



EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación
y Desarrollo Tecnológico

Tesis de Maestría

Visión robótica de baja resolución,
con recursos limitados

presentada por

Ing. Fernando Abisai Luna Santander

como requisito para la obtención del grado de
Maestro en Ciencias de la Computación

Director de tesis

Dr. Raúl Pinto Elías

Codirector de tesis

Dr. José Ruiz Ascencio

Cuernavaca, Morelos, México. Marzo de 2023.



Cuernavaca, Mor., **17/febrero/2023**

OFICIO No. DCC/044/2023

Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFCIO

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes del Comité Tutorial de FERNANDO ABISAI LUNA SANTANDER, con número de control M21CE016, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **“VISIÓN ROBÓTICA DE BAJA RESOLUCIÓN, CON RECURSOS LIMITADOS”** y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

RAÚL PINTO ELÍAS
Director de tesis

JOSÉ RUIZ ASCENCIO
Codirector de tesis

ANDREA MAGADÁN SALAZAR
Revisor 1

NIMROD GONZÁLEZ FRANCO
Revisor 2

C.e.p. Depto. Servicios Escolares
Expediente / Estudiante

MYHP/ibm





Cuernavaca, Mor., **23/febrero/2023**
No. De Oficio: **SAC/048/2023**
Asunto: **Autorización de impresión de tesis**

FERNANDO ABISAI LUNA SANTANDER
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 **“VISIÓN ROBÓTICA DE BAJA RESOLUCIÓN, CON RECURSOS LIMITADOS”**, 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.

ATENTAMENTE

Excelencia en Educación Tecnológica®
“Conocimiento y tecnología al servicio de México”

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/RMA



Dedicatoria

A Dios principalmente por brindarme la vida, la salud y la oportunidad de realizar mis estudios de maestría para poder llevar a cabo este proyecto de investigación.

A mi familia por su amor, consejo, motivación y apoyo incondicional. Los quiero mucho.

A la memoria de mis abuelitos Clara y Fernando.

Agradecimientos

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

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

A mi director de tesis el Dr. Raúl Pinto Elías y a mi codirector el Dr. José Ruiz Asencio, por su asesoramiento durante el desarrollo de esta tesis, por brindarme sus consejos, su apoyo y su paciencia.

A mis revisores la Dra. Andrea Magadán Salazar y el Dr. Nimrod González Franco, por su crítica y comentarios que fueron fundamentales para la realización de esta tesis.

Al M.C. Jose Omar de Jesus Trujillo Quintero, compañero de generación del área de Inteligencia Artificial por su asesoramiento en la construcción y pruebas de funcionamiento del robot utilizado en este proyecto, pero sobre todo por su gran amistad y apoyo.

A mis amigos y compañeros del Departamento de Ciencias Computacionales (Merino, Karen, Andros, Mario, Alan, Diego, Fer, Becca, Ariel, Kevin, Valentín, Eydi, Vere, Areli y Luis) con los que he compartido grandes momentos, que, con su apoyo, sus consejos y su grata amistad hicieron que esta experiencia fuese inolvidable.

A mi familia por su amor, consejo, motivación y apoyo incondicional a lo largo de mi vida y durante el desarrollo de este proyecto.

Resumen

La visión robótica tiene objetivos cuyo denominador común es darles mayor autonomía a los robots. Los robots móviles con visión generan grandes cantidades de datos durante sus desplazamientos. Estas masas de datos representan un problema para su procesamiento en tiempo real en computadoras con recursos limitados.

Dada la problemática en este trabajo se desarrolla un sistema de navegación robótica utilizando la información de flujo óptico obtenido de fragmentos (ventanas) de imágenes de baja resolución, con el fin de montarlo sobre un robot controlado por una computadora con recursos de procesamiento limitados, como la *Rapsberry Pi*, que permita que el robot pueda estimar su posición y evadir obstáculos en *tiempo real*.

Para el cálculo del flujo óptico se implementaron los algoritmos de Horn-Schunk y Lucas-Kanade, posteriormente los datos fueron ingresados a algoritmos de clasificación (*CART*, *Adaboost* y *RandomForest*) para la estimación del estado del robot, mientras que para la evasión de obstáculos se presenta una estrategia en base a las magnitudes de flujo óptico en las ventanas de la imagen.

Como resultado, se obtuvo que los datos de flujo óptico calculados con el algoritmo de Lucas-Kanade, dieron mejores resultados con el algoritmo de clasificación *RandomForest* para la estimación del estado del robot y, en la técnica de evasión de obstáculos los datos del algoritmo de Lucas-Kanade logró una eficacia entre el 75% y el 100% evadiendo los obstáculos presentes durante las trayectorias (promedio de 88.47%). Asimismo, los tiempos de procesamiento permiten realizar el proceso de navegación de 5 a 7 pares de fotogramas por segundo, logrando realizar el proceso en *tiempo real*.

Palabras clave: navegación robótica, flujo óptico, baja resolución, tiempo real.

Abstract

Robotic vision has goals whose common denominator is to give robots greater autonomy. Mobile robots with vision generate large amounts of data during their movements. These masses of data represent a problem for real-time processing on computers with limited resources.

Given these problems, this work develops a robotic navigation system using optical flow information obtained from fragments (windows) of low-resolution images, controlled by a computer with limited processing resources such as the Raspberry Pi that allows the robot to estimate its position and avoid obstacles in real time.

For the calculation of the optical flow, the Horn-Schunck and Lucas-Kanade algorithms were implemented, then the data were entered into classification algorithms (CART, Adaboost and RandomForest) to estimate the state of the robot and for obstacle avoidance, a strategy based on the magnitudes of optical flow in the image windows was presented.

As a result, it was obtained that the optical flow data calculated with the Lucas-Kanade algorithm gave better results with the RandomForest classification algorithm for the estimation of the robot state and in the obstacle avoidance technique the data of the Lucas-Kanade algorithm achieved an effectiveness between 75% and 100% avoiding the obstacles present during the trajectories (average 88.47%). Also, the processing times allow to perform the navigation process from 5 to 7 pairs of frames per second, achieving the process in real time.

Keywords: robotic navigation, optical flow, low resolution, real time.

Índice

Resumen	i
Abstract	ii
Índice de figuras	v
Índice de tablas.....	vii
Índice de algoritmos.....	viii
Lista de acrónimos.....	ix
Capítulo 1 Introducción	1
1.1 Contexto teórico/práctico	1
1.1.1 Marco conceptual.....	1
1.1.2 Antecedentes del proyecto.....	12
1.1.3 Estado del arte.....	13
1.2 Descripción del problema	22
1.2.1 Delimitación del problema específico	22
1.2.2 Descripción de la complejidad del problema	23
1.3 Descripción de la solución	23
1.4 Discusión.....	23
Capítulo 2 Propuesta de solución	25
2.1 Planteamiento de la solución.....	25
2.1.1 Objetivo	25
2.1.2 Alcances y limitaciones	25
2.2 Modelo conceptual de la solución.....	26
2.2.1 Adquisición de las imágenes.....	26
2.2.2 Preprocesamiento	27
2.2.3 Conversión de la imagen del sistema RGB a escala a grises.....	27
2.2.4 Estimación de las derivadas parciales	28
2.2.5 Cálculo del flujo óptico	28
2.2.6 Procesamiento de los valores de flujo óptico obtenidos	29
2.2.7 Toma de decisiones.....	32
2.3 Criterios de control	32
2.3.1 Métrica para la evaluación de los algoritmos de flujo óptico.....	33
2.3.2 Métricas para clasificación (Barrios Arce, 2019).....	33
2.3.3 Métrica para evaluación para la técnica de evasión de obstáculos	35
2.4 Conclusiones.....	35
Capítulo 3 Análisis, diseño e implementación del sistema	36
3.1 Entorno de desarrollo	36

3.1.1 Librerías utilizadas	36
3.2 Diseño del sistema de navegación robótica.....	37
3.2.1 Parámetros iniciales.....	37
3.2.2 Descripción de los procesos y rutinas del sistema	40
3.2.3 Estados de control	45
3.3 Interfaz del sistema	45
3.4 Conclusiones.....	47
Capítulo 4 Experimentación y análisis de los resultados	48
4.1 Recursos utilizados	48
4.1.1 Robot rodante	48
4.1.2 <i>Datasets</i>	49
4.2 Experimentación.....	51
4.2.1 Fase 1. Pruebas de funcionamiento (robot, algoritmos de flujo óptico)	51
4.2.2 Fase 2. Pruebas de algoritmos de flujo óptico para la detección del estado del robot	53
4.2.3 Fase 3. Pruebas de funcionamiento del sistema navegación completo (detección de estado y evasión de obstáculos).....	57
4.2.4 Tiempos de procesamiento y rendimiento del sistema de navegación	65
4.3 Análisis de los resultados obtenidos.....	66
Capítulo 5 Conclusiones y trabajo futuro.....	69
5.1 Conclusiones generales	69
5.2 Objetivos alcanzados	70
5.3 Aportaciones	71
5.4 Trabajos futuros	71
5.5 Actividades académicas adicionales	71
Referencias	73
Anexo A. Algoritmos de clasificación	77
Anexo B. Construcción del robot rodante.....	79
Anexo C. Constancias de participaciones	83

Índice de figuras

Figura 1.1 Detección del flujo óptico en el plano de la imagen. El ejemplo muestra tres cuadros, que muestran el movimiento de la silueta de una cabeza. El flujo óptico se representa como la correspondencia de píxeles de contorno entre el cuadro 1 y 2, así como el cuadro 2 y 3 (Raudies, 2013).	5
Figura 1.2 Máscaras de convolución utilizadas para las derivaciones parciales en x , y y t	8
Figura 1.3 Las tres derivadas parciales de las intensidades de las imágenes k y $k+1$, con base en la ubicación del pixel j , i se estiman cada una de ellas. Aquí el índice de columna j corresponde a la dirección x de la imagen, el índice de fila i a la dirección y , mientras que k indica la dirección del tiempo (Imagen k e imagen $k+1$).	8
Figura 1.4 El laplaciano se estima restando el valor de un punto de una media ponderada de los valores de los puntos vecinos. Aquí se muestran los pesos adecuados por los que se pueden multiplicar los valores (B. K.P. Horn & Schunck, 1981).....	9
Figura 1.5. Comportamiento del flujo óptico en un robot aéreo (Green <i>et al.</i> , 2004).....	14
Figura 2.1 Serie de pasos de la propuesta de solución	26
Figura 2.2 Cambio de posición de un pixel en diferentes imágenes separadas por un tiempo dt (Barron & Thacker, 2005).....	27
Figura 2.3 Tamaño y distribución de las ventanas sobre una imagen de 80×64 píxeles..	27
Figura 2.4 Comportamiento esperado en el eje horizontal (azul) y de la componente vertical (verde) de flujo óptico ante los movimientos de desplazamiento y giro. Cabe señalar que las representaciones de los valores fueron ampliadas con fines de demostración.	30
Figura 2.5 Vector de características con los promedios de los valores de flujo óptico de las cuatro ventanas para las pruebas con los clasificadores.	31
Figura 2.6 Comportamiento de las magnitudes de flujo óptico ante la presencia de obstáculos a diferentes distancias (Back <i>et al.</i> , 2020).....	31
Figura 2.7 Matriz de confusión binaria.....	33
Figura 3.1 Procesos, rutinas y condiciones de control del sistema de navegación robótica propuesto.....	39
Figura 3.2 Proceso de captura de los fotogramas, recorte de ventanas y transformación escala de grises.....	40
Figura 3.3. Procesamiento de la imagen para la rutina de seguimiento de luz guía: a) Imagen obtenida de la cámara para el sistema de navegación, b) Imagen binarizada con los umbrales de HSV, c) Imagen dividida horizontalmente en proporción 62% y 38%.	44
Figura 3.4 Elementos de la interfaz del sistema de navegación	46
Figura 3.5 Estructura de la interfaz del sistema de navegación.....	46
Figura 4.1 Conexiones entre los componentes del robot rodante.....	49
Figura 4.2 Robot rodante utilizado para la experimentación.....	49
Figura 4.3 Área de experimentación	59

Figura 4. 4 Ruta para el experimento 9	61
Figura 4.5 Funcionamiento del robot para la detección del estado del robot: a) sin movimiento, b) atrás, c) derecha, d) izquierda y la detección de la presencia de obstáculos e) libre f) obstáculo a la derecha y g) obstáculo a la izquierda.	61
Figura 4.6 Ruta del experimento 10	62
Figura 4.7 Ruta del experimento 11	64
Figura 4.8 Distribución del tiempo de procesamiento	65
Figura 4.9 Uso del CPU y de la memoria RAM durante las experimentaciones	65
Figura B.1 Ubicación de los componentes del robot rodante	80
Figura B.2 Interfaz de la aplicación Arduino Bluetooth Robot Car	81
Figura B.3 Comportamiento de los motorreductores del robot en las diferentes instrucciones.....	82
Figura C.1 Constancia de presentación de artículo en la 8ª Jornada de Ciencia y Tecnología Aplicada.....	83
Figura C.2 Constancia de presentación de poster en la Escuela de Inteligencia Computacional y Robótica 2022	84
Figura C.3 Constancia de presentación de poster en la 9ª Jornada de Ciencia y Tecnología Aplicada.....	85

Índice de tablas

Tabla 1.1 Resumen de los trabajos del estado del arte	21
Tabla 4.1 Datasets para experimento de la fase 1.....	50
Tabla 4.2 Datasets para los experimentos de la fase 2 y 3.....	50
Tabla 4.3 Resultados del experimento 2.....	52
Tabla 4.4 Tiempos de procesamiento del experimento 3.....	53
Tabla 4.5 Resultados de clasificación con datos del algoritmo de Lucas-Kanade.....	54
Tabla 4.6 Resultados de clasificación con datos del algoritmo de Horn-Schunck.....	54
Tabla 4.7 Resultados de clasificación del experimento 4.....	55
Tabla 4.8 Resultados de clasificación del experimento 5.....	57
Tabla 4.9 Tiempos de procesamiento del experimento 8.....	60
Tabla 4.10 Tiempos de procesamiento del experimento 9.....	62
Tabla 4.11 Tiempos de procesamiento del experimento 10.....	63
Tabla 4.12 Tiempos de procesamiento del experimento 11.....	64
Tabla 4.13 Resultados de los experimentos de la fase 1.....	66
Tabla 4.14 Resultados de los experimentos de la fase 2.....	66
Tabla 4.15 Resultados de los experimentos de la fase 3.....	67
Tabla 5. 1 Comentarios de los objetivos específicos	70
Tabla 5. 2 Resumen del sistema de navegación robótica propuesta en base a las características resaltadas del estado del arte	70
Tabla B.1 Interpretación de las señales de comunicación de entrada y salida de la Raspberry Pi al ESP32	81

Índice de algoritmos

Algoritmo 1.1. Cálculo del flujo óptico mediante el método de Horn-Schunck	9
Algoritmo 1.2 Cálculo del flujo óptico mediante el método de Lucas-Kanade	11
Algoritmo 3.1 Rutina de estado de error	42
Algoritmo 3.2 Rutina de detección de obstáculos	43
Algoritmo 3.3 Rutina de seguimiento de luz guía	44

Lista de acrónimos

AUV: Vehículos Submarinos Autónomos, por sus siglas en inglés.

