0; nmt[1][3]=t[1];
   nmt[2][0]=0; nmt[2][1]=0; nmt[2][2]=1; nmt[2][3]=t[2];
   nmt[3][0]=0; nmt[3][1]=0; nmt[3][2]=0; nmt[3][3]=t[3];
}

#define NUEVA_MATRIZ_ROTACIONX3D(nmt,ang){
   nmt[0][0]=1; nmt[0][1]=0; nmt[0][2]=0; nmt[0][3]=0;
   nmt[1][0]=0; nmt[1][1]=cos(ang); nmt[1][2]=-sin(ang); nmt[1][3]=0;
   nmt[2][0]=0; nmt[2][1]=sin(ang); nmt[2][2]= cos(ang); nmt[2][3]=0;
   nmt[3][0]=0; nmt[3][1]=0; nmt[3][2]=0; nmt[3][3]=1;
}

#define NUEVA_MATRIZ_ROTACIONY3D(nmt, ang){
   nmt[0][0]= cos(ang); nmt[0][1]=0; nmt[0][2]=sin(ang); nmt[0][3]=0;
   nmt[1][0]=0; nmt[1][1]=1; nmt[1][2]=0; nmt[1][3]=0;
   nmt[2][0]=-sin(ang); nmt[2][1]=0; nmt[2][2]=cos(ang); nmt[2][3]=0;
   nmt[3][0]=0; nmt[3][1]=0; nmt[3][2]=0; nmt[3][3]=1;
}

#define NUEVA_MATRIZ_ROTACIONZ3D(nmt, ang){
   nmt[0][0]=cos(ang); nmt[0][1]=-sin(ang); nmt[0][2]=0; nmt[0][3]=0;
   nmt[1][0]=sin(ang); nmt[1][1]= cos(ang); nmt[1][2]=0; nmt[1][3]=0;
   nmt[2][0]=0; nmt[2][1]=0; nmt[2][2]=1; nmt[2][3]=0;
   nmt[3][0]=0; nmt[3][1]=0; nmt[3][2]=0; nmt[3][3]=1;
}

#define NUEVA_MATRIZ_ESCALACION3D(nmt, s){
   nmt[0][0]=s[0]; nmt[0][1]=0; nmt[0][2]=0; nmt[0][3]=0;
   nmt[1][0]=0; nmt[1][1]=s[1]; nmt[1][2]=0; nmt[1][3]=0;
   nmt[2][0]=0; nmt[2][1]=0; nmt[2][2]=s[2]; nmt[2][3]=0;
   nmt[3][0]=0; nmt[3][1]=0; nmt[3][2]=0; nmt[3][3]=1;
}

#define PREMULTIPLICA_MATRICES3D(mt1, mt2, mtr){
   mtr[0][0] = mt1[0][0]*mt2[0][0] + mt1[0][1]*mt2[1][0] + mt1[0][2]*mt2[2][0] + mt1[0][3]*mt2[3][0];
   mtr[0][1] = mt1[0][0]*mt2[0][1] + mt1[0][1]*mt2[1][1] + mt1[0][2]*mt2[2][1] + mt1[0][3]*mt2[3][1];
```

```
  mtr[0][2] = mt1[0][0]*mt2[0][2] + mt1[0][1]*mt2[1][2] + mt1[0][2]*mt2[2][2] + mt1[0][3]*mt2[3][2];
  mtr[0][3] = mt1[0][0]*mt2[0][3] + mt1[0][1]*mt2[1][3] + mt1[0][2]*mt2[2][3] + mt1[0][3]*mt2[3][3];
  mtr[1][0] = mt1[1][0]*mt2[0][0] + mt1[1][1]*mt2[1][0] + mt1[1][2]*mt2[2][0] + mt1[1][3]*mt2[3][0];
  mtr[1][1] = mt1[1][0]*mt2[0][1] + mt1[1][1]*mt2[1][1] + mt1[1][2]*mt2[2][1] + mt1[1][3]*mt2[3][1];
  mtr[1][2] = mt1[1][0]*mt2[0][2] + mt1[1][1]*mt2[1][2] + mt1[1][2]*mt2[2][2] + mt1[1][3]*mt2[3][2];
  mtr[1][3] = mt1[1][0]*mt2[0][3] + mt1[1][1]*mt2[1][3] + mt1[1][2]*mt2[2][3] + mt1[1][3]*mt2[3][3];
  mtr[2][0] = mt1[2][0]*mt2[0][0] + mt1[2][1]*mt2[1][0] + mt1[2][2]*mt2[2][0] + mt1[2][3]*mt2[3][0];
  mtr[2][1] = mt1[2][0]*mt2[0][1] + mt1[2][1]*mt2[1][1] + mt1[2][2]*mt2[2][1] + mt1[2][3]*mt2[3][1];
  mtr[2][2] = mt1[2][0]*mt2[0][2] + mt1[2][1]*mt2[1][2] + mt1[2][2]*mt2[2][2] + mt1[2][3]*mt2[3][2];
  mtr[2][3] = mt1[2][0]*mt2[0][3] + mt1[2][1]*mt2[1][3] + mt1[2][2]*mt2[2][3] + mt1[2][3]*mt2[3][3];
  mtr[3][0] = mt1[3][0]*mt2[0][0] + mt1[3][1]*mt2[1][0] + mt1[3][2]*mt2[2][0] + mt1[3][3]*mt2[3][0];
  mtr[3][1] = mt1[3][0]*mt2[0][1] + mt1[3][1]*mt2[1][1] + mt1[3][2]*mt2[2][1] + mt1[3][3]*mt2[3][1];
  mtr[3][2] = mt1[3][0]*mt2[0][2] + mt1[3][1]*mt2[1][2] + mt1[3][2]*mt2[2][2] + mt1[3][3]*mt2[3][2];
  mtr[3][3] = mt1[3][0]*mt2[0][3] + mt1[3][1]*mt2[1][3] + mt1[3][2]*mt2[2][3] + mt1[3][3]*mt2[3][3];
}

#define MULTIPLICA_MT_P(mt, p, pp){
  pp[0] = mt[0][0]*p[0] + mt[0][1]*p[1] + mt[0][2]*p[2] + mt[0][3]+p[3];
  pp[1] = mt[1][0]*p[0] + mt[1][1]*p[1] + mt[1][2]*p[2] + mt[1][3]+p[3];
  pp[2] = mt[2][0]*p[0] + mt[2][1]*p[1] + mt[2][2]*p[2] + mt[2][3]+p[3];
  pp[3] = mt[3][0]*p[0] + mt[3][1]*p[1] + mt[3][2]*p[2] + mt[3][3]+p[3];
}
```

El segundo archivo contiene la definicion de prototipos y variables globales referentes a las lecturas adquiridas del laser para trasformarlas en vértices 3D. El código es el siguiente:

```
/*
Archivo: get_data.h
Funcion: Definicion de prototipos y variables globales
  referentes a las lecturas adquiridas del laser
Nota: El archivo de las lecturas del rastreo del laser
  debe tener el siguiente formato:
  ang0 dist0 ang1 dis1 .... angn-1 distn-1 angn distn
  ang_giro_base desplazamiento ang_elevacion ang_giro_torre
/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define NVERTICES 361 /* Numero de vertices (lecturas) por rastreo del laser */
#define DMAXV 200.0 /* Distancia maxima etre vertices en milimetros */
```

```
#define DMINV 5.0 /* Distancia minima etre vertices en milimetros */
//#define DELTAZ 34 /* Distnacia del centro del laser al sistema de referencia X,Y,Z */
#define DELTAZ 45.0 /* Distancia en Z del sensor laser al sistema de referencia de rotacion y elevacion del pan-tilt*/
#define DELTAY 8.0 /* Distancia en Y del sensor laser al sistema de referencia de elevacion del pan-tilt*/

#define DEGTORAD(x) ((x)*0.017453292519943296)

struct vertice{
   float x, y, z;
};
typedef struct vertice VERTICE;

struct lista_vertices{
   VERTICE verts[NVERTICES];
   struct lista_vertices *ptr_sig;
   struct lista_vertices *ptr_ant;
};
typedef struct lista_vertices *LSTVERT;

VERTICE vertices[NVERTICES]; /* Coordenadas (x,y,z) correspondientes a cada lectura en milimetros */
float fDatos[NVERTICES*2]; /* Datos del rastreo: angulo (grados) y distancia (milimetros) */
float fDesp; /* Desplazamiento de la base del robot (milimetros) */
float fAng_giro_base; /* Angulo de giro de la base del robot (grados) */
float fAng_elev; /* Angulo de elevacion del pan-tilt (grados) */
float fAng_giro_torre; /* Angulo de giro del pan-tilt (grados) */

int get_data(FILE *fpd); /* Funcion principal para obtener los datos */

float get_long(VERTICE v1, VERTICE v2); /* Funcion para calcular la longitud entre dos vertices */

void ReduceToUnit(float vec[3]);
void calcNormal(VERTICE v[3], float out[3]);

LSTVERT crea_lista_nula();
int lista_vacia(LSTVERT l);
int insertar_lista(LSTVERT *l, VERTICE verts[NVERTICES]);
void borrar_lista(LSTVERT *l);
```

El tecer archivo contiene la implementación de las funciones necesarias para transformar las lecturas del télemetro láser en vértices 3D. El código es el siguiente:

```
/*
Archivo: get_data.c
Funcion: implementacion de las funciones utilizadas para obtener
   la informacion obtenida del rastreo del laser
```

```
Ultima modificacion: 4/Mayo/2004 (jjap)
/

#include "transgehom.h"
#include "get_data.h"

/*
Funcion: get_ang_dist(FILE)
Ambito: Local a este archivo
Proposito: Leer de archivo la informacion de angulo y distancia
por cada lectura del rastreo.
Retorna: Verdadero (1) en caso de exito, Falso (0) en caso de fallo.
/
static int get_ang_dist(FILE *fpd){
  int i;

  for(i = 0; i < NVERTICES*2; i+=2){
    if(fscanf(fpd," %f%f",&fDatos[i], &fDatos[i+1]) != 2)
      return 0;
  }
  return 1;
}

/*
Funcion: get_movimientos(FILE)
Ambito: Local a este archivo
Proposito: Leer de archivo la informacion de: angulo de giro de la base,
  desplasamiento de la base, angulo de elevacion (pan-tilt) y
  angulo de giro del pan-tilt.
Retorna: Verdadero (1) en caso de exito, Falso (0) en caso de fallo.
/
static int get_movimientos(FILE *fpd){
  if(fscanf(fpd," %f%f%f%f",&fDesp, &fAng_giro_base, &fAng_elev, &fAng_giro_torre) != 4)
    return 0;
  return 1;
}

/*
Funcion: calcula_vertices(void)
Ambito: Local a este archivo
Proposito: Calcula el correspondiente vertice, en coordenadas (x,y,z), de cada una
  de las lecturas reportadas por rastreo del laser; basandose en el angulo
  de disparo, el angulo de elevacion y la distancia reportada por la lectura.
  El angulo de rastreo es de cada 0.5 grados sobre el plano XZ comenzando con -90 direccion
  de -X pasando por 0 (direccion de Z) y finalizando en 90 X,
  y el angulo de elevacion sobre el eje Y
```

```
Retorna: Verdadero (1) en caso de exito, Falso (0) en caso de fallo.
/
static int calcula_vertices(void){
   int i, j;
   float mroty[TAMH3][TAMH3], mrotx[TAMH3][TAMH3];
   float mpre0[TAMH3][TAMH3], mpre1[TAMH3][TAMH3];
   float p[TAMH3], pp[TAMH3];
   float tyz[TAMH3], mtyz[TAMH3][TAMH3];

   for(i = 0, j = 0; i < NVERTICES; i++, j+=2){

      p[0] = fDatos[j+1] * sin(DEGTORAD(fDatos[j]));
      p[1] = 0;
      p[2] = fDatos[j+1] * cos(DEGTORAD(fDatos[j]));
      p[3] = 1;

      tyz[0]=0;
      tyz[1]= DELTAY;
      tyz[2]= DELTAZ;
      tyz[3]=1;
      NUEVA_MATRIZ_TRASLACION3D(mtyz,tyz);
      NUEVA_MATRIZ_ROTACIONX3D(mrotx,DEGTORAD(-fAng_elev));
      NUEVA_MATRIZ_ROTACIONY3D(mroty,DEGTORAD(fAng_giro_torre));

