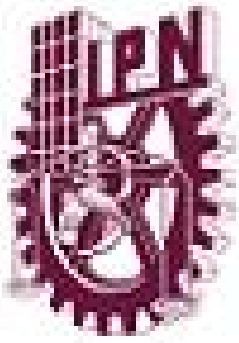


INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

"GENERACIÓN AUTOMÁTICA DE MAPAS DE ENTORNO USANDO UN ROBOT MÓVIL CON VISIÓN POR COMPUTADORA"

T E S I S

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

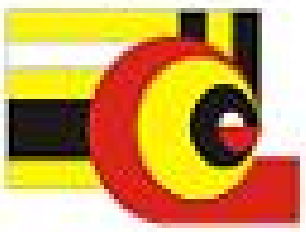
P R E S E N T A

ING. ÁNGEL GABRIEL ZANATTA JUÁREZ

DIRECTOR DE TESIS

DR. MARCO A. MORENO ARMENDÁRIZ

MÉXICO, D. F.
2010



Índice general

Índice de Figuras

Índice de tablas

Glosario

ASIC:	Circuitos integrados desarrollados para aplicaciones específicas. Son circuitos hechos a la medida para un uso en particular, en vez de ser concebidos para propósitos de uso general se usan para una función específica. Por ejemplo: un chip diseñado únicamente para ser usado en un teléfono móvil. Uno de los usos de los dispositivos FPGA's es modelar este tipo de circuitos.
CCD:	Por sus siglas en inglés *Charge Coupled Device*, al igual que CMOS es una tecnología que se utiliza para fabricar las celdas fotosensibles de una cámara. Su característica es que tiene una circuitería más simple, todos los pixeles comparten un convertidor análogo general.
CMOS:	Por sus siglas en inglés *Complementary Metal-Oxide* semiconductor es una tecnología que puede ser utilizada para fabricar las celdas fotosensibles que producen una imagen digital dentro de una cámara. Su principal característica es que la conversión de luz a una señal digital ocurre en cada pixel, aunque usualmente tiene menor calidad que la tecnología CCD.
FPGA:	Por sus siglas en inglés *Field Programmable Gate Array*, es una tecnología que permite que un chip sea configurado vía un lenguaje de descripción de hardware HDL por el usuario para crear un circuito con un comportamiento particular.
HDL:	Por sus siglas en inglés *Hardware Description Language, es* un lenguaje funcional con el que se describe la lógica y operación de un circuito. Los dos lenguajes más comunes de este tipo son: VHDL y Verilog.
NCC:	Es una medida estadística de similitud entre dos señales, por ejemplo en el caso de imágenes se usa para verificar si cierta ventana o parche dentro de una imagen es igual a otro, basándose en la suma de intensidades.
OpenCV	Librería de funciones para visión por computadora, bajo la licencia de código abierto BSD (*Berkeley Software Distribution*), es libre para uso académico y comercial.

SIFT:	Por sus siglas en inglés *Scale Invariant Feature Transform,* es un rasgo visual basado en diferencias de gaussianas, sobre una pirámide construida a partir de una imagen en diferentes escalas.
SLAM:	Por sus siglas en inglés *Simultaneous Localization And Mapping*, es una técnica relativamente reciente para resolver el problema de creación de mapas en línea. Por el tipo de sensores existen algunas variantes: • Visual SLAM: Pose un conjunto de sensores, y cuenta al menos con una cámara, puede recibir información sobre las señales de control del robot. • Mono SLAM: Utiliza exclusivamente una cámara como sensor, usualmente además es una cámara libre es decir que puede realizar movimientos en las tres dimensiones y los 3 ángulos de rotación.
SOPC:	Por sus siglas en ingles *System on a Programmable Chip*, se refiere a la idea del uso e interconexión de soft-hardware, componentes prediseñados pero configurables, para ser dispuestos en un diseño a conveniencia del diseñador.

Resumen

La comunidad científica de robots móviles tiene como uno de sus objetivos el de crear robots autónomos, si bien este objetivo implica resolver varios problemas: planeación de rutas, leyes de control, evasión de obstáculos, etc., el tema de interés para este trabajo es la creación de mapas de entorno, recientemente un tema de investigación muy activo. Un robot móvil que carece de información sobre su entorno, incluso de su propia posición inicial al arranque, pero que posee la habilidad de crear un mapa de su entorno mientras lo explora, podría luego definir y ejecutar trayectorias a cierta zona visitada previamente para poder realizar algún objetivo en particular sin intervención de un operador.

Este trabajo es una implementación en hardware de Visual SLAM, una técnica para la creación automática de mapas de entorno, por lo que se ejecutará sobre un dispositivo FPGA montado sobre un robot móvil dotado con una cámara digital.

En particular esta implementación de Visual SLAM, trabaja con imágenes del techo de una zona particular aislada con suelo plano y liso, una altura conocida y con marcas artificiales para facilitar su detección. La prueba consiste en hacer recorrer al robot móvil cierta trayectoria dentro de esta zona, el mapa construido deberá contener la posición de las marcas observables en su trayectoria así como su propia posición en varios periodos de tiempo, al final se comprueban con datos reales para verificar la exactitud del mapa obtenido.

Abstract

Scientific community of Mobile Robotics has among its objectives to create a truly autonomous robot. Such objective requires solving several types of problems like path planning, control laws, obstacle evasion and so on. The subject on this work is mapping, recently a very active field of scientific research. A mobile robot with no information about its environment not even its own initial position at start but still capable of creating a map while exploring, would provide the means to later create and execute paths to a certain visited place where the robot can accomplish its particular objective without an operator.

This project is a Visual SLAM hardware implementation, a technique for automatic map building. This implementation runs on a FPGA device board on top of a mobile robot equipped with a digital camera.

This particular implementation of Visual SLAM makes use of images of the ceiling of a particular isolated zone with a flat and smooth floor with known height and artificial references (placed in advance) to facilitate correct detection and tracking. As a test, the robot is made to follow a certain trajectory (manually operated) inside this controlled zone. At the end, the constructed map should contain the position of the artificial references that were visible from the path followed by the robot. Along the position of the references, the position of the robot itself should also be stored with at certain periodicity and reported. This data should be compared with the real position of the marks measured manually to compare the accuracy of the constructed map.

Capítulo 1

Introducción

1.1 Navegación autónoma

Iniciemos este trabajo con la discusión de ¿qué es un robot? Esta palabra fue utilizada en 1921 en la obra de teatro R.U.R. (Robots Universales Rossum) del escritor checo Karel Čapek, para referirse a trabajadores artificiales, el término se derivaba de la palabra checa: *robota* que significa trabajo forzado.

De acuerdo con la Asociación Japonesa Industrial de Robots (JIRA) existen 6 clases de robots de acuerdo a su modo de operación:

1. **Manejo manual**: Aparatos con varios grados de libertad, pero controlados manualmente por un operador.
2. **Secuencia fija**: Aparatos que ejecutan una secuencia predeterminada e inalterable de pasos para llevar a cabo su tarea. Alterar el flujo es muy complejo.

3. **Secuencia variable**: Aparatos que ejecutan una secuencia de pasos igual que el tipo anterior pero que el flujo es relativamente fácil de alterar.
4. **Almacena y reproduce**: Aparatos que son controlados inicialmente por operadores, pero capaces de almacenar la secuencia de pasos indicada para luego ejecutarla de modo automático.
5. **Control numérico**: Aparatos que son programados con una serie de movimientos en forma paramétrica en lugar de indicarle la tarea manualmente.
6. **Inteligente**: Aparatos con los medios para comprender el estado de su entorno, capaces de llevar a cabo su tarea aún en la presencia de cambios en las condiciones de trabajo.

El Instituto de Robótica de América (RIA) define a un robot como: un aparato o manipulador reprogramable, multifuncional designado para mover partes, herramientas u otros objetos especiales por medio de una serie de movimientos programados para la realización de sus diferentes tareas, por lo que únicamente las clases 4, 5 y 6 antes mencionadas son aceptadas como robots [1].

Si bien estas definiciones varían, queda claro que un robot es cierto aparato o entidad que puede ser programado para realizar cierta tarea específica en el mundo real y que para lograrlo requiere la habilidad de desplazarse en cierto entorno y ser capaz de interactuar con otros objetos existentes relevantes para su tarea, por lo que cierta interpretación del estado de su entorno es necesaria.

Si bien todo robot actúa en el mundo real y tiene partes móviles, debemos hacer la distinción entre robots móviles y fijos: Los robots móviles tienen actuadores especializados que les permiten desplazarse dentro de cierto medio y su movimiento es regido por su locomoción y limitado por la física del medio en el cual se desplaza; los robots fijos si bien pueden contar con varios grados de libertad para realizar tareas muy precisas, están sujetos a una base inmóvil, por lo que únicamente operan en un rango limitado de espacio. Actualmente la mayoría

de los robots son de este tipo: fijos, y operan como manipuladores con fines industriales en líneas de ensamble. También existen diversos ejemplos de robots móviles para exploración y servicio, aunque en mucho menor número. En estas dos áreas las condiciones de trabajo rara vez pueden ser controladas. Algunos ejemplos de robots especializados para tareas en diversos entornos son presentados en [2] donde se presentan detalles de la locomoción de varios tipos de robots: articulados, aéreos, acuáticos, etc., en este trabajo únicamente se hablará de robots de control diferencial.

La robótica es la ciencia de la percepción y manipulación del mundo real a través de dispositivos mecánicos controlados por computadoras, lamentablemente la mayor parte de la robótica aún se encuentra en su infancia. Pero su investigación es motivada en gran parte por el potencial que tendrían agentes autónomos para cambiar a la sociedad, por mencionar algunas posibilidades: podríamos delegar la responsabilidad del manejo de vehículos a robots para reducir la posibilidad de accidentes, utilizarlos en ambientes hostiles para el ser humano como la construcción, minería, el manejo de desechos tóxicos o simplemente como robots de servicio que se hagan cargo de las tarea tediosas del hogar como barrer, sacar la basura, pasear al perro, etc. [3].

De acuerdo con [1] la creación de un robot móvil autónomo es un campo de investigación fascinante por dos razones principalmente:

- Por el reto que implica el convertir una computadora con ruedas, que puede percibir sólo ciertas propiedades físicas de su entorno (por medio de sus sensores), en un agente inteligente capaz de identificar rasgos, patrones, a localizar, crear mapas y navegar. Tal reto requiere la participación simultánea de múltiples disciplinas para lograrlo, se requiere no una especialización sino la combinación. La ingeniería y las ciencias computacionales son sin duda los elementos esenciales, pero cuando las preguntas sobre comportamiento inteligente deben ser contestadas,

inteligencia artificial, ciencias cognitivas, filosofía y sicología ofrecen teorías y respuestas.

- Los robots móviles son una gran aproximación a agentes inteligentes, y un gran sueño del ser humano es el de crear aparatos que puedan imitar a seres vivos. Los conceptos de percepción y acción están fuertemente unidos en los seres vivos, al interactuar con el entorno los animales anticipan el resultado de sus acciones y predicen el comportamiento de otros objetos.

Un robot móvil que puede y debe desplazarse con cierta libertad en un entorno dado para realizar cierta tarea, tiene ahora que enfrentarse con el problema de navegación, que puede resumirse a tres preguntas de acuerdo con [5]: ¿en dónde estoy?, ¿a dónde debo ir?, ¿y cómo puedo llegar ahí?, es de aquí que surge la necesidad de técnicas para planeación de rutas, evasión de obstáculos y localización y mapeo, éste último el tema de esta tesis.

El problema de mapeo tiene probablemente más de tres décadas de ser un campo de investigación, al inicio se alimentaba al robot con un mapa prefabricado (el problema de localización), pero en realidad tal modelo no podría crear a un robot autónomo, el robot debería ser capaz de crear el mapa por sí mismo (el problema de mapeo).

La construcción de mapas implica determinar la posición de objetos de interés relativa a cierta referencia global como el plano cartesiano. Para llevar a cabo esta labor el robot debe hacer mediciones relativas a tales objetos o entidades de interés por lo que requiere conocer su propia posición, pero debido a la inexactitud de su movimiento antes debe deducirla a partir de la posición de los objetos de interés, requiriendo que ambos problemas sean atendidos concurrentemente [4].

Esta dependencia entre los problemas dio origen al término SLAM, por sus siglas en inglés: *Simultaneous Localization and Mapping*, (aunque inicialmente tuvo el nombre CML: *Concurrent Localization and Mapping*) [6]. El término fue utilizado

formalmente por primera vez en un artículo en 1995 en [5], desde entonces se han implementado diversas variantes de esta idea sobre todo tipo de robots: articulados [13], acuáticos [14], aéreos [15] e incluso sin robots [11] y utilizando distintos tipos de sensores o grupos de sensores. Inicialmente se utilizaron sensores de odometría o inerciales para mediciones relativas junto con sensores de distancia [12], después con sensores de visión, a esta combinación se le llama usualmente Visual SLAM [10]. En [9] puede encontrarse una detallada explicación de varios tipos de sensores e incluso de detalles de su uso para resolver el problema de localización y mapeo, en este trabajo únicamente se explicará el funcionamiento de una cámara, el único sensor que se utilizará para este trabajo.

La información que contiene una imagen sobre el estado del entorno es muy abundante y detallada pero no es sencillo extraerla, las técnicas habituales de procesamiento de imágenes implican sobreponer a cada pixel de la imagen una ventana de convolución, que es una matriz de tamaño variable pero usualmente cuadradas y de longitud impar (3x3, 5x5, etc.) Luego de hacer cierto cálculo con los datos sobrepuestos se almacena la suma o el valor ponderado en el pixel que fungió como centro. Tales algoritmos son computacionalmente costosos al ser ejecutados en un CPU de propósito general ya que por cada pixel de la imagen se recorren todos los pixeles de la ventana de convolución.

Una opción al problema de la alta complejidad computacional de las técnicas de procesamiento de imágenes digitales es utilizar circuitos con lógica especializada. FPGA por sus siglas en inglés: *Field Programmable Gate Array* es una tecnología que puede ofrecernos una solución: crear varias unidades pequeñas de cálculo que en conjunto puedan realizar múltiples operaciones de manera simultánea, utilizando un dispositivo de este tipo es posible definir un circuito de propósito específico y probar múltiples veces en hardware tal y como si el diseño hubiese sido construido físicamente. Tal solución puede ofrecer gran flexibilidad para realizar tareas donde se requiere tiempo real, sin embargo la complejidad y el tiempo de desarrollo suelen ser significativamente altos.

1.2 Motivación

La motivación de este trabajo es proveer los medios a un robot para ser una entidad autónoma, que pueda responder la pregunta ¿en dónde estoy?, y en futuro pueda responder las siguientes dos preguntas de la navegación: ¿a dónde debo ir?, ¿y cómo puedo llegar ahí?

El uso de una cámara como sensor permite obtener una representación muy completa del estado del entorno, en la mayoría de los casos, aunque su análisis es computacionalmente complejo, sobre todo para entidades que requieren de respuesta en tiempo real, por lo cual se propuso que el diseño obtenido se implementara sobre un dispositivo FPGA.

Este proyecto plantea el diseño e implementación de la infraestructura necesaria para la solución de una parte del problema de navegación autónoma. Esta base puede ser útil, al menos para inspirar, a otros investigadores a incursionar en el tema de localización y mapeo o en general de navegación autónoma.

1.3 Objetivos y alcances

Objetivo general:

Diseñar e implementar una solución al problema de localización y mapeo para un robot móvil usando una cámara como único sensor.

Objetivos particulares:

1) Construir un robot móvil que proporcione una trayectoria, para explorar el entorno.
    a) El robot es de control diferencial.
    b) La trayectoria que realiza es una línea recta, pero alterable a fin de realizar pruebas específicas.
    c) El robot cuenta con evasión básica de obstáculos.
2) Diseñar y evaluar en software técnicas necesarias para resolver diversos aspectos de localización y mapeo.

a) Detección y búsqueda activa de las marcas artificiales.
b) Cálculo de posición de las marcas y del robot.
c) Creación y actualización del mapa del entorno.

3) Implementación del diseño sobre el dispositivo FPGA.
   a) La lógica relacionada con la creación y actualización del mapa debe ser ejecutada sobre un dispositivo FPGA, pero se comunica con una PC para la presentación de resultados.
   b) El dispositivo FPGA controla dos módulos externos: una cámara CMOS, su único sensor y una pantalla gráfica LCD para corroborar la calidad de las imágenes.

Alcances:

- El sistema de localización y mapeo trabaja en el plano. Las referencias tienen dos componentes: (x, y). El robot tiene además de su posición, su orientación: (x, y, ϕ).
- Toma como referencias marcas artificiales de fácil detección. Las marcas, iguales entre sí, están distribuidas en el techo de la zona en donde el robot circula. La técnica de detección de estas marcas depende de la iluminación, por lo que la cámara debe ser reconfigurada según las condiciones existentes.

1.4 Metodología

El primer objetivo particular, la construcción de un robot móvil, pretende ofrecer pleno control del movimiento y la distribución física de los componentes necesarios para la fabricación de un mapa (tarjeta de desarrollo FPGA, cámara y baterías). Para ello se decidió diseñar y fabricar un robot que sirviera específicamente como base móvil de la tarjeta y cámara disponibles para este proyecto. Se puso especial interés en que la cámara pudiera cambiar su posición y orientación, que la velocidad lineal y radial fueran fáciles de modificar, y que pudiera albergar el banco de baterías necesarias para alimentar al robot.

El siguiente paso fue buscar literatura técnica sobre creación de mapas, esto con el fin de tener propuestas para cumplir el segundo objetivo particular referente a técnicas de localización y mapeo. Los artículos y tesis con métodos probabilísticos como SLAM son los más abundantes y simples por lo que fue ésta la técnica que se seleccionó.

Una vez que se seleccionó la técnica, se analizó la viabilidad de que fuese implementada en hardware para cumplir el tercer objetivo particular referente a la implementación sobre un dispositivo FPGA. Se concluyó que la implementación completa en hardware sería demasiado compleja por lo que se decidió incluir un CPU de propósito general dentro del diseño para que realizara los cálculos complejos requeridos para la creación y actualización del mapa. El hardware únicamente alimentaría a ese sistema con información obtenida por la cámara.

Para facilitar el trabajo de la parte responsable de obtener datos sobre las marcas se decidió utilizar marcas artificiales, esto ayuda a reducir el número de operaciones y tiempo necesario para encontrar su posición, información requerida para la construcción del mapa. Sin embargo como anexo a este documento se incluye la experiencia que se tuvo con algunas técnicas para la extracción de marcas naturales. Aún con este detalle, el objetivo general de hacer la implementación sobre un dispositivo FPGA de esta técnica de localización y mapeo se logró.

1.5 Esquema de la tesis

Capítulo 1: Se explican brevemente los retos de la navegación autónoma de robots para hacer notar la utilidad de la creación de mapas y la autolocalización en ellos para justificar los objetivos de esta tesis.

Capítulo 2: Una explicación de la técnica de SLAM y sus implicaciones como una técnica probabilística. Se mencionan algunas variantes del estado del arte que resuelven algunos de sus problemas.

Capítulo 3: Se ofrecen fundamentos necesarios para la comprensión del diseño propuesto, el capítulo está dividido en tres partes: el funcionamiento de una cámara digital, el único medio del robot para conocer el estado de su entorno; Técnicas de análisis digital de imágenes, la creación de mapas requiere de objetos que puedan ser detectados y referenciados, por lo que se mencionan algunas técnicas existentes para seleccionar y seguirlos en una secuencia de imágenes; Finalmente se presenta la tecnología FPGA, una breve historia, su funcionamiento y como se construye un diseño.

Capítulo 4: Una explicación a detalle de la solución propuesta, pasando por aspectos mecánicos, electrónicos, diseño HDL, y finalmente el método propuesto para la construcción del mapa.

Capítulo 5: Se explican a detalle los experimentos realizados, se presentan los resultados y las conclusiones. Finalmente se discuten posibles temas para trabajo futuro.

Capítulo 2

Estado del Arte

Como se explicó en el primer capítulo, este proyecto es una implementación de SLAM, una técnica para resolver el problema de autolocalización y mapeo. El presente capítulo hablará sobre algunas otras técnicas de mapeo existentes a fin de compararlas contra SLAM, se explican sus fundamentos, soluciones existentes y los problemas con los que se sigue enfrentando la comunidad científica.

2.1 Problemas de la navegación

Antes de iniciar con el verdadero tema de este capítulo, se presentan dos problemas comunes en navegación para comprender los beneficios de SLAM.

El primero es el concepto de navegación por estima, usada antiguamente para navegación marítima. Se asumía que se podía calcular la posición final si se hacia la suma de direcciones y distancias tomadas durante una travesía, si bien en principio es cierto que la suma de vectores debería darnos el punto final, el hecho era que no se podía determinar con exactitud cada uno de esos vectores, factores

como el viento, el oleaje irregular hacían que fuera poco preciso el cálculo. Este mismo método es el que usan los robots que dependen de odometría o sensores inerciales para estimar su posición y dirección, en general por exacto que sea el sensor tendrá cierto error y luego de varias estimaciones en donde el error se acumula dejará de tener sentido (fig. 2.1), por lo que se requiere de algo para comprobar el rumbo.

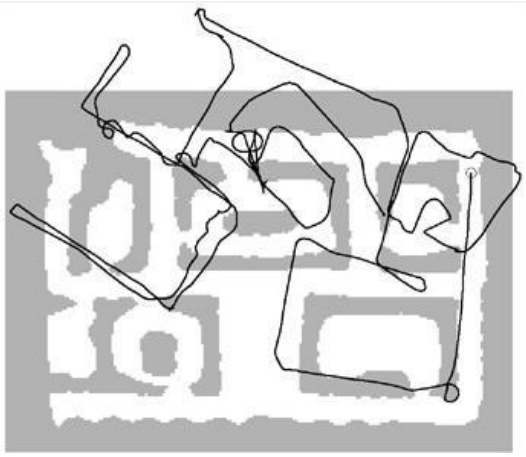


Fig. 2.1 Datos de un robot equipado con *encoders* en sus llantas

Lo cual nos introduce al segundo tema, el uso de referencias. Una referencia cuya posición se conoce de antemano, una montaña, el faro en una isla, etc. nos indicaría nuestra posición absoluta, de poder observarla. Pero no siempre se puede contar con una referencia global, sin embargo es posible tomar como referencia cualquier punto fijo y usarlo para determinar un cambio de rumbo y distancia relativa, si se deja de observar el punto inicial simplemente se toma uno nuevo, todos nosotros usamos este método para desplazarnos, de hecho cuando toda referencia es removida del campo de visión de una persona, ésta tiende a seguir una trayectoria errática [39].

2.2 Técnicas para la creación de mapas

Desde hace tiempo las técnicas para creación y uso de mapas ya sean modelos geométricos o topológicos han sido dominadas por métodos probabilísticos; Esto se debe a que la fuente es la misma, datos métricos de los

sensores y la respuesta de la comunidad científica al problema de la incertidumbre en las mediciones de los sensores es probabilidad [48].

¿Por qué probabilidad?, la respuesta es simple. No es posible asegurar todas las variantes posibles en el mundo real para una correcta interpretación de los datos entregados por los sensores. El modelo cinemático que se utiliza para describir el movimiento de un robot o vehículo y el modelo de observación para describir el entorno son simplificaciones.

Por ejemplo un sensor de distancia varía su medición dependiendo del voltaje, de las condiciones de luz, del tipo de superficie con la que interactúa, incluso es posible que entregue respuestas falsas por la física del mismo sensor, como en el caso de sonares. Si bien es posible agregar filtros, reguladores y demás aditamentos al sensor, el error jamás será cero. Ahora imaginemos que el sensor está montado sobre un robot que circula, de acuerdo con su modelo, sobre una superficie plana y por lo tanto el robot únicamente se mueve sobre los ejes X y Z, sin embargo, la realidad es que el robot circula por un superficie que puede tener pequeñas imperfecciones: la separación entre mosaicos, o una inclinación no uniforme; El propio movimiento del robot puede provocar cambios en Y, incluso una inclinación, ninguna considerada en el modelo por lo que la información del sensor es mal interpretada.

Algunas técnicas existentes para la creación de mapas son mencionadas a continuación:

- Basadas en el filtro de Kalman.
- Basadas en el algoritmo de expectativa maximizada.
- Cuadrícula de ocupación.

I. Basadas en el filtro de Kalman.

Si bien SLAM es técnicamente el problema y no la solución, desde sus inicios se ha asociado al uso del filtro de Kalman, o alguna variante, para lidiar con el ruido de los sensores al construir un mapa. Se hablará más a detalle de esta técnica en

el siguiente apartado, pero en esta sección se mencionarán algunas de sus características para hacer la comparación con otras técnicas existentes.

El filtro de Kalman es un filtro de Bayes que representa la probabilidad a posteriori con Gaussianas, distribuciones unimodales que pueden ser representadas compactamente por un pequeño número de parámetros. En el contexto de creación de mapas con robots se refiere el vector completo de estado x, que representa el mapa m y la posición del robot s.

$$x_t = (s_t, m)^T \tag{2.1}$$

Para un robot que navega sobre un plano la posición del robot s, es representada por tres variables, sus coordenadas cartesianas (s_x, s_y) y su dirección s_θ. Las marcas son un conjunto de coordenadas cartesianas, asumiendo que el mapa tiene k marcas: $\{ (m_x^1, m_y^1), (m_x^2, m_y^2), \dots (m_x^k, m_y^k) \}$ por lo tanto el vector x tiene una dimensión de $2k + 3$, El filtro de Kalman representan la probabilidad a posteriori con una media μ y una matriz de covarianza Σ.

$$x_t = \left(s_x, s_y, s_\theta, m_x^1, m_y^1, m_x^2, m_y^2, \dots, m_x^k, m_y^k\right)^T \tag{2.2}$$

La media μ siendo el punto de mayor probabilidad en que se encuentra el estado tiene la misma dimensión que el vector de estado $2k + 3$, sin embargo la matriz de covarianza Σ que denota la relación de cada componente en todos los demás tiene una dimensión $(2k + 3)^2$.

El filtro trabaja asumiendo que se cumplen ciertos requerimientos: El modelo cinemático es lineal y el ruido que lo acompaña tiene un distribución normal; El modelo de observación también debe tener esas características; La incertidumbre inicial debe tener una distribución normal. Para poder utilizar este tipo de filtro cuando el modelo cinemático es no lineal se utiliza una aproximación usando una serie de Taylor. El resultado es que puede escribirse la función de transición como una función lineal con ruido con distribución normal.

Una de las ventajas de este método es el hecho de que puede elaborar el mapa de forma incremental, mientras mantiene la incertidumbre de cada marca, característica útil para navegación; Algunos de los puntos negativos es que requiere de múltiples mediciones de la misma marca desde diferentes puntos a fin de eliminar el ruido; No ofrece asociación de datos, eso es responsabilidad del sistema que recupera las marcas; Las marcas son vistas como simples puntos, no tienen una forma geométrica detallada que los describa.

II. Expectativa maximizada.

Usualmente conocidos como EM por sus siglas, son algoritmos probabilísticos basados en la estimación de la máxima verosimilitud. De las técnicas actuales es la mejor solución a la asociación de datos [48]. Aún si todas las marcas son parecidas entre sí puede generar mapas consistentes, sin embargo no mantiene la incertidumbre del mapa, en lugar de ello elabora una evaluación sucesiva en el espacio de todos los mapas para intentar y encontrar el mapa más cercano a la realidad, es decir itera sobre todos los datos múltiples veces por lo que no pueden ser elaborados de forma incremental.

Los algoritmos basados en EM tienen dos pasos: el paso de predicción o simplemente E, donde la probabilidad a posteriori sobre la posición del robot es calculada para un mapa dado; el paso de maximización o M, en donde se calcula el mapa más adecuado dada cierta posición del robot, el resultado de estos procesos son una serie incremental de mapas: $m^0, m^1, .., m^n$ iniciando por m^0, un mapa vacío.

Formalmente la función que está siendo maximizada es la predicción sobre la probabilidad conjunta de los datos de las marcas d^t y la trayectoria del robot s^t: $\{s_1, .., s_n\}$

$$m^{i+1} = argmax_m E_{s^t}\left[\log\, p(d^t, s^t \mid m) \middle|\, m^i, d^t\right] \tag{2.3}$$

El mapa m^{i+1} es obtenido a partir del mapa m^i sin embargo la trayectoria del robot es desconocida por lo que se calcula la verosimilitud sobre todos las posibles trayectorias que el robot pudo haber tomado.

$$m^{i+1} = argmax_m \sum_T \int p(s_T \mid m^i,\ d^t) \log\ p(z_T \mid s_T, m) \partial s_T \tag{2.4}$$

La diferencia clave a la localización común es que en este caso los datos de todo el intervalo $\{1, .., t\}$ son usados para estimar la probabilidad a posteriori de la posición del robot. El hecho de que puede resolver el problema de asociación la hace una técnica útil, pero para eliminar la limitación de no poder operar de forma incremental han surgido algoritmos híbridos que muestran características de aquellos basados en el filtro de Kalman y de EM.

Fig. 2.2a Mapa generado a partir de los datos sin procesar de un sonar

Fig. 2.2b Mapa generado con EM sobre los datos de un sonar

III. Cuadrícula de ocupación.

Este tipo de algoritmo no resuelve SLAM (autolocalización y mapeo), pero es una técnica de mapeo dada la posición del robot recurrente en la literatura. El problema central que trata de resolver es el de generar un mapa consistente a partir de mediciones con ruido o incompletas. Aun conociendo la posición del robot no es siempre fácil saber si un punto en el entorno está ocupado o no.

Las cuadrículas de ocupación crean mapas probabilísticos en una cuadrícula, y pueden ser en 2 ó 3 dimensiones. Normalmente el algoritmo implementado es un filtro de Bayes pero que calcula la probabilidad a posteriori de la ocupación de cada celda del mapa. Sea *(x,y)* las coordenadas de una celda del mapa y $m_{x,y}$ su ocupación una variable binaria que indica si está o no ocupado.

$$\frac{p(m_{x,y}|z^t,\, s^t)}{1-p(m_{x,y}|z^t,\, s^t)} = \frac{p(m_{x,y}|z_t, s_t)}{1-p(m_{x,y}|z_t, s_t)} \frac{1-p(m_{x,y})}{p(m_{x,y})} \frac{p(m_{x,y}|z^{t-1},\, s^{t-1})}{1-p(m_{x,y}|z^{t-1},\, s^{t-1})} \tag{2.5}$$

Este algoritmo goza de reputación por ser robusto y fácil de implementar, pero una de sus grandes desventajas es que no tiene manera de soportar incertidumbre sobre la posición del robot, otra desventaja es que asume ruido independiente (por el filtro de Bayes) pero si el error es correlacionado puede provocar mapas incongruentes.