AVG: Vehículos Terrestres Autónomos, por sus siglas en inglés.

CART: Árboles de Clasificación y Regresión, por sus siglas en inglés.

FPS: Fotogramas por segundo.

HS: Algoritmo de flujo óptico de Horn-Schunck .

IMU: Unidad de Medición Inercial, por sus siglas en inglés.

LK: Algoritmo de flujo óptico de Lucas-Kanade.

MAV: Micro Vehículo Aéreo, por sus siglas en inglés.

HSV: Modelo de color (del inglés *Hue, Saturation, Value* – Matiz, Saturación, Valor).

RGB: Modelo de color (del inglés *Red, Green, Blue* – Rojo, Verde, Azul).

UAV: Vehículos Aéreos no Tripulados, por sus siglas en inglés.

Capítulo 1

Introducción

La visión robótica tiene objetivos cuyo denominador común es darles mayor autonomía a los robots. Los robots móviles con visión generan grandes cantidades de datos durante sus desplazamientos. Estas masas de datos representan un problema para su procesamiento en tiempo real en computadoras con recursos limitados. Dentro de los algoritmos de navegación se encuentran los algoritmos basados en características y los métodos directos y dentro de los métodos directos se encuentran los algoritmos de flujo óptico de los cuales algunos de estos han sido inspirados biológicamente en la visión de los insectos (Serres & Ruffier, 2017)

Los insectos voladores se están estudiando en estos días como si fueran micro vehículos aéreos equipados con sensores inteligentes, que requieren muy pocos recursos cerebrales. Los hallazgos obtenidos en estos volantes naturales han demostrado ser extremadamente valiosos cuando se trata de diseñar sensores ópticos artificiales compactos de bajo peso capaces de realizar tareas de procesamiento visual de manera robusta en diversas condiciones ambientales (Serres & Viollet, 2018).

En este documento se presenta el desarrollo de un sistema de navegación robótica capaz de reconocer el estado de su entorno y evadir obstáculos en *tiempo real*. Para este sistema se implementaron los algoritmos de flujo óptico de Horn- Schunck (1981) y Lucas-Kanade (1981) usando imágenes de baja resolución. Para este trabajo se define la baja resolución como una resolución que contenga entre $n \times 10^2$ (ciento de pixeles) a $n \times 10^3$ (miles de pixeles).

1.1 Contexto teórico/práctico

1.1.1 Marco conceptual

En esta sección se describen conceptos relacionados con el desarrollo de esta tesis, con la finalidad de lograr una mejor comprensión. Se describirán conceptos como sistemas de navegación robótica, métodos directos, flujo óptico, evasión de obstáculos sin entrenamiento previo y sistemas en tiempo real.

1.1.1.1 Sistemas de navegación robótica

La navegación puede definirse como un proceso para determinar un camino adecuado y seguro entre el punto de partida y el punto de destino por el que se desplaza un robot (Bonin-Font *et al.*, 2008). Asimismo, la navegación en un robot móvil puede definirse como la capacidad del robot de desplazarse en un entorno determinado, o la ciencia de guiar a un robot móvil que puede moverse en un entorno. En cuanto a los problemas que acompañan a este proceso, la navegación robótica puede definirse en tres preguntas: "¿Dónde estoy?", "¿A dónde voy?" y "¿Cómo llego?". Las primeras dos preguntas pueden responderse complementando al robot con sensores adecuados, mientras que la tercera pregunta puede resolverse con un sistema eficaz de planificación de la navegación. El sistema de navegación en sí está directamente relacionado con la presencia de sensores utilizados en los robots y la estructura del entorno, y eso significa que siempre coincidirán el propósito de construcción del robot y el entorno en el que el robot será operado (Rahmani *et al.*, 2015).

La navegación basada en la visión ha progresado enormemente con la implementación de diversos vehículos autónomos, ya sea en tierra (Vehículos Terrestres Autónomos, AGV por sus siglas en inglés), en el agua (Vehículos Submarinos Autónomos, AUV por sus siglas en inglés), y en el aire (Vehículos Aéreos no Tripulados, UAV por sus siglas en inglés) (Rahmani *et al.*, 2015).

Independientemente del tipo de vehículo o robot, los sistemas que utilizan la visión para la navegación pueden dividirse a grandes rasgos en los que necesitan un conocimiento previo de todo el entorno y los que perciben el entorno mientras navegan por él (Bonin-Font *et al.*, 2008).

Los **sistemas de navegación que utilizan mapas métricos** (*Metric map-using navigation systems*) necesitan disponer de un mapa completo del entorno antes de iniciar la tarea de navegación. Los sistemas de navegación que construyen mapas métricos (*Metric map-building navigation systems build*) construyen el mapa completo por sí mismos y lo utilizan en la fase de navegación posterior. Otros sistemas incluidos en esta categoría son capaces de auto localizarse en el entorno simultáneamente durante la construcción del mapa. Existen otros tipos de sistemas de navegación basados en la construcción de mapas, como por ejemplo los sistemas visuales basados en sonares (*visual sonar-based systems*) o los sistemas basados en mapas locales (*local map-based systems*). Estos sistemas recogen datos del entorno mientras navegan y construyen un mapa local que se utiliza como soporte para la navegación segura en línea. Este mapa local incluye datos específicos sobre obstáculos y espacio libre de una porción reducida del entorno, que suele estar en función del campo de visión de la cámara (Bonin-Font *et al.*, 2008).

Los **sistemas de navegación sin mapa** (*Mapless navigation systems*) incluyen sobre todo técnicas reactivas que utilizan pistas visuales derivadas de la segmentación de una imagen, el flujo óptico o el seguimiento de características entre fotogramas. En estos sistemas no existe una representación global del entorno; éste se percibe a medida que el sistema navega, reconoce objetos o rastrea puntos de referencia (Bonin-Font *et al.*, 2008).

1.1.1.2 Evasión de obstáculos

La evasión de obstáculos es el problema central de los robots móviles. Su objetivo es permitir a los robots móviles explorar un entorno desconocido sin chocar con otros objetos. Es la base de diversas tareas, como vigilancia, rescate, etc. (Tai *et al.*, 2016).

El principio básico de la evasión de obstáculos consiste en detectar obstáculos y calcular las distancias entre el robot y los obstáculos. Cuando un obstáculo se acerca, el robot debe evitarlo o dar la vuelta siguiendo las instrucciones del módulo de evasión de obstáculos (Lu *et al.*, 2018). En general, las técnicas de detección de obstáculos pueden dividirse en tres grupos: basadas en sensores, basadas en imágenes e híbridas.

Las técnicas basadas en sensores consisten en medir la distancia mediante sensores como radares, ultrasónicos, infrarrojos, etc. Sin embargo, estos sensores son incapaces de obtener suficiente información en entornos complejos debido a su limitado campo de visión y rango de medición (Lu *et al.*, 2018).

Una alternativa a los sensores son las cámaras que proporcionan información visual sobre el entorno por el que se desplaza el robot. Las cámaras son sensores pasivos y las utilizan numerosos algoritmos basados en imágenes para detectar obstáculos utilizando valores de escala de grises y características de puntos o bordes. Pueden proporcionar detalles sobre la cantidad de movimiento o desplazamiento del robot y el color, la forma y el tamaño del obstáculo. Además de permitir la detección de obstáculos en tiempo real y de forma segura, las técnicas basadas en imágenes no se ven alteradas por los ruidos electromagnéticos del entorno. Además, los datos visuales pueden utilizarse para guiar al robot mediante diversas técnicas de navegación basadas en imágenes disponibles en la actualidad. Los métodos de detección de obstáculos basados en imágenes pueden dividirse en monoculares y estereoscópicos. Los primeros utilizan una sola imagen, mientras que los segundos emplean imágenes captadas por dos cámaras sincronizadas. Los métodos monoculares se dividen en cuatro grupos: basados en la apariencia, basados en el movimiento, basados en la profundidad y basados en la expansión (Badrloo *et al.*, 2022).

Los **métodos basados en la apariencia** consideran un obstáculo como un objeto en primer plano contra un fondo uniforme (es decir, el suelo o el cielo). Se basan en un conocimiento previo del fondo relevante en forma de características de borde, color, textura o forma. La detección de obstáculos se realiza a partir de imágenes individuales tomadas secuencialmente con una cámara montada delante del robot. La imagen adquirida se examina para ver si se ajusta a las características del cielo o del suelo; si no es así, se considera un pixel de obstáculo (Badrloo *et al.*, 2022).

En los **métodos basados en el movimiento**, se supone que los objetos cercanos tienen movimientos bruscos que pueden detectarse mediante vectores de movimiento en la imagen. El proceso consiste en tomar dos imágenes o fotogramas sucesivos en muy poco tiempo. Al principio, se extraen varios puntos de coincidencia en ambos fotogramas. A continuación, se calculan los vectores de desplazamiento de los puntos coincidentes. Dado que los objetos más cercanos a la cámara tienen desplazamientos mayores, cualquier punto con un valor de desplazamiento que supere un umbral determinado se considera un

pixel obstáculo. El flujo óptico es el método utilizado en la mayoría de los enfoques basados en el movimiento (Badrloo *et al.*, 2022).

Al igual que los métodos basados en el movimiento, los enfoques **basados en la profundidad** obtienen información de profundidad a partir de imágenes tomadas por una sola cámara. Hay dos formas de conseguirlo, la primera es el movimiento estereoscópico y la segunda el aprendizaje profundo. En el primero se colocan dos cámaras a los lados del robot y se capturan un par de imágenes consecutivas. Aunque estas imágenes se toman con una sola cámara, pueden considerarse como un par de imágenes estereoscópicas, a partir de las cuales puede estimarse la profundidad de los puntos del objeto. Para ello, se buscan puntos coincidentes en las imágenes. A continuación, se calcula la profundidad de los puntos del objeto mediante cálculos estándar de estimación de la profundidad. Los píxeles cuya profundidad es inferior a un valor umbral se consideran obstáculos (Badrloo *et al.*, 2022).

En los **métodos basados en la expansión** se emplean el mismo principio utilizado por los seres humanos para detectar obstáculos, es decir la tasa de expansión del objeto entre imágenes consecutivas. Como es sabido, un objeto aumenta continuamente de tamaño cuando se acerca. Por lo tanto, la determinación de obstáculos, puntos y/o regiones en dos imágenes secuenciales puede utilizarse para estimar el valor de ampliación del objeto. Este valor podría calcularse entre áreas homólogas, distancias o incluso las escalas SIFT de los puntos extraídos. En los algoritmos basados en la expansión, si el valor de ampliación relativo a un objeto supera un umbral específico, dicho objeto se considera un obstáculo (Badrloo *et al.*, 2022).

Por último, las técnicas híbridas integran datos de sensores activos y pasivos, lo que beneficia y también adolece de los puntos débiles de ambos sistemas. Se han realizado varios estudios en este campo. Sin embargo, hay que tener en cuenta que la fusión de los datos de múltiples fuentes presenta dificultades y complejidades, como la calibración precisa de sistemas multisensor. Además, sólo se han realizado unos pocos estudios sobre el uso en tiempo real de múltiples sensores heterogéneos, normalmente asíncronos. Los datos recogidos por varios sensores no son homogéneos y suelen requerir un complicado procesamiento de datos (Badrloo *et al.*, 2022).

1.1.1.3 Métodos directos

Los métodos directos son métodos de estimación de movimiento, que recuperan los parámetros desconocidos directamente de las cantidades medibles de imagen en cada pixel de la imagen. Esto es en contraste a los "métodos basados en características", que primero extraen un conjunto disperso de puntos clave de cada imagen por separado y posteriormente son recuperados y analizados para determinar el movimiento y la forma, minimizan una medida de error que se basa en distancias entre los puntos clave localizados en la imagen, mientras que los métodos directos minimizan una medida de error que se basa en información de imagen directa recopilada de todos los píxeles de la imagen.

