SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Odometría mediante visión artificial usando métodos directos

presentada por

Lic. Charles Fernando Velázquez Dodge

como requisito para la obtención del grado de

Maestro en Ciencias de la Computación

Director de tesis
Dr. José Ruiz Ascencio

Codirector de tesis
Dr. Raúl Pinto Elías

Cuernavaca, Morelos, México, julio 2019.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Cuernavaca, Morelos a 19 de junio del 2019
OFICIO No. DCC/062/2019

Asunto: **Aceptación de documento de tesis**

DR. GERARDO V. GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial del **Lic. Charles Fernando Velázquez Dodge**, con número de control M15CE095, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis profesional titulado **"Odometría mediante visión artificial usando métodos directos"** y hemos encontrado que se han realizado todas las correcciones y observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

DIRECTOR DE TESIS

Dr. José Ruiz Ascencio
Doctor en Ciencias
5009035

CO-DIRECTOR DE TESIS

Dr. Raúl Pinto Elías
Doctor en Ciencias en la Especialidad de Ingeniería Eléctrica
3890453

REVISOR 1

Dr. Manuel Mejía Lavalle
Doctor en Ciencias Computacionales
8342472

REVISOR 2

M.C. Gerardo Reyes Salgado
Maestro en Ciencias de la Computación
2493370

C.p. M.E. Guadalupe Garrido Rivera - Jefa del Departamento de Servicios Escolares.
Estudiante
Expediente

NACS/lmz

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Cuernavaca, Mor., 21 de junio de 2019
OFICIO No. SAC/231/2019

Asunto: Autorización de impresión de tesis

LIC. CHARLES FERNANDO VELÁZQUEZ DODGE
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS
DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Odometría mediante visión artificial usando métodos directos"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

SEP TecNM
CENTRO NACIONAL
DE INVESTIGACIÓN
Y DESARROLLO
TECNOLÓGICO
SUBDIRECCIÓN
ACADÉMICA

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO

C.p. Mtra. Guadalupe Garrido Rivera .- Jefa del Departamento de Servicios Escolares.
Expediente

GVGR/mcr

Dedicatoria

A toda mi familia:

Que me han apoyado en todo momento

A mis compañeros y amigos:

Que me ayudaron y brindaron su amistad

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología por el apoyo económico otorgado para realizar mis estudios de maestría.

Al Centro Nacional de Investigación y Desarrollo Tecnológico, por permitirme utilizar sus instalaciones y brindarme los recursos necesarios que permitieron realizar mis estudios de maestría.

A mi director de tesis, el Dr. José Ruiz Ascencio, por asesorarme durante el desarrollo de esta tesis, por brindarme su consejo, apoyo y paciencia.

A mi codirector, el Dr. Raúl Pinto Elías y a mis revisores el Dr. Manuel Mejía Lavalle y el Dr. Gerardo Reyes, por su crítica y comentarios que fueron fundamentales para la realización de esta tesis.

A los compañeros del área de electrónica, en especial a Gerardo Ortiz Torres y Ricardo Schacht Rodríguez, por brindarme su consejo, experiencia y amistad.

A los compañeros del área de computación, en especial a Christian Hernandez Becerra, por apoyarme con su conocimiento cuando lo necesitaba y hacer críticas constructivas que aportaron al desarrollo de esta tesis, a Leonel González Vidales y a Karen Loreli Zaragoza Jiménez por su apoyo y amistad incondicional.

Resumen

En este documento de tesis se reporta una investigación de los métodos directos y semi-directos para odometría visual. Como resultado se presenta un algoritmo de visión por computadora basado en estos métodos capaz de brindar información de localización en tiempo real y construir un mapa en un espacio de tres dimensiones. Con ayuda de este algoritmo se obtuvo información de localización de un robot móvil y mapas de sus trayectorias recorridas.

Se hace una revisión de los antecedentes en TecNM-CENIDET, principalmente los más recientes y que tienen alguna relación con los métodos directos, se describen las bases teóricas para entender el modelo matemático de transformaciones en dos y tres dimensiones, algoritmos de visión para localización y mapeo, métodos directos y semi-directos, posteriormente, se analizan artículos del estado del arte en donde se utilizan algunos métodos de visión relacionados con el tema, algunas implementaciones y aplicaciones.

Se describe el sistema general, hardware y librerías utilizadas. Se describen los algoritmos desarrollados para filtrado de señales y eliminación de ruido en sensores que funcionan en conjunto con el algoritmo de visión, se explica con detalle el método directo implementado, se muestran resultados de experimentos en cinco circuitos diferentes donde se obtuvieron resultados con un error de 4% aproximadamente en superficies con altos contrastes y un error de hasta un 22% en superficies lisas con pocos contrastes, la velocidad del algoritmo es comparable con la visión humana, ya que es capaz de funcionar a 24 cuadros por segundo.

Abstract

This thesis reports research on direct and semi-direct methods for visual odometry. As a result, a computer vision algorithm based on direct methods is developed, capable of providing location information in real time and building a map in 3D. The algorithm has been used to obtain the location and trajectory mapping for a mobile robot.

The foundations include a review of the most recent and closely related works done at TecNM-CENIDET on direct methods, an overview of the mathematical models used for transformations in two and three dimensions, vision algorithms for localization and mapping using direct and semi-direct methods of the current state-of-the-art, and an analysis and discussion of their applications and implementation.

The next sections show the general system description, its hardware and software libraries. The algorithms developed for signal filtering of inertial sensors that work in conjunction with the vision algorithm, with results and experimentation. The algorithm is tested on five different circuits, the results obtained a 4% of minimum error on surfaces with high contrast and an error of up to 22% on smooth surfaces with little contrast, the algorithm is capable of working at 24 frames per second, which is comparable with human vision.

Índice

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

A continuación se presenta una introducción al tema de tesis, algunos antecedentes de trabajos realizados en TecNM-CENIDET (Tecnológico Nacional de México - Centro Nacional de Investigación y Desarrollo Tecnológico) una breve descripción del problema, los objetivos, propuesta de solución y la organización de la tesis.

1.1 Introducción

La visión robótica es un área multidisciplinaria que utiliza tanto la ciencia como la ingeniería. Se relaciona principalmente con la visión por computadora, la robótica, aprendizaje máquina o automático y el control. Utiliza información de sensores ópticos con algoritmos aplicados en sistemas de cómputo. Los sensores retroalimentan al sistema y dan como resultado una acción física que interactúa con el entorno [Owen-Hill, 2016].

El área de visión robótica tiene gran cantidad de retos a enfrentar, uno de ellos, es el problema de la localización de un vehículo en un entorno desconocido que al mismo tiempo necesita tener información del entorno que lo rodea. A esto se le conoce como localización y

mapeo simultáneo o SLAM *(Simultaneous Localization and Mapping)* por sus siglas en inglés. Por lo tanto, se le llama algoritmo o técnica de SLAM al encargado de resolver este problema.

Las técnicas de SLAM se pueden encontrar en diferentes áreas, por ejemplo, en aplicaciones de realidad aumentada [Schöps, 2014][Klein, 2007], robots autónomos [Faessler, 2015] [Forster, 2015], escáner y reconstrucción 3D [Newcombe, 2015]. Cuando se habla de una técnica de SLAM que resuelve el problema usando sensores ópticos como lo son las cámaras de video, se le conoce como una técnica de V-SLAM *(Visual Simultaneous Localization and Mapping)*[Scaramuzza, 2011][Mur-Artal, 2015].

Algo similar a lo que es el V-SLAM, es la odometría visual o VO *(Visual Odometry)* por sus siglas en inglés. La odometría visual es un proceso por el cual se puede obtener la egomoción o movimiento 3D aparente de un agente utilizando una o más cámaras. La diferencia de la odometría visual con V-SLAM es que con la odometría visual se obtiene la información de posición de manera local, minimizando y corrigiendo su error cada cierto número de imágenes o poses encontradas (*Windowed Bundle Adjustment*), aunque puede generar un mapa del entorno, no es la principal prioridad de la odometría visual, se podría decir que es un método simplificado del V-SLAM, por otro lado el V-SLAM construye un mapa con el menor error posible realizando una optimización global de las poses en el mapa (*Bundle Adjustment*) [Shum, 1999][Scaramuzza, 2011][Fraundorfer, 2012].

La clasificación de los algoritmos de visión por el número de cámaras utilizadas es:

- Monocular cuando se usa una sola cámara
- Binocular cuando se usan dos cámaras

En la mayoría de los casos no suelen ser más de dos cámaras, ya que con sólo dos cámaras es posible tener la información suficiente para mantener una buena precisión. Sin embargo, hay algunos algoritmos que utilizan más cámaras, como lo es la visión trinocular y el algoritmo "*Third-Eye Technique*" [Klette, 2014].

La forma en que muchos métodos de V-SLAM y VO funcionan, es utilizando la estructura por movimiento o SfM *(Structure from Motion).* La odometría visual es un caso particular de SfM, el cual simultáneamente recupera la estructura 3D junto con la pose de la cámara representada por la rotación y traslación. Esto se obtiene mediante la correspondencia de características y/o regiones o parches de pixeles.[Favaro, 2001] [Shum, 1999][Scaramuzza, 2011].

Una forma de clasificar los métodos de VO y V-SLAM, según si trabajan con características y/o parches de pixeles, son:

- Métodos basados en características.
- Métodos directos o métodos basados en apariencia.
- Métodos semi-directos o híbridos.

Este trabajo está enfocado en métodos directos y semi-directos funcionando en tiempo real, sin embargo, también se mostrará un poco de información teórica de los métodos basados en características [Irani, 1999][Favaro, 2001][Forster, 2014][Engel, 2014][Scaramuzza, 2011].

Aunque la odometría visual es una de las técnicas más precisas, también está la odometría inercial, donde algunos autores comparan los sensores inerciales con el sistema vestibular de los seres vivos [Scaramuzza, 2011][Luinge, 2002]. Al sensor inercial utilizado para esta tarea se le llama unidad de medición inercial o IMU (*Inertial Measurement Unit*) por sus siglas en inglés. Este sensor es frecuentemente utilizado y da buenos resultados, se puede complementar con algoritmos de visión [Scaramuzza, 2011][Scaramuzza, 2014][Stumberg, 2018][Forster, 2015][Faessler, 2015].

Un algoritmo de navegación visual-inercial o VIN (*Visual-Inertial Navegation*) por sus siglas en inglés, es la combinación de un método visual con la información inercial de una IMU. Los algoritmos Visión-Inercial demuestran ser más precisos que los algoritmos que sólo utilizan la visión, además de que permiten recuperar la escala. El problema de escala se presenta principalmente cuando se utiliza visión monocular y varios maneras de solucionarlo, por ejemplo, se puede iniciar observando un objeto conocido, se puede utilizar otros sensores como se había mencionado anteriormente u obtenerlo si se restringen algunos movimientos [Davison, 2003][Carlone, 2015][Stumberg, 2018][Scaramuzza, 2009].

Con el propósito de implementarlo en un robot móvil de bajo consumo energético, se utiliza un sistema embebido [Forster, 2014][Faessler, 2015]. Los sistemas embebidos se definen como sistemas de cómputo especializados para realizar funciones concretas y no generales, la interfaz humano-computador es mínima, en muchos casos realiza tareas de manera automática en tiempo real según los sensores vayan obteniendo nueva información. La ventaja de estos sistemas es que son reducidos, pueden obtener información de gran variedad de sensores y consumen poca energía, por lo que es hace idóneos para robots pequeños [Holt, 2014][Jiménez, 2014].

La implementación en un robot móvil permite evaluar su desempeño con procesamiento fuera de línea (*offline*) con información previamente obtenida (imágenes y valores de los

sensores) y en tiempo real. Se define tiempo real cuando los sensores están interactuando constantemente con el entorno y enviando información, en cuanto la información requerida llega al procesador, se ejecuta el algoritmo arrojará la información resultante deseada. Esto se debe realizar aproximadamente 30 veces por segundo, ya que generalmente las cámaras trabajan a esa velocidad [Davison, 2003]. Comparado con el ser humano, se dice que el cerebro puede procesar de 10 a 12 imágenes por segundo. En la historia cinematográfica se ha usado una velocidad de 16 a 24 cuadros por segundo, ya que se considera la velocidad idónea para tener la sensación de continuidad en un video [Read, 2000]. Esto es muy similar a la velocidad en la cual se considera como tiempo real.

Se considera principalmente utilizar métodos directos de visión debido a que en en el actual estado del arte han mostrado buenos resultados [Peidong, 2017][Raluca, 2017][Engel, 2016] [Forster, 2015][Faessler, 2015][Engel, 2014][Forster, 2014][Schöps, 2014][Newcombe, 2011].

1.2 Antecedentes

En TecNM-CENIDET, la visión robótica es un área ampliamente estudiada desde hace tiempo dentro del grupo de IA (Inteligencia Artificial). Un ejemplo de estos trabajos, es la tesis que tiene como título “Visión binocular para robot móvil en exteriores simulados”, donde se trabajó con visión estéreo, utilizó el método Bouguet para la calibración y rectificación, se evaluaron tres algoritmos de correspondencia por bloques implementados en OpenCV: BM (*Block Matching*), SGBM (*Semi-Global Block Matching*) y BT (*Birchfield-Tomasi*). Estos métodos por bloques tienen cierta similitud con los métodos directos por el uso de parches de pixeles, que en este caso lo llaman bloques. En su trabajo, se construyó un robot móvil con dos motores que con éxito pudo navegar evadiendo obstáculos en “exteriores simulados”. Dentro de sus trabajos futuros quedó pendiente realizar SLAM y la implementación de VO [González, 2011].

Un trabajo más reciente es la tesis “Navegación, Localización y Mapeo de Robots Móviles para Trayectorias Pre-especificadas por Imágenes”, donde se diseñó un sistema binocular que por medio de una secuencia de imágenes, permite generar un mapa de trayectorias con la librería LIBVISO2 y ubicarse dentro del mismo. Con esta información le es posible al robot móvil ir navegando y reconocer lugares explorados por medio de las características en las imágenes. Utilizó la librería OpenRobotino para el robot FESTO y OpenCV para procesar las imágenes. Desarrolló un método de VO y entre sus aportaciones está la recuperación de la pose de una escena antes vista. Este trabajo es uno de los pocos donde se hace un mapa con el cual se puede apreciar la trayectoria del robot, también brinda información interesante sobre las librerías que utilizó para generar el mapa [Vergara, 2015] .

La tesis “Experimentación con Odometría Visual por Métodos Directos” , utilizó por primera

vez métodos de visión directos para la odometría visual y la generación de un mapa 2D. Por medio transformaciones en secuencias de imágenes, se pudo encontrar la homografía y obtener información de pose por cada par de imágenes. Para encontrar la transformación óptima se utilizaron varios métodos, sin embargo, uno que destaca y es muy mencionado en la literatura, es el método de Gauss-Newton. Todo esto con funciones de OpenCV. Experimentó con varias secuencias de imágenes conocidas y otras propias, su principal aportación fue la utilización de métodos directos, de los cuales no se habían trabajado anteriormente en TecNM-CENIDET y aunque da buenos resultados, su enfoque es demasiado complejo y no es posible implementarlo para que funcione en tiempo real [Vela, 2017].

Por último se tiene la tesis “Sistema de Navegación Inercial Asistido por Visión para Robots Móviles Terrestres” utilizó una unidad de medición inercial para determinar la posición del robot, sin embargo dado a la deriva que tiene la doble integración de la aceleración, se tuvo que auxiliar de la visión para resolver este problema. Al igual que en este trabajo, utilizó sistemas embebidos como *Arduino* y *Raspberry Pi 3* para capturar la información inercial y procesarla junto con las imágenes capturadas por la cámara, aunque sus resultados fueron variados, se demostró que ambos sensores se complementan muy bien y pueden dar buenos resultados. Para el algoritmo de visión utilizó métodos basados en características para obtener el flujo óptico y determinar el movimiento de la cámara [Navarrete, 2018].

1.3 Descripción del problema

El problema es la implementación de un algoritmo basado en métodos directos en un robot móvil, donde el algoritmo debe ser rápido y eficiente como para ser implementado para la navegación robótica en tiempo real.

En comparación con trabajos que se están realizando en TecNM-CENIDET, no se había tomado en cuenta la generación de mapas 3D con métodos directos, ni su aplicación para la navegación robótica en tiempo real. Encontrar la pose por medio de la homografía con métodos directos, tiene un alto costo computacional [Vela, 2017].

Dentro del problema también está el desarrollo del software en un sistema embebido, el cual tiene recursos limitados de procesamiento.

1.4 Objetivos

Desarrollar e implementar un sistema basado en métodos directos que pueda ser utilizado para la navegación robótica.

- Describir métodos directos utilizados en navegación robótica.
- Desarrollar un método directos que pueda ejecutarse en tiempo real.
- Desarrollar el software bajo un sistema embebido.
- Experimentar con el sistema desarrollado.

1.5 Propuesta de solución

Para la solución, se propone lo siguiente:

- Simplificar la transformación para la alineación de las imágenes.
- Utilizar el algoritmos de optimización de bajo costo computacional.
- Obtener información adicional de una IMU y combinarla con el sistema de visión para reducir costo computacional y mejorar resultados.

1.6 Organización de la tesis

El documento de tesis está compuesto por siete capítulos.

En el capítulo uno, se muestra información general para ponerse en contexto con el problema. Se da a conocer antecedentes de CENIDET que se relacionan con el tema, se describe el problema, los objetivos de la tesis y la propuesta de solución.

En el capítulo dos, se presentan las bases teóricas, como lo son la teoría de grupos, topología algebraica, diferentes modelos para la representación de rotaciones, conceptos de SLAM y VO. De forma detallada muestra información de los métodos directos, semi-directos y basados en características, algoritmos de optimización que serán útiles posteriormente y conceptos de filtrado de señales.

En el capítulo tres, se puede ver diferentes algoritmos en el estado del arte, tanto de métodos directos, como semi-directos y basados en características. Se presentan algunas aplicaciones de los algoritmos y al final se tiene la comparación de los algoritmos.

Dentro del capítulo cuatro está toda la información del sistema que se va a emplear para la experimentación, la forma en que interactúan las diferentes partes, el tipo de robot y la tarjeta utilizada para el sistema embebido, los sensores, las herramientas y librerías que se utilizaron para el desarrollo del software.

En el capítulo cinco, se detalla el algoritmo que se desarrolló, la forma en que selecciona la información dentro del mapa de bits y el algoritmo de seguimiento utilizado. Por último se expone el procedimiento empleado para combinar la información visual con la inercial.

Dentro del capítulos seis, están los experimentos y pruebas realizadas. Se mide y compara el poder de cómputo de una computadora convencional con procesador core i5 y los sistemas embebidos utilizados con procesadores de arquitectura ARM. Se hacen pruebas de los algoritmos y se evalúa su desempeño en funcionamiento en diferentes entornos, por último se compara con algunos algoritmos mostrados anteriormente en el estado del arte y se hacen algunas pruebas del sistema completo en tiempo real.

Por último, en el capítulo siete se tienen las conclusiones y trabajos futuros, en donde se aportarán algunos puntos de vista en relación a los resultados obtenidos.

Capítulo 2: Marco teórico

En este capítulo se describirán las bases teóricas para comprender mejor los capítulos restantes, principalmente se mostrarán modelos matemáticos con grupos y álgebra de Lie (información más detallada en Anexo A), éstos son usados para transformaciones en objetos que se desplazan y rotan en entornos de tres dimensiones como un robot móvil, también se puede aplicar a objetos de dos dimensiones, como las imágenes pixeladas. Se describirán algoritmos de optimización, estos son útiles para resolver una gran variedad de problemas como lo es la alineación de imágenes, mejoramiento de rutas y trayectorias, encontrar trasformaciones para la pose de un robot, calibración de cámaras de video, etc. Por último, se describirán conceptos y tipos de odometría visual, por ejemplo, la visión monocular y binocular, métodos directos, basados en características y semi-directos.

2.1 Transformaciones con matrices y grupos de Lie

Como ya se había mencionado antes, muchos de los grupos de Lie representan

transformaciones es espacios euclidianos y cada uno de los elementos del grupo de Lie es una matriz. A continuación se explicará con más detalle cada uno de los grupos de Lie útiles para realizar esta tarea, la siguiente tabla muestra de forma compacta los grupos de Lie utilizados para transformaciones [Eade, 2013][Eade, 2014].

Tabla 2.1: Grupos de Lie para transformaciones

Grupo	Descripción	Nombre del grupo	Dimensión
$\boldsymbol{SO}(2,\mathbb{R})$	Rotaciones 2D	Especial Ortogonal 2D	1
$\boldsymbol{SO}(3,\mathbb{R})$	Rotaciones 3D	Especial Ortogonal 3D	3
$\mathbb{H}_1$	Rotaciones 3D	Cuaternio	4
$\boldsymbol{SE}(2,\mathbb{R})$	Transformación de cuerpo rígido 2D	Especial Euclidiano	3
$\boldsymbol{SE}(3,\mathbb{R})$	Transformación de cuerpo rígido 3D	Especial Euclidiano	6
$\mathbf{Sim}(2,\mathbb{R})$	Transformación similar 2D	Similar	4
$\mathbf{Sim}(3,\mathbb{R})$	Transformación similar 3D	Similar	7
$\boldsymbol{Aff}(2,\mathbb{R})$	Transformación Afín 2D	Afín	6
$\boldsymbol{Aff}(3,\mathbb{R})$	Transformación Afín 3D	Afín	10
$\boldsymbol{SL}(3,\mathbb{R})$	Homografía	Especial Lineal	8

2.1.1 Rotaciones en SO(n)

El grupo **SO**(n,$\mathbb{R}$) es el grupo especial ortogonal real asociada al álgebra de Lie $\mathfrak{so}$(n,$\mathbb{R}$), representa las matrices de rotación para *n* dimensiones. Las matrices **R** que pertenecen a este grupo de Lie cumplen con lo siguiente:

$$R \in \boldsymbol{SO}(n) \qquad \textit{(Ecuación: 2.1)}$$

$$R^t = R^{-1} \qquad \textit{(Ecuación: 2.2)}$$

$$det(R) = 1 \qquad \textit{(Ecuación: 2.3)}$$

Para rotaciones en 2D se utiliza el grupo **SO**(2,$\mathbb{R}$), sin embargo, utilizando una notación más simplificada se usará como **SO**(2), ya que todas las rotaciones serán en espacios reales. La estructura de las matrices **A** que pertenece a este grupo se presenta de la siguiente forma:

$$A = \begin{bmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{bmatrix} \qquad \textit{(Ecuación: 2.4)}$$

Donde:

$$A \in \boldsymbol{SO}(2) \subset \mathbb{R}^{2x2} \qquad \textit{(Ecuación: 2.5)}$$

$$A^t = A^{-1} \qquad \textit{(Ecuación: 2.6)}$$

$$det(A) = 1 \qquad \textit{(Ecuación: 2.7)}$$

Para rotaciones en 3D se utiliza el grupo **SO**(3,$\mathbb{R}$), sin embargo, de igual manera que en el grupo anterior, se usará una notación más simplificada por lo que su notación será **SO**(3). En este grupo hay diferentes modelos para representar las rotaciones, a continuación se verá detalladamente cada uno [Eade, 2013][Eade, 2014][Bronshtein, 2015].

Rotación sobre un eje:

En este modelo se presentan tres estructuras diferentes, cada una permite un giro sobre un eje específico. La matriz de rotación $\mathbf{R_x}$, permite girar sobre el eje **x** [Bronshtein, 2015]:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad \textit{(Ecuación: 2.8)}$$

$\mathbf{R_y}$ es la rotación sobre el eje **y**:

$$R_y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \qquad \textit{(Ecuación: 2.9)}$$

$\mathbf{R_z}$ es la rotación sobre el eje **z**:

$$R_z(\psi)=\begin{bmatrix}\cos(\psi) & -\sin(\psi) & 0\\ \sin(\psi) & \cos(\psi) & 0\\ 0 & 0 & 1\end{bmatrix} \qquad \textit{(Ecuación: 2.10)}$$

Ángulos de Cardán

Los ángulos de cardán son un modelo para rotaciones que se describen de la siguiente forma:

$$R(\theta,\phi,\psi)=\begin{bmatrix}\cos(\phi)\cos(\psi) & -\sin(\psi)\cos(\psi)+\cos(\psi)\sin(\phi)\sin(\theta) & \sin(\psi)\sin(\theta)+\cos(\psi)\sin(\phi)\cos(\theta)\\ \sin(\psi)\cos(\phi) & \cos(\psi)\cos(\theta)+\sin(\psi)\sin(\phi)\sin(\theta) & -\cos(\psi)\sin(\theta)+\sin(\psi)\sin(\phi)\cos(\theta)\\ -\sin(\phi) & \cos(\phi)\sin(\theta) & \cos(\phi)\cos(\theta)\end{bmatrix}$$

(Ecuación: 2.11)

Si se observan las ecuaciones de la rotación sobre un eje, se puede apreciar que el modelo de ángulos de Cardán es la composición de las matrices anteriores.

$$R(\theta,\phi,\psi)=R_z(\psi)R_y(\phi)R_x(\theta) \qquad \textit{(Ecuación: 2.12)}$$

También hay que tener en cuenta que el producto de las matrices no es conmutativo, por lo cual:

$$R_x(\theta)R_y(\phi)R_z(\psi)\neq R_z(\psi)R_y(\phi)R_x(\theta) \qquad \textit{(Ecuación: 2.13)}$$

Una desventaja de este modelo es el llamado bloqueo de Cardán que sucede en el siguiente caso:

$$R(\theta,90°,\psi)=\begin{bmatrix}0 & \sin(\theta-\psi) & \cos(\theta-\psi)\\ 0 & \cos(\theta-\psi) & \sin(\theta-\psi)\\ -1 & 0 & 0\end{bmatrix} \qquad \textit{(Ecuación: 2.14)}$$

Cuando se tiene un resultado similar a la ecuación anterior, el resultado muchas veces no es el esperado debido a la pérdida de un grado de libertad [Bronshtein, 2015].

Ángulos de Euler

Los ángulos de Euler son otro modelo muy utilizado, aunque muchas veces se toman a los ángulos de Cardán como ángulos de Euler, su definición es diferente, la matriz de rotación para ángulos de Euler es:

$$R(\theta,\phi,\psi)=\begin{bmatrix}\cos(\psi)\cos(\theta)-\sin(\psi)\cos(\phi)\sin(\theta) & -\cos(\psi)\sin(\theta)-\sin(\psi)\cos(\phi)\cos(\theta) & \sin(\psi)\sin(\phi)\\ \sin(\psi)\cos(\phi)+\cos(\psi)\cos(\phi)\sin(\theta) & -\sin(\psi)\sin(\theta)+\cos(\psi)\cos(\phi)\cos(\theta) & -\cos(\psi)\sin(\phi)\\ \sin(\phi)\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\end{bmatrix}$$

(Ecuación: 2.15)

La diferencia se puede apreciar en la siguiente ecuación [Bronshtein, 2015]:

$$R(\theta,\phi,\psi)=R_z(\psi)R_x(\phi)R_z(\theta)$$ *(Ecuación: 2.16)*

Rotación en torno a un eje arbitrario

La rotación utilizando este modelo se hace mediante un vector normalizado y un ángulo. El vector se define de la siguiente forma:

$$\vec{\boldsymbol{a}}=(a_x,a_y,a_z)$$ *(Ecuación: 2.17)*

$$|\vec{\boldsymbol{a}}|=1$$ *(Ecuación: 2.18)*

Las matrices utilizadas para este modelo son:

$$R_1=\begin{bmatrix}\frac{a_x}{\sqrt{a_x^2+a_z^2}} & 0 & \frac{a_z}{\sqrt{a_x^2+a_z^2}}\\ 0 & 1 & 0\\ \frac{-a_z}{\sqrt{a_x^2+a_z^2}} & 0 & \frac{a_x}{\sqrt{a_x^2+a_z^2}}\end{bmatrix}$$ *(Ecuación: 2.19)*

$$R_2=\begin{bmatrix}\sqrt{a_x^2+a_z^2} & a_y & 0\\ -a_y & \sqrt{a_x^2+a_z^2} & 0\\ 0 & 0 & 1\end{bmatrix}$$ *(Ecuación: 2.20)*

$$R_3=R(\gamma)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\gamma) & -\sin(\gamma)\\ 0 & \sin(\gamma) & \cos(\gamma)\end{bmatrix}$$ *(Ecuación: 2.21)*

Finalmente, la composición de la matriz de rotación es:

$$R(\gamma,\vec{\boldsymbol{a}})=R_1^{-1}R_2^{-1}R_3R_2R_1$$ *(Ecuación: 2.22)*

Da como resultado:

$$R(\gamma,\vec{\boldsymbol{a}})=\begin{bmatrix}\cos(\gamma)+a_x^2(1-\cos(\gamma)) & a_x a_y(1-\cos(\gamma))-a_z\sin(\gamma) & a_x a_z(1-\cos(\gamma))+a_y\sin(\gamma)\\ a_y a_x(1-\cos(\gamma))+a_z\sin(\gamma) & \cos(\gamma)+a_y^2(1-\cos(\gamma)) & a_y a_z(1-\cos(\gamma))-a_x\sin(\gamma)\\ a_z a_x(1-\cos(\gamma))-a_y\sin(\gamma) & a_z a_y(1-\cos(\gamma))+a_x\sin(\gamma) & \cos(\gamma)+a_z^2(1-\cos(\gamma))\end{bmatrix}$$

(Ecuación: 2.23)

La matriz resultante también es del grupo de Lie ortogonal por lo que [Bronshtein, 2015]:

$$R^t(\gamma,\vec{\boldsymbol{a}})=R^{-1}(\gamma,\vec{\boldsymbol{a}})$$ *(Ecuación: 2.24)*

2.1.2 Cuaternios

Los cuaternios tienen la misma notación para su grupo de Lie y para su álgebra, se representa como $\mathbb{H}_1$. Además de ser un grupo de Lie, son una extensión de los números complejos y también son llamados números de Hamilton o números hiper-complejos. Están compuestos de la siguiente manera:

$$q_0+q_1\boldsymbol{i}+q_2\boldsymbol{j}+q_3\boldsymbol{k} \quad \in \quad \mathbb{H}_1$$ *(Ecuación: 2.25)*

Donde **i** , **j** y **k** son números similares a los números imaginarios. Cuando se tiene un cuaternio sin la parte real $\mathbf{q_0}$, se le puede denominar "cuaternio puro" y se representa como $\mathbb{H}_0$:

$$q_1\boldsymbol{i}+q_2\boldsymbol{j}+q_3\boldsymbol{k} \quad \in \quad \mathbb{H}_0$$ *(Ecuación: 2.26)*

Sin embargo, en este documento se usará en todo momento los cuaternios $\mathbb{H}_1$, por lo que se eliminará el subíndice y se mostrará de manera simplificada como $\mathbb{H}$.