      PREMULTIPLICA_MATRICES3D(mroty,mtyz,mpre0);
      PREMULTIPLICA_MATRICES3D(mpre0,mrotx,mpre1);
      MULTIPLICA_MT_P(mpre1,p,pp);

      vertices[i].x = pp[0];
      vertices[i].y = pp[1];
      vertices[i].z = pp[2];
   }
   return 1;
}

/*
Funcion: get_data(FILE)
Ambito: funcion exportada
Proposito: Leer de archivo la informacion del angulo y distancia por
   cada lectura del rastreo, ademan el angulo de giro de la base,
   desplasamiento de la base, angulo de elevacion (pan-tilt);
Retorna: Verdadero (1) en caso de exito, Falso (0) en caso de fallo.
/
int get_data(FILE *fpd){
   int res = 1;
```

```
  if(!get_ang_dist(fpd))
    res = 0;
  if(!get_movimientos(fpd))
    res = 0;

  calcula_vertices();

  return res;
}

/*
Funcion: get_long(VERTICE v1, VERTICE v2)
Ambito: funcion exportada
Proposito: Calcular la longitud entre dos vertices (v1, v2)
Retorna: La longitud (milimetros).
/
float get_long(VERTICE v1, VERTICE v2){
  float i, j, k;

  i = v2.x - v1.x;
  j = v2.y - v1.y;
  k = v2.z - v1.z;

  return sqrt(i*i + j*j + k*k);
}

void ReduceToUnit(float vec[3]){
  float length;

  // Calculate the length of the vector
  length = (float) sqrt( (vec[0]*vec[0]) + (vec[1]*vec[1]) + (vec[2]*vec[2]));

  if(length == 0.0f)
  length = 1.0f;

  // Dividing each element by the length whill result in a unit notmal vector
  vec[0]/=length;
  vec[1]/=length;
  vec[2]/=length;
}

void calcNormal(VERTICE v[3], float out[3]){
  float v1[3], v2[3];
  static const int x = 0;
  static const int y = 1;
  static const int z = 2;
```

```
   // Calculate two vectors from the three points
   v1[x] = v[0].x - v[1].x;
   v1[y] = v[0].y - v[1].y;
   v1[z] = v[0].z - v[1].z;

   v2[x] = v[1].x - v[2].x;
   v2[y] = v[1].y - v[2].y;
   v2[z] = v[1].z - v[2].z;

   // Take the cross product of the two vectors to get
   // the normal vector, which will be stored in out[]
   out[x] = v1[y]*v2[z] - v1[z]*v2[y];
   out[y] = v1[z]*v2[x] - v1[x]*v2[z];
   out[z] = v1[x]*v2[y] - v1[y]*v2[x];

   // Normalize the vector (shorten length to one)
   ReduceToUnit(out);
}

////////////////////////////
// funciones para la lista //
////////////////////////////
LSTVERT crea_lista_nula(){
   return NULL;
}

int lista_vacia(LSTVERT l){
   return l == NULL;
}

int insertar_lista(LSTVERT *l, VERTICE verts[NVERTICES]){
   LSTVERT new, temp;
   int i;

   new = malloc(sizeof(struct lista_vertices));
   if(new == NULL) {
      perror("Memoria insuficiente para temp (insertar_lista)");
   return 0;
   }
   for(i = 0; i < NVERTICES; i++){
      new->verts[i].x = verts[i].x;
      new->verts[i].y = verts[i].y;
      new->verts[i].z = verts[i].z;
   }
   new->ptr_sig = NULL;
```

```
  if(*l == NULL){
    *l = new;
  }
  else{
    temp = *l;
    while(temp->ptr_sig != NULL)
      temp = temp->ptr_sig;
      temp->ptr_sig = new;
    }
  return 1;
}

void borrar_lista(LSTVERT *l){
  LSTVERT temp;
  while(!lista_vacia(*l)){
    temp = *l;
    *l = temp->ptr_sig;
    free(temp);
  }
}
```

D.2. Construcción del modelo 3D con OpenGL

Este código contiene las funciones y comandos de OpenGL necesarios para realizar la construcción del modelo 3D a partir de una lista de vértices, obtenida con el código anterior.

```
/*
Opengl draw_laser. Construccion de modelo 3D

/

#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

#include "gl2ps.h"

#include "get_data.h"
```

```
/*
Globals...
/
int Width; /* Width of window */
int Height; /* Height of window */
int win; /* Identificador de la ventana principal */
FILE *fpd; /* File pointer to data laser */
char szDataFile[256]; /* Name of data file */
int banPerspective; /* Perspective or Ortho */
int banDots; /* Puntos del laser */
int banQuads; /* Graficar cuadros */
int banFill; /* Graficar con relleno */
int epuntos; /* Puntos a eliminar de los extremos */

GLfloat ambientLight[] = /* Bright white light */
{0.3f, 0.3f, 0.3f, 1.0f};
GLfloat diffuseLight[] =
{0.3f, 0.3f, 0.3f, 1.0f};

LSTVERT lstverts; /* lista de vertices */

struct CAMERA
{
   float position[3];
   float orientation[3];
} cameraData;

struct BASE
{
   float position[3];
   float orientation;
} baseData;

/*
Functions...
/
void Redraw(void);
void Resize(int width, int height);
void SetupViewing(void);
void InitWorld(void);
void KeyboardSpecialFunc(int kf, int mouseX, int mouseY);
void KeyboardFunc(unsigned char k, int mouseX, int mouseY);
void leer_vertices(void);
int valida_vertice(VERTICE v1, VERTICE v2);
void writefile(int format, int sort, int options, int nbcol,
char *filename, char *extension);
```

```
/*
'main()' -
/
int /* O - Exit status */
main(int argc, /* I - Number of command-line arguments */
   char *argv[]) /* I - Command-line arguments */
{
   if(argc < 2){
      printf("Faltan parametros.\Uso:%s archivodatos.txt\",argv[0]);
      exit(1);
   }
   strcpy(szDataFile,argv[1]);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGB — GLUT_DOUBLE — GLUT_DEPTH);
   glutInitWindowSize(792, 573);
   win = glutCreateWindow("Draw_laser");

   glutDisplayFunc(Redraw);
   glutReshapeFunc(Resize);
   glutSpecialFunc(KeyboardSpecialFunc);
   glutKeyboardFunc(KeyboardFunc);


   InitWorld();

   glutMainLoop();
   return (0);
}

/*
'InitWorld()' -
/
void
InitWorld(void)
{
   // Camera initial position and orientation
   // Initially at Origin, 55cm above the ground
   cameraData.position[0] = 0.0f;
   cameraData.position[1] = -5.5f;
   cameraData.position[2] = -3.2f;

   // Looking down negative Z Axis (North). Positive X is to the right
   cameraData.orientation[0] = 0.0f;
   cameraData.orientation[1] = 0.0f;
   cameraData.orientation[2] = 0.0f;
```

```
   // Base initial position and orientation
   // Initially at Origin, on the ground
   baseData.position[0] = 0.0f;
   baseData.position[1] = 0.0f;
   baseData.position[2] = 0.0f;
   baseData.orientation = 0.0f;
/*
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_CULL_FACE);
   glFrontFace(GL_CW);
/
   //glClearColor(0.7, 0.7, 1.0, 1.0);
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glPolygonMode(GL_FRONT, GL_LINE);
   glPolygonMode(GL_BACK, GL_LINE);

   banPerspective = 1;
   banDots = 0;
   banQuads = 1;
   banFill = 0;

   epuntos = 0;
   lstverts = crea_lista_nula();
   leer_vertices();
}

void leer_vertices(void){

   if((fpd = fopen(szDataFile,r")) == NULL){
      printf("Imposible abrir el archivo %s\",szDataFile);
      exit(1);
   }

   while(!feof(fpd)){
      get_data(fpd);
      insertar_lista(&lstverts, vertices);
   }
   fclose(fpd);
}

int valida_distancia(VERTICE v1, VERTICE v2 ){
   float long_vertice;

   long_vertice = get_long(v1,v2);
   if(long_vertice <= DMAXV && long_vertice >= DMINV)
      return 1;
```

```
  return 0;
}

int valida_cuadrilatero(VERTICE v1, VERTICE v2, VERTICE v3, VERTICE v4)
{
  if(!valida_distancia(v1, v2))
    return 0;
  if(!valida_distancia(v1, v3))
    return 0;
  if(!valida_distancia(v1, v4))
    return 0;
  return 1;
}

int valida_triangulo(VERTICE v1, VERTICE v2, VERTICE v3)
{
  if(!valida_distancia(v1, v2))
    return 0;
    if(!valida_distancia(v1, v3))
    return 0;
  return 1;
}

/*write file eps from gl*/
void writefile(int format, int sort, int options, int nbcol,
  char *filename, char *extension){
  FILE *fp;
  char file[256];
  int state = GL2PS_OVERFLOW, buffsize = 0;
  GLint viewport[4];

  strcpy(file, filename);
  strcat(file, ".");
  strcat(file, extension);

  viewport[0] = 0;
  viewport[1] = 0;
  viewport[2] = Width;
  viewport[3] = Height;

  fp = fopen(file, "wb");

  if(!fp){
    printf("Ünable to open file %s for writing\", file);
    exit(1);
```

```
  }

  printf("Saving image to file %s... ", file);
  fflush(stdout);

  while(state == GL2PS_OVERFLOW){
    buffsize += 1024*1024;
    gl2psBeginPage(file, "test", viewport, format, sort, options,
      GL_RGBA, 0, NULL, nbcol, nbcol, nbcol,
      buffsize, fp, file);
    Redraw();
    state = gl2psEndPage();
  }

  fclose(fp);

  printf("Done!\");
  fflush(stdout);
}



/* The keyboard callback function: */
void KeyboardFunc(unsigned char k, int mouseX, int mouseY)
{
  float fAngularVelocityX = 0.0f;
  float fAngularVelocityY = 0.0f;
  float fAngle;

  //gl2ps
  int opt;
  char *ext;
  static int format = GL2PS_EPS;
  //

  switch (k)
  {
  case 'w': /* up tilt */
  case 'W':
    fAngularVelocityX = -3.409f;
  break;
  case 's': /* down tilt */
  case 'S':
    fAngularVelocityX = 3.409f;
  break;
  case 'a':
```

```
case 'A':
  fAngularVelocityY = -21.77415;
break;
case 'd':
case 'D':
  fAngularVelocityY = 21.77415;
break;
case 'p':
case 'P':
  banPerspective = 1;
  Resize(Width, Height);
break;
case 'o':
case 'O':
  banPerspective = 0;
  Resize(Width, Height);
  break;
case 'x':
case 'X':
  banDots = !banDots;
  Resize(Width, Height);
break;
case 'c':
case 'C':
  banQuads = !banQuads;
  Resize(Width, Height);
break;
case 'f':
case 'F':
  banFill = !banFill;
  Resize(Width, Height);
break;
case 27:
  borrar_lista(&lstverts);
  exit(0);
break;
case 'e': //gl2ps
case 'E':
  //opt = GL2PS_DRAW_BACKGROUND;
  //ext = (format == GL2PS_EPS) ? "eps": "pdf";
  //writefile(format, GL2PS_SIMPLE_SORT, opt, 0, .ºutSimple", ext);
  opt = GL2PS_BEST_ROOT — GL2PS_OCCLUSION_CULL;
  ext = (format == GL2PS_EPS) ? "eps": "PS";
  writefile(format, GL2PS_BSP_SORT, opt, 0, .ºutSimple", ext);
break;
case '+':
```