El punto inicial de la mayoría de los métodos directos es la restricción de constancia de brillo, dado dos imágenes $J(x, y)$ e $I(x, y)$,

$$J(x, y) = I(x + u(x, y), y + v(x, y)) \quad (1.1)$$

Donde (x, y) son las coordenadas del pixel y (u, v) denota el desplazamiento del pixel (x, y) entre las dos imágenes. Suponiendo pequeños valores de (u, v) y linealizando I alrededor de (x, y) , se obtiene la siguiente restricción:

$$I_x u + I_y v + I_t = 0 \quad (1.2)$$

donde (I_x, I_y) son las derivadas espaciales del brillo de la imagen, e $I_t = I - J$. Todas las cantidades en estas ecuaciones son funciones de la posición de la imagen (x, y) , por lo tanto, cada pixel proporciona una de esas ecuaciones que restringe el desplazamiento de ese pixel. Sin embargo, dado que el desplazamiento de cada pixel está definido por dos cantidades u y v , la restricción de brillo por sí sola es insuficiente para determinar el desplazamiento de un pixel. Una segunda restricción es proporcionada por un "modelo de movimiento global", es decir, un modelo que describe la variación del movimiento de la imagen (Irani & Anandan, 1999).

En comparación con los métodos basados en características, los métodos directos tienen ventaja, llegando a tener mayor precisión para la estimación del movimiento, esto debido a los fundamentos matemáticos que acompañan a la implementación de estos métodos. (Vela, 2017).

1.1.1.4 Flujo óptico

El término flujo óptico (en inglés *optic flow*) se refiere a la velocidad a la que se mueve la textura en un plano focal de la imagen como resultado de movimiento relativo entre el observador y los objetos del entorno (figura 1.1) (Barrows & Neely, 2000).

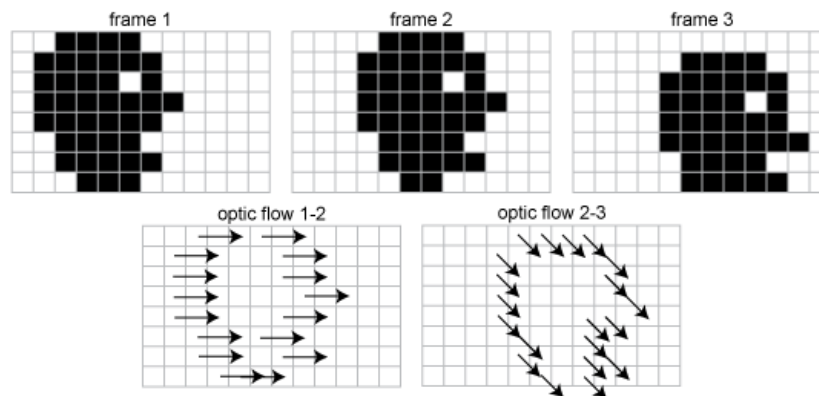


Figura 1.1 Detección del flujo óptico en el plano de la imagen. El ejemplo muestra tres cuadros, que representa el movimiento de la silueta de una cabeza. El flujo óptico se representa como la correspondencia de pixeles de contorno entre el cuadro 1 y 2, así como el cuadro 2 y 3 (Raudies, 2013).

El flujo óptico se formula típicamente como un campo vectorial sobre una imagen, en la que los vectores definen la velocidad a la que se mueve la textura en el plano de la imagen (Barrows & Neely, 2000).

De acuerdo con Sanahuja (*et al.*) (2011) y de la Rosa-Vidal (*et al.*) (2020a) los algoritmos de cálculo de flujo se pueden clasificar en cuatro categorías principales:

- **Algoritmos basados en técnicas diferenciales:** basados en las derivadas espacio-temporales del patrón de intensidad de las imágenes (medido como niveles de gris).
- **Algoritmos basados en la correspondencia (*matching*):** el flujo óptico se define a partir de la correspondencia regional de una secuencia de imágenes. Las imágenes se dividen en varias regiones y la mejor coincidencia entre dos imágenes consecuentes y sus regiones describe el flujo óptico en cada instante.
- **Algoritmos basados en análisis de frecuencia:** utilizan información de la fase o la energía de la salida de filtros sintonizados en velocidad.
- **Algoritmos basados en el análisis de fase:** el flujo óptico es el resultado del comportamiento de fase de la imagen cuando se le aplican varios filtros de pase de banda.

La mayoría de estas técnicas se componen de las siguientes tres etapas de procesamiento:

1. **Filtrado previo**, con el fin de destacar estructuras de interés y mejorar la relación señal-ruido de la entrada, se suele realizar un suavizado espacial y temporal con filtros de Gauss, que normalmente se lleva a cabo mediante convolución con máscaras discretas que se centran consecutivamente en los distintos puntos (píxeles) de la imagen que se procesa.
2. **Extracción de medidas o estructuras básicas de la imagen**, como derivadas espacio-temporales o superficies locales de correlación.
3. **Integración de la información obtenida**, por medio de regularización, correlación o mínimos cuadrados (Sanahuja, 2011).

Para este trabajo se seleccionaron los algoritmos de Horn-Schunck (1981) y de Lucas-Kanade (1981) que pertenecen a los algoritmos basados en técnicas diferenciales.

Ecuación de restricción de Flujo Óptico

La base del flujo óptico diferencial es la ecuación de restricción de movimiento que se describe a continuación (Barron & Thacker, 2005).

Se propone que $J(x, y, t)$ es el pixel central en una vecindad de $n \times n$ píxeles, y se mueve una pequeña distancia $(\delta y, \delta x)$ en un tiempo (δt) hacia $I(x + \delta x, y + \delta y, t + \delta t)$. Se dice entonces que $J(x, y, t)$ y su desplazamiento $I(x + \delta x, y + \delta y, t + \delta t)$ son imágenes del mismo punto y por lo tanto se tiene:

$$J(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \quad (1.3)$$

Esta suposición constituye la base de la ecuación de restricción del movimiento en 2D y se cumple en una primera aproximación (pequeñas traslaciones locales) siempre que $\delta x, \delta y, \delta t$ no sean demasiado grandes. Haciendo uso de una expansión en serie de Taylor de primer orden sobre $I(x, y, t)$ en la ecuación (1.3) y con el uso de las fórmulas mencionadas por Barron y Thacker (2005) se obtiene:

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0 \quad (1.4)$$

Donde u y v son las componentes x y y de la velocidad de la imagen o flujo óptico y $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}, \frac{\partial I}{\partial t}$ son las derivadas de la intensidad de la imagen en (x, y, t) .

También se han introducido las abreviaturas adicionales I_x, I_y , e I_t para las derivadas parciales del brillo de la imagen con respecto a x, y y t , respectivamente.

$$I_x u + I_y v + I_t = 0 \quad (1.5)$$

Más compactadamente, la ecuación (1.5) se escribiría:

$$(I_x, I_y) \cdot (v_x, v_y) + I_t = 0 \quad \text{ó} \quad \nabla I \cdot \vec{v} + I_t = 0 \quad (1.6)$$

Donde ∇I es el gradiente de intensidad espacial, \vec{v} es el flujo óptico en el pixel (x, y) e I_t es la derivada temporal de la intensidad del brillo (Barron & Thacker, 2005).

Estimación de las derivadas parciales

Los algoritmos de flujo óptico de Horn-Schunck (1981) y de Lucas-Kanade (1981) son técnicas diferenciales por lo que el flujo óptico se calcula a través de derivadas espacio-temporales de una secuencia de imágenes (de la Rosa, Guerrero, & Lenero, 2020). Por lo tanto, independientemente del algoritmo de cálculo del flujo óptico, primero se procede a estimar las derivadas de la luminosidad a partir del conjunto discreto de mediciones de luminosidad de la imagen disponibles. Aunque hay muchas fórmulas para la diferenciación aproximada, para este proyecto se utiliza un conjunto que da una estimación de I_x, I_y e I_t (Horn & Schunck, 1981).

$$I_x \approx \frac{1}{4} \{ I_{i,j+1,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i+1,j,k} + I_{i,j+1,k+1} - I_{i,j,k+1} + I_{i+1,j+1,k+1} - I_{i+1,j,k+1} \} \quad (1.7)$$

$$I_y \approx \frac{1}{4} \{ I_{i+1,j,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i,j+1,k} + I_{i+1,j,k+1} - I_{i,j,k+1} + I_{i+1,j+1,k+1} - I_{i,j+1,k+1} \} \quad (1.8)$$

$$I_t \approx \frac{1}{4} \{ I_{i,j,k+1} - I_{i,j,k} + I_{i+1,j,k+1} - I_{i+1,j,k} + I_{i,j+1,k+1} - I_{i,j+1,k} + I_{i+1,j+1,k+1} - I_{i+1,j+1,k} \} \quad (1.9)$$

Donde I_x, I_y e I_t son las derivadas parciales con respecto a x, y y t e I representan los valores de intensidad de la imagen original en escala de grises. Las fórmulas antes mencionadas se pueden representar en máscaras de convolución para la imagen como se ve en la figura 1.2. La relación en el espacio y el tiempo en este cálculo se muestra en la figura 1.3.

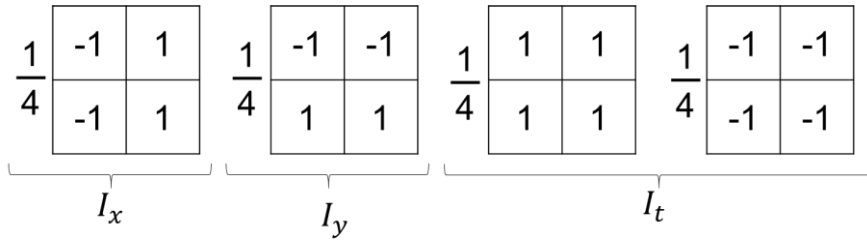


Figura 1.2 Máscaras de convolución utilizadas para las derivaciones parciales en x , y y t .

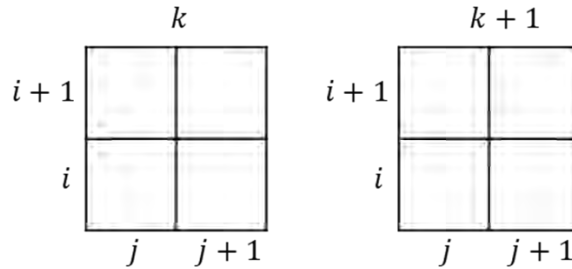


Figura 1.3 Las tres derivadas parciales de las intensidades de las imágenes k y $k+1$, con base en la ubicación del pixel j , i se estiman cada una de ellas. Aquí el índice de columna j corresponde a la dirección x de la imagen, el índice de fila i a la dirección y , mientras que k indica la dirección del tiempo (Imagen k e imagen $k+1$).

Algoritmo de Horn-Schunck

Este algoritmo propone una forma de determinar el flujo óptico basada en el cálculo de los gradientes espaciales y temporales con iteraciones.

Este método considera que, si cada punto del patrón de brillo puede moverse de forma independiente, hay pocas esperanzas de recuperar la velocidad. En este caso, los puntos vecinos de los objetos tienen velocidades similares y el campo de velocidad de los patrones de brillo en la imagen varía suavemente en casi todas partes. Se pueden esperar discontinuidades en el flujo cuando un objeto ocluye a otro (Horn & Schunck, 1981).

El método combina la ecuación de restricción de flujo óptico (1.6) con el término de regularización para estimar el campo de velocidad $\vec{v} = (v_x, v_y)$ y minimizando la cantidad de error al cuadrado queda la siguiente ecuación:

$$E(x, y) = \int_D (\nabla I \cdot \vec{v} + I_t)^2 + \alpha^2 \left[\left(\frac{\partial v_x}{\partial x} \right)^2 + \left(\frac{\partial v_x}{\partial y} \right)^2 + \left(\frac{\partial v_y}{\partial x} \right)^2 + \left(\frac{\partial v_y}{\partial y} \right)^2 \right] dx dy \quad (1.10)$$

Donde las derivadas parciales son los términos de error, y α es un factor de suavidad que se agrega para completar la ecuación de restricción de flujo óptico (Barranco *et al.*, 2017).

Las ecuaciones iterativas son utilizadas para minimizar la ecuación (1.20) y la velocidad de la imagen puede obtenerse a partir de las ecuaciones de Gauss Seidel que resuelven las ecuaciones de Euler-Lagrange adecuadas:

$$u^{n+1} = \bar{u}^n - I_x [I_x \bar{u}^n + I_y \bar{v}^n + I_t] / (\alpha^2 + I_x^2 + I_y^2) \quad (1.11)$$

$$v^{n+1} = \bar{v}^n - I_y [I_x \bar{u}^n + I_y \bar{v}^n + I_t] / (\alpha^2 + I_x^2 + I_y^2) \quad (1.12)$$

Donde \bar{u} y \bar{v} son los promedios locales y se definen como sigue:

$$\begin{aligned}\bar{u}_{i,j,k} &= \frac{1}{6}\{u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\} \\ &+ \frac{1}{12}\{u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}\}\end{aligned}\quad (1.13)$$

$$\begin{aligned}\bar{v}_{i,j,k} &= \frac{1}{6}\{v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\} \\ &+ \frac{1}{12}\{v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\}\end{aligned}\quad (1.14)$$

Las ecuaciones (1.13) y (1.14) se representa como una máscara de convolución como se muestra en la figura 1.4.

1/12	1/6	1/12
1/6	-1	1/6
1/12	1/6	1/12

Figura 1.4 El laplaciano se estima restando el valor de un punto de una media ponderada de los valores de los puntos vecinos. Aquí se muestran los pesos adecuados por los que se pueden multiplicar los valores (Horn & Schunck, 1981)

El proceso de obtención de flujo óptico se muestra en el algoritmo 1.1 el cual está basado en la teoría presentada en esta sección y en el trabajo de Barron & Thacker (2005).