La relación que tiene la parte imaginaria de estos números es la siguiente:

$$\boldsymbol{i}^2=\boldsymbol{j}^2=\boldsymbol{k}^2=\boldsymbol{ijk}=-1$$ *(Ecuación: 2.27)*

$$\boldsymbol{ij}=\boldsymbol{k}=-\boldsymbol{ji}$$ *(Ecuación: 2.28)*

$$\boldsymbol{jk}=\boldsymbol{i}=-\boldsymbol{kj}$$ *(Ecuación: 2.29)*

$$\boldsymbol{ki}=\boldsymbol{j}=-\boldsymbol{ik}$$ *(Ecuación: 2.30)*

Representado en forma de matriz es:

$$R(\gamma,\underline{a})=\begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2q_1q_2-2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 2q_1q_2+2q_0q_3 & q_0^2-q_1^2+q_2^2-q_3^2 & 2q_2q_3-2q_0q_2 \\ 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \quad \textit{(Ecuación: 2.31)}$$

En donde:

$$q_0=\frac{\cos(\gamma)}{2} \quad \textit{(Ecuación: 2.32)}$$

$$q_1=\frac{\sin(\gamma)}{2} \quad \textit{(Ecuación: 2.33)}$$

$$q_2=\frac{\sin(\gamma)}{2} \quad \textit{(Ecuación: 2.34)}$$

$$q_3=\frac{\sin(\gamma)}{2} \quad \textit{(Ecuación: 2.35)}$$

En los cuaternios sólo se puede rotar sobre un eje a la vez y esto se especifica en:

$$\underline{a}=\begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix} \quad \textit{(Ecuación: 2.36)}$$

En los valores $\mathbf{q_1}$, $\mathbf{q_2}$ y $\mathbf{q_3}$, solo uno de ellos debe ser considerado para rotación, mientras que los demás se toman como cero. Cada uno de los números representa un eje que puede ser utilizado para rotación [Bronshtein, 2015].

2.1.3 Transformaciones en SE(n)

El grupo de Lie **SE**(n,ℝ) contiene todas las matrices que representan trasformaciones de cuerpos rígidos y se asocia con el álgebra (n, ℝ. Como ya se había hecho antes, se simplificará la notación representando el grupo como **SE**(n). Este grupo está compuesto por una matriz de rotación del grupo **SO**(n) y con un vector de traslación $\mathbb{R}^n$. Considerando la matriz **A** como un elemento del grupo **SE**(n), entonces:

$$\boldsymbol{A}=\begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \\ 0 & 1 \end{bmatrix}\in \boldsymbol{SE}(n) \quad \textit{(Ecuación: 2.37)}$$

Donde:

$$\boldsymbol{R}\in \boldsymbol{SO}(n) \quad \textit{(Ecuación: 2.38)}$$

$$\boldsymbol{T}\in \mathbb{R}^n \quad \textit{(Ecuación: 2.39)}$$

Suponiendo que se quiere realizar una trasformación de cuerpo rígido de dos dimensiones sobre un objeto, se utilizaría una matriz **B** con la siguiente forma:

$$\boldsymbol{B} \in \boldsymbol{SE}(2) \qquad \textit{(Ecuación: 2.40)}$$

$$\boldsymbol{B} = \begin{bmatrix} \cos(\theta) & -sen(\theta) & x \\ sen(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \qquad \textit{(Ecuación: 2.41)}$$

Para transformaciones de cuerpo rígido de tres dimensiones se hace de la misma manera, solo que se utiliza para la rotación **R** una matriz de tipo **SO**(3) de cualquier modelo (ángulos de Euler, ángulos de Cardán, etc.) y un vector de tres dimensiones con **x**, **y** y **z** [Bronshtein, 2015] [Eade, 2013][Eade, 2014].

2.1.4 Transformaciones en Sim(n)

El grupo de Lie de transformación similar **Sim**(n,ℝ) con su álgebra asociada $\mathfrak{sim}$(n, ℝ), de igual manera, se usará como **Sim**(n). Los elementos que componen este grupo son matrices de trasformación capaces de realizar cambio de escala, rotaciones y traslaciones. Se asemeja mucho con matrices del grupo **SE**(n), sin embargo se le añade el factor de escala $\mathbf{s^{-1}}$. Una matriz **A** que pertenece al grupo **Sim**(n), tendría la siguiente forma [Eade, 2013][Eade, 2014]:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \\ 0 & s^{-1} \end{bmatrix} \in \mathbf{Sim}(n) \qquad \textit{(Ecuación: 2.42)}$$

Donde:

$$\boldsymbol{R} \in \boldsymbol{SO}(n) \qquad \textit{(Ecuación: 2.43)}$$

$$\boldsymbol{T} \in \mathbb{R}^n \qquad \textit{(Ecuación: 2.44)}$$

2.1.5 Transformaciones en Aff(n)

El grupo de Lie para transformaciones afines **Aff**(n,ℝ) con su álgebra $\mathfrak{aff}$(n, ℝ), el cual abreviaremos a **Aff**(n), está formado de elementos que son matrices que representan transformaciones de rotación con traslación, escala y sesgo. Para añadir sesgo o inclinaciones, la matriz de trasformación para objetos 2D debe ser de la forma [Eade, 2013] [Eade, 2014]:

$$\boldsymbol{R} \in \boldsymbol{SO}(2) \qquad \textit{(Ecuación: 2.45)}$$

$$\boldsymbol{R}=\begin{bmatrix}\cos(\theta) & -sen(\theta)\\ sen(\theta) & \cos(\theta)\end{bmatrix} \qquad \textit{(Ecuación: 2.46)}$$

La matriz **A** representa rotación y sesgo, puede ser de dos formas:

$$\boldsymbol{A}=\begin{bmatrix}1 & a\\ b & 1\end{bmatrix}\begin{bmatrix}\cos(\theta) & -sen(\theta)\\ sen(\theta) & \cos(\theta)\end{bmatrix} \qquad \textit{(Ecuación: 2.47)}$$

$$\boldsymbol{A}=\begin{bmatrix}\cos(\theta) & -sen(\theta)\\ sen(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}1 & a\\ b & 1\end{bmatrix} \qquad \textit{(Ecuación: 2.48)}$$

Teniendo en cuenta que el producto no es conmutativo, puede ser de dos maneras. Los valores **a** y **b** de la ecuación anterior son los valores para sesgar o inclinar.

Por último, se añade la traslación y escala, para que la matriz **B** resultante, sea de la forma:

$$\boldsymbol{B}=\begin{bmatrix}\boldsymbol{A} & \boldsymbol{T}\\ 0 & s^{-1}\end{bmatrix}\in \boldsymbol{Aff}(2) \qquad \textit{(Ecuación: 2.49)}$$

$$\boldsymbol{T}\in\mathbb{R}^2 \qquad \textit{(Ecuación: 2.50)}$$

Se hace algo muy parecido para obtener matrices de trasformación afín 3D, solo que en vez de usar **SO**(2), se utilizará **SO**(3), multiplicado o pre-multiplicado por una matriz así:

$$\boldsymbol{A}=\begin{bmatrix}1 & a & c\\ b & 1 & e\\ d & f & 1\end{bmatrix}\boldsymbol{R} \qquad \textit{(Ecuación: 2.51)}$$

o así:

$$\boldsymbol{A}=\boldsymbol{R}\begin{bmatrix}1 & a & c\\ b & 1 & e\\ d & f & 1\end{bmatrix} \qquad \textit{(Ecuación: 2.52)}$$

Donde:

$$\boldsymbol{R}\in\boldsymbol{SO}(3) \qquad \textit{(Ecuación: 2.53)}$$

Ahora se tiene como valores de inclinación **a**, **b**, **c**, **d**, **e** y **f**. Por último, se procede a completar la matriz **B** con la escala y traslación:

$$\boldsymbol{B}=\begin{bmatrix}\boldsymbol{A} & \boldsymbol{T}\\ 0 & s^{-1}\end{bmatrix}\in \boldsymbol{Aff}(3) \qquad \textit{(Ecuación: 2.54)}$$

$$\boldsymbol{T}\in\mathbb{R}^3 \qquad \textit{(Ecuación: 2.55)}$$

2.1.6 Grupo SL(n)

El grupo de Lie especial lineal **SL**(n, ℝ) asociadas al álgebra $\mathfrak{sl}$(n, ℝ), está integrado por matrices cuadradas con determinante igual a uno. Como se ha hecho anteriormente, se utilizará la notación simplificada del grupo, que es **SL**(n). Específicamente lo que interesa de este grupo es cuando tiene dimensión tres, ya que **SL**(3) representa entre otras cosas, las homografías en 2D. La homografía tiene 8 grados de libertad que incluyen rotación, traslación, escala, inclinación o sesgo y cambio de perspectiva. En sus subgrupos está el grupo **Aff**(2) y **SO**(3). De manera que [Eade, 2013][Eade, 2014]:

$$\boldsymbol{H}\in\boldsymbol{SL}(3)\subset\mathbb{R}^{3x3} \qquad \textit{(Ecuación: 2.56)}$$

$$det(\boldsymbol{H})=1 \qquad \textit{(Ecuación: 2.57)}$$

2.2 Algoritmos de optimización

La optimización de un problema es la tarea de buscar los valores para satisfacer determinadas expectativas. Para esto se necesita una función objetivo la cual es utilizada para medir la idoneidad de la posible solución. El dominio de la función es el conjunto de posibles soluciones y recibe el nombre de espacio de soluciones.

El rango de la función generalmente es un escalar que se utiliza para medir el resultado. Dependiendo del problema, la tarea de optimización puede tratarse de minimizar o maximizar el resultado de la función.

Al valor que parece ser una posible solución que no es la mejor, pero aún así parece buena comparada con sus vecinos o valores adyacentes, se le conoce como óptimo local, por otro lado, a la mejor solución de todas, se le llama óptimo global.

Según el algoritmo de optimización puede clasificarse como:

Métodos analítico

Consisten en realizar alguna transformación analítica de la función objetivo para determinar los puntos de interés.

Métodos matemáticos iterativos

Estos métodos aplican una operación repetidamente hasta que se cumple un criterio determinado. Algunos ejemplos son el método de Newton y el método de ascenso (o descenso) de gradientes.

Métodos metaheurísticos

Se denomina heurísticos a las técnicas basadas en la experiencia o en analogías con otros procesos que resultan útiles para resolver un problema. En el área de la optimización, los heurísticos son técnicas que suelen ayudar a optimizar un problema, habitualmente tomando soluciones aleatorias y mejorándolas progresivamente mediante el heurístico en cuestión.

El prefijo "meta" de los métodos metaheurísticos se debe a que tales heurísticos son aplicables a cualquier problema, pues los métodos no presuponen nada sobre el problema que pretenden resolver y por tanto son aplicables a una gran variedad de problemas. En general se hablará de "metaheurístico" o simplemente de "método" para hablar de cada técnica en general, y de "algoritmo" para la aplicación de una técnica a un problema concreto, si bien por motivos históricos esta nomenclatura no siempre se sigue (como en el caso de los algoritmos genéticos).

Los métodos metaheurísticos no garantizan la obtención de la solución óptima, pero suelen proporcionar soluciones aceptablemente buenas para problemas inabordables con técnicas analíticas o más exhaustivas [Benítez, 2013].

2.2.1 Algoritmo de optimización por enjambre de partículas

El método de optimización con enjambres de partículas es un algoritmo metaheurístico y debe su nombre a que utiliza un conjunto de soluciones candidatas que se comportan como partículas moviéndose por un espacio: el espacio de soluciones. El movimiento de las partículas se determina por su velocidad, que depende de la posición del mejor punto que ha encontrado cada una de ellas en su recorrido y del mejor punto encontrado por el enjambre. [Benítez, 2013]

Se define por las siguientes ecuaciones:

$$v_{k+1}^{i} = \omega v_{k}^{i} + \alpha \gamma \left(p_{k}^{i} - x_{k}^{i} \right) + \beta \gamma \left(p_{k}^{g} - x_{k}^{i} \right) \qquad \textit{(Ecuación: 2.58)}$$

$$x_{k+1}^{i} = x_{k}^{i} + v_{k}^{i} \qquad \textit{(Ecuación: 2.59)}$$

Donde:

v = Velocidad

i = Índice de la partícula

k = Pasos en el tiempo

$\alpha \cdot$ = Cognitiva

$\gamma \cdot$ = Valor aleatorio

p^{i} = Mejor posición local de la partícula

$\beta \cdot$ = Social

p^{g} = Mejor posición global

x = Posición de la partícula

$\omega \cdot$ = Inercia

En el algoritmo de optimización por enjambre de partículas inicialmente se selecciona un número de partículas **i**, su posición inicial **x** puede ser aleatoria o si se tiene una noción de el óptimo global, se pueden posicionar partículas en puntos estratégicos. Para determinar la nueva posición de las partículas en necesario calcular la velocidad de las partículas, donde hay varios factores importantes, por ejemplo, el valor *social* es el encargado de darle mayor peso a la partícula mejor posicionada, lo que tiene como consecuencia que las demás partículas tiendan a acercarse a esta, el valor de *cognitiva* le da más peso a la mejor posición cercana a la partícula. Los valores de *cognitiva* y *social* son perturbados con números aleatorios que tienen como objetivo salir de óptimos locales, por otra parte, el valor de *inercia* tiene como objetivo hacer que las partículas tengan un aumento o descenso en su velocidad [Velázquez, 2016].

2.2.2 Algoritmo de optimización por descenso del gradiente

Este algoritmo es un método iterativo matemático utilizado para problemas de optimización no lineales. Consiste en tener una solución aproximada al problema, esa solución es la

posición **x**, posteriormente, en la siguiente iteración **k**, la posición se actualiza dada la siguiente ecuación:

$$x_{k+1} = x_k - \lambda_k \nabla f(x_k) \qquad \textit{(Ecuación: 2.60)}$$

En donde **λ** es un valor que puede acelerar la convergencia, este valor se puede omitir dándole el valor de uno, sin embargo, si se opta por usar este valor se puede asignar un valor mayor a uno para acelerar o menor a uno para hacer una búsqueda más fina, por otra parte, también se puede fabricar una estrategia para variar el valor en cada iteración, normalmente la estrategia es disminuir el valor en cada iteración hasta hacerlo cercano a cero. El valor de **∇***f(x)*, es el gradiente de la función dedicado a medir las variaciones de la función en la dirección del vector. Se conforma de las derivadas parciales de la siguiente forma:

$$\nabla f(x_n) = \frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \qquad \textit{(Ecuación: 2.61)}$$

Si el algoritmo no converge con el óptimo, siempre se puede usar **λ** con un valor muy pequeño, como ya se había mencionado anteriormente, Sin embargo, si el valor es muy bajo, puede alentar la convergencia ya que deberá dar muchos pasos para llegar al óptimo [Madsen, 2004][Bronshtein, 2015].

2.2.3 Algoritmo de optimización de Gauss-Newton

Al igual que el algoritmo de descenso del gradiente, también es un método matemático iterativo para problemas de optimización no lineal. En este algoritmo como en los anteriores, inicialmente se posiciona con valores cercanos a la solución, la posición **x** se actualiza en cada iteración **k** según la forma:

$$x_{k+1} = x_k - \boldsymbol{H}^{-1} \boldsymbol{J}^T \qquad \textit{(Ecuación: 2.62)}$$

En donde **J** es una matriz Jacobiana formada de derivadas parciales de primer orden de la función objetivo:

$$\boldsymbol{J_{ij}} = \frac{\partial f_i}{\partial x_j} \qquad \textit{(Ecuación: 2.63)}$$

Por otra parte **H** es una matriz Hesiana. Conformada por derivadas parciales de segundo order de la función objetivo:

$$\boldsymbol{H_{ij}} = \frac{\partial^2 f}{\partial x_i \partial x_j} \qquad \textit{(Ecuación: 2.64)}$$

Sin embargo, es posible simplificar el proceso utilizando la siguiente aproximación:

$$\boldsymbol{H} \approx 2\boldsymbol{J}^{T}\boldsymbol{J} \qquad \textit{(Ecuación: 2.65)}$$

Este algoritmo garantiza una convergencia pero en ocasiones puede ser más lento que el método de descenso del gradiente. [Madsen, 2004][Bronshtein, 2015][Engel, 2014]

2.2.4 Algoritmo de optimización de Levenberg-Marquardt

Este algoritmo también es iterativo matemático para resolver problemas de optimización no lineal. El algoritmo de Levenberg-Marquardt también llamado LM, intenta aprovechar la aceleración para la convergencia de λ junto con el paso que garantiza la convergencia de Gauss-Newton. La actualización de la posición **x** está dada por la forma:

$$x_{k+1} = x_k - \left(\boldsymbol{H} + \lambda_k \nabla f(x_k)\right)^{-1}\boldsymbol{J}^{T} \qquad \textit{(Ecuación: 2.66)}$$

La estrategia es hacer pequeño λ en cada iteración para garantizar alcanzar el valor óptimo. [Madsen, 2004][Bronshtein, 2015][Lourakis, 2005]

2.3 Odometría visual

La odometría visual es el proceso incremental utilizado para obtener la posición de una o más cámaras que se mueven en un vehículo. Comparado con otros tipos de odometría, esta suele ser muy precisa. Por ejemplo, si se calcula la odometría en base al movimiento de las ruedas de un vehículo, las ruedas pueden derrapar o no dar el resultado deseado debido al terreno en el que se desplaza. En vehículos aéreos, el GPS suele tener problemas de comunicación en lugares cerrados, además de tener un margen de error alto para aplicaciones en robótica [Scaramuzza, 2011].

2.3.1 Visión binocular

A los algoritmos de odometría visual que utilizan dos cámaras de video se les conoce como binoculares, estos tipos de algoritmos aprovechan la línea base **B** conocida entre las dos cámaras para obtener información 3D del entorno.

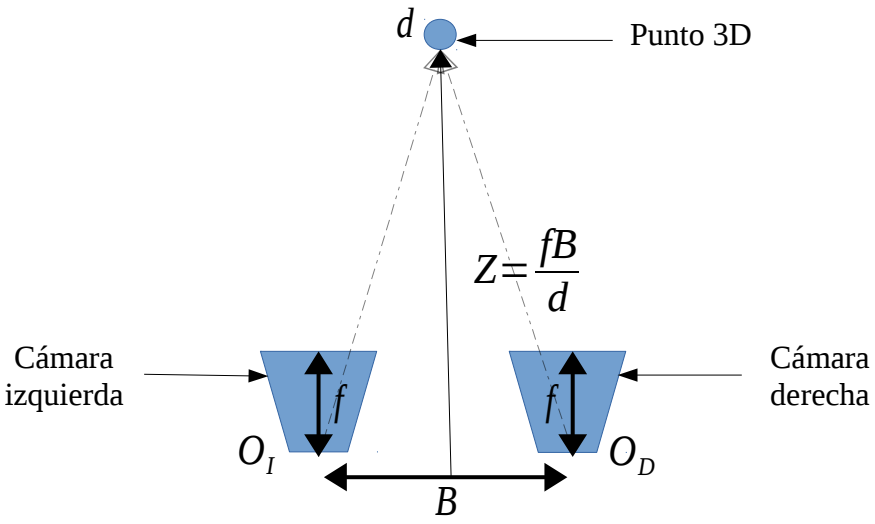


Figura 2.1: Visión binocular.

La línea base **B** es la distancia que hay entre las dos cámaras, cada cámara tiene un sensor, cuyo centro el es la posición de origen **O**, la distancia que hay del sensor al lente de la cámara es la distancia focal **f**, que no será de ayuda para obtener la escala del mundo real, ambas cámaras son capaces de ver un mismo punto que tiene una distancia de disparidad **d**, una vez que se tiene toda esta información, se puede saber la distancia de profundidad **Z** que hay del centro de la línea base a cualquier punto de la escena vista por ambas cámaras.

Para obtener la disparidad **d** es necesario hacer una correspondencia entre las imágenes del par binocular, esto se hace utilizando características o alineando las imágenes o parches de las imágenes, a continuación se muestran dos imágenes del par binocular sobrepuestas, donde las flechas indican la disparidad que hay entre las imágenes:

Figura 2.2: Disparidad binocular

La disparidad entre las imágenes de un par binocular proporcionan algunas ventajas, ya que si las cámaras están alineadas, no se tendrá que hacer una búsqueda por toda la imagen para encontrar correspondencias, bastará con buscar sobre la línea epipolar, como se muestra a continuación:

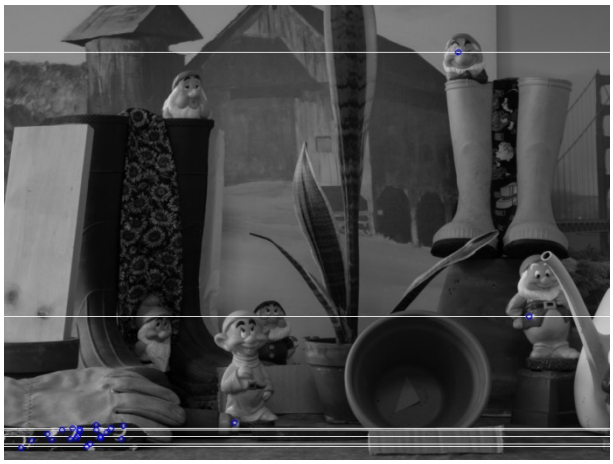


Figura 2.3: Líneas epipolares

Las líneas blancas son líneas epipolares y los puntos algunas características que destacan esquinas en la imagen.

Cuando se tiene el valor de la línea base **B**, los valores de disparidad y distancia focal, se procede a calcular la profundidad **Z** de todas las correspondencias, para con esto poder formar algún mapa de entorno, ese mapa puede ser en un entorno 3D o simplemente un mapa de profundidad sobre una imagen 2D como el que se muestra a continuación:

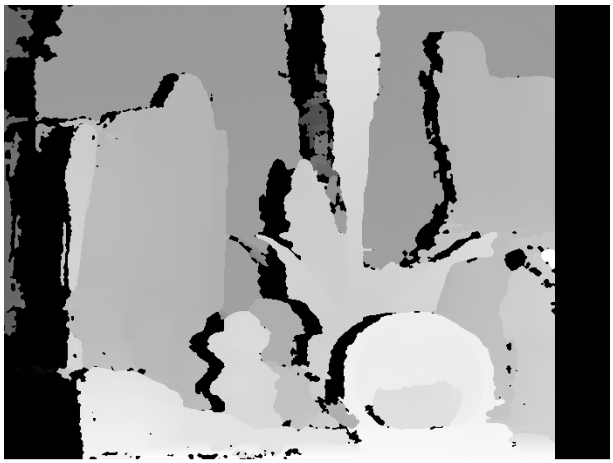


Figura 2.4: Mapa de profundidad

En el mapa de profundidad se pueden ver las partes más blancas como las más cercanas a la cámara y las más obscuras como las más lejanas [Klette, 2014].

2.3.2 Visión monocular

Para la visión monocular el problema no es más fácil, al usar una sola cámara, existe mayor incertidumbre, ya que para tener algo parecido a una línea base se debe mover la cámara, como se muestra a continuación:

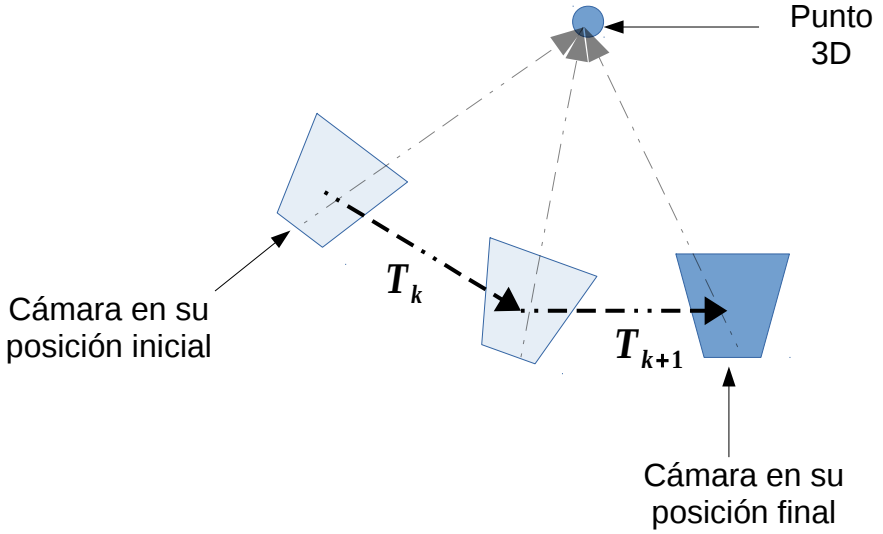


Figura 2.5: Movimiento de la cámara

La transformación **T** es dinámica y debe ser encontrada en cada intervalo de tiempo **k** para poder calcular la profundidad de cada punto 3D visto por la cámara. Al igual que en la visión binocular, se deben encontrar correspondencias, solo que en este caso las correspondencias se hacen con imágenes de una misma cámara, las imágenes usadas para la correspondencia están en intervalos de tiempo **k** diferentes generalmente consecutivos, una vez que se obtiene la disparidad y la trasformación de la cámara, se procede a hacer el cálculo de profundidad para la generación de mapa de profundidad y trayectoria. Al no tener una base fija como en la visión binocular, la escala del mapa puede variar e ir acumulando error, por lo que generar mapas precisos a gran escala es una tarea difícil para la visión monocular [Davison, 2003][Scaramuzza, 2011][Engel, 2014].

2.3.3 Métodos indirectos o basados en características

Los métodos basados en características funcionan en 5 pasos:

1) Se debe capturar por lo menos dos imágenes de una misma escena pero con pose diferente, debe haber coincidencia de información entre las dos imágenes.

2) Se extraen características de ambas imágenes, pueden ser puntos, esquinas, algún objeto o marcador. Algunos algoritmos usados son FAST y ORB, por ejemplo. Cada una de las características debe tener algún o algunos valores, puede ser alguna etiqueta o algo que permita evaluarlo con las características de la otra imagen.

3) Posteriormente se hace una correspondencia de características, donde debe coincidir el mismo punto, esquina, objeto, etc. de una imagen con la otra imagen. Esto se puede hacer comparando cada uno de sus valores encontrando los más parecidos.

4) Una vez que se tiene la correspondencia de características, se procede a hacer una triangulación con el valor de disparidad que tiene cada par de características. Con esta información se podrá obtener la información del entorno que lo rodea y estimar su posición.

5) Por último se va creando un mapa con cada una de las poses obtenidas, la cual deberá ser la trayectoria recorrida. Es común ir actualizando y mejorando los resultados según **n** numero de poses anteriores [Scaramuzza, 2011][Forster, 2014] [Engel, 2016].

2.3.4 Métodos directos o métodos basados en apariencia

Los métodos directos funcionan en 4 pasos:

1) Se debe capturar por lo menos dos imágenes de una misma escena pero con pose diferente, debe haber coincidencia de información entre las dos imágenes.

2) Se compara directamente la intensidad de los pixeles de las dos imágenes y se trata de alinear la imagen o los parches de pixeles.

3) Una vez alineadas las dos imágenes, se procede a hacer una triangulación con los valores de la transformación de las imágenes. Esto dará la información del entorno que lo rodea y se podrá estimar su posición.