```
      if(epuntos < NVERTICES)
        epuntos++;
      Resize(Width, Height);
    break;
    case '-':
      if(epuntos > 0)
        epuntos--;
      Resize(Width, Height);
    break;
    }

    fAngle = fAngularVelocityX;
    fAngle += cameraData.orientation[0];

    if((fAngle < 90.0f) && (fAngle > -90.0f))
      cameraData.orientation[0] = fAngle;

    cameraData.orientation[1] += fAngularVelocityY;

    glutPostRedisplay();
}

/* The keyboard Special Function callback function: */
void KeyboardSpecialFunc(int kf, int mouseX, int mouseY)
{
    float fLinearVelocity = 0.0f;
    float fAngularVelocityY = 0.0f;
    float fAngularVelocityZ = 0.0f;
    float fXDelta,fZDelta;

    switch (kf)
    {
    case GLUT_KEY_UP:
      fLinearVelocity = 0.57125f;
    break;
    case GLUT_KEY_DOWN:
      fLinearVelocity = -0.57125f;
    break;
    case GLUT_KEY_LEFT:
      fAngularVelocityY = -2.34987f;
    break;
    case GLUT_KEY_RIGHT:
      fAngularVelocityY = 2.34987f;
    break;
    }
```

```
  baseData.orientation += fAngularVelocityY;

  // Update linear position
  fXDelta = fLinearVelocity * (float)(sin(DEGTORAD(baseData.orientation)));
  fXDelta += baseData.position[0];
  fZDelta = fLinearVelocity * (float)(cos(DEGTORAD(baseData.orientation)));
  fZDelta += baseData.position[2];

  baseData.position[0] = fXDelta;
  baseData.position[2] = fZDelta;

  glutPostRedisplay();
}

/*
'SetupViewing()' - Setup viewing transformations for the current
position and orientation...
/
void
SetupViewing(void)
{
  //Get the new camera position and update the viewing transformation
  // accordingly
  //UpdatePosition(&cameraData); // Based on keyboard

  // Position lights, and camera location
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity(); // Z-X-Y (Boom specific)

  glRotatef(cameraData.orientation[2], 0.0f, 0.0f, 1.0f);
  glRotatef(cameraData.orientation[0], 1.0f, 0.0f, 0.0f);
  glRotatef(cameraData.orientation[1]+baseData.orientation, 0.0f, 1.0f, 0.0f);
  glTranslatef(-baseData.position[0],cameraData.position[1],baseData.position[2]);
}

/*
'Redraw()' - Redraw the window...
/
void
Redraw(void)
{
  int i;
  LSTVERT temp, temp_sig;
  float normal[3];
  //int cont_c = 0; //contador de cuadros validos.
  //int cont_t = 0; //contador de triangulos validos.
```

```
SetupViewing();

// Clear the window with current clearing color
glClear(GL_COLOR_BUFFER_BIT — GL_DEPTH_BUFFER_BIT);
if(banFill){
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);
  glEnable(GL_LIGHTING);
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
  glEnable(GL_LIGHT0);
  glEnable(GL_COLOR_MATERIAL);
  glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
  glPolygonMode(GL_BACK, GL_LINE);
  glPolygonMode(GL_FRONT, GL_FILL);
  glColor3f(0.95, 0.95, 0.95);
}else{
  glColor3f(0.0, 0.0, 0.0);
  glDisable(GL_LIGHTING);
  glDisable(GL_COLOR_MATERIAL);
  glPolygonMode(GL_BACK, GL_LINE);
  glPolygonMode(GL_FRONT, GL_LINE);
}

temp = lstverts;
temp_sig = lstverts->ptr_sig;
while(temp != NULL && temp_sig != NULL){
  for(i = epuntos; i < NVERTICES-1-epuntos; i++){

    if(banDots)
      glBegin(GL_POINTS);
    else if(banQuads)
      glBegin(GL_QUADS);
    else
      glBegin(GL_TRIANGLES);

    if(banDots —— banQuads){
      if(valida_cuadrilatero(temp->verts[i], temp->verts[i+1], temp_sig->verts[i], temp_sig->verts[i+1])){

        {VERTICE v[3] = {temp->verts[i],temp_sig->verts[i],temp_sig->verts[i+1]};
        calcNormal(v,normal);
        glNormal3fv(normal);}
        glVertex3f(temp->verts[i].x/100, temp->verts[i].y/100, temp->verts[i].z/100);
```

```
        {VERTICE v[3] = {temp_sig->verts[i],temp_sig->verts[i+1],temp->verts[i+1]};
        calcNormal(v,normal);
        glNormal3fv(normal);}
        glVertex3f(temp_sig->verts[i].x/100, temp_sig->verts[i].y/100, temp_sig->verts[i].z/100);

        {VERTICE v[3] = {temp_sig->verts[i+1],temp->verts[i+1],temp->verts[i]};
        calcNormal(v,normal);
        glNormal3fv(normal);}
        glVertex3f(temp_sig->verts[i+1].x/100, temp_sig->verts[i+1].y/100, temp_sig->verts[i+1].z/100);

        {VERTICE v[3] = {temp->verts[i+1],temp->verts[i],temp_sig->verts[i]};
        calcNormal(v,normal);
        glNormal3fv(normal);}
        glVertex3f(temp->verts[i+1].x/100, temp->verts[i+1].y/100, temp->verts[i+1].z/100);

        //cont_c++;
      }

    }
    else{
      if(valida_triangulo(temp->verts[i],temp_sig->verts[i],temp_sig->verts[i+1])){

        {VERTICE v[3] = {temp->verts[i],temp_sig->verts[i],temp_sig->verts[i+1]};
        calcNormal(v,normal);
        glNormal3fv(normal);}
        glVertex3f(temp->verts[i].x/100, temp->verts[i].y/100, temp->verts[i].z/100);
        glVertex3f(temp_sig->verts[i].x/100, temp_sig->verts[i].y/100, temp_sig->verts[i].z/100);
        glVertex3f(temp_sig->verts[i+1].x/100, temp_sig->verts[i+1].y/100, temp_sig->verts[i+1].z/100);

        //cont_t++;
      }
      if(valida_triangulo(temp_sig->verts[i+1],temp->verts[i+1],temp->verts[i])){
        {VERTICE v[3] = {temp_sig->verts[i+1],temp->verts[i+1],temp->verts[i]};
        calcNormal(v,normal);
        glNormal3fv(normal);}
        glVertex3f(temp_sig->verts[i+1].x/100, temp_sig->verts[i+1].y/100, temp_sig->verts[i+1].z/100);
        glVertex3f(temp->verts[i+1].x/100, temp->verts[i+1].y/100, temp->verts[i+1].z/100);
        glVertex3f(temp->verts[i].x/100, temp->verts[i].y/100, temp->verts[i].z/100);

        //cont_t++;
      }
    }
    glEnd();
  }
  temp = temp_sig;
  temp_sig = temp->ptr_sig;
```

```
    }

    //printf("\cuadros: %d\triangulos: %d\",cont_c, cont_t);
    /* Finish up */
    glutSwapBuffers();
}


/*
'Resize()' - Resize the window...
/
void
Resize(int width, /* I - Width of window */
    int height) /* I - Height of window */
{
    float fAspect;
    GLfloat lightPos[] = {0.0f, 5.5f, 0.0f, 1.0f};

    /* Save the new width and height */
    Width = width;
    Height = height;

    /* Reset the viewport... */
    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if(height == 0)
        fAspect = 1;
    else
        fAspect = (float)width / (float)height;