Fig. 2.3a Mapa generado utilizando un algoritmo basado en la cuadrícula de ocupación

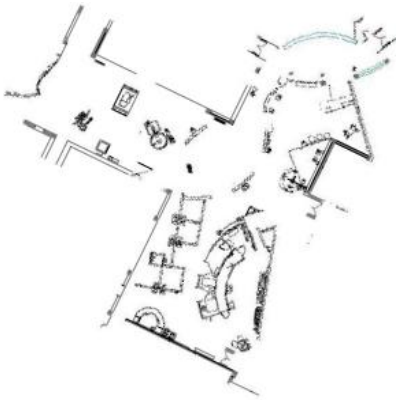
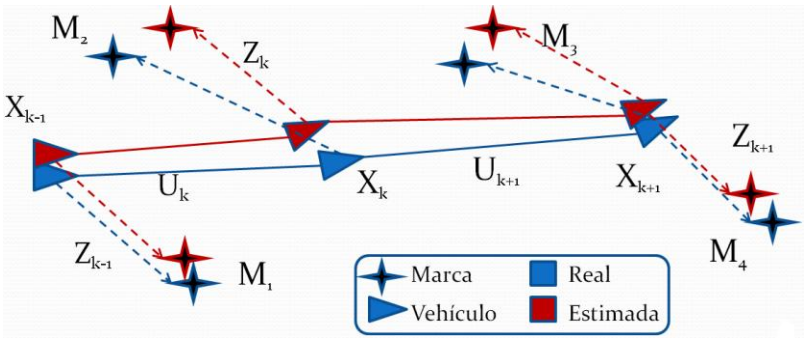


Fig. 2.3b Plano de la zona de pruebas

2.3 SLAM

SLAM es un proceso en que un robot móvil construye un mapa del entorno de forma incremental y al mismo tiempo deduce su posición. El mapa es constituido por referencias que tienen alguna característica interesante, y el robot no tiene conocimiento previo sobre la posición de éstas. Pongamos un ejemplo, imaginemos un robot recorriendo cierto entorno y observando marcas; Dividamos

este proceso en espacios discretos de tiempo *k*, en cada momento *k* existe un vector de estado x_k que describe la posición y orientación del robot; Un vector de control u_k aplicado en *k-1*; Un vector m_i que representa la posición de cada marca, las marcas son estáticas; Finalmente un vector de observación z_{ik} que es la medición del grupo de sensores disponibles del robot de la marca m_i en el tiempo *k* (fig. 2.4) [38]



Vector de estado
$X_{0:k} = \{x_o, x_1, \dots x_k\}$

Vector de control
$U_{0:k} = \{u_o, u_1, \dots u_k\}$

Marcas
$m = \{m_o, m_1, \dots m_n\}$

Vector de observaciones
$Z_{0:k} = \{z_o, z_1, \dots z_k\}$

Fig. 2.4 Problema fundamental que trata de resolver SLAM

SLAM es un método probabilístico que requiere que la distribución de la probabilidad (ec. 2.1) sea calculada para todo tiempo *k.* Esta función describe la probabilidad combinada a posteriori de la posición de lax marcas y del robot en el tiempo *k* dado un conjunto histórico del vector de control, del vector de mediciones y un estado inicial.

$$P(x_k, m | Z_{0:k}, U_{o:k}, x_0) \tag{2.6}$$

En general un método recursivo es deseable, iniciando con la estimación de la distribución en el tiempo *k-1*, la probabilidad es calculada usando el teorema de Bayes alimentado por el vector de control y de observación. Este cálculo requiere que un modelo de la transición de estado y otro de la observación sea definido.

El modelo de observación describe la probabilidad de hacer una medición z_k, cuando la posición del robot y de la marcas son conocidas (ec. 2.7).

$$P(z_k | x_k, m) \tag{2.7}$$

El modelo de transición de estado, que desde este punto llamaremos modelo cinemático, describe un proceso de Markov, en donde se plantea que el estado actual únicamente depende del inmediato anterior y el vector de control, independiente de las observaciones y el mapa.

$$P(x_k|x_{k-1}, u_k) \tag{2.8}$$

SLAM puede ahora plantearse como una forma recursiva de dos pasos secuenciales: predicción y actualización.

$$P(x_k, m|Z_{0:k}, U_{o:k}, x_0) = \int P(x_k|x_{k-1}, u_k) \cdot P(x_{k-1}, m|Z_{0:k-1}, U_{o:k-1}, x_0) \partial x_{k-1} \tag{2.9}$$

$$P(x_k, m|Z_{0:k}, U_{o:k}, x_0) = \frac{P(z_k|x_k, m) \cdot P(x_k, m|Z_{0:k-1}, U_{o:k}, x_0)}{P(z_k|z_{o:k-1}, U_{o:k})}$$

El problema de la construcción del mapa (conocer el estado de las marcas) puede ser descrito como una densidad condicional (ec. 2.10), en donde se asume que se conoce la posición del robot, al menos determinísticamente. El mapa *m* es entones construido al fusionar mediciones desde varios puntos.

$$P(m|X_{o:k}, Z_{0:k}, U_{0:k}) \tag{2.10}$$

El modelo de observación (ec. 2.6) hace explicita la dependencia de mediciones sobre las referencias y sobre el robot, por lo que no es posible realizar la siguiente partición:

$$P(z_k|x_k, m) \neq P(x_k|z_k)P(m|z_k) \tag{2.11}$$

Se sabe que tal partición crea estimaciones inconsistencias [42, 43]. Se puede asumir que el error entre la estimación y la posición real de las referencias es común, ya que la razón es la misma: un error en la estimación de la posición del robot, punto desde donde se toman las mediciones. Ambos datos están altamente correlacionados. La distancia relativa entre dos marcas cualesquiera m_i, m_j puede ser estimada con una alta precisión, aunque no se tiene certeza sobre la posición absoluta de ninguna de ellas. En SLAM la correlación entre la estimación de la posición de las marcas se incrementa monotónicamente mientras más mediciones sean realizadas. El proceso puede visualizarse como una red de cables que

conecta a todas las referencias como un gran bloque, conforme más mediciones son realizadas menores son los ajustes a esos cables.

2.3 Soluciones a SLAM

Solucionar el problema de SLAM implica encontrar una representación del modelo cinemático y de observación que permitan un cálculo eficiente y consistente de la distribución a priori y posteriori (ec. 2.9). Por mucho el más utilizado es EFK [12], *Extended Kalman Filter*, el filtro extendido de Kalman basado en el cálculo de jacobianos de los modelos cinemáticos y de observación; otra opción es FastSLAM [44] basado en filtrado de partículas que puede reducir la complejidad temporal de EKF. Aunque menos difundidos, también existen otras variantes del filtro del Kalman: UKF [45], *Unscented Kalman Filter* y recientemente MEKF [46], *Mean Extended Kalman Filter.* En general tratan de eliminar los problemas de congruencia que ocurren al linealizar los modelos no lineales típicos de un vehículo moviéndose en un plano o en el espacio.

2.4 Problemas por resolver de SLAM

La comunidad científica involucrada en investigación sobre SLAM trabaja en varias partes del proceso, por mencionar algunas:

- Buscar nuevos modelos para la representación del problema.
- Reducir la complejidad de alguno de los modelos existentes mediante un uso eficiente del mapa. Por ejemplo la división del mapa en varios submapas conectados [41] o actualización selectiva de marcas [47], usualmente estas técnicas enfrentan además el problema de cierre de circuitos, reconocer que los mapas se traslapan.
- Proponer el uso de un conjunto de rasgos para selección y búsqueda activa de puntos de interés, a este problema se le llama asociación de datos.

- Recuperación en caso de que el vehículo se “pierda”. Aún si el vehículo no es capaz de calcular el movimiento que lo llevó hasta su posición actual, tener la habilidad de reconocer que se trata de un punto revisitado [49], a este problema se le llama secuestro.
- Implementaciones en hardware [40].
- Modelar ambientes dinámicos donde las marcas no son estáticas. Por ejemplo modelar automóviles o peatones en una calle.
- Representación de las marcas con formas geométricas con área (en un plano) o volumen (en el espacio.)

Capítulo 3

Fundamentos

Como se mencionó en la introducción, este capítulo trata de dar al lector una breve explicación de temas que en este proyecto se utilizan como base para cumplir el objetivo de la tesis pero que en realidad son temas independientes: cámaras digitales, análisis digital de imágenes y dispositivos FPGA's.

3.1 Cámaras digitales

Una cámara digital CMOS es el único sensor con el que cuenta el robot móvil para percibir el estado del entorno, por lo que esta sección tiene como intención la de introducir algunos conceptos básicos, útiles para la interpretación de los datos que se obtienen de ella y con los que se trabaja para la creación del mapa.

3.1.1 Modelo de una cámara estenopeica

Iniciemos por el modelo de una cámara estenopeica, esquematizado en la fig. 3.1. Éste es un modelo de una cámara ideal que no tiene ciertas características de una cámara real, pero que nos ayudará como base para comprender el funcionamiento básico de una cámara y nos permitirá luego proponer un modelo más adecuado.

El modelo de una cámara estenopeica consiste en un plano que tiene un orificio infinitesimal llamado centro óptico (señalado como *O*) y que es el origen del sistema de coordenadas de 3 dimensiones de la cámara. En el cual *X* apunta hacia alguno de los lados de la cámara, *Y* apunta hacia arriba y *Z* indica la profundidad o distancia. El eje *Z* también es la dirección hacia donde apunta la cámara, por lo cual se le llama eje óptico o principal. Paralelo a este plano, detrás del origen y a una distancia focal *f*, se encuentra el plano de la imagen, que cuenta con su propio sistema de coordenadas, *V* que apunta hacia arriba, *U* que apunta hacia alguno de los lados y cuyo centro es llamado punto principal (señalado como *C)*.

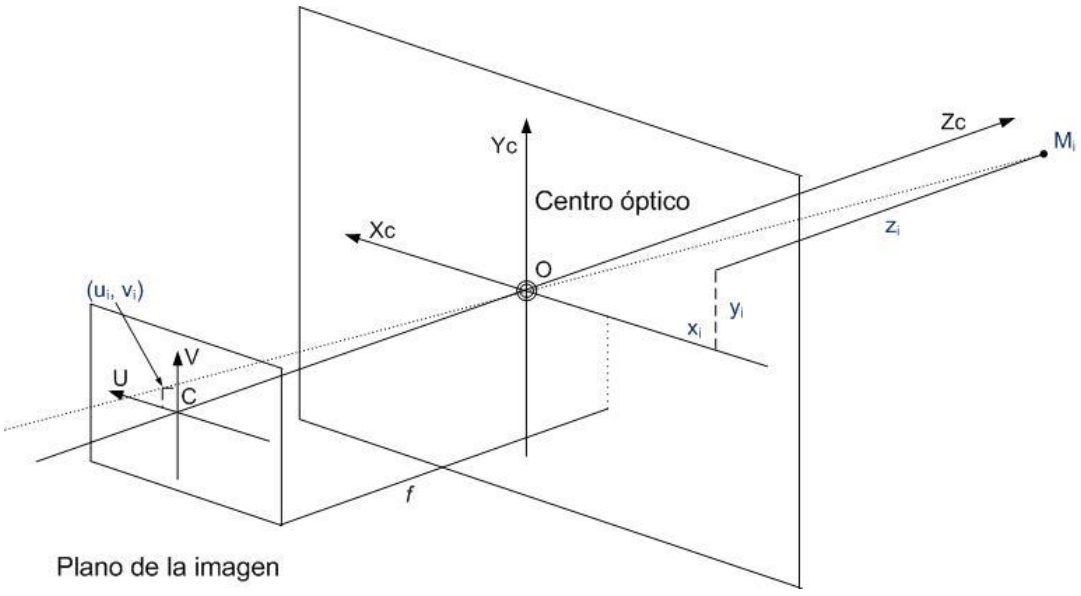


Fig. 3.1 Modelo de una cámara estenopeica.

La luz proveniente del mundo real pasa a través del centro óptico y es proyectada en el plano de la imagen, debe notarse que la imagen en ese plano está invertida. Un punto cualquiera M_i ubicado en el mundo real en las coordenadas (x_i, y_i, z_i) es proyectado en las coordenadas del plano de la imagen en el punto (u_i, v_i) bajo la siguiente ecuación:

$$\begin{pmatrix} -\frac{u_i}{f} \\ -\frac{-v_i}{f} \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{pmatrix} \Rightarrow \begin{pmatrix} -u_i \\ -v_i \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} f \\ \frac{y_i}{z_i} f \end{pmatrix} \tag{3.1}$$

Debe notarse que para un punto del mundo real M_i y una distancia focal f, es posible determinar un único punto (u_i, v_i) en coordenadas de la imagen. Sin embargo, para un punto en la imagen (u_i, v_i), no es posible determinar un único punto M_i. Dos puntos M_i y M_j que mantengan la misma relación entre sus coordenadas (ec. 3.2) se proyectarán en la misma coordenada de imagen (u_i, v_i).

$$\left(\frac{x_i}{z_i}\right) = \left(\frac{x_j}{z_j}\right) \wedge \left(\frac{y_i}{z_i}\right) = \left(\frac{y_j}{z_j}\right) \tag{3.2}$$

Modifiquemos este modelo añadiendo un plano virtual de la imagen delante del origen a la misma distancia f para obtener una proyección no invertida de la imagen, esquematizado en la fig. 3.2.

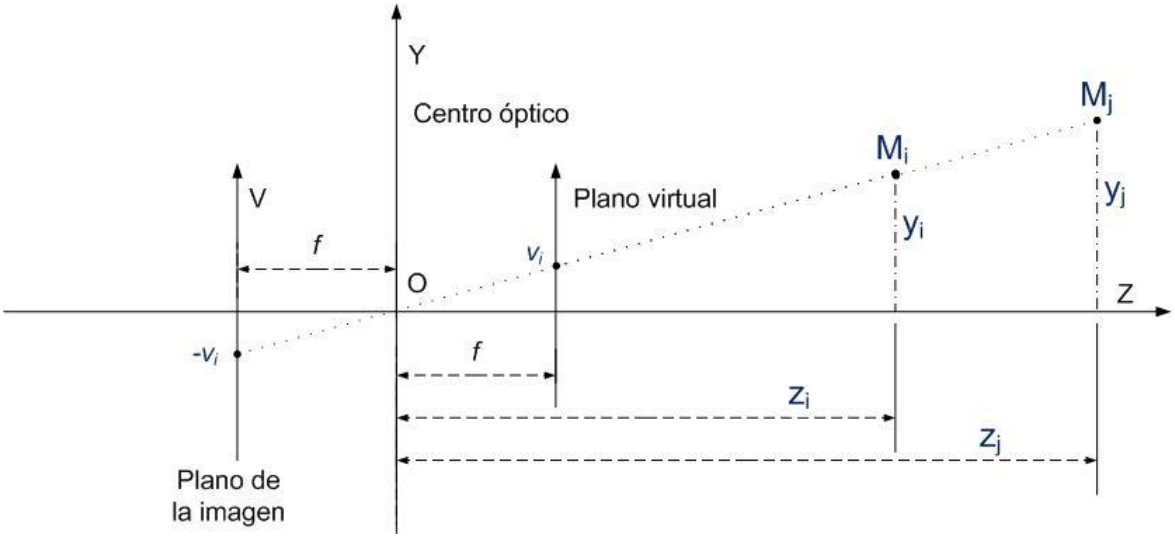


Fig. 3.2 Cámara estenopeica con un plano virtual

Esto nos permite colocar la relación de la ec. 3.1 en una forma más conveniente, descrita en la ecuación ec. 3.3:

$$\begin{pmatrix} \frac{u_i}{f} \\ \frac{v_i}{f} \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{pmatrix} \Rightarrow \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} f \\ \frac{y_i}{z_i} f \end{pmatrix} \tag{3.3}$$

Es importante mencionar algunas de las características que este modelo puede mostrar: Al incrementar la distancia entre un objeto y el centro óptico de la cámara, la proyección se vuelve más pequeña; líneas paralelas a un plano no paralelo al plano de la imagen no serán paralelas en la imagen proyectada, imaginemos que observamos una línea paralela al eje *Z*; para cada punto M_i de la línea, sus componentes x_i y y_i son constantes mientras que z_i varia, esto provoca que puntos más alejados se vean más cerca al centro de la imagen *C* (fig. 3.1) en una línea horizontal en la proyección, la proyección inducida en el plano de la imagen es llamada proyección perspectiva. La fig. 3.3 muestra un clásico ejemplo de este efecto con las vías férreas.

Fig. 3.3 Proyección perspectiva

3.1.2 Adaptación del modelo

La base del funcionamiento de una cámara digital real es en esencia igual al descrito por el modelo presentado, pero debemos añadir algunos detalles importes a nuestro modelo para poder modelarla adecuadamente.

- El plano de la imagen en las cámaras reales es cierto elemento fotosensible y el plano virtual no existe, pero se valen de algún mecanismo para hacer la inversión. En el caso de este proyecto por ejemplo, la memoria de la cámara CMOS es leída en forma inversa. Otra característica de este elemento fotosensible es que es finito, por lo que el ángulo visión es limitado como se muestra en la fig. 3.4 con los ángulos α_v y α_u descritos en la ec. 3.4.

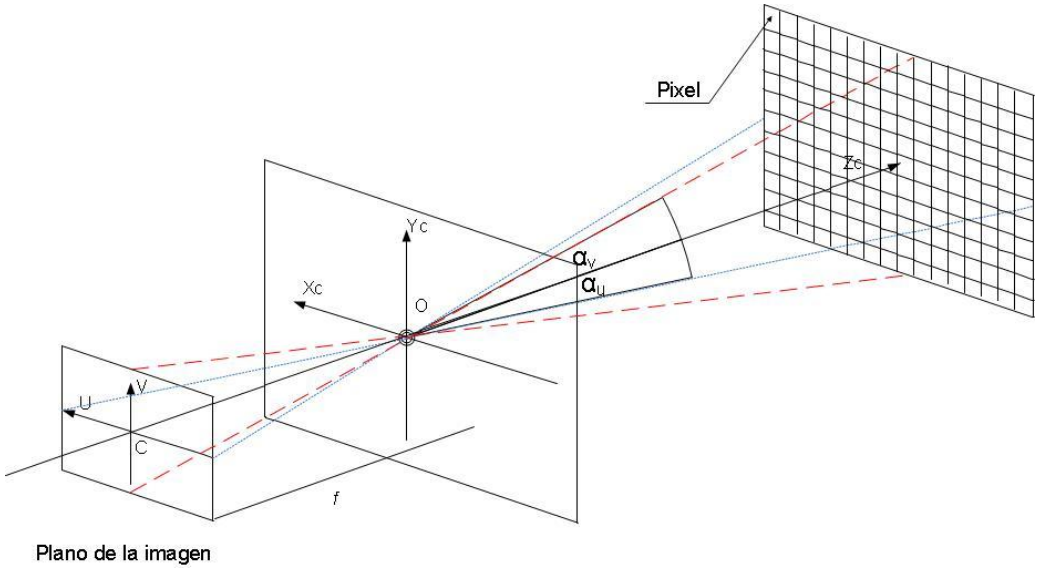


Fig. 3.4 Ángulo de visión de una cámara

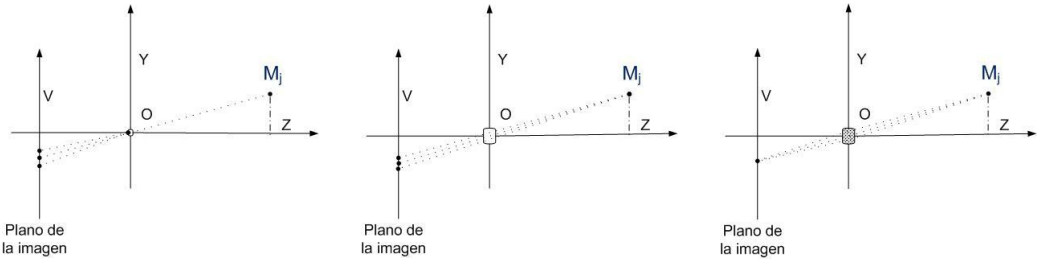
$$\left[\frac{\alpha_u}{\alpha_v}\right] = \left[\frac{2\left(\arctan\left(\frac{U}{2f}\right)\right)}{2\left(\arctan\left(\frac{\mathrm{V}}{2f}\right)\right)}\right] \tag{3.4}$$

- El elemento fotosensible es una cuadrícula, un espacio discreto de cierta dimensión o resolución. Los cuadros, llamados pixeles tienen cierto ancho W_P y alto H_P, que usualmente son indicados por el fabricante en las especificaciones técnicas. La resolución en pixeles se especifica como una multiplicación del ancho W por el alto H, en cámaras pequeñas como webcams las resoluciones comunes son: 320x240, 640x480, etc., la

relación entre largo y alto es usualmente 4:3 pero recientemente también se utilizan resoluciones en formatos 16:9 como 1920x1080. Modifiquemos entonces la ec. 3.4 para incluir estos detalles para llegar a la ec. 3.5 que considera la resolución en pixeles y las dimensiones reales de un pixel.

$$\left[\frac{\alpha_u}{\alpha_v}\right] = \left[\frac{2\left(\arctan\left(\frac{W \cdot W_P}{2f}\right)\right)}{2\left(\arctan\left(\frac{H \cdot H_P}{2f}\right)\right)}\right] \tag{3.5}$$

- El centro óptico, el orificio por donde la luz pasa, no puede ser muy pequeño ya que el elemento fotosensible ya sea CMOS o CCD no recibiría suficiente luz y obtendríamos imágenes demasiado obscuras. Aún si pasara suficiente luz, un orificio muy pequeño provoca difracción y como resultado imágenes borrosas (fig. 3.5a), sin embargo, un orificio muy grande permitiría que varios puntos del mundo real sean reflejados en un mismo punto de la imagen (fig. 3.5b) provocando una imagen borrosa. Fuera de foco es como comúnmente se le llama a este problema. En las cámaras reales se utilizan uno o más lentes para que suficiente luz pueda entrar hacia el elemento fotosensible sin producir los defectos mencionados (fig. 3.5c).



a) Orificio muy pequeño b) Orificio muy grande c) Orificio con lentes

Fig. 3.5 Efectos en la imagen al alterar el tamaño del centro óptico

- La fabricación de la cámara no es perfecta, ya sea por el elemento fotosensible, los lentes o simplemente el ensamble ocurren efectos indeseables. De acuerdo con [16] estos defectos pueden ser clasificados en

distorsiones radiales, provocados por errores con la forma de los lentes; y tangenciales, provocados por errores con la alineación del sensor fotosensible. Puesto que el error es característico de un modelo en especial, debe existir un proceso particular para modelarla, calibración.

3.1.3 Calibración

Como se mencionó, existen errores en la producción o ensamble de una cámara que producen que cierto objeto que debería ser observado en cierta coordenada de la imagen sea observado en otro, un punto que debe ser considerado para este proyecto que depende de mediciones de la cámara para determinar la posición de referencias.

En [16] se describen 5 coeficientes para describir las imperfecciones de una cámara: (k_1, k_2, k_3) para distorsiones radiales y (p_1, p_2) para distorsiones tangenciales sin embargo el proceso de fabricación ha mejorado y de acuerdo con [17] es conveniente y recomendable trabajar únicamente con los primeros dos coeficientes radiales e ignorar el resto.

Para calibrar la cámara se utilizó el toolkit de Matlab® ya que el proceso es interactivo y tiene mejor documentación que la versión de OpenCV, pero ambas implementan el mismo modelo de calibración por lo que los resultados deben ser iguales.

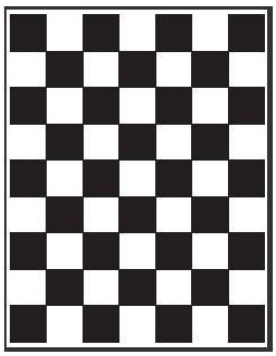


Fig. 3.6 Cuadrícula utilizada para la calibración

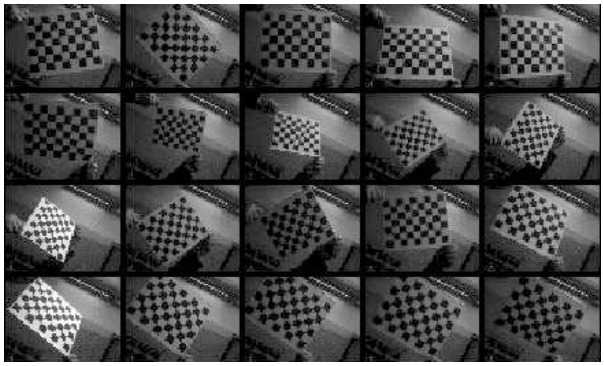


Fig. 3.7 Imágenes utilizadas para la calibración

El proceso consiste en tomar una serie de imágenes a una cuadrícula tipo ajedrez, cuadros blancos y negros intercalados sobre una superficie plana (fig. 3.6), las imágenes se deben tomar variando la posición y distancia como se muestra en la fig. 3.7. No se especifica cuantas son necesarias pero en [20] se recomiendan al menos 15, en este caso se utilizaron 20.

El usuario debe indicar a la herramienta varios datos sobre la cuadrícula: primero delimitarla dentro de la imagen para conocer su orientación; el número de cuadros por renglón y por columna; así como la altura y ancho reales de un cuadro, este proceso se repite para cada imagen suministrada. Con los datos suministrados la herramienta tratará de detectar las esquinas (los límites de cada cuadro) en cada imagen. Con la posición estimada de las esquinas, calculará los parámetros extrínsecos de la cámara (posición y orientación del centro óptico) para luego determinar la posición en donde se deberían encontrar realmente las esquinas en cada imagen. Puede entonces calcular el error en la posición para finalmente modelar la distorsión con los coeficientes antes mencionados. Si el lector está interesado en el proceso así como otras posibles notaciones de los resultados y métodos de calibración se recomienda leer [16,17,18,19] sobre los cuales se basa el modelo de la herramienta. Los resultados se muestran en la tabla 3.1.

Distancia focal	d_u:	388.52751
	d_v:	386.92471
Coeficientes de distorsión	$k_{1:}$	-0.24740
	$k_{2:}$	0.38481

Tabla 3.1 Resultados de la calibración de la cámara.

La distancia focal está expresada con el ancho y alto de un pixel.

Con estos parámetros es posible aproximar las coordenadas correctas de cierto objeto a partir de sus coordenadas distorsionadas por ejemplo la fig. 3.8a muestra una foto tal cual se obtiene de la cámara y la fig. 3.8b su versión no distorsionada. Es evidente que la imagen resultante remueve el efecto que hace que las líneas rectas tengan cierta curvatura cuando se alejan del centro de la imagen, sin

embargo el hecho de que algunos pixeles de la imagen resultante sean una combinación ponderada de varios pixeles de la imagen original puede provocar que la imagen sea borrosa, además el área visible es menor que el de la imagen original y es computacionalmente costosa de calcular, por lo que usualmente siempre se trabaja con imágenes distorsionadas y únicamente las coordenadas de cierto objeto de interés son trasladadas a las coordenadas no distorsionadas. Para tal motivo se introduce la siguiente ec. 3.6, propuestas por [17], una modificación a esta ecuación es utilizada en algunos proyectos de Mono SLAM como [10,21,22].

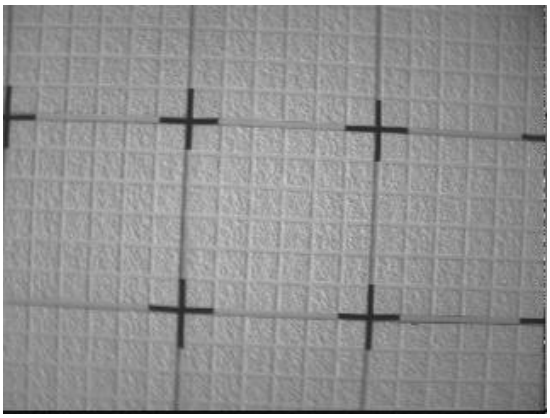


Fig. 3.8a Imagen original distorsionada

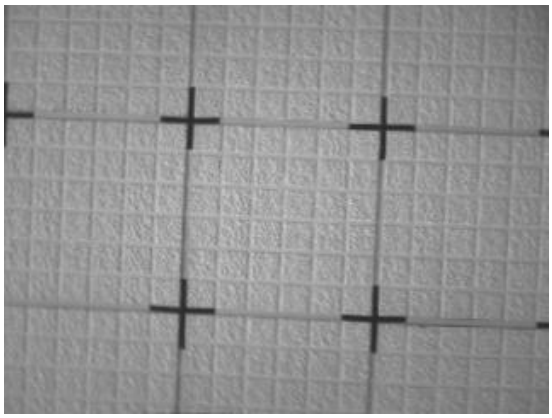


Fig. 3.8b Imagen no distorsionada

$$\left[\frac{u_u}{v_u}\right] = \left[\frac{\left((u_d - C_u)(1 + k_1 r^2 + k_2 r^4)\right) + C_u}{\left((v_d - C_v)(1 + k_1 r^2 + k_2 r^4)\right) + C_v}\right] \tag{3.6}$$

$$r = \sqrt{\left(\frac{1}{f_u}(u_d - C_u)\right)^2 + \left(\frac{1}{f_v}(v_d - C_v)\right)^2}$$

(f_u, f_v) Distancia focal expresada en el ancho y alto físico de un pixel.
(u_u, v_u) Coordenadas no distorsionadas de la imagen.
(u_d, v_d) Coordenadas distorsionadas de la imagen.
(C_u, C_v) Coordenadas del punto *C*, centro de la imagen.
(k_1, k_2) Coeficientes de distorsión.

3.2 Análisis digital de imágenes

La sección previa explica el funcionamiento de una cámara con el fin de poder extraer de una imagen las coordenadas de puntos de interés para nuestro mapa, pero siempre se asumió que se conocía el punto *(u_i, v_i)* en la imagen. En

esta sección se discutirán algunas técnicas para seleccionar puntos y seguirlos durante una secuencia de imágenes, pero antes de hablar de técnicas, iniciemos con algunos conceptos y definiciones útiles.

3.2.1 Imagen

Primero que nada sobre la imagen. Como se explicó, la imagen es cierta cuadricula en un elemento fotosensible que podemos expresar como una función que mapea para cada par *(u, v)* un valor *I*, todos ellos enteros positivos (ec. 3.8).

$$f(u,v) = I \qquad u \in \mathbb{Z}, \mathrm{v} \in \mathbb{Z}, \mathrm{I} \in \mathbb{Z} \tag{3.8}$$

Las imágenes pueden tener varios canales para producir colores, y existen varios formatos para representarlos por ejemplo: YUV, CMY y RGB, éste último, seguramente el más utilizado, contiene un vector por cada par *(u,v)* de 3 elementos, (uno para cada color: **R**: rojo; **G**: verde; **B**: azul). Al valor o cada uno de los valores que se asocian al par de coordenadas en la imagen se le llama **intensidad**, un valor entero positivo, usualmente representado por 8 bits (256 diferentes valores). Algunas técnicas de procesamiento de imágenes se basan en un elemento estadístico que se obtiene a partir de estas intensidades: el **histograma.** El histograma es una representación gráfica de la frecuencia con que aparecen en la imagen todos los posibles valores de intensidad. Con el histograma es posible, en ciertos casos, separar objetos de otros al filtrar por un rango, dando entrada al siguiente punto **segmentación**.