4) Por último se va creando un mapa con cada una de las poses obtenidas, la cual deberá ser la trayectoria recorrida. Es común ir actualizando y mejorando los resultados según **n** numero de poses anteriores [Irani, 1999][Scaramuzza, 2011][Engel, 2014][Forster, 2014][Engel, 2016].

2.3.5 Métodos semi-directos o híbridos

Los métodos directos funcionan en 5 pasos:

1) Se debe capturar por lo menos dos imágenes de una misma escena pero con pose diferente, debe haber coincidencia de información entre las dos imágenes.

2) Se extraen características de ambas imágenes, pueden ser puntos, esquinas, algún objeto o marcador. Algunos algoritmos usados son FAST y ORB, por ejemplo.

3) Se forman parches de pixeles en la posición de las características obtenidas y se compara directamente la intensidad de los parches de pixeles de las dos imágenes para tratar de alinearlos.

4) Una vez alineado los parches, se procede a hacer una triangulación con los valores de las transformaciones.. Esto dará la información del entorno que lo rodea y se podrá estimar su posición.

5) Por último se va creando un mapa con cada una de las poses obtenidas, la cual deberá ser la trayectoria recorrida. Es común ir actualizando y mejorando los resultados según **n** numero de poses anteriores [Scaramuzza, 2011][Forster, 2014] [Engel, 2016].

Discusión

Resulta de utilidad conocer los modelos matemáticos utilizados, como por ejemplo, los grupos y álgebras de Lie, esto por que es muy común encontrar en la literatura estos términos cuando se trata de representar transformaciones en dos y tres dimensiones. En la mayoría de los grupos de Lie utilizados, se tiene como elemento matrices, sin embargo, no son matrices cualquiera, ya que tienen que seguir criterios para no salir del grupo para que siempre sea una trasformación válida. Las álgebras de Lie dan la libertad de moverse dentro del espacio tangente del grupo, el cual es más simple para realizar búsquedas y encontrar la pose óptima de una trasformación, siempre utilizando el mapa exponencial y logarítmico (ver Anexo A) para trasladarse del álgebra al grupo y viceversa.

Los algoritmos de optimización vistos son en esencia los más utilizados para encontrar una trasformación óptima, este problema de optimización es no lineal y aunque se puede encontrar el valor óptimo, no siempre es la solución a nuestro problema, en especial cuando se trabaja con imágenes pixeladas en donde el ruido, la iluminación, los pocos contrastes, entre otros factores, afectan el resultado obtenido.

Por último, los conceptos básicos y la clasificación de los algoritmos son importantes para este tema, ya que este trabajo utiliza solo métodos directos o semi-directos con visión monocular para la solución del problema.

Capítulo 3: Estado del arte

A continuación se mostrará trabajos relacionados con métodos directos de visión, la referencia de cada trabajo aparece al final de su sección.

3.1 “DTAM: Dense Tracking and Mapping in Real-Time”

Se presenta un algoritmo de estructura por movimiento que funciona en tiempo real con la ayuda del procesamiento de la unidad gráfica de procesamiento o GPU por sus siglas en inglés, el resultado es una reconstrucción densa y detallada de la estructura.

El algoritmo funciona en un principio como un sistema estándar de visión binocular basado en características, pero una vez se consiguen los primeros fotogramas clave, el sistema cambia a completamente a un método de seguimiento denso y mapeo basado en la alineación completa de las imágenes. Cada pose obtenida de la alineación de las imágenes está representada por una matriz perteneciente al grupo de Lie **SE**(3) de la forma:

$$R_{wc} \in \boldsymbol{SO}(3) \qquad \textit{(Ecuación: 3.1)}$$

$$T_{wc} = \begin{bmatrix} \boldsymbol{R_{wc}} & \boldsymbol{c_w} \\ 0 & 1 \end{bmatrix} \in \boldsymbol{SE}(3) \qquad \textit{(Ecuación: 3.2)}$$

Donde $\mathbf{T_{wc}}$ describe los puntos de trasferencia entre la cámara y el mundo real y $\mathbf{c_w}$ es la posición del centro óptico de la cámara.

La forma en que alinean las imágenes es mediante un nuevo método de optimización matemático iterativo para problemas no convexos. La función del error fotométrico a optimizar es de la forma:

$$F(\psi) = \frac{1}{n} \sum_{u \in \Omega} (f_u(\psi))^2 \qquad \textit{(Ecuación: 3.3)}$$

$$\psi \in \mathbb{R}^6 \qquad \textit{(Ecuación: 3.4)}$$

Donde **u** son los pixeles, **Ω** es toda la imagen **f** es la función que evalúa la diferencia de las imágenes y **Ψ** son los valores del álgebra de Lie $\mathfrak{se}(3)$ que representan la pose.

Posteriormente se mapea para obtener la pose con la función costo:

$$T_{lv}(\psi) = \exp\Big(\sum_{i=1}^{6} \psi_i \underset{\boldsymbol{SE}(3)}{gen_i}\Big) \qquad \textit{(Ecuación: 3.5)}$$

Donde $\mathbf{T_{lv}}$ es la aproximación de la pose real.

Mientras se va reconstruyendo el modelo y refinando el mapa de profundidad de texturas densas, se va haciendo completamente autónomo y no necesita rastreo de partes de la estructura o extraer características [Newcombe, 2011].

Discusión

A pesar de que el algoritmo funciona en tiempo real, tiene un alto costo computacional, requiere hardware especializado y el código no está disponible al público, lo que hace difícil su evaluación. No es viable si se utiliza un sistema embebido de bajo consumo ya que utiliza un GPU NVIDIA GTX 480 y un CPU corei7.

3.2 "SVO: Fast Semi-Direct Monocular Visual Odometry"

Este método de odometría visual llamado SVO que funciona en tiempo real, es denominado semi-directo, ya que utiliza extracción de características por medio del algoritmo FAST. Para la extracción de características con FAST, se recomienda utilizar imágenes con texturas de alta frecuencia. Por otra parte, aunque hay una extracción de características, la correspondencia de las características es realizada mediante métodos directos.

El algoritmo utiliza dos hilos, uno para estimar el movimiento y el otro para ir generando un mapa. La estimación del movimiento se hace mediante un modelo de alineación de parches de las imágenes. Los parches son regiones de 4x4 pixeles alrededor de una esquina (característica de FAST) que se alinean minimizando el error fotométrico dado por:

$$\delta I(\boldsymbol{T},\boldsymbol{u}) = I_k(\pi(T\cdot\pi^{-1}(\boldsymbol{u},du))) - I_{k-1}(\boldsymbol{u}) \qquad \forall\, \boldsymbol{u}\in\Re \qquad \textit{(Ecuación: 3.6)}$$

$$\boldsymbol{T_{k,k-1}} = \underset{T_{k,k-1}}{arg\,min}\frac{1}{2}\sum_{i\in\Re}\|\delta I(T_{k,k-1},\boldsymbol{u_i})\|^2 \qquad \textit{(Ecuación: 3.7)}$$

En donde **T** es la pose que pertenece al grup de Lie **SE**(3), **k** es el tiempo, **du** es la distancia de profundidad de una coordenada, **I** es una imagen en $\mathbb{R}^2$, **π** es una proyección de $\mathbb{R}^3$ a $\mathbb{R}^2$, **δI** es la diferencia fotométrica de las coordenadas **u** de la imagen 2D. $\Re$ es una región con una distancia conocida.

Al ser una ecuación no lineal, se puede resolver por un método iterativo matemático como Gauss-Newton. Dada la estimación de una pose relativa, se puede actualiza **T**(ξ) para estimar los parámetros de la transformación, en donde ξ pertenece al álgebra de Lie $\mathfrak{se}(3)$.

$$\delta I(\xi,\boldsymbol{u_i}) = I_k(\pi(\hat{\boldsymbol{T}}_{\boldsymbol{k,k-1}}\cdot\boldsymbol{p_i})) - I_{k-1}(\pi(\boldsymbol{T}(\xi)\cdot\boldsymbol{p_i})) \qquad \textit{(Ecuación: 3.8)}$$

$$p_i = \pi^{-1}(\boldsymbol{u_i},du) \qquad \textit{(Ecuación: 3.9)}$$

Para encontrar el paso óptimo para la actualización se calcula la derivada del error fotométrico poniéndolo en cero.

$$\sum_{i\in\Re}\nabla\delta I(\xi,\boldsymbol{u_i})^T I(\xi,\boldsymbol{u_i}) = 0 \qquad \textit{(Ecuación: 3.10)}$$

Por definición se calcula el Jacobiano de la forma:

$$\boldsymbol{J_i} = \nabla\delta I(0,\boldsymbol{u_i}) \qquad \textit{(Ecuación: 3.11)}$$

Una vez que se obtiene la alineación, se tiene una nueva suposición de las características 3D. En base a la nueva suposición, se intenta alinear el mapa 3D, con esto se reducirá la deriva. Cada puto 3D estimado, es proyectado a 2D y se procede a minimizar el error nuevamente, solo que ahora bajo una transformación afín **A** que pertenece al grupo de Lie **Aff**(2). Esto de la forma:

$$\boldsymbol{u}'_i = arg\,min \frac{1}{2} \| \boldsymbol{I_k}(\boldsymbol{u}'_i) - \boldsymbol{A} \cdot \boldsymbol{I_r}(\boldsymbol{u_i}) \|^2 \qquad \textit{(Ecuación: 3.12)}$$

Con cada **u'** se obtiene la nueva pose mediante la optimización de la siguiente función:

$$\boldsymbol{T_{k,w}} = \underset{T_k, w}{arg\,min} \frac{1}{2} \sum_{i \in \Re} \| \boldsymbol{u_i} - \pi(\boldsymbol{T_{k,w}} w \mathrm{P}_i) \|^2 \qquad \textit{(Ecuación: 3.13)}$$

Dado a que es una función no lineal se utiliza Gauss-Newton para encontrar la solución. La optimización del mapa es local (*Local Bundle Adjustment*) en configuraciones que requieren precisión, mientras que cuando se requiere velocidad, no se realiza la optimización. Puede funcionar a 55 cuadros por segundo en un sistema embebido *Odroid*, mientras que en una computadora core i7 puede trabajar a 300 cuadros por segundo.

Durante la generación del mapa, en un segundo hilo, se utilizan modelos basados en distribuciones de probabilidad para hacerlo robusto a puntos erróneos o mal ubicado en el mapa (*outiers*).

En la experimentación demuestra ser un algoritmo preciso que funciona en tiempo real utilizando como plataforma un mini vehículo aéreo [Forster, 2014].

Discusión

El código del algoritmo está disponible, sin embargo, tiene errores en tiempo de ejecución por lo que no es posible evaluarlo adecuadamente. Con pláticas con el Dr. José Martínez Carranza de INAOE, comentó que tampoco pudo utilizarlo adecuadamente, y posteriormente, el autor del artículo también confirmó que hay cosas que tienen que arreglarse en el código.

3.3 "LSD-SLAM: Large-scale direct monocular SLAM"

Presenta un método directo monocular llamado LSD-SLAM que funciona en tiempo real, permite la generación de mapas 3D semi-densos y precisos de entornos grandes basado en

la alineación directa de las imágenes. Esto se obtiene filtrando gran cantidad de pixeles de las dos imágenes con una línea base pequeña. A diferencia de los otros métodos, este método utiliza un seguimiento directo que opera con en el álgebra de de Lie $\mathfrak{sim}(3)$, permitiendo hacer correcciones en la escala. Este método detecta el cierre de ciclos en trayectorias para optimizar el mapa y reducir en mayor medida el error acumulado. Utiliza el método de optimización de Gauss-Newton y Levenberg-Marquard (no lo menciona pero se ve en el código fuente) para la alineación y optimización de la pose, además, usa un filtro de probabilidad para reducir el ruido en el mapa.

La pose expresa utilizando el álgebra de Lie $\mathfrak{se}(3)$ que son mapeados al grupo **SE**(3) utilizando el mapa exponencial

$$\xi \in \mathfrak{se}(3) \qquad \textit{(Ecuación: 3.14)}$$

$$G = \exp_{\mathfrak{se}(3)}(\xi) \qquad \textit{(Ecuación: 3.15)}$$

$$G = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \in \boldsymbol{SE}(3) \qquad \textit{(Ecuación: 3.16)}$$

Para alinear las imágenes, se debe minimizar el error fotométrico de la función de costo **E**:

$$r_i^2(\xi) = I_{ref}(\boldsymbol{p_i}) - I(\omega(\boldsymbol{p_i}, D_{ref}(\boldsymbol{p_i}), \xi))^2 \qquad \textit{(Ecuación: 3.17)}$$

$$E(\xi) = \sum_i r_i^2(\xi) \qquad \textit{(Ecuación: 3.18)}$$

Para optimizar con Gauss-Newton se utiliza una aproximación del Jacobiano **J** de la forma:

$$\boldsymbol{J} = \frac{\delta r(\varepsilon \circ \xi^{(n)})}{\delta \varepsilon} | \varepsilon = 0 \qquad \textit{(Ecuación: 3.19)}$$

$$\delta \xi^n = (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T r(\xi^{(n)}) \qquad \textit{(Ecuación: 3.20)}$$

Por otra parte, también se propone el esquema de mínimos cuadrados ponderados, que lo hace más robusto al ruido y oclusiones.

$$E(\xi) = \sum_i \omega(\xi) r_i^2(\xi) \qquad \textit{(Ecuación: 3.21)}$$

$$\delta \xi^n = (\boldsymbol{J}^T \boldsymbol{W} \boldsymbol{J})^{-1} \boldsymbol{J}^T \boldsymbol{W}\, r(\xi^{(n)}) \qquad \textit{(Ecuación: 3.22)}$$

Obteniendo la pose en $\mathfrak{se}(3)$ minimizando la varianza normalizada con el error fotométrico:

$$E_p(\xi_{ji}) = \sum_{\boldsymbol{p} \in \Omega D_i} \| \frac{r_p^2(\boldsymbol{p}, \xi_{ji})}{\sigma_{rp}^2(\boldsymbol{p}, \xi_{ji})} \| \quad \textit{(Ecuación: 3.23)}$$

En donde:

$$r_p(\boldsymbol{p}, \xi_{ji}) = I_i(\boldsymbol{p}) - I_j(\omega(\boldsymbol{p_i}, D_i(\boldsymbol{p}), \xi_{ji})) \quad \textit{(Ecuación: 3.24)}$$

$$\sigma_{rp}^2(\boldsymbol{p}, \xi_{ji}) = 2\sigma_I^2 + \left(\frac{\partial r_p(\boldsymbol{p}, \xi_{ji})}{\partial D_i(\boldsymbol{p})}\right)^2 V_i(\boldsymbol{p}) \quad \textit{(Ecuación: 3.25)}$$

Sabiendo que cada fotograma se compone de:

$$K_i = (I_i, D_i, V_i) \quad \textit{(Ecuación: 3.26)}$$

Donde **I** es la imagen, **D** el mapa de profundidad inverso y **V** la varianza de la profundidad inversa.

Los mejores resultados se dan cuando se cambia el álgebra de Lie $\mathfrak{se}(3)$ por al álgebra de Lie $\mathfrak{sim}(3)$, el uso de esta última álgebra permite reducir la deriva de la escala por medio de la alineación. Por lo que la nueva función para minimizar es de la forma:

$$E_p(\xi_{ji}) = \sum_{\boldsymbol{p} \in \Omega D_i} \| \frac{r_p^2(\boldsymbol{p}, \xi_{ji})}{\sigma_{rp}^2(\boldsymbol{p}, \xi_{ji})} + \frac{r_d^2(\boldsymbol{p}, \xi_{ji})}{\sigma_{rd}^2(\boldsymbol{p}, \xi_{ji})} \| \quad \textit{(Ecuación: 3.27)}$$

Donde:

$$r_d(\boldsymbol{p}, \xi_{ji}) = \boldsymbol{p}'_3 - D(\boldsymbol{p}'_{1,2}) \quad \textit{(Ecuación: 3.28)}$$

$$\sigma_{rd}^2(\boldsymbol{p}, \xi_{ji}) = V_j(\boldsymbol{p}'_{1,2}) + \left(\frac{\partial r_d(\boldsymbol{p}, \xi_{ji})}{\partial D_j(\boldsymbol{p}'_{1,2})}\right)^2 + V_i(\boldsymbol{p}) + \left(\frac{\partial r_d(\boldsymbol{p}, \xi_{ji})}{\partial D_i(\boldsymbol{p})}\right)^2 \quad \textit{(Ecuación: 3.29)}$$

$$\boldsymbol{p}' = \omega_s(\boldsymbol{p}, D_i(\boldsymbol{p}), \xi_{ji}) \quad \textit{(Ecuación: 3.30)}$$

En los resultados se puede ver mapas de distancias largas (500m) y de buena calidad [Engel, 2014].

Discusión

El código está disponible y es posible probarlo bajo el entorno de ROS (*Robotic Operativa System*), sin embargo, cuando se ejecuta en tiempo real hay que ser cuidadoso, el algoritmo se pierde fácilmente y deja de funcionar. Los movimientos de la cámara deben ser pequeños

y sin muchas rotaciones, el entorno debe tener texturas y estar bien iluminado.

3.4 "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle"

En este artículo se puede encontrar el diagrama de un sistema que utiliza el método semi-directo SVO aplicado a un mini vehículo aéreo o MAV por sus siglas en inglés. Justifica que para este tipo de vehículos, es necesario utilizar dispositivos livianos y de bajo consumo, por lo que un sensor que proporcione imagen con profundidad (*RGB-D*) o un sensor que mida distancia por láser (*LIDAR*), no serían eficientes para ser utilizados. Por lo que su plataforma está compuesta de la siguiente manera:

Tabla 3.1: Información de MAV

Componente	Peso (g)
Armazón y hélices	119
Motores y controladores	70
PX4FMU, PX4IOAR	35
Hardkernel Odroid-U3	49
Cámara	16
Batería de 1350mA h	99
Otros	54
Total	**442**

Utiliza el algoritmo SVO para la estimación de movimiento por medio de la cámara, que combina con la información de la IMU y la dinámica del sistema. La fusión de los sensores se realiza mediante un filtro de Kalman, este algoritmo no fue modificado por el autor y está disponible bajo licencia *open-source*. De manera general, el control es del sistema está dado de esta forma:

$$\tau_{des} = \boldsymbol{J}\begin{bmatrix} G_{\theta\phi}(\theta_{des}-\theta) \\ G_{\theta\phi}(\phi_{des}-\phi) \\ G_{\psi}(\psi_{des}-\psi) \end{bmatrix} \omega \times \boldsymbol{J}\,\omega \qquad (1)$$ *(Ecuación: 3.31)*

Donde:

$\tau_{des} = Torque\ deseado$
$\boldsymbol{J} = Vector\ de\ inercia$
$G_{\theta\phi} = Ganancia\ de\ Roll\ y\ Pitch$
$G_{\psi} = Ganancia\ de\ Yaw$
$\omega = Velocidad\ angular$

Para la calibración de la IMU se utilizaron las siguientes ecuaciones para el giroscopio:

$$(2)\quad \widetilde{\omega} = \omega + b_{\omega} + n_{\omega} \qquad \textit{(Ecuación: 3.32)}$$

Donde:

$b_{\omega} = Valor\ de\ calibración\ de\ giroscopio$
$n_{\omega} = Ruido\ del\ giroscopio$
$\widetilde{\omega} = Velocidad\ angular\ aproximado$
$\omega = Velocidad\ angular$

Cuando el vehículo está suspendido en el aire:

$$(3)\quad \widetilde{\omega} = b_{\omega} + n_{\omega} \qquad \textit{(Ecuación: 3.33)}$$

Para estimar un nuevo valor de calibración, mientras está suspendido en el aire:

$$(4)\quad \hat{b_{\omega}} = \frac{1}{n}\sum_{k=1}^{n} \widetilde{\omega}_k \qquad \textit{(Ecuación: 3.34)}$$

Para la calibración de acelerómetro:

$$(5)\quad \widetilde{a} = c + \hat{a} + b_{acel} + n_{acel} \qquad \textit{(Ecuación: 3.35)}$$

Donde:

$b_{acel} = Valor\ de\ calibración\ de\ acelerómetro$
$n_{acel} = Ruido\ de\ acelerómetro$
$\widetilde{a} = Aceleración\ aproximada$
$\hat{a} = Aceleración\ perturbada\ por\ factores\ externos$
$c = Masa\ normalizada\ con\ el\ empuje$

Cuando el vehículo está suspendido en el aire:

$$(6)\quad \tilde{a} = -g + b_{acel} + n_{acel} \qquad \textit{(Ecuación: 3.36)}$$

Para estimar un nuevo valor de calibración, mientras está suspendido en el aire:

$$(7)\quad \hat{b_{acel}} = \frac{1}{n}\sum_{k=1}^{n} \tilde{a}_k + g \qquad \textit{(Ecuación: 3.37)}$$

El sistema es robusto y funciona de manera autónoma, tanto en interiores como en exteriores [Faessler, 2015].

Discusión

La información que se destaca en este artículo, es la de la calibración y uso del IMU, ya que es muy similar a la utilizada en este trabajo.

3.5 "Direct Sparse Odometry"

En este trabajo, se presenta un método directo monocular disperso. Utiliza bloques de pixeles de 32x32 para obtener a los mejores candidatos para ser rastreados, no solo siguen los altos contrastes, ya que muchas veces, las texturas suaves proporcionan robustez a los cambios de iluminación. La estrategia de rastreo utilizada se inspira en el algoritmo LSD-SLAM. Conforme se adquieren nuevos puntos, va remplazando a los viejos.

Tiene un sistema que administra los fotogramas que usará, inspirado en ORB-SLAM toma varios fotogramas y de ahí selecciona los mejores utilizando ciertos criterios.

El error fotométrico completo de todos los fotogramas se calcula por:

$$E_{photo} = \sum_{i \in F} \sum_{p \in P} \sum_{j \in obs(p)} E_{pj} \qquad \textit{(Ecuación: 3.38)}$$

Donde **F** son todos los fotogramas, **P** todos los puntos y **obs(p)** son los puntos visibles. Donde:

$$E_{pj} = \sum_{p \in N_p} \omega_p \| I_j(\boldsymbol{p}' - b_j) - \frac{t_j e^{aj}}{t_i e^{ai}} (I_i \boldsymbol{p} - b_j) \| \qquad \textit{(Ecuación: 3.39)}$$

En donde $\mathbf{N_p}$ es el conjunto de pixeles de la suma del cuadrado de las diferencias, **I** son imágenes, **p** son puntos candidatos de la imagen, **p'** son puntos proyectados en 3D, **t** es el tiempo de exposición, **b** y **e** son valores de la función de transferencia del brillo.

Al igual que en algoritmos anteriores las poses están dadas en matrices que pertenecen al grupo de Lie **SE**(3)

$$T_j T_i^{-1} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \in \boldsymbol{SE}(3) \qquad \textit{(Ecuación: 3.40)}$$

En donde **T** son transformaciones en los fotogramas.

La optimización se realiza con Gauss-Newton, además de que también hace una optimización local para el mapa [Engel, 2016].

Discusión general

El código está disponible y funciona perfectamente, no se pierde tanto como el algoritmo LSD-SLAM, sin embargo no resuelve el problema del todo, muchas rotaciones no son interpretadas adecuadamente y sigue teniendo problemas cuando la cámara no se mueve suavemente.

3.6 Comparación de algoritmos de visión

Tabla 3.2: Comparación de métodos directos

Método	Tipo	Disponibilidad de código fuente	¿Funciona el código?	Librarías utilizadas	Observaciones
DTAM	Directo	No	Sí	*Sin información*	Funciona en tiempo real pero tiene un alto costo computacional.
SVO	Semi-Directo	Sí	No	OpenCV, Sophus, libCVD, g2o, Vikit	En el artículo dice que funciona en tiempo real en sistemas embebidos. Pero el código diponible no funciona para probarlo.
LSD-SLAM	Directo	Sí	Sí	Sophus, Point Cloud, openFabMap, BLAS,g2o	Funciona en tiempo real pero el seguimiento se pierde fácilmente.
DSO	Directo	Sí	Sí	Pangolin, Eigen3, Boost, OpenCV	El seguimiento es mejor que LSD-SLAM pero sigue teniendo problemas.

Discusión

Los algoritmos de V-SLAM y odometría visual fueron probados en su mayoría, exceptuando DTAM, esto por que el autor no pone el código fuente a disposición del público y aunque hay versiones creadas por terceros, se descartó su evaluación, ya que en el artículo menciona que requiere un equipo de cómputo poderoso y dado a que se busca reducir el costo computacional y usar recursos minimalistas en un sistema embebido, DTAM queda descartado.

LSD-SLAM y DSO fueron los algoritmos que más se probaron y se llegó a la conclusión de que eran lentos e incapaces de funcionar correctamente en un sistema embebido además de que tienen problemas para preservar el seguimiento de los pixeles, si se realiza un movimiento grande, los algoritmos dejan de funcionar, otro problema, es la interpretación de los giros, muchas veces las rotaciones no son interpretadas adecuadamente y la trayectoria del mapa sale mal. Estos algoritmos requieren una buena rectificación de las imágenes, ya que sin esto el mapa se distorsiona por el lente de la cámara.

Según la información presentada en el artículo, SVO funciona bien bajo un sistema embebido, sin embargo, la implementación que pone a disposición el autor tiene errores en tiempo de ejecución, por lo cual no fue posible hacer pruebas con el algoritmo.

Con lo que se muestra en el estado del arte se puede observar que el costo computacional es grande por la gran cantidad de pixeles que se procesan, además de que según la trasformación empleada, la optimización puede ser muy complicada, por ejemplo, si se usa una homografía, se tendrá que encontrar nueve valores diferentes, en cambio, si se usa una trasformación *similar* de tres dimensiones se tendrán que encontrar solo 7 valores.

Capítulo 4: Descripción del sistema

En este capítulo se describirán cada una de las partes del sistema, Que consiste de un robot móvil como plataforma de pruebas, el robot lleva a bordo una cámara de video viendo hacia el suelo, sobre de la cámara está la unidad de medida inercial, ambas sobre el eje de rotación del robot. Como procesamiento se tiene a la tarjeta *Tinker Board* y una laptop, dentro de estos dispositivos reside un software especializado para que las plataformas puedan trabajar en conjunto. A continuación se describirán con más detalle estas partes.

4.1 Esquema general

El robot tiene una configuración de tres ruedas, de las cuales dos se mueven con motor de corriente continua y una es una rueda loca, los motores se mueven utilizando un *puente H L298N* lo que permite hacerlos rotar en ambas direcciones.

La tarjeta utilizada como sistema embebido es una *Tinker Board* de la marca Asus, esta se encarga de leer los sensores, procesar la información y mandarla a una computadora personal o laptop mediante una red de área local. Dentro de la PC o laptop se encuentra el

programa que genera el mapa de la trayectoria del robot, con el programa también es posible enviar información al robot por medio de una interfaz de usuario.