    if(banPerspective)
        gluPerspective(80.0, fAspect, 0.1, 100.0);
    else
        glOrtho(-100, 100, -100, 100, -100, 100);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
}
```

Apéndice E

Ajuste de líneas rectas

Este apéndice describe tres algoritmos para ajustar un conjunto de puntos a una línea recta, dos de estos problemas: mínimos cuadrados y estimador Lorentziano; son planteados como un problema de optimización. El tercer algoritmo se basa en el método de componentes principales o eigenvectores.

E.1. Ajuste por mínimos cuadrados

El método de mínimos cuadrados [Duda76] es el más usual para ajustar una línea recta a un conjunto de puntos. La componente y se asume funcionalmente dependiente de la componente x. El problema del ajuste se puede definir como sigue: dado un conjunto de puntos $\{(x_i, y_i)\}$ para $i = 1, 2, ..., n$ determinar a y b tal que la línea $y = ax + b$ sea la que mejor se ajuste al conjunto de puntos, en el sentido de que la sumatoria de los errores al cuadrado entre y_i y los valores de la línea $y = ax + b$ sea mínima.

Así, la ecuación del error a minimizar es:

$$E(a, b) = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2 \tag{E.1}$$

Para que sea un mínimo se debe cumplir que $\frac{\partial E(a,b)}{\partial a} = 0$ y $\frac{\partial E(a,b)}{\partial b} = 0$, y:

$$\frac{\partial E(a, b)}{\partial a} = -2 \sum_{i=1}^{n} [y_i - (ax_i + b)] x_i = 0 \tag{E.2}$$

$$\frac{\partial E(a, b)}{\partial b} = -2 \sum_{i=1}^{n} [y_i - (ax_i + b)] = 0 \tag{E.3}$$

De las ecuaciones E.2 y E.3 respectivamente se obtiene:

$$nb + a\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \tag{E.4}$$

$$b\sum_{i=1}^{n} x_i + a\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i \tag{E.5}$$

Así las Ecuaciónes E.4 y E.5, conocidas como ecuaciónes normales, resultan en un mínimo. Las cuales pueden escribirse en forma matricial de la siguiente manera:

$$\begin{bmatrix} \sum_{i=1}^{n} x_i & n \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix} \tag{E.6}$$

Resolviendo el sistema de ecuaciones que representa la notación matricial en E.6 se tiene una fórmula directa para calcular a y b:

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{E.7}$$

$$b = \frac{\left(\sum_{i=1}^{n} y_i\right)\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} x_i y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{E.8}$$

E.2. Ajuste mediante el estimador Lorentziano

Dado un conjunto de puntos $\{(x_i, y_i)\}$ para $i = 1, 2, ..., n$, se pretenden determinar los parámetros a y b de la ecuación de la recta $y = ax + b$ que se ajuste a la mayoría de los puntos. El estimador Lorentziano utiliza $\rho_\sigma(x) = \log(1 + \frac{1}{2}(\frac{x}{\sigma})^2)$ para ponderar el error. Así se tiene que la función de error $E(a, b)$ a minimizar es:

$$E(a, b) = \sum_{i=1}^{n} \rho([ax_i + b] - y_i) \tag{E.9}$$

es decir:

$$E(a, b) = \sum_{i=1}^{n} log\left(1 + \frac{1}{2}\left(\frac{[ax_i + b] - y_i}{\sigma}\right)^2\right) \tag{E.10}$$

Minimizar $E(a, b)$ implica resolver un sistema de ecuaciones simultáneas para a y b. Estas ecuaciones simultáneas están dadas por las condiciones que deben de cumplirse para que $E(a, b)$ sea un mínimo, y son sus correspondientes derivadas parciales igualadas a 0:

$$\begin{aligned} f_a(a,b) &= \frac{\partial E(a,b)}{\partial a} = 0 \\ f_b(a,b) &= \frac{\partial E(a,b)}{\partial b} = 0 \end{aligned} \tag{E.11}$$

Así para resolver el sistema de ecuaciones E.11, f_a y f_b son aproximadas mediante una linealización local [McCallum99], asumiendo que estas son diferenciables, se tiene que:

$$\begin{aligned} f_a(a+\delta a, b+\delta b) &\approx f_a(a,b) + \frac{\partial f_a}{\partial a}\delta a + \frac{\partial f_a}{\partial b}\delta b \\ f_b(a+\delta a, b+\delta b) &\approx f_b(a,b) + \frac{\partial f_b}{\partial a}\delta a + \frac{\partial f_b}{\partial b}\delta b \end{aligned} \tag{E.12}$$

Reagrupando y reescribiendo de forma matricial E.12 se obtiene un sistema de ecuaciones del tipo $A\ \delta X = -B$, es decir:

$$\begin{bmatrix} \frac{\partial f_a}{\partial a} & \frac{\partial f_a}{\partial b} \\ \frac{\partial f_b}{\partial a} & \frac{\partial f_b}{\partial b} \end{bmatrix} \begin{bmatrix} \delta a \\ \delta b \end{bmatrix} = - \begin{bmatrix} f_a \\ f_b \end{bmatrix} \tag{E.13}$$

De E.10 y E.3 se puede obtener una expresion para f_a:

$$\begin{aligned} f_a = \frac{\partial E(a,b)}{\partial a} &= \frac{\partial}{\partial a}\sum_{i=1}^{n} log\left(1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2\right) \\ &= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2} \frac{\partial}{\partial a}\left(1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2\right) \\ &= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2} \left(\frac{[ax_i+b]-y_i}{\sigma}\right)\left(\frac{x_i}{\sigma}\right) = \sum_{i=1}^{n} \frac{x_i([ax_i-b]-y_i)}{\sigma^2\left(1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right) \\ &= \sum_{i=1}^{n} \frac{2x_i([ax_i-b]-y_i)}{2\sigma^2+([ax_i-b]-y_i)^2} \end{aligned} \tag{E.14}$$

y para f_b:

$$
\begin{aligned}
f_b = \frac{\partial E(a,b)}{\partial b} &= \frac{\partial}{\partial b}\sum_{i=1}^{n} log\left(1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n}\frac{1}{1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2}\frac{\partial}{\partial b}\left(1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n}\frac{1}{1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)\left(\frac{1}{\sigma}\right) = \sum_{i=1}^{n}\frac{([ax_i-b]-y_i)}{\sigma^2\left(1+\frac{1}{2}\left(\frac{[ax_i+b]-y_i}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right) \qquad \text{(E.15)} \\
&= \sum_{i=1}^{n}\frac{2([ax_i-b]-y_i)}{2\sigma^2+([ax_i-b]-y_i)^2}
\end{aligned}
$$

y los elementos de $A_{3\times 3}$ son:

$$
\begin{aligned}
a_{1,1} = \frac{\partial f_a}{\partial a} &= \frac{\partial}{\partial a}\sum_{i=1}^{n}\frac{2x_i([ax_i+b]-y_i)}{2\sigma^2+([ax_i-b]-y_i)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\frac{\partial}{\partial a}2x_i([ax_i+b]-y_i)-\left(2x_i([ax_i+b]-y_i)\frac{\partial}{\partial a}\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)2x_i^2-(2x_i([ax_i+b]-y_i)2([ax_i+b]-y_i)x_i)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i^2\left(2\sigma^2+([ax_i+b]-y_i)^2\right)-4x_i^2([ax_i+b]-y_i)^2}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \qquad \text{(E.16)} \\
&= \sum_{i=1}^{n}\frac{2x_i^2\left(2\sigma^2+([ax_i+b]-y_i)^2-2([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i^2\left(2\sigma^2-([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2}
\end{aligned}
$$

$$
\begin{aligned}
a_{1,2} = \frac{\partial f_a}{\partial b} &= \frac{\partial}{\partial b}\sum_{i=1}^{n}\frac{2x_i([ax_i+b]-y_i)}{2\sigma^2+([ax_i-b]-y_i)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\frac{\partial}{\partial b}2x_i([ax_i+b]-y_i)-\left(2x_i([ax_i+b]-y_i)\frac{\partial}{\partial b}\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)2x_i-(2x_i([ax_i+b]-y_i)2([ax_i+b]-y_i)(1))}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i\left(2\sigma^2+([ax_i+b]-y_i)^2\right)-4x_i([ax_i+b]-y_i)^2}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \qquad \text{(E.17)} \\
&= \sum_{i=1}^{n}\frac{2x_i\left(2\sigma^2+([ax_i+b]-y_i)^2-2([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i\left(2\sigma^2-([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2}
\end{aligned}
$$

$$
\begin{aligned}
a_{2,1} = \frac{\partial f_b}{\partial a} &= \frac{\partial}{\partial a}\sum_{i=1}^{n}\frac{2([ax_i+b]-y_i)}{2\sigma^2+([ax_i-b]-y_i)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\frac{\partial}{\partial a}2([ax_i+b]-y_i)-\left(2([ax_i+b]-y_i)\frac{\partial}{\partial a}\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)2x_i-(2([ax_i+b]-y_i)2([ax_i+b]-y_i)x_i)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i\left(2\sigma^2+([ax_i+b]-y_i)^2\right)-4x_i([ax_i+b]-y_i)^2}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i\left(2\sigma^2+([ax_i+b]-y_i)^2-2([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2x_i\left(2\sigma^2-([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2}
\end{aligned}
\tag{E.18}
$$

$$
\begin{aligned}
a_{2,2} = \frac{\partial f_b}{\partial b} &= \frac{\partial}{\partial b}\sum_{i=1}^{n}\frac{2([ax_i+b]-y_i)}{2\sigma^2+([ax_i-b]-y_i)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\frac{\partial}{\partial b}2([ax_i+b]-y_i)-\left(2([ax_i+b]-y_i)\frac{\partial}{\partial b}\left(2\sigma^2+([ax_i+b]-y_i)^2\right)\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)2(1)-(2([ax_i+b]-y_i)2([ax_i+b]-y_i)(1))}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2\left(2\sigma^2+([ax_i+b]-y_i)^2\right)-4([ax_i+b]-y_i)^2}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2\left(2\sigma^2+([ax_i+b]-y_i)^2-2([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2} \\
&= \sum_{i=1}^{n}\frac{2\left(2\sigma^2-([ax_i+b]-y_i)^2\right)}{\left(2\sigma^2+([ax_i+b]-y_i)^2\right)^2}
\end{aligned}
\tag{E.19}
$$

Dado que la matriz $B = \nabla$ (Jacobiano) y la matriz $A = \nabla^2$ (Hessiano), finalmente se aplica el método de optimización no lineal Newton-Levenberg-Marquardt (ver Sección 3.4.2) para encontrar los parámetros a y b, tal que la recta $y = ax + b$ se ajusta a la mayoría de los puntos despreciando los datos atípicos gracias al estimador Lorentziano.

E.3. Ajuste por componentes principales y proyección ortogonal

En esta sección se aborda la metodología y heurística para calcular la línea recta que mejor se ajusta a un conjunto de puntos aplicando el método de componentes principales, así como también, el cálculo de la proyección ortogonal de un punto sobre la línea ajustada aprobechando la representación de la recta que se obtiene al aplicar dicho método de ajuste.

Ajuste de líneas por componentes principales

Los componentes principales, también llamados eigenvectores, sirven para obtener la dispersión de un conjunto de puntos[Gonzalez87]. La dispersión de los puntos está definida por los eigenvalores de la matriz de covarianza **C** asociada a esos puntos, dicha matriz esta dada por:

$$\mathbf{C} = \begin{bmatrix} C_{xx} & C_{xy} \\ C_{yx} & C_{yy} \end{bmatrix} \tag{E.20}$$

$$C_{xx} = \left(\frac{1}{N} \sum_{i=0}^{N-1} x_i^2 \right) - m_x^2 \tag{E.21}$$

$$C_{yx} = C_{xy} = \left(\frac{1}{N} \sum_{i=0}^{N-1} x_i y_i \right) - m_x m_y \tag{E.22}$$

$$C_{yy} = \left(\frac{1}{N} \sum_{i=0}^{N-1} y_i^2 \right) - m_y^2 \tag{E.23}$$

Donde m_x y m_y corresponden al centro de masa (centroide) del conjunto de puntos, y se obtienen mediante:

$$m_x = \frac{1}{N} \sum_{i=0}^{N-1} x_i \tag{E.24}$$

$$m_y = \frac{1}{N} \sum_{i=0}^{N-1} y_i \tag{E.25}$$

A partir de los valores propios (eigenvalores) de la matriz **C** se calculan los eigenvectores, los cuales dan la dirección de la dispersión. Para obtener los eigenvalores es necesario resolver el polinomio característico de **C**:

$$f(\lambda) = det[\mathbf{C} - \lambda I] \tag{E.26}$$

donde *det* es el determinante de una matriz, e I es una matriz identidad. Desarrollando el determinante y teniendo en cuenta que $C_{x,y} = C_{yx}$ se obtiene:

$$f(\lambda) = \lambda^2 - (C_{xx} + C_{yy})\lambda + C_{xx}C_{yy} - C_{xy}^2 \tag{E.27}$$

Haciendo $f(\lambda) = 0$ y resolviendo para λ se obtiene:

$$\lambda_1 = \frac{C_{xx} + C_{yy} + \sqrt{(C_{xx} - C_{yy})^2 + 4C_{xy}^2}}{2} \tag{E.28}$$

$$\lambda_2 = \frac{C_{xx} + C_{yy} - \sqrt{(C_{xx} - C_{yy})^2 + 4C_{xy}^2}}{2} \tag{E.29}$$

Para determinar los vectores propios (eigenvectores) asociados a cada valor propio (eigenvalor) λ_1 y λ_2, sólo se tienen que hallar las soluciones no triviales del sistema homogéneo de ecuaciones lineales $(C - \lambda_i I)X = 0$, para $i = 1, 2$, donde I es una matriz identidad de dos por dos. Sin embargo, nos interesa almenos una solución no trivial para cada eigenvalor. Es decir, dos eigenvectores X_1 y X_2 que pueden ser calculados como:

$$\begin{aligned} X_i &= \begin{pmatrix} 1 \\ \frac{C_{xy}}{\lambda_i - C_{yy}} \end{pmatrix} \; con \;\; \lambda_i \neq C_{yy} \\ X_i &= \begin{pmatrix} \frac{C_{xy}}{\lambda_i - C_{xx}} \\ 1 \end{pmatrix} \; con \;\; \lambda_i \neq C_{xx} \\ para \;\; i &\in \{1, 2\} \end{aligned} \tag{E.30}$$

Los eigenvectores están alineados con los dos ejes principales del conjunto de puntos, y como comúnmente la mayor dispersión de los puntos se encuentra sobre la línea que describen dichos puntos, entonces el mejor ajuste de línea está en la dirección del eigenvector principal [Duda76] (ver Figura E.1). Por esta razón los eigenvectores pueden ser utilizados para ajustar líneas rectas a un conjunto de puntos.

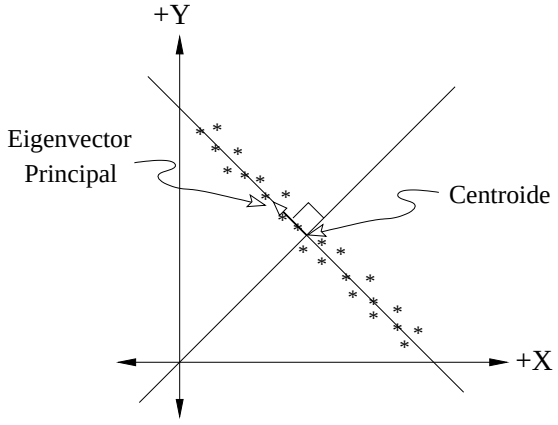


Figura E.1: Ajuste de una línea recta a un conjunto de puntos por el método de componentes principales (eigenvectores).

Sin embargo, como se mostró en la Sección 3.4 el método de componentes principales, a pesar de considerar los errores en forma lineal, se ve afectado por los datos atípicos. Para solucionar dicho problema se realiza una técnica de crecimiento para el ajuste de la línea. En un principio la línea está definida solo por tres puntos, en nuestro caso, el punto más cercano de l_{j+1} en l_j y los vecinos derecho e izquierdo de ese punto más cercano. En el ajuste pueden participar un número n de puntos, $\frac{n}{2}$ vecinos izquierdos y $\frac{n}{2}$ vecinos derechos.

La heurística aplicada en la técnica para el crecimiento de la línea es la siguiente: si la distancia de un punto vecino a la última línea ajustada es menor a cierto umbral, entonces, momentáneamente se considera adicionar el punto vecino en el ajuste, sino, se toma el siguiente vecino; si al adicionar un vecino cercano a la línea, el error del ajuste crece, entonces es descartado y se toma el siguiente vecino izquierdo o derecho, según sea el caso, hasta completar los $\frac{n}{2}$ vecinos en ambos lados.

Una vez que la línea es ajustada al conjunto de puntos vecinos, se calcula el punto sobre dicha línea que sea el más cercano al punto de l_{j+1}, para ello se utiliza la proyección ortogonal. A continuación se ve como calcular la proyección ortogonal tomando ventaja de los vectores propios.

Proyección ortogonal

Aprovechando la representación vectorial para la línea recta, es decir, el eigenvector principal que describe la línea, se utiliza el producto punto para calcular la proyección ortogonal de un punto cualquiera de coordenadas (x, y), a la línea descrita por el eigenvector, y así obtener las coordenadas (x_c, y_c) de la intersección entre la proyección y la línea (ver Figura E.2). El procedimiento a realizar es el siguiente: sean (v_x, v_y) las coordenadas unitarias del eingenvector principal, (m_x, m_y) el centroide del conjunto de puntos, se tiene que:

$$\begin{aligned} x_c &= p * v_x + m_x \\ y_c &= p * v_y + m_y \end{aligned} \tag{E.31}$$

donde p es el producto punto entre el vector del punto (x, y) al centro de masa (m_x, m_y) por el eigenvector principal, es decir:

$$p = (x - m_x, y - m_y) \cdot (v_x, v_y) = (x - m_x) * v_x + (y - m_y) * v_y \tag{E.32}$$

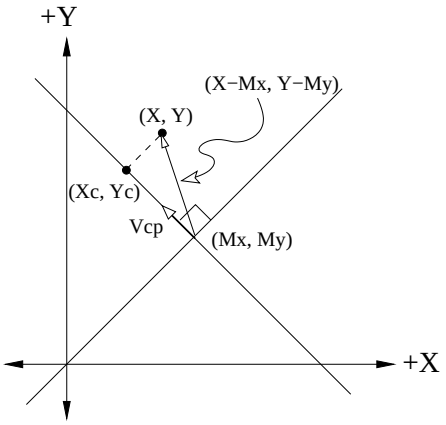


Figura E.2: Cálculo del punto (x_c, y_c) usando la proyección ortogonal de un punto cualquiera (x, y) sobre la línea descrita por el eigenvector principal v_{cp}.

Apéndice F

Cálculo de la localización local usando el estimador Lorentziano

Este apéndice contiene el cálculo correspondiente de la localización local usando el estimador Lorentziano, por medio de la minimización de la función del error en el mapeo de puntos (Ecuación F.1), para determinar la localización local mediante el alineamiento de dos rastreos consecutivos l_j y l_{j+1}, de puntos $[(x_{j,1}, y_{j,1}), ..., (x_{j,n}, y_{j,n})]$ y $[(x_{j+1,1}, y_{j+1,1}), ..., (x_{j+1,n}, y_{j+1,n})]$, para $n = 361$, respectivamente. La función de error a minimizar es:

$$E = \sum_{i=1}^{n} (\rho_\sigma(e_{x,i})) + \rho_\sigma(e_{y,i})) = \sum_{i=1}^{n} log\left(1 + \frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + log\left(1 + \frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \quad \text{(F.1)}$$

Donde $e_{x,i}$ son los errores de distancia en x, y $e_{y,i}$ los errores de distancia en y, es decir:

$$\begin{aligned} e_{x,i} &= \left(x'_{j+1,i} - x_{j,\psi(i)}\right) \\ e_{y,i} &= \left(y'_{j+1,i} - y_{j,\psi(i)}\right) \end{aligned} \quad \text{(F.2)}$$

El punto $(x'_{j+1,i}, y'_{j+1,i})$ representa el punto $(x_{j+1,i}, y_{j+1,i})$ después de la transformación rígida Φ, dada por una rotación θ y una traslación t_x y t_y:

$$\begin{bmatrix} x'_{j+1,i} \\ y'_{j+1,i} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{j+1,i} \\ y_{j+1,i} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad \text{(F.3)}$$

Y el punto $(x_{j,\psi(i)}, y_{j,\psi(i)})$ representa el punto más cercano en l_j asociado al punto i-ésimo de l'_{j+1}; siendo l'_{j+1} el rastreo l_{j+1} después aplicarle la transformación rígida Φ.

Así, la Ecuación F.1 está en función de tres parámetros: θ, t_x y t_y; es decir, $E = E(\theta, t_x, t_y)$. Por lo tanto, minimizar la Ecuación F.1 implica resolver un sistema de ecuaciones simultáneas para cada uno de los parámetros. El sistema de ecuaciones está dado por las condiciones que deben de cumplirse para que $E(\theta, t_x, t_y)$ sea un mínimo, estas ecuaciones son las correspondientes derivadas parciales igualadas a 0:

$$\begin{aligned} f_1(\theta, t_x, t_y) &= \frac{\partial E(\theta, t_x, t_y))}{\partial \theta} = 0 \\ f_2(\theta, t_x, t_y) &= \frac{\partial E(\theta, t_x, t_y))}{\partial t_x} = 0 \\ f_3(\theta, t_x, t_y) &= \frac{\partial E(\theta, t_x, t_y))}{\partial t_y} = 0 \end{aligned} \tag{F.4}$$

Para resolver el sistema de ecuaciones F.4, f_1, f_2 y f_3 son aproximadas mediante una linealización local [McCallum99], asumiendo que estas son diferenciables, se tiene que:

$$\begin{aligned} f_1(\theta + \delta\theta, t_x + \delta t_x, t_y + \delta t_y) &\approx f_1(\theta, t_x, t_y) + \frac{\partial f_1}{\partial \theta}\delta\theta + \frac{\partial f_1}{\partial t_x}\delta t_x + \frac{\partial f_1}{\partial t_y}\delta t_y \\ f_2(\theta + \delta\theta, t_x + \delta t_x, t_y + \delta t_y) &\approx f_2(\theta, t_x, t_y) + \frac{\partial f_2}{\partial \theta}\delta\theta + \frac{\partial f_2}{\partial t_x}\delta t_x + \frac{\partial f_2}{\partial t_y}\delta t_y \\ f_3(\theta + \delta\theta, t_x + \delta t_x, t_y + \delta t_y) &\approx f_3(\theta, t_x, t_y) + \frac{\partial f_3}{\partial \theta}\delta\theta + \frac{\partial f_3}{\partial t_x}\delta t_x + \frac{\partial f_3}{\partial t_y}\delta t_y \end{aligned} \tag{F.5}$$

Reagrupando y reescribiendo de forma matricial F.5 se obtiene un sistema de ecuaciones $A\ \delta\Phi = -B$, es decir:

$$\begin{bmatrix} \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial t_x} & \frac{\partial f_1}{\partial t_y} \\ \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial t_x} & \frac{\partial f_2}{\partial t_y} \\ \frac{\partial f_3}{\partial \theta} & \frac{\partial f_3}{\partial t_x} & \frac{\partial f_3}{\partial t_y} \end{bmatrix} \begin{bmatrix} \delta\theta \\ \delta t_x \\ \delta t_y \end{bmatrix} = - \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{F.6}$$

Donde $A_{3\times3}$ corresponde al Hessiano (∇^2), $B_{3\times1}$ corresponde al Jacobiano (∇), y $\delta\Phi_{3\times1}$ son los parámetros que estamos buscando.

De F.1 y F.4 se puede obtener una expresión para f_1:

$$\begin{aligned}
f_1 &= \frac{\partial E(\theta,t_x,t_y))}{\partial\theta} \\
&= \frac{\partial}{\partial\theta}\sum_{i=1}^{n} log\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + log\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2}\frac{\partial}{\partial\theta}\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + \frac{1}{1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2}\frac{\partial}{\partial\theta}\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2}\left(\frac{e_{x,i}}{\sigma}\right)\frac{1}{\sigma}\frac{\partial e_{x,i}}{\partial\theta} + \frac{1}{1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2}\left(\frac{e_{y,i}}{\sigma}\right)\frac{1}{\sigma}\frac{\partial e_{y,i}}{\partial\theta} \\
&= \sum_{i=1}^{n} \frac{e_{x,i}}{\sigma^2\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right)\frac{\partial e_{x,i}}{\partial\theta} + \frac{e_{y,i}}{\sigma^2\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right)\frac{\partial e_{y,i}}{\partial\theta} \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{2\sigma^2+e_{x,i}^2}\frac{\partial e_{x,i}}{\partial\theta} + \frac{2e_{y,i}}{2\sigma^2+e_{y,i}^2}\frac{\partial e_{y,i}}{\partial\theta}
\end{aligned} \tag{F.7}$$

Para f_2:

$$\begin{aligned}
f_2 &= \frac{\partial E(\theta,t_x,t_y))}{\partial t_x} \\
&= \frac{\partial}{\partial t_x}\sum_{i=1}^{n} log\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + log\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2}\frac{\partial}{\partial t_x}\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + \frac{1}{1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2}\frac{\partial}{\partial t_x}\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2}\left(\frac{e_{x,i}}{\sigma}\right)\frac{1}{\sigma}\frac{\partial e_{x,i}}{\partial t_x} + \frac{1}{1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2}\left(\frac{e_{y,i}}{\sigma}\right)\frac{1}{\sigma}\frac{\partial e_{y,i}}{\partial t_x} \\
&= \sum_{i=1}^{n} \frac{e_{x,i}}{\sigma^2\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right)\frac{\partial e_{x,i}}{\partial t_x} + \frac{e_{y,i}}{\sigma^2\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right)\frac{\partial e_{y,i}}{\partial t_x} \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{2\sigma^2+e_{x,i}^2}\frac{\partial e_{x,i}}{\partial t_x} + \frac{2e_{y,i}}{2\sigma^2+e_{y,i}^2}\frac{\partial e_{y,i}}{\partial t_x}