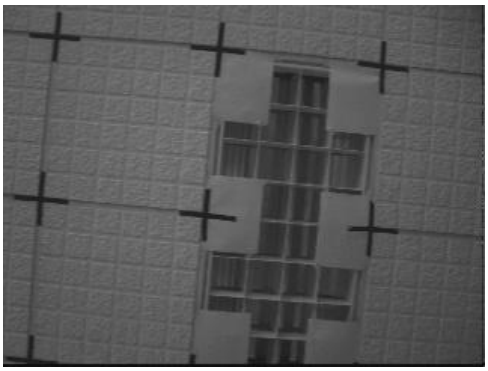


Fig. 3.9a Imagen fuente del histograma

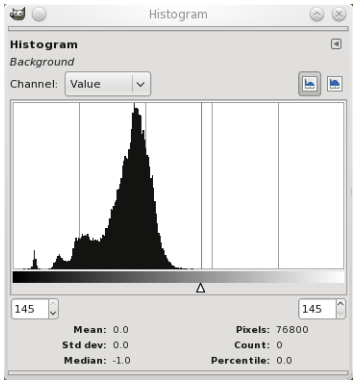


Fig. 3.9b Histograma de la imagen

3.2.2 Segmentación

Como se mencionó anteriormente la imagen tiene una cantidad considerable de datos, pero para obtener la información que se desea, primero se debe procesar la imagen. Para este proyecto que busca construir un mapa de su entorno, es vital encontrar objetos en las imágenes que sirvan como referencias, así como ser capaces de seguir tales objetos durante una secuencia de imágenes o cuadros.

Algunos autores proponer incluir un proceso de separación de los objetos. En [32] se define el proceso de segmentación como la acción de dividir $f(u,v)$, en regiones R_1, R_2, ..., R_n, conectadas no traslapadas (ec. 3.9). Una región conectada hace referencia al hecho de que cada pixel tiene al menos otro pixel con valor distinto a 0 en su vecindad (ya sea hacia las 4 u 8 direcciones posibles con un desplazamiento mínimo en la cuadrícula)

$$\bigcup_{1}^{i} R_i \subseteq f(u,v) \tag{3.9}$$

$$R_i \cap R_j = \emptyset \quad \forall i,j \; i \neq j$$

En el caso más sencillo podemos pensar que existe un único tipo de objeto de interés y que el resto de la imagen es fondo; cada objeto tiene un valor o rango característico de intensidad y distinto al fondo. En algunos casos es posible controlar las variables de iluminación y saber de antemano la intensidad de cierto objeto para definir un valor o **umbral** por ejemplo, una banda transportadora es iluminada siempre desde los mismos puntos, con intensidad constante y predefinida. En otros casos se pueden utilizar métodos estadísticos para obtener un umbral automáticamente como Otsu [31]. Cualquiera que sea el método para obtener el umbral, el proceso que se sigue es **umbralar.** Se filtran los valores de la intensidad que no cumplan con la regla del umbral, y el resultado se simplifica a una imagen binaria, sólo 2 posibles valores de intensidad: 0 y 1, para indicar si existe un objeto o no en cierta posición.

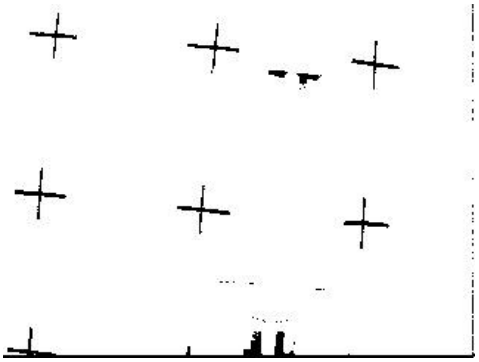


Fig. 3.10a Imagen resultante al umbralar a 40

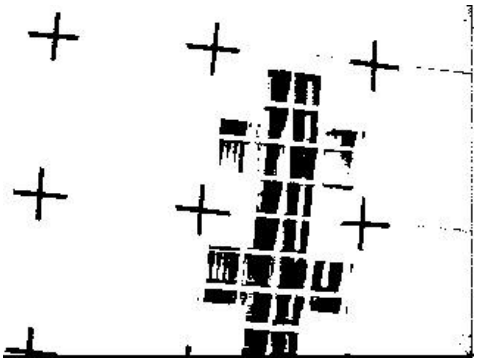


Fig. 3.10b Imagen resultante al umbralar a 60

Luego de analizar las figuras 3.10a y 3.10b, que resultan de umbralar la fig. 3.9, es fácil concluir que son contadas las situaciones donde se puede hacer una segmentación limpia utilizando umbralado directamente, por lo que usualmente se utiliza como parte de un proceso más largo.

3.2.3 Rasgos descriptivos

Podemos ver en la imagen anterior (3.9a) que existen ciertos objetos de interés en la escena, pero la pregunta es ¿cómo describirlos, qué características son útiles para poder volver a detectarlos aún en presencia de cambios?, este apartado es sobre algunos rasgos descriptivos comúnmente utilizados para obtener características visuales de objetos:

I. **Bordes:** Son cambios relativamente bruscos en la intensidad de la imagen, idealmente en la frontera de cada objeto o región. Así que se busca una discontinuidad calculando la primer o segunda derivada de la imagen, al resultado se le llama gradiente. Para calcular la derivada de una imagen se hace uso de ventanas de convolución, una para cada dirección *u* y *v.* Existen varias propuestas: Roberts, Sobel, Scharr. Uno de los detectores de bordes más utilizados es Canny [23] ya que es tolerante al ruido y la imagen resultante tiene bordes con grosor de 1 pixel, aunque en realidad es un algoritmo con 3 pasos donde el primero es el cálculo del gradiente, usualmente utilizando Sobel.

a) Roberts

+1	0
0	-1

G_u

0	+1
-1	0

G_v

b) Sobel

-1	0	+1
-2	0	+2
-1	0	+1

G_u

-1	-2	-1
0	0	0
+1	+2	+1

G_v

c) Scharr

+3	+10	+3
0	0	0
-3	-10	-3

G_u

+3	0	-3
+10	0	-10
+3	0	-3

G_v

Tabla 3.2 Ventanas de convolución para cálculo de gradiente

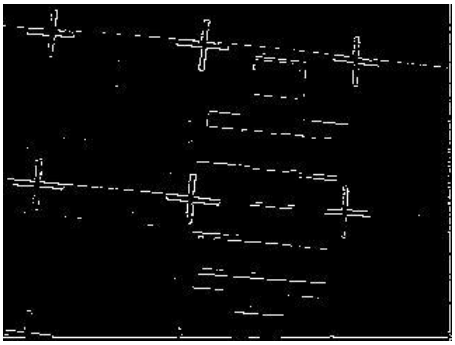


Fig. 3.11a Bordes utilizando Roberts

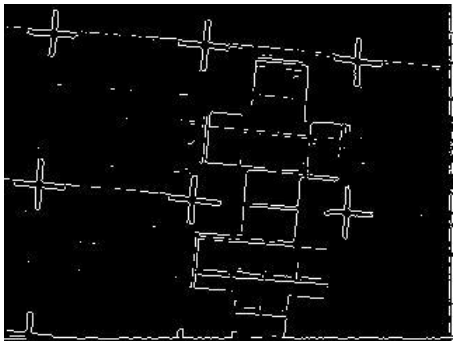


Fig. 3.11b Bordes utilizando Sobel

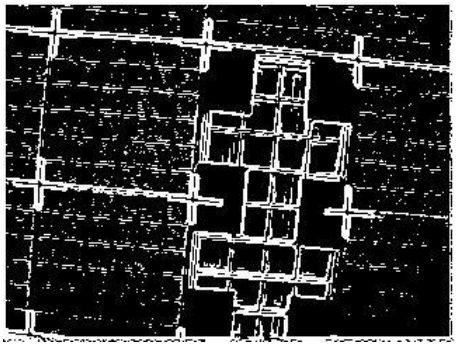


Fig. 3.11c Bordes utilizando Scharr

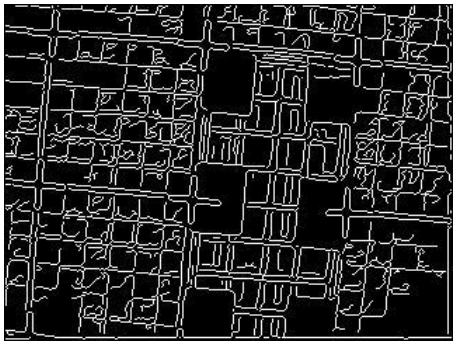


Fig. 3.11d Bordes utilizando Canny

II. Esquinas: Al observar sólo parte de un objeto que no tiene señas particulares como una línea recta, ocurre un problema, cambios en la escena no pueden ser notados. Imaginemos que observamos el borde una puerta, pero en la imagen no puede verse donde inicia o donde termina la línea, aun si la cámara cambia de posición, en la imagen no es posible saber si se sigue observando el mismo segmento de la línea o no. Este problema es conocido como el problema de la apertura y es una característica indeseable de una línea recta solitaria. Por lo cual, la detección de esquinas produce un descriptor más robusto que una línea. Su funcionamiento también parte del gradiente pero la

decisión se basa en los valores propios o eigenvalores de la matriz espacial del gradiente (ec. 3.10), en general el eigenvalor más pequeño debe superar cierto umbral. Ejemplos de estas técnicas: Harris [24] y KLT [25] (por las iniciales de sus autores: Kanade, Lucas y Tomasi), esta última es la más común para el cálculo de flujo óptico y también es común en proyectos de Visual SLAM [35,36].

$$G[f(u,v)] = \begin{bmatrix} G_u \\ G_v \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta u} \\ \frac{\delta f}{\delta v} \end{bmatrix} \tag{3.10}$$

$$\|G[f(u,v)]\| = \sqrt{G_u^2 + G_v^2}$$

$$M = \sum\sum \begin{bmatrix} G_u^2 & G_u G_v \\ G_u G_v & G_v^2 \end{bmatrix}$$

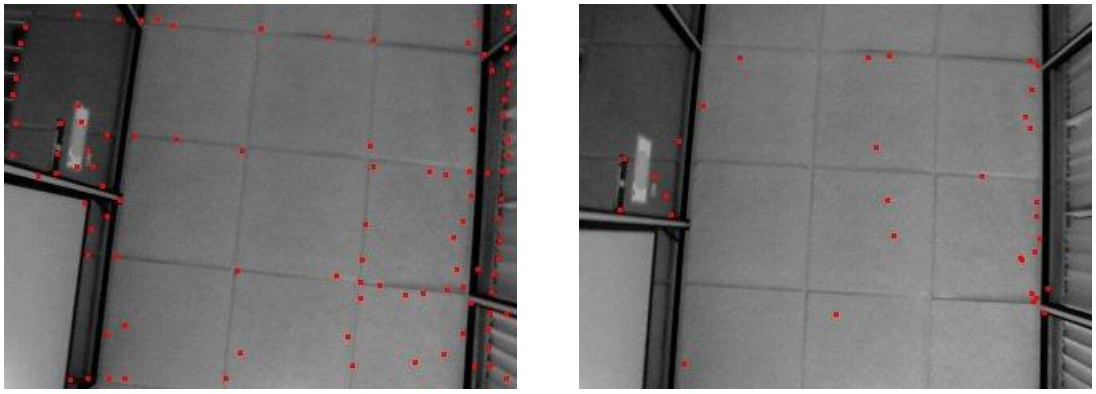


Fig. 3.12 Puntos de interés seleccionados en dos imágenes de una secuencia utilizando KLT 100 puntos fueron seleccionados en la primera imagen, sólo 32 fueron detectados de nuevo

III. **Texturas:** Las técnicas basadas en los conceptos anteriormente mencionados son las más utilizadas en ambientes donde se requiere de un rápido procesamiento, como en el caso de este proyecto. Pero es interesante mencionar una técnica relativamente reciente que está siendo utilizada para diversos proyectos de reconocimiento por ser invariante a cambios de escala, rotación, tolerante a cambios afines y de iluminación: SIFT [27] por sus siglas en inglés: *Scale Invariant Feature Transform,* es una técnica para la selección de puntos de interés esencialmente basada en la diferencia de gaussianas. La

imagen se convoluciona repetidamente con un filtro gaussiano, la diferencia de gaussianas se calcula simplemente restando a cada imagen convolucinada su predecesor hasta alcanzar el umbral de 2σ, luego se reescala la imagen y se empieza de nuevo el proceso (ec. 3.11). Su cálculo es computacionalmente costoso y su descriptor es un vector de 128 campos por lo que también es costoso de almacenar y de comparar entre sí. Se han desarrollado alternativas a esta técnica que buscan reducir el costo de procesar y almacenar el descriptor como PCA-SIFT [29] o SURF [30]. Una implementación de Visual SLAM usando SIFT puede revisarse en [28] y una comparación de eficiencia entre Harris, KLT y SIFT para Visual SLAM puede encontrarse en [37].

$$D(u,v,\sigma) = L(u,v,k\sigma) - L(u,v,\sigma) \tag{3.11}$$

$$D(u,v,\sigma) = \big(G(u,v,k\sigma) - G(u,v,\sigma)\big) \otimes I(u,v)$$

$$G(u,v,k\sigma) = \frac{1}{2\pi\rho^2} exp\left(\frac{-(u^2+v^2)}{2\rho^2}\right)$$

$\otimes$: Convolución.

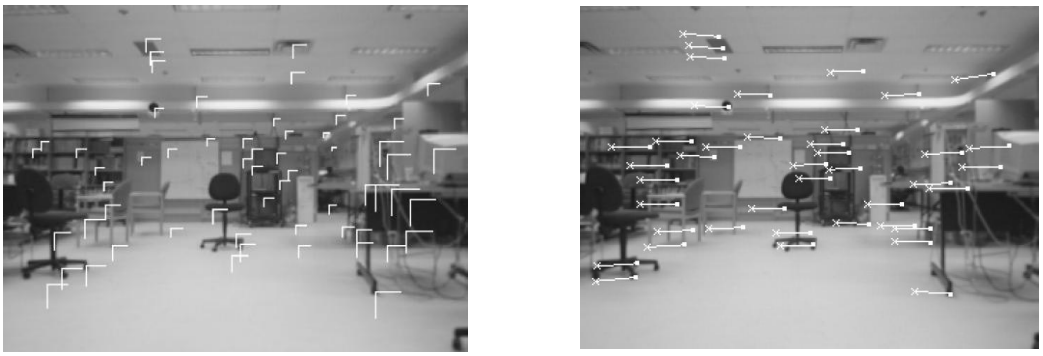


Fig. 3.13 Puntos de interés seleccionados en dos imágenes de una secuencia utilizando SIFT

60 puntos fueron seleccionados en la primera imagen, sólo 41 fueron detectados de nuevo

Es importante señalar que si bien existen numerosas y valiosas aportaciones en el tema, ninguna puede resolver bajo cualquier condición el problema de selección de marcas, o puntos de interés, como también es común llamarle a éste proceso. No hay una mejor técnica sino una mejor adaptada a ciertas condiciones, por lo que se debe escoger y ajustar dependiendo del entorno.

3.3 Dispositivos FPGA's

Los dispositivos FPGA's o arreglos de compuertas programables son circuitos integrados digitales que contienen bloques lógicos e interconexiones reconfigurables. Los programadores definen un diseño de propósito específico para ser implementado sobre estos dispositivos, la flexibilidad de este modelo les permite realizar una amplia variedad de tareas.

3.3.1 Aplicaciones

Cuando el primer FPGA apareció en escena a mediados de la década de los 80's, los FPGA's eran ampliamente usados para implementar *glue logic* (término usado para referirse a las relativamente pequeñas unidades de lógica que son usadas para conectar dispositivos, funciones o bloques lógicos más grandes), máquinas de estado de complejidad media, y tareas de procesamiento de datos relativamente limitados. A principios de la década de los 90's su gran mercado era la industria de telecomunicaciones que involucraba el procesamiento de grandes bloques de datos. Sin embargo conforme el tamaño y desempeño de los dispositivos FPGA's aumentó hacia finales de esa misma década, su uso también incluía aplicaciones para el consumidor, el sector automotriz e industrial.

Los FPGA's son típicamente usados para modelar diseños ASIC (circuitos integrados desarrollados para aplicaciones específicas) o para proporcionar una plataforma de hardware sobre la cual verificar la implementación física de nuevos algoritmos. Sin embargo, su bajo costo de desarrollo y su corto tiempo para comercializar suponen que son cada vez más fáciles de encontrar en productos finales (algunos de los vendedores de FPGA's avanzados actualmente tienen dispositivos que compiten directamente con los ASIC's).

Desde principios del año 2000, los FPGA's de alto desempeño cuentan ya con millones de compuertas y sus precios son relativamente accesibles. Algunos de estos dispositivos exhiben núcleos de microprocesadores embebidos, interfaces

de alta velocidad de entrada/salida como tarjetas de red o adaptadores de video, etc. Los FPGA's de hoy en día pueden ser usados para implementar casi cualquier cosa, incluyendo dispositivos de comunicación, radar, procesamiento digital de señales (DSP); todo en forma de componentes de un sistema sobre un chip o *system-on-chip* (SoC) que contienen elementos de hardware y software.

Para ser un poco más específicos, los FPGA´s son actualmente empleados en cuatro áreas del mercado avanzado:

- **ASIC y silicio personalizado:** Los FPGA´s de hoy son cada vez más usados para implementar una variedad de diseños que solo podrían haber sido realizados sobre ASIC's y silicio personalizado, por lo que hoy en día son una alternativa más económica.
- **Procesamiento digital de señales:** El DSP de alta velocidad ha sido tradicionalmente implementado usando específicamente microprocesadores hechos a la medida llamados procesadores digitales de señales o *digital signal processors* (DSP's). Sin embargo, los FPGA´s de hoy contienen multiplicadores embebidos, ruteo aritmético dedicado, y una gran cantidad de RAM sobre el chip, los cuales facilitan operaciones de DSP. Cuando estas características son combinadas con el paralelismo masivo proporcionado por los FPGA's es posible obtener un mejor desempeño que en un DSP.
- **Microcontroladores embebidos:** Funciones pequeñas de control han sido típicamente manejadas por procesadores embebidos de propósito especial llamados microcontroladores. Estos dispositivos de bajo costo contienen un programa sobre el chip y memoria de instrucciones, temporizadores, y periféricos de entrada/salida plegados alrededor de un núcleo de procesador. Los precios de los FPGA´s están bajando, y aún los dispositivos más pequeños ahora tienen suficiente capacidad para implementar un núcleo de procesador combinado con una selección de funciones personalizadas de entrada/salida. Haciendo que esta tecnología sea atractiva para aplicaciones de control embebido.

- **Comunicaciones de capa física:** Los FPGA´s han sido ampliamente usados para implementar *glue logic* interconectando los chips de comunicación de la capa física y las capas del protocolo del nivel de red. El hecho es que los FPGA´s pueden contener múltiples transceptores de alta velocidad que implican que funciones de red y de comunicación pueden ser consolidadas en un solo dispositivo.
- **Cómputo reconfigurable:** Esto se refiere a la explotación inherente del paralelismo y la reconfigurabilidad proporcionada por los FPGA´s para crear hardware que acelere algoritmos de software. Varias compañías están actualmente construyendo enormes motores de cómputo reconfigurables basados en FPGA´s para tareas que se benefician del paralelismo como clasificación, análisis criptográfico, etc. [26]

3.3.2 Historia

Los primeros circuitos integrados programables fueron generalmente conocidos como dispositivos lógicos programables o *Programmable Logic Devices* (PLD's). Los componentes originales entraron en escena en 1970 en forma de Memorias programables de solo lectura o *Programmable Read-Only Memories* (PROM's), después surgieron unos nuevos dispositivos llamados *Complex PLD's* (CPLD's). A menudo suele encontrarse el término *Simple PLD* (SPLD), que muchas veces es usado como sinónimo de PLD, sin embargo, otras fuentes consideran a los PLD's como un superconjunto que contiene a los SPLD's y a los CPLD's. En la Fig. 3.14 se muestran algunas variantes de los PLD´s.

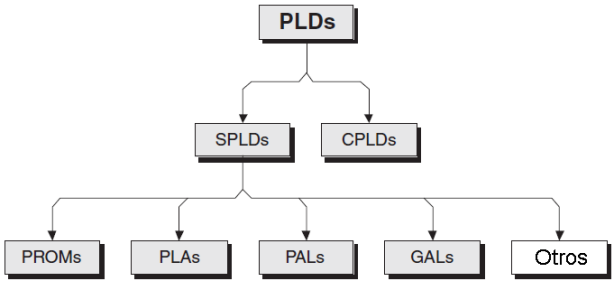


Fig. 3.14 Variantes de PLD´s.

Los primeros PLD´s simples fueron las PROM´s, las cuales aparecieron alrededor de 1970. Una forma de visualizar estos dispositivos consistía en verlos como un arreglo fijo de funciones AND controlando un arreglo programable de funciones OR, por ejemplo, la fig. 3.15 muestra una PROM de 3-entradas y 3-salidas. Éstas fueron originalmente usadas para almacenar instrucciones de programas y datos constantes. Sin embargo, los ingenieros de diseño también las usaron para implementar funciones lógicas simples como tablas de búsqueda o *LookUp Table* (LUT) y máquinas de estado. Para ocuparse de las limitaciones impuestas por la arquitectura PROM, el siguiente paso en la evolución de los PLD's fueron los arreglos lógicos programables (PAL's), los cuales llegaron alrededor de 1975. Estos dispositivos fueron mayormente usados como PLD´s configurables porque los arreglos AND y OR eran programables. La fig. 3.16 muestra un PAL de 3-entradas y 3-salidas. A finales de los 70's y principios de los 80's empezaron a surgir PLD's más sofisticados conocidos como CPLD´s. Un gran avance ocurrió en 1984, cuando la empresa Altera introdujo un CPLD basado en una combinación de tecnologías CMOS y EPROM que permitió conseguir enormes densidades, con un consumo relativamente bajo de energía.

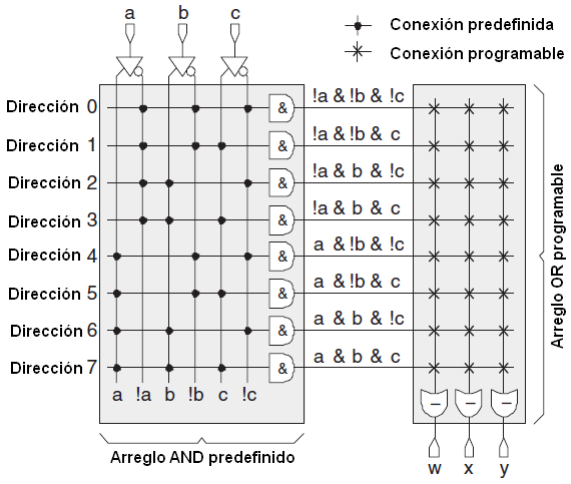


Fig. 3.15 PROM no programada

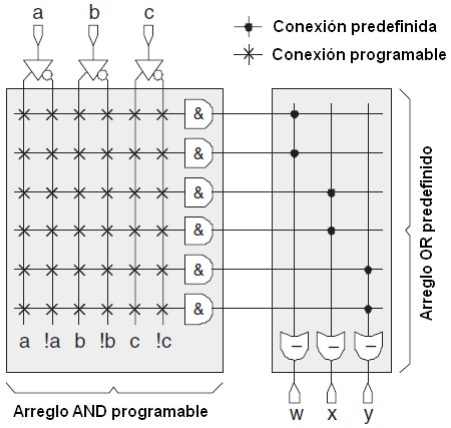


Fig. 3.16 PAL no programada

El avance conceptual de Altera consistió en utilizar un arreglo de interconexiones centrales con menos del 100% de conectividad. Esto incrementó la complejidad de las herramientas de software de diseño, pero mantuvo la velocidad, potencia y costo de estos dispositivos escalables.

En la fig. 3.17 se muestra una representación de alto nivel de un CPLD. En realidad, todas estas estructuras están formadas sobre una misma pieza de silicio. Por ejemplo, la matriz de interconexión programable puede contener muchos conductores, pero esto es más de lo que puede ser manejado por los bloques individuales de SPLD's, los cuales son capaces de ser alojados en un número limitado, probablemente sólo una tercera parte. Así que los bloques SPLD son interconectados usando una matriz con alguna forma de multiplexor programable (fig. 3.18).

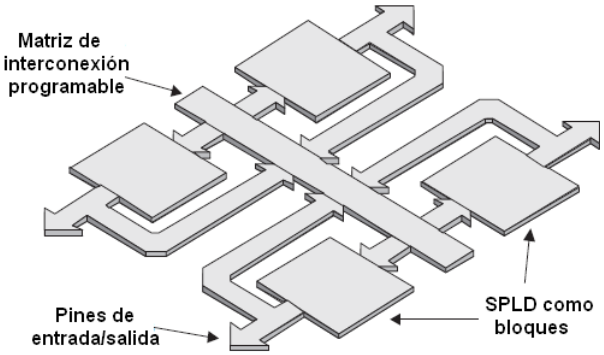


Fig. 3.17 Estructura de un CPLD genérico

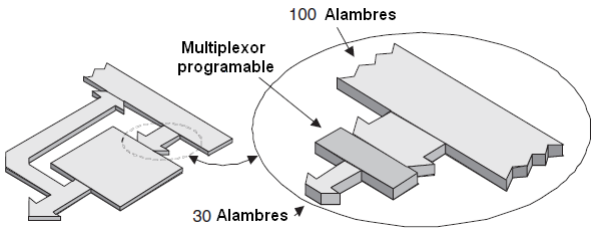


Fig. 3.18 Uso de multiplexores programables

Conforme fue incrementando la complejidad, surgieron cuatro clases de ASIC, estos son: arreglos de compuertas, ASIC's estructurados, dispositivos de celdas estándar y chips completamente personalizados.

En el caso de dispositivos completamente personalizados, los ingenieros de diseño tienen completo control sobre la capa de la máscara usada para fabricar los chips de silicio. Los vendedores de ASIC no prefabrican cualquier componente de silicio y no proporcionan bibliotecas de funciones y compuertas lógicas predefinidas. El diseño de dispositivos completamente personalizados es

altamente complejo y consume tiempo, pero los chips resultantes contienen la máxima cantidad de lógica con un gasto mínimo de silicio.

Para resolver los problemas asociados con los arreglos de compuertas, los dispositivos de celdas estándar llegaron a estar disponibles a principios de los 80's. Estos componentes tienen muchas similitudes con los arreglos de compuertas. El concepto de celda estándar permite a cada función lógica ser creada utilizando el mínimo número de transistores con componentes no redundantes, y las funciones pueden ser posicionadas para facilitar las conexiones entre ellas. Los dispositivos de celdas estándar, sin embargo, proporcionan una utilización más cerca del óptimo de silicio que los arreglos de compuertas.

Los ASIC's estructurados aparecieron a inicios de los 90's. La idea es que estos dispositivos pueden ser personalizados solo en las capas de metal (justo como un arreglo de compuertas). La diferencia es que, debido a la gran complejidad de los bloques de ASIC estructurado, muchas de las capas de metal están predefinidas.

A inicios de los 80's, al parecer hubo un vacío en la continuidad de los circuitos integrados. A finales de esta época, hubieron dispositivos programables como SPLD's y CPLD's, los cuales eran altamente configurables y de rápido diseño, pero no soportaban funciones grandes o complejas. También hubo ASIC´s, estos podían soportar funciones extremadamente grandes y complejas, pero fueron dolorosamente caros para diseñar.

Los primeros FPGA's desarrollados por Xilinx estuvieron disponibles en el mercado alrededor de 1984, estaban basados en el concepto de bloques lógicos programables, los cuales consistían de una tabla de búsqueda (LUT), un registro que podría actuar como un flip-flop o latch, y un multiplexor. La fig. 3.19 muestra un bloque lógico programable simple (los bloques lógicos en los FPGA's modernos pueden ser significativamente más complejos). Cada FPGA contiene un número grande de estos bloques lógicos programables.

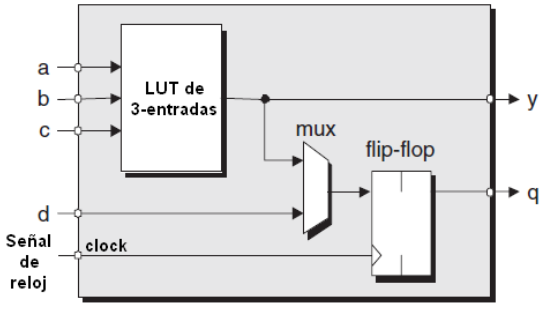


Fig. 3.19 Elementos clave formando un bloque lógico programable simple

Un FPGA completo contiene un gran número de bloques lógicos programables "islas" cercadas por un "mar" de interconexiones programables (fig. 3.20). Como es usual, esta ilustración de alto nivel es solo una representación abstracta. En realidad, todo está implementado con transistores e interconexiones sobre una misma pieza de silicio usando técnicas de creación estándar de circuito integrados.

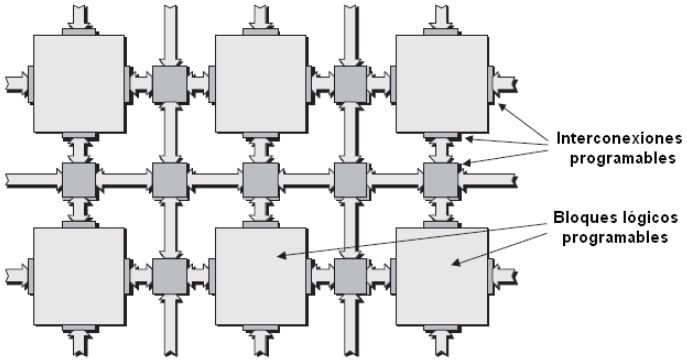


Fig. 3.20 Vista simple de una arquitectura genérica de un FPGA

Finalmente los FPGA´s resultaron ser el puente del vacío provocado entre los PLD's y los ASIC's. De una u otra manera, ellos fueron altamente configurables y gozaron de rapidez en el diseño y tiempo de modificación asociados con PLD's. Igualmente, podrían ser usados para implementar funciones enormes y complejas que estuvieron previamente en el dominio de los ASIC's [26].