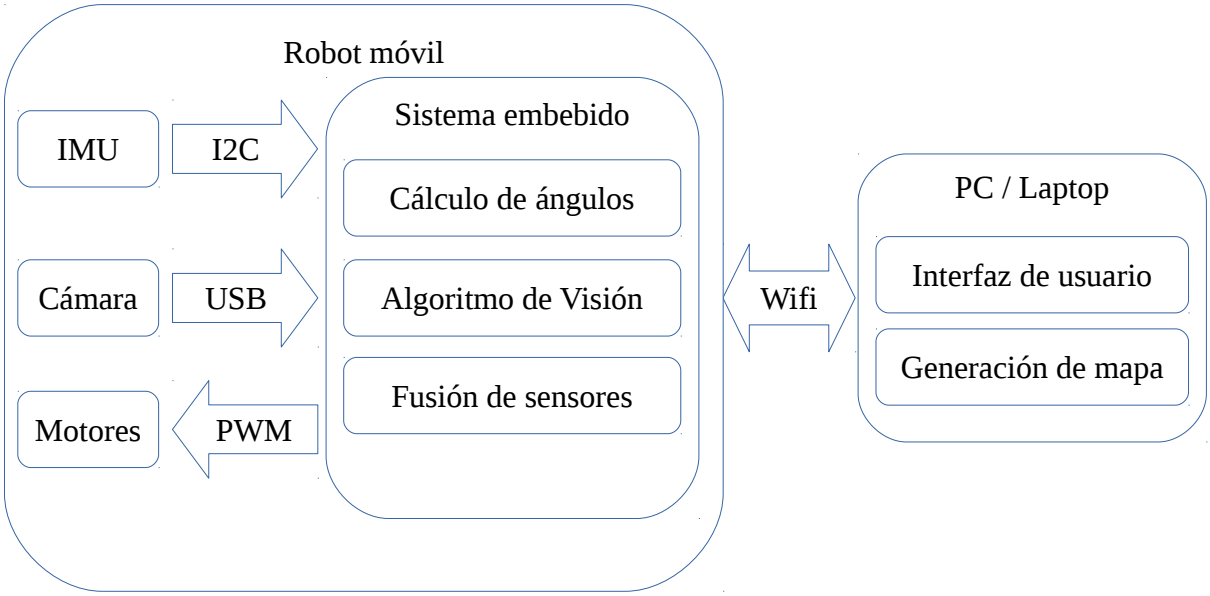


Figura 4.1: Diagrama general del sistema

4.2 Robot móvil

El chasis del robot es de fibra de vidrio y acrílico, tiene una medida aproximada de 20 por 11.5 cm, las ruedas con las que se mueve tienen un diámetro de 6.5 cm con 3 cm de ancho, la rueda loca tiene 3cm de diámetro y 1cm de ancho. Como se muestra a continuación:

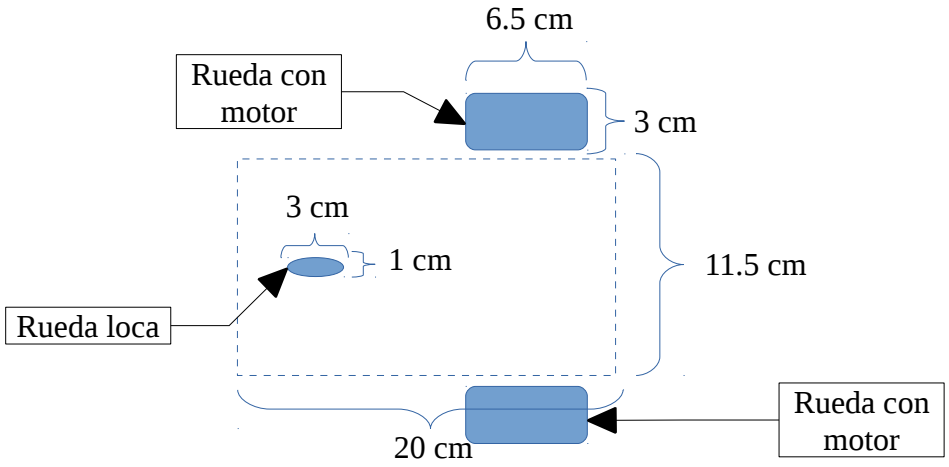


Figura 4.2: Medidas del robot móvil

Aquí se muestra un modelo del robot visto desde arriba

La tarjeta que se utiliza como sistema embebido, la cámara, la unidad de medición inercial y el *puente H* se ubican de la siguiente manera:

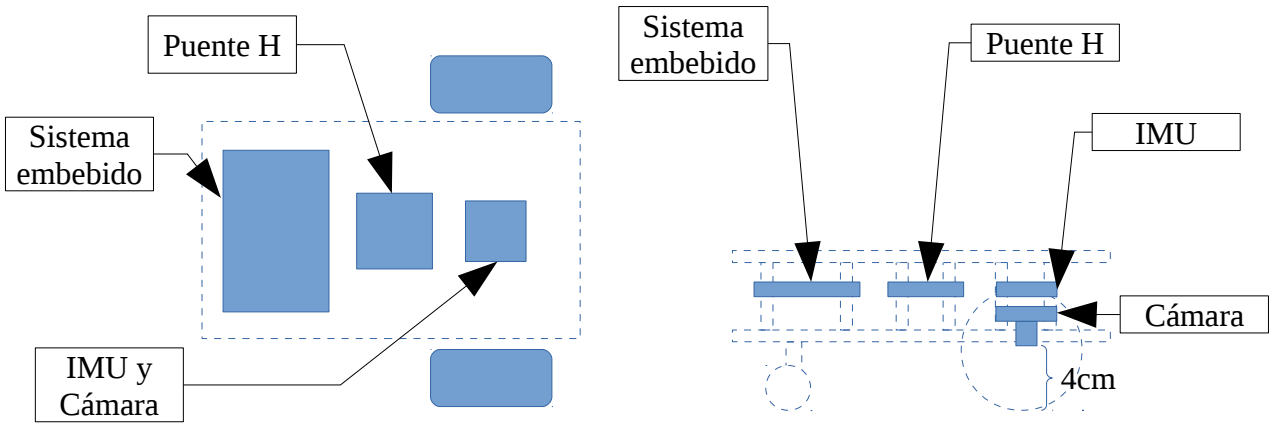


Figura 4.3: Componentes del robot móvil

Del lado izquierdo se muestra la vista superior del robot, mientras que en el lado derecho se muestra la vista lateral del robot.

Los motores son controlados con ayuda de un puente *H L298N* y una tarjeta usada como sistema embebido. El sistema embebido a bordo del robot envía señales PWM (modulación por ancho de pulsos) para controlar la velocidad y dirección de los motores, lo que le permite avanzar, retroceder, girar en sentido de las manecillas del reloj y en sentido contrario sobre su propio eje, a este tipo de robot con ruedas, se lo conoce como robot diferencial de tres ruedas.

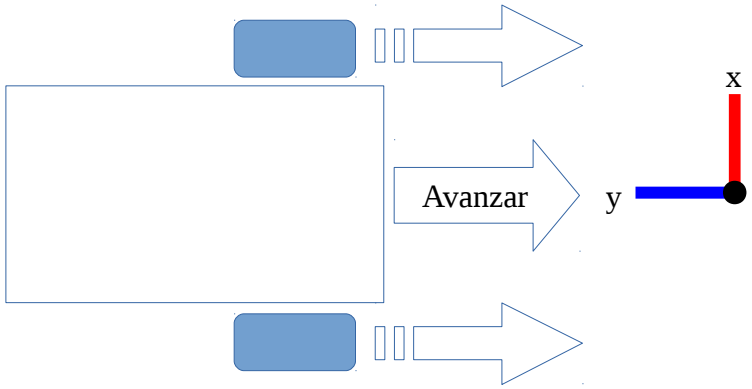


Figura 4.4: Movimiento avanzar

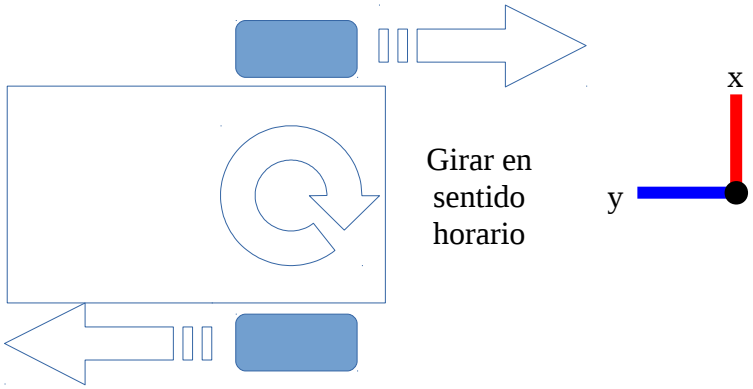


Figura 4.5: Girar en sentido horario

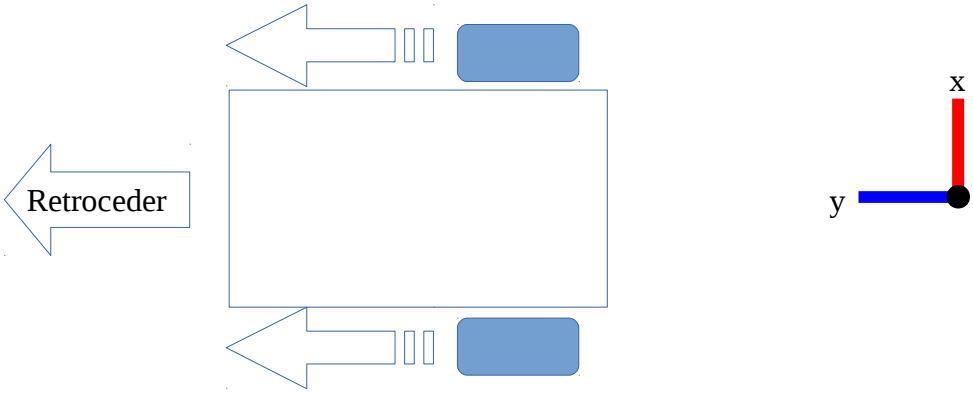


Figura 4.6: Movimiento retroceder

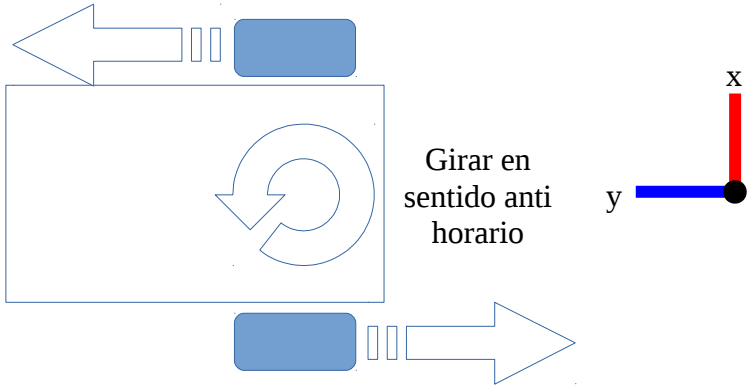


Figura 4.7: Girar en sentido anti horario

4.3 Sistema embebido

Un sistema embebido puede definirse en términos generales como un dispositivo que contiene hardware y componentes de software estrechamente acoplados para realizar una sola función, forma parte de un gran sistema, no pretende ser programado de forma independiente por el usuario y se espera que funcione con una interacción humana mínima o nula. Dos características adicionales son muy comunes en los sistemas embebidos: operación reactiva y fuertemente restringida.

La mayor parte del sistema embebido interactúa directamente con los procesos o el ambiente, tomando decisiones sobre la marcha, basadas en sus entradas. Esto hace necesario que el sistema tenga que ser reactivo, respondiendo en tiempo real para procesar las entradas para asegurar una operación adecuada. Además, estos sistemas operan en entornos restringidos donde la memoria, la potencia de cálculo y la fuente de alimentación es limitada. Por otra parte, los requisitos de producción, en la mayoría de los casos debido al volumen, imponen restricciones de alto costo a los diseños [Jiménez, 2014].

4.3.1 Tinker Board

Tinker Board es una computadora de una sola placa o SBC por sus siglas en inglés, pertenece a la marca Asus y tiene su propio sistema operativo llamado TinkerOS que está basado en Debian, sin embargo también se puede usar otros sistemas operativos como *Armbian* o *Android*, tiene interfaz serial de display o DSI por sus siglas en inglés, interfaz serial para cámara o CSI por sus siglas en inglés y interfaz de transmisor-receptor asíncrono universal o UART por sus siglas en inglés. La tarjeta *Tinker Board* cuenta con las siguientes característica:

Tabla 4.1: Características de Tinker Board

Procesador	Rockchip Quad core 1.8 GHz ARM Cortex-A17 (32-bit)
GPU	ARM Mali-T764 GPU
Memoria RAM	DDR2 de canal Dual de 2GB
Almacenamiento	Memoria micro SD
USB 2.0	4
Gigabit Ethernet	1
Bluetooth V4.0	1
Wi-Fi	1
HDMI 1.4 a	1
Consumo	10-12 Watts

La tarjeta *Tinker Board* tiene características superiores a la utilizada en el algoritmo SVO, y en algunas pruebas de rendimiento demuestra que da buenos resultados en procesamiento de imágenes. Ver *https://www.youtube.com/watch?v=k4kMEbeORLo*.

4.4 Sensores

Los sensores son los instrumentos de entrada que permiten captar información del entorno. En este trabajo, los sensores principales son una cámara de la marca *KAYETON* con un obturador global (*Global Shutter*) que trabaja a 90 cuadros por segundo y una IMU de la marca *Adafruit* de 10 grados de libertad o DOF (*Degrees of Freedom*).

4.4.1 Cámara

Se utiliza una cámara modelo *KYT-U030-GS01* de la marca *KAYETON*, el sensor que es *OmniVision OV7251* y trabaja a 90 cuadros por segundo, esto lo hace ideal para algoritmos de visión robótica, ya que como se había visto antes, la velocidad de la cámara es fundamental para la precisión en los métodos directos y semi-directos [Forster, 2014].

La cámara cuenta con un obturador global, que también es requerido para tener buenos resultados en métodos directos, principalmente por que evitan la distorsión de la imagen, esto es mencionado en la información técnica del código fuente de LSD-SLAM y DSO.

La resolución de la cámara es VGA (640x480) y es monocromática, lo cual no es ningún impedimento ya que SVO utiliza una cámara con resolución similar (752×480) y LSD-SLAM recomienda usar una resolución VGA, la cuestión de que sea monocromática tampoco es problema debido a que LSD-SLAM, SVO y DSO convierten del modelo RGB a escala de grises.

La interfaz de comunicación es USB 2.0 con soporte OTG (*On-The-Go*) y UVC (*USB Video Class*) para los sistemas operativos Windows, GNU/Linux y Android y entrega la información con formato de compresión MJPEG. Todas las especificaciones de la cámara se encuentran en:

https://www.kayetoncctv.com/global-shutter-monochrome-90fps-webcam-vga-640-x-480p-otg-uvc-usb-camera-module/.

4.4.2 Unidad de medición Inercial

La unidad de medición inercial utilizada es de la marca Adafruit, proporciona 3 valores por medio del giroscopio con el sensor L3GD20H, tomando en cuenta el movimiento del robot, la mayoría de los giros se harán sobre el eje **z**, movimiento denominado Guiñada, los giros sobre el eje **x** denominados alabeo y los giros sobre el eje **y** denominados cabeceo, son prácticamente nulos en este tipo de vehículos.

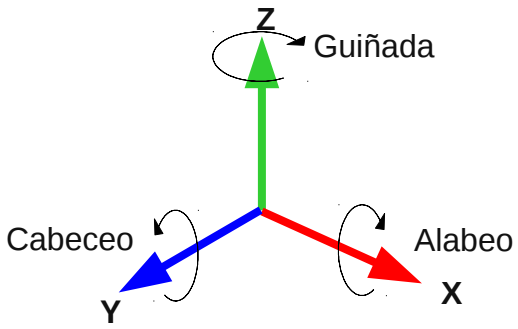


Figura 4.8: Ejes de rotación 3D

Los 3 valores del magnetómetro y los 3 valores del acelerómetro que proporciona el sensor LSM303, son utilizados como complemento para determinar las rotaciones. Por último el sensor BMP180 proporciona valores de presión atmosférica y temperatura, este sensor no es utilizado en este trabajo, sin embargo es posible utilizarlo para determinar la altura **z** por medio de la presión atmosférica.

En total se tienen 10 valores para determinar la posición o 10 grados de libertad (10 DOF) y un valor de temperatura que queda aislado, ya que no da información de posición. El IMU utiliza un voltaje de 3v a 5v y se comunica a través de una interfaz I2C, sin embargo, también se puede hacer lecturas análogas de los sensores. [Adafruit, 2014]

4.5 Software

Es software está desarrollado en su totalidad con lenguaje de programación C / C++ utilizando el compilador GCC en su versión 7.2, se utilizan las librerías multiplataforma de Qt version 5.9.1 para el diseño de la interfaz gráfica, también se utilizan las herramientas de desarrollo de protocolos web como QWebSocket, los relojes de interrupción con QTimer y librerías del núcleo básico de Qt. OpenCV es una librería utilizada principalmente para procesar las imágenes utilizando la GPU, ya que las últimas versiones se combina muy bien con OpenCL y hace muy simple realizar esta tarea, la versión utilizada es la 3.1.0, además se usa para capturar las imágenes de la cámara y mostrarlas en pantalla con ayuda de Qt, también se utilizan algoritmos para extraer características y aplicar filtros de detección de bordes. Por último, para la generación del mapa se utiliza OpenGL en su versión 4.5.0 y GLU en su verión 1.3, estas librerías permiten generar gráficos 3D aprovechando la GPU de la computadora, ayudan a realizar transformaciones y cambiar la perspectiva del entorno. Todo esto se utilizó sobre el sistema operativo GNU/Linux Kubuntu 17.10 y sobre TinkerOS.

4.5.1 Software desarrollado

Se desarrolló un software capaz de trabajar en SBCs, se utiliza como software embebido y como requisito mínimo requiere conexión a una red local, Qt 5.3 o superior, WiringPi 2.50 o superior y OpenCV 3.0 o superior. Con el software se puede controlar al robot móvil, la cámara irá capturando imágenes para procesalas y mandar la información a una estación para ser visualizada como un mapa 3D.

El software desarrollado para la estación, requiere conexión a Internet, Qt 5.3 o superior, OpenGL 4.0 o superior y OpenCV 3.0 o superior. Este software recibe la información del robot y va generando un mapa 3D, también se le puede mandar instrucciones al robot por medio de la interfaz gráfica.

También se desarrolló software para pruebas *offline*, requiere Qt 5.3 o superior, OpenGL 4.0 o superior y OpenCV 3.0 o superior. Con este software se puede generar mapas 3D utilizando imágenes y datos del IMU previamente capturadas.

Otro software es el utilizado para medir el rendimiento de algoritmos de optimización, este requiere Qt 5.3 o superior, OpenGL 4.0 o superior y OpenCV 3.0 o superior. Con el se puede calcular el error para posteriormente mostrarlo en un gráfico 3D, marca el óptimo y la alineación obtenida.

Por último, está un entorno para hacer pruebas exclusivas de visión, generando un mapa 2D, este requiere Qt 5.3 o superior y OpenCV 3.0 o superior.

En todas las aplicaciones 3D se utilizaron librerías de OpenGL para hacer rotaciones, sin embargo, en algunos casos especiales se utilizaron implementaciones propias para modelar rotaciones con cuaternios.

Capítulo 5: Algoritmo Visión-Inercial basado en métodos directos

A continuación de presentará el algoritmo de odometría desarrollado basado en métodos directos. Se muestra con detalle la selección de los parches, la alienación e interpretación de la pose, la calibración y filtrado de la IMU, así como el cálculo de los ángulos de rotación con acelerómetro, magnetómetro y giroscopio. Por último se combina la información del método directo de visión con la información procesada del IMU.

5.1 Esquema general

El algoritmo desarrollado se basa en métodos directos por que aprovecha la intensidad de todos los pixeles, sin embargo, se realiza una selección de parches de pixeles para evitar utilizar zonas demasiado lisas y con pocas texturas, esto por ser zonas difíciles para la alineación de los parches, esto se debe a la forma que tiene el error fotométrico que por lo regular no tiene forma convexa.

Para interpretar las rotaciones se utiliza una IMU a la que se le aplica una calibración y una serie de filtros para eliminar el ruido y así mejorar su precisión, posteriormente se complementa con la información del algoritmo de visión para obtener la pose completa que es interpretada en el entorno 3D utilizando un cuaternio y un vector de traslación.

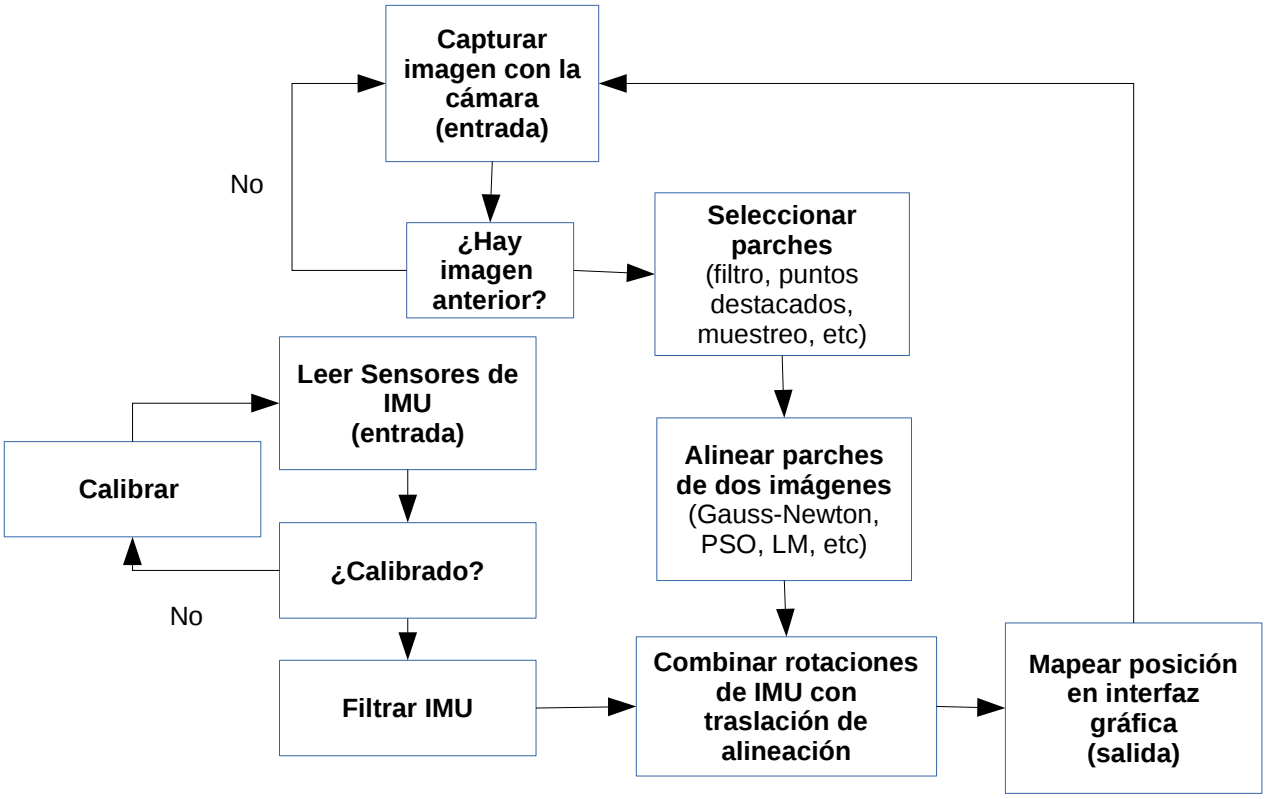


Figura 5.1: Diagrama general del algoritmo Visual-Inercial

El algoritmo funciona en 3 hilos, un hilo se dedica a leer la IMU y corregir los valores con la calibración, posteriormente se aplican los filtros pasa bajos y el filtro complementario, otro hilo aplica una interrupción cada 100ms para ir integrando con el intervalo de tiempo constante la velocidad angular, por último está el hilo que captura las imágenes de la cámara y las procesa.

5.2 Método directo

Este método directo se compone de 3 pasos, captura de imágenes, selección de parches, alineación y cálculo de traslación. Hasta este punto son tres pasos, sin embargo, esto es solo para la parte de visión, ya que de manera paralela se está calculando las rotaciones con la

IMU y se está mapeando las poses obtenidas.

5.2.1 Captura de imágenes

Para iniciar se debe capturar por lo menos dos imágenes y al tratarse de un sistema monocular, esto implica que al primer tiempo solo se obtendrá una imagen y no será hasta un segundo tiempo que se podrá ejecutar el algoritmo completo.

5.2.2 Selección de parches

En esta parte se itera **i** en toda la imagen **I**, con un tamaño fijo de pixeles **n** x **n** que conforma **S**, dentro de esa área, se forma un parche $\mathfrak{R}$ de tamaño **m** x **m**, en donde el subíndice **k** es el paso en el tiempo, **x** y **y** son los argumentos independientes de cada función que determina la posición en cada matriz :

$$\boldsymbol{S_k^i}(x,y) \subset \mathbb{R}^{n x n} \wedge \mathfrak{R}_k^i(x,y) \subset \mathbb{R}^{m x m} \qquad \textit{(Ecuación: 5.1)}$$

$$\boldsymbol{S_k^i}(x,y) \in I_k(x,y) \wedge \mathfrak{R}_k^i(x,y) \in \boldsymbol{S_k^i}(x,y) \qquad \textit{(Ecuación: 5.2)}$$

donde:

$$\forall\, m < n \qquad \textit{(Ecuación: 5.3)}$$

La matriz **I** pertenece al dominio de la imagen de tamaño **a** x **h**:

$$I_k(x,y) : \Omega \subset \mathbb{R}^{a x h} \qquad \textit{(Ecuación: 5.4)}$$

Donde **a** es el ancho de la imagen y **h** la altura.

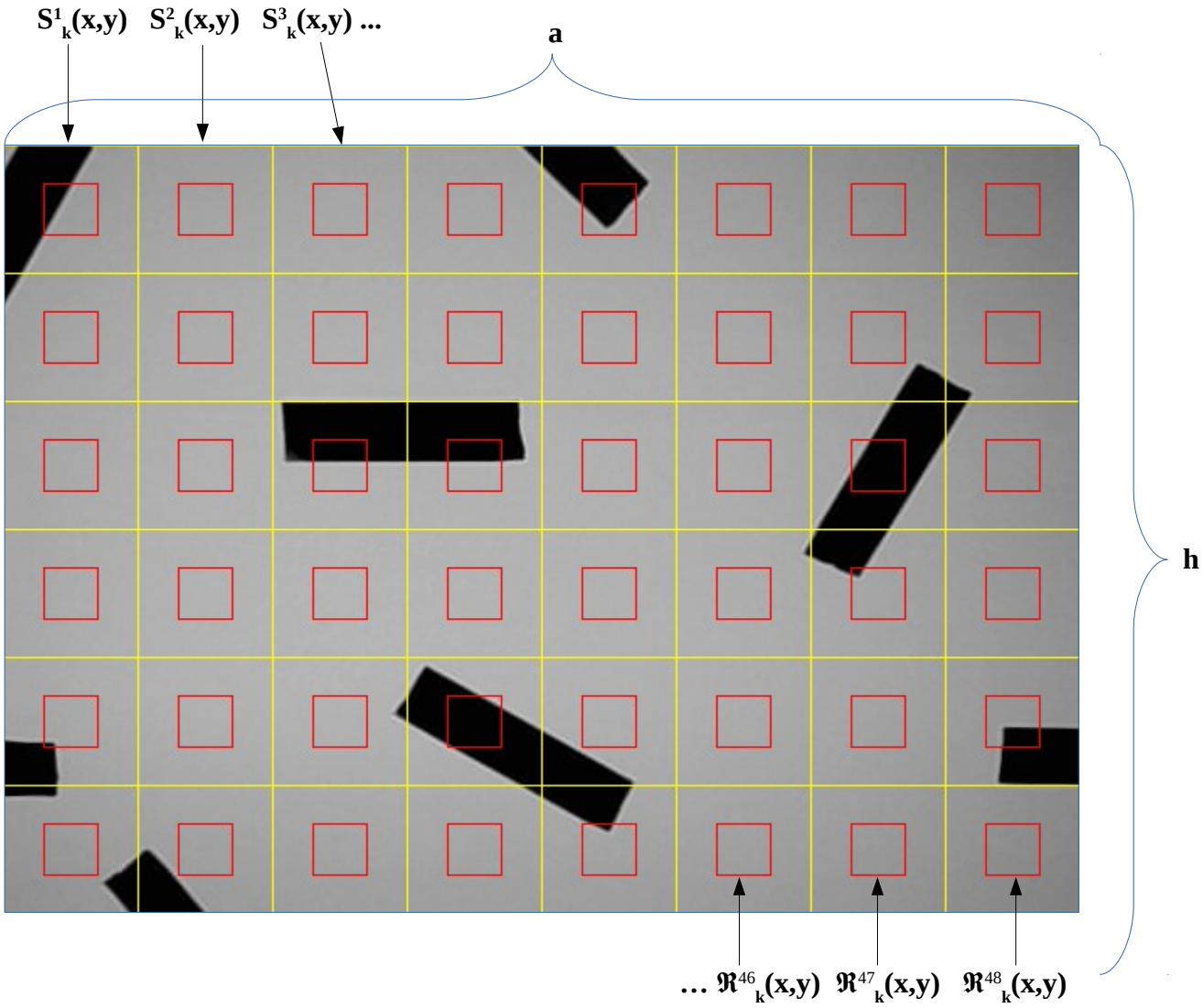


Figura 5.2: Distribución de parches

Al parche se le aplica un filtrado con el operador de Sobel vertical y horizontal, posteriormente se realiza una segmentación aplicando una binarización con un umbral **Δ**, se multiplican los resultados del Sobel vertical y horizontal binarizados, posteriormente se contarán los pixeles blancos, si la cantidad de pixeles blancos sobrepasan el umbral **Θ**, entonces será un parche candidato para seguimiento. Lo anteriormente expresado se podría definir de la forma:

$$\boldsymbol{W^h}(x,y) \subset \mathbb{R}^{w \times w} \wedge \boldsymbol{W^v}(x,y) \subset \mathbb{R}^{w \times w} \qquad \textit{(Ecuación: 5.5)}$$

$$\boldsymbol{W^h}(x,y)=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \qquad \textit{(Ecuación: 5.6)}$$

$$\boldsymbol{W^v}(x,y)=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \qquad \textit{(Ecuación: 5.7)}$$

Los argumentos **x** y **y** son independientes en cada función y determinan la posición de las matrices.