\end{aligned} \tag{F.8}$$

Y para f_3:

$$\begin{aligned}
f_3 &= \frac{\partial E(\theta,t_x,t_y))}{\partial t_y} \\
&= \frac{\partial}{\partial t_y}\sum_{i=1}^{n} log\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + log\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2}\frac{\partial}{\partial t_y}\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right) + \frac{1}{1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2}\frac{\partial}{\partial t_y}\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \frac{1}{1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2}\left(\frac{e_{x,i}}{\sigma}\right)\frac{1}{\sigma}\frac{\partial e_{x,i}}{\partial t_y} + \frac{1}{1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2}\left(\frac{e_{y,i}}{\sigma}\right)\frac{1}{\sigma}\frac{\partial e_{y,i}}{\partial t_y} \\
&= \sum_{i=1}^{n} \frac{e_{x,i}}{\sigma^2\left(1+\frac{1}{2}\left(\frac{e_{x,i}}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right)\frac{\partial e_{x,i}}{\partial t_y} + \frac{e_{y,i}}{\sigma^2\left(1+\frac{1}{2}\left(\frac{e_{y,i}}{\sigma}\right)^2\right)}\left(\frac{2}{2}\right)\frac{\partial e_{y,i}}{\partial t_y} \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{2\sigma^2+e_{x,i}^2}\frac{\partial e_{x,i}}{\partial t_y} + \frac{2e_{y,i}}{2\sigma^2+e_{y,i}^2}\frac{\partial e_{y,i}}{\partial t_y}
\end{aligned} \tag{F.9}$$

Donde las correspondientes derivadas para F.2, es decir para $e_{x,i}$ y $e_{y,i}$ son:

$$\begin{aligned}
\frac{\partial e_{x,i}}{\partial \theta} &= \frac{\partial}{\partial \theta}\left(x'_{j+1,i} - x_{j,\psi(i)}\right) = \frac{\partial x'_{j+1,i}}{\partial \theta} = \frac{\partial}{\partial \theta}\left(x_{j+1,i}cos(\theta) - y_{j+1,i}sin(\theta) + t_x\right) \\
&= -x_{j+1,i}sin(\theta) - y_{j+1,i}cos(\theta) \\
\frac{\partial e_{x,i}}{\partial t_x} &= \frac{\partial}{\partial t_x}\left(x'_{j+1,i} - x_{j,\psi(i)}\right) = \frac{\partial x'_{j+1,i}}{\partial t_x} = \frac{\partial}{\partial t_x}\left(x_{j+1,i}cos(\theta) - y_{j+1,i}sin(\theta) + t_x\right) \\
&= 1 \\
\frac{\partial e_{x,i}}{\partial t_y} &= \frac{\partial}{\partial t_y}\left(x'_{j+1,i} - x_{j,\psi(i)}\right) = \frac{\partial x'_{j+1,i}}{\partial t_y} = \frac{\partial}{\partial t_y}\left(x_{j+1,i}cos(\theta) - y_{j+1,i}sin(\theta) + t_x\right) \\
&= 0 \\
\frac{\partial e_{y,i}}{\partial \theta} &= \frac{\partial}{\partial \theta}\left(y'_{j+1,i} - y_{j,\psi(i)}\right) = \frac{\partial y'_{j+1,i}}{\partial \theta} = \frac{\partial}{\partial \theta}\left(x_{j+1,i}sin(\theta) + y_{j+1,i}cos(\theta) + t_y\right) \\
&= x_{j+1,i}cos(\theta) - y_{j+1,i}sin(\theta) \\
\frac{\partial e_{y,i}}{\partial t_x} &= \frac{\partial}{\partial t_x}\left(y'_{j+1,i} - y_{j,\psi(i)}\right) = \frac{\partial y'_{j+1,i}}{\partial t_x} = \frac{\partial}{\partial t_x}\left(x_{j+1,i}sin(\theta) + y_{j+1,i}cos(\theta) + t_y\right) \\
&= 0 \\
\frac{\partial e_{y,i}}{\partial t_y} &= \frac{\partial}{\partial t_y}\left(y'_{j+1,i} - y_{j,\psi(i)}\right) = \frac{\partial y'_{j+1,i}}{\partial t_y} = \frac{\partial}{\partial t_y}\left(x_{j+1,i}sin(\theta) + y_{j+1,i}cos(\theta) + t_y\right) \\
&= 1
\end{aligned} \tag{F.10}$$

Sea:

$$\begin{aligned}
D_{x,i} &= 2\sigma^2 + e_{x,i}^2 \\
D_{y,i} &= 2\sigma^2 + e_{y,i}^2
\end{aligned} \tag{F.11}$$

Finalmente los elementos de la matriz $B_{3\times 1}$ (∇) son:

$$\begin{aligned}
B = -\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} &= -\begin{bmatrix} \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial \theta} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial \theta} \\ \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_x} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_x} \\ \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_y} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_y} \end{bmatrix} \\
&= -2\begin{bmatrix} \sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial \theta} + \frac{e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial \theta} \\ \sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_x} + \frac{e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_x} \\ \sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_y} + \frac{e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_y} \end{bmatrix}
\end{aligned} \tag{F.12}$$

Los elementos de la matriz $A_{3\times 3}$ (∇^2) son:

$$
\begin{aligned}
a_{1,1} = \frac{\partial f_1}{\partial \theta} &= \frac{\partial}{\partial \theta} \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}} \frac{\partial x'_{j+1,i}}{\partial \theta} + \frac{2e_{y,i}}{D_{y,i}} \frac{\partial y'_{j+1,i}}{\partial \theta} \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}} \frac{\partial}{\partial \theta} \left(\frac{\partial x'_{j+1,i}}{\partial \theta} \right) + \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{\partial}{\partial \theta} \left(\frac{2e_{x,i}}{D_{x,i}} \right) \\
&\quad + \frac{2e_{y,i}}{D_{y,i}} \frac{\partial}{\partial \theta} \left(\frac{\partial y'_{j+1,i}}{\partial \theta} \right) + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{\partial}{\partial \theta} \left(\frac{2e_{y,i}}{D_{y,i}} \right) \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}} \frac{\partial^2 x'_{j+1,i}}{\partial \theta^2} + \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{D_{x,i} \frac{\partial}{\partial \theta}(2e_{x,i}) - 2e_{x,i} \frac{\partial}{\partial \theta}(D_{x,i})}{D_{x,i}^2} \\
&\quad + \frac{2e_{y,i}}{D_{y,i}} \frac{\partial^2 y'_{j+1,i}}{\partial \theta^2} + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{D_{y,i} \frac{\partial}{\partial \theta}(2e_{y,i}) - 2e_{y,i} \frac{\partial}{\partial \theta}(D_{y,i})}{D_{y,i}^2} \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}} \frac{\partial^2 x'_{j+1,i}}{\partial \theta^2} + \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{D_{x,i}(2) \frac{\partial x'_{j+1,i}}{\partial \theta} - 2e_{x,i} \frac{\partial}{\partial \theta}\left(2\sigma^2 + e_{x,i}^2\right)}{D_{x,i}^2} \\
&\quad + \frac{2e_{y,i}}{D_{y,i}} \frac{\partial^2 y'_{j+1,i}}{\partial \theta^2} + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{D_{y,i}(2) \frac{\partial y'_{j+1,i}}{\partial \theta} - 2e_{y,i} \frac{\partial}{\partial \theta}\left(2\sigma^2 + e_{y,i}^2\right)}{D_{y,i}^2} \\
&= 2 \sum_{i=1}^{n} \frac{e_{x,i}}{D_{x,i}} \frac{\partial^2 x'_{j+1,i}}{\partial \theta^2} + \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{D_{x,i} \frac{\partial x'_{j+1,i}}{\partial \theta} - 2e_{x,i}^2 \frac{\partial x'_{j+1,i}}{\partial \theta}}{D_{x,i}^2} \\
&\quad + \frac{e_{y,i}}{D_{y,i}} \frac{\partial^2 y'_{j+1,i}}{\partial \theta^2} + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{D_{y,i} \frac{\partial y'_{j+1,i}}{\partial \theta} - 2e_{y,i}^2 \frac{\partial y'_{j+1,i}}{\partial \theta}}{D_{y,i}^2}
\end{aligned}
\tag{F.13}
$$

$$
\begin{aligned}
a_{1,2} = \frac{\partial f_1}{\partial t_x} &= \frac{\partial}{\partial t_x} \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}} \frac{\partial x'_{j+1,i}}{\partial \theta} + \frac{2e_{y,i}}{D_{y,i}} \frac{\partial y'_{j+1,i}}{\partial \theta} \\
&= \sum_{i=1}^{n} \frac{2e_{x,i}}{D_{x,i}} \frac{\partial}{\partial t_x} \left(\frac{\partial x'_{j+1,i}}{\partial \theta} \right) + \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{\partial}{\partial t_x} \left(\frac{2e_{x,i}}{D_{x,i}} \right) \\
&\quad + \frac{2e_{y,i}}{D_{y,i}} \frac{\partial}{\partial t_x} \left(\frac{\partial y'_{j+1,i}}{\partial \theta} \right) + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{\partial}{\partial t_x} \left(\frac{2e_{y,i}}{D_{y,i}} \right) \\
&= \sum_{i=1}^{n} \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{D_{x,i} \frac{\partial}{\partial t_x}(2e_{x,i}) - 2e_{x,i} \frac{\partial}{\partial t_x}(D_{x,i})}{D_{x,i}^2} + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{D_{y,i} \frac{\partial}{\partial t_x}(2e_{y,i}) - 2e_{y,i} \frac{\partial}{\partial t_x}(D_{y,i})}{D_{y,i}^2} \\
&= \sum_{i=1}^{n} \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{D_{x,i}(2) \frac{\partial x'_{j+1,i}}{\partial t_x} - 2e_{x,i} \frac{\partial}{\partial t_x}\left(2\sigma^2 + e_{x,i}^2\right)}{D_{x,i}^2} \\
&\quad + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{D_{y,i}(2) \frac{\partial y'_{j+1,i}}{\partial t_x} - 2e_{y,i} \frac{\partial}{\partial t_x}\left(2\sigma^2 + e_{y,i}^2\right)}{D_{y,i}^2} \\
&= 2 \sum_{i=1}^{n} \frac{\partial x'_{j+1,i}}{\partial \theta} \frac{D_{x,i} \frac{\partial x'_{j+1,i}}{\partial t_x} - 2e_{x,i}^2 \frac{\partial x'_{j+1,i}}{\partial t_x}}{D_{x,i}^2} + \frac{\partial y'_{j+1,i}}{\partial \theta} \frac{D_{y,i} \frac{\partial y'_{j+1,i}}{\partial t_x} - 2e_{y,i}^2 \frac{\partial y'_{j+1,i}}{\partial t_x}}{D_{y,i}^2}
\end{aligned}
\tag{F.14}
$$

$$
\begin{aligned}
a_{1,3} = \frac{\partial f_1}{\partial t_y} &= \frac{\partial}{\partial t_y}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial\theta} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial\theta} \\
&= \sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial}{\partial t_y}\left(\frac{\partial x'_{j+1,i}}{\partial\theta}\right) + \frac{\partial x'_{j+1,i}}{\partial\theta}\frac{\partial}{\partial t_y}\left(\frac{2e_{x,i}}{D_{x,i}}\right) \\
&\quad + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial}{\partial t_y}\left(\frac{\partial y'_{j+1,i}}{\partial\theta}\right) + \frac{\partial y'_{j+1,i}}{\partial\theta}\frac{\partial}{\partial t_y}\left(\frac{2e_{y,i}}{D_{y,i}}\right) \\
&= \sum_{i=1}^{n}\frac{\partial x'_{j+1,i}}{\partial\theta}\frac{D_{x,i}\frac{\partial}{\partial t_y}(2e_{x,i}) - 2e_{x,i}\frac{\partial}{\partial t_y}(D_{x,i})}{D_{x,i}^2} + \frac{\partial y'_{j+1,i}}{\partial\theta}\frac{D_{y,i}\frac{\partial}{\partial t_y}(2e_{y,i}) - 2e_{y,i}\frac{\partial}{\partial t_y}(D_{y,i})}{D_{y,i}^2} \\
&= \sum_{i=1}^{n}\frac{\partial x'_{j+1,i}}{\partial\theta}\frac{D_{x,i}(2)\frac{\partial x'_{j+1,i}}{\partial t_y} - 2e_{x,i}\frac{\partial}{\partial t_y}\left(2\sigma^2 + e_{x,i}^2\right)}{D_{x,i}^2} \\
&\quad + \frac{\partial y'_{j+1,i}}{\partial\theta}\frac{D_{y,i}(2)\frac{\partial y'_{j+1,i}}{\partial t_y} - 2e_{y,i}\frac{\partial}{\partial t_y}\left(2\sigma^2 + e_{y,i}^2\right)}{D_{y,i}^2} \\
&= 2\sum_{i=1}^{n}\frac{\partial x'_{j+1,i}}{\partial\theta}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial t_y} - 2e_{x,i}^2\frac{\partial x'_{j+1,i}}{\partial t_y}}{D_{x,i}^2} + \frac{\partial y'_{j+1,i}}{\partial\theta}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial t_y} - 2e_{y,i}^2\frac{\partial y'_{j+1,i}}{\partial t_y}}{D_{y,i}^2}
\end{aligned}
\tag{F.15}
$$

$$
\begin{aligned}
a_{2,1} = \frac{\partial f_2}{\partial\theta} &= \frac{\partial}{\partial\theta}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_x} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_x} = \sum_{i=1}^{n}\frac{\partial}{\partial\theta}\frac{2e_{x,i}}{D_{x,i}} \\