3.3.3 Arquitectura de los FPGA's

La mayoría de los FPGA's están basados en el uso de celdas de configuración SRAM, lo que significa que pueden ser configurados una y otra vez. Las principales ventajas de esta técnica son que los diseños pueden ser rápidamente implementados y probados, mientras evolucionan los estándares y protocolos pueden ser reubicados de una manera relativamente fácil. Además, cuando el sistema es activado por primera vez, el FPGA puede inicialmente ser programado para desempeñar una función tal como auto-prueba o prueba tarjeta/sistema, y puede posteriormente ser reprogramado para desempeñar su tarea principal. Otra gran ventaja de las estrategias basadas en SRAM es que estos dispositivos están a la vanguardia en tecnología. Los vendedores de FPGA's pueden pensar en el hecho de que muchas otras compañías especializadas en dispositivos de memoria gastan enormes recursos en investigación y desarrollo de esta área. Además, las celdas SRAM son creadas usando exactamente la misma tecnología CMOS como el resto del dispositivo.

En el pasado, los dispositivos de memoria fueron usados para calificar los procesos de fabricación asociado con un nuevo nodo de tecnología. Recientemente, la mezcla de tamaño, complejidad y regularidad asociada con las últimas generaciones de FPGA's han resultado en que estos dispositivos sean usados para esta tarea. Una ventaja del uso de los FPGA´s sobre dispositivos de memoria para calificar el proceso de fabricación es que, si hay un defecto, la estructura del FPGA es tal que es más fácil de identificar y localizar el problema.

Desafortunadamente, un aspecto negativo de los dispositivos basados en SRAM es que tienen que ser reconfigurados cada vez que el sistema sea activado. Esto requiere el uso de un dispositivo especial de memoria externo (el cual tiene un costo asociado y consumo sobre la tarjeta) o de un microprocesador sobre la tarjeta.

Otra desventaja de los dispositivos basados en SRAM es que puede ser difícil proteger la propiedad intelectual en la forma del diseño. Esto es porque el archivo de configuración usado para programar el dispositivo es almacenado en un dispositivo de memoria externo. Existe una buena noticia a pesar de esto, algunos FPGA's basados en SRAM soportan el concepto de *encriptamiento de trama de bits*. En este caso, el dato de configuración final es encriptado antes de que sea almacenado en el dispositivo de memoria. La clave de encriptamiento es cargada en un registro especial del FPGA. En conjunción con alguna lógica asociada, esta clave permite que la configuración pueda ser desencriptada cuando esté siendo cargada en el dispositivo.

A diferencia de los dispositivos basados en SRAM, los cuales son programados mientras residen en el sistema, los dispositivos basados en *antifuse* son programados fuera de línea usando un dispositivo programador especial.

Los promotores de los FPGA's basados en *antifuse* están orgullosos de apuntar a una variedad (no significativa) de ventajas. Primero que todo, estos dispositivos son no volátiles (sus datos de configuración permanecen mientras el sistema está inactivo), lo que significa que hay disposición inmediata cuando el sistema es iniciado (encendido). Su no volatilidad implica que no requieren de memoria externa para almacenar sus datos de configuración, lo cual ahorra costos en componentes adicionales y conserva el estado de la tarjeta.

Una notable ventaja de los FPGA's basados en *antifuse* es el hecho de que su estructura interconectada es naturalmente "tolerante a la radiación", lo que significa que son relativamente inmunes a los efectos de la radiación. Una vez que un *antifuse* ha sido programado, este no puede ser alterado.

Los FPGA's basados en E^2PROM o FLASH son similares a sus contrapartes basados en SRAM en que sus celdas de configuración están conectadas juntas en una larga cadena de registros de desplazamiento. Estos dispositivos pueden ser configurados fuera de línea usando un dispositivo programador. Algunas versiones

son programables en sistema, pero su tiempo de programación es aproximadamente tres veces el de un componente basado en SRAM.

Es común clasificar a los FPGA's como de *grano fino* o *grano grueso*. En el caso de la arquitectura de grano fino, cada bloque lógico puede ser usado para implementar solo una función muy simple. Por ejemplo, podría ser posible configurar el bloque para actuar como cualquier función de 3-entradas, tal como una compuerta lógica (AND, OR, NAND, etc.) o un elemento de almacenamiento (flip flop tipo-D, latch tipo-D, etc.).

Por otro lado, para implementar *glue logic* y estructuras irregulares como máquinas de estados, las arquitecturas de grano fino son particularmente eficientes cuando ejecutan algoritmos que se benefician de las implementaciones masivamente paralelas. Se dice que estas arquitecturas ofrecen algunas ventajas con interés particular en tecnología de síntesis lógica, la cual es convertida hacia arquitecturas ASIC de grano fino.

En el caso de arquitecturas de grano grueso, cada bloque lógico contiene una cantidad relativamente grande de lógica comparada contra sus contrapartes de grano fino. Por ejemplo, un bloque lógico podría contener cuatro LUT's de 4-entradas, cuatro multiplexores, cuatro flip flops tipo-D, y alguna lógica rápida de acarreo.

Una consideración importante con respecto a la granularidad de la arquitectura es que las implementaciones de grano fino requieren un número relativamente grande de conexiones dentro y fuera de cada bloque comparado con la cantidad de funcionalidad que pueden ser soportados por aquellos bloques. Como la granularidad de los bloques incrementa con grano medio o más alto, la cantidad de conexiones dentro de los bloques disminuye comparado con la cantidad de funcionalidad que pueden soportar. Esto es importante porque la interconexión de interbloques programables cuenta para la gran mayoría de retardos asociados con señales a medida que se propagan a través de un FPGA.

Hay dos encarnaciones fundamentales de bloques lógicos programables usados para formar arquitecturas de grano medio: las basadas es MUX (multiplexores) y las basadas en LUT (*lookup table*):

- **Basadas en MUX:** Como ejemplo de aquellas basadas en MUX, considere una forma en la que la función de 3-entradas *y=(a&b)|c* podría ser implementada usando un bloque que contiene solo multiplexores (fig. 3.21). El dispositivo puede ser programado tal que cada entrada al bloque es representada con un '0' lógico o un '1' lógico, o verdadero y la versión inversa de una señal (*a*, *b*, o *c* en este caso) viniendo desde otro bloque o desde una entrada primaria al dispositivo. Esto permite a cada bloque ser configurado en miles de formas para implementar una gran cantidad de funciones posibles (en la fig. 3.21 la 'x' señala una entrada irrelevante, es decir, puede ser cero o uno, no importa).

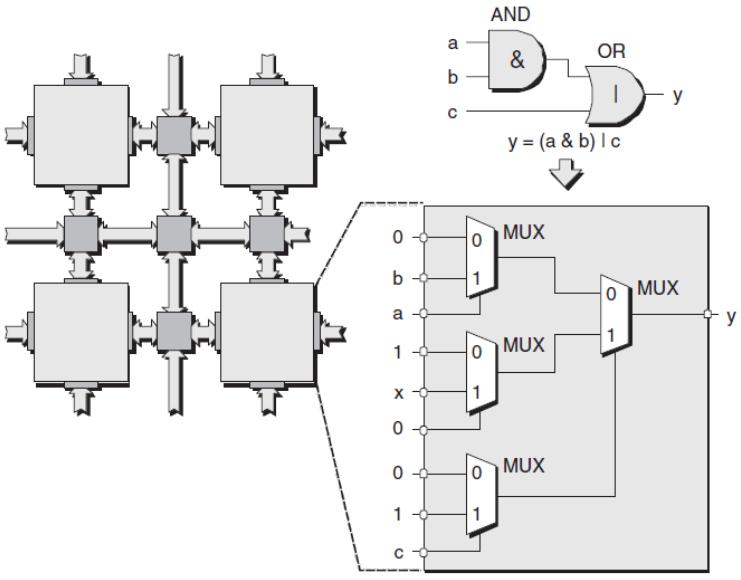


Fig. 3.21 Bloque lógico basado en MUX (multiplexores).

- **Basadas en LUT:** El concepto fundamental de las LUT's es relativamente simple. Un grupo de señales de entrada es usado como un índice (apuntador) a una tabla de búsqueda. El contenido de esta tabla es organizada tal que la celda apuntada por cada combinación a la entrada

contiene el valor deseado. Suponiendo que la LUT está formada de celdas SRAM (podrían ser celdas *antifuse*, E^2PROM o FLASH), una técnica común es usar las entradas para seleccionar la celda SRAM usando una cascada de compuertas de transmisión como se muestra en la fig. 3.22. Si una compuerta de transmisión está habilitada (activa), esta pasa la señal recibida a la salida. Si la compuerta está deshabilitada, la señal de salida es eléctricamente desconectada del alambre. El símbolo de la compuerta de transmisión con un pequeño círculo indica que esta compuerta estará activa con un cero lógico sobre la entrada de control.

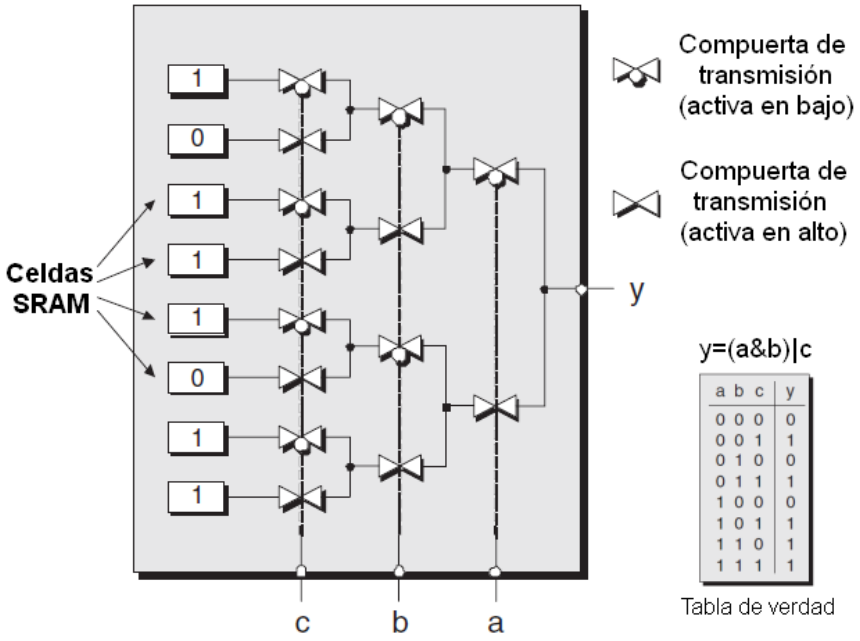


Fig. 3.22 LUT basada en compuertas de transmisión

Un bloque lógico puede tener además de una o más LUT's, otros elementos tales como multiplexores y registros. Cada vendedor de FPGA's nombra las cosas a su manera, por ello, el bloque de construcción del núcleo en un FPGA moderno de Xilinx es llamado celda lógica o *logic cell* (LC). Entre otras cosas, una LC comprende una LUT de 4-entradas (que puede actuar como una RAM de 16*1 ó un registro de desplazamiento de 16-bits), un multiplexor y un registro (fig. 3.23).

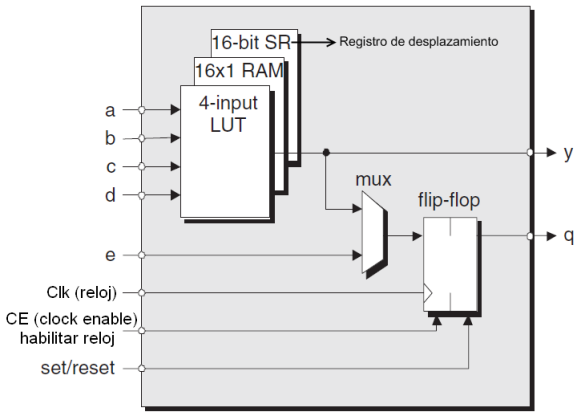


Fig. 3.23 Celda lógica simplificada de Xilinx

El registro puede ser configurado para actuar como un flip flop o como un latch. La polaridad del reloj (flanco de subida o flanco de bajada) puede ser configurada, así como la polaridad de las señales de habilitación del reloj y set/reset (activas en alto o activas en bajo).

El equivalente del bloque de construcción del núcleo en un FPGA de Altera es llamado elemento lógico o *logic element* (LE). Hay diferencias entre un LC de Xilinx y un LE de Altera, pero los conceptos son similares.

En un nivel de jerarquía más elevado encontramos los *slices* y los bloques lógicos configurables o *configurable logic block* (CLB) de Xilinx. Los slices pueden contener dos o más LC's y los CLB's pueden contener cuatro o más slices. En el caso de Altera encontramos bloques de arreglos lógicos o *Logic Array Blocks (LAB's)* que consisten de varios LE's.

Cada fabricante de FPGA´s incorpora elementos interesantes en sus arquitecturas, a continuación se mencionan y explican brevemente algunos:

- **RAM embebida:** Muchas aplicaciones requieren el uso de memoria, por tanto los FPGA´s incluyen secciones relativamente grandes de memoria RAM embebida conocidas como *e-RAM* o bloque RAM. Dependiendo de la arquitectura del componente, estos bloques pueden encontrarse alrededor del dispositivo, dispersos sobre el chip u organizados en columnas.

- **Multiplicadores embebidos, sumadores, MAC's:** Algunas funciones como los multiplicadores son lentos si son implementados conectando un gran número de bloques lógicos programables. Como este tipo de funciones son requeridas para muchas aplicaciones, muchos FPGA's incorporan bloques multiplicadores embebidos. Estos son usualmente alojados en las cercanías de los bloques de memoria RAM embebidos. Una operación muy común en aplicaciones de procesamiento digital de señales o *Digital Signal Proccessing* (DSP) es la llamada multiplica-y-acumula o *multiply-and-accumulate* (MAC), esta operación multiplica dos números y suma el resultado a un total almacenado en un acumulador.
- **Pines de entrada/salida:** Los FPGA´s de hoy pueden tener 1000 o más pines, los cuales están organizados en un arreglo a través de la base del encapsulado. Cuando el chip de silicio se encuentra dentro del encapsulado, las estrategias de encapsulado flip-chip permiten a los pines de alimentación, tierra, reloj y de entrada/salida ser presentados a través de la superficie del chip.

3.3.4 Lógica programable

Como consecuencia de la creciente necesidad de integrar un mayor número de dispositivos en un gran circuito integrado, se desarrollaron herramientas de diseño que auxilian al ingeniero a integrar diseños con mayor complejidad. En la década de los años 50's varias empresas tenían sus propios estándares, todos orientados al área industrial pero no existían fuera de la empresa que los inventó. También existían opciones del ámbito universitario pero no tenían soporte o mantenimiento adecuados. Fue hasta la década de los años 80's que surgieron lenguajes de descripción de hardware como VHDL, Verilog, ABEL, AHDL, etc. Estos lenguajes permitían abordar un problema lógico a nivel funcional, definiendo entradas y salidas, lo cual facilita la elaboración de un diseño complejo. Una de las características de los lenguajes de descripción de hardware es que permiten

describir en diferentes niveles de abstracción: funcional o transferencia de registros.

Hoy en día existen dos lenguajes de descripción de hardware ampliamente difundidos y soportados por las herramientas de diseño de FPGA's: VHDL y Verilog. En general estos lenguajes son equivalentes y es posible sustituir línea a línea uno por otro, aunque su sintaxis es distinta. En este proyecto se utilizó en general VHDL por lo que solo se discutirá sobre este lenguaje.

VHDL es un estándar de IEEE diseñado por un comité por lo que no tiene dependencia con algún proveedor, tipo de diseño o dispositivo en particular. Utiliza una notación formal y permite el reúso de código.

La estructura general de un programa VHDL está formada por módulos o entidades de diseño, cada uno de ellos compuesto por un conjunto de declaraciones e instrucciones que definen y describen el comportamiento de un sistema digital.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ANDGATE is
port (
IN1 : in std_logic;
IN2 : in std_logic;
OUT1: out std_logic);
end ANDGATE;

architecture RTL of ANDGATE is
begin
OUT1 <= IN1 and IN2;
end RTL;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ANDOR is
port (IN1, IN2, IN3: in STD_LOGIC;
OUT1: out STD_LOGIC);
end;

architecture STRUCTURE of ANDOR is

component ANDGATE
port (IN1, IN2: in STD_LOGIC;   OUT1: out STD_LOGIC);
end component;
component ORGATE
port (IN1, IN2: in STD_LOGIC;   OUT1: out STD_LOGIC);
end component;
signal B: STD_LOGIC;