Se obtendrán los nuevos parches **H** y **V**, donde **H** es el área a la cual se le aplicó el operador Sobel horizontal y **V** es el área a la cual e le aplicó el operador Sobel vertical, ambos operadores aplicados sobre el parche $\mathfrak{R}$ de la imagen:

$$H_k^i(x,y)\subset\mathbb{R}^{m\,x\,m}\wedge V_k^i(x,y)\subset\mathbb{R}^{m\,x\,m} \qquad \textit{(Ecuación: 5.8)}$$

$$H_k^i(x,y)=\underbrace{\sum_{x'}^{w}\sum_{y'}^{w}W^h(x',y')\mathfrak{R}_k^i(x+x',y+y')}_{\textit{Convolución}} \qquad \textit{(Ecuación: 5.9)}$$

$$V_k^i(x,y)=\underbrace{\sum_{x'}^{w}\sum_{y'}^{w}W^v(x',y')\mathfrak{R}_k^i(x+x',y+y')}_{\textit{Convolución}} \qquad \textit{(Ecuación: 5.10)}$$

Los parches resultantes se pueden ejemplificar de la siguiente manera:

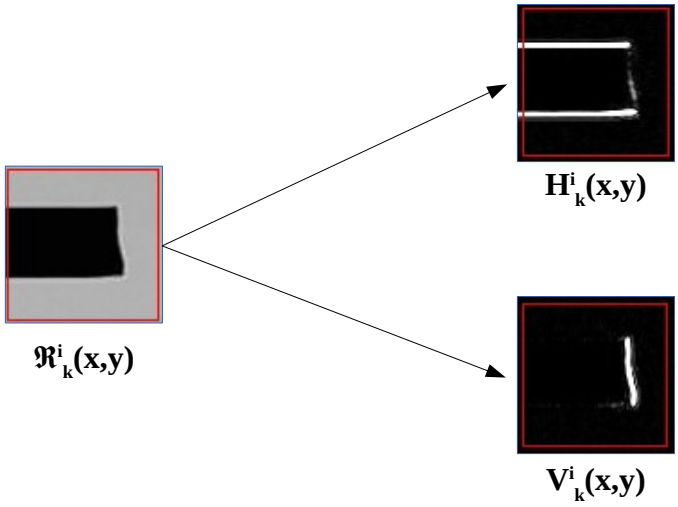


Figura 5.3: Parche con filtro Sobel

Esto se aplica para toda **x** y **y** de **H** y **V**, recordando que **i** es el índice de cada área de la imagen y **k** es el paso en el tiempo. Posteriormente se binariza tanto **H** como **V** de la forma:

$$Hb_k^i(x,y) \subset \mathbb{R}^{m\,x\,m} \wedge Vb_k^i(x,y) \subset \mathbb{R}^{m\,x\,m} \qquad \textit{(Ecuación: 5.11)}$$

$$Hb_k^i(x,y) = \begin{Bmatrix} 0 & \textbf{\textit{si}} \quad H_k^i(x,y) > \Delta \\ 1 & \textbf{\textit{si no}} \end{Bmatrix} \qquad \textit{(Ecuación: 5.12)}$$

$$Vb_k^i(x,y) = \begin{Bmatrix} 0 & si \quad V_k^i(x,y) > \Delta \\ 1 & si\,no \end{Bmatrix} \qquad \textit{(Ecuación: 5.13)}$$

Los parches resultantes de la binarización serían algo como lo que se muestra a continuación:

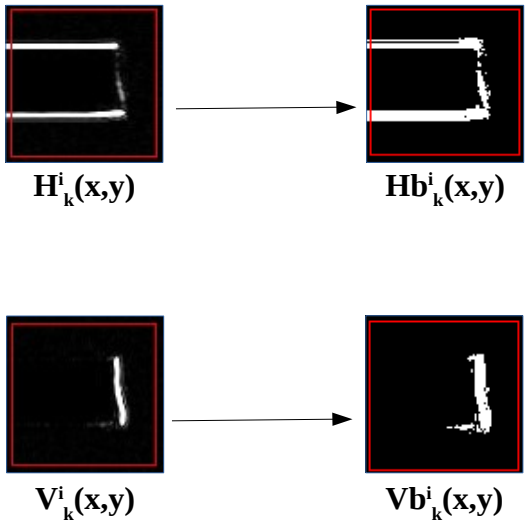


Figura 5.4: Parches binarizados

Los parches **Hb** y **Vb** son el resultado de la binarización con un umbral **Δ**, la binarización se aplica sobre todo **x** y **y**, donde posteriormente se multiplican ambas matrices.

$$\chi_k^i(x,y) \subset \mathbb{R}^{nxn} \qquad \textit{(Ecuación: 5.14)}$$

$$\chi_k^i(x,y) = Hb_k^i(x,y) \cdot Vb_k^i(x,y) \qquad \textit{(Ecuación: 5.15)}$$

El parche resultante se vería de la siguiente manera:

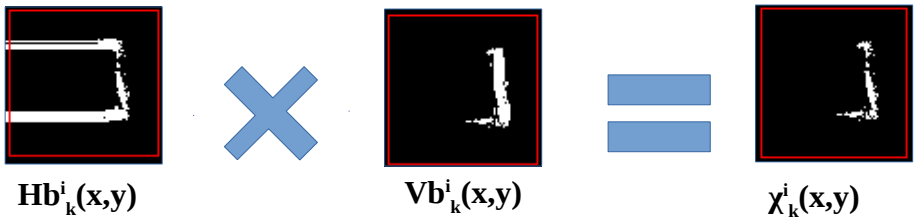


Figura 5.5: Parche resultante

Al resultado **χ** se le aplica una suma sobre todos sus pixeles, para ser evaluados con un umbral **Θ**:

$$\mathfrak{R}_k^j(x,y)=\left\{\mathfrak{R}_k^i(x,y) \quad \textbf{\textit{si}} \quad \sum_{x=0}^{m}\sum_{y=0}^{m}\chi_k^i(x,y)>\Theta\right\} \qquad \textit{(Ecuación: 5.16)}$$

Ahora, todos los parches seleccionados tienen el superíndice **j**.

5.2.3 Alineación y cálculo de traslación

Una vez seleccionados los parches, se procede a calcular el error entre los parches:

$$\delta E_k^j(x,y)=\sum_{x'=0}^{m}\sum_{y'=0}^{m}\left(\omega(x,y,\mathfrak{R}_k^j(x',y'))-\mathfrak{R}_{k-1}^j(x',y')\right)^2 \quad \textit{(Ecuación: 5.17)}$$

Donde **ω** es una función que mueve el parche sobre el espacio **S**, con esto obtenemos la función de costo:

$$\underset{x,y}{argmin} \quad m(x,y)=\delta E_k^j(x,y) \qquad \textit{(Ecuación: 5.18)}$$

Se calcula el movimiento **m** conformado por **x** y **y** por medio de un algoritmo de optimización, esto se aplica a todos los parches seleccionados de superíndice **j**, todas las **x** y **y** se almacenan en el vector **p**, cuyo tamaño es igual a **g**, que es la cantidad de parches seleccionados **j**, posteriormente se obtiene el vector de traslación **t** de la siguiente forma:

$$\boldsymbol{p}=\begin{bmatrix} x^j & y^j \\ x^{j+1} & y^{j+1} \\ x^{j+2} & y^{j+2} \\ \vdots & \vdots \\ x^g & y^g \end{bmatrix} \qquad \textit{(Ecuación: 5.19)}$$

$$t(x,y)=\frac{1}{g}\sum_{j=0}^{g}\boldsymbol{p}^j \qquad \textit{(Ecuación: 5.20)}$$

La traslación **x** y **y** de **t** es la media de los elementos del vector **p**.

5.3 Cálculo de rotaciones con IMU

La IMU utilizada trabaja a diferentes configuraciones, la configuración utilizada para el acelerómetro, es utilizando una alta resolución dando el valor 0x08 al registro 4, con una velocidad de 100hz asignando el valor 0x57 al registro 1, el magnetómetro trabaja a 30hz asignando al registro cero el valor 0x14, el giroscopio trabaja con una lectura de 2000 grados

por segundo asignando al registro 4 el valor 0x20. Todos estos sensores son utilizados en conjunto para el cálculo de las rotaciones, cada uno aporta valores importantes que se complementan entre sí.

Este sensor es ruidoso, por lo que en primera instancia debe de calibrarse antes de calcular los ángulos de cabeceo dado por **θ**, alabeo dado por **Φ** y guiñada dado por **ψ**.

5.3.1 Calibración de acelerómetro

Los ángulos obtenidos directamente del sensor tienen ruido y están desajustados por lo que se debe obtener el valor de desajuste (*offset*) de la siguiente manera, tomando en cuenta los valores ruidosos del acelerómetro:

$$\tilde{a} = \hat{a} + b_{acel} + n_{acel} \quad (5)$$ *(Ecuación: 5.21)*

Donde:

b_{acel} = Valor de calibración de acelerómetro

n_{acel} = Ruido de acelerómetro

$\tilde{a}$ = Aceleración aproximada

$\hat{a}$ = Aceleración por perturbaciones externas

g = Fuerza de gravedad

Primero de debe capturar **n** datos de la IMU sin perturbaciones, para posteriormente aplicar una media, considerando siempre que hay un vector de gravedad **g** y a la vez, en este caso se considera que el ruido es nulo .

$$\hat{b_{acel}} = \frac{1}{n}\sum_{k=1}^{n} \tilde{a}_k + g$$ *(Ecuación: 5.22)*

Esto se hace para cada eje, una vez obtenidos lo valores de calibración, se procede a aplicar la calibración:

$$\tilde{x} = \hat{x} + b_{acel}^{x} + n_{acel}^{x} + g^{x}$$ *(Ecuación: 5.23)*

$$\tilde{y} = \hat{y} + b_{acel}^{y} + n_{acel}^{y} + g^{y}$$ *(Ecuación: 5.24)*

$$\tilde{z} = \hat{z} + b^{z}_{acel} + n^{z}_{acel} + g^{z} \qquad \textit{(Ecuación: 5.25)}$$

Donde:

b^{x}_{acel} = Valor de calibración de acelerómetro para el eje **x**

b^{y}_{acel} = Valor de calibración de acelerómetro para el eje **y**

b^{z}_{acel} = Valor de calibración de acelerómetro para el eje **z**

n^{x}_{acel} = Ruido del acelerómetro en el eje **x**

n^{y}_{acel} = Ruido del acelerómetro en el eje **y**

n^{z}_{acel} = Ruido del acelerómetro en el eje **z**

$\tilde{x}$ = Aceleración aproximada del el eje **x**

$\tilde{y}$ = Aceleración aproximada del el eje **y**

$\tilde{z}$ = Aceleración aproximada del el eje **z**

$\hat{x}$ = Aceleración por perturbaciones externas en el eje **x**

$\hat{y}$ = Aceleración por perturbaciones externas en el eje **y**

$\hat{z}$ = Aceleración por perturbaciones externas en el eje **z**

g^{x} = Fuerza de gravedad proyectada sobre el el eje **x**

g^{y} = Fuerza de gravedad proyectada sobre el el eje **y**

g^{z} = Fuerza de gravedad proyectada sobre el eje **z**

Con la calibración aplicada y sin aceleraciones externas, se observa que:

$$\sqrt{\tilde{x}^2 + \tilde{y}^2 + \tilde{z}^2} \approx 9.81 \qquad \textit{(Ecuación: 5.26)}$$

5.3.2 Cálculo de ángulos con acelerómetro y magnetómetro

Para obtener los ángulos por medio del acelerómetro y megnetómetro, una vez calibrado, se le aplica lo siguiente:

$$\theta_{acel} = \arctan 2(Y_{acel}, Z_{acel})$$ *(Ecuación: 5.27)*

$$\phi_{acel} = \arctan\left(\frac{-X_{acel}}{Y_{acel}\sin(\theta_{acel})}\right) + Z_{acel}\cos(\theta_{acel})$$ *(Ecuación: 5.28)*

$$\psi_{acel} = \arctan 2(Z_{mag}\sin(\theta_{acel}) - Y_{mag}\cos(\theta_{acel}), X_{mag}\cos(\phi_{acel}) + Y_{mag}\sin(\phi_{acel})\sin(\theta_{acel}) + Z_{mag}\sin(\phi_{acel})\cos(\theta_{acel}))$$

(Ecuación: 5.29)

Tomando en cuenta que:

θ_{acel} = Ángulo de alabeo obtenido con el acelerómetro

ϕ_{acel} = Ángulo de cabeceo obtenido con el acelerómetro

ψ_{acel} = Ángulo de guiñada obtenido con acelerómetro y magnetómetro

X_{acel} = Aceleración en **x** (calibrada y filtrada)

Y_{acel} = Aceleración en **y** (calibrada y filtrada)

Z_{acel} = Aceleración en **z** (calibrada y filtrada)

X_{mag} = Valor de magnetómetro en el eje **x**

Y_{mag} = Valor de magnetómetro en el eje **y**

Z_{mag} = Valor de magnetómetro en el eje **z**

Los ángulos de alabeo, cabeceo y guiñado son obtenidos gracias a la proyección de la aceleración de la gravedad en cada uno de los ejes, los sensores y la respuesta es rápida y sin deriva, sin embargo, tiene bastante ruido y es susceptible a vibraciones de alta frecuencia.

5.3.3 Calibración de giroscopio

Para la calibración del giroscopio, se realiza una operación similar a la de la calibración del acelerómetro, se hace de la siguiente forma:

$$\tilde{\omega} = \omega + b_{\omega} + n_{\omega} \qquad \textit{(Ecuación: 5.30)}$$

Donde:

b_{ω} = Valor de calibración del giroscopio

ω = Velocidad angular

n_{ω} = Ruido del giroscopio

$\tilde{\omega}$ = Velocidad angular aproximada

Para obtener el valor de calibración del giroscopio se debe obtener **n** muestras para aplicar:

$$\hat{b_{\omega}} = \frac{1}{n}\sum_{k=1}^{n} \tilde{\omega}_k \qquad \textit{(Ecuación: 5.31)}$$

Esto se hace para cada uno de los ejes y una vez calibrado se obtiene:

$$\tilde{\omega}^{x} = \omega^{x} + b_{\omega}^{x} + n_{\omega}^{x} \qquad \textit{(Ecuación: 5.32)}$$

$$\tilde{\omega}^{y} = \omega^{y} + b_{\omega}^{y} + n_{\omega}^{y} \qquad \textit{(Ecuación: 5.33)}$$

$$\tilde{\omega}^{z} = \omega^{z} + b_{\omega}^{z} + n_{\omega}^{z} \qquad \textit{(Ecuación: 5.34)}$$

Donde:

b_{ω}^{x} = Valor de calibración del giroscopio para el eje **x**

b_{ω}^{y} = Valor de calibración del giroscopio para el eje **y**

b_{ω}^{z} = Valor de calibración del giroscopio para el eje **z**

ω^{x} = Velocidad angular en el eje **x**

ω^{y} = Velocidad angular en el eje **y**

ω^z = Velocidad angular en el eje **z**

n_ω^x = Ruido del giroscopio en el eje **x**

n_ω^y = Ruido del giroscopio en el eje **y**

n_ω^z = Ruido del giroscopio en el eje **z**

$\widetilde{\omega}^x$ = Velocidad angular aproximada en el eje **x**

$\widetilde{\omega}^y$ = Velocidad angular aproximada en el eje **y**

$\widetilde{\omega}^z$ = Velocidad angular aproximada en el eje **z**

Una vez aplicada la calibración, si existen perturbaciones externas, se podrá observar que:

$$\widetilde{\omega}^x \approx 0 \qquad \textit{(Ecuación: 5.35)}$$

$$\widetilde{\omega}^y \approx 0 \qquad \textit{(Ecuación: 5.36)}$$

$$\widetilde{\omega}^z \approx 0 \qquad \textit{(Ecuación: 5.37)}$$

5.3.4 Cálculo de ángulos con giroscopio

Para obtener los ángulos por medio del giroscopio, una vez calibrado, se le aplica lo siguiente:

$$\theta_{giro} = \int X_{giro}\, dt \qquad \textit{(Ecuación: 5.38)}$$

$$\phi_{giro} = \int Y_{giro}\, dt \qquad \textit{(Ecuación: 5.39)}$$

$$\psi_{giro} = \int Z_{giro}\, dt \qquad \textit{(Ecuación: 5.40)}$$

Donde las integraciones son implementadas con métodos numéricos usando la regla del trapecio, las variables de las ecuaciones anteriores son:

θ_{giro} = Ángulo de alabeo obtenido con giroscopio

ϕ_{acel} = Ángulo de cabeceo obtenido con giroscopio

ψ_{giro} = Ángulo de guiñada obtenido con giroscopio

X_{giro} = Velocidad angular sobre el eje **x** (calibrada y filtrada)

Y_{giro} = Velocidad angular sobre el eje **y** (calibrada y filtrada)

Z_{giro} = Velocidad angular sobre el eje **z** (calibrada y filtrada)

Los ángulos de alabeo, cabeceo y guiñado obtenidos con el giroscopio son precisos y no son afectados mucho por el ruido, sin embargo, tiene una deriva creciente que va aumentando conforme pasa el tiempo.

5.3.5 Filtro de señales

El filtrado de señales sirve para disminuir el ruido y combinar los valores obtenidos mediante un filtro complementario.

Filtro pasa bajas

Este filtro está hecho para evitar pasar altas frecuencias como las que se presentan en muchos de los sensores, en este caso se usará un filtro pasa bajo discreto de la siguiente forma:

$$\alpha > 0 \wedge \alpha < 1 \qquad \textit{(Ecuación: 5.41)}$$

$$\beta_{k+1} = \beta_{k-1} \cdot \alpha + \beta_k \cdot (1 - \alpha) \qquad \textit{(Ecuación: 5.42)}$$

Donde:

β = Señal

α = Valor del filtro

k = Paso en el tiempo

Filtro complementario

El filtro complementario combina los valores de sensores aprovechando lo mejor de cada uno, es similar al filtro anterior, esta dado de la siguiente manera:

$$\alpha > 0 \wedge \alpha < 1 \qquad \textit{(Ecuación: 5.43)}$$

$$\tau_{k+1} = \beta_k \cdot \alpha + \gamma_k \cdot (1 - \alpha) \qquad \textit{(Ecuación: 5.44)}$$

Donde:

τ = Es el nuevo valor resultante del filtro

β = Valor de sensor 1 (por ejemplo: ángulo por acelerómetro)

γ = Valor de sensor 2 (por ejemplo: ángulo por giroscopio)

α = Valor del filtro

k = Paso en el tiempo

El filtro complementario mejora los resultados de los sensores de la IMU ya que en el corto plazo se utilizan los datos del giroscopio, porque es muy preciso, no es susceptible al ruido ni a perturbaciones. A largo plazo, se utilizarán los datos del acelerómetro ya que no tiene deriva.

5.3.6 Cálculo de ángulo completo

A continuación se muestra de manera global el procedimiento para el cálculo de los ángulos:

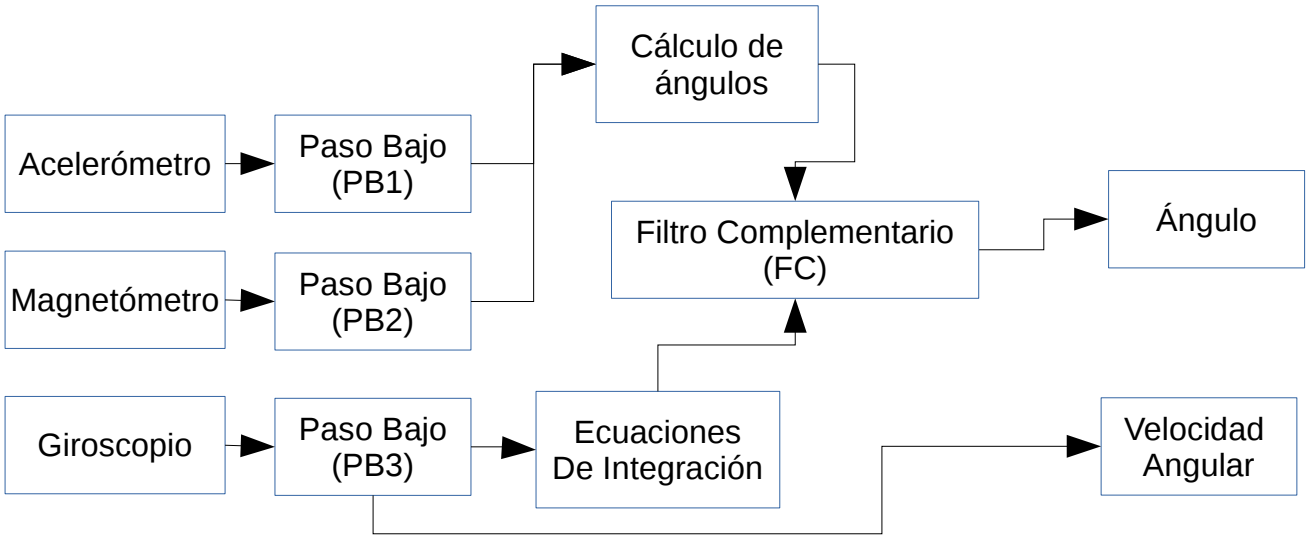


Figura 5.6: Diagrama de filtros

Tabla 5.1: Valores alfa de los filtros

Filtro	Alpha
PB1	0.9
PB2	0.9
PB3	0.6
FC	0.8

5.4 Fusión de sensores Visión-Inercial

Para la combinación de los sensores se representa la pose **T** con una matriz del grupo **SE**(3), de tal manera, que la parte **R** de la matriz será dada por la IMU y el vector **t** será dado por el algoritmo de visión.

$$\boldsymbol{T}=\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\\ 0 & 1\end{bmatrix}\in\boldsymbol{SE}(n) \qquad \textit{(Ecuación: 5.45)}$$

Durante el funcionamiento del algoritmo de odometría, ambas partes del algoritmo se estarán calculando, sin embargo, también se estará evaluando la velocidad angular **ω** sobre el eje **z**, ya que si sobrepasa el umbral **μ**, se dejarán de calcular las traslaciones del vector **t**, esto se debe a que el algoritmo de visión desarrollado no interpreta rotaciones y como consecuencia, las rotaciones pueden ser interpretadas como traslaciones. El umbral anteriormente mencionado puede variar según la velocidad del robot, sin embargo, en este caso será cuando sobrepase 3 grados por segundo, esto se determinó analizando los datos capturados.

Capítulo 6: Experimentación y Resultados

En esta parte se realizan experimentos del algoritmo, se hacen pruebas de los filtros propuestos para la IMU, se compara el filtro propuesto con ORB, se experimenta con el sistema completo recorriendo cinco diferentes circuitos en diferentes entornos y diferentes configuraciones.

6.1 Prueba de filtrado de IMU

Para filtrar la señal del IMU se utilizan filtros pasa bajos y un filtro complementario, las siguientes ilustraciones muestran la comparación de los ángulos calculados sin filtrar con los ángulos calculados con el filtro propuesto, la prueba se realizó durante 25 segundos, con una frecuencia de muestreo de 0.025 segundos, los resultados se muestran a continuación:

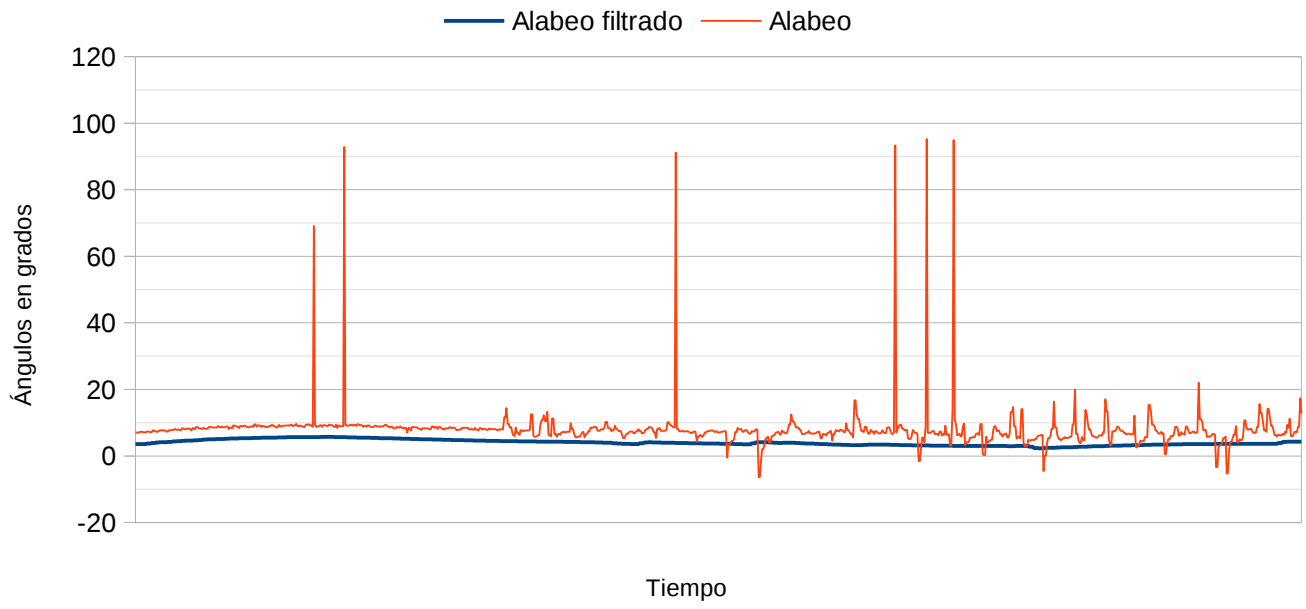


Figura 6.1: Señal de alabeo

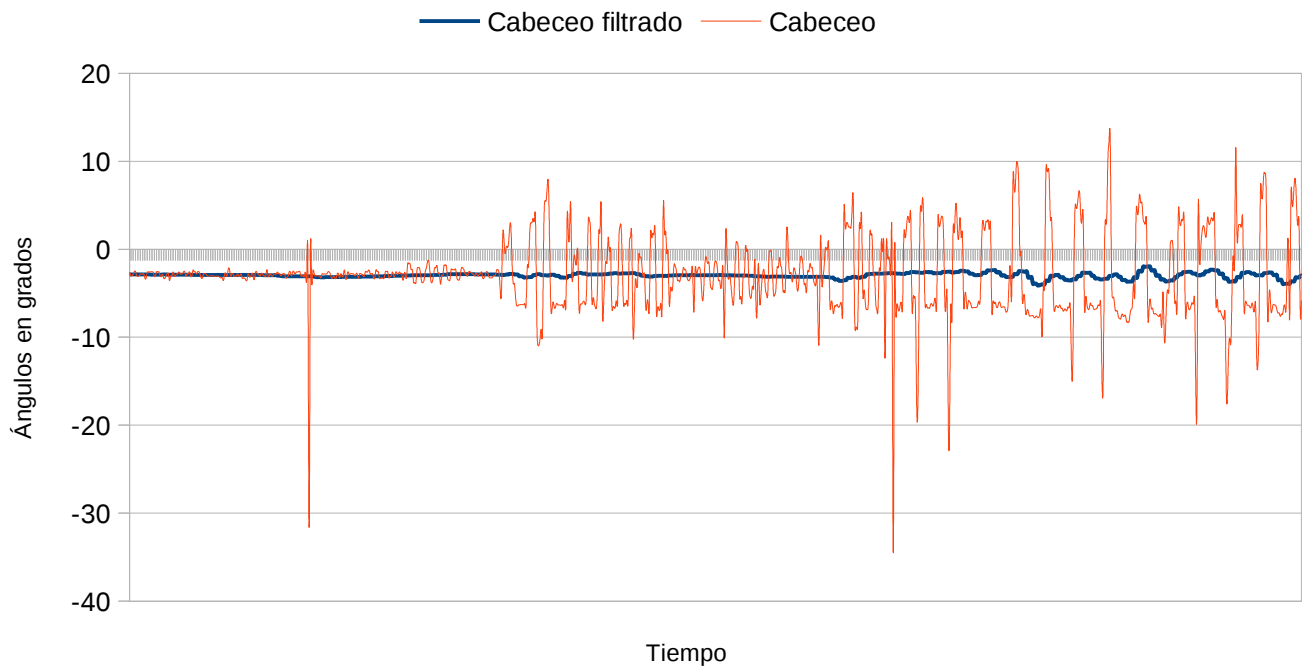


Figura 6.2: Señal de cabeceo

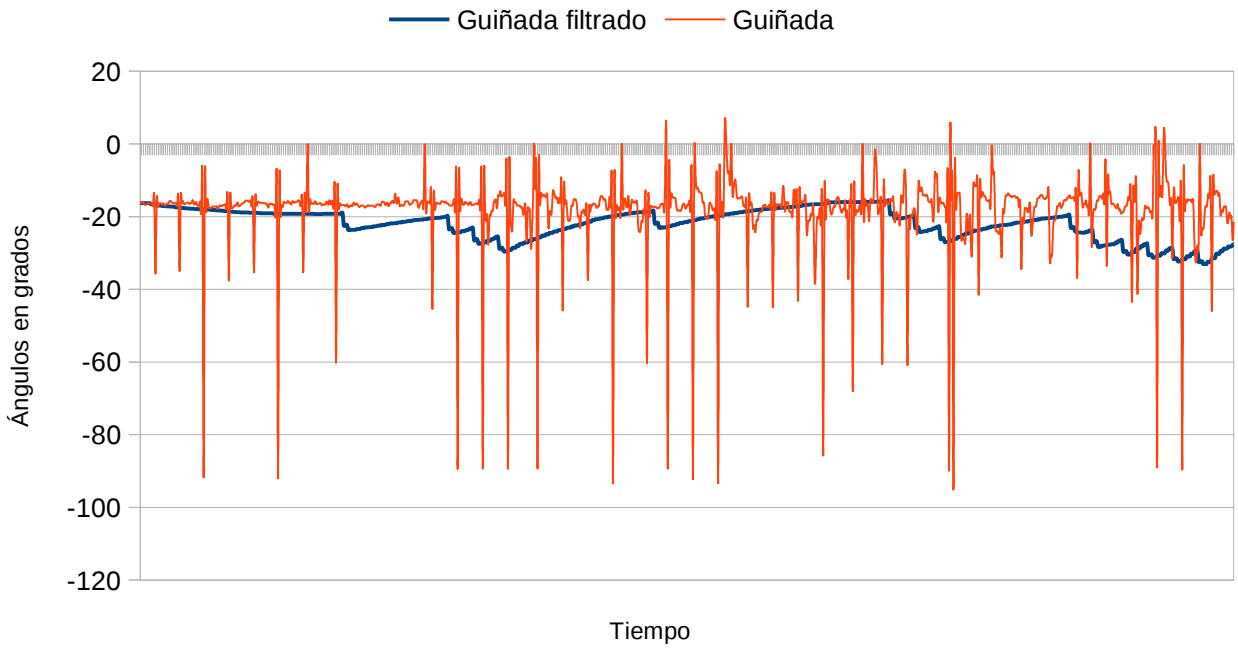
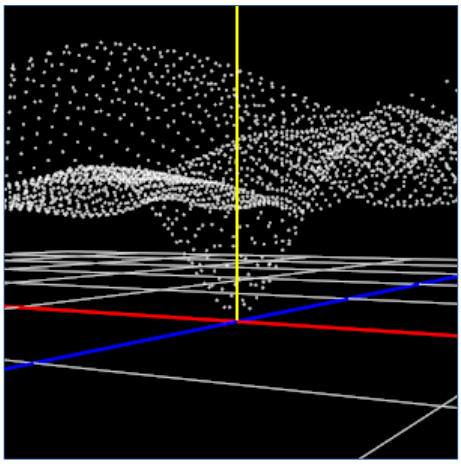