Algoritmo 1.1. Cálculo del flujo óptico mediante el método de Horn-Schunck

Datos: Par de imágenes sucesivas (separadas por un determinado tiempo) y valor de **alpha** y número de iteraciones.

Resultado: Valores de flujo óptico en **u** y **v** para cada pixel de la imagen.

Inicio

1. Se obtiene las derivadas parciales en I_x, I_y e I_t (ecuaciones 1.7, 1.8 y 1.9)

2. **Para** i de 0 al número de iteraciones **hacer**:

2.1 Convolución de los valores de **u** con el kernel de promediado (ecuación 1.13)

2.2 Convolución de los valores de **v** con el kernel de promediado (ecuación 1.14)

2.3 Obtener los nuevos valores de **u** (ecuación 1.11)

2.4. Obtener los nuevos valores de **v** (ecuación 1.12)

3. **Fin**

Fin

Algoritmo de Lucas-Kanade

El método de Lucas-Kanade obtiene el flujo óptico por el principio de estimación de mínimos cuadrados. La suposición en este caso es que los puntos vecinos de un punto específico del objeto varían suavemente y poseen exactamente la misma velocidad. La ecuación de constancia del brillo puede representarse en formato de matriz (Basak et al., 2012) como:

$$\nabla I^T U = -I_t \quad (1.15)$$

donde, $\nabla_I = [I_x I_y]^T$ y $U = [u v]^T$. Por lo tanto, la estimación por mínimos cuadrado de la función de error se puede escribir como:

$$\xi(u, v) = \Sigma \Sigma [I(x + \delta x, y + \delta y, t + \delta t) + I(x, y, t)]^2 \quad (1.16)$$

Suponiendo que el desplazamiento es pequeño y aproximadamente constante dentro de la vecindad de un punto del objeto, la expansión de Taylor de $I(x + \delta x, y + \delta y, t + \delta t)$ sería:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + u \frac{\partial I}{\partial x} + v \frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} \quad (1.17)$$

Sustituyendo esta expresión en la ecuación (1.17) se obtiene,

$$\xi(u, v) = \Sigma \Sigma [u I_x + v I_y + I_t] \quad (1.18)$$

Ahora, el valor optimizado de las velocidades del flujo se logrará minimizando esta función de error con respecto a u y v . Esto dará lugar a un sistema de ecuaciones lineales, cuya representación matricial producirá la siguiente forma estándar de Lucas-Kanade (Basak et al., 2012)

$$U = E^{-1} d \quad (1.19)$$

Donde $E = \begin{bmatrix} \Sigma \Sigma I_x^2 & \Sigma \Sigma I_x I_y \\ \Sigma \Sigma I_x I_y & \Sigma \Sigma I_y^2 \end{bmatrix}$ y $d = \begin{bmatrix} -\Sigma \Sigma I_x I_t \\ -\Sigma \Sigma I_y I_t \end{bmatrix}$.

Finalmente, aplicando las propiedades de las matrices y los productos matriciales correspondientes se obtiene:

$$u = \frac{-\Sigma \Sigma I_y^2 \Sigma \Sigma I_x I_t + \Sigma \Sigma I_x I_y \Sigma \Sigma I_y I_t}{\Sigma \Sigma I_x^2 \Sigma \Sigma I_y^2 - (\Sigma \Sigma I_x I_y)^2} \quad (1.20)$$

$$v = \frac{\Sigma \Sigma I_x I_t \Sigma \Sigma I_x I_y - \Sigma \Sigma I_x^2 \Sigma \Sigma I_y I_t}{\Sigma \Sigma I_x^2 \Sigma \Sigma I_y^2 - (\Sigma \Sigma I_x I_y)^2} \quad (1.21)$$

El cálculo de estas velocidades de flujo óptico de u y v involucran un pixel central y una cantidad de vecinos elegida por el usuario.

El proceso de obtención de flujo óptico se muestra en el algoritmo 1.2 el cual está basado en el trabajo de (Basak et al., 2012) y en la teoría presentada en esta sección.

Algoritmo 1.2 Cálculo del flujo óptico mediante el método de Lucas-Kanade

Datos: Par de imágenes sucesivas (separadas por un determinado tiempo) y tamaño de las ventanas de integración.

Resultado: Valores de flujo óptico en u y v para cada pixel de la imagen.

Inicio

1. Se obtiene las derivadas parciales en I_x, I_y e I_t (ecuaciones 1.7, 1.8 y 1.9).
2. Se define el valor de **mitadVentana** con la parte entera de la división del tamaño de la ventana entre 2
3. **Para** $i = (\text{mitadVentana}+1)$ a (número de filas en I_x , - **mitadVentana**) con paso de 1 **hacer**:
 - 3.1 **Para** $j = (\text{mitadVentana}+1)$ a (número de columnas en I_x , - **mitadVentana**) con paso de 1 **hacer**:
 - 3.1.1. $\text{filas} = i - \text{mitadVentana}$ a $i + \text{mitadVentana}$
 - 3.1.2. $\text{columnas} = j - \text{mitadVentana}$ a $j + \text{mitadVentana}$
 - 3.1.3 Se seleccionan las regiones de las ventanas para I_x, I_y e I_t

$$\text{temp}x = I_x(\text{filas}, \text{columnas});$$

$$\text{temp}y = I_y(\text{filas}, \text{columnas});$$

$$\text{temp}t = I_t(\text{filas}, \text{columnas});$$
 - 3.1.4 Se obtienen los valores de u (ecuación 1.20)
 - 3.1.5 Se obtienen los valores de v (ecuación 1.21)

3.2 **Fin**4. **Fin****Fin**

1.1.1.5 Sistemas de tiempo real

En 1990 Stankovic y Ramamritham definieron los sistemas de tiempo real como aquellos sistemas en los que la corrección del sistema depende no sólo de los resultados lógicos de cálculos, sino también del momento en que se producen los resultados. Este tipo de sistemas dependen de las características del entorno en el que se desarrollan y del tiempo en el que se computan las tareas (Granillo & Beltran, 2018).

1.1.1.6 Tiempo real en la navegación robótica

Dados los conceptos presentados anteriormente, para este trabajo en el que se desarrolla un sistema de navegación para un robot móvil, el término tiempo real es dependiente de factores que se describen a continuación.

La velocidad de captura de fotogramas por segundo (fps) de una cámara montada sobre un robot en movimiento es la cantidad de fotogramas que se obtienen del sensor por segundo (ecuación 1.22). Esta velocidad es importante en los sistemas de navegación basados en el movimiento dado que si la velocidad de captura de fotogramas por segundo es muy baja las intensidades de los pixeles de las capturas realizadas por la cámara de un robot en movimiento no coinciden para realizar el cálculo del movimiento de los mismos. Por otro lado, una velocidad de fotogramas por segundo muy alta puede hacer que el movimiento de los pixeles de los fotogramas capturados sea muy pequeño y el robot tome decisiones erróneas en base a esta información.

$$fps = \frac{\text{fotogramas capturados}}{\text{tiempo (s)}} \quad (1.22)$$

La velocidad de procesamiento de un par de fotogramas en un tiempo determinado (ecuación 1.23) está determinando por la capacidad de la computadora que controla el robot. Esta velocidad en los sistemas de navegación basado en el movimiento contempla el cálculo del desplazamiento de las intensidades de los píxeles y el procesamiento de los datos calculados. Entre mayor sea esta velocidad el robot podrá responder de mejor manera ante la presencia de obstáculos.

$$\text{velocidad de procesamiento} = \frac{\text{pares de fotogramas procesados}}{\text{tiempo (s)}} \quad (1.23)$$

Por último, la velocidad a la que se desplaza un robot móvil el cual consiste en la distancia recorrida en un tiempo determinado (ecuación 1.24) es importante para un sistema de navegación basado en el movimiento dado que ante la presencia de obstáculos frente al robot es necesario que el robot pueda tomar la decisión a tiempo para poder esquivarlos, por lo que una velocidad de desplazamiento alta puede hacer que el robot no pueda responder a tiempo ante la presencia de un obstáculo.

$$\text{velocidad del robot móvil} = \frac{\text{distancia recorrida por el robot (cm)}}{\text{tiempo (s)}} \quad (1.24)$$

Por lo tanto, para que un sistema de navegación robótica funcione en tiempo real es necesario que las velocidades de captura, procesamiento y desplazamiento sean las adecuadas para el tipo de robot utilizado (terrestre, aéreo o acuático), el entorno en el que se vaya utilizar (interiores o exteriores) y la tarea para el cual fue construido.

Para este proyecto se define que un sistema funciona en *tiempo real* cuando las velocidades de captura, procesamiento y desplazamiento son las adecuadas para un robot rodante (terrestre) que evade obstáculos en espacios interiores.

1.1.2 Antecedentes del proyecto

En el Centro Nacional de Investigación y Desarrollo Tecnológico (TecNM/CENIDET), más específicamente dentro del grupo de inteligencia artificial se han desarrollado algunas tesis sobre la aplicación de los métodos directos, estos métodos son base para la realización de este proyecto y por lo tanto conforman los antecedentes para esta tesis, así también existen trabajos en cuanto a navegación de robots móviles. A continuación, se mencionan los más relevantes.

Vela (2017) en su trabajo “Experimentación con Odometría Visual por Métodos Directos” revisa diferentes trabajos relacionados con los métodos directos, con el objetivo de evaluarlos e implementarlos en un sistema de navegación robótica. Vela realizó la caracterización del funcionamiento de estos modelos, a través de la experimentación con secuencias de videos, con la finalidad de obtener resultados comparables, asimismo utilizando la calidad de la reconstrucción de la trayectoria de la cámara (monocular) como criterio de evaluación. Como aportación, propuso un método que calcula la odometría visual haciendo uso de la librería *OpenCV* para calcular la homografía entre dos imágenes tomando en cuenta solo las intensidades de los píxeles. Finalmente, aunque obtuvo buenos

resultados, su enfoque es demasiado complejo y no fue posible implementarlo en tiempo real.

En el trabajo de (Velázquez, 2019) titulado “Odometría mediante visión artificial usando métodos directos” se reporta una investigación de los métodos directos y semidirectos para odometría visual. Como resultado, este trabajo 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. Se utilizó una unidad de medición inercial (IMU, por sus siglas en inglés) para reducir el costo computacional de la aplicación de métodos semidirectos en un sistema embebido como la *Tinker Board*. Sin embargo, dado el costo computacional, para generar el mapa de la trayectoria tuvieron la necesidad de realizar movimientos pequeños de la cámara para evitar que estos algoritmos perdieran el seguimiento del parche o región de píxeles.

En el trabajo de (Navarrete, 2019) titulado “Sistema de Navegación Inercial Asistido por Visión para Robots Móviles Terrestres” se presenta un estudio e implementación de técnicas de odometría inercial con la IMU, con el objetivo de fusionar sus mediciones con datos visuales y con esto lograr reducir la incertidumbre en la estimación de la posición de un robot móvil obtenida por un Sistema de Navegación Inercial. Para este trabajo Navarrete utilizó sistemas embebidos como una placa *Arduino* y una *Raspberry Pi 3* para capturar la información inercial y procesarla junto con las imágenes capturadas por la cámara, aunque sus resultados fueron variados, su sistema de navegación logró reducir el error en la estimación de la posición del robot al apoyar al sistema inercial con los datos obtenidos del flujo óptico. Para el algoritmo de visión utilizó métodos basados en características, más específico el algoritmo de Lucas-Kanade (LK) para obtener el flujo óptico y determinar el movimiento de la cámara.

1.1.3 Estado del arte

En esta sección se presentan una parte de los trabajos del estado del arte estudiados para el desarrollo de este proyecto.

1.1.3.1 Desarrollo de sensores de flujo óptico

Inspirado en la noción de flujo óptico, desde finales de la década de los '90, los sensores de movimiento visual se han desarrollado y aplicado en una variedad de prototipos voladores. A continuación, se presentan los trabajos analizados.

En el trabajo de (Barrows & Neely, 2000) desarrollado en el *Naval Research Laboratory* (NRL) de los Estados Unidos se muestra el trabajo del desarrollo de un sensor de flujo óptico compacto para navegación en pequeña escala que incluye las tareas de evasión de obstáculos, control de altitud, aterrizaje y otras tareas similares. Se hace mención del uso de un sensor “híbrido” cuyos objetivos son conservar parte del tamaño compacto, el bajo consumo de energía y las características de alta velocidad de los circuitos analógicos. En este enfoque, el procesamiento frontal se realiza en el dominio de señal analógica o mixta utilizando fotorreceptores de forma especial, circuitos analógicos y puertas digitales simples.

En el trabajo de (Green *et al.*, 2004) se desarrolla un Micro Vehículo Aéreo (MAV, por sus siglas en inglés) adaptando del comportamiento y los patrones de vuelo de los insectos, capaz de controlar el vuelo y evadir obstáculos. Se menciona que los insectos hacen un uso intensivo de la visión, especialmente el flujo óptico, para percibir el entorno los insectos realizan una variedad de tareas en entornos complejos utilizando sus capacidades de detección de flujo óptico natural. Esto aplicado en el vuelo de un MAV, la presencia de obstáculos cercanos producirá magnitudes de flujo óptico más altas (figura 1.5). Aplicando los métodos de navegación de los insectos desarrollaron un microsensor de flujo óptico con un peso de 4.8 gramos.