begin
G1: ANDGATE port map (IN1, IN2, B);
G2: ORGATE port map (IN3, B, OUT1);
end;
```

Tabla 3.3 Ejemplo de código VHDL

El código anterior describe al componente ANDOR, que agrupa a otros dos componentes: AND y OR para que funcionen como un bloque más complejo que ejecuta la operación *y=(a&b)|c* (fig. 3.21). El lenguaje permite de esta manera una programación modular y la reutilización de código.

Capítulo 4

Desarrollo de la propuesta

Este trabajo es una implementación de Visual SLAM, que en favor de la meta de crear robots autónomos, ejecuta toda su lógica desde una tarjeta de desarrollo con un dispositivo FPGA. Un sistema que con recursos limitados busca resolver un objetivo muy particular, crear un mapa de su entorno utilizando visión.

El robot puede ser dividido en dos partes físicas: la parte inferior o base que es la plataforma móvil y la parte superior donde se encuentra la tarjeta de desarrollo FPGA y sus aditamentos donde el mapa es creado y administrado.

4.1 Plataforma móvil

La plataforma móvil tiene tres responsabilidades:

I. Proveer soporte para el resto de los componentes.
II. Proveer energía eléctrica al resto de los componentes.
III. Proveer una trayectoria.

I. Proveer soporte para el resto de los componentes.

La plataforma tiene dos bases de acrílico de 6 mm. de grosor donde los componentes están asegurados. En la base inferior están colocados los motores de corriente directa a cada costado de la parte posterior, cada uno de ellos esta empotrado a un reductor mecánico de plástico, y éste a su vez con las llantas; una media esfera de plástico colocada debajo de la base inferior funciona como rueda omnidireccional, ésta se montó sobre una base que la coloca a la misma altura que las llantas traseras; los sensores de distancia a cada costado del frente y finalmente el arreglo de baterías al centro.

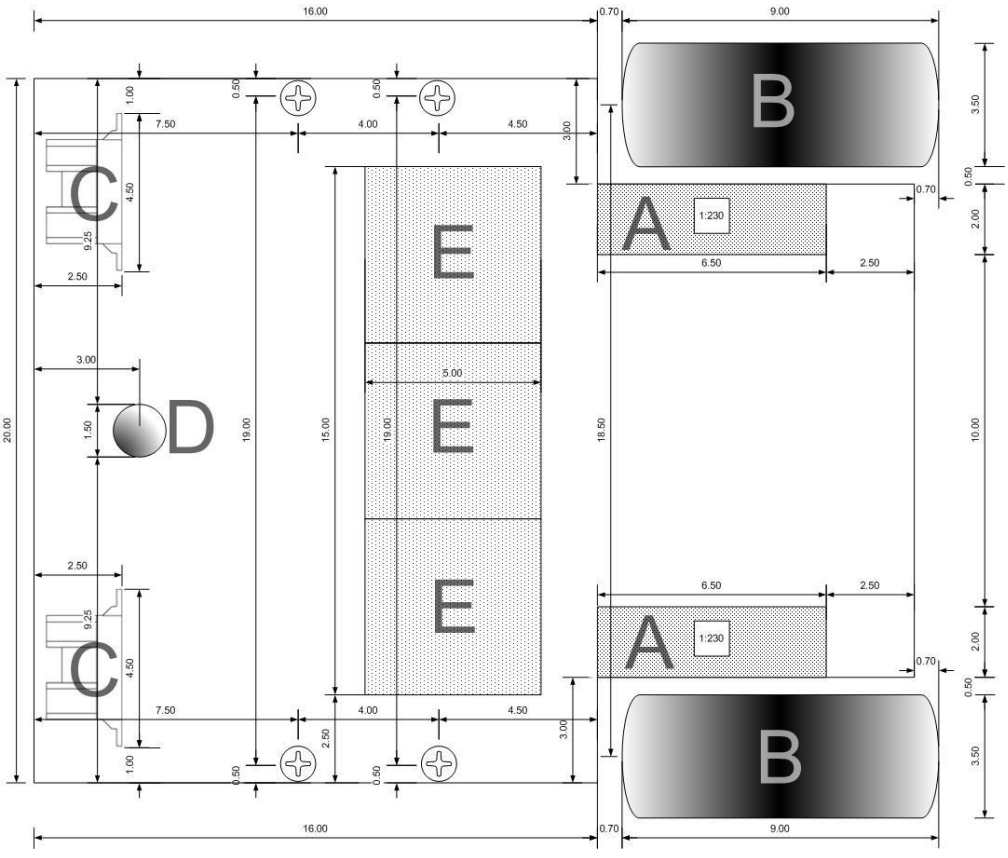


Fig. 4.1 Esquemático base inferior

Código	Descripción	Unidades
A	Motor reductor 1:230.	2
B	Llanta de 9cm de diámetro y 3.5 cm de ancho.	2

C	Sensor de distancia infrarrojo SHARP GP2Y0A02YK0F.	2
D	Esfera de plástico de 1.5cm de diámetro.	1
E	Baterías de ácido selladas de 6volts, puede utilizar baterías de 1 Amp/hr o 4 Amp/hr.	3

Tabla 4.1 Elementos base inferior

Las baterías de 6 volts 1 amp/hr proveen una autonomía corta, alrededor de 45 minutos, pero no deforman la forma de las llantas por lo que se prefirió mantener esta configuración. La configuración con baterías de 4Amp/hr será preferible junto con una técnica de control que contrarreste el efecto que provoca la deformación de las llantas durante las vueltas. Sobre el arreglo de baterías se encuentra una placa fenólica con un microcontrolador encargado de general la trayectoria, sobre el cual se hablará posteriormente.

En la base superior se encuentran los soportes sobre los que se coloca la tarjeta FPGA; y la pantalla gráfica LCD.

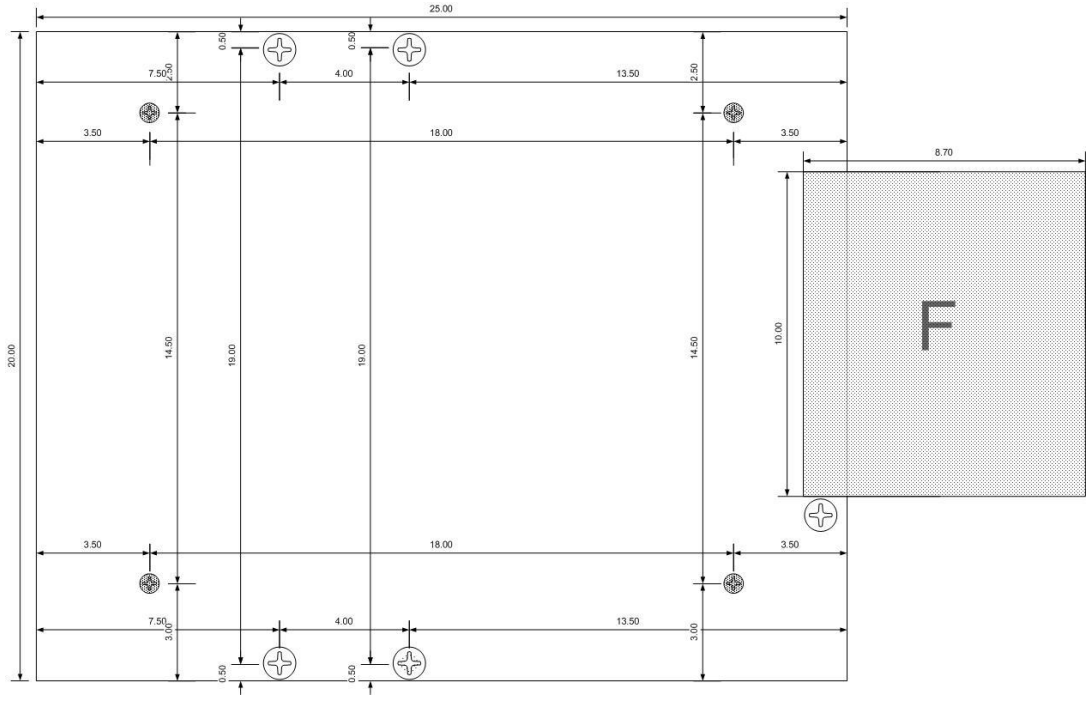


Fig. 4.2 Esquemático base superior

Código	Descripción	Unidades
F	Pantalla gráfica LCD de 3.6’, 320x240 Terasic TRDB_LCM	1

Tabla 4.2 Elementos base superior

Sobre la base superior de acrílico se encuentran soportes metálicos para asegurar y ajustar la altura de la cámara que enfoca hacia arriba para detectar marcas en el techo. La cámara se coloca sin inclinación, paralela a la superficie sobre la que circula el robot, de esta manera la parte inferior de una imagen contiene objetos enfrente del robot, mientras que objetos en la parte superior de la imagen están situados detrás del robot.

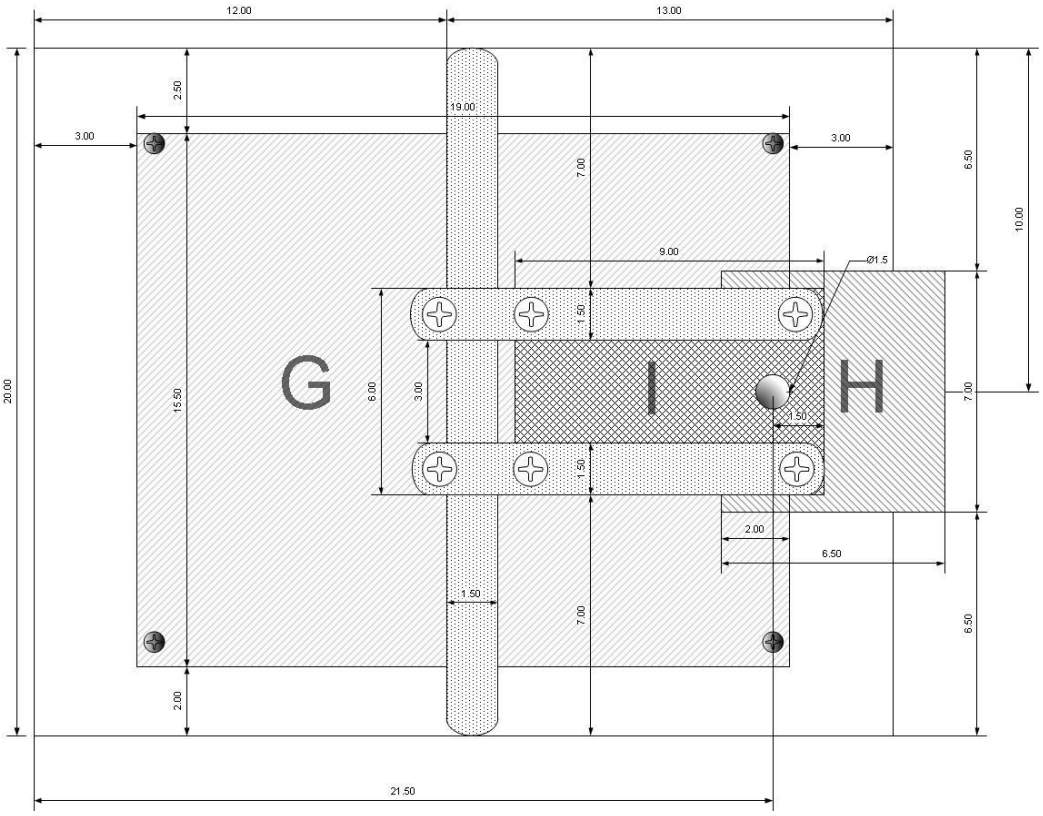


Fig. 4.3 Esquemático base cámara

Como se explicó en el apartado sobre cámaras digitales, por efectos de óptica la imagen dentro de una cámara es proyectada al revés, aunque en este caso la cámara está diseñada para invertir el orden de lectura, produciendo una imagen

con el primer pixel siendo el pixel superior de la izquierda tal como se almacenan en un archivo común de mapa de bits. Además de la cámara, se encuentra una tarjeta adaptadora conectada a la tarjeta FPGA que sirve para conectar los conectores GPIO de la cámara y la pantalla al bus HSMC de la tarjeta de desarrollo FPGA.

Código	Descripción	Unidades
G	Tarjeta de desarrollo de FPGA Cyclone III, Altera EP3C120F780	1
H	Tarjeta adaptadora GPIO-HSTC Terasic	1
I	Cámara CMOS de 5 Mega pixeles Terasic D5M.	1

Tabla 4.3 Elementos base cámara

Se incluye además diagramas esquemáticos de perfil (fig. 4.4a) y frente (fig. 4.4b) y fotos reales de perfil (fig. 4.5a) y de la base superior (fig. 4.5b) para explicar detalles sobre la disposición y altura de los elementos en los costados y entre las bases, no hay más elementos que declarar.

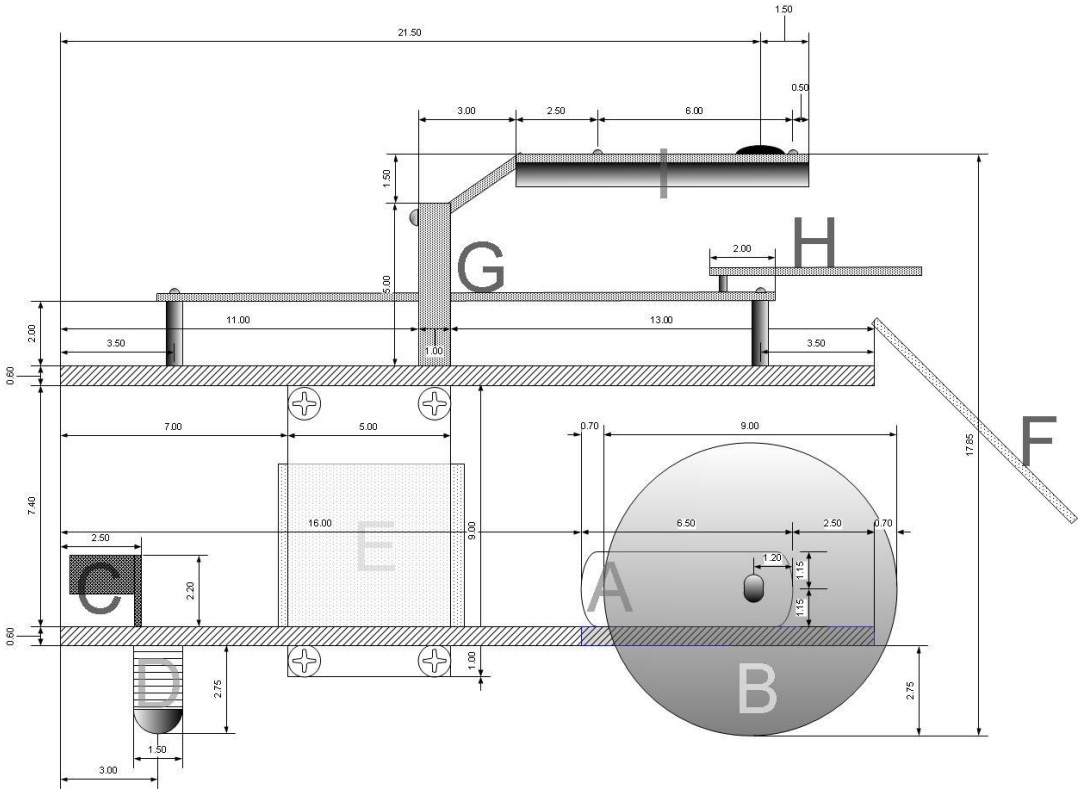


Fig. 4.4a Esquemático de perfil

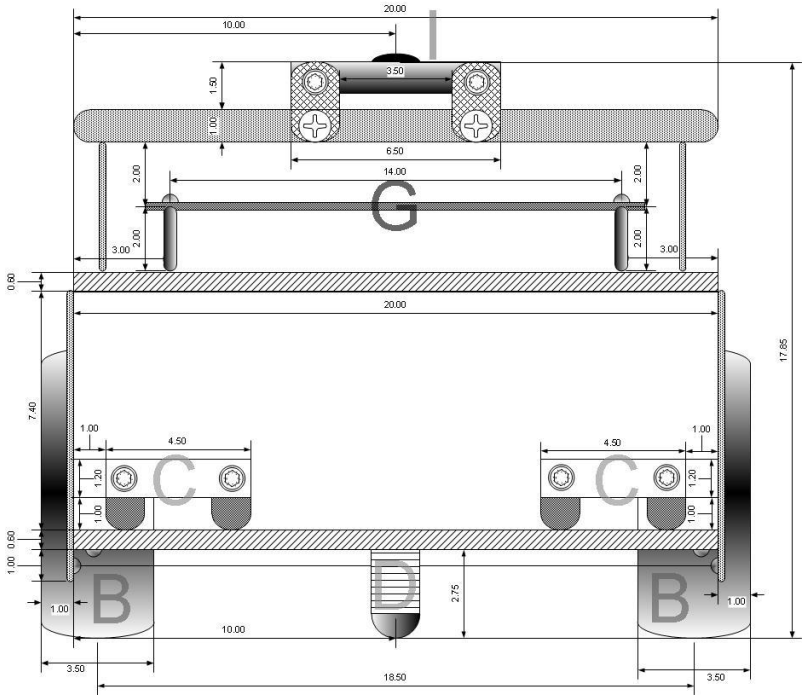


Fig. 4.4b Esquemático frontal

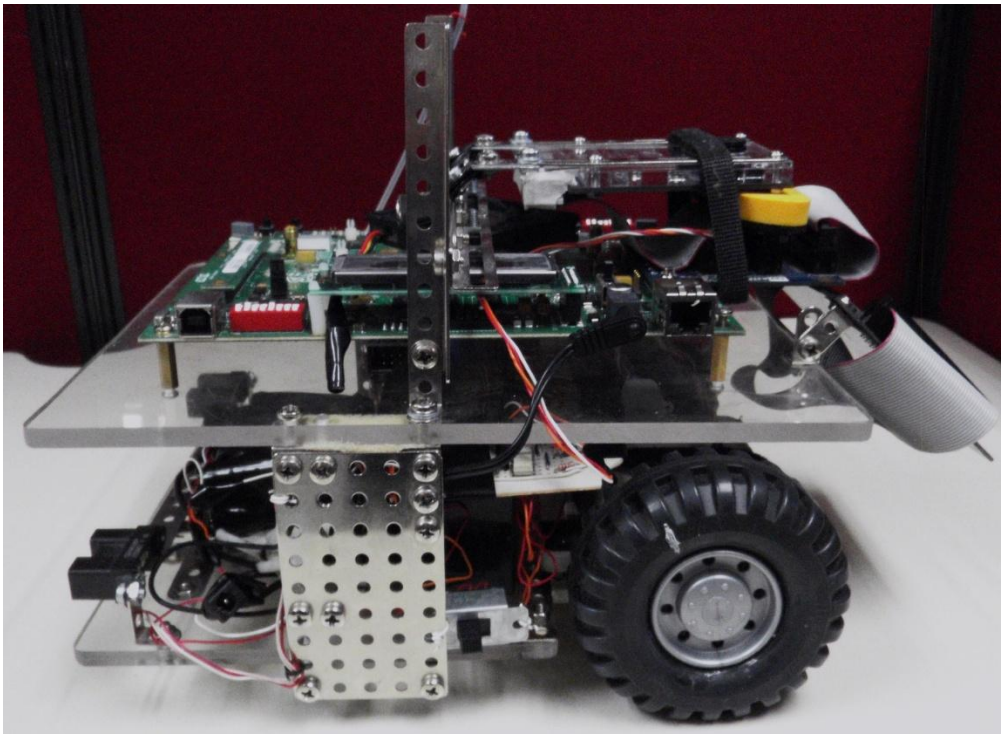


Fig. 4.5a Fotografía de perfil

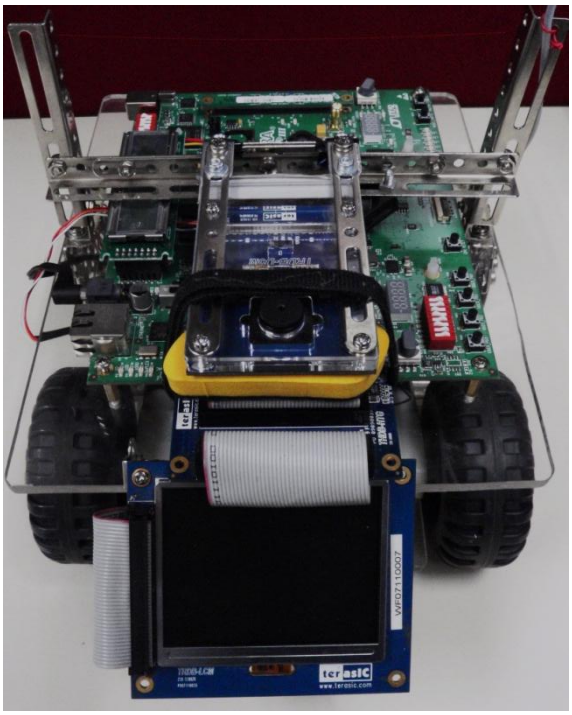


Fig. 4.5b Fotografía de base superior

II. Proveer energía eléctrica al resto de los componentes

Si dividimos la solución propuesta en este proyecto de manera eléctrica, podemos encontrar tres circuitos:

Circuito	Partes principales	Voltaje / Consumo máximo	Batería(s)
Control de dirección	Microcontrolador PIC16F887A Circuito de radiofrecuencia Sensores de distancia (2)	6 volts / 1.74 watts	1
Tracción	Motores (2)	12 volts / 1.44 watts	1,2
Tarjeta FPGA	Tarjeta de desarrollo Cyclone III Pantalla gráfica LCD Cámara CMOS	18 volts / 7.02 watts	1,2,3

Tabla 4.4 Consumo eléctrico por circuito

La primera batería es un punto débil en este acomodo; ya que todos los circuitos se alimentan de ella, provocando que su voltaje disminuya rápidamente y que los circuitos dejen de funcionar.

III. Proveer una trayectoria

A fin de permitir que el robot observe referencias que añadir al mapa, la plataforma produce movimiento. Pretende seguir una línea recta, aunque en realidad es fácilmente afectado por imperfecciones del piso y no tiene retroalimentación sobre ello por lo que tiende a desviarse. También ofrece soporte para evasión básica de obstáculos por medio de dos sensores de distancia indicados en el primer apartado (fig. 4.1:C). Si alguno de los sensores detecta un objeto a menos de 40 cm intentará evadirlo girando al lado opuesto al que se detectó. Sin embargo no se utilizaron para el experimento ya que la zona de pruebas (donde se colocaron marcas artificiales) no está limitada por paredes, por lo que el robot simplemente saldría de la zona si tratará de depender de los sensores para generar cierta trayectoria. Para generar una trayectoria específica se decidió incluir un circuito de radiofrecuencia para indicar a la plataforma que de vuelta hacia la derecha o izquierda en pequeños impulsos.

El robot como se había mencionado anteriormente, es de control diferencial y el modelo cinemático que normalmente es utilizado para describirlo está en la fig. 4.6. Cuenta con dos motores a los costados que impulsan de manera independiente a ruedas con radios r_D y r_I separadas por una distancia L. Las llantas podrían deformarse por una mala distribución del peso o cambios bruscos de dirección, haciendo que los radios y la distancia L no sean constantes. Si bien es posible modelar estas condiciones, para obtener estimaciones precisas de odometría [33] por ejemplo, como regla general si el modelo es muy complejo puede resultar peor incluirla en un filtro de Kalman [34]. En este proyecto se asume que r_D y r_I son iguales y que junto con L son constantes.

Las ecuaciones de la cinemática de este modelo están en la ec. 4.1, normalmente el ángulo ϕ se toma con respecto al eje X, sin embargo en esta propuesta resulta práctico relacionarlo con el eje Y, ya que se trabaja con imágenes de una cámara alineada a ese eje, otra opción sería asumir que el eje coordenado de la imagen esta rotado 90 °.

$$\dot{x} = v \cdot sin(\phi) \qquad v = \frac{r}{2}\omega_D + \frac{r}{2}\omega_I \tag{4.1}$$

$$\dot{y} = v \cdot cos(\phi) \qquad \omega = \frac{r}{L}\omega_I - \frac{r}{L}\omega_D$$

$$\dot{\phi} = \omega$$

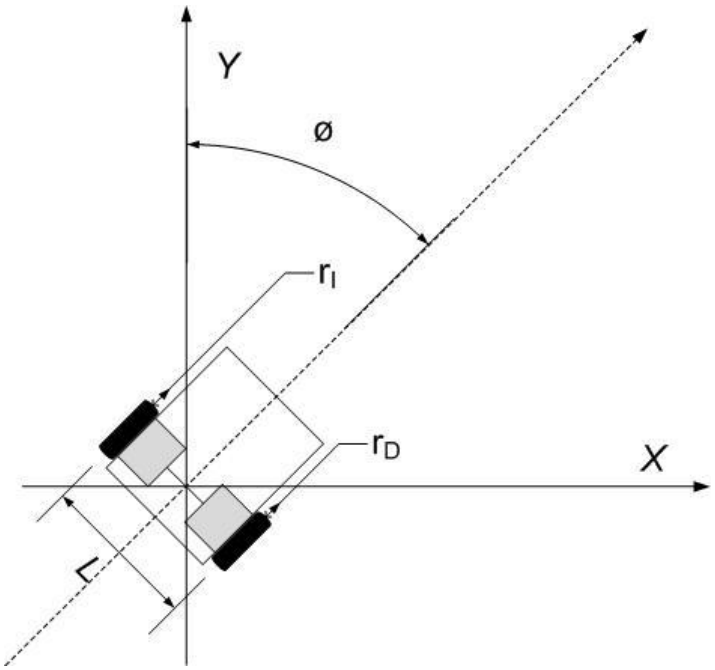


Fig. 4.6 Modelo cinemático de un robot control diferencial

La relación entre ω_I, ω_D y la reducción de potencia en los motores se detalla en la tabla 4.5, para cada uno de los posibles estados del movimiento del robot: ir hacia delante, dar vuelta (izquierda o derecha) o evadir un obstáculo cambiando la dirección hacía la dirección opuesta de donde el obstáculo se haya detectado.

Evento	Dirección	ω_I	ω_D
Delante		20%	20%
Vuelta	Derecha	15%	40%
	Izquierda	40%	15%
Evasión	Derecha	45%	12%
	Izquierda	12%	45%

Tabla 4.5 Relación entre velocidades angulares y reducción de potencia por evento.

En la tabla 4.6 se indican los datos de velocidad lineal, circulando en línea recta, velocidad lineal y angular durante una vuelta, asumiendo 12 volts.

Evento	Velocidad lineal	Velocidad angular
Delante	0.185 mts/seg	
Vuelta	0.112 mts/seg	±23 °/seg

Tabla 4.6 Velocidades lineal y angular de la plataforma

La lógica de la trayectoria está programada sobre un microprocesador PIC16F88A, instalado en una placa fenólica sobre el banco de baterías. Se detalla el circuito con la fig.4.7 y la lista de piezas relevantes que lo componen en la tabla 4.7.

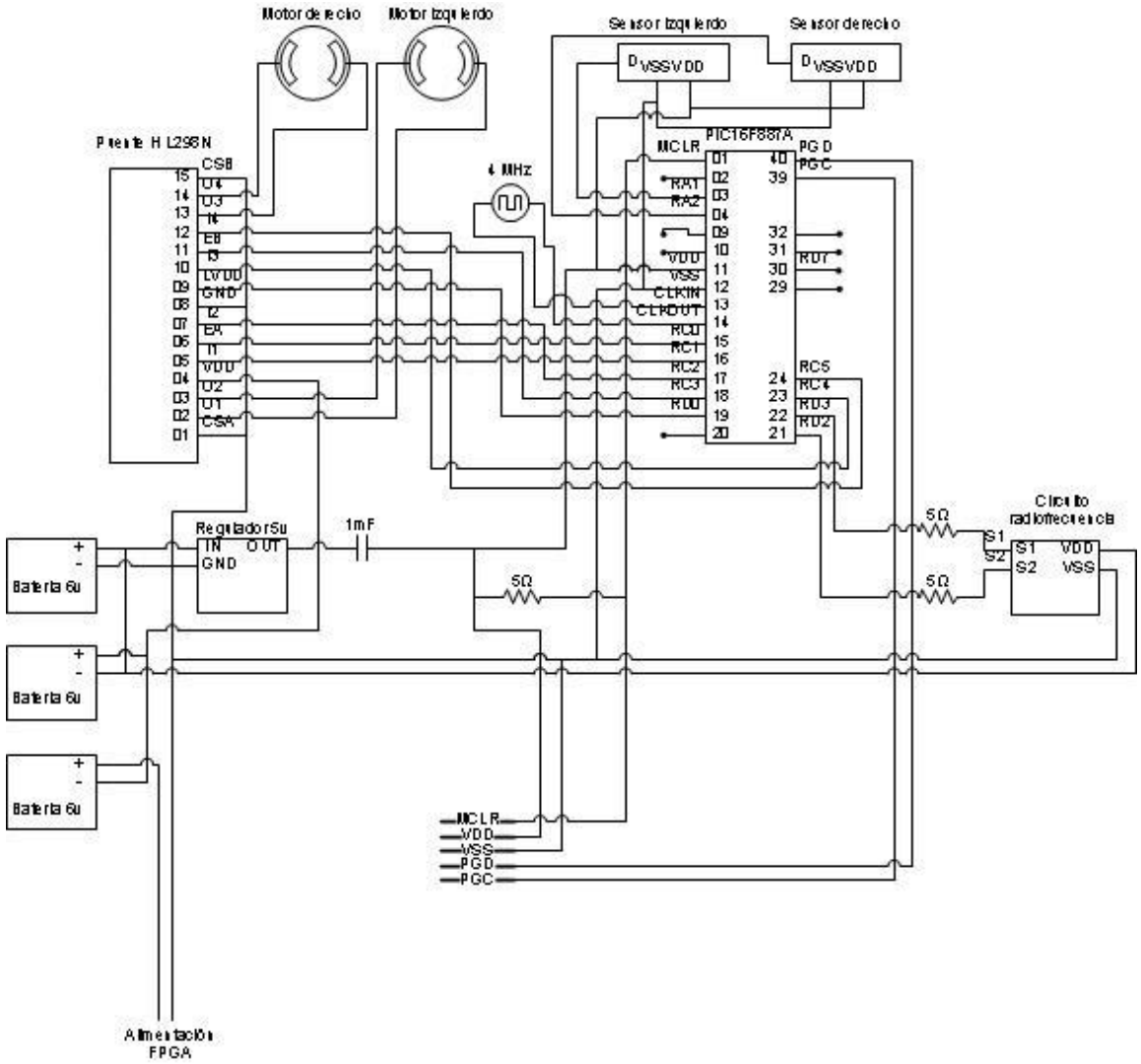


Fig. 4.7 Diagrama esquemático del circuito sobre la placa fenólica

Código	Descripción	Unidades
J	Microprocesador PIC16F887A	1
K	Puente H L298N 2 Amp	1
L	Reloj de 4 MHz	1
M	Circuito de radiofrecuencia 2 canales	1

Tabla 4.7 Elementos principales en la placa fenólica

4.2 Soporte en hardware

Antes de iniciar con la descripción del hardware diseñado, debemos describir a detalle las piezas utilizadas para ejecutar el algoritmo antes descrito (tabla 4.3). Fueron listados al inicio de este capítulo, pero solo para saber su ubicación. En esta ocasión se mencionarán algunas de sus características, las que son relevantes para este proyecto.

4.2.1 Características de componentes

Fig. 4.8 Imágenes de la tarjeta FPGA y aditamentos

Ubicación	Descripción
fig. 4.3 - G	Tarjeta de desarrollo FPGA Cyclone III, Altera EP3C120F780
fig. 4.3 – H	Tarjeta adaptadora GPIO-HSTC Terasic
fig. 4.2 – F	Pantalla gráfica LCD de 3.6', 320x240 Terasic TRDB_LCM
fig. 4.3 - I	Cámara CMOS de 5 Mega pixeles Terasic D5M.

Tabla 4.8 Ubicación de la tarjeta FPGA y aditamentos

Elementos lógicos:	119K, EP3C120F780
Memoria interna:	3888 Kbits, MK9
Memoria externa:	256Mb, DDR2 SRAM
PLL:	4
Relojes:	50Mhz y 125Mhz
Entrada / salida:	8 LED's Pantalla LCD de 2x16 Arreglo de 8 interruptores 4 Botones 2 puertos HSMC

a) Tarjeta FPGA

Conectores:	3 GPIO 1 HSMC

b) Tarjeta adaptadora GPIO-HSTC

Resolución max:	2592x1944
Formato de color:	RGB, Bayer
Tamaño de pixel:	$2.2\mu m$x$2.2\mu m$
Resolución ADC:	12bits
Max. número cuadros/seg:	15 c/s usando resolución máx.
Conector:	GPIO

c) Cámara, D5M

Resolución:	320x240
Formato de color:	RGB
Conector:	GPIO

d) Pantalla gráfica LCD, LCM

Tabla 4.9 Características de la tarjeta FPGA y aditamentos

4.2.2 Descripción del diseño en hardware

El diseño en hardware se desarrolló con la herramienta de Altera, Quartus II v10. Tiene como nodo raíz un diagrama de bloques (fig. 4.10). El diagrama lógico, jerárquico de los componentes desarrollados para este proyecto está en la fig. 4.9, en esté diagrama aparece el tipo de bloques, no las instancias.

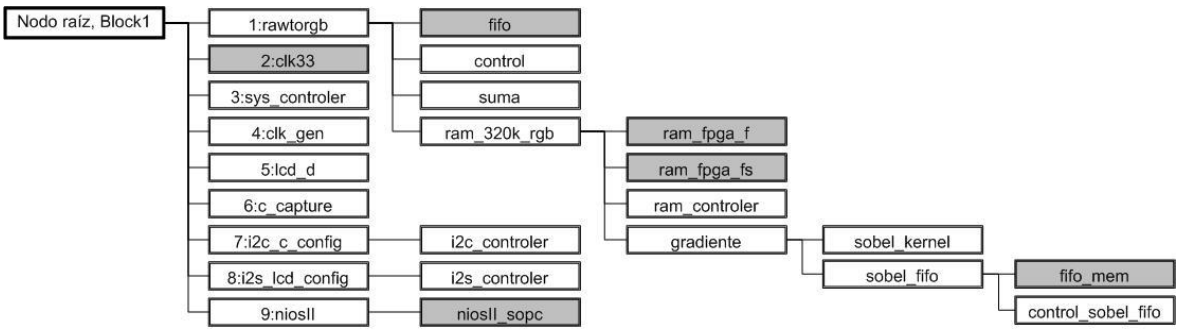


Fig. 4.9 Diagrama lógico de componentes del diseño,
los bloques obscuros indican que fueron generadas por el software de diseño.
El número al inicio de los componentes, indica el código en la fig. 4.10

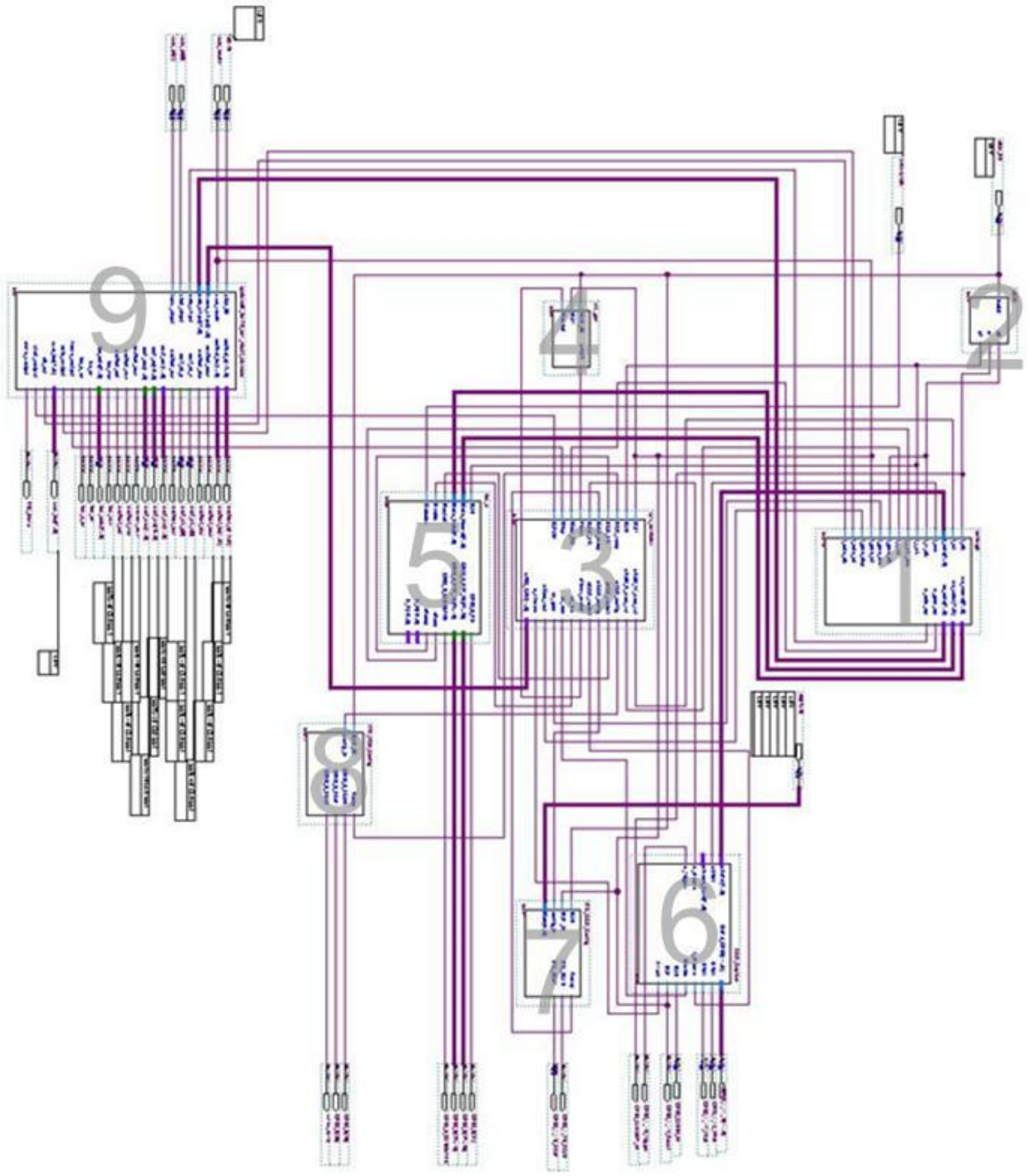


Fig. 4.10 Diagrama de bloque, nodo raíz del diseño

Iniciemos la descripción del diseño por el diagrama de flujo del componente “sys_controler” (fig. 4.11), éste coordina la mayor parte de las actividades, regula el ciclo de captura de cuadros para permitir que la capa que hace uso de las imágenes pueda accederlas.

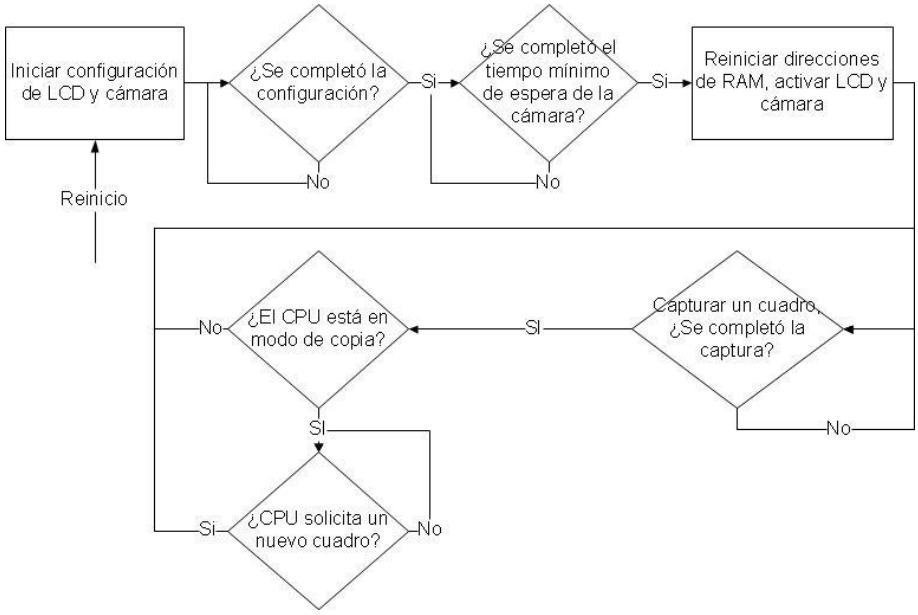


Fig. 4.11 Diagrama flujo del componente “sys_controler”

Dos puntos importantes sobre este flujo: el componente “clk_gen” indica un cierto tiempo que se debe esperar antes de intentar procesar un cuadro, ya que al encender la cámara y durante cierto tiempo el elemento fotosensible no está activo y la información es simple ruido; El otro punto es que si el procesador no está en modo captura, la cámara capturará cuadros en forma libre y serán mostrados en la pantalla gráfica LCD, por lo que se puede hacer pruebas de la calidad de la imagen en ese punto, sin involucrar al resto de los componentes.

Continuemos con la configuración de la cámara y la pantalla LCD, bloques “i2c_c_config” y “i2s_lcd_config” respectivamente (fig. 4.9:7 y 4.9:8). Estos bloques funcionan bajo el protocolo I^2C. Una explicación detallada del funcionamiento del protocolo esta fuera del alcance de esta tesis, pero una descripción básica de su uso nos será útil para comprender la configuración específica que se utiliza.

El protocolo I^2C funciona sobre dos señales: un bus serial SDATA y un reloj SCLCK. Sobre el canal de datos (SDATA) se envía una cadena de pares: registro y valor. El registro tiene 8 bits, pero el último indica si es una lectura o escritura, por lo que solo se tienen 127 posibles direcciones. En el caso particular de esta

cámara y pantalla LCD el dato es de 16 bits por lo que se usan dos ciclos de transferencia de 8 bits. La función específica de cada registro así como la interpretación de su valor es definido por el fabricante. Por ejemplo para indicar el tiempo de exposición a la cámara, se debe escribir en el registro Rx09 el número de ciclos de reloj que la cámara debe mantener expuesto el elemento fotosensible. El registro Rx08 indica la parte alta del mismo dato; es decir la cámara tiene un registro de 32bits para definir el tiempo de exposición, la parte alta 31:16 en el registro Rx08 y la parte baja 15:0 en el registro Rx09. Puesto que el número de datos y los datos a enviar son distintos se utilizan compontes distintos: "i2s_controller" para la cámara y "i2c_controller" para el LCD (tabla 4.10 y 4.11).

Registro		Valor	
Uso	Dirección (8 bits)	Dato (16 bits)	Efecto
Fila inicial	0x01	0x0042	Comenzar por la fila 66
Columna inicial	0x02	0x0020	Comenzar por la columna 32
Número de filas	0x03	0x077F	Leer 1920 filas
Número de columnas	0x04	0x09FF	Leer 2560 columnas
Filas a saltar (muesteo)	0x22	0x0003	Leer 1 de cada 4 filas
Columnas a saltar (muesteo)	0x23	0x0003	Leer 1 de cada 4 columnas
Ganancia a verde1	0x2B	0x01XX	Ganancia análoga en los tres colores Los 5 bits menos significativos son configurables vía interruptores de la tarjeta (el color verde tiene controles distintos por cada línea)
Ganancia a azul	0x2C	0x01XX	
Ganancia a rojo	0x2D	0x01XX	
Ganancia a verde2	0x2E	0x01XX	
Modo de Lectura	0x20	0xC000	Leer en modo normal filas y columnas
Tiempo de exposición	0x09	0x0300	Espera 0x300 ciclos
Control de PLL interno	0x10	0x0000	Apagar PLL, usar reloj externo

Tabla 4.10 Configuración de la cámara

Registro		Valor	
Uso	Dirección (8 bits)	Dato (16 bits)	Efecto
Formato de datos	0x02	0x0002	QVGA, RGBDummy (320x3x240)

Tabla 4.11 Configuración de la pantalla LCD

I. Captura de un cuadro

Es importante detallar el procedimiento de recuperación de una imagen, describiendo el proceso que ocurre desde que se captura por el elemento fotosensible hasta que se almacena en la memoria interna del FPGA. Como se detalló en la configuración de la cámara (tabla 4.10), la imagen se solicita con una dimensión de 2560x1920pixeles pero iniciando desde el pixel (32,66) con el objetivo de centrarla. Existen tres tipos de celdas en el sensor, negras, borde y de imagen (fig. 4.12), las negras siempre tienen un valor en 0, las celdas del borde pueden ser leídas para obtener algún valor predeterminado (en negro por omisión), por lo que deben considerarse al momento de centrar la imagen; además se indica que únicamente se lea 1 pixel por cada 4, lo cual nos da una imagen con 98.5% del rango visible de la cámara, pero mucho más pequeña 640x480.

La imagen proveniente de la cámara está en formato de color Bayer. En este formato se cuenta con los 3 colores de RGB, pero en cada fila el color verde alterna pixeles con el color rojo o azul, únicamente uno de los dos, en la siguiente línea el color restante se alterna con el color verde, dando una distribución de 50% verde, 25% de rojo y 25% de azul (fig. 4.18).

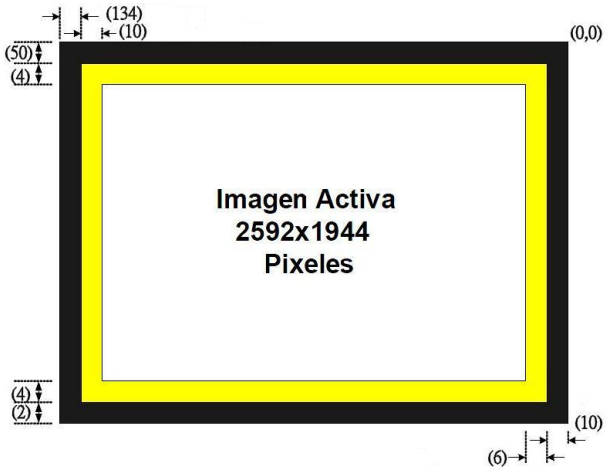


Fig. 4.12 Distribución de pixeles en la cámara

G1	R	G1	R	G1	R	G1	R	G1
B	G2	B	G2	B	G2	B	G2	B
G1	R	G1	R	G1	R	G1	R	G1
B	G2	B	G2	B	G2	B	G2	B
G1	R	G1	R	G1	R	G1	R	G1
B	G2	B	G2	B	G2	B	G2	B

Fig. 4.13 Formato Bayer