Figura 6.3: Señal de guiñada

6.2 Prueba de selección de parches

Para seleccionar los parches candidatos se comparó el filtro propuesto con ORB, mostrando que el filtro propuesto tiene mejor desempeño en velocidad, sin embargo, ORB da mejores resultados para obtener esquinas de mejor calidad en la mayoría de los casos, esto se puede ver en la función de error fotométrica generada cuando se toma como centro una característica de ORB para crear un parche, como se puede ver a continuación:

Punto ORB en la imagen



Error fotométrico

Figura 6.4: Error fotométrico de parche seleccionado por ORB

Por otra parte, mientras que el filtro propuesto da buenos resultados para propósito del método directo desarrollado, también hay que tener en cuenta que el filtro deja pasar ruido si no se seleccionan bien los umbrales, como se puede ver a continuación, la función generada es parecida:

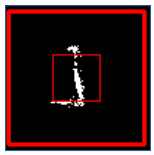
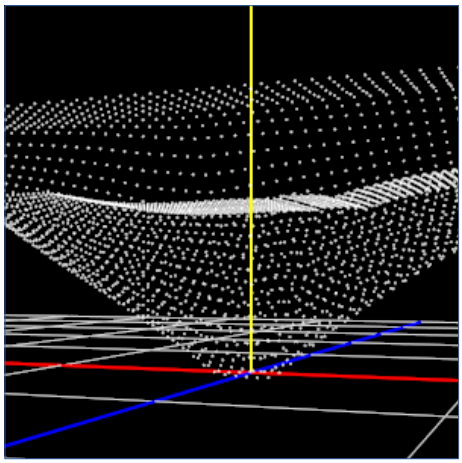


Imagen con filtro propueto



Error fotométrico

Figura 6.5: Error fotométrico de parche seleccionado por filtro

6.2.1 Prueba de rendimiento

Para su rendimiento, se midió el tiempo de procesamiento del filtro propuesto y ORB en imágenes conocidas que pueden encontrarse en el siguiente enlace:

https://homepages.cae.wisc.edu/~ece533/images/

A continuación se muestran algunas pruebas de rendimiento realizadas en la Laptop Dell Inspiron 14, tomando el tiempo en segundos, se realizaron tres ejecuciones sobre la misma imagen:

Tabla 6.1: Comparación filtro propuesto y ORB

Prueba	Imagen	Tamaño	Tiempo de ORB	Tiempo del filtro propuesto
1	lena.png	512x512	0.016834	0.005564
2			0.011813	0.005336
3			0.017047	0.005335
4	baboon.png	512x512	0.024105	0.005526
5			0.021811	0.005277
6			0.025615	0.00568
7	girl.png	768x512	0.017704	0.010626
8			0.018682	0.01055
9			0.015278	0.010218

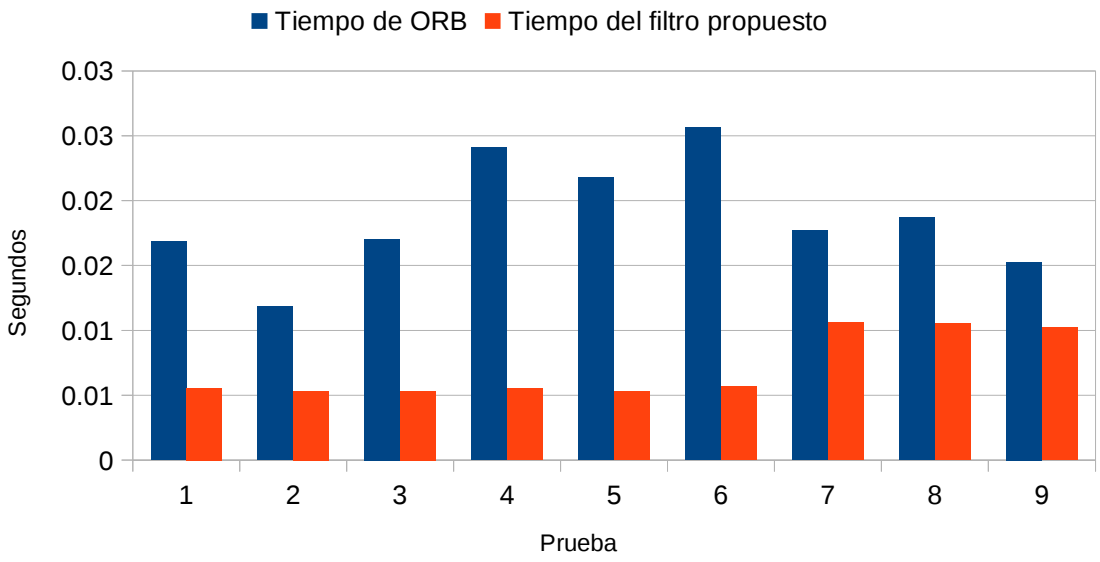


Figura 6.6: Comparación de filtro propuesto y ORB

6.3 Pruebas en circuitos y generación de mapa 3D

Para evaluar su desempeño se calculó el error en circuitos cerrados midiendo la distancia euclidiana de la posición final al punto de origen, donde es mejor cuanto más cercano a %0 sea, la ecuación es la siguiente:

$$Error = 100 - \frac{Distancia\,recorrida * 100}{Distancia\,recorrida + Posición\,actual\,al\,Origen} \quad \textit{(Ecuación: 6.1)}$$

Otra métrica son los cuadros por segundo procesados que será de ayuda para medir la velocidad del algoritmo:

$$Cuadros\ por\ segundo = \frac{Imágenes\ procesadas}{Tiempo\ transcurrido\ en\ segundos} \quad \textit{(Ecuación: 6.2)}$$

Por último, otra métrica de relación entre el tiempo de procesamiento y el tiempo real.

$$Relación\ tiempo\ real = 100 * \frac{Tiempo\ de\ proceamiento}{Tiempo\ real\ del\ recorrido} \quad \textit{(Ecuación: 6.3)}$$

6.3.1 Algoritmo de visión

El algoritmo de visión fue probado en una laptop Dell Inspiron 14, se evaluó en 5 circuitos diferentes y tres superficies diferentes, el circuito uno y dos se realizaron sobre una superficie de papel periódico, el circuito tres y cuatro fue sobre suelo de cemento y el circuito 5 fue sobre una mesa de plástico. El circuito uno es un rectángulo, el dos tiene forma de trapecio, el circuito tres es un cuadrado, el circuito cuatro y cinco tienen forma triangular, las imágenes se grabaron a 90 cuadros por segundo con una iluminación de tipo LED con la que cuenta el robot.

Circuito 1:

El circuito uno es un rectángulo de 20 cm por 12 cm, con una distancia total de 64 cm, la superficie del recorrido es papel periódico y el recorrido consta de 6600 imágenes capturadas en 73 segundos. A continuación se muestra un mapa que se generó durante el recorrido:

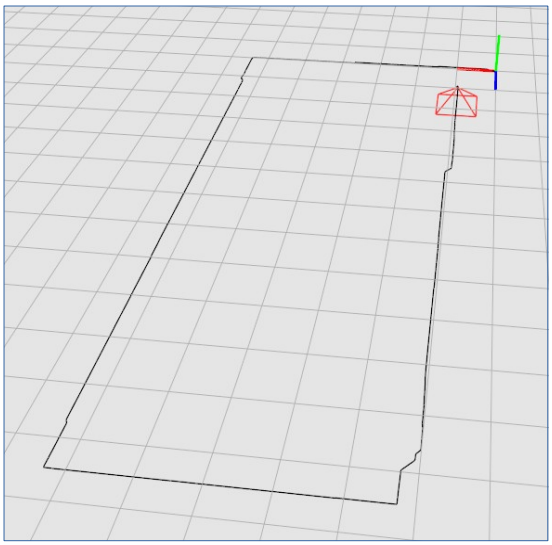


Figura 6.7: Circuito 1

Se realizaron nueve pruebas en este circuito obtenido los siguientes resultados:

Tabla 6.2: Circuito 1

Prueba	Tamaño de parche	Área de búsqueda	Umbral de filtro	Cuadros por segundo	Relación tiempo real	Error
1	32	60	8	24.57	366.30%	4.13%
2	40	60	8	24.42	368.55%	4.19%
3	64	80	8	23.75	378.95%	4.76%
4	32	60	16	26.1	344.83%	4.39%
5	40	60	16	24.78	363.20%	4.41%
6	64	80	16	24.26	370.98%	4.62%
7	32	60	24	25.36	354.89%	4.42%
8	40	60	24	24.97	360.43%	4.43%
9	64	80	24	24.06	374.06%	4.58%

Tanto el error como la velocidad de cuadros por segundo y la relación con el tiempo real tienen muy pocas variaciones, teniendo la prueba uno como la que obtuvo menor error con un porcentaje de 4.13%, mientras que la prueba tres fue la que obtuvo el error más grande con 4.76%. Esto se puede apreciar en la siguiente figura:

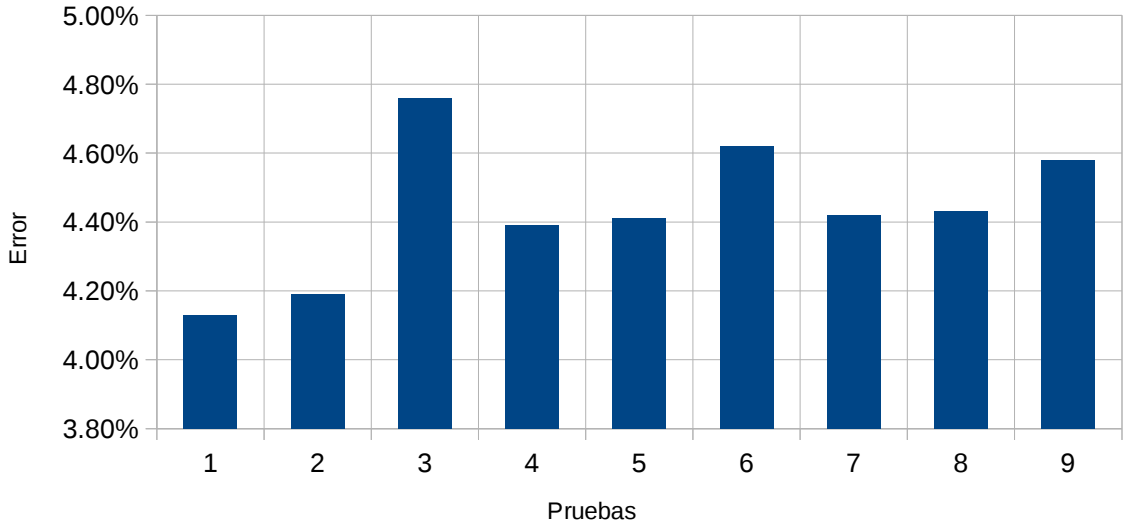


Figura 6.8: Error de pruebas del circuito 1

En cuestión de velocidad, la prueba cuatro fue la mejor teniendo una velocidad de 26.1 cuadros por segundo, como se puede apreciar continuación:

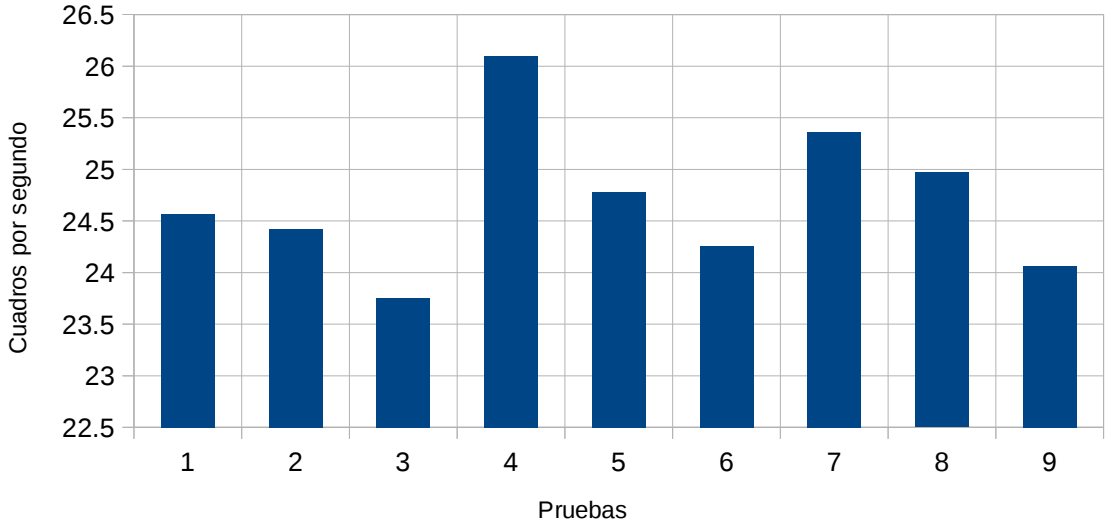


Figura 6.9: Cuadros por segundo de pruebas del circuito 1

Circuito 2:

El circuito dos es un trapecio con un perímetro de 56 cm, sus dimensiones son 20cm por 11cm por 14cm por 11cm, la superficie del recorrido es papel periódico y consta de 6639 imágenes capturadas en 73 segundos. A continuación se muestra un mapa que se generó durante el recorrido:

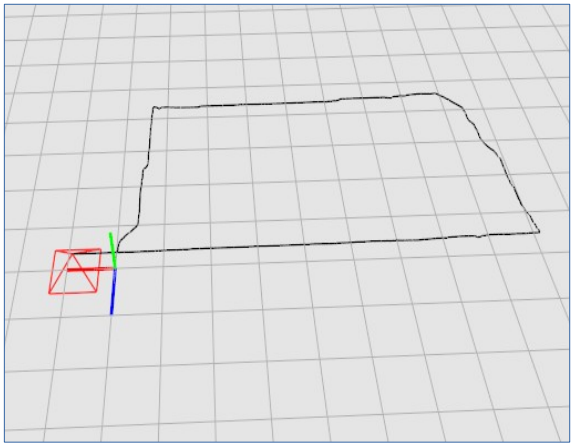


Figura 6.10: Circuito 2

Durante las nueve pruebas e obtuvieron los siguientes datos:

Tabla 6.3: Circuito 2

Prueba	Tamaño de parche	Área de búsqueda	Umbral de filtro	Cuadros por segundo	Relación tiempo real	Error
1	32	60	8	34.48	261.02%	4.78%
2	40	60	8	32.76	274.73%	5.27%
3	64	80	8	31.48	285.90%	6.48%
4	32	60	16	36.22	248.48%	4.95%
5	40	60	16	34.21	263.08%	5.54%
6	64	80	16	33.21	271.00%	6.63%
7	32	60	24	35.93	250.49%	5.04%
8	40	60	24	37.56	239.62%	5.53%
9	64	80	24	31.53	285.44%	6.63%

A comparación del circuito uno, el error resultó más variado, la prueba uno resultó ser la

mejor valorada con un error de 4.78%, mientras que las pruebas seis y nueve obtuvieron los peores resultados con un error de 6.63%. Esto se puede apreciar en la siguiente figura:

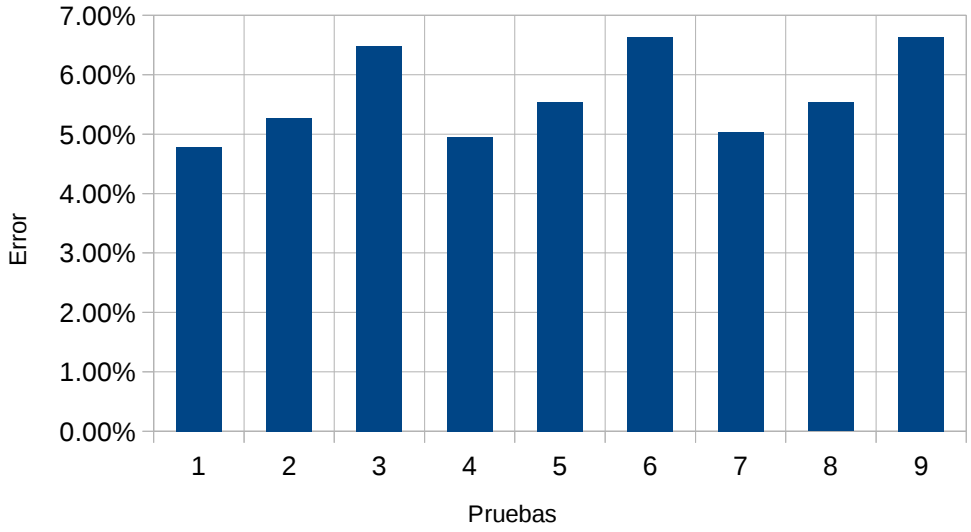


Figura 6.11: Error de pruebas del circuito 2

En la velocidad de procesamiento se tiene la prueba ocho como la más rápida, alcanzando 37.56 cuadros por segundo, como se muestra a continuación:

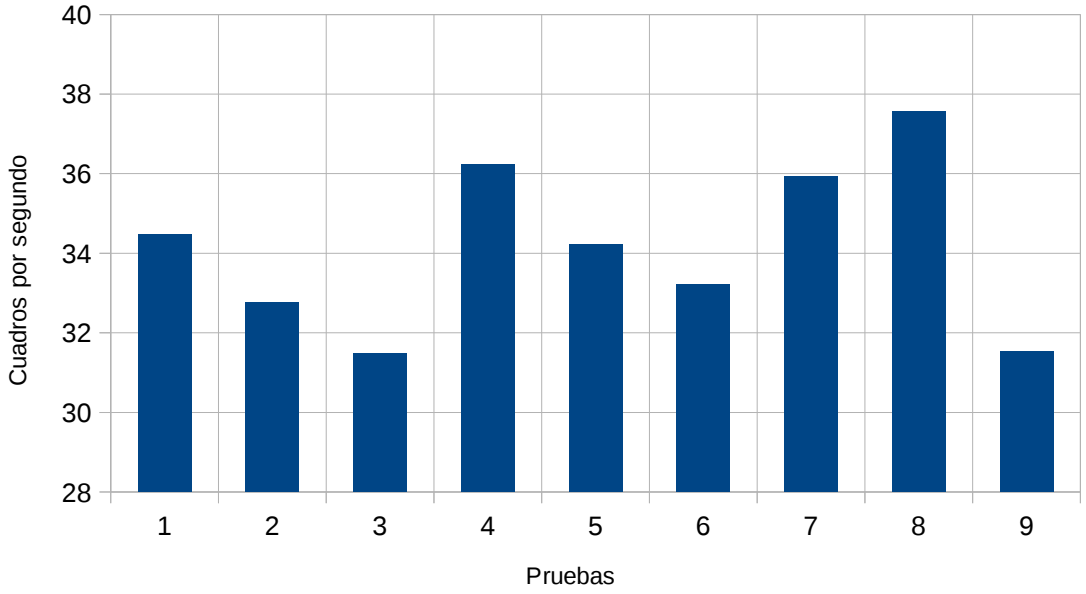


Figura 6.12: Cuadros por segundo de pruebas del circuito 2

Circuito 3:

Este circuito es un cuadrado de 25 cm por 25 cm, lo que da un total de 100 cm de perímetro para el recorrido, la superficie del circuito es de cemento con pocas texturas y el recorrido consta de 9487 imágenes capturadas en 105 segundos. A continuación se muestra un mapa de la trayectoria generado por el algoritmo:

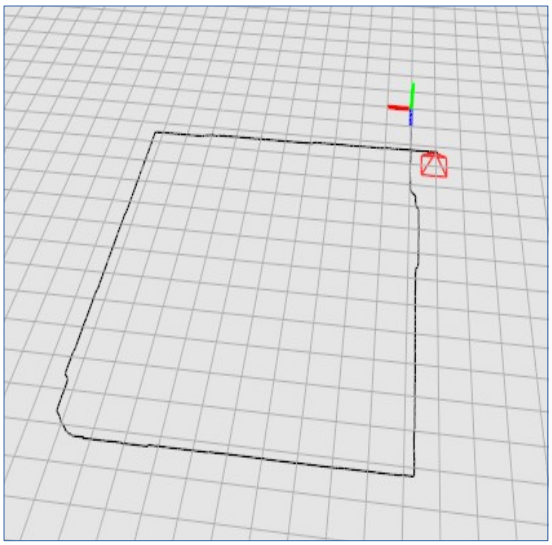


Figura 6.13: Circuito 3

La información de nueve pruebas realizadas se encuentra en la siguiente tabla:

Tabla 6.4: Circuito 3

Prueba	Tamaño de parche	Área de búsqueda	Umbral de filtro	Cuadros por segundo	Relación tiempo real	Error
1	32	60	8	34.16	263.47%	7.11%
2	40	60	8	34.6	260.12%	7.41%
3	64	80	8	30.8	292.21%	7.70%
4	32	60	16	35.19	255.75%	7.36%
5	40	60	16	32.25	279.07%	7.48%
6	64	80	16	31.77	283.29%	7.70%
7	32	60	24	35.25	255.32%	7.67%
8	40	60	24	32.92	273.39%	7.67%
9	64	80	24	31.17	288.74%	7.75%

En este circuito el error no es muy variable, resultando la prueba uno con la mejor valoración con un valor de 7.11%, mientras que la prueba nueve fue la que obtuvo peores resultados con un error de 7.75%, como se muestra a continuación:

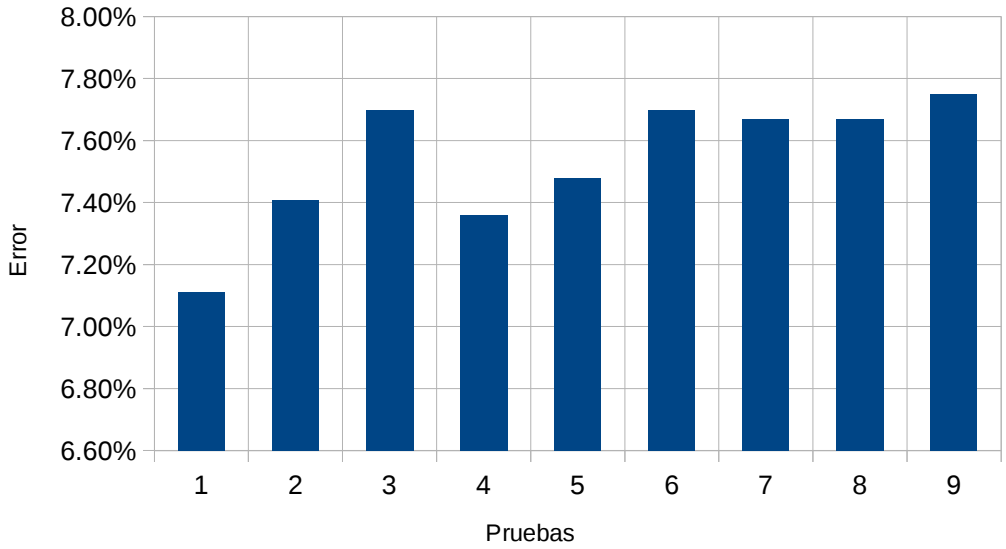


Figura 6.14: Error de pruebas del circuito 3

Para la velocidad de procesamiento, la prueba siete fue la mejor con una velocidad de 35.25 cuadros por segundo, esto se muestra a continuación:

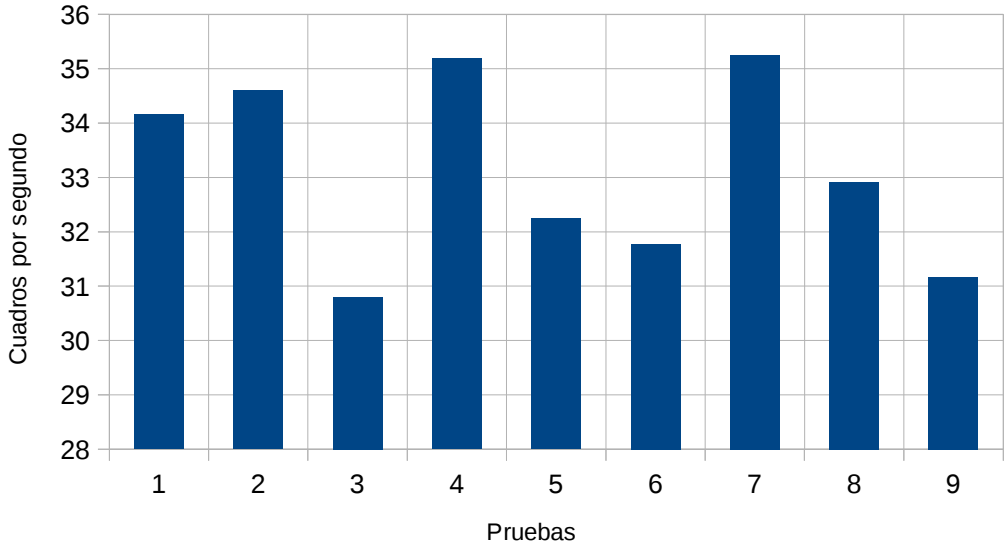


Figura 6.15: Cuadros por segundo de pruebas del circuito 3

Circuito 4:

El circuito tiene forma triangular con dimensiones de 25cm por 25cm por 35cm, la superficie es de cemento con pocas texturas y consta de 7037 imágenes capturadas en 78 segundos. El mapa de la trayectoria generado por el algoritmo es como el que se muestra a continuación:

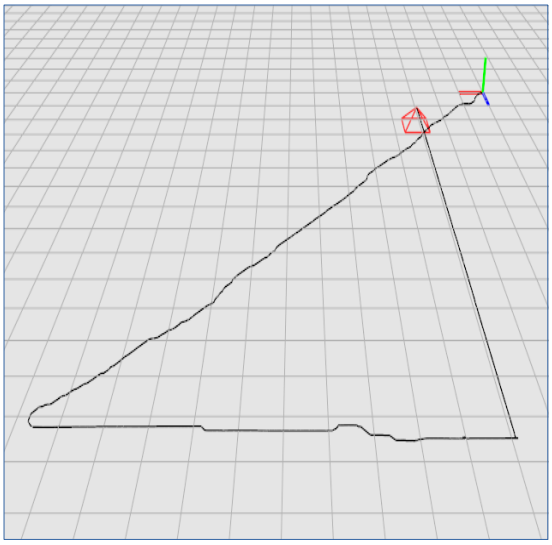


Figura 6.16: Circuito 4

A continuación e presenta los resultados obtenidos de nueve pruebas:

Tabla 6.5: Circuito 4

Prueba	Tamaño de parche	Área de búsqueda	Umbral de filtro	Cuadros por segundo	Relación tiempo real	Error
1	32	60	8	33.26	270.60%	7.86%
2	40	60	8	28.63	314.36%	8.01%
3	64	80	8	31.67	284.18%	8.55%
4	32	60	16	34.58	260.27%	9.20%
5	40	60	16	32.67	275.48%	8.73%
6	64	80	16	26.04	345.62%	8.57%
7	32	60	24	35.84	251.12%	10.02%
8	40	60	24	33.66	267.38%	9.55%
9	64	80	24	31.26	287.91%	8.83%