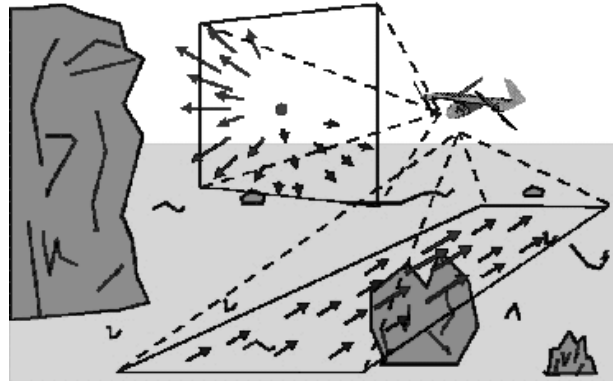


Figura 1.5. Comportamiento del flujo óptico en un robot aéreo (Green *et al.*, 2004)

En el trabajo de (Conroy *et al.*, 2009) se presenta una implementación de navegación visual inspirada en insectos que utiliza descomposiciones espaciales del flujo óptico instantáneo para extraer información de proximidad local. Este planteamiento se probó en un entorno de un corredor con un cuadricóptero donde todos los sensores y procesamiento, incluida la altitud, la orientación y el control del bucle exterior se realizan a bordo. Se logra la detección y el procesamiento visuales omnidireccionales utilizando una placa de cámara *Surveyor*, que consta de un procesador de punto fijo *Blackfin* de 500 MHz, que se conecta a una cámara *Omnivision OV 7725*. La cámara captura imágenes de tamaño 160 x 120 a 55 fotogramas por segundo (fps).

En el trabajo de (Duhamel *et al.*, 2013) se presenta el diseño y fabricación de un sensor de altitud basado en flujo óptico llamado *Tam4*, con peso de 31 mg, para un micro robot volador con un peso de 68 mg. Este sensor está diseñado a medida para bajo consumo y aplicaciones masivas como sistemas embebidos y robótica. El sensor consta de una matriz de 4x32 pixeles (128 pixeles), optimizada para detectar movimiento a lo largo de un solo eje. Los pixeles de este sensor tienen una relación de aspecto de 8:1 con un tamaño de pixel de 336x42 μm , de modo que el plano focal es un cuadrado con un área de 1.34 mm^2 . La altitud estimada se calcula a partir del flujo óptico generado al capturar una imagen de la pantalla texturizada colocada frente al sensor óptico.

En el trabajo de (Park *et al.*, 2019) se reporta la fabricación de un sensor híbrido (analógico/digital) de flujo óptico 2D de baja potencia para construir unos ojos compuestos artificiales, que pueden proporcionar una capacidad de detección de campo de visión

amplio para la navegación autónoma de micro vehículos aéreos (MAV). Inspirado por la visión de insectos, desarrollaron un algoritmo de detección del flujo óptico bidimensional basado en marcas temporales para escalar el número de sensores desplegados a baja potencia. El sensor fabricado estima flujos ópticos 2D de 16 bits hasta un máximo de 1.96 rad/s. El tamaño total del chip es $3.09 \times 4.18 \text{ mm}^2$, incluidas las almohadillas de entrada/salida. El chip contiene una matriz de 64×64 píxeles.

1.1.3.2 Desarrollo/Implementación de técnicas de visión robótica con uso del flujo óptico

Aparte del desarrollo de sensores de flujo óptico, se encuentran trabajos donde a través del cálculo del flujo óptico aplicado a imágenes consecutivas obtenidas de cámaras, *datasets* o simulaciones desarrollan sistemas de navegación para robots rodantes o voladores. A continuación, se presentan los trabajos analizados.

En el trabajo de (Soria & Carelli, 2006) se trata el problema del uso de imágenes panorámicas para evitar obstáculos en la navegación de un robot móvil en interiores. Para este propósito hace uso de una cámara con un espejo omnidireccional instalado en el robot, para adquirir imágenes panorámicas con un campo de visión de 180 grados frente del robot. Al principio del ciclo de control, se obtienen dos vistas panorámicas. El algoritmo adoptado para realizar este cálculo es el de la estimación de mínimos cuadrados con fusión de Kalman. Esta implementación divide la imagen en regiones de 9×9 píxeles, cada una asociada a un solo vector de flujo óptico, tal que se obtiene un campo de 15×30 vectores de flujo óptico. Para determinar los tiempos para el contacto se consiguen las coordenadas del foco de expansión (*Focus Of Expansion*, FOE) del campo de flujo óptico.

En el trabajo de (Bermudez & Fearing, 2009) se aborda el desarrollo del control de vuelo de un ornitóptero de 7 gramos disponible comercialmente con una cámara a bordo y un módulo de CPU con una masa de 2.5 gramos y una batería de 2.6 gramos. Dentro del hardware utilizado se encuentra un microchip *dsPIC33FJ128MC706* microprocesador de 16 bits en funcionamiento a 40 MHz y una cámara *OmniVision OV7660FSL* VGA. La información recibida por la cámara era en blanco y negro con una resolución de 160×120 (QQVGA) a una tasa de 25fps y después guardada en los 16KB de la memoria RAM del dsPIC. El poco espacio de memoria limitó la adquisición de imágenes a una resolución final de 18×13 , para lo cual se hizo la convolución de la imagen con un filtro gaussiano de 3×3 . El algoritmo de flujo óptico utilizado en este trabajo es el algoritmo de correlación del detector de movimiento elemental estándar (EMD, por sus siglas en inglés).

En el trabajo de (Conroy *et al.*, 2009) se presenta una implementación de navegación visual inspirada en insectos que utiliza descomposiciones espaciales del flujo óptico instantáneo para extraer información de proximidad local. El enfoque se demuestra en un entorno de corredor en un cuadricóptero MAV donde todos los sensores y procesamiento, incluida la altitud, la orientación y el control del bucle exterior se realizan a bordo. Este trabajo logra la detección y el procesamiento visuales omnidireccionales utilizando una placa de cámara *Surveyor*, que consta de un procesador de punto fijo *Blackfin* de 500 MHz, que se conecta a una cámara *OmniVision OV 7725*. La cámara captura imágenes de tamaño 160×120 a

55 fps. Se obtiene un campo de visión de 360 grados a partir de un espejo parabólico instalado sobre el cuadricóptero.

En el trabajo de (Sanahuja *et al.*, 2011) se presenta la implantación de control de movimiento de un robot móvil basado en el flujo óptico. La aportación de este trabajo es la presentación de una estrategia de cálculo del flujo óptico que pueda ser implantado con recursos limitados en los cuales se desee un funcionamiento en tiempo real. Para el desarrollo del algoritmo de flujo óptico se usó un algoritmo que consiste en hacer una interpolación de la imagen (12A, *Image Interpolation Algorithm*). Para la aplicación de este algoritmo se utilizó un robot móvil *e-puck*. Dentro de sus características destacan un acelerómetro 3D y una cámara con una resolución de 640×480 a 4 fps. Dado que la resolución es demasiado grande para la memoria del microcontrolador del robot se aplicaron funciones de reducción de la resolución sobre la ventana de la imagen a 320×1 píxeles.

En el trabajo de (Liyanage & Perera, 2012) se presenta el desarrollo de un sistema para guiar a una persona con discapacidad visual a evitar obstáculos mediante información auditiva y táctil. Para demostrar los conceptos básicos investigados, se diseñó y desarrolló un prototipo consistente en un mundo de realidad virtual. También emplea un algoritmo de flujo óptico existente para la estimación del movimiento, otras técnicas de procesamiento de imágenes, síntesis de voz para la retroalimentación auditiva y programación integrada para la retroalimentación táctil. El algoritmo utilizado para el cálculo de flujo óptico fue el de Horn-Schunck y la técnica empleada para la detección de obstáculos consta de dividir la imagen en dos partes de manera vertical y determinar mediante la diferencia de la suma de magnitudes de la mitad de la imagen (izquierda, derecha) con la menor suma de magnitudes de flujo óptico.

En el trabajo de (Conradt, 2015) se presenta un método simplificado para calcular el flujo óptico en un sistema de visión integrado embebido, adecuado para robots voladores miniaturizados en interiores. Para este propósito, el autor hace uso de una combinación de un sensor de visión de uso general y un algoritmo de procesamiento adaptado para extraer el flujo óptico en tiempo real utilizando un hardware de procesamiento minimalista, que puede operar a bordo de pequeños robots autónomos. En este artículo hacen uso de un sensor de visión dinámica (DVS por sus siglas en inglés) con una resolución de 128×128 píxeles que responde a los cambios temporales de contraste a nivel de cada pixel. Este sensor combinado con una interpretación del flujo óptico global determina el tipo de movimiento.

En el trabajo de (Sanchez *et al.*, 2015) se propone un método para evitar obstáculos, sin calcular constantemente el campo de flujo óptico. Sólo se calcula cuando el robot está cerca de colisionar con un obstáculo, y así, utiliza la divergencia del campo de flujo para decidir qué dirección debe tomar. Se han realizado experimentos físicos con un robot real en entornos desconocidos.

En el trabajo de (Li *et al.*, 2016) se propone un método de odometría visual usando el flujo óptico en una cámara monocular. A pesar de que muchos trabajos hacen uso de la

navegación basada en características, aquí se propone diseñar una aproximación del flujo óptico sin rasgos distintivos (*featureless*). Para esto utilizaron el algoritmo de Lukas-Kanade (LK) con una rutina RANSAC (*Random Sample Consensus*) para recuperar el movimiento de la cámara de forma robusta. En este trabajo fue probado con el *dataset* HRBB el cual consta de imágenes con una resolución de 640×360 a 30 fps.

En el trabajo de (Agrawal *et al.*, 2017) se propone una estrategia de navegación basada en visión para un vehículo aéreo no tripulado (Unmanned Aerial Vehicle, UAV) que navega por cañones urbanos. El UAV tiene una sola cámara orientada hacia delante y se basa en el flujo óptico calculado a partir de la secuencia de imágenes captadas por la cámara. La estrategia de balance para la evasión de obstáculos está basada en la diferencia de flujo óptico entre los dos lados de la imagen (derecho e izquierdo).

En el trabajo de (Sabo *et al.*, 2017) se propone un sistema robótico barato y ligero para modelar la visión de los insectos. El sistema se monta y se prueba en una plataforma robótica para aplicaciones móviles, y luego se evalúan la cámara y los modelos de visión de los insectos. Se analiza el potencial del sistema para su uso en la incorporación de procesos visuales de alto nivel (es decir, la detección de movimiento) y también para desarrollo de la navegación basada en la visión para la robótica en general. El flujo óptico es calculado a partir de imágenes con una resolución de 75×75 píxeles.

En el trabajo de (McGuire *et al.*, 2017) se presenta un algoritmo de visión por computadora altamente eficiente llamado *Edge-FS* para la determinación de velocidad y profundidad. Para este algoritmo se utilizó una cámara estéreo con resoluciones de 128×96 píxeles, esta cámara fue conectada a un microprocesador STM32F4 con una memoria de 196 Kb.

En el trabajo de (Chirarattananon, 2018) se explica un método de flujo óptico directo en el contexto de la estimación de la altitud de vuelo. Consideraron un robot volador con una cámara monocular mirando hacia abajo. También desarrollaron una rutina de estimación basada en un filtro de Kalman para incorporar las medidas de una IMU a proporcionar el factor de escala necesario para estimar la distancia absoluta. Las imágenes obtenidas de una resolución de 320×240 píxeles y el algoritmo para determinar el flujo óptico entre imágenes consecutivas fue el de Lucas- Kanade (LK). Para este algoritmo se detectaban 50 características de cada imagen para posteriormente aplicar el algoritmo de LK, todo ese proceso fue desarrollado en *Python* con la librería *OpenCV*. En las pruebas tomaron el error cuando el flujo óptico es procesado a 30 fps y 60 fps no obteniendo diferencias significativas para demostrar que el uso de 60 fps fuera más preciso.

En el trabajo de (Granillo & Beltran, 2018) se presenta la implementación de un algoritmo de flujo óptico que asegura el seguimiento y control de la trayectoria en un dron cuadrotor en tiempo real, determinando la distancia entre el dron y un determinado objetivo así como el cálculo de la velocidad y su dirección de desplazamiento mediante la implementación del algoritmo Lucas-Kanade (LK). El propósito central de este trabajo es lograr una solución de bajo costo y más rápida (debido a la dinámica del UAV); además de la implementación de un Sistema de Posicionamiento Global (GPS) utilizando un Sistema de Navegación Inercial

(INS, por sus siglas en inglés) para generar una trayectoria aérea, por lo que puede ser generada en tiempo real convirtiendo al UAV en una aeronave semi-autónoma.

En el trabajo de (Meneses *et al.*, 2018) se presenta un robot de bajo costo, controlado por una *Raspberry Pi*, cuyo sistema de navegación se basa en la visión. La estrategia utilizada ha consistido en la identificación de obstáculos mediante el reconocimiento de patrones de flujo óptico. Su estimación se realiza mediante el algoritmo Lucas-Kanade (LK), que puede ser ejecutado por la *Raspberry Pi* sin perjudicar su rendimiento. Finalmente, utilizan un clasificador basado en máquinas de vectores de soporte (SVM, por sus siglas en inglés) para identificar los patrones de esta señal asociados al movimiento de los obstáculos.

En el trabajo de (Yadav & Khandekar, 2018) se presenta una técnica de movimiento y evasión de obstáculos de un robot. El cálculo del flujo óptico es obtenido mediante un método diferencial que preserva la intensidad del punto en movimiento, con los valores obtenidos se obtiene información del foco de expansión (*Focus Of Expansión* FOE), tiempo para el contacto (*Time To Contact*, TTC) y profundidad con el fin de brindar la distancia y la velocidad del movimiento en relación con el tiempo. Para este proceso utilizan una cámara con resolución de 640 x 480 píxeles a 30 fps conectada a una computadora que a través de MATLAB calcula el flujo óptico y el TTC para cada fotograma. Dependiendo del TTC, las instrucciones son enviadas a un robot a través del puerto serial de una placa *Arduino*.