Aun cuando las marcas estáticas negras sobre fondo blanco podrían ser detectadas con cualquiera de los colores del formato de Bayer directamente, se

prefirió dejar abierta la posibilidad de que las marcas fueran de cualquier color, por lo que se incluyó un proceso para transformar los pixeles a RGB y de ahí a escala de grises. Si bien es posible hacer un proceso de interpolación para mantener la resolución original, a cambio de calidad, para los fines de este proyecto es preferible manejar una imagen más pequeña, por lo que cada 4 pixeles (dos por fila) del formato de Bayer, se extraen los componentes RGB, los dos verdes se promedian, mientras que azul y rojo se toman directamente. Lo cual produce una imagen de la mitad de resolución: 320x240, si bien se calculan los componentes de cada color, éstos nunca se almacenan.

De la resolución original de 12 bits sólo se toman los 8 bits más representativos 11:4, los colores rojo y azul se dividen en factores iguales de 12.5% y cada verde entre un factor de 25%, para conformar el valor de cada pixel de la imagen en escala de grises (fig. 4.10). La pérdida de intensidad se compensa con la ganancia configurada vía los interruptores de la tarjeta FPGA (Tabla 4.10). En general es posible obtener imágenes de calidad aceptable (para este proyecto), tanto en una zona iluminada con lámparas o la luz del día por la tarde.

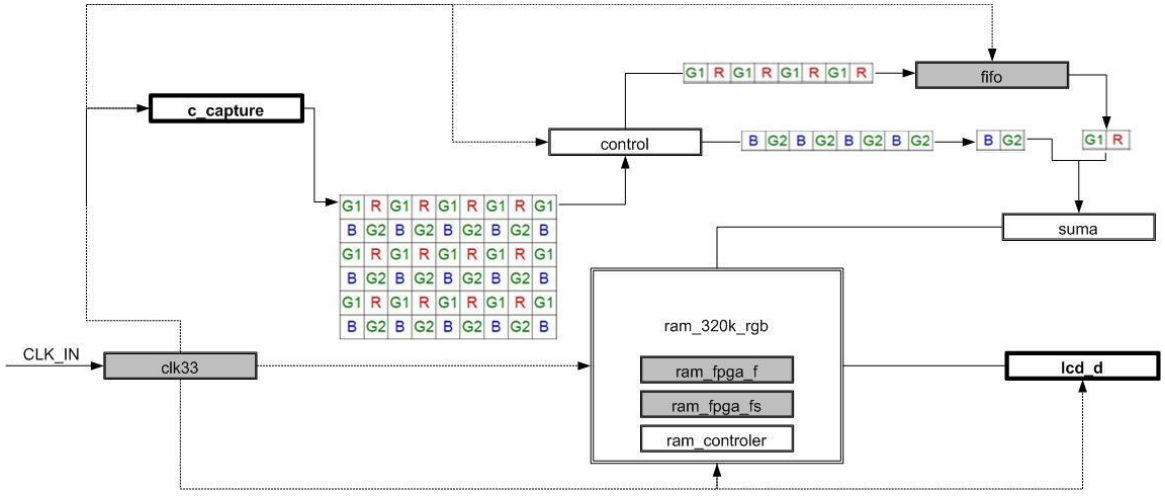


Fig. 4.14 Diagrama de operaciones para captura de un cuadro

A grandes rasgos la relación de los componentes se describe gráficamente en la fig. 4.14. El componente “c_capture” toma los datos de los pines de la cámara, pixel a pixel en cada ciclo de reloj, leyendo todas las columnas de cada fila hasta

completar la imagen. El reloj de 60 Mhz que se utiliza para este fin se seleccionó de acuerdo con la configuración de la cámara para obtener imágenes con suficiente contraste y nitidez en la presencia de movimiento para ser útiles a este proyecto. El componente “clk33” es una *megafunción* de Altera, un macro que genera un componente configurable de uso común, como una RAM o un PLL, en este caso es un PLL con tres salidas, la primera utilizada por la cámara.

Supervisado por el componente “control”; las filas impares de la imagen se almacenan en una cola, implementada por el componente “fifo”, las filas pares pasan directamente al componente “suma” que realiza el método descrito anteriormente para generar un pixel en escala de grises (cada dos filas, cada dos columnas).

El dato resultante es almacenado en memoria interna del FPGA utilizando dos componentes generados: ram_fpga_f de 2^{16} bytes y ram_fgpa_fs de 2^{14} bytes en donde se almacena la imagen; la lógica para escribir y leer los datos de la imagen almacenada se encuentra en “ram_controler”. Existen cuatro instancias de estos componentes; es decir existen cuatro arreglos para almacenar la imagen, todos ellos controlados por “ram_controler”. La imagen preveniente de “control” se escribe en tres de ellos al mismo tiempo.

Los componentes descritos anteriormente utilizan dos relojes distintos, como puede observarse, al componente “suma”, se le entregan 4 datos, pero esto ocurre en dos ciclos de reloj, recordemos que la imagen es leída pixel a pixel, así que por cada dos ciclos de reloj que se usan para obtener datos de la cámara se almacena uno en escala de grises, una relación de 2:1, por lo que la segunda salida de “clk33”, es un reloj de 30 Mhz.

Finalmente la imagen es leída por la pantalla LCD, utilizando la tercera salida de “clk33” a 40 Mhz, puesto que el LCD espera los tres colores se envía el mismo valor tres veces, generando una imagen en escala de grises en la pantalla LCD.

II. Administración de memoria

Una vez que la imagen se almacena en la memoria interna del FPGA, ésta puede ser transferida al CPU para que sea procesada. Como se mencionó en el párrafo anterior existen cuatro arreglos de memoria para almacenar una imagen:

a. En el primer arreglo se escribe la salida del componente “control” (la imagen en escala de grises), y es leído por la pantalla LCD, que tiene ciertos requerimientos que no la hacen compatible con el resto.
b. El segundo arreglo se utiliza para que un módulo de pre procesamiento tenga acceso a la imagen en escala de grises.
c. El tercer arreglo se utiliza para que la salida del módulo de pre procesamiento se muestre en la pantalla LCD, el LCD podría cambiar la fuente entre el primero y tercer arreglo vía un interruptor.
d. El cuarto arreglo se utilizaría para que la salida del módulo de pre procesamiento se envíe al procesador, sin embargo para los alcances de esta tesis, el pre procesamiento se efectúa en software y por lo tanto, la imagen en escala de grises también se copia en este arreglo.

El componente “gradiente” que aparece en los diagramas, no será explicado en este capítulo pero se menciona como anexo.

III. Transferencia al procesador

El producto de diseño de hardware de Altera, Quartus II incluye la herramienta *SOPC Builder*, con la se puede crear sistemas computacionales completos al incorporar módulos pre diseñados (procesadores, memoria, controladores de hardware, etc.) e interconectarlos. La herramienta genera código VHDL o Verilog para seguir con el flujo normal de desarrollo en FPGA. El diseño para este proyecto toma los siguientes elementos dicha herramienta*:*

- Procesador Nios II completo, con unidad de punto flotante de 32 bits (IEEE 754-1985)
- Controlador DDR2

- Controlador de pantalla de texto LCD 2x16
- JTAG, conexión entre PC y tarjeta FPGA
- Múltiples puertos paralelos de entrada y salida

La definición de hardware generada por la herramienta es usada por el producto de diseño de software Nios II IDE, un compilador de lenguaje C para el procesador Nios II. Bibliotecas de sistema son ajustadas y compiladas al diseño creado en *SOPC Builder*, por ejemplo si se incluye el controlador para la pantalla LCD de texto de 2x16, soporte para este se incluye en los archivos a compilar. Es sobre este mismo producto que se programa la lógica para la administración del mapa.

El procesador vía código en lenguaje C, puede comunicarse con el resto del diseño en hardware para:

- Arrancar o detener el movimiento del robot.
- Solicitar un nuevo cuadro.
- Transferir una imagen.

Estas operaciones utilizan puertos paralelos de entrada y salida, y en el caso de los últimos dos usos se utiliza un protocolo simple pero deficiente (fig. 4.15) para permitir que el procesador efectúe una secuencia de solicitudes.

En el caso de la transferencia de la imagen se aumenta considerablemente el tiempo que pasa entre cuadro y cuadro limitando la tasa de cuadros por segundo que pueden ser tomados, por lo que se utiliza un bus de 128 bits. Se pueden tomar aproximadamente 6 cuadros por segundo (sin considerar el tiempo de procesamiento de cada imagen ya que no es constante.)

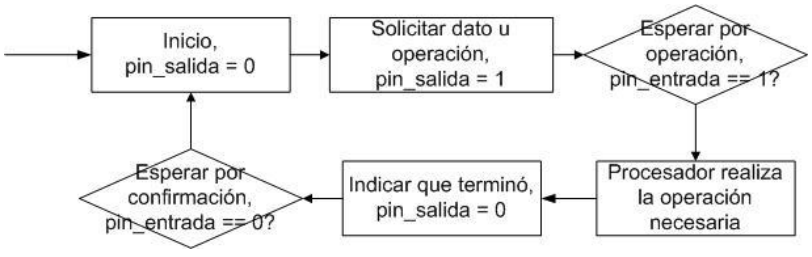


Fig. 4.15 Protocolo de comunicación del procesador

Además de estos usos, el código en C puede leer dos botones de la tarjeta FPGA. El primero es para iniciar y el segundo es para imprimir los resultados o transferir las imágenes almacenadas (que se presentan en este documento) ya que para este fin se requiere que la tarjeta FPGA esté conectada a la PC.

4.3 Soporte en software

Una vez detallada la mayoría de los detalles físicos: forma y acomodo de los elementos del robot; así como los detalles eléctricos y mecánicos de la plataforma móvil para generar movimiento y aunque menos interesante pero vital, la distribución de energía eléctrica al robot, podemos entrar al tema principal, el mapa.

4.3.1 Administración del mapa

El mapa es un arreglo o lista de tamaño variable con las referencias que han sido observadas hasta el momento. Las referencias en cualquier proyecto de SLAM tiene al menos sus coordenadas, y usualmente en el caso de proyectos específicos de Visual SLAM rasgos visuales para identificar a las marcas entre sí, como: parches de la imagen, medidas de similitud basadas en la intensidad como NCC o descriptores de rasgos visuales como KLT o SIFT mencionados en el capítulo de fundamentos. Sin embargo esta implementación trabaja con marcas artificiales idénticas entre sí, incluso en su orientación. Las marcas están en un plano paralelo al que circula el robot (el techo), por lo que cada referencia solo requiere almacenar su coordenada *(x, y).*

El primer nodo del mapa es el robot mismo, recordemos que además de crear un mapa, el otro objetivo de SLAM es la autolocalización dentro de éste. Si bien el robot no es una marca ni mucho menos es estático, las mediciones a las marcas siempre son relativas a la posición actual del robot. El nodo del robot tiene además orientación, las mediciones además de trasladadas pueden estar rotadas. Para fines de evaluación se almacena el histórico de su posición y orientación.

Las marcas son objetos puntuales, es decir no tienen área, únicamente sus coordenadas, obtenidas a partir de su centro geométrico. El robot no es la excepción en ese sentido, el centro de la imagen, a donde apunta el centro óptico, es el punto que se toma como la posición del robot. Cuando el robot arranca, el mapa no tiene ninguna marca, únicamente el nodo que lo representa.

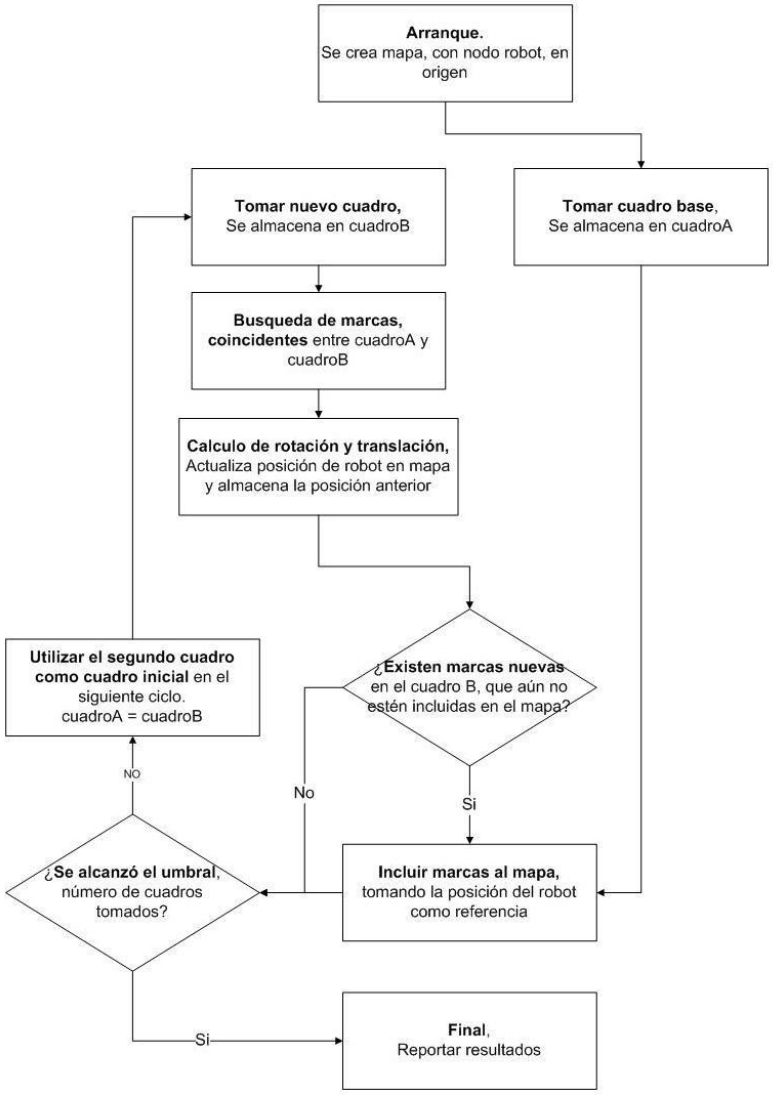


Fig. 4.16 Flujo del proceso de construcción del mapa

En la fig. 4.16 se presenta el diagrama de flujo del proceso completo que sigue esta implementación, es importante recalcar que mientras se ejecuta este flujo la plataforma está moviéndose.

I. Arranque

Como se mencionó, al inicio el mapa no tiene marcas, únicamente existe el nodo del robot y este inicia con coordenadas *(0,0)* y una orientación de *0* ° (con respecto a *Y.*) El punto donde toma la primer imagen se vuelve el origen y la orientación del robot es la orientación con la que se toman los ejes *X* y *Y*.

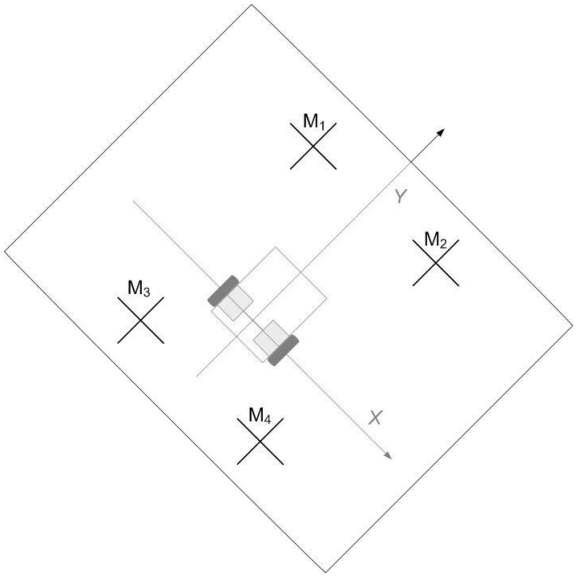


Fig. 4.17a Vista a través del techo del arranque

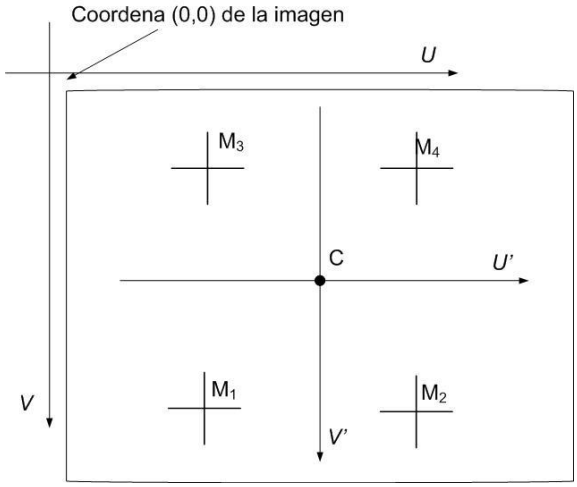


Fig. 4.17b Imagen resultante al arranque

Antes de explicar la técnica para extraer la posición de las referencias, se deben hacer notar dos puntos importantes en la imagen de la fig. 4.17b: Si se considera el sistema común de coordenadas de una imagen donde el origen *(0,0)* es la esquina superior izquierda y *V* avanza hacia abajo, parece tener sentido que las marcas al frente estén debajo, pero no que las marcas que estaban detrás del robot aparezcan delante del origen de la imagen. Si se considera el punto *C* como el origen, las marcas parecen estar invertidas, las cosas al frente se ven en la parte inferior, normalmente un cuadrante negativo por lo que se propone cambiar el sentido del eje *V'* haciéndolo coincidir con la posición de la cámara. En ocasiones se requiere pasar de un sistema de coordenadas a otro y puede resultar confuso; El segundo punto es que una secuencia de imágenes no puede ofrecer datos absolutos sobre la dirección hacia la que circula un robot, sólo cambios

relativos entre cuadros, por ejemplo si el robot se mueve hacía *Y*+ (fig. 4.18a) en la secuencia de imágenes parecería que las marcas se mueven de abajo hacia arriba y si el robot se moviese hacia *Y*- (fig. 4.18b) el efecto en las imágenes sería exactamente el mismo (fig. 4.18a y fig. 4.18d), por supuesto las marcas irían pasando en forma inversa pero al menos en esta etapa es imposible distinguir entre ellas.

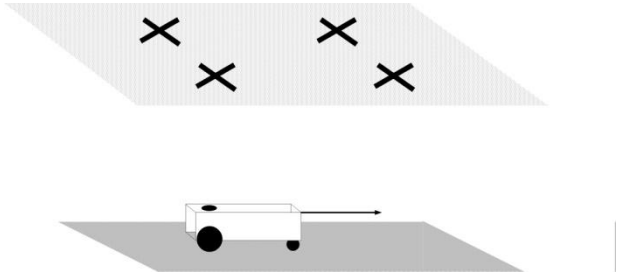


Fig. 4.18a Robot moviéndose hacia *Y*+

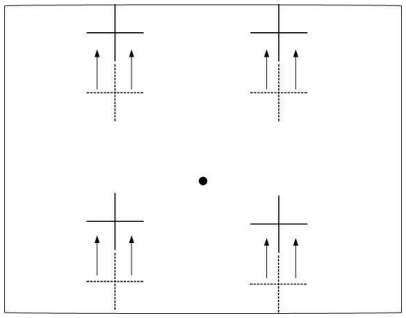


Fig. 4.18b Movimiento de marcas cuando el robot se mueve hacia *Y*+

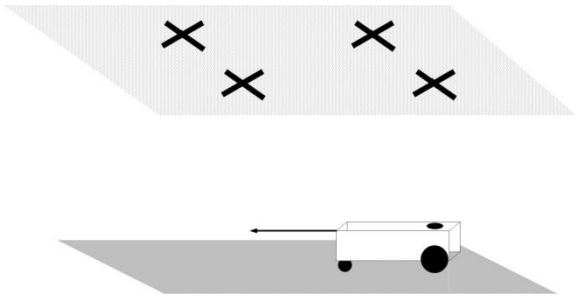


Fig. 4.18c Robot moviéndose hacia *Y*-

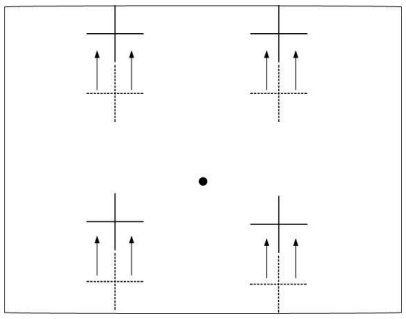


Fig. 4.18d Movimiento de marcas cuando el robot se mueve hacia *Y*-

II. Extraer información de una imagen

Dentro del flujo existen dos pasos referentes a la captura un cuadro, son iguales, pero el flujo hace la distinción. Al arranque no hay una imagen anterior con que comparar, así que sin calcular cambio de posición y orientación del robot se añaden las marcas al mapa. En el siguiente ciclo el cuadro anterior es reusado para comparar contra el más reciente y calcular el cambio de posición y orientación del robot antes de agregar nuevas marcas, en caso de que existan.

Utilizando la imagen de 320x240 pixeles en escala de grises obtenida en la sección 4.2.2 y transferida a la memoria a la que tiene acceso el procesador, se inicia el procesamiento de la imagen.

Umbralado, sacando provecho del contraste que tienen las marcas artificiales, cruces negras de 20 cm x 20 cm x 1.8 cm sobre un fondo blanco, el techo, se utiliza un simple umbralado para extraerlas, el umbral propuesto es dinámico ya que dentro de la zona donde se realizan pruebas la cantidad de luz varía, pero la relación entre el número de pixeles de fondo y el número de pixeles de objetos es mucho mayor y más o menos constante, por lo que se propone utilizar un factor (obtenido de manera empírica) del promedio de la intensidad de los pixeles de la imagen, todo pixel con una intensidad menor al umbral es considerado parte de un objeto.

$$p = \sum_{j=0}^{j=alto} \sum_{i=0}^{i=ancho} f(j,i) \tag{4.2}$$

$$Umbral = \frac{9 \cdot p}{alto \cdot ancho \cdot 16}$$

Etiquetado, luego de umbralar la imagen, se etiquetan las regiones conectadas en una vecindad de 4 pixeles usando una tabla de equivalencias. El proceso recorre todos los pixeles de la imagen. Si el pixel pertenece a un objeto $f(u,v) \neq 0$. Se le asigna una etiqueta, en caso de que exista un pixel en la fila inmediata superior y este sea parte de un objeto, se reúsa su etiqueta, lo mismo ocurre con el pixel de la columna anterior, sin embargo si ambos existen y pertenecen a objetos distintos (etiquetas distintas) se debe crear una entrada en la tabla de equivalencias con ambas etiquetas y utilizar cualquiera de las dos, finalmente si no existe ninguna etiqueta en los pixeles de la fila o columna anterior se asigna una etiqueta nueva.

Fig. 4.19a Zona de pruebas

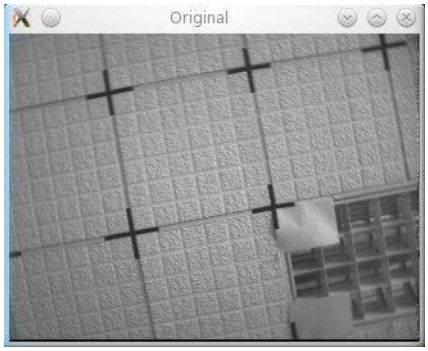


Fig. 4.19b Imagen original

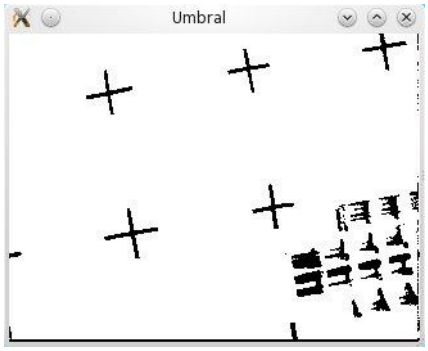


Fig. 4.19c Imagen umbralada

Creación de marcas, se vuelve a recorrer la imagen completa, pero esta vez haciendo uso de la tabla de equivalencias y las etiquetas asignadas, con ellas se calculan área y centro geométricos de cada región, las coordenadas de la imagen *U* y *V* se transforman al sistema de coordenadas *U'* y *V'* (tomando a *C* como origen (fig.4.17b) usando la ec. 4.3. Se filtran las regiones que no pueden pertenecer a las marcas artificiales usando el área, sólo regiones entre 100 y 400 pixeles son tomadas en cuenta. Finalmente se construyen las estructuras de datos para almacenar las referencias, y éstas a su vez se ligan en un mapa temporal: *cuadroA* para referirse al cuadro anterior y *cuadroB* para referirse al cuadro actual, como se explicó al inicio del apartado. Se utiliza el identificador de la región como identificador de la marca.

$$Mt_x^i = r_x^i - \frac{ancho}{2}$$
$$Mt_y^i = r_y^i - \frac{alto}{2} \quad (4.3)$$

(Mt_x^i, Mt_y^i), Coordenadas de la marca *i* relativa a la posición actual del robot

(r_x^i, r_y^i) Coordenadas de la región *i* en la imagen

$(ancho, alto)$ Resolución en pixeles de la imagen (320x240)

III. **Añadir marcas al mapa general**

En esta etapa se trabaja con el mapa temporal que agrupa a las marcas detectadas en la imagen procesada del paso anterior, este mapa es en sí mismo un sistema de coordenadas (fig. 4.17b) pero esta rotado y trasladado con respecto al origen que arbitrariamente se creó al arranque (fig. 4.20). Si este mapa temporal proviene del arranque estará trasladado *(0,0)* y rotado *0* °. Para hacer la incorporación de las marcas se utilizan las siguientes ecuaciones:

$$M_x^i = R_x + Mt_x^i \cdot cos(R_\emptyset) - Mt_y^i \cdot sen(R_\emptyset)$$
$$M_y^i = R_y + Mt_x^i \cdot sen(R_\emptyset) + Mt_y^i \cdot cos(R_\emptyset) \quad (4.4)$$

(M_x^i, M_y^i), Coordenadas de la marca *i* en el mapa general

$(R_x, R_y, R_\emptyset)$ Posición y orientación del robot respecto al mapa general

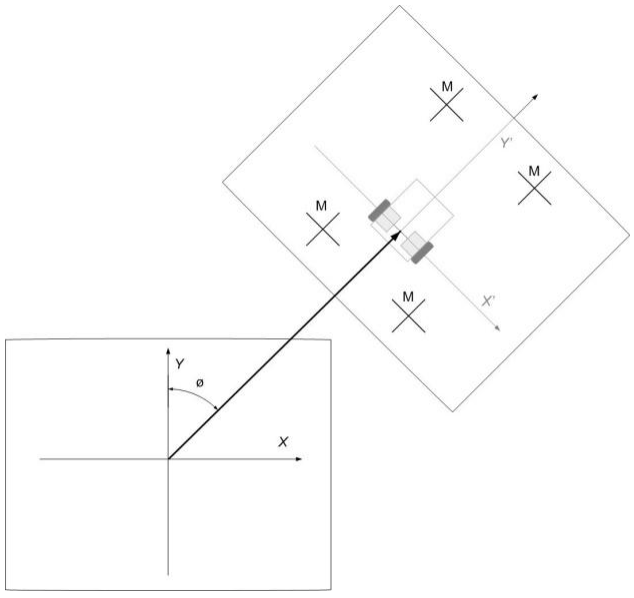


Fig. 4.20 Rotación y traslación de ejes

Sólo se añaden marcas nuevas, que estén presentes en el nuevo cuadro, pero sin correspondencia en el cuadro inicial o las provenientes del arranque. Sin embrago, es posible que se esté revisitando una marca que salió del rango de visión. Por lo tanto este módulo es también responsable de decidir si se trata de una marca revisitada y puesto que no existe un parámetro, como un rasgo visual con el cual comparar, se utiliza únicamente la distancia euclidiana para decidir. Si la marca está a una distancia menor de 20 pixeles (alrededor de 10 cm. en el plano), se asume que es la misma, en caso contrario asigna un identificador único a cada marca nueva. En cada ciclo verifica si la condición de paro (un número máximo de cuadros a procesar) se ha cumplido.