El error resultante de las pruebas fue muy variado donde la primera prueba fue la que obtuvo mejor desempeño con un error fue de 7.86%, mientras que la prueba siete fue la peor con un error del 10.02%. La figura que se presenta a continuación presenta la información de manera gráfica:

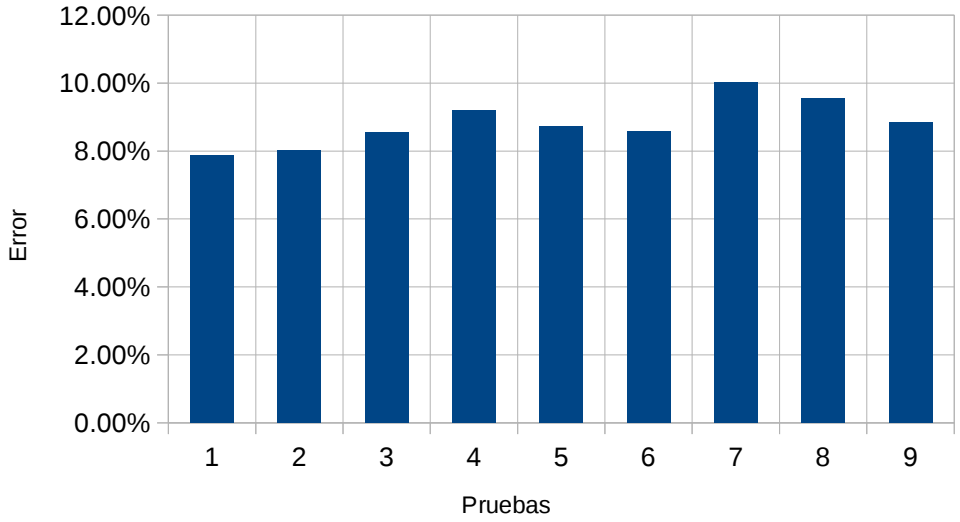


Figura 6.17: Error de pruebas del circuito 4

La prueba que dio mejor rendimiento en velocidad fue la prueba siete con 35.84 cuadros por segundo, como se puede ver a continuación:

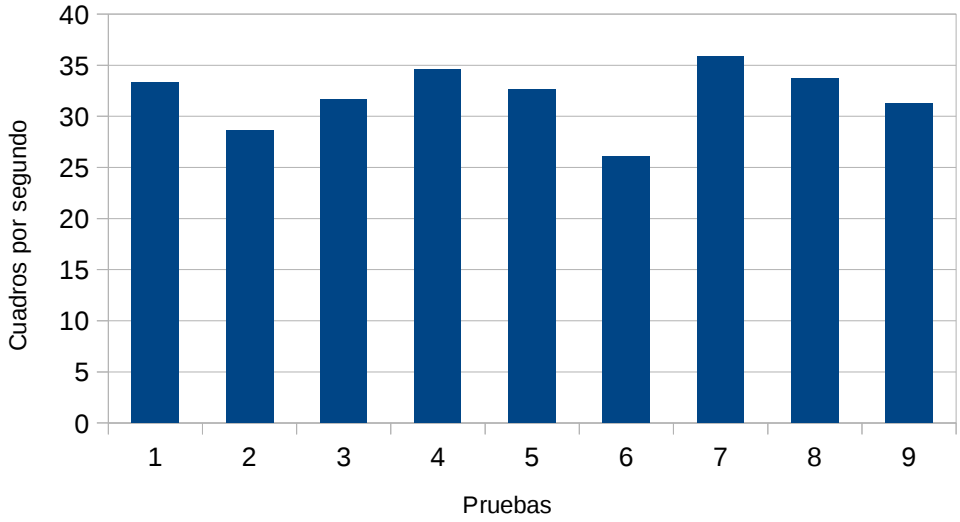


Figura 6.18: Cuadros por segundo de pruebas del circuito 4

Circuito 5:

Este circuito tiene forma triangular con un un perímetro de 102cm, las dimensiones son 30cm por 30 cm por 42 cm y la superficie es de una mesa de plástico, que al igual que las superficies de cemento, es lisa con pocas texturas, el recorrido tiene 8074 imágenes que fueron capturadas en 89 segundos. Un ejemplo del mapa de la trayectoria generado por el algoritmo es como el que se muestra a continuación:

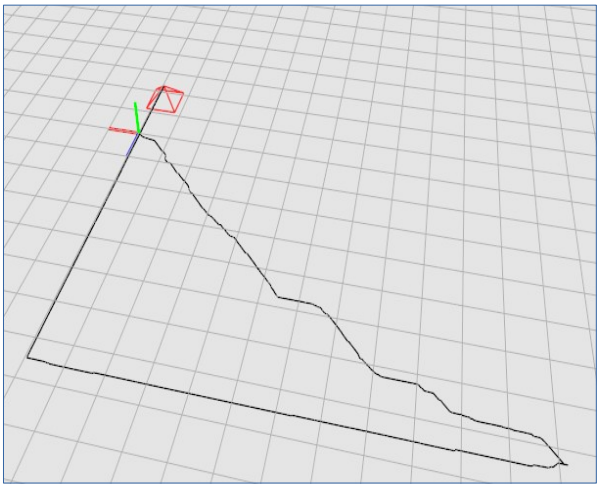


Figura 6.19: Circuito 5

La información resultante de nueve pruebas se muestra a continuación:

Tabla 6.6: Circuito 5

Prueba	Tamaño de parche	Área de búsqueda	Umbral de filtro	Cuadros por segundo	Relación tiempo real	Error
1	32	60	8	39.11	230.12%	12.60%
2	40	60	8	33.27	270.51%	12.80%
3	64	80	8	31.38	286.81%	13.54%
4	32	60	16	42.37	212.41%	21.23%
5	40	60	16	41.26	218.13%	7.00%
6	64	80	16	39.58	227.39%	8.66%
7	32	60	24	43.03	209.16%	22.65%
8	40	60	24	41.94	214.59%	7.77%
9	64	80	24	41.79	215.36%	9.92%

El error mínimo fue de la prueba cinco con un error del 7%, mientras que la prueba con el mayor error fue la siete con un error del 22.65%, esto se puede apreciar en la siguiente figura:

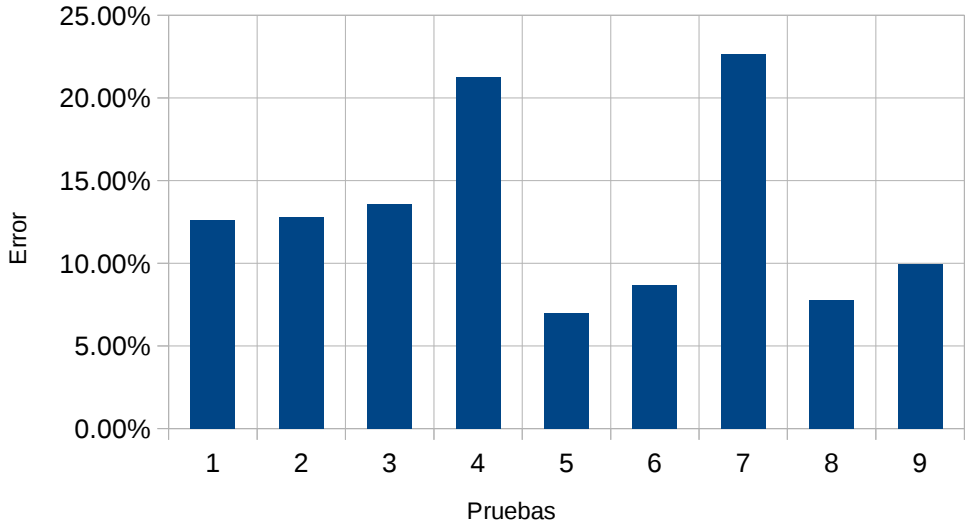


Figura 6.20: Error de pruebas del circuito 5

La prueba que obtuvo mayor velocidad fue la siete con 43.03 cuadros por segundo como se muestra a continuación:

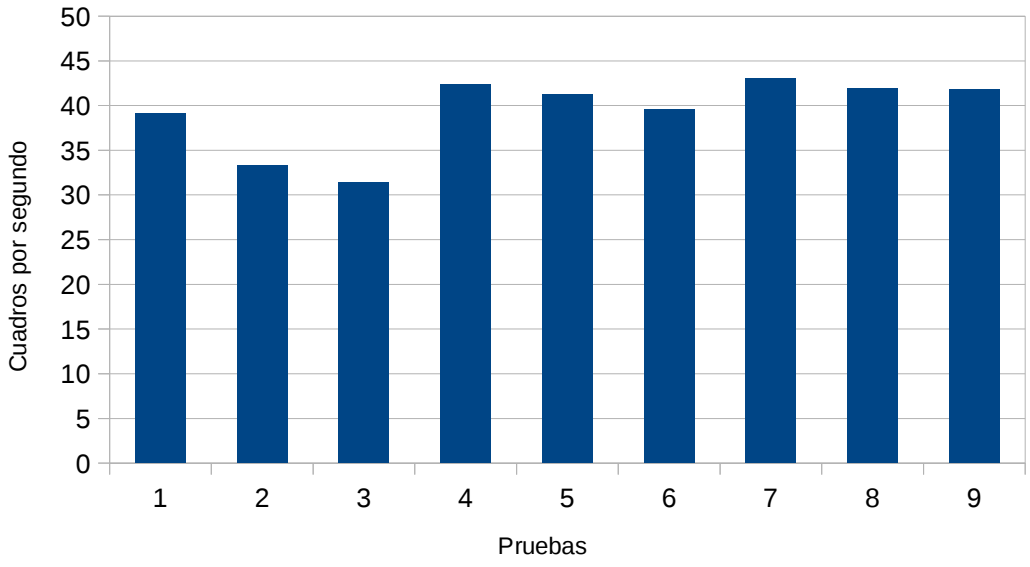


Figura 6.21: Cuadros por segundo de pruebas del circuito 5

Discusión

Como se puede ver en los experimentos, los dos primeros circuitos dieron mejores resultados, esto es porque la superficie del papel periódico posee grandes contrastes que facilita el seguimiento de los parches, por otra parte, la mesa de plástico obtuvo el peor desempaño, ya que la superficie es muy lisa y con pocos contrastes. Comparando el error entre los circuitos se puede observar lo siguiente:

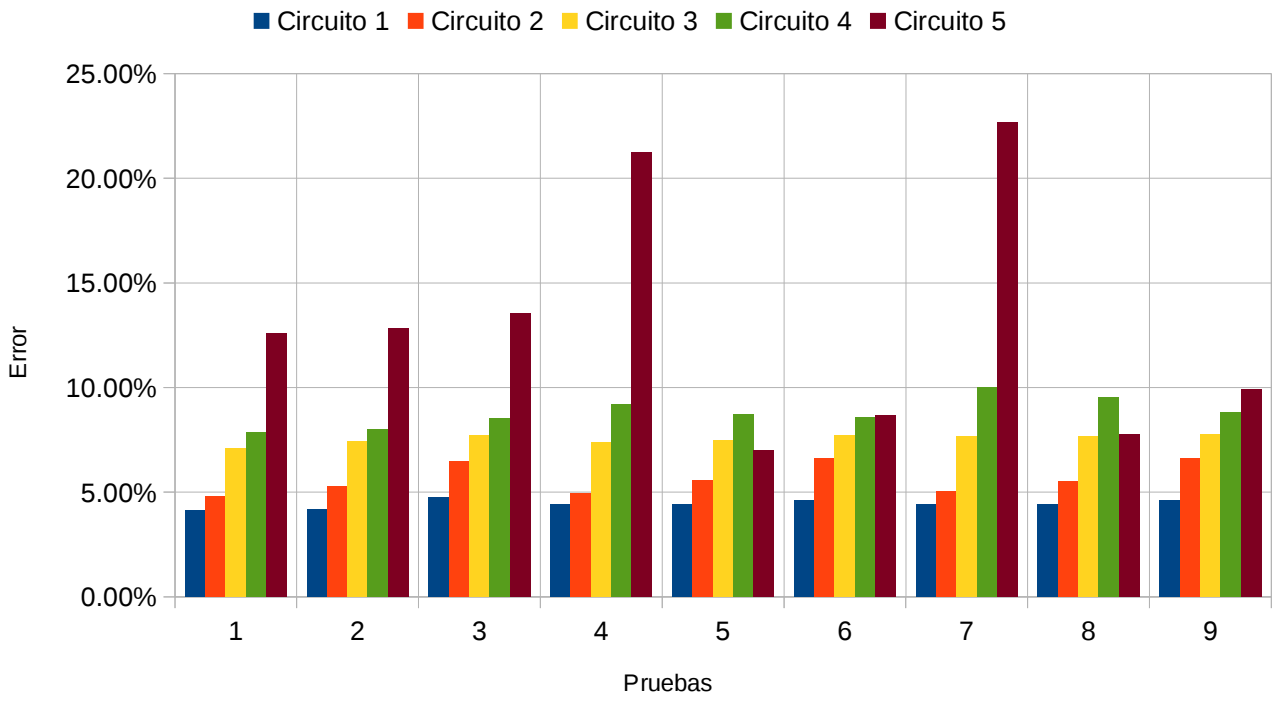


Figura 6.22: Comparación de error entre circuitos

Otro factor importante, fue el umbral del filtro, ya que a menor sea el umbral, mayor cantidad de pixeles podrán ser procesados, en superficies lisas con pocos contrastes conviene usar un umbral bajo. Por ejemplo, si comparamos el error según los umbrales con el mismo tamaño de parche y área de búsqueda de cada circuito, tenemos:

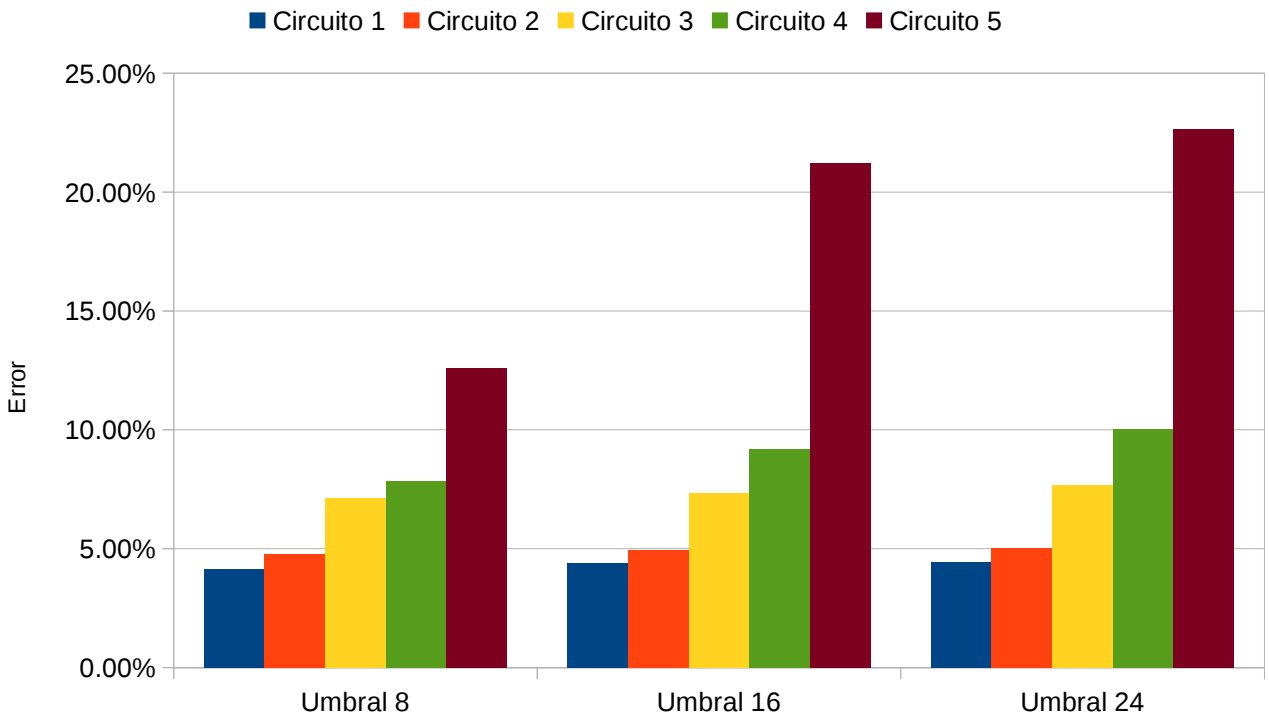


Figura 6.23: Comparación de umbral entre circuitos

Por último el tamaño de los parches y espacio de búsqueda toma relevancia según la velocidad de la cámara, si se tiene una cámara rápida, se tendrá buen rendimiento con áreas pequeñas de búsqueda, mientra que si la cámara es lenta, se tendrá mejores resultados con áreas de búsqueda grandes, cabe destacar que además de la velocidad de la cámara, la velocidad en que se mueve el vehículo también interviene mucho, esto se puede ver en el circuito cinco donde se tiene mejores resultados agrandando el parche, como se puede ver a continuación:

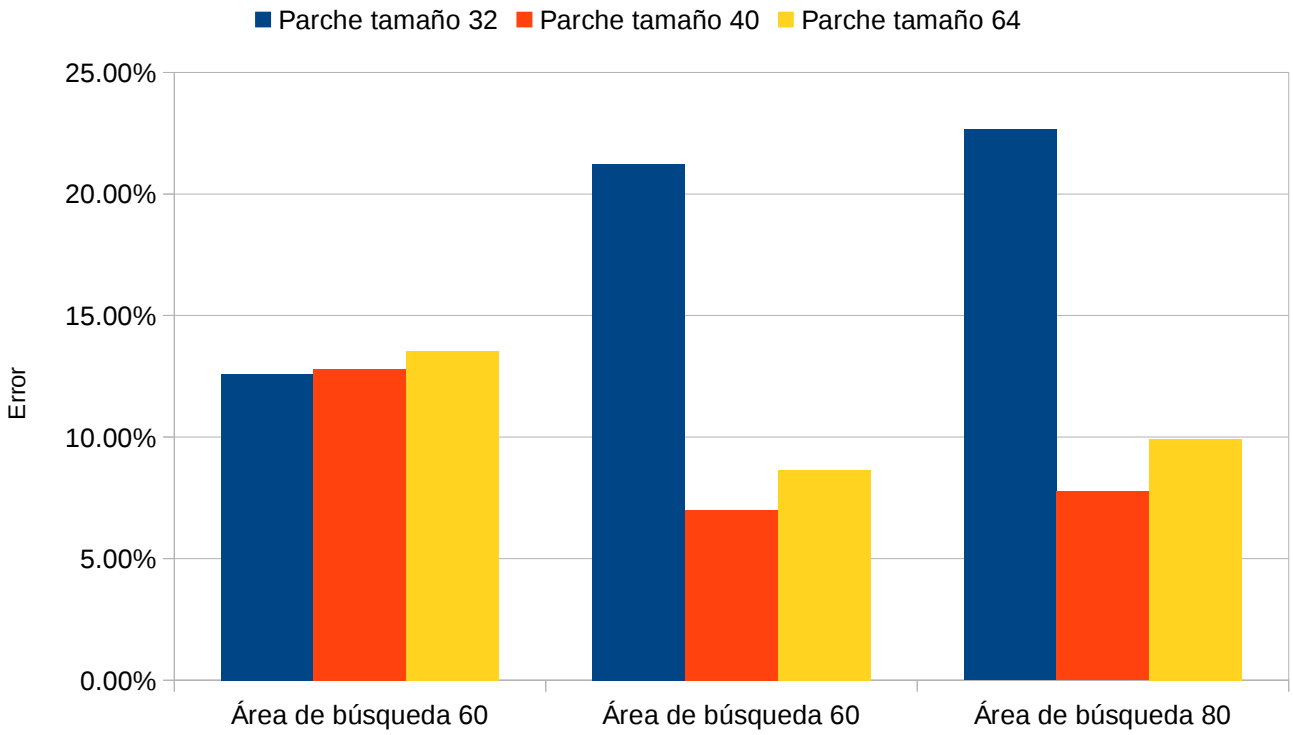


Figura 6.24: Comparación de tamaño de parche

La velocidad depende principalmente de la superficie, cuando se tienen superficies con texturas y altos contrastes, se seleccionarán más parches para seguir, mientras que en superficies lisas con pocas texturas se seguirán menos parches y por ende, el costo de cómputo será menor, comparando los cuadros por segundo de el circuito uno con el circuito cinco se podrán ver estos resultados:

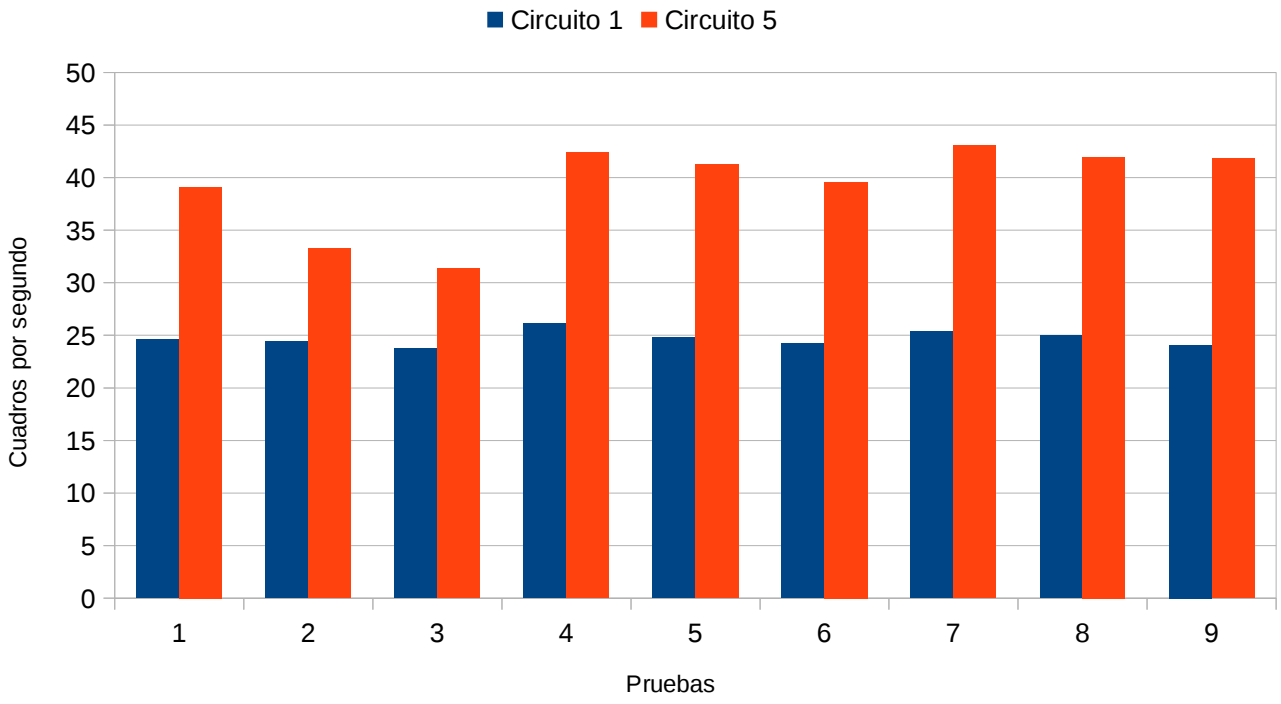


Figura 6.25: Cuadros por segundo de circuito 1 y circuito 5

Capítulo 7: Conclusión y Trabajo Futuro

En este capítulo se presentan las conclusiones finales, así como los productos, aportaciones y trabajo futuro.

7.1 Conclusiones finales

Este trabajo tiene como propósito adentrarse en los métodos directos para describirlos y desarrollar un algoritmo basándose en estos métodos, que funcionen en tiempo real y aprovechar su precisión, sin embargo, a pesar de que se utilizó una IMU para reducir el costo computacional de estos métodos y se usaron transformaciones muy simples, el costo computacional sigue siendo alto y no es posible trabajar con ellos en tiempo real a la perfección, se necesitan movimientos muy pequeños de la cámara para evitar que estos algoritmos pierdan el seguimiento del parche o región de pixeles, si los parches de pixeles se alejan mucho de un intervalo de tiempo a otro, la función a optimizar se vuelve sumamente

complicada con óptimos locales que hace el problema intratable con métodos de optimización convencionales. Utilizar transformaciones de mayores magnitudes como la homografía o una trasformación afín, mejora los resultados con respecto a la precisión [Forster, 2014], sin embargo el tiempo de cómputo crece exponencialmente, la ventaja de la precisión subpixel solo se puede aprovechar en transformaciones que hacen interpolaciones, como lo es la trasformación afín, similar, euclidiana u homografía, si se hace una trasformación donde solo se desplaza en los dos ejes de la imagen, no se verán estas ventajas, esto se vio en el estado del arte donde los mejores resultados eran teniendo trasformaciones de este tipo [Forster, 2014][Engel, 2014].

Posiblemente la odometría no sea la mejor aplicación para los métodos directos, sino la reconstrucción de escenas, ya que aprovecha toda la información de la imagen generando reconstrucciones 3D con mucho mayor detalle que los algoritmo que utilizan características [Newcombe, 2015][Engel, 2014][Stumberg, 2018].

Anteriormente en TecNM-CENIDET se probaron los métodos directos con resultados buenos en porcentaje de error, teniendo un mínimo de 0.02% de error y un máximo de 12.51% de error, mientras que en este trabajo se obtuvo un error mínimo de 4.13% de error y un máximo de 22.65%, sin embargo en este trabajo se procesaron miles de imágenes en unos cuantos minutos, mientras que en trabajos anteriores de procesaban cientos de imágenes en cuestión de horas [Vela, 2017]. Por otra parte, se utilizó una IMU con visión para simplificar algunas trasformaciones que restaban velocidad al procesamiento, demostrando que la IMU y la visión se complementan perfectamente en ciertas tareas como ya se había analizado anteriormente en TecNM-CENIDET, donde se eliminaba la deriva generada de la IMU con ayuda de la visión [Navarrete, 2018].

7.2 Productos

La investigación de este tema dio como resultado varios productos como lo son:

- Código de software para computar, optimizar y graficar el error por métodos directos.
- Código de software para generar mapas en 3D con ayuda de cuaternios.
- Código de software para sistema embebido, capaz de leer y filtrar datos de la IMU y conectarse con una estación donde genera mapas 3D.
- Código de software de método directo para generar mapas por medio de imágenes y datos de IMU previamente capturados.
- Artículo publicado " Análisis del algoritmo de optimización por enjambre de partículas por medio de una aplicación gráfica 3D"
- Documento de tesis

7.3 Aportaciones

Las aportaciones de este trabajo son las siguientes:

- Se aportó con información teórica de álgebra y grupos de Lie.
- Implementación de cuaternios para rotaciones.
- Implementación de algoritmos de optimización.
- Un filtro propuesto para detectar regiones fáciles de seguir.
- Entornos visuales 3D para experimentación.
- Filtros para IMU y controladores para usarlo con GPIO de un sistema embebido.

7.4 Trabajo futuro

Dado el limitado poder de cómputo que se posee actualmente, los métodos directos no han podido aprovecharse del todo, sin embargo, es posible que utilizando GPU pueda mejorar su rendimiento; el algoritmo desarrollado tiene la cámara siempre viendo para el piso lo que hace imposible detectar obstáculos frente del robot, otra mejora sería que la cámara detecte rotaciones junto con la IMU para mejorar la precisión y de manera inversa, la IMU pueda aportar información de movimiento de aceleraciones y obtener una complementación completa con los dos sensores.

Capítulo 8: Referencias

[Adafruit, 2014] Adafruit,"Adafruit 10-DOF IMU Breakout - L3GD20H + LSM303 + BMP180," Engineered in NYC Adafruit, 2014.

[Benítez, 2013] Raúl Benítez, Gerard Escudero & Samir Kanaan,"Inteligencia artificial avanzada," Universitat Oberta de Catalunya, 2013.

[Blanco, 2014] Jose-Luis Blanco,"A tutorial on SE(3) transformation parameterizationsand on-manifold optimization," Reporte técnico por el Grupo de Percepción y Robótica en Universidad de Málaga, 2014.

[Bronshtein, 2015] Bronshtein, I.N., Semendyayev, K.A., Musiol, G., Mühlig, H,"Handbook of Mathematics," Springer, 2015.

[Carlone, 2015] Christian Forster , Luca Carlone, Frank Dellaert y Davide Scaramuzza,"IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation," Robotics† and Perception Group, University of Zurich, Switzerland. Schoolof Interactive Computing, Georgia Institute of Technology, Atlanta, GA, USA., 2015.

[Davison, 2003] Andrew J. Davison,"Real-Time Simultaneous Localisation and Mapping with a SingleCamera," Robotics Research Group, Dept. of Engineering Science, University of Oxford, 2003.

[Eade, 2013] Ethan Eade,"Lie groups for 2D and 3D Transformations," Uber Advanced Technologies Center, 2013.

[Eade, 2014] Ethan Eade,"Lie groups for Computer Vision," Uber Advanced Technologies Center, 2014.