El trabajo de (Ponce *et al.*, 2018) presenta un sistema monocular de estimación de distancia utilizando una red neuronal convolucional (CNN) y el cálculo del flujo óptico. Para el cálculo del flujo óptico utilizaron un método basado en la transformación de Hermite (HT) el cual es robusto ante el ruido, es preciso con la información de las texturas y de poco costo computacional. Después de calcular el flujo óptico sobre un conjunto de imágenes de 256x256, este conjunto de datos se ingresó como entrada para entrenar una CNN. También entrenaron otra CNN con la información obtenida mediante un sensor de distancia. Al final obtuvieron que el CNN que fue entrenada con los valores del flujo óptico tuvo una mejor precisión.

En el trabajo de (Bernardo *et al.*, 2019) se presenta la caracterización de un sensor de flujo óptico para la aplicación en la navegación integrada. El algoritmo utilizado es el de Lucas-Kanade (LK) basado en puntos destacados, que para este trabajo usa 50 puntos destacados por imagen. A partir de las dos imágenes se obtiene el desplazamiento del punto en cuestión. Para la navegación utilizaron una resolución de 160 x 120 de un módulo de cámara conectado a una *Raspberry Pi* 3B+. Obtuvieron un sensor de flujo óptico capaz de estabilizar la velocidad en un algoritmo de navegación de un cuadricóptero o pequeños vehículos no tripulados (UAV).

En el trabajo de (Károly *et al.*, 2019) se propone una red neuronal profunda de segmentación de objetos en movimiento para evitar obstáculos dinámicos en el caso de la navegación de robots móviles en interiores. La red se construyó sobre un modelo de aprendizaje profundo preentrenado para el reconocimiento de imágenes RGB. En este trabajo se codificaron los datos de flujo óptico calculados con el método de Gunnar Farneback, y los fotogramas de vídeo juntos en un formato que puede ser procesado por la

red preentrenada. Los resultados de la evaluación empírica muestran que la red propuesta predice máscaras de segmentación satisfactorias en los datos de prueba para la tarea de evitar obstáculos dinámicos en la navegación de robots móviles en interiores mientras opera en tiempo real y siendo robusta al movimiento de la cámara y a la aparición de objetos en movimiento.

En el trabajo de (Moya-Albor *et al.*, 2019) se presenta una metodología para el control autónomo de la evasión de obstáculos utilizando una estimación de flujo óptico bioinspirada. La principal diferencia con otros métodos es que utilizaron un modelo que incluye una transformada de Hermite (*Hermite Transformation*, HT) y una máscara perceptiva. Utilizaron una plataforma robótica física para probar el algoritmo de control, donde debido a la estructura del chasis se definieron movimientos de avance, retroceso y giro. El robot tiene una cámara RGB para capturar imágenes de la trayectoria y luego calcular la estimación del flujo óptico. Como estrategia para la evasión de obstáculos dividen la imagen en tres regiones horizontales espaciadas, todas con la altura de la imagen, que representan la magnitud izquierda, centro y derecha del flujo óptico. Posteriormente se hace un conteo las magnitudes de flujo que superan un umbral y las normalizan en función de la región de máxima densidad para ingresarlo a un conjunto de reglas difusas.

En el trabajo de (Nadour *et al.*, 2019) se presenta el diseño de un sistema de navegación visual para robots móviles en interiores basado en controladores lógicos difusos (FLC) y en el enfoque de flujo óptico (OF). El sistema de control propuesto contiene dos controladores de lógica difusa *Takagi-Sugeno* para la evasión de obstáculos y la búsqueda de objetivos basados en la adquisición de vídeo y el algoritmo de procesamiento de imágenes. El primer controlador de dirección utiliza valores de OF calculados mediante el algoritmo Horn-Schunck para detectar y estimar las posiciones de los obstáculos. Para extraer información sobre el entorno, la imagen se divide en dos partes. El segundo FLC se utiliza para guiar al robot en la dirección del destino final. La eficacia del enfoque propuesto se verifica en simulación utilizando *Visual Reality Toolbox*. Los resultados de la simulación demuestran que el sistema de control visual permite la navegación autónoma sin colisión con obstáculos.

En el trabajo de (Skelton *et al.*, 2019) se presenta un algoritmo de estimación del movimiento en tiempo real y en un sistema embebido. Para esto, utilizan un algoritmo de flujo óptico de densidad basado en el algoritmo de Lucas-Kanade (LK). En este algoritmo el flujo óptico es estimado por el seguimiento de 10 puntos clave de la imagen. Sin embargo, por robustez, las escenas se sembraron con 50 puntos clave mediante el método Shi-Tomasi. Para promover el conocimiento de toda la escena, los puntos clave se vieron obligados a tener un mínimo separación espacial de 2 píxeles a 180×36 y 8 píxeles para 720×144 . El procesamiento de las imágenes estuvo cargo de la tarjeta de NVIDIA Jetson TK1, se midió el tiempo que tarda cada algoritmo en producir una estimación de la energía de movimiento a partir de la entrada visual. Como resultado el algoritmo de Lucas-Kanade con una resolución de 180×36 produjo resultados más consistentes que otros algoritmos.

En el trabajo de (Back *et al.*, 2020) se presenta sistema de navegación seguidor de carriles para bicicleta mediante una red neuronal convolucional entrenada para el control de un

UAV. Se controla el UAV para que siga un sendero determinado manteniendo su posición cerca del centro del sendero utilizando la CNN. Además, para volver a la trayectoria original cuando el UAV se sale de la trayectoria o la cámara pierde el rastro debido a perturbaciones como el viento, los comandos de control de la CNN se almacenan durante un cierto periodo y se utilizan para recuperarse de dichas perturbaciones. Para evitar obstáculos durante la navegación, se utiliza el flujo óptico calculado con otra CNN para determinar la maniobra segura.

En el trabajo de (de la Rosa, Guerrero, & Lenero, 2020) se reporta un sistema personalizado para el cálculo del flujo óptico en tiempo real. Para este fin, implementan un sensor de imagen de baja resolución y un microcontrolador que ejecuta un algoritmo de detección de flujo óptico en tiempo real. Para el cálculo del flujo óptico hicieron uso de una versión simplificada del método de interpolación que usa el emparejamiento regional de imágenes con resolución de 18x18 para reducir el cálculo que se realiza en un microcontrolador con bajas capacidades computacionales. Como resultado realizan el procesamiento del cálculo de flujo óptico en un tiempo de 0.7 ms con imágenes capturadas a 5 fps. Además, dentro de los tipos de algoritmos de flujo óptico, en otro artículo del mismo autor (de la Rosa, Guerrero, & Leñero, 2020) se presenta el resultado de su trabajo donde hacen mención de que la programación del sistema fue en un lenguaje de bajo nivel con el fin de obtener el mejor desempeño.

En el trabajo de (Wang *et al.*, 2020) se propone un algoritmo para estimar los estados (posición y velocidad) de los UAVs basados en el flujo óptico de imágenes obtenidas de una cámara a bordo durante la fase de aterrizaje. En primer lugar, extraen los puntos de control de un objetivo visual detectando un marcador como objetivo de aterrizaje. Posteriormente se calculan la altitud y la posición de los UAV por lo que se propone un método basado en la interpolación de puntos de esquina del algoritmo Lucas-Kanade para calcular el flujo óptico denso. A continuación, se obtiene la integral del flujo óptico esférico mediante el campo de flujo óptico y algunas restricciones sobre la actitud del UAV. Por último, la velocidad del UAV se calcula a partir de la combinación de la integral del flujo óptico con la pose estimada del UAV.

En el trabajo de (Baillieul & Kang, 2021) se parte del concepto de flujo óptico para plantear un sistema de navegación robótica con la información de una cámara de dos píxeles utilizando el método de flujo óptico Lagrangiano, con el cual se rastrean los puntos característicos de la imagen a medida que se desplazan por la retina. Esta forma de detección visual es la base de muchas aplicaciones estándar de visión por ordenador, como Lukas-Kanade y Horn-Schunck. El flujo óptico Lagrangiano tiene sus propias dificultades, entre las que destaca el hecho de que se ve muy afectado por los componentes rotacionales del movimiento. Se muestran estrategias combinadas de dirección y detección para mitigar los efectos de los movimientos de rotación.

En el trabajo de (Editya *et al.*, 2022) se propone utilizar un método de estimación de la dirección para ayudar en la investigación forense de la colisión de un dron. Se aplicaron métodos de flujo óptico, concretamente Lucas-Kanade, Horn-Schunck y Gunnar-

Farnerback. Basándose en los experimentos realizados, la técnica Lucas-Kanade logra una mejor estimación de la dirección proporcionando un vector de movimiento específico.

En la tabla 1.1 se muestra el resumen de los trabajos del estado del arte.

Tabla 1.1 Resumen de los trabajos del estado del arte

Autor(es), año	Robot utilizado	Cámara utilizada	Resolución utilizada	Algoritmo	Propósito
(Barrows & Neely, 2000)	Micro robot aéreo (MAV)	Sensor de flujo óptico	1 × 18	-----	Control de vuelo y evitar colisiones
(Soria & Carelli, 2006)	Robot rodante	-----	-----	<i>Least-mean-squares optical flow calculation</i>	Evasión de obstáculos
(Bermudez & Fearing, 2009)	<i>Flapping wing robot</i>	-----	18 × 13	<i>Elementary motion detector (EMD) correlation algorithm</i>	Estimación de dirección de vuelo
(Sanahuja <i>et al.</i> , 2011)	Robot móvil e-puck (rodante)	Cámara y acelerómetro 3D	320 × 1	<i>12A, Image Interpolation Algorithm</i>	Evasión de obstáculos
(Duhamel <i>et al.</i> , 2013)	<i>Flapping-wing microrobot</i>	Tam4	4 × 32	-----	Control de altitud de vuelo
(Dai & Li, 2015)	-----	Simulación virtual	-----	Horn-Schunck (HS)	Evasión de obstáculos en ambientes desconocidos
(Sanchez <i>et al.</i> , 2015)	Robot rodante	-----	640 × 480	Horn-Schunck (HS)	Evasión de obstáculos
(Agrawal <i>et al.</i> , 2017)	Vehículo aéreo (UAV)	Simulación virtual	150 × 150	Horn-Schunck (HS)	Navegación de vehículos aéreos en cañones urbanos
(McGuire <i>et al.</i> , 2017)	Micro robot aéreo (MAV)	Cámara RGB estéreo	128 × 96	<i>EdgeFlow</i>	Estimación de velocidad y evasión de obstáculos
(Chirarattananon, 2018)	Drone (UAV)	Cam-5CRO-U	320 × 240	Lucas-Kanade (LK)	Estimación de altitud de vuelo
(Granillo & Beltran, 2018)	Drone (UAV)	-----	-----	Lucas-Kanade Piramidal (PLK)	Generación de trayectoria
(Meneses <i>et al.</i> , 2018)	Robot rodante de dos motores	LG AN-VC500	320 × 240	Lucas-Kanade (LK)	Evasión de obstáculos
(Yadav & Khandekar, 2018)	Robot rodante	Camara USB	320 × 240	Estimación basada en una teoría para la deformación (Basado en algoritmo de HS)	Evasión de obstáculos

Tabla 1.1 Resumen de los trabajos del estado del arte (Continuación)

Autor(es), año	Robot utilizado	Cámara utilizada	Resolución utilizada	Algoritmo	Propósito
(Ponce <i>et al.</i> , 2018)	Robot rodante	Simulation virtual	256 × 256	Estimación de la profundidad 3D de la región (Basado en algoritmo de LK)	Estimación de distancia de objetos
(Moya-Albor <i>et al.</i> , 2019)	Robot rodante de cuatro motores	Módulo de cámara v2 para Raspberry	-----	Horn-Schunck (HS) con Transformada de Hermite	Evasión de obstáculos
(Nadour <i>et al.</i> , 2019)	Simulación virtual	Cámara virtual	320 × 480	Horn-Schunck (HS)	Evasión de obstáculos
(Back <i>et al.</i> , 2020)	Drone (UAV)	LifeCam HD-3000	160 × 120	Estimación vía CNN	Navegación por senderos y evasión de obstáculos
(de la Rosa, Guerrero, & Lenero, 2020)	-----	CMOS ADNS-2610	18 × 18	Método de interpolación de imagen	Cálculo del flujo óptico
(Wang <i>et al.</i> , 2020)	Drone (UAV)	Cámara a bordo	-----	Lucas-Kanade (LK)	Estimación de pose y velocidad
(Baillieul & Kang, 2021)	Robot rodante de dos motores	-----	-----	Lagrangiano	Navegación y evasión de obstáculos
(Editya <i>et al.</i> , 2022)	Drone (UAV)	-----	-----	Lucas-Kanade (LK)	Estimación de dirección para investigación forense de colisión

1.2 Descripción del problema

1.2.1 Delimitación del problema específico

Los robots móviles con visión generan grandes cantidades de datos durante sus desplazamientos.

Como se presentó en trabajos previos dentro del CENIDET como los de Vela (2017) y Velázquez (2019) el uso de métodos directos o semidirectos es muy costoso computacionalmente hablando, haciendo que el procesamiento no se pueda realizar en tiempo real.

Como se mencionó en el estado del arte, existen diferentes trabajos donde inspirados en el proceso de vuelo de los insectos voladores se han desarrollado sensores de flujo óptico, así también se han desarrollado sistemas de navegación donde el procesamiento de las imágenes se realiza en computadoras con diferentes capacidades de procesamiento. Sin embargo, para este trabajo se utilizó una computadora con recursos de procesamiento más limitados como la *Raspberry Pi*.