Una marca en esta implementación mide 24 bytes y contiene:

- 2 Números flotantes de 4 bytes.
- 3 Enteros de 4 bytes.
- 1 Apuntador a la siguiente marca de 4 bytes.

Despreciando el espacio que se requiere para el propio código de administración del mapa, el espacio para las estructuras de datos del mapa general, mapas temporales del *cuadroA* y *cuadroB* y el nodo del robot (que en conjunto no superan los 5 Mb y asumiendo que no se almacenan las imágenes luego de ser procesadas) se podrían almacenar alrededor de 2.5 millones de marcas en los 64 Mb disponibles que se tienen. Sin embargo de implementarse un filtro probabilístico como el filtro extendido de Kalman con una complejidad de $O(n^3)$, no podría trabajar con más de 100 marcas sin afectar la tasa de cuadros por segundo necesaria para evitar errores de asociación.

IV. Búsqueda de marcas coincidentes

Este proceso asocia una marca detectada en la imagen anterior (y que por lo tanto existe en el mapa global) con otra marca de la imagen actual; es decir lleva a cabo una búsqueda activa de las marcas cuadro a cuadro. Trabaja con dos mapas temporales, referenciados en el diagrama de flujo como *cuadroA* y *cuadroB* (fig.

4.16). En cada ciclo se reúsa el cuadro anterior *cuadroA=cuadroB*, se toma una nueva imagen (desde una posición distinta), se procesa y el mapa temporal resultante se almacena como *cuadroB*.

Como se explicó anteriormente las marcas son idénticas entre sí, y puesto que están alineadas con el acomodo del techo falso incluso su orientación es igual, sin embargo la distancia entre las marcas artificiales es de al menos 60 cm por lo que es posible utilizar un umbral de distancia con respecto al cuadro anterior para decidir si se trata de la misma marca. Se definió un umbral de 50 pixeles, (alrededor de 25 cm. en el plano) para decidir si una marca es la misma que la detectada anteriormente, en caso de que existan varias opciones dentro del radio de búsqueda, se opta por la más cercana. Las marcas en el *cuadroB* que no hayan sido relacionadas con marcas del *cuadroA* se clasifican como nuevas.

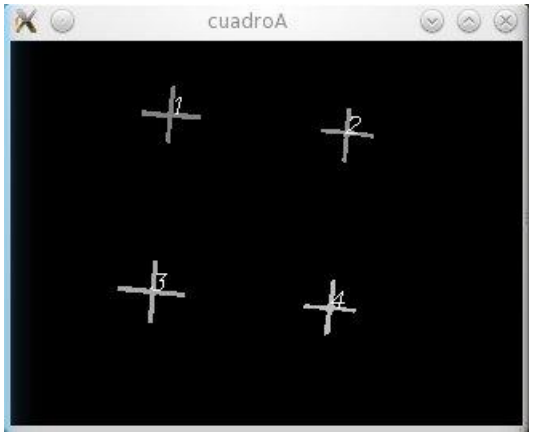


Fig. 4.21a Búsqueda activa de marcas, *cuadroA*

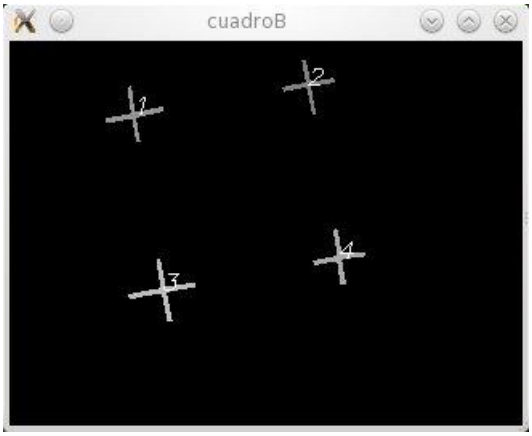


Fig. 4.21b Búsqueda activa de marcas, *cuadroB*

Este método es confiable siempre y cuando el cambio entre imágenes sea pequeño, la relación de la velocidad a la que se desplaza el robot y la frecuencia con que toma y procesa una imagen determina el éxito de este método.

V. Calculo de rotación y traslación

Come se puede ver en las imágenes de búsqueda activa de marcas entre el *cuadroA* y *cuadroB* (fig. 4.21a y 4.21b) existe un cambio de traslación y de rotación. Para determinarlos se propone utilizar parejas de marcas con coincidencia entre los dos cuadros, por ejemplo se selecciona la pareja de marcas

(M_1, M_2), transformamos a la primer marca M_1 en el origen y calculamos la dirección a la que se encuentra la segunda marca M_2, esto se calcula en ambos cuadros, el orden no es importante, pero sí que se use el mismo en ambos cuadros, finalmente se calcula la diferencia como se muestra en la ec. 4.5.

$$\mathrm{A}_{\mathrm{x}}^{\mathrm{i:j}} = MtA_x^j - MtA_x^i \qquad \mathrm{B}_{\mathrm{x}}^{\mathrm{i:j}} = MtB_x^j - MtB_x^i$$
$$\mathrm{A}_{\mathrm{y}}^{\mathrm{i:j}} = MtA_y^j - MtA_y^i \qquad \mathrm{B}_{\mathrm{y}}^{\mathrm{i:j}} = MtB_y^j - MtB_y^i \tag{4.5}$$
$$\mathrm{A}_{\phi}^{\mathrm{i:j}} = arctan\left(\frac{\mathrm{A}_{\mathrm{x}}^{\mathrm{i:j}}}{\mathrm{A}_{\mathrm{y}}^{\mathrm{i:j}}}\right) \qquad \mathrm{B}_{\phi}^{\mathrm{i:j}} = arctan\left(\frac{\mathrm{BA}_{\mathrm{x}}^{\mathrm{i:j}}}{\mathrm{A}_{\mathrm{y}}^{\mathrm{i:j}}}\right)$$
$$\Delta_{\phi}^{\mathrm{i:j}} = \mathrm{B}_{\phi}^{\mathrm{i:j}} - \mathrm{A}_{\phi}^{\mathrm{i:j}}$$

(MtA_x^k, MtA_y^k) Coordenadas de M_k relativa a la posición actual del robot, en *cuadroA*

(MtB_x^k, MtB_y^k) Coordenadas de M_k relativa a la posición actual del robot, en *cuadroB*

$\mathrm{A}_{\phi}^{\mathrm{i:j}}, \mathrm{B}_{\phi}^{\mathrm{i:j}}, \Delta_{\phi}^{\mathrm{i:j}}$ Ángulo entre marcas M_i y M_j en *cuadroA*, *cuadroB* y la diferencia entre ellos.

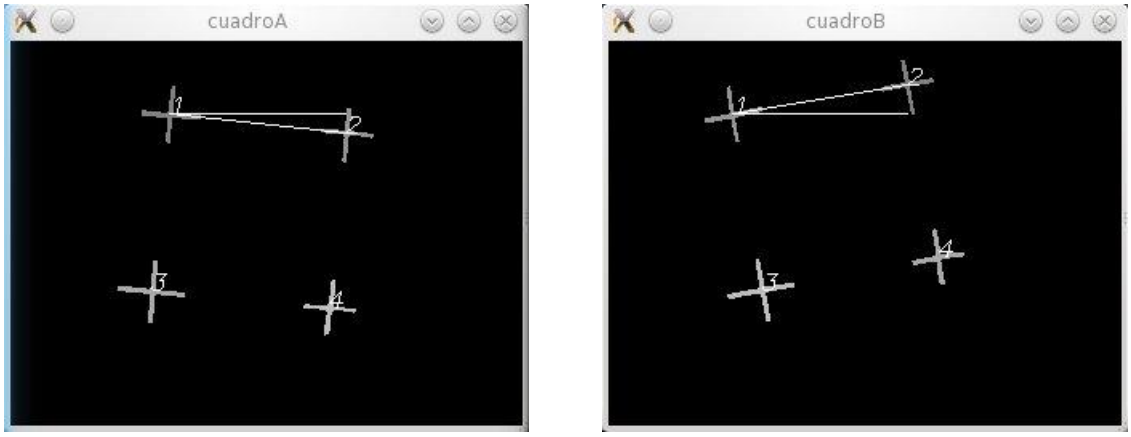


Fig. 4.22 Cálculo de ángulo entre marcas (M_1, M_2) en *cuadroA* y *cuadroB*

A continuación se calcula el ángulo y la distancia entre la marca M_1 y el centro de la imagen del *cuadroA*. Al ángulo obtenido se le suma la diferencia de ángulos entre marcas $\Delta_{\phi}^{1:2}$ previamente calculado (ec. 4.5), este nuevo ángulo describe la posición que tenía el robot con respecto a la marca M_1 en el *cuadroA* junto con la rotación que sufrió la marca M_1 en el *cuadroB*. Utilizando este ángulo y la distancia original entre el robot y la marca M_1 se calculan los componentes de la distancia considerando la rotación que sufrió en el *cuadroB*. Finalmente para terminar este paso, a los componentes rotados de la distancia se les resta las coordenadas de

la marca M_1 del *cuadroB*. Ésta es la posición original del robot, el punto desde donde se tomó la imagen del *cuadroA*, pero visto en el *cuadroB* (fig. 4.23).

$$\phi_x^C = \arctan\left(\frac{MtA_x^i}{MtA_y^i}\right) \qquad d = \sqrt{MtA_x^{i\,2} + MtA_y^{i\,2}} \qquad r_x = d_x - MtB_x^i \qquad (4.6)$$

$$\phi_x^{C'} = \phi_x^C + \Delta_\phi^{i:j} \qquad d_x = sin\left(\phi_x^{C'}\right) \cdot d \qquad r_y = d_y - MtB_y^i$$

$$d_y = cos\left(\phi_x^{C'}\right) \cdot d$$

ϕ_x^C Ángulo entre el origen C y la marca M_i en *cuadroA*

$\phi_x^{C'}$ Ángulo entre el origen C y la marca M_i, considerando el la rotación de los ejes.

d Distancia entre en centro de la imagen y la marca M_i

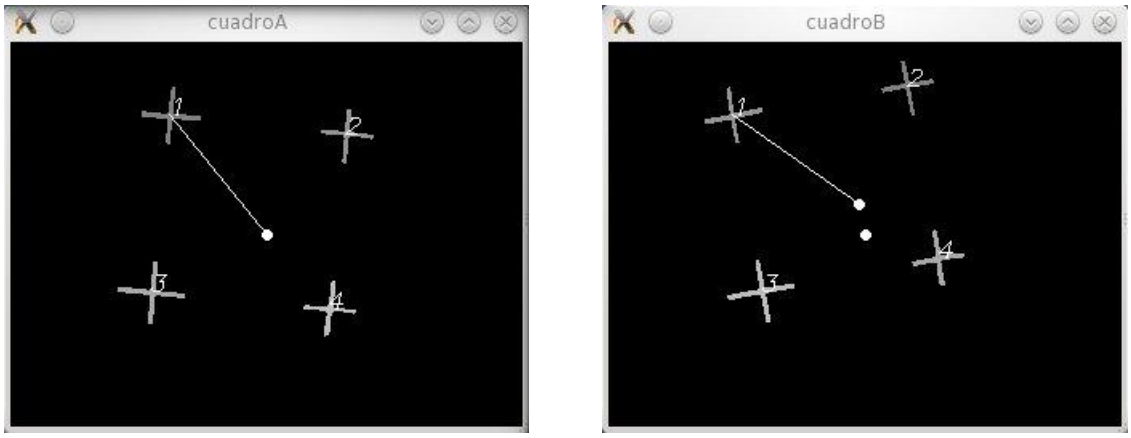


Fig. 4.23 Cálculo de la posición inicial del robot bajo las coordenadas del *cuadroB*

A partir de la posición original se puede calcular la traslación y rotación que efectuó el robot, calculando de nuevo la variación de los ángulos entre cuadros, pero ahora con respecto a la posición inicial del robot (ec. 4.7), un factor a considerar es que el sistema de coordenadas de las marcas está basado en tomar el centro de la imagen como origen, por lo que al usar coordenadas del *cuadroA* en el *cuadroB*, primero se deben transformar. Una vez calculada el cambio de rotación y translación, de nuevo se utiliza la matriz de rotación para actualizar la posición del robot; la orientación se debe actualizar después de hacer la actualización de la posición.

$$\Delta_x = \frac{\sum r_x}{n} \qquad \Delta_y = \frac{\sum r_y}{n} \qquad \Delta_\emptyset = \frac{\sum\left(\mathrm{B}_\phi^c - (\phi_x^c + \pi)\right)}{\mathrm{n}}$$

$$R_x = R_x + \Delta_x \cdot cos(R_\emptyset) - \Delta_y \cdot sen(R_\emptyset) \qquad (4.7)$$
$$R_y = R_x + \Delta_x \cdot sen(R_\emptyset) + \Delta_y \cdot cos(R_\emptyset)$$
$$R_\emptyset = R_\emptyset + \Delta_\emptyset$$

B_ϕ^c Ángulo entre el origen del *cuadroA*, y la marca M_i en el cuadroB

n Numero de combinaciones, que se calcularon

Puesto que este cálculo se hace para cada par de marcas se tendrán múltiples resultados, por lo que se promedian. Es posible detectar una marca erróneamente asociada entre cuadros cuando los resultados que la ligan difieren en comparación del resto.

VI. Fallo en la autolocalización, secuestro

El método para calcular la transformación de translación y rotación del robot depende de que al menos dos marcas sean reconocidas entre cuadros consecutivos; si esta condición no se cumple el robot no podrá calcular el movimiento efectuado y aunque puede volver a calcular el desplazamiento cuando esta condición se cumpla de nuevo, el robot ya estará perdido puesto que no sabe desde que punto efectuó ese último desplazamiento. Esto hace al mapa inservible ya que el robot es incapaz de autolocalizarse en éste.

El robot en ese momento no sabe en donde se encuentra y puesto que cada marca es idéntica a las demás no puede aplicar una búsqueda en el mapa por las marcas que tiene visible en ese momento; Le sería imposible distinguir entre una nueva zona con nuevas marcas o una que revisita con marcas que ya tiene incorporadas en el mapa por lo que le es imposible reautolocalizarse, y es una condición de la que el robot no puede recuperarse.

Supongamos que las marcas son distinguibles entre sí, no necesariamente con una firma única, pero que un grupo de ellas, junto con la información de su posición, haga a tal conjunto único dentro del mapa. Por ejemplo si la firma de

algún rasgo visual es un valor de entre $[1 - 100]$, sin duda habría muchas marcas con valores repetidos en un mapa con varios cientos de marcas, pero es posible que una región tenga cierta combinación característica; una marca con la firma *k* tiene como vecinas inmediatas (cierto radio de búsqueda) a cuatro marcas con firmas: v_1, v_2, v_3, v_4 que individualmente no son únicas pero que en grupo es muy probable que identifique precisamente una cierta región. Un algoritmo para resolver el problema de secuestro antes descrito (donde el robot no puede seguir una parte de su trayectoria) es RANSAC, *Random Sampling Consensus*, y es utilizado para resolver problemas de movimiento muy veloz de la cámara y oclusión a las marcas por objetos más cercanos [49].

En el caso particular de esta implementación no se hace ningún tratamiento especial por lo que el robot asume que el último punto donde se autolocalizó, es el mismo desde donde vuelve a ver marcas y por lo tanto a menos que en verdad no se haya movido de su lugar, dañara el mapa.

VII. Presentación de resultados

Simplemente se imprimen todas las marcas del mapa, junto con el histórico de la posición y orientación del robot, estos datos son luego usados en Matlab® para graficar los resultados que se presentan en este documento.

La gráfica generada (fig. 4.24) utiliza un punto para indicar la posición de las marcas, y junto a éste coloca el identificador numérico de cada marca. La posición del robot se representa con una *"O"* que tiene una flecha que indica la orientación del robot en ese punto. Las unidades de los datos son presentados en pixeles, pues es así como se utilizan durante todas las operaciones, pero ya que se trabaja con dos planos paralelos entre sí (la superficie donde circula el robot y el techo donde se encuentran las marcas artificiales) es posible utilizar un simple factor (1.85 pixeles / cm) para convertirlo.

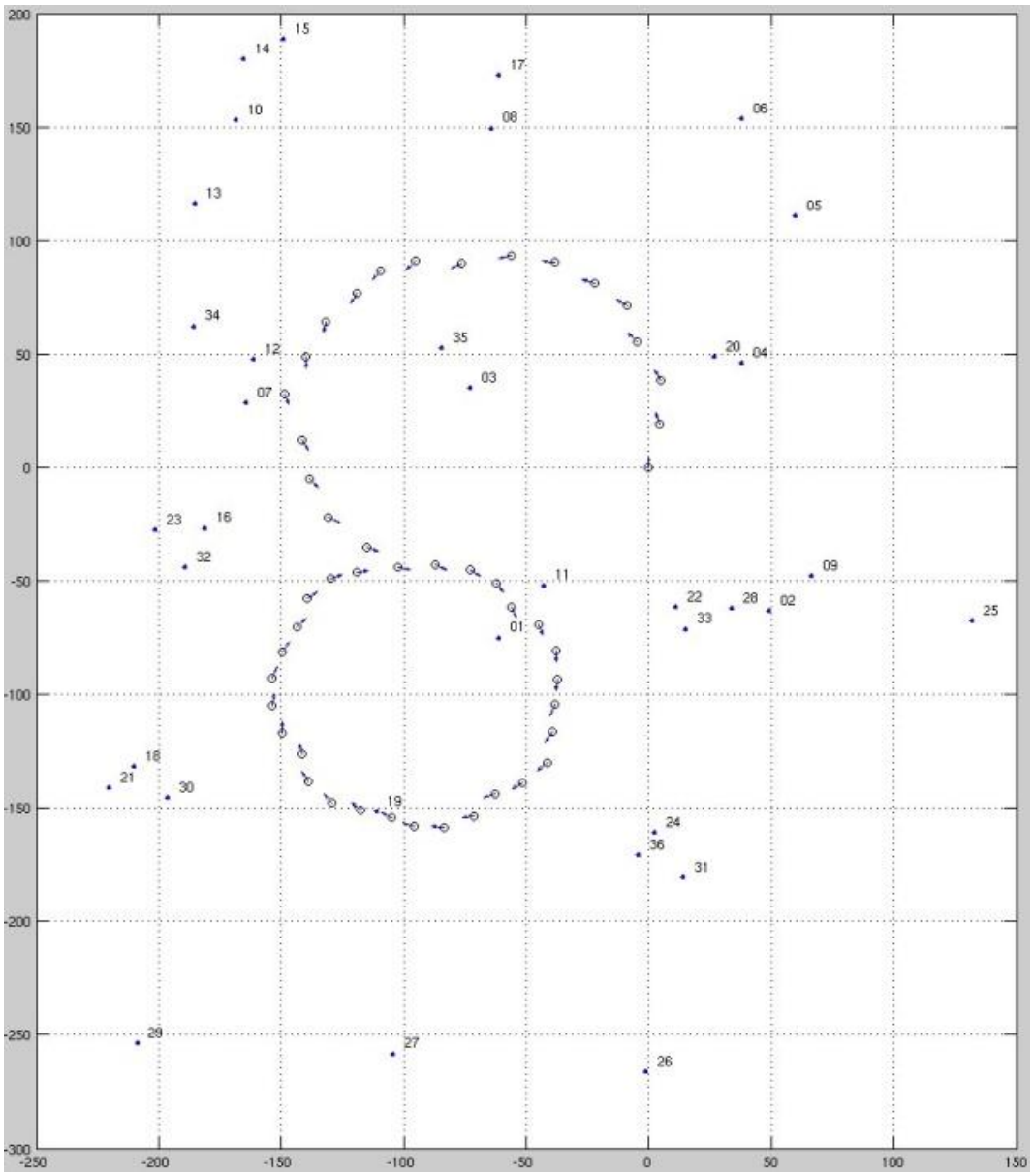


Fig. 4.24 Mapa generado por el robot luego de realizar dos vueltas completas con sentidos opuestos, una trayectoria en forma de “8”

Capítulo 5

Experimentos y resultados

En este capítulo se explican los experimentos realizados, se discuten los resultados obtenidos y se presentan posibles temas de trabajo futuro.

5.1 Experimentos

Se preparó en el techo de una zona relativamente abierta un pequeño campo de marcas artificiales (fig. 4.19a), se colocaron 27 de ellas en un arreglo rectangular de 3x5 y 4x3, se alinearon al contorno del techo falso cuadrado que tiene 61 cm de lado. Las marcas fueron fabricadas con cinta negra de 1.8 cm. de ancho, se colocaron en forma de cruz de 20 cm. x 20 cm. En esta zona se realizaron tres experimentos que se explican a continuación.

En el primero, el robot recorre todo el largo de la pista y de regreso, en algunos puntos se corrigió la trayectoria para evitar que las marcas salieran del rango de visión, se presenta a continuación la gráfica resultante en centímetros.

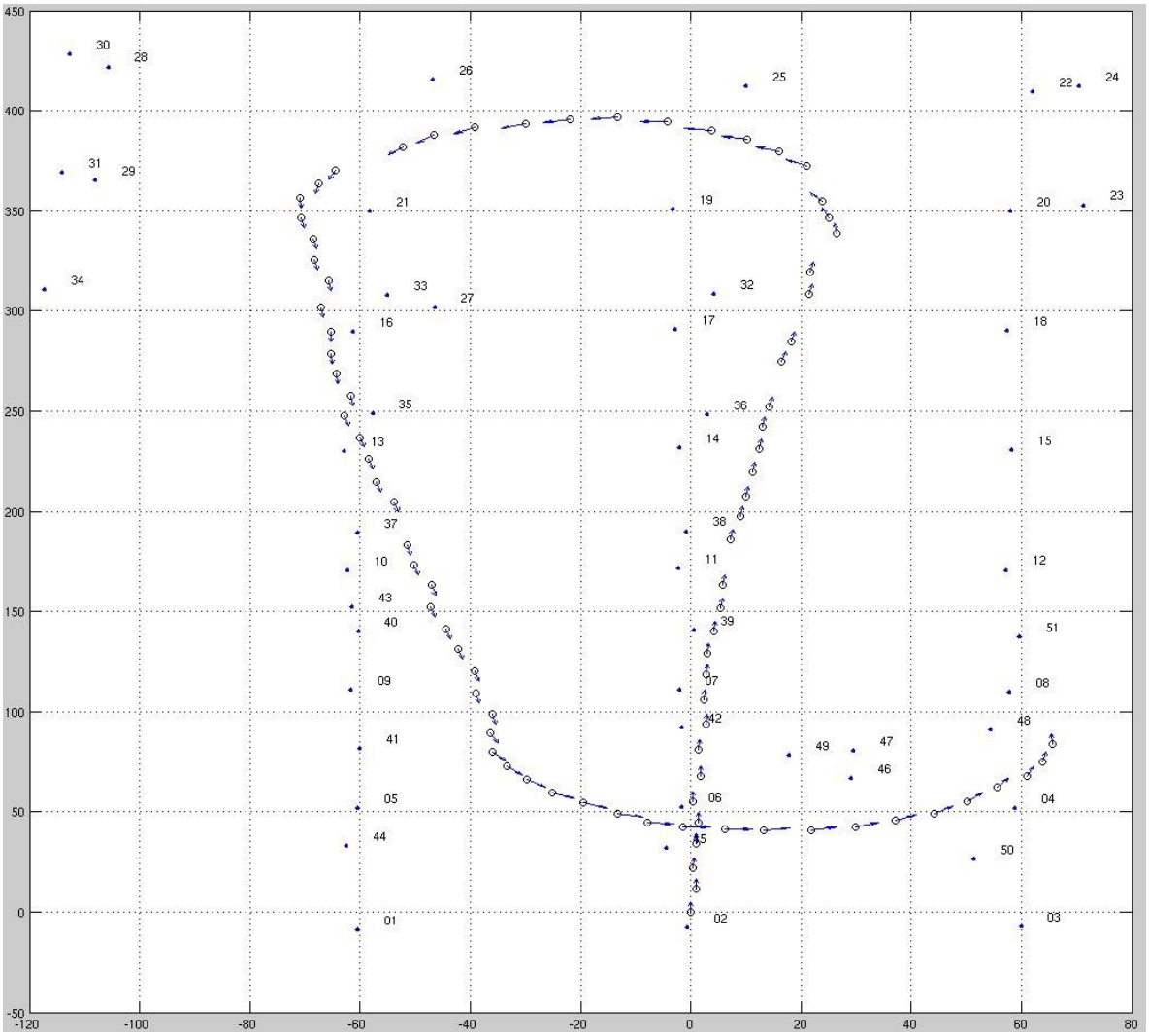


Fig. 5.1 Grafica generada del experimento 1

En el mapa se reportan 50 marcas, y como se recuerda únicamente había 27, puede verse claramente que en mucho casos creó una nueva marca para referenciar a la misma, ejemplos: 29-31, 30-26, 16-27-33, y en algunos casos tomó marcas naturales (fig. 5.2) 46,47,49, por lo que es claro que el mapa es inexacto, el error más grande detectado es la marca 20 (58.06, 350) y la 23 (71.21, 352.9), ambos en realidad son el mismo objeto (fig. 5.3a y 5.3b). El primer error es algo que se conoce en SLAM como asociación de datos, la habilidad de asociar observaciones de un mismo objeto aún en la presencia de discrepancias como valores de un rasgo visual; El segundo es la diferencia de 13.46 cm, cerca de 25

pixeles entre las estimaciones sobre la posición del mismo objeto, si bien el error es de aproximadamente 3 % con respecto al largo de 4.5 mts que tiene la zona de pruebas, el error de la estimación de las marcas esta correlacionado entre si ya que depende de la estimación de la posición del robot, es decir el mapa y no solo esa marca tienen un error proporcional.

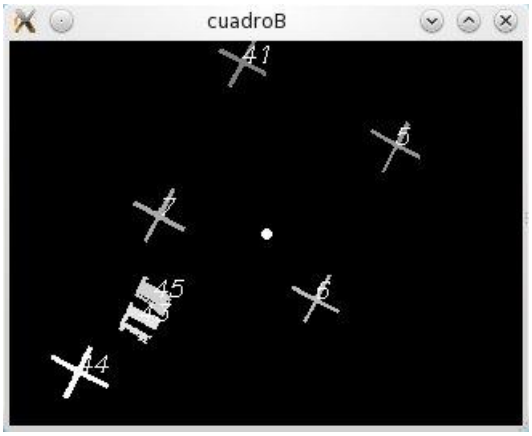
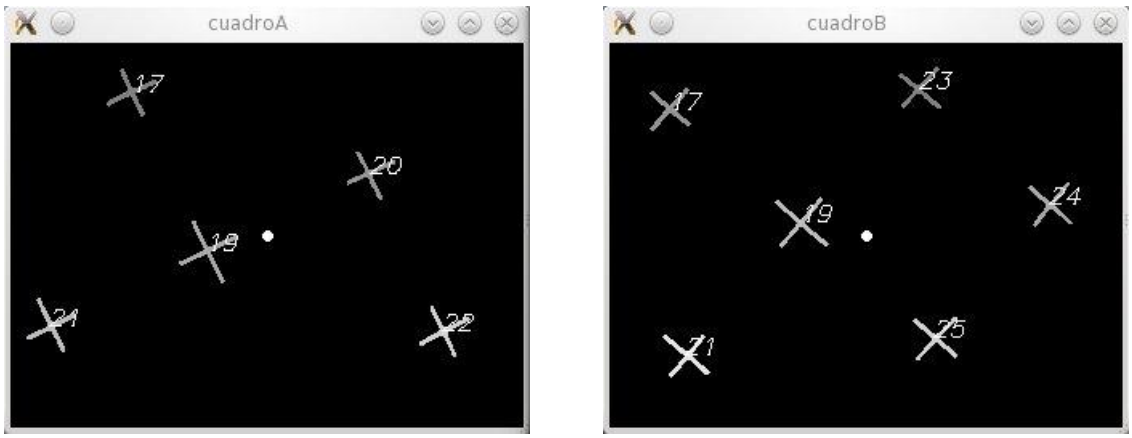


Fig. 5.2 Referencias naturales



a) b)

Fig. 5.3 Error en asociación de datos, marcas 20-23 y 22-24

En el segundo experimento se ejecuta una trayectoría con la forma de un “8”, el primer círculo girando inicialmente al contrario de las manecillas del reloj y el siguiente en el sentido de las manecillas de reloj (fig. 5.4).

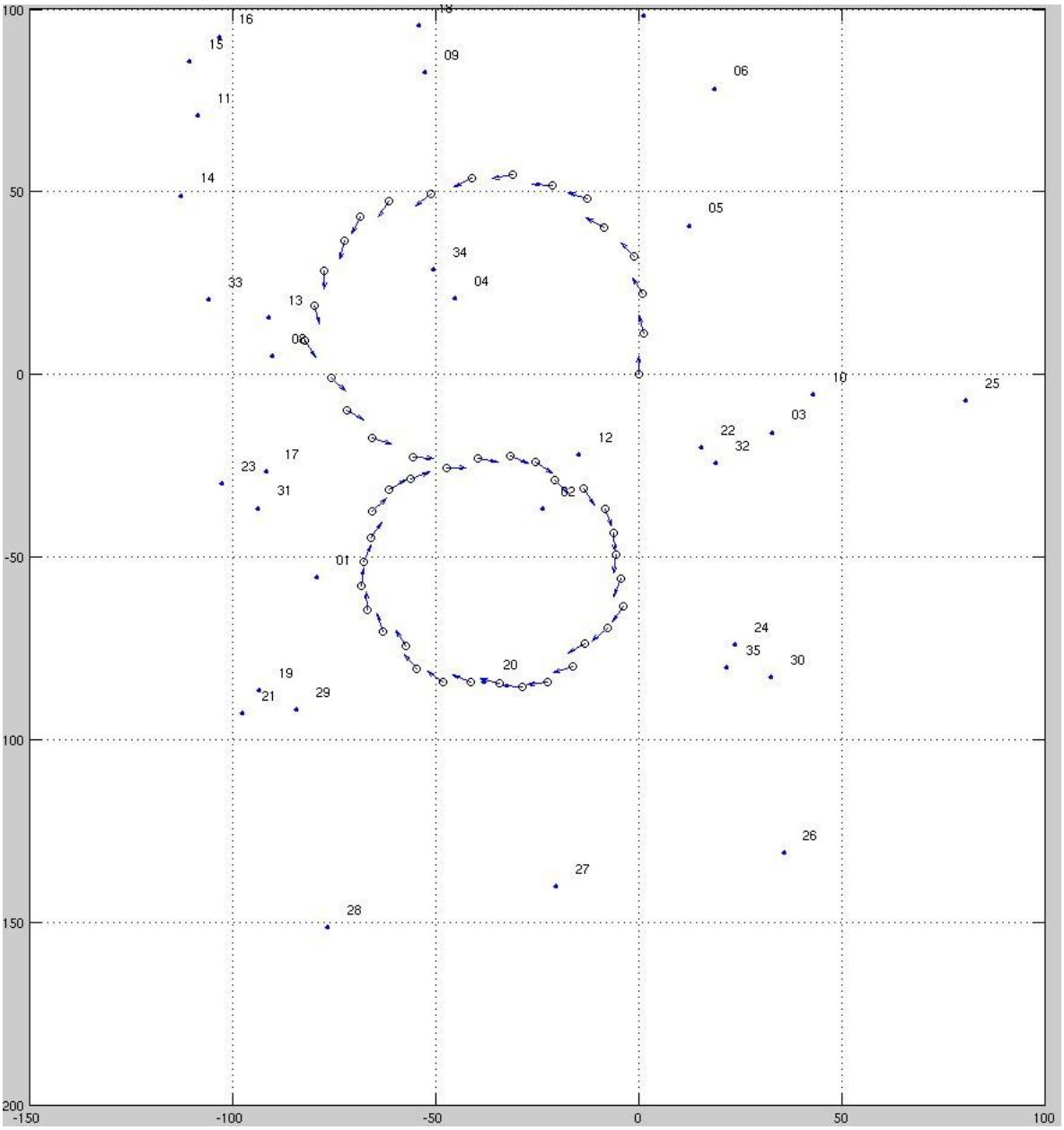
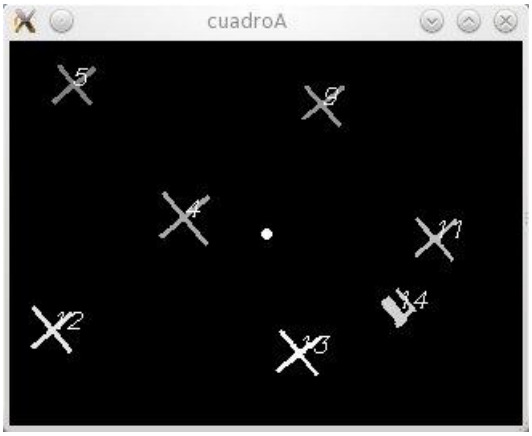
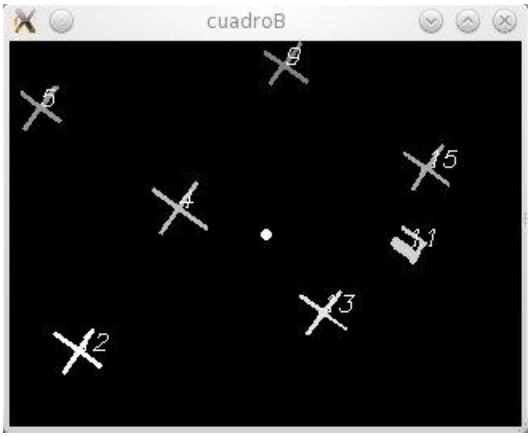


Fig. 5.4 Grafica generada del experimento 2

En este experimento ocurre el mismo detalle que en el experimento 1, no logra asociar las marcas y crea nuevas: 3-10-22-32, y ocurre un nuevo detalle, un error en la búsqueda activa (fig. 5.5). Este nuevo error es provocado porque la posición original de una marca es ocupada por una marca distinta, y el algoritmo presentado en el capítulo anterior únicamente utiliza la distancia como parámetro de decisión.





a) b)

Fig. 5.5 Error en búsqueda activa de referencias, marca 14-11

En el tercer experimento se hizo que el robot saliera de la zona de pruebas, perdiendo visibilidad con las marcas y provocando que el robot no pudiera calcular su desplazamiento (fig. 5.6 y 5.8). Su posición actual es ahora desconocida, sin embargo para el robot simplemente no existe cambio en la posición.

Cuando el robot regresa a la zona de pruebas, con una orientación invertida (fig. 5.7 y 5.9) asume erroneamente que está en el mismo punto donde dejo de observar las marcas. Las marcas que observa son de hecho parte del mapa que ya tenía almacenado pero la posición es malinterpretada y crea nuevas marcas y en una posición falsa. El mapa ya no es consistente y el robot no sabe realmente donde se encuentra.

Fig. 5.6 Fotografía del robot saliendo de la zona de pruebas

Fig. 5.7 Fotografía del robot regresando a la zona de pruebas

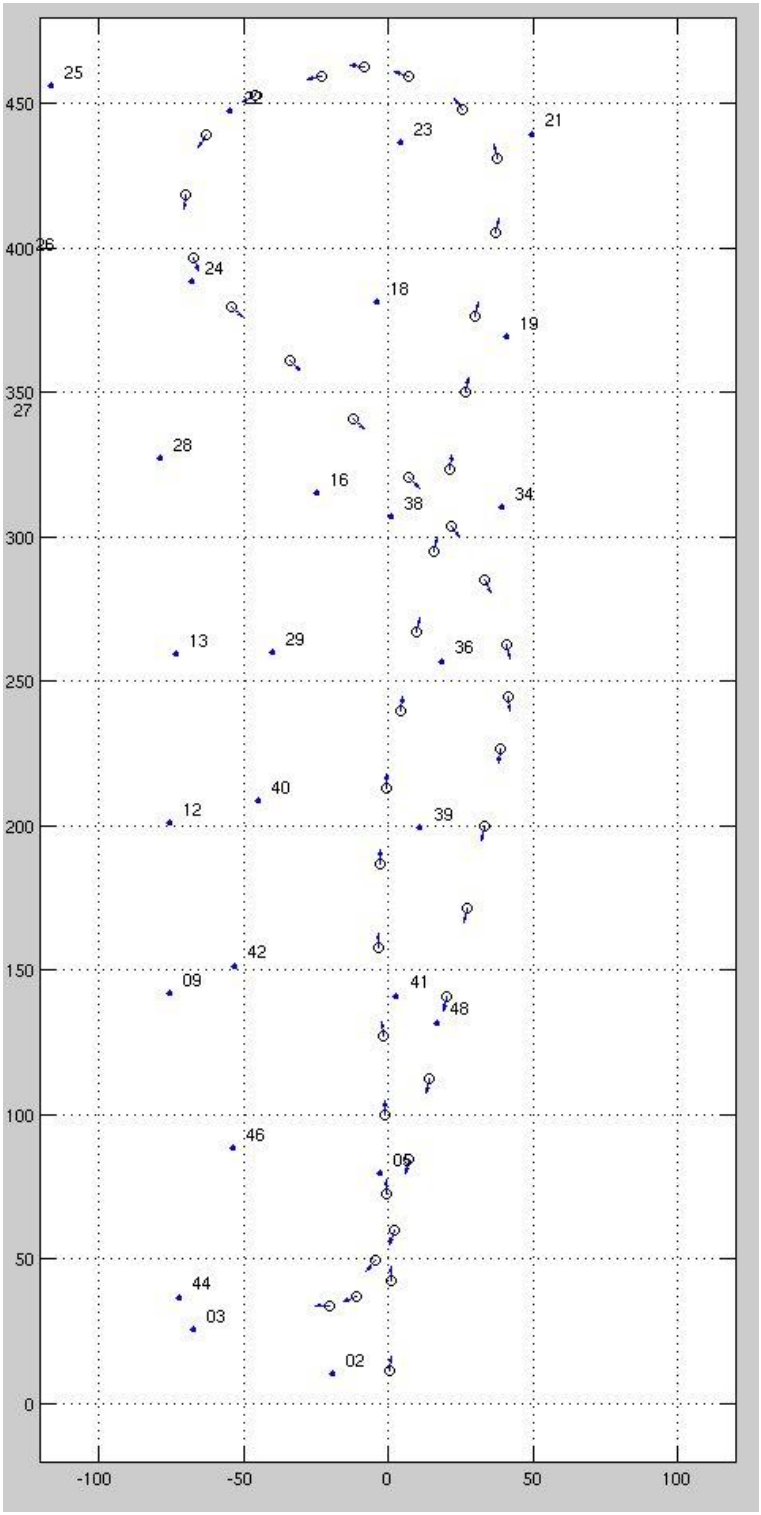


Fig. 5.8 Gráfica generada del experimento 3, antes de salir de la zona de pruebas

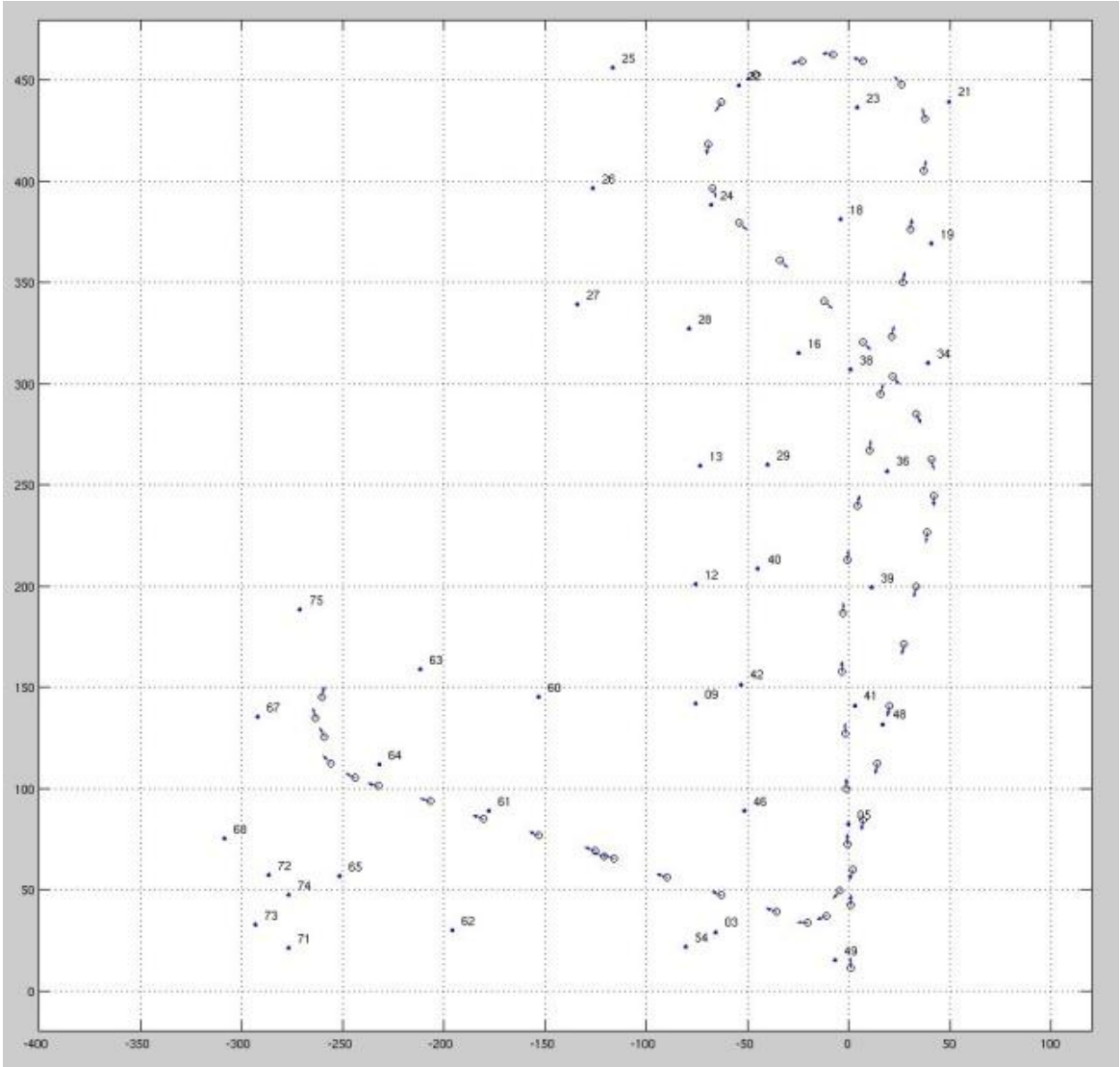


Fig. 5.9 Gráfica generada del experimento 3, después de regresar a la zona de pruebas

5.2 Conclusiones

El objetivo general, implementar una solución al problema de localización y mapeo para un robot móvil utilizando visión, se cumplió. El mapa fue creado y refleja los objetos que existen en el mundo real, además de permitirle al robot conocer su posición en todo momento con respecto a la posición de arranque. Sin embargo, como se pudo notar, éste tiene ciertas inconsistencias que se discuten a continuación.

El objetivo particular, construir un robot móvil se cumplió satisfactoriamente, si bien existen varias posibles mejoras mecánicas para darle estabilidad a la cámara no es impedimento para el funcionamiento de la técnica propuesta.

El objetivo particular, implementación sobre un dispositivo FPGA también se cumplió satisfactoriamente, aunque existen posibles mejores que eliminarían errores en la búsqueda activa. Como se mencionó en el capítulo 4, el éxito de esta técnica depende de que el cambio entre imágenes sea pequeño, en este proyecto la transferencia de imágenes y su procesamiento consumen tiempo considerable, sobre todo en el caso de giros puede provocar errores como el de la fig. 5.5. Para eliminar este tipo de error se podría: mejorar el protocolo de transferencia, aumentar el número de datos enviados por solicitud o mejor aún, eliminar la necesidad de transferir la imagen para que sea procesada. Para la última solución mencionada se requeriría que se diseñe y construya en hardware un bloque que pueda obtener las coordenadas de los centros geométricos de las marcas.

Finalmente el objetivo particular, diseño y evaluación de técnicas para localización y mapeo se cumplió satisfactoriamente sin embargo, es el punto que mayor área de mejora tiene. El error en la asociación de datos es consecuencia de la falta de un método probabilístico que permita que la posición de las marcas se reevalúe con cada cuadro que es analizado. El método propuesto en este trabajo asume que no existe error entre estimaciones y mantiene la primer hipótesis sobre la posición de la marca e indirectamente del robot como una verdad absoluta, cuando el robot regresa a una zona visitada y no puede encontrar las marcas en la posición indicada por el mapa crea nuevas marcas, lo cual significa que la posición de robot es incongruente. Una técnica adaptada como EKF podría mantener una mejor estimación de la posición real de las marcas ya que mantiene un nivel de incertidumbre sobre este mismo dato.

5.3 Trabajo futuro

El hecho de que se requiera colocar marcas artificiales en donde el robot circula rompe la idea de navegación autónoma. No es deseable o factible en muchos casos colocar marcas tan evidentes, por lo que una de las primeras ideas de mejora es darle la habilidad de poder circular sin la necesidad de alterar el entorno.

El utilizar marcas naturales implicaría seleccionar un tipo de entorno que definiría el tipo de imágenes que se pueden obtener y junto con ellas una técnica adecuada para la selección de marcas. No existe técnica de análisis digital de imágenes que nos permita encontrar puntos bajo cualquier condición. Junto con este punto si se decidiera trabajar con marcas en el espacio y ya no en el plano se tendrían implicaciones considerables para el cálculo de la posición de las marcas y por lo tanto la posición del robot.

El siguiente punto es el filtro probabilístico, éste debe poder ser ejecutado en un tiempo adecuado de acuerdo a la velocidad del robot y limitado por el poder de cómputo del procesador, por lo que el número de marcas debería ser bajo o se debería utilizar un método que permita la división en submapas.

Otro punto mucho más general es sobre las dos preguntas restantes de la navegación autónoma, ¿a dónde debo ir?, ¿y cómo puedo llegar ahí?, sin duda una vez que se construye un mapa se debe poner en uso. Ya existen ideas que mezclan técnicas para lograr esto, como RatSLAM que es una mezcla de SLAM y planeación de trayectorias. Así que el siguiente objetivo sería controlar el movimiento del robot para ir a un lugar definido mientras se ejecuta SLAM para corroborar la ejecución de la trayectoria.

Referencias

[1] U. NEHMZOW. *Mobile robotics: a practical introduction 2ed. Springer 2003.* ISBN: 1852337265.

[2] R. SIEGWART, I. R. NOURBAKHSH*. Introduction to autonomous mobile robots.* The MIT Press 2004. ISBN: 026219502X.

[3] S. THRUN, W . BURGARD, D. FOX*. Probabilistics Robotics.* The MIT Press 2005. ISBN: 0262201623.

[4] S. THRUN, D. FOX, W. BURGARD. *A probabilistic approach to concurrent mapping and localization for mobile robots*. Mach. Learning Autonomous. Robots. 1998 Vol. 31, pp. 29-53.

[5] H. DURRANT-WHYTE, D. RYE, and E. NEBOT, *Localization of automatic guided vehicles*. Robotics Research: The 7th International Symposium (ISRR'95), G. Giralt and G. Hirzinger, Eds. New York: Springer Verlag. 1996 pp. 613–625.

[6] T. BAILEY, H.F. DURRANT-WHYTE. *Simultaneous Localization and Mapping: Part I*. IEEE Robotics & Automation Magazine. Jun 2006. pp. 99-108.

[7] H. CLOSET, K. M. LYNCH, S. HUTCHINSON, G. KANTOR, W. BURGARD, L. E. KAVRAKI, S. THRUN. *Principles of robot motion theory, algorithms and implementations*. The MIT Press. 2005. ISBN: 0262033275.

[8] J. LEONARD, H.F. DURRANT-WHYTE. *Mobile Robot Localization by Tracking Geometric Beacons*. IEEE Transactions on Robotics and Automation. 1991 Vol. 7, No. 3, pp. 376-382.

[9] J. BORENSTEIN, H. R. EVERETT, and L. FENG. *Navigating Mobile Robots: Sensors and Techniques*. A. K. Peters, Ltd. Wellesley, MA. 1996.

[10] A.J. DAVISON, D. REID, D. MOLTON, O. STASSE. *MonoSLAM: Real-Time single camera SLAM*. IEEE Transactions on pattern analysis and machine learning, 2007 Vol. 29, No. 6, pp. 1052-1067.

[11] A.J. DAVISON, W.W. MAYOI, D.W. MURRAY*. Real-Time Localization and Mapping with Wearable Active Vision*. IEEE International Symposium on mixed and augmented Reality (ISMAR). 2003 pp. 18-27.

[12] M.W.M.G. DISSANAYAKE, P. NEWMAN, S. CLARK, H.F. DURRANT-WHYTE. *A Solution to the Simultaneous Localization and Map Building (SLAM) Problem*. IEEE Transaction on Robotics and Automation, 2001 Vol. 17 No. 3 pp. 229-241.

[13] R. TELLÉZ, F. FERRO, D. MORA, D. PINYOL, D. FACONTI. *Autonomous humanoid navigation using laser and odometry data*. Humanoid Robotics. 8th IEEE-RAS International Conference. 2008 pp. 500-506.

[14] J.M. SAEZ, A. HOGUE, F. ESCOLANO, M. JENKIN. *Underwater 3D SLAM through entropy minimization*. Robotics and Automation 2006, ICRA 2006. pp. 3362-3567.

[15] K. CELIK, M. SOON-JO CHUNG CLAUSMAN, A.K. SOMANI. *Monocular vision SLAM for indoor vehicles.* Intelligent Robots and Systems 2009, IROS 2009 pp. 1566-1573.

[16] D.C. BROWN. *Close-range camera calibration*. Photogrammetric Engineering, 37(8), 1971, pp. 855–866.

[17] Z. Zhang. *Flexible camera calibration by viewing a plane from unknown orientations*. Computer Vision. The Proceedings of the 7th IEEE International Conference on, 1 vol.1, 1999. pp. 666–673.

[18] J. HEIKKILA, O. SILVEN. *A four-step camera calibration procedure with implicit image correction.* Computer Vision and Pattern Recognition. IEEE Computer Society Conference 1997 pp. 1106-1112.

[19] T.A. Clarke, J.G. Fryer. *The Development of Camera Calibration Methods and Models*, Photogrammetric Record, April 1998 16(91) pp. 51-66.

[20] S. ALBRECHT. An *Analysis of VISUAL MONO-SLAM*, Master's Thesis. Universität Osnabrück. 2009.

[21] J. CIVERA, A.J.DAVISON, J.M.M. MONTIEL. *Inverse Depth Parametrization for Monocular SLAM*. IEEE Transactions on Robotics. October 2008 24(5) pp. 932–945.

[22] Davison, A.J. *Real-Time Simultaneous Localization and Mapping with a Single Camera.* In ICCV '03: Proceedings of the 9th IEEE International Conference on Computer Vision. 2003. pp. 1403-1410. ISBN: 0769519504.

[23] J. CANNY. *A computational approach to edge detection.* Patter Analysis and Machine Learning. 1986 8(6). pp. 679-698.

[24] C. HARRIS, M. STEPHENS. *A combined corner and edge detection. Proc. of 4th Alvey Vision Conference, Manchester 1988 . pp. 189-192.*

[25] J. SHI, C. TOMASI*. Good features to track*. Proc. of the International Conference on Computer Vision and Pattern Recognition. 1994. pp. 593-600.

[26] C. MAXFILED. *The Design Warrior's Guide to FPGAs. Devices, Tools and Flows.* ELSEVIER, E.U.A., 2004. ISBN: 0750676043.

[27] D. G. LOWE, *Distinctive image features from scale-invariant keypoints*, *International Journal of Computer Vision.* 2004 60 (2). pp. 91-110.

[28] S. SE, D.G. LOWE, J. LITTLE. *Vision-based global localization and mapping for mobile robots*. *IEEE Transactions on Robotics.* 2005 21(3). pp. 364-375.

[29] Y. KE, R. SUKTHANKAR. *PCA-SIFT: A More Distinctive Representation for Local Image Descriptors*, *Proc. conf. Computer Vision and Pattern Recognition*. 2004. pp. 511-517.

[30] H. BAY, T. TUYTELAARS, L. V. GOOL. *SURF: Speeded Up Robust Features. 9th European Conference on Computer Vision* 2006.

[31] N. Otsu. *A threshold selection method from gray-level histograms*. *IEEE Trans. Sys., Man.,* 1979 *Cyber.* 9: pp. 62–66.

[32] R. C. GONZALEZ, R. E. WOODS. *Digital Image Processing*, 3rd edition. Prentice Hall 2007. ISBN: 013168728X.

[33] G. Antonelli, S. Chiaverini. Linear Estimation of the Odometric Parameters for Differential-Drive Mobile Robots. Intelligent Robots and Systems. 2006 9(15). pp. 3287-3292.

[34] T.D. Larsen, K.L. Hansen, N.A. Andersen, R. Ole. Design of Kalman filters for mobile robots; evaluation of the kinematic and odometric approach. Control Applications. Proceedings of the IEEE International Conference. 1999 (2). pp. 1021 - 1026.

[35] B. CLIPP, C. ZACH. J.M. JONGWOO LIM FRAHM, M. POLLEFEYS. *Adaptive, real-time visual simultaneous localization and mapping.* Applications of Computer Vision Workshop 2009 (7)8. pp. 1-8.

[36] MYUNG HWANGBO, K. JUN-SIK, T. KANADE. *Inertial-aided KLT feature tracking for a moving camera*. Intelligent Robots and Systems. IROS IEEE/RSJ International Conference. 2009 (10)15 pp. 1909-1916.

[37] J. KLIPPENSTEIN, HONG ZHANG. *Performance evaluation of visual SLAM using several feature extractors*. Intelligent Robots and Systems. IROS IEEE/RSJ International Conference. 2009 10(15) pp. 1574-1581.

[38] H. DURRANT-WHYTE, T. BAILEY. *Simultaneous localization and mapping, tutorial*. Robotics & Automation Magazine, IEEE. 2006 (13) – 2. pp. 99-110.

[39] J. L. SOUMAN, I. FRISSEN, M. N. SREENIVASA, M. O. ERNST, *Walking Straight into Circles.* Current Biology 2009(19)-18. pp. 1538-1542.

[40] V. BONATA, V. PERON, R. WOLF, D.F. DE HOLANDA, J.A.M., MARQUES, E. CARDOSO. *An FPGA Implementation for a Kalman Filter with Application to Mobile Robotics.* 2007.

[41] L.M PAZ, J.D TARDOS, J. NEIRA*. Divide and Conquer: EKF SLAM in O(n).* Robotics, IEEE Transactions. 2008(24)-5. pp. 1107-1120

[42] H.F. DURRANT-WHYTE. *Uncertain geometry in robotics*. IEEE Trans. Robot. Automat. 1988 (4)-1. pp. 23–31.

[43] R. SMITH, P. CHEESMAN. *On the representation of spatial uncertainty Int.* J. Robot. Res. 1987(5)-4. pp. 56–68.

[44] M. MONTECARLO, S. THRUN. *Simultaneous localization and mapping with unknown data association using FastSLAM*. Robotics and Automation. 2003 (9)-2. pp. 1985–1991.

[45] G.P. HUANG, A.I. MOURIKIS, S.I. ROUMELIOTIS. *On the complexity and consistency of UKF-based SLAM*. Robotics and Automation. 2009 (3). pp. 4401-4408.

[46] Z. WU, Z. CHUNXIA, G. JIANHUI. *The Study of Improving Kalman Filters Family for Nonlinear SLAM*. JF Journal of Intelligent & Robotic Systems. 2009 (56)-5. Springer Netherlands. pp. 543-564. ISSN: 0921-0296.

[47] M. MONTECARLO, S. THRUN, D. KOLLER, B. WEGBREIT. *FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges*. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. 2003.

[48] S. THRUN. *Robotic mapping: A survey*. Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann. 2002.

[49] B. WILLIAMS, P. SMITH. I REID. *Automatic Relocalisation for a Single-Camera Simultaneous Localisation and Mapping System. Robotics and Automation*. 2007 (4). pp. 2784-2790.

Anexos

A. Técnicas de recuperación de marcas

Al inicio de este proyecto se deseaba utilizar marcas naturales y este apartado describe la experiencia con tres técnicas de recuperación de marcas que se intentaron utilizar antes de decidir utilizar marcas artificiales.

A.1 Mono SLAM

Mono SLAM es una especialización de Visual SLAM que utiliza una única cámara como sensor, pero ésta se puede mover en las tres dimensiones y rotar libremente. El vector que describe su estado tiene 13 variables [10], pero es posible que el número de variables que describen el estado de la cámara varíe entre implementaciones.

El primer problema y la principal consideración de esta técnica es el cálculo de profundidad, el hecho de tener una única cámara, obliga a que se utilice triangulación con un segundo cuadro en una nueva posición la cual se desconoce (si se conociera no habría problema que resolver), por lo que usualmente los experimentos que utilizan ésta técnica tienen ciertas peculiaridades:

- Tienen marcas iniciales a una distancia conocida, por ejemplo un cuadrado de cierto tamaño y a cierta distancia que es utilizado como base para interpretar el movimiento de la cámara antes de poder aprender nuevas marcas.
- Se mueven en forma paralela al plano que observan, esto ayuda a calcular la profundidad de una nueva marca, los movimientos de frente o hacia atrás tienen poco variación (disparidad) en las imágenes sobre un mismo objeto, por lo que no aportan información valiosa.

- Requieren de una tasa cuadros por segundo alta para determinar su movimiento, 15 o 30 cuadros por segundo cuando se trata de una cámara que se sostiene con una mano.
- El autor que propuso esta técnica al inicio ideó un método separado del flujo del mapa para inicializar una marca. Antes de incorporarla al mapa se llevan a cabo una sería de pruebas tratando de adivinar la posición real de la marca trazando una línea en el espacio, luego de varios cuadros, alrededor de 100 con movimientos paralelos con respecto a la marca, es posible agregarla al mapa [20], este método obliga a que la marca se encuentre en un rango de distancia conocido [10]. El nuevo método que propone no limita el rango de distancia y permite incorporar marcas sin saber su posición real inmediatamente al flujo normal. En lugar de utilizar la posición *(x,y,z)* de la marca utiliza la posición, orientación el robot y una función que describe una línea en el espacio que debe intersectar con la marca, a este función le llama profundidad o distancia inversa, conforme la incertidumbre sobre la posición es menor a cierto umbral se reemplaza por sus coordenadas *(x,y,z)*. Aún si nunca se logra calcular la profundidad real al menos se utiliza para calcular la orientación del robot. Si bien no requiere de marcas iniciales con una posición conocida requiere que sea posible que se añadan varias marcas rápidamente al mapa, por lo que deben existir varios objetos cerca de la cámara [21].

La dimensión del mapa y el tipo de movimiento requerido hicieron de esta técnica un algoritmo demasiado complejo para implementar en un dispositivo FPGA. Aun cuando se añadió un procesador al diseño, este tiene un reloj de 100 MHz por lo que su capacidad de cómputo es muy limitada para este fin.

Se buscó otro método que requiriera menos poder de cómputo al sacar provecho de que el robot puede describir su movimiento con únicamente tres componentes: dos ejes y un ángulo, pero manteniendo marcas naturales en el espacio.

A.2 KLT

Inicialmente se trató de explotar la idea de que la zona inicial de pruebas serían pasillos con abundantes líneas que forman esquinas utilizando KLT; Como se explicó en la sección de análisis digital de imágenes, KLT es un método para detección y seguimiento de esquinas basada en los eigenvalores de la matriz espacial del gradiente. No solo describe el rasgo a seleccionar sino un método eficiente para localizarlo en una imagen posterior, pero requiere que exista información del gradiente cerca del punto inicial de búsqueda, la posición de la esquina en el cuadro original [25].

En este caso las pruebas iniciales se llevaron a cabo con una cámara digital comercial con videos a 15 o 30 cuadros por segundo, las pruebas en código aun tomando únicamente 5 cuadros por segundo lograban buenos resultados de selección y seguimiento de esquinas. Sin embargo la cámara de este proyecto tiene una área de visión angosta en comparación, por lo que usualmente las esquinas que podía detectar desde el suelo estaban muy lejos de él y la mayor parte de la imagen estaba ocupada por el suelo o el techo del pasillo los cuales provocaban errores constantes por los cambios de luz; otro detalle es que con la resolución limitada por la memoria interna del chip FPGA las marcas lejanas se veían borrosas y provocaban una baja tasa de éxito para volver a detectar cierta esquina.

Para este fin se desarrolló un componente de hardware que calcula el gradiente, mediante Sobel (fig A.1), la parte inicial del algoritmo de KLT, sin embargo por las razones antes mencionadas no se continuó con la implementación de esta técnica.

A.3 Detección de líneas

El siguiente método propuesto fue utilizar la cuadrícula natural del techo falso del centro de investigación, esto implica reducir las marcas a únicamente dos dimensiones. Además las marcas se encontrarían en un plano paralelo al plano

donde el robot circula por lo que harían mucho más simples los cálculos de observación y eliminaría la incertidumbre sobre la distancia o profundidad desde el robot a una marca.

Para este nuevo método la cámara dejó de apuntar al frente y se apuntó hacia arriba (como se documenta en el capítulo 4), para obtener la retícula se pretendía aprovechar el componente que calcula el gradiente y luego utilizar morfología matemática para encontrar las líneas y los puntos donde coinciden, las esquinas.

Lamentablemente el contraste entre los bordes es muy bajo por lo que en pruebas se observó que se requería un elemento de estructura muy grande (13-21 pixeles) que busca una línea recta en varios posibles ángulos, aún con esto a veces se confundía con el dibujo interno de cada mosaico. Tal algoritmo no es simple de ser diseñado en hardware y el tiempo de ejecución en software no era permisible.

Por lo que se decidió utilizar marcas artificiales con un alto contraste que pueden ser encontradas luego de aplicar un umbral y que es fácil de implementar en hardware.

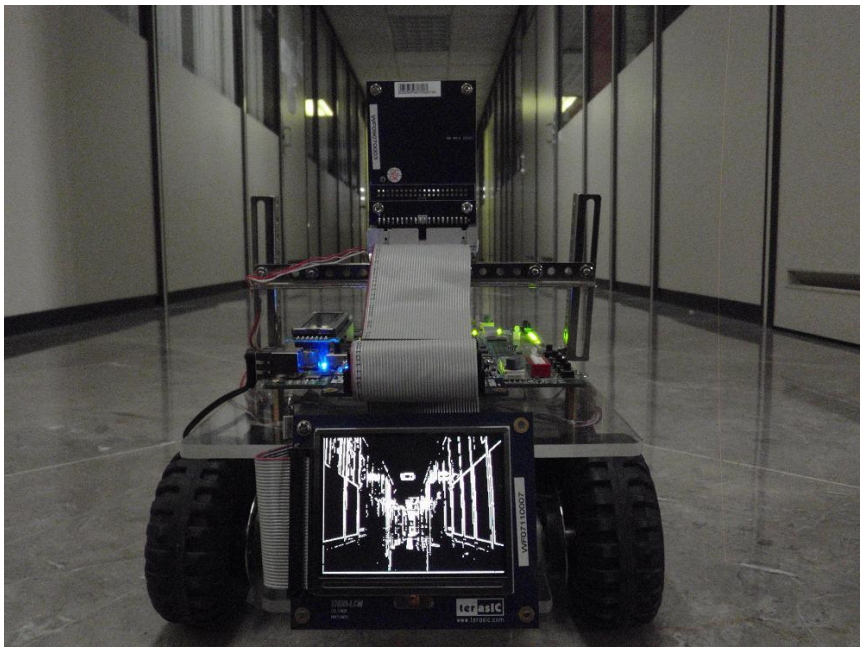


Fig. A.1 Gradiente calculado en tiempo real