[Engel, 2014] J. Engel, T. Schöps, y D. Cremers,"LSD-SLAM: Large-scale direct monocular SLAM," European Conference on Computer Vision (ECCV), 2014.

[Engel, 2016] J. Engel, V. Koltun y D. Cremers,"Direct Sparse Odometry," In arXiv:1607.02565, 2016.

[Faessler, 2015] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, y D. Scaramuzza,"Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerialvehicle," Journal of Field Robotics, 2015.

[Favaro, 2001] P. Favaro, H. Jin, and S. Soatto,"A semidirect approach to structure from motion," IEEE Intl. Conf. on Image Analysis and Processing, 2001.

[Feltey, 2008] Daniel Feltey,"Matrix Groups," VIGRE en University of Chicago, 2008.

[Forster, 2014] Christian Forster, Matia Pizzoli y Davide Scaramuzza,"SVO: Fast Semi-DirectMonocular Visual Odometry," Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), 2014.

[Forster, 2015] Christian Forster, Matthias Faessler, Flavio Fontana, Manuel Werlberger y Davide Scaramuzza,"Continuous On-Board Monocular-Vision–based Elevation Mapping Applied to Autonomous Landing of Micro Aerial Vehicles," IEEE International Conference on Robotics and Automation (ICRA), 2015.

[Fraundorfer, 2012] Friedrich Fraundorfer and Davide Scaramuzza,"Visual Odometry Part II: Matching, Robustness, Optimization, and Applications," IEEE Robotics & Automation Magazine (RAM), 2012.

[Fuentes, 2013] Luis Alexander Fuentes y Victor Edgardo López Sandoval,"Introducción a las Algebras de Lie," Tesis de licenciatura en Universidad del Salvador, 2013.

[González, 2011] Sergio Alejandro González Segura,"Visión binocular para robot móvil en exteriores simulados," Tesis de maestría en Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), 2011.

[Holt, 2014] Alan Holt y Chi-Yu Huang,"Embedded Operating Systems A Practical Approach," Springer, 2014.

[Irani, 1999] M. Irani & P. Anandan ,"All About Direct Methods," Proc.Workshop Vis. Algorithms: Theory Pract, 1999.

[Jiménez, 2014] Manuel Jiménez, Rogelio Palomera y Isidoro Couvertier,"Introduction to Embedded Systems Using Microcontrollers and the MSP430," Springer, 2014.

[Klein, 2007] G. Klein y D. Murray,"Parallel Tracking and Mapping for Small AR Workspaces," IEEE and ACM International Symposium on Mixed and Augmented Reality(ISMAR), 2007.

[Klette, 2014] Reinhard Klette,"Concise Computer Vision: An Introductioninto Theory and Algorithms," Springer, 2014.

[Lee, 2000] John M. Lee,"Introduction to Smooth Manifolds," University of Washington, 2000.

[Lourakis, 2005] Manolis I. A. Lourakis,"A Brief Description of theLevenberg-Marquardt Algorithm Implemened by levmar," Foundation for Research and Technology - Hellas (FORTH), 2005.

[Luinge, 2002] Hendrik Johannes Luinge ,"INERTIAL SENSING OF HUMAN MOVEMENT," Twente University Press, 2015.

[Madsen, 2004] K. Madsen, H.B. Nielsen & O. Tingleff,"METHODS FOR NON-LINEAR LEAST SQUARES PROBLEMS," Informatics and Mathematical Modelling Technical University of Denmark, 2004.

[Mur-Artal, 2015] Raúl Mur-Artal , J. M. M. Montiel & Juan D. Tardós ,"ORB-SLAM: A Versatile and Accurate Monocular SLAM System," IEEE Transactions on Robotics , 2015.

[Navarrete, 2018] Larisa Navarrete Olmedo,"Sistema de Navegación Inercial Asistido por Visión paraRobots Móviles Terrestres," Tesis de maestría en Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), 2018.

[Newcombe, 2011] R. a. Newcombe, S. J. Lovegrove, y A. J. Davison,"DTAM: Dense Tracking and Mapping in Real-Time," IEEE Int. Conf. on Computer Vision (ICCV), 2011.

[Newcombe, 2015] Richard Newcombe, Dieter Fox y Steve Seitz,"DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[Owen-Hill, 2016] Alex Owen-Hill,"Robot Vision vs Computer Vision: What's the Difference?," https://blog.robotiq.com/robot-vision-vs-computer-vision-whats-the-difference?success=true, 2016.

[Peidong, 2017] Peidong Liu, Lionel Heng, Torsten Sattler , Andreas Geiger y Marc Pollefeys,"Direct Visual Odometry for a Fisheye-Stereo Camera," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.

[Peralta, 2013] Raúl Peralta Lozada,"Sistema de Odometría Visual Monocular para Robots Móviles," Universidad Nacional Autónoma de México, 2013.

[Raluca, 2017] Raluca Scona , Simona Nobili , Yvan R. Petillot y Maurice Fallon,"Direct Visual SLAM Fusing Proprioception for a Humanoid Robot," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.

[Read, 2000] Paul Read & Mark-Paul Meyer," Restoration of Motion Picture Film ," Butterworth-Heinemann, 2000.

[Ruiz, 2014] Carlos Ruiz Jiménez,"Apuntes De Física Fundamental," Reporte técnico, 2014.

[Sagle, 1973] Arthur A . Sagle & Ralph E. Walde,"Introduction To Lie Groups And Lie Algebras," Academic Press, 1973.

[Salas, 2014] Renato F. Salas-Moreno,"Dense Semantic SLAM," Tesis de doctorado en Imperial College London, 2014.

[Scaramuzza, 2009] Davide Scaramuzza, Friedrich Fraundorfer, Marc Pollefeys, Roland Siegwart,"Absolute Scale in Structure from Motion from a Single Vehicle Mounted Cameraby Exploiting Nonholonomic Constraints," Autonomous Systems Lab & Computer Vision and Geometry Group, ETH Zurich, 2009.

[Scaramuzza, 2011] Davide Scaramuzza y Friedrich Fraundorfer,"Visual Odometry Part I: The First 30 Years and Fundamentals," IEEE Robotics & Automation Magazine (RAM), 2011.

[Scaramuzza, 2014] Davide Scaramuzza, Michael C. Achtelik, Lefteris Doitsidis, Friedrich Fraundorfer, Elias Kosmatopoulos,Agostino Martinelli, Markus W. Achtelik, Margarita Chli, Savvas Chatzichristofis, Laurent Kneip, Daniel Gurdan,Lionel Heng, Gim Hee Lee, Simon Lynen, Lorenz Meier, Marc Pollefeys, Alessandro Renzaglia, Roland Siegwart,Jan Carsten Stumpf, Petri Tanskanen, Chiara Troiani y Stephan Weiss,"Vision-ControlledMicro Flying Robots From System Design to Autonomous Navigationand Mapping in GPS-Denied Environments," IEEE ROBOTICS & AUTOMATION MAGAZINE, 2014.

[Schöps, 2014] T. Schöps, J. Engel, y D. Cremers,"Semi-dense visualodometry for AR on a smartphone," In International Sym-posium on Mixed and Augmented Reality (ISMAR), 2014.

[Shum, 1999] Harry Shum, Zhengyou Zhang, Qifa Ke,"Efficient Bundle Adjustment with Virtual Key Frames: A Hierarchical Approach to Multi-Frame Structure from Motion," Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR) , 1999.

[Stumberg, 2018] Lukas von Stumberg, Vladyslav Usenko y Daniel Cremers,"Direct Sparse Visual-Inertial Odometry using Dynamic Marginalization," International Conference on Robotics and Automation, 2018.

[Vela, 2017] Virna Viridiana Vela Rincón,"Experimentación con Odometría Visual porMétodos Directos," Tesis de maestría en Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), 2017.

[Velázquez, 2016] Charles F. Velázquez Dodge & M. Mejía Lavalle,"Análisis del algoritmo de optimización por enjambre de partículas por medio de una aplicación gráfica 3D," COMIA, 2016.

[Vergara, 2015] Andrés Vergara Bahena,"Navegación, Localización y Mapeo de Robots Móviles paraTrayectorias Pre-especificadas por Imágenes," Tesis de maestría en Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), 2015.

Anexo A

En este anexo se presenta con más detalle información de grupos y álgebras de Lie.

A.1 Estructuras algebraicas

La noción de estructura se enfoca en el rol matemático y sus aplicaciones. Por lo que una estructura algebraica, es un conjunto con operaciones definidas. Entonces, cualquiera de las operaciones definidas aplicada a ese conjunto, da como resultado otro elemento del mismo conjunto [Fuentes, 2013][Bronshtein, 2015].

A.2 Definición de Grupo

Un grupo, es una estructura algebraica. Se dice que es un grupo **G,** es aquel conjunto que cumple con lo siguiente:

1) $\boldsymbol{G} \neq \emptyset$ *(Ecuación: A.1)*

2) $\boldsymbol{G}\, x \boldsymbol{G} \rightarrow \boldsymbol{G}$ *(Ecuación: A.2)*

3) $\forall \{a, b, c\} \in \boldsymbol{G} \rightarrow (a \circ b) \circ c = a \circ (b \circ c)$ *(Ecuación: A.3)*

4) $\forall\ a \in \boldsymbol{G} \wedge \exists e \in \boldsymbol{G} \rightarrow a \circ e = e \circ a = a$ *(Ecuación: A.4)*

5) $\forall\ a \in \boldsymbol{G}\ \ \exists\ a^{-1} \in \boldsymbol{G} \rightarrow a \circ a^{-1} = a^{-1} \circ a = e$ *(Ecuación: A.5)*

Donde e es un elemento neutro, identidad o unidad del grupo **G** [Blanco, 2014][Ruiz, 2014] [Fuentes, 2013][Feltey, 2008][Bronshtein, 2015].

Para dejar más claro, por ejemplo, el conjunto de los números naturales con la operación suma.

$$(\mathbb{N},+) \qquad \textit{(Ecuación: A.6)}$$

No es un grupo, debido a que los número naturales no tienen inverso.

$$-3+3=0 \quad -3\notin\mathbb{N} \qquad \textit{(Ecuación: A.7)}$$

Como se puede ver en la ecuación *A.5*, es requisito la existencia de un número inverso dentro del conjunto para ser considerado un grupo.

Por otra parte, el conjunto de los números enteros con la operación suma, sí es un grupo.

$$(\mathbb{Z},+) \qquad \textit{(Ecuación: A.8)}$$

$$-3+3=0 \quad -3\in\mathbb{Z} \qquad \textit{(Ecuación: A.9)}$$

A.3 Definición de Grupo Abeliano

Una estructura algebraica sobre un conjunto que tiene estructura de grupo **G** se le llama Abeliano si es conmutativo [Fuentes, 2013][Ruiz, 2014][Bronshtein, 2015]:

$$\forall\ \{a,b\}\in\boldsymbol{G} \rightarrow a\circ b=b\circ a \qquad \textit{(Ecuación: A.10)}$$

Un ejemplo de grupo Abeliano, es el conjunto de los número racionales con la operación suma:

$$(\mathbb{Q},+) \qquad \textit{(Ecuación: A.11)}$$

$$\frac{1}{2}+\frac{1}{3}=\frac{1}{3}+\frac{1}{2} \qquad \textit{(Ecuación: A.12)}$$

Por otra parte, El grupo simétrico de tres elementos $\mathbf{S}_3$ no es Abeliano con la operación de la permutación. Si ponemos como ejemplo que se tiene los elementos { 1,2,3 } entones:

$$A=(12)(13)=\{3,1,2\} \qquad \textit{(Ecuación: A.13)}$$

$$B=(13)(12)=\{2,3,1\} \qquad \textit{(Ecuación: A.14)}$$

$$B\neq A \qquad \textit{(Ecuación: A.15)}$$

Otro grupo no Abeliano, es el grupo general lineal **GL**(n,$\mathbb{R}$), que es el grupo de matrices cuadradas invertibles.

$$A=\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}=\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix} \qquad \textit{(Ecuación: A.16)}$$

$$B=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix} \quad \textit{(Ecuación: A.17)}$$

$$B\neq A \quad \textit{(Ecuación: A.18)}$$

A.4 Definición de Subgrupo

Un subconjunto **H** de **G** es un subgrupo de **G** si se cumple que:

1) $\boldsymbol{H}\neq\emptyset$ *(Ecuación: A.19)*

2) $\boldsymbol{H}\leqslant\boldsymbol{G}$ *(Ecuación: A.20)*

3) $\forall\ \{a,b\}\in\boldsymbol{H}\rightarrow a\circ b\in\boldsymbol{H}$ *(Ecuación: A.21)*

4) $\forall\ a\in\boldsymbol{H}\rightarrow a^{-1}\in\boldsymbol{H}$ *(Ecuación: A.22)*

Un ejemplo es el grupo aditivo de enteros de módulo 2, ques es un subgrupo del grupo aditivo de enteros de módulo 8. Ambos conjuntos son grupos y son Abelianos [Fuentes, 2013] [Ruiz, 2014][Bronshtein, 2015].

$$\mathbb{Z}_2=\{0,1\}\wedge\mathbb{Z}_8=\{0,1,2,3,4,5,6,7\}\rightarrow\mathbb{Z}_2\subset\mathbb{Z}_8 \quad \textit{(Ecuación: A.23)}$$

A.5 Definición de Anillo

Se dice que un conjunto **R** es un anillo si está definido por la operación suma y producto respectivamente [Fuentes, 2013][Ruiz, 2014][Feltey, 2008][Bronshtein, 2015]:

$$\boldsymbol{R}\neq\emptyset \quad \textit{(Ecuación: A.24)}$$

$$a+b=c\wedge ab=c\ \forall\ \{a,b,c\}\in\boldsymbol{R} \quad \textit{(Ecuación: A.25)}$$

$$a+(b+c)=(a+b)+c\wedge a(bc)=(ab)c\ \forall\ \{a,b,c\}\in\boldsymbol{R} \quad \textit{(Ecuación: A.26)}$$

$$\forall\ a\in\boldsymbol{R}\wedge\exists e\in\boldsymbol{R}\rightarrow a+e=e+a=a \quad \textit{(Ecuación: A.27)}$$

$$\forall\ a\in\boldsymbol{R}\wedge\exists e\in\boldsymbol{R}\rightarrow ae=ea=a \quad \textit{(Ecuación: A.28)}$$

A.6 Definición de Campo

Un campo o cuerpo tiene una estructura de anillo, es Abeliano y además se le añade la propiedad del elemento inverso y ser divisible con los elementos del campo, excepto por el elemento neutro o cero[Fuentes, 2013][Ruiz, 2014][Feltey, 2008][Bronshtein, 2015]:

$$\forall\ a \in \boldsymbol{F}\ \ \exists\ a^{-1} \in \boldsymbol{F} \rightarrow a + a^{-1} = a^{-1} + a = e \qquad \textit{(Ecuación: A.29)}$$

$$\forall\ a \in \boldsymbol{F}\ \ \exists\ a^{-1} \in \boldsymbol{F} \rightarrow a\, a^{-1} = a^{-1}\, a = e \qquad \textit{(Ecuación: A.30)}$$

El elemento neutro del campo con la operación suma es cero, mientras que con la operación del producto es uno. Ejemplo de campo, son los numero reales, racionales y complejos.

A.7 Definición de espacio vectorial

Es llamado espacio vectorial o espacio lineal **V** sobre un campo **F** de escalares, a la estructura algebraica que cumple con ciertas propiedades, donde:

$$\boldsymbol{V} \neq \emptyset \qquad \textit{(Ecuación: A.31)}$$

$$\forall\ \{a, b\} \in \boldsymbol{V}\ \ \exists\ c = a + b \in \boldsymbol{V} \qquad \textit{(Ecuación: A.32)}$$

$$\forall\ \alpha \in \boldsymbol{F}\ \ \exists\ a \in \boldsymbol{V}\ \wedge\ \alpha\, a \in \boldsymbol{V} \qquad \textit{(Ecuación: A.33)}$$

Los elementos de **V** son elementos arbitrarios y los elementos de **F** son escalares.

$$\forall\ \{\alpha, \beta\} \in \boldsymbol{F} \qquad \textit{(Ecuación: A.34)}$$

$$\forall\ \{a, b, c\} \in \boldsymbol{V} \qquad \textit{(Ecuación: A.35)}$$

Con la operación suma:

$$\boldsymbol{V}\, x\, \boldsymbol{V} \rightarrow \boldsymbol{V} \qquad \textit{(Ecuación: A.36)}$$

$$(a, b) \rightarrow a + b \qquad \textit{(Ecuación: A.37)}$$

Las propiedades con la suma son:

$$\forall \{a, b, c\} \in \boldsymbol{V} \rightarrow a + b = b + a \qquad \textit{(Ecuación: A.38)}$$

$$\forall \{a, b, c\} \in \boldsymbol{V} \rightarrow (a + b) + c = a + (b + c) \qquad \textit{(Ecuación: A.39)}$$

$$\forall\ a \in \boldsymbol{V} \wedge\ \exists\ e \in \boldsymbol{V} \rightarrow a + e = e + a = a \qquad \textit{(Ecuación: A.40)}$$

$$\forall\ a\in \boldsymbol{V}\ \ \exists\ a^{-1}\in \boldsymbol{V} \rightarrow a+a^{-1}=a^{-1}+a=e \qquad \textit{(Ecuación: A.41)}$$

El elemento e es neutro y en la operación suma es igual a cero.

Recordando que alfa y beta son escalares, la operación de producto sobre un escalar son:

$$\boldsymbol{F}\ x\ \boldsymbol{V}\rightarrow \boldsymbol{V} \qquad \textit{(Ecuación: A.42)}$$

$$(a,\alpha)\rightarrow a\cdot\alpha \qquad \textit{(Ecuación: A.43)}$$

Tomando en cuanta que el elemento e es neutro y es igual a uno, las propiedades con el producto son:

$$\forall\ a\in \boldsymbol{V}\ \ \exists\ e\in \boldsymbol{V} \rightarrow a\cdot e=e\cdot a=a \qquad \textit{(Ecuación: A.44)}$$

$$\forall\ a\in \boldsymbol{V}\wedge\{\alpha,\beta\}\in \boldsymbol{F}\ \ \rightarrow \alpha(\beta\, a)=(\alpha\beta)a \qquad \textit{(Ecuación: A.45)}$$

$$\forall\ a\in \boldsymbol{V}\wedge\{\alpha,\beta\}\in \boldsymbol{F}\ \ \rightarrow (\alpha+\beta)a=\alpha\, a+\beta\, a \qquad \textit{(Ecuación: A.46)}$$

$$\forall\ \{a,b\}\in \boldsymbol{V}\wedge\alpha\in \boldsymbol{F}\ \ \rightarrow \alpha(ab)=\alpha\, a+\alpha\, b \qquad \textit{(Ecuación: A.47)}$$

Todos los elementos de un espacio vectorial, son llamados vectores y a los elementos de los campos o cuerpos, son llamados escalares [Fuentes, 2013][Bronshtein, 2015].

A.8 Definición de Generadores

Para los conjuntos finitos se puede definir un subconjunto llamado sistema generador, que consiste en un conjunto de elementos de un grupo que por medio de su producto o producto inverso, genera al grupo. Para generar el espacio vectorial $\mathfrak{so}(3,\mathbb{R})$ del grupo especial ortogonal de tres dimensiones **SO**(3,ℝ), tenemos a los generadores [Bronshtein, 2015][Eade, 2013][Ruiz, 2014]:

$$G_1=\begin{bmatrix}0&0&0\\0&0&-1\\0&1&0\end{bmatrix}\quad G_2=\begin{bmatrix}0&0&1\\0&0&0\\-1&0&0\end{bmatrix}\quad G_3=\begin{bmatrix}0&-1&0\\1&0&0\\0&0&0\end{bmatrix} \qquad \textit{(Ecuación: A.48)}$$

A.9 Definición de Variedad

Una variedad $\mathfrak{M}$ de n dimensiones, es un espacio topológico donde cada punto **p** pertenece a la variedad y localmente, se puede ubicar en un espacio euclidiano. Es decir, el vecindario de cada punto **p** es un conjunto abierto homeomorfo a $\mathbb{R}^n$ [Blanco, 2014][Lee, 2000][Sagle, 1973].

A.10 Variedad Suave Diferenciable

Cuando definimos que una variedad es un espacio topológico, tenemos que una variedad pude tener líneas curvas, superficies, círculos, parábolas, etc. en n dimensiones. Extendiendo esto al cálculo, podemos obtener la derivada entre dos puntos de la variedad, cuando la derivada es diferente de cero en todas las direcciones del punto, se dice que es diferenciable. Por otro lado, tiene una estructura suave si es infinitamente diferenciable en todo el dominio y en cualquier orden, incluyendo a los números complejos. Es decir la función pertenece a la clase $\mathbf{C^{\infty}}$. Alguno ejemplos de variedades suaves, son las esferas y los toroides de *n-dimensiones*.[Salas, 2014][Blanco, 2014][Sagle, 1973][Lee, 2000]

A.11 Definición de Espacio tangente de una variedad

El espacio tangente de una variedad es un espacio vectorial de cada punto p de la variedad. Este espacio vectorial está compuesto de puntos no singulares, forma un hiperplano de la misma dimensión de la variedad y se obtiene a partir de la diferenciación del punto **p** [Blanco, 2014][Sagle, 1973].

A.12 Grupos de Lie y Álgebras de Lie

Los grupos y álgebra de Lie son herramientas matemáticas, que en este caso son necesarias para poder modelar la pose del robot móvil, componer transformaciones tanto en las imágenes pixeladas como en el software que graficar la información. En este trabajo, se presentarán transformaciones en espacios no mayores de tres dimensiones, sin embargo, es posible utilizar estas herramientas en cualquier cantidad de dimensiones. También son útiles para reducir el número de parámetros al momento de realizar transformaciones y por ende, es posible reducir el costo computacional. Cuando se optimiza una transformación o conjunto de trasformaciones que conforman una trayectoria, los grupos y álgebras de Lie también juegan un papel importante [Salas, 2014][Eade, 2014][Peralta, 2013][Eade, 2013].

A.13 Grupo de Lie

Dentro de los grupos continuos hay un conjunto denominado grupos de Lie, en honor del matemático noruego Sophus Lie que los estudió a finales del siglo XIX.

Una de las ideas subyacentes en la noción de grupo de Lie es la de movimiento. Los grupos de Lie por excelencia son los que representan movimientos en espacios euclidianos, cada

movimiento se puede componer con otro movimiento y forma un nuevo movimiento que también pertenece al grupo y además depen “diferenciablemente” de los movimientos anteriores que lo formaron.

Una vez comprendidos los conceptos anteriores, la definición de un grupo de Lie es muy simple. Definiremos a los grupos de Lie como una variedad diferenciable con estructura de grupo. Además de ser un subconjunto no vacío que pertenece a un espacio $\mathbb{R}^n$.

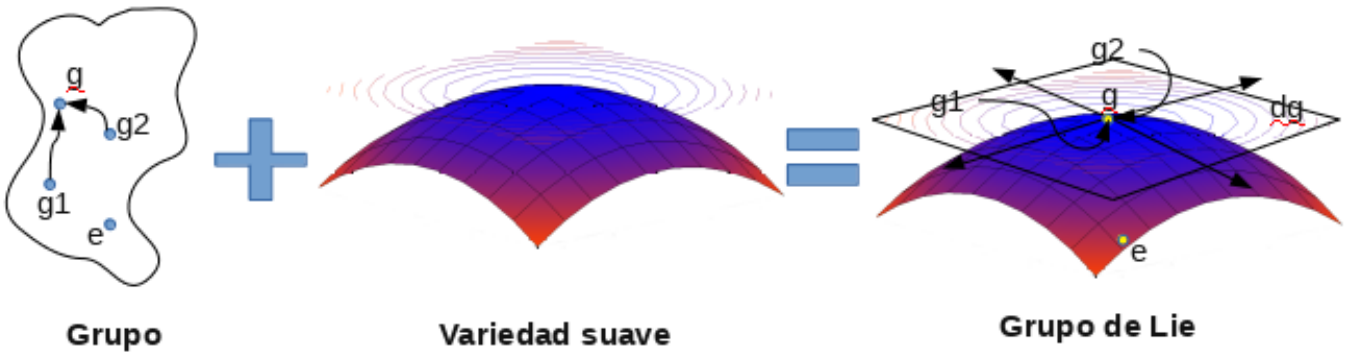


Figura A.1: Grupo de Lie

A continuación se muestra una breve lista de grupos de Lie importantes [Blanco, 2014][Ruiz, 2014][Sagle, 1973][Feltey, 2008]:

Tabla A.1: Grupos de Lie importantes

Grupo	Descripción	Nombre del grupo
$\boldsymbol{GL}(n,\mathbb{C})$	*no singular n x n matrices complejas*	General Lineal complejo
$\boldsymbol{GL}(n,\mathbb{R})$	*no singular n x n matrices reales*	General Lineal real
$\boldsymbol{SL}(n,\mathbb{C})$	$X \in \boldsymbol{GL}(n,\mathbb{C}) : det(X) = 1$	Especial Lineal complejo
$\boldsymbol{SL}(n,\mathbb{R})$	$X \in \boldsymbol{GL}(n,\mathbb{R}) : det(X) = 1$	Especial Lineal real
$\boldsymbol{O}(n,\mathbb{R})$	$X \in \boldsymbol{GL}(n,\mathbb{R}) : X^t = X^{-1}$	Ortogonal real
$\boldsymbol{SO}(n,\mathbb{R})$	$X \in \boldsymbol{O}(n,\mathbb{R}) \cap \boldsymbol{SL}(n,\mathbb{R})$	Especial Ortogonal real
$\boldsymbol{O}(p,q)$	$X \in \boldsymbol{GL}(p+q,\mathbb{R}) : I_{p\cdot q} X^t I_{p\cdot q}^{-1} = X^{-1}$	Ortogonal indefinido
$\boldsymbol{SO}(p,q)$	$\boldsymbol{O}(p,q) \cap \boldsymbol{SL}(p+q,\mathbb{R})$	Especial Ortogonal indefinido
$\boldsymbol{Sp}(n,\mathbb{C})$	$X \in \boldsymbol{GL}(2n,\mathbb{C}) : J_n X^t J_n^{-1} = X^{-1}$	Simplécticoreal complejo
$\boldsymbol{Sp}(n,\mathbb{R})$	$\boldsymbol{SL}(n,\mathbb{C}) \cap \boldsymbol{GL}(n,\mathbb{R})$	Simplécticoreal real
$\boldsymbol{U}(n)$	$X \in \boldsymbol{GL}(n,\mathbb{C}) : \overline{X^t} = X^{-1}$	Unitario
$\boldsymbol{SU}(n)$	$\boldsymbol{U}(n) \cap \boldsymbol{SL}(n,\mathbb{C})$	Especial Unitario

A.13.1 Álgebra de Lie

Al espacio tangente de un Grupo de Lie se le llama Álgebra de Lie. En toda Álgebra de Lie $\mathfrak{g}$, existe la operación bilineal llamada forma de Lie o corchetes de Lie:

$$[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \rightarrow \mathfrak{g} \qquad \textit{(Ecuación: A.49)}$$

Al utilizar los corchetes de Lie en algún elemento del álgebra, debe cumplir con la identidad de Jacobi y la anti-conmutativoidad, como se muestra a continuación [Blanco, 2014] [Fuentes, 2013][Sagle, 1973]:

$$[A_x + B_y, z] = A[x,z] + B[y,z] \ , \ [z, A_x + B_y] = A[z,x] + B[z,y] \qquad \textit{(Ecuación: A.50)}$$

$$[x,x] = 0 \qquad \textit{(Ecuación: A.51)}$$

$$[x,y]=-[y,x] \qquad \textit{(Ecuación: A.52)}$$

$$[x,[y,z]]+[z,[x,y]]+[y,[z,x]]=0 \qquad \textit{(Ecuación: A.53)}$$

A.13.2 Mapa exponencial y logarítmico

Para todo grupo de Lie **G** hay una álgebra $\mathfrak{g}$ asociada, a las cuales se le pueden aplicar dos funciones. La función exponencial mapea un elemento del álgebra de Lie a la variedad del grupo de Lie, mientras que la función logarítmica hace lo contrario, mapea un elemento de la variedad del grupo de Lie al espacio vectorial del álgebra de Lie [Blanco, 2014][Eade, 2014] [Eade, 2013].

$$\exp:\mathfrak{g}\rightarrow \boldsymbol{G} \qquad \textit{(Ecuación: A.54)}$$

$$\ln:\boldsymbol{G}\rightarrow \mathfrak{g} \qquad \textit{(Ecuación: A.55)}$$

Anexo B

Se puede encontrar el código fuente de el software desarrollado en:

- **Software para el robot:** *https://github.com/CharlesVD/software_embebido_robot_MD*
- ***Software para la estación:*** *https://github.com/CharlesVD/software_estacion*
- ***Software para pruebas offline:*** *https://github.com/CharlesVD/software_offline_MD*
- ***Software de optimización:*** *https://github.com/CharlesVD/software_optimizacion_MD*
- ***Software para mapas 2D:*** *https://github.com/CharlesVD/software_mapa2D_MD*