Con los recursos mencionados, se abordó el problema de reducir el alto costo computacional de la visión robótica mediante el procesamiento del flujo óptico utilizando imágenes capturadas de baja resolución. Se busca como producto final, un sistema capaz de detectar la percepción de su entorno (movimiento de la escena) y su estado actual (movimiento o en reposo) en *tiempo real*. Además, este sistema es aplicado a un dispositivo rodante de manera embebida.

1.2.2 Descripción de la complejidad del problema

Dentro de las complejidades está la obtención de imágenes con resoluciones bajas suficientes para poder obtener la información necesaria y al mismo tiempo implique un bajo costo computacional. Otra parte del problema es el cálculo del flujo óptico ya que la capacidad de procesamiento de un sistema embebido como la *Raspberry Pi* es limitada.

Otras complejidades son las causadas por el entorno como la variación de luminosidad, ya sea en un entorno abierto o cerrado, así como entornos sin textura (áreas lisas o de un solo color) que podrían provocar un mal cálculo del flujo óptico haciendo que el dispositivo que implemente el sistema tome decisiones erróneas.

1.3 Descripción de la solución

Dada la problemática descrita en la sección anterior, se plantea desarrollar un sistema de navegación basado en el movimiento del entorno usando imágenes de baja resolución con el fin de reducir el costo computacional del procesamiento del flujo óptico utilizando todos los píxeles de la imagen, para que un robot con una computadora con recursos limitados de procesamiento pueda navegar y evadir obstáculos en *tiempo real*.

1.4 Discusión

De los sistemas de navegación robótica basados en el movimiento las técnicas de flujo óptico son las más utilizadas. Los métodos que perciben el movimiento de la escena mediante el procesamiento de todos los píxeles de la imagen dan resultados más precisos a comparación de los métodos basados en características, esto debido a los fundamentos matemáticos que acompañan a la implementación de estos métodos; sin embargo, el uso de todos los píxeles de la imagen generan grandes cantidades de datos durante el desplazamiento de un robot lo que dificulta su procesamiento en robots que cuentan con computadoras con recursos limitados.

De los trabajos presentados del estado del arte se pueden destacar como referencia para este proyecto los siguientes:

En cuanto al cálculo de flujo óptico con imágenes de baja resolución se destaca el trabajo que presenta (de la Rosa, Guerrero, & Lenero, 2020) donde se realiza el cálculo de flujo óptico con imágenes de una resolución de 18×18 píxeles a 5 fotogramas por segundo (fps), logrando tiempos de procesamiento de 0.7ms. Sin embargo, este tiempo se limita al cálculo del flujo óptico.

Con respecto a la estimación del estado del robot, se destacan las técnicas utilizadas en los trabajos de (Barrows *et al.*, 2002) y (Duhamel *et al.*, 2013) donde mediante las señales

de sensores de flujo óptico se puede determinar la dirección de desplazamiento del robot. En trabajos más recientes se ha podido utilizar el flujo óptico para obtener la trayectoria (Granillo & Beltran, 2018) o velocidad de un robot (McGuire *et al.*, 2017). Sin embargo, en este proyecto dada la escasa información de parte del cálculo de flujo óptico, la estimación del estado del robot está limitada a una estimación cualitativa (estático, adelante, atrás, derecha, izquierda).

En cuanto a la evasión de obstáculos se pueden destacar las técnicas aplicadas por (Sanchez *et al.*, 2015) y (Back *et al.*, 2020) donde mediante la división de la imagen en dos partes (izquierda, derecha) y la representación de las magnitudes de los vectores de flujo óptico de dichas partes (ya sean la suma o el promedio de las mismas) se puede determinar la existencia de un obstáculo y se puede mandar la instrucción al robot para poder evadirlo.

Otro punto a destacar es que la mayoría de los trabajos en el estado del arte utilizan los algoritmos de Horn-Schunck (1981) o Lucas-Kanade (1981) ya sean los algoritmos tradicionales o modificaciones de los mismos.

Dado los diferentes propósitos de los trabajos (control de vuelo, distancia de objetos, control de aterrizaje, etc.), así como el uso de diferentes resoluciones, cámaras y dispositivos de procesamiento, no es posible comparar de manera cuantitativa los resultados entre los trabajos del estado del arte.

Por último, dada la problemática descrita y la revisión de los trabajos antecedentes y del estado del arte se plantea desarrollar un sistema de navegación robótica que mediante el uso de imágenes de baja resolución pueda realizar el proceso de navegación y evasión de obstáculos en un robot con una computadora con recursos limitados. Este sistema de navegación se evaluará en base al desempeño del robot en la evasión de obstáculos con los parámetros establecidos dentro del concepto de *tiempo real*.

Capítulo 2

Propuesta de solución

En este capítulo se muestran los objetivos, alcances, limitaciones, la descripción a detalle de la propuesta de solución y las métricas a utilizar.

2.1 Planteamiento de la solución

Dada la problemática planteada en el capítulo anterior (sección 1.2) se plantea desarrollar un sistema de navegación con imágenes de baja resolución con el fin de reducir el costo computacional para poder implementarlo en un robot con una computadora con recursos de procesamiento limitados. A continuación, se presentan los objetivos, alcances y limitaciones.

2.1.1 Objetivos

- **Objetivo General**

Programar un sistema de navegación robótica inspirado en la visión de insectos voladores capaz de evitar obstáculos en *tiempo real* en espacios reducidos, implementado con visión de baja resolución y técnicas de flujo óptico.

- **Objetivos específicos**

1. Estudiar las técnicas de aplicación de los métodos directos y flujo óptico.
2. Desarrollar un algoritmo que pueda evitar obstáculos en *tiempo real*.
3. Implementar el algoritmo en un robot móvil rodante.
4. Evaluar y mejorar, de ser necesario, el algoritmo propuesto

2.1.2 Alcances y limitaciones

Dentro de los alcances y limitaciones del sistema de navegación propuesto se encuentra que:

- El sistema de navegación deberá ejecutarse en un procesador o microcontrolador de capacidad *Raspberry Pi* o inferior.
- El sistema de navegación podrá usarse para controlar un robot móvil rodante.
- Los datos a procesar deberán ser entre $n \times 10^2$ o $n \times 10^3$ pixeles.

- El resultado del proceso deberá ser una estimación gruesa del movimiento en 2 dimensiones tanto de desplazamiento (adelante, atrás) como de giro (derecha, izquierda).
- Dado que los métodos de flujo óptico tienden a errar en grandes áreas sin textura (Wang *et al.*, 2020), las áreas de experimentación, así como los obstáculos deben presentar diversidad de textura o diferencias de colores.

2.2 Modelo conceptual de la solución

Como solución al problema planteado se propone un sistema de navegación robótica que mediante el uso de ventanas (fragmentos) de secuencias de imágenes de muy baja resolución pueda reducir el costo computacional del proceso del cálculo del flujo óptico, la detección del estado del robot y la detección de obstáculos. Para esta propuesta de solución se presenta la serie de pasos que se muestra en la figura 2.1. En esta sección se explica cada uno de estos pasos.

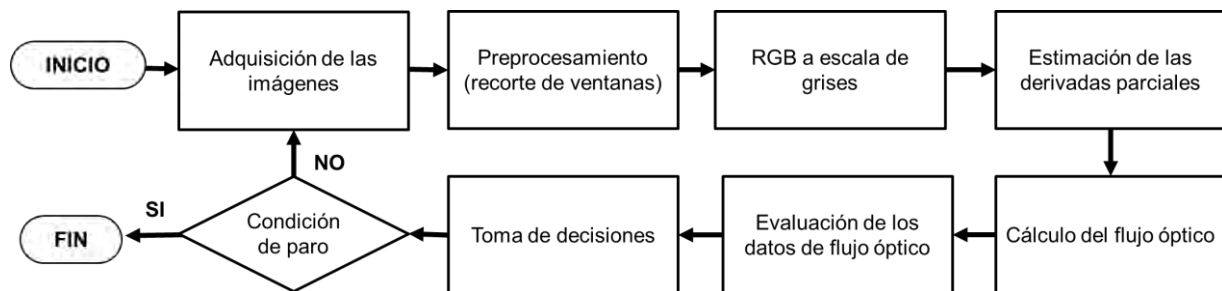


Figura 2.1 Serie de pasos de la propuesta de solución

2.2.1 Adquisición de las imágenes

Como primer paso, con el fin de reducir el costo computacional del sistema de navegación se propone usar una baja resolución de base. Dada la dependencia de los algoritmos de flujo óptico a la textura de los entornos y las limitaciones del hardware utilizado en este proyecto se propone el uso de las resoluciones de 160×128 y 80×64 .

En este paso también se propone capturar las imágenes de baja resolución en pares de fotogramas consecutivos en un periodo corto, esto permitirá que las intensidades de los píxeles de un fotograma tomado en un tiempo t puedan ser detectadas en un fotograma siguiente tomado en un tiempo $t + dt$, esto con el fin de poder calcular el desplazamiento de las intensidades de los píxeles (dx , dy) con los algoritmos de flujo óptico (figura 2.2). Para este propósito se propone adquirir las imágenes con un periodo de separación entre $1/30$ y $1/60$ de segundo.

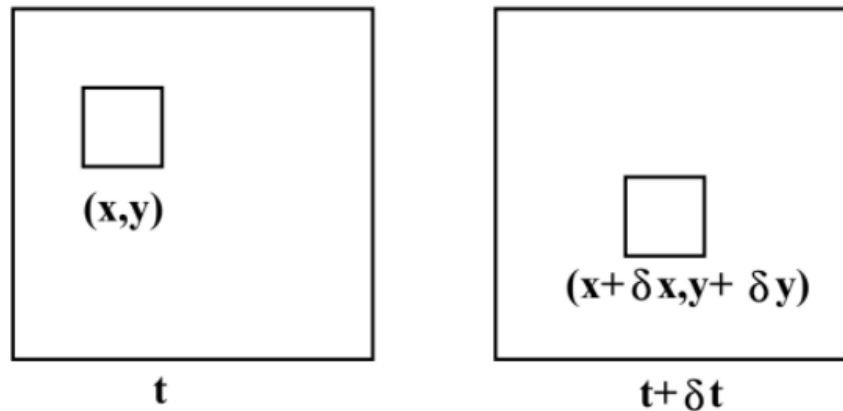


Figura 2.2 Cambio de posición de un pixel en diferentes imágenes separadas por un tiempo dt (Barron & Thacker, 2005)

2.2.2 Preprocesamiento

Otra medida para reducir el costo computacional durante el proceso de navegación es el uso de ventanas (fragmentos) de la imagen ubicadas en lugares estratégicos. Considerando que las ubicaciones de las ventanas están distribuidas de la siguiente manera: dos ventanas horizontales en los extremos laterales, y dos ventanas verticales en el extremo superior e inferior, se puede esperar que el sistema de navegación tenga la información necesaria para su funcionamiento. En base a la resolución mínima propuesta se escogieron los tamaños de ventana de 5×30 , 7×30 y 15×30 píxeles, dado que permite que las ventanas abarquen las áreas estratégicas de la imagen sin llegar a ocluirse entre sí (figura 2.3).

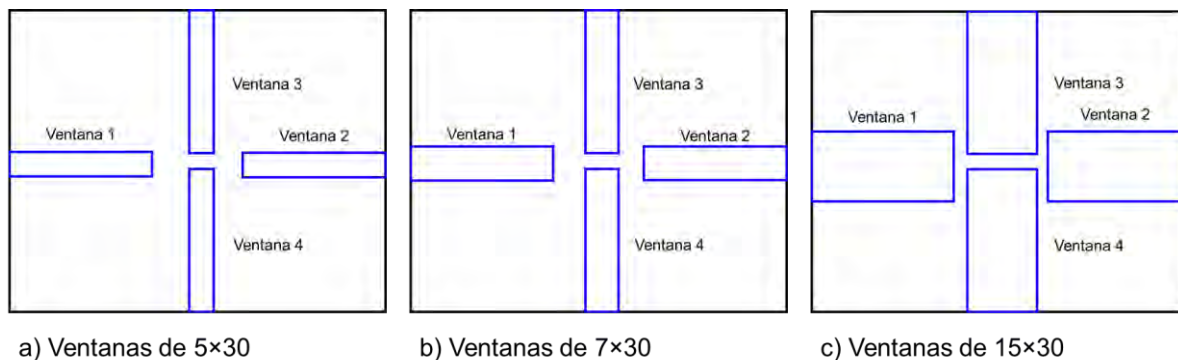


Figura 2.3 Tamaño y distribución de las ventanas sobre una imagen de 80×64 píxeles

2.2.3 Conversión de la imagen del sistema RGB a escala a grises

Dado que los algoritmos de flujo óptico implementados hacen uso de la imagen en escala de grises, se realiza el proceso de transformación de los valores de las intensidades de píxeles del sistema RGB (Rojo, Verde y Azul) a escala de grises. Esto se realiza sobre las ventanas obtenidas en el paso anterior.

Para cada pixel de una imagen en el espacio de color RGB la transformación a escala de grises se realiza mediante la siguiente formula¹:

$$\text{RGB a escala de grises: } Y \leftarrow 0.299 * R + 0.587 * G + 0.114 * B \quad (2.1)$$

Donde Y es la intensidad del pixel en escala de grises y R , G y B son las intensidades del pixel en rojo, verde y azul respectivamente.

2.2.4 Estimación de las derivadas parciales

Los algoritmos de flujo óptico de Horn-Schunck (1981) y de Lucas-Kanade (1981) son basados en técnicas diferenciales por lo que el flujo óptico se calcula a través de derivadas espacio-temporales de una secuencia de imágenes (de la Rosa, Guerrero, & Lenero, 2020). Por lo tanto antes del procesamiento del cálculo de flujo óptico se hace la estimación de las derivadas parciales de I_x , I_y e I_t (Horn & Schunck, 1981).

$$I_x \approx \frac{1}{4} \{ I_{i,j+1,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i+1,j,k} + I_{i,j+1,k+1} - I_{i,j,k+1} + I_{i+1,j+1,k+1} - I_{i+1,j,k+1} \} \quad (2.2)$$

$$I_y \approx \frac{1}{4} \{ I_{i+1,j,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i,j+1,k} + I_{i+1,j,k+1} - I_{i,j,k+1} + I_{i+1,j+1,k+1} - I_{i,j+1,k+1} \} \quad (2.3)$$

$$I_t \approx \frac{1}{4} \{ I_{i,j,k+1} - I_{i,j,k} + I_{i+1,j,k+1} - I_{i+1,j,k} + I_{i,j+1,k+1} - I_{i,j+1,k} + I_{i+1,j+1,k+1} - I_{i+1,j+1,k} \} \quad (2.4)$$

Donde I_x , I_y e I_t son las derivadas parciales con respecto x , y y t e I representan los valores de intensidad de la imagen original en escala de grises. Una mayor explicación de estas fórmulas se encuentra en la sección 1.1.1.4.

2.2.5 Cálculo del flujo óptico

Una vez teniendo las derivadas parciales, se procede al cálculo de flujo óptico mediante los algoritmos seleccionados. En el caso del algoritmo de Horn-Schunck (1981) el desplazamiento (u en el eje x y v en el eje y) de la intensidad de un pixel en un par de fotogramas consecutivos se calcula de la siguiente manera:

$$u^{n+1} = \bar{u}^n - I_x [I_x \bar{u}^n + I_y \bar{v}^n + I_t] / (\alpha^2 + I_x^2 + I_y^2) \quad (2.5)$$

$$v^{n+1} = \bar{v}^n - I_y [I_x \bar{u}^n + I_y \bar{v}^n + I_t] / (\alpha^2 + I_x^2 + I_y^2) \quad (2.6)$$

Donde u y v es el desplazamiento de la intensidad del pixel en los ejes x y y respectivamente, I_x , I_y e I_t son las derivadas parciales con respecto x , y y t , \bar{u} y \bar{v} son los promedios locales (este cálculo se encuentra en la sección 1.1.1.4) y por último se puede ver α^2 juega un papel importante sólo para las zonas donde el gradiente de luminosidad es pequeño, evitando resultados erróneos de la velocidad de flujo estimada ocasionados por el ruido en las derivadas estimadas (Horn & Schunck, 1981).

¹ Color conversions- OpenCV documentation
https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html#color_convert_rgb_gray

En cuanto al algoritmo de Lucas-Kanade (1981) el desplazamiento de la intensidad en un pixel en un par de fotogramas consecutivos se calcula de la siguiente manera:

$$u = \frac{-\sum I_{yi}^2 \sum I_{xi} I_{ti} + \sum I_{xi} I_{yi} \sum I_{yi} I_{ti}}{\sum I_{xi}^2 \sum I_{yi}^2 - (\sum I_{xi} I_{yi})^2} \quad (2.7)$$

$$v = \frac{\sum I_{xi} I_{ti} \sum I_{xi} I_{yi} - \sum I_{xi}^2 \sum I_{yi} I_{ti}}{\sum I_{xi}^2 \sum I_{yi}^2 - (\sum I_{xi} I_{yi})^2} \quad (2.8)$$

Donde u y v es el desplazamiento de la intensidad del pixel en los ejes x y y respectivamente e I_x, I_y, I_t son las derivadas parciales con respecto x, y y t del pixel central y n pixeles vecinos.

De los dos algoritmos se obtienen dos arreglos bidimensionales del tamaño de la ventana procesada con los valores de flujo óptico en u y v .

2.2.6 Procesamiento de los valores de flujo óptico obtenidos

Una vez teniendo los valores de flujo óptico se procede al procesamiento de los valores para poder estimar el estado del robot y si se presenta algún obstáculo frente a él.

2.2.6.1 Estimación del estado del robot

De acuerdo con la teoría de los algoritmos de flujo óptico, se pueden esperar los siguientes comportamientos de los valores de flujo óptico durante los desplazamientos y giros de un robot rodante (figura 2.4):

- Cuando el robot está en reposo (**sin movimiento**) los valores de flujo óptico en las componentes horizontal y vertical son cercanos a cero en las cuatro ventanas.
- Cuando el robot está haciendo un desplazamiento hacia adelante (**acercamiento**) los valores de flujo óptico del eje horizontal en la ventana 2 y la componente vertical de la ventana 4 son positivos, mientras que los valores del eje horizontal de la ventana 1 y la componente vertical de la ventana 3 son negativos.
- Cuando el robot está haciendo un desplazamiento hacia atrás (**alejamiento**) los valores de flujo óptico del eje horizontal en la ventana 2 y la componente vertical de la ventana 4 tienden a ser negativos, mientras que los valores del eje horizontal de la ventana 1 y la componente vertical de la ventana 3 tienden a ser positivos.
- Cuando el robot hace un giro hacia la **derecha** los valores de flujo óptico en el eje horizontal tienden a ser negativos en las cuatro ventanas.
- Cuando el robot hace un giro hacia la **izquierda** los valores de flujo óptico en el eje horizontal tienden a ser positivos en las cuatro ventanas.

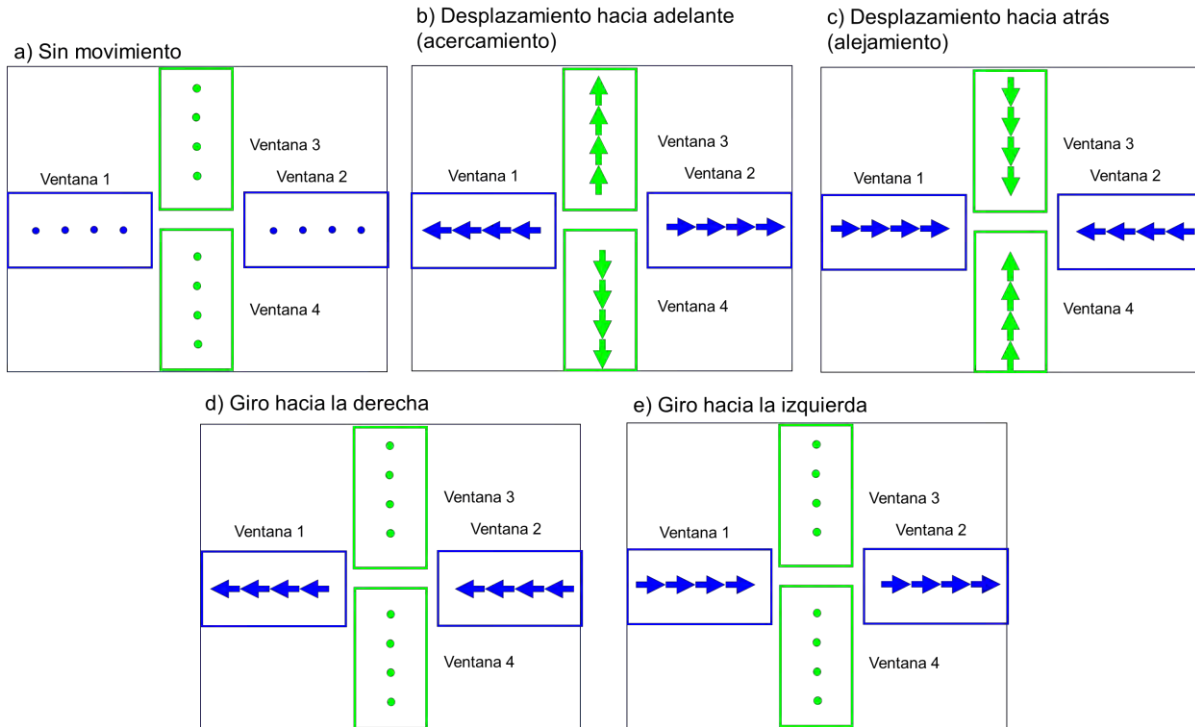


Figura 2.4 Comportamiento esperado en el eje horizontal (azul) y de la componente vertical (verde) de flujo óptico ante los movimientos de desplazamiento y giro. Cabe señalar que las representaciones de los valores fueron ampliadas con fines de demostración.

A pesar de que se cumplan los comportamientos esperados, los valores de flujo óptico pueden variar por diferentes factores como la presencia de obstáculos o diferentes condiciones del entorno (de textura, de luminosidad o de área). Por lo tanto, se propuso realizar pruebas de algoritmos de clasificación con los valores de flujo óptico obtenidas de las cuatro ventanas. Para este proyecto se escogieron tres algoritmos de clasificación basados en árboles de decisión: *CART*, *Adaboost* y *RandomForest* (en el anexo A se describen a detalle estos algoritmos). Con el uso de estos algoritmos se espera que, con los datos de flujo óptico, los algoritmos puedan generar un modelo en base a un conjunto de reglas que permita estimar el estado del robot (sin movimiento, adelante, atrás, derecha, izquierda) en diferentes circunstancias y no solo cuando los valores de comporten de manera idónea.

Para los algoritmos de clasificación seleccionados se propone el uso de un vector de características compuesta de los promedios de los valores de flujo óptico en las componentes horizontal y vertical de cada ventana (ecuaciones 2.9 y 2.10).

$$u_{promedio} = \frac{\sum_{i=0}^h \sum_{j=0}^w u[i, j]}{h * w} \tag{2.9}$$

$$v_{promedio} = \frac{\sum_{i=0}^h \sum_{j=0}^w v[i, j]}{h * w} \tag{2.10}$$

Donde $u_{promedio}$ y $v_{promedio}$ son los promedios de los valores de flujo óptico en la eje horizontal y vertical respectivamente, $u[i, j]$ y $v[i, j]$ representan el desplazamiento de la intensidad del pixel ubicado en la posición $[i, j]$; por último, h y w representan las dimensiones (alto, ancho) de la ventana. Estos promedios fueron representados como un vector de características conformado de ocho valores junto con la etiqueta del estado real del robot (figura 2.5).

u1	v1	u2	v2	u3	v3	u4	v4	Estado real
----	----	----	----	----	----	----	----	-------------

Figura 2.5 Vector de características con los promedios de los valores de flujo óptico de las cuatro ventanas para las pruebas con los clasificadores.

2.2.6.2 Evasión de obstáculos

Para la evasión de obstáculos se propone un método basado en los trabajos de Sanchez *et al.* (2015) y Back *et al.* (2020) el cual consiste en obtener un promedio de las magnitudes de los vectores de flujo óptico. La estrategia utilizada es que cuando el robot está avanzando, el objeto que está más cerca proporciona un movimiento rápido que el objeto más lejano, por lo tanto, las magnitudes de flujo óptico de la ventana (izquierda, derecha, superior o inferior) que presente un obstáculo serán mayores (figura 2.6).

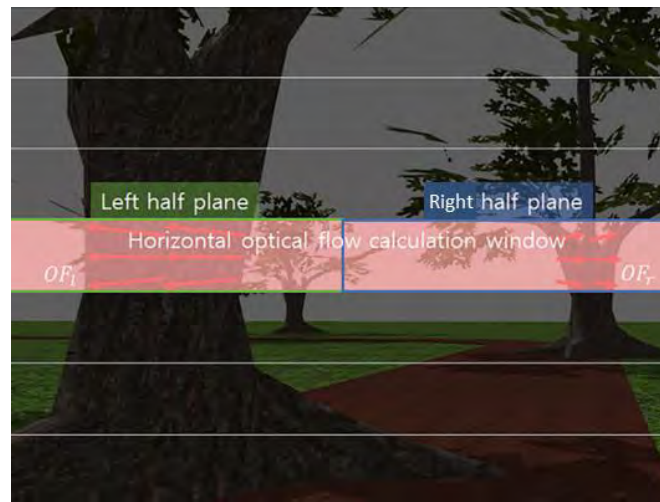


Figura 2.6 Comportamiento de las magnitudes de flujo óptico ante la presencia de obstáculos a diferentes distancias (Back *et al.*, 2020)

En los trabajos de Back *et al.* (2020), Dai & Li (2015), Sanchez *et al.*, (2015), Souhila & Karim (2007) y Yadav & Khandekar (2018) se divide el plano de la imagen en dos o más partes iguales, en este caso se utilizan los promedios de las magnitudes de cada una de las ventanas para determinar la presencia de obstáculos en el plano de la imagen. Por lo tanto, para determinar si existe disparidad entre las ventanas izquierda (ventana 1) y derecha (ventana 2) o superior (ventana 3) e inferior (ventana 4) se utilizan las siguientes fórmulas de balance:

$$e_{balance\ horizontal} = |OF_{ventana1} - OF_{ventana2}| \quad (2.11)$$

$$e_{balance\ vertical} = |OF_{ventana3} - OF_{ventana4}| \quad (2.12)$$

Donde e representa el balance entre dos ventanas y OF representa el promedio de las magnitudes de cada ventana el cual es calculado de la siguiente manera:

$$OF_{ventana} = \frac{\sum_{i=0}^h \sum_{j=0}^w \sqrt{u[i,j]^2 + v[i,j]^2}}{h * w} \quad (2.13)$$

Donde $u[i,j]$ y $v[i,j]$ representan el desplazamiento de la intensidad del pixel ubicado en la posición $[i,j]$, por último h y w representan las dimensiones (alto, ancho) de la ventana.

El funcionamiento de estas fórmulas es que, cuando se llegue a presentar un obstáculo en alguna de las ventanas se produzca que