&= 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial}{\partial\theta}(e_{x,i}) - e_{x,i}\frac{\partial}{\partial\theta}(D_{x,i})}{D_{x,i}^2} = 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial\theta} - e_{x,i}\frac{\partial}{\partial\theta}\left(2\sigma^2 + e_{x,i}^2\right)}{D_{x,i}^2} \\
&= 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial\theta} - 2e_{x,i}^2\frac{\partial x'_{j+1,i}}{\partial\theta}}{D_{x,i}^2}
\end{aligned}
\tag{F.16}
$$

$$
\begin{aligned}
a_{2,2} = \frac{\partial f_2}{\partial t_x} &= \frac{\partial}{\partial t_x}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_x} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_x} = \sum_{i=1}^{n}\frac{\partial}{\partial t_x}\frac{2e_{x,i}}{D_{x,i}} \\
&= 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial}{\partial t_x}(e_{x,i}) - e_{x,i}\frac{\partial}{\partial t_x}(D_{x,i})}{D_{x,i}^2} = 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial t_x} - e_{x,i}\frac{\partial}{\partial t_x}\left(2\sigma^2 + e_{x,i}^2\right)}{D_{x,i}^2} \\
&= 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial t_x} - 2e_{x,i}^2\frac{\partial x'_{j+1,i}}{\partial t_x}}{D_{x,i}^2}
\end{aligned}
\tag{F.17}
$$

$$
\begin{aligned}
a_{2,3} = \frac{\partial f_2}{\partial t_y} &= \frac{\partial}{\partial t_y}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_x} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_x} = \sum_{i=1}^{n}\frac{\partial}{\partial t_y}\frac{2e_{x,i}}{D_{x,i}} \\
&= 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial}{\partial t_y}(e_{x,i}) - e_{x,i}\frac{\partial}{\partial t_y}(D_{x,i})}{D_{x,i}^2} = 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial t_y} - e_{x,i}\frac{\partial}{\partial t_y}\left(2\sigma^2 + e_{x,i}^2\right)}{D_{x,i}^2} \\
&= 2\sum_{i=1}^{n}\frac{D_{x,i}\frac{\partial x'_{j+1,i}}{\partial t_y} - 2e_{x,i}^2\frac{\partial x'_{j+1,i}}{\partial t_y}}{D_{x,i}^2}
\end{aligned}
\tag{F.18}
$$

$$
\begin{aligned}
a_{3,1} = \frac{\partial f_3}{\partial\theta} &= \frac{\partial}{\partial\theta}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_y} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_y} = \sum_{i=1}^{n}\frac{\partial}{\partial\theta}\frac{2e_{y,i}}{D_{y,i}} \\
&= 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial}{\partial\theta}(e_{y,i}) - e_{y,i}\frac{\partial}{\partial\theta}(D_{y,i})}{D_{y,i}^2} = 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial\theta} - e_{y,i}\frac{\partial}{\partial\theta}\left(2\sigma^2 + e_{y,i}^2\right)}{D_{y,i}^2} \\
&= 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial\theta} - 2e_{y,i}^2\frac{\partial y'_{j+1,i}}{\partial\theta}}{D_{y,i}^2}
\end{aligned}
\tag{F.19}
$$

$$\begin{aligned} a_{3,2} = \frac{\partial f_3}{\partial t_x} &= \frac{\partial}{\partial t_x}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_y} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_y} = \sum_{i=1}^{n}\frac{\partial}{\partial t_x}\frac{2e_{y,i}}{D_{y,i}} \\ &= 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial}{\partial t_x}(e_{y,i}) - e_{y,i}\frac{\partial}{\partial t_x}(D_{y,i})}{D_{y,i}^2} = 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial t_x} - e_{y,i}\frac{\partial}{\partial t_x}\left(2\sigma^2 + e_{y,i}^2\right)}{D_{y,i}^2} \\ &= 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial t_x} - 2e_{y,i}^2\frac{\partial y'_{j+1,i}}{\partial t_x}}{D_{y,i}^2} \end{aligned} \tag{F.20}$$

$$\begin{aligned} a_{3,3} = \frac{\partial f_3}{\partial t_y} &= \frac{\partial}{\partial t_y}\sum_{i=1}^{n}\frac{2e_{x,i}}{D_{x,i}}\frac{\partial x'_{j+1,i}}{\partial t_y} + \frac{2e_{y,i}}{D_{y,i}}\frac{\partial y'_{j+1,i}}{\partial t_y} = \sum_{i=1}^{n}\frac{\partial}{\partial t_y}\frac{2e_{y,i}}{D_{y,i}} \\ &= 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial}{\partial t_y}(e_{y,i}) - e_{y,i}\frac{\partial}{\partial t_y}(D_{y,i})}{D_{y,i}^2} = 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial t_y} - e_{y,i}\frac{\partial}{\partial t_y}\left(2\sigma^2 + e_{y,i}^2\right)}{D_{y,i}^2} \\ &= 2\sum_{i=1}^{n}\frac{D_{y,i}\frac{\partial y'_{j+1,i}}{\partial t_y} - 2e_{y,i}^2\frac{\partial y'_{j+1,i}}{\partial t_y}}{D_{y,i}^2} \end{aligned} \tag{F.21}$$

Donde:

$$\begin{aligned} \frac{\partial^2 x'_{j+1,i}}{\partial\theta^2} &= \frac{\partial}{\partial\theta}\left(\frac{\partial x'_{j+1,i}}{\partial\theta}\right) = \frac{\partial}{\partial\theta}\left(-x_{j+1,i}sin(\theta) - y_{j+1,i}cos(\theta)\right) \\ &= \left(-x_{j+1,i}cos(\theta) + y_{j+1,i}sin(\theta)\right) \\ \frac{\partial^2 y'_{j+1,i}}{\partial\theta^2} &= \frac{\partial}{\partial\theta}\left(\frac{\partial y'_{j+1,i}}{\partial\theta}\right) = \frac{\partial}{\partial\theta}\left(x_{j+1,i}cos(\theta) - y_{j+1,i}sin(\theta)\right) \\ &= \left(-x_{j+1,i}sin(\theta) - y_{j+1,i}cos(\theta)\right) \end{aligned} \tag{F.22}$$

Así el sistema de ecuaciones está dado por $A_{3\times3}$, $B_{3\times1}$ y $\delta\Phi_{3\times1}$ de elementos:

$$2\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}\begin{bmatrix} \delta\theta \\ \delta t_x \\ \delta t_y \end{bmatrix} = -2\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{F.23}$$

Debido a que el factor 2 afecta a todos los elementos de $A_{3\times3}$, y del otro lado de la igualdad a todos los elementos de $B_{3\times1}$, se omite.

Para encontrar los parámetros de la transformación rígida Φ: θ, t_x y t_y; que han de alinear dos rastreos consecutivos, se aplica el método de optimización no lineal Newton-Levenberg-Marquardt (ver Sección 3.4.2).

Referencias

[3DV04] 3DV Systems, Israel Building 7, Yokneam Industrial Park. *3DV's Technology*, 2004.
URL `http://www.dvsystems.com/technology/technology.html`

[Arts04] Arts, T. Technological Arts, 2004.
URL `http://technologicalarts.com/. Updated: 2004/09/22`

[Barrientos97] Barrientos, A., Balaguer, C., Penin, L. F., y Arcil, R. *Fundamentos de Robótica.* McGraw Hill, 1997.

[Biber04] Biber, P., Andreasson, H., Duckett, T., y Schilling, A. 3D modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera. *En IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004).* September 28 - October 2 2004.

[Black96] Black, M. J. y Rangarajan, A. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19:57–92, jul. 1996.

[Boehler03] Boehler, W., Vincent, M. B., y Margs, A. Investigating laser scanner accuracy. *XIX CIPA Symposium at Antalya Turkey*, October 2003.

[Borenstein96] Borenstein, J., Everett, B., y Feng, L. *Navigating Mobile Robots: Systems and Techniques.* A.K. Peter, Ltd., Wellesley, MA, 1996.

[Borja01] Borja, C. A. *Sistema de Sonares para la Localización y Caracterización de Superficies 2D Usando Tiempo de Vuelo y Triangulación.* Proyecto

Fin de Carrera, Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey, Programa de Graduados en Computación, Información y Comunicaciones, 2001.

[Carrez03] Carrez, S. GNU for 68HC11/68HC12, 2003.
URL `http://stephane.carrez.free.fr/. Updated 09/23/2003`

[Collins04] Collins, B., Bombe, A., Mass, D., Dennedy, D., Hagserg, K., y Kinneberg, S. IEEE 1394 for linux, 2004.
URL `http://www.linux1394.org/. Updated 2004/02/29`

[Corrêa02] Corrêa, W. T., Oliveira, M. M., Silva, C. T., y Wang, J. Modeling and rendering of real environments. Inf. téc., Department of Computer Science, Princeton University, 2002.

[Dias03] Dias, P., Sequeira, V., Vaz, F., y Goncalves, J. G. M. Registration and fusion of intensity and range data for 3D modelling of real world scenes. IEEE, October 2003.

[Douxchamps04] Douxchamps, D. The IEEE1394 digital camera list, 2004.
URL `http://www.tele.ucl.ac.be/PEOPLE/DOUXCHAMPS/ieee1394/cameras/`

[Duda76] Duda, R. O. y Hart, P. E. *Pattern Classification and Scene Analysis.* Wiley-Interscience, 1976.

[Dudek00] Dudek, G. y Jenkin, M. *Computational Principles of Mobile Robotics.* Cambridge University Press, 2000.

[Fangi01] Fangi, G., Gagliardini, G., y Malinverni, E. S. Fast and accurate close range 3D modelling by laser scanning system. 2001. University of Ancona Italy.

[Forge04] Forge, S. Linux PCMCIA information page, 2004.
URL `http://pcmcia-sc.sourceforge.net/`

[Geuzaine04] Geuzaine, C. GL2PS, an OpenGL to PostScript Printing Library, 2004.
URL `http://www.geuz.org/gl2ps/`

[Gonzalez87] Gonzalez, R. C. y Wintz, P. *Digital Image Processing.* Addison-Wesley, 2ª ed$^{\underline{\text{ón}}}$., 1987.

[Gutmann96] Gutmann, J. y Schlegel, C. Amos: Comparison of scan matching approaches for self-localization in indoor environments. *En Proc. of the Ist Euromicro Workshop on Advance Mobile Robots.* IEEE Computer Society Press, 1996.

[Hähnel03] Hähnel, D., Burgard, W., y Thrun, S. Learning compact 3D models of indoor and outdoor environments with a mobile robot. *En Robotics and Autonomous Systems 44 (2003).* ELSEVIER, 2003.

[Hampel86] Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., y Stahel, W. A. *Robust Statistics: The Approach Based on Influence Functions.* John Wiley and Sons, New York, N.Y, 1986.

[Horstmann02] Horstmann, M., Winter, A., Födisch, M., y Hermes, T. Virtual reconstruction of environments using 3Dmosaicing techniques. 2002. University of Bremen, Center for computing Technologies Image Processing Department.

[Inc.04a] Inc., I. Triac detector, 2004.
URL `http://www.isocom.com/Quicklist4/MOC3042.htm`

[Inc.04b] Inc., R. H. redhat, 2004.
URL `http://www.redhat.com/`

[Iocchi00] Iocchi, L., Konolige, K., y Bajrachayra, M. Visually realistic mapping of a planar environment with stereo. 2000.

[Jácome04] Jácome, C. y Sánchez, E. Sistema de control electrónico para silla de ruedas eléctrica, 2004.
URL `http://www.iec.uia.mx/proy/titulacion/pr04/proy05/index.html`

[Jensfelt01] Jensfelt, P. *Approaches to Mobile Robot Localization in Indoor Environments.* Tesis Doctoral, Royal Institute of Technology (KTH), De-

partment of Signals an Systems, SE-100 44 Stockholm, Sweden, 2001. URL `http://www.s3.kth.se/`

[Jones98] Jones, J. L., Flynn, A. M., y Seiger, B. A. *Mobile Robots.* Cambridge University Press, 1998.

[Kaess01] Kaess, M., Arkin, R. C., y Rossignac, J. Compact encoding of robot-generated 3D maps for efficient wireless transmission. 2001. College of Computing, Georgia Institute of Technology.

[Kilgard96] Kilgard, M. J. *The OpenGL Utility Toolkit (GLUT) Programming Interface.* Silicon Graphics, Inc., Nov 13 1996. API Version 3.

[Klein00] Klein, K. y Sequeira, V. View planning for the 3D modelling of real world scenes. *En Proceedings of the IEEE International Conference on Robots and Systems.* IEEE, 2000.

[Kroah-Hartman04] Kroah-Hartman, G. y Gowdy, S. J. Linux USB, 2004. URL `http://linux-usb.org/`

[Latecki04] Latecki, L. J., Lakaemper, R., Sun, X., y Wolter, D. Building polygonal maps from laser range data. *ECAI Int. Cognitive Robotics Workshop*, August 2004.

[Liu01] Liu, Y., Emery, R., Chakrabartl, O., Burgard, W., y Thrun, S. Using EM to learn 3D models with mobile robots. *En Proceedings of the International Conference on Machine Learning (ICML).* 2001.

[Lu97] Lu, F. y Milios, E. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1997.

[McCallum99] McCallum, W. G., Gleason, A. M., y Hughes-Hallett, D. *Cálculo de Varias Variables.* CECSA, 1999.

[McComb00] McComb, G. *The Robot Bulder's Bonanza.* Mc Graw Hill, 2ª ed$^{\underline{\text{ón}}}$., 2000.

[Miles04] Miles, A. *Automatic 3D registration using range and colour data.* Proyecto Fin de Carrera, Carleton University, School of Computer Science, OttawaCarleton Institure for Computer Science, May 2004.

[Mot02] Motorola Inc. *MC9S12DP256B Device User V02.11*, 2002. URL `http://e-www.motorola.com/file/soft_dev_tools/data_sheet/9S12DP256B-ZIP.zip`

[Musliner95] Musliner, D. J. y Larsen, R. A. The SMARTV project: Smart model autonomous real-time veicles. Januray 1995. The University of Mayland, Institure for Advanced Computer Studies. URL `http://www.cs.umd.edu/users/musliner/sonar/tr.ps`

[Nüchter02] Nüchter, A., Surmann, H., y Hertzberg, J. Planning robot motion for 3D digitalization of indoor environments. 2002. Fraunhofer Institute for Autonomous Intelligent Systems (AIS).

[onLaptops04] on Laptops, L. Linux on laptops, 2004. URL `http://www.linux-on-laptops.com/. Updated 10/13/2004`

[Pervölz04] Pervölz, K., Nüchter, A., Surmann, H., y Hertzberg, J. Automatic reconstruction of colored 3D models. 2004. Fraunhofer Institute for Autonomous Intelligent Systems (AIS).

[Porikli98] Porikli, F. M. Image simplification by robust estimator based reconstruction filter. 1998. Mitsubishi Electric Research Labs.

[Press86] Press, W., Flannery, B., Teukolsky, S., y Vetterling, W. *Numerical recipes, the art of scientific computing.* Cambridge University Press, 1986.

[Proxim04] Proxim. Proxim wireless networks, 2004. URL `http://www.proxim.com/`

[Richard00] Richard, S., Wright, J., y Sweet, M. *OpenGL Super Bible.* Waite Group Press, 2ª ed$^{\underline{\text{ón}}}$., 2000.

[Román94] Román, J. B. *Algebra Lineal.* McGraw Hill, 1994.

[Romero01] Romero, L. *Construcción de mapas y localización de robots móviles: un enfoque probabilísta.* Tesis Doctoral, Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Cuernavaca, nov. 2001.

[Romero04a] Romero, L. Protocolo de comunicación serial con el robot, 2004. Universidad Michoacana de San Nicolás de Hidalgo. Facultad de Ingeniería Eléctrica. División de Estudios de Posgrado.

[Romero04b] Romero, L. y Arellano, J. J. Building an open platform for mobile robotics. *En International Symposium on Robotics and Automation (ISRA 2004).* August 25-27 2004.

[Rousseeuw03] Rousseeuw, P. J. y Leroy, A. M. *Robust Regression and Outlier Detection.* John Wiley and Sons, 2003.

[Rusinkiewicz01] Rusinkiewicz, S. Sensing for graphics, 2001.
URL `http://www.cs.princeton.edu/courses/archive/fall01/cs597d`

[Scheibe04] Scheibe, K., Scheele, M., y Klette, R. Data fusion and visualization of panoramic images and laser scans. 2004.

[Semiconductors99] Semiconductors, N. LMD18200 3A, 55V H-Bridge, 1999.
URL `http://www.national.com/LM/LMD18200.pdf/`

[Semple98] Semple, J. G. y Kneebone, G. T. *Algebraic Projective Geometry.* Oxford Classic Texts in the Physical Sciences, 1998.

[Sequeira95] Sequeira, V., Goncalves, J. G. M., y Ribeiro, M. I. 3D scene modelling from multiple range views. 1995. European Commission, Joint Research Centre.

[Shreiner00] Shreiner, D., Woo, M., Neide, J., y Davis, T. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.1.* Addison-Wesley, 2ª ed$^{\underline{\text{ón}}}$., 2000.

[SICa] SICK, INC., 6900 West 110th street, Bloomington mn 55438 USA. *LMS/LMI 400. Definition of telegram between the user interface and LMS ol LMI system via RS 422/RS 232.*
URL `http://www.sickoptic.com`

[SICb] SICK, INC., 6900 West 110th street, Bloomington mn 55438 USA. *Proximity Laser Scanner PLS, Technical Description.*
URL `http://www.sickoptic.com`

[Stallman01] Stallman, R. The GNU project, 2001.
URL `http://www.gnu.org/. Updated: 2004/05/26`

[Surmann01] Surmann, H., Lingemann, K., Nüchter, A., y Hertzberg, J. Fast acquiring and analysis of three dimensional laser range data. November 2001. Institute for Autonomous Intelligent Systems.

[Surmann03] Surmann, H., Nüchter, A., y Hertzberg, J. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, November 2003. Institute for Autonomous Intelligent Systems.

[Thrun00] Thrun, S., Burgard, W., y Fox, D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. *En Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, San Francisco, CA, 2000.

[Thrun02] Thrun, S. Robotic mapping: A survey. *En* G. Lakemeyer y B.Ñebel, eds., *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaulmann, 2002.

[Thrun03] Thrun, S., Martin, C., Liu, Y., y Hähnel, H. D. A real-time expectation maximization algorithm for acquiring to multi-planar and of indoor environments with mobile robots. *IEEE Transactions on Robotics and Automation*, 2003.

[TM01a] TM, P. E. *GM9234S031 Loc-Cog DC Servo Motor*. PITTMAN, 343 Godshall Drive, Harleysville, PA 19348. http://www.pittmannet.com/, 2001.
URL `http://www.pennmotion.com/part_nom_database/pdf/GM9234S031.pdf`

[TM01b] TM, P. E. *GM9236S027 Loc-Cog DC Servo Motor.* PITTMAN, 343 Godshall Drive, Harleysville, PA 19348. http://www.pittmannet.com/, 2001.
URL `http://www.pennmotion.com/part_nom_database/pdf/GM9236S027.pdf`

[Tourrilhes99] Tourrilhes, J. Wireless LAN resources for Linux, 1999.
URL `http://atrey.karlin.mff.cuni.cz/vojtech/joystick/`

[Trucco] Trucco, E. y Verri, A. *Introductory Techiques for 3-D Computer Vision.* Prentice Hall.

[Uni03] Universidad Rey Juancarlos. Grupo de Sistemas y Comunicaciones. Grupo de Robótica, Los Madrazo 9, 28080 Madrid Espana. *Mapas*, 2003.
URL `http://gsyc.escet.urjc.es/docencia/asignaturas/robotica/transpas/mapas.pdf`

[Vieyra04] Vieyra, A.Ñ. *Fusión de lecturas de un telémetro láser con un sistema de visión estéreo trinocular para detectar obstáculos en tres dimensiones.* Proyecto Fin de Carrera, Universidad Michoacana de San Nicloás de Hidalgo, Facultad de Ingeniería Eléctrica, División de Estidios de Posgrado, Abril 2004.

[Vojtech99] Vojtech, P. The linux joystick driver, 1999.
URL `http://atrey.karlin.mff.cuni.cz/vojtech/joystick/`

[Walpole86] Walpole, R. E. y Myers, R. H. *Probabilidad y Estadística para Ingenieros.* Interamericana, 3ª ed$^{\underline{\text{ón}}}$., 1986.

[Williams02] Williams, M. Building the linux of the robot world. *PCWorld*, mar. 2002.

Glosario

Actuador Dispositivo empleado para mover o impulsar el robot o algunas de sus partes.

Ambientes estructurados Son aquellos donde se guarda cierta geometría como líneas y planos paralelos u ortogonales.

CCD Acrónimo del Inglés *Charge Coupled Device*. Se refiere a un dispositivo de captura de imágenes basado en una matriz de semiconductores.

Datos atípicos Medición errónea originada por la sensibilidad de un sensor a cierto fenómeno.

Enrrollamiento Del Inglés *Windings*. Se refiere al sentido de conexión de los vértices. Puede ser en sentido de las manecillas del reloj o en contra.

Estimador robusto Hace referencia a un estimador estadístico que es insensible a una suposición ideal para la cual el estimador es óptimo.

Fotogrametría Ciencia o técnica cuyo objetivo es el conocimiento de las dimensiones de objetos en el espacio a través de la medida en fotografías.

Frustum Es una sección de una pirámide vista desde la punta hasta la parte ancha.

Gamepad Dispositivo de entrada usado para interactuar con un videojuego ya sea de consola o de computadora. También es conocido como control de mandos o control manual. En nuestro caso se utiliza para controlar los movimientos del robot.

GLUT Acrónimo del Inglés *OpenGL Utility Toolkit*. Es un conjunto de herramientas para escribir programas en OpenGL independientes del sistema de ventanas.

GNLM Acrónimo del método de optimización no lineal Gauss-Newton-Levenberg-Marquardt.

GNU Acrónimo del Inglés *General Public License*. Se refiere a Hardware o Software de licencia pública.

Joystick Dispositivo de entrada utilizado en un juego de consola o de computadora. Literalmente es una palanca de juegos.

Locomoción diferencial Sistema de locomoción formado por dos ruedas diametralmente opuestas situadas en un eje perpendicular a la dirección del robot.

Mapa Representación interna del ambiente en el cual se encuentra el robot.

Modelo virtual Representación de escenas reales con objetos producidos por un sistema informático que da la sensación de su existencia real.

NLM Acrónimo del método de optimización no lineal Newton-Levenberg-Marquardt.

Oclusión Ocultar un objeto con otro que obstruye el campo de visión.

Pan-Tilt Sistema robótico de Giro-Inclinación.

Plataforma de adquisición Se refiere al tipo y configuración de Hardware utilizado para la adquisición de datos.

PWM Acrónimo del Inglés *Pulse Width Modulator*. Se refiere a la técnica de Modulación de Ancho de Pulso. Es utilizada para controlar dispositivos o para proveer un voltaje variable de corriente continua.

Rastreo Conjunto de 361 lecturas de rango reportadas por el telémetro láser.

Reflectividad de las superficies Capacidad de la superficie de cierto objeto para reflejar la luz emitida por el telémetro láser. Tal reflexión de la luz puede ser afectada por el tipo de material y la textura del objeto

Rendering Proceso complejo de creación de imágenes de forma similar a como las persivimos en el mundo real en la computadora.

RGB Acrónimo del Inglés *Red Gren Blue* (Rojo Verde Azul). Es un modelo de color en el cual es posible representar un color mediante la mezcla de tres colores primarios: rojo verde y azul.

Robot Dispositivo generalmente mecánico que desempeña tareas automáticamente ya sea de acuerdo a supervisión humana directa a través de un programa predefinido o siguiendo un conjunto de reglas generales.

Sensor Transductor que convierte algún fenómeno físico en señales eléctricas que el microcontrolador puede leer.

Sistemas estacionarios Sistemas de adquisición de datos que carecen de movimiento automático. Deben ser ajustados manualmente por un usuario.

SLAM Acrónimo del Inglés *Simultaneous Localization And Mapping*. Se refiere a la localización local y construcción de mapas 2D simultáneos.

TOF Acrónimo del Inglés *Time Of Flight*. Se refiere al principio de reflexión de la luz llamado: Tiempo de Vuelo.

Transformación rígida Transformación que preserva todas las distancias y por ello preserva el tamaño y la forma. Esta dada por ángulo de rotación y una traslación.

Volumen visible Es un sistema de referencia cartesiano utilizado para la visualización ortogonal del modelo 3D.

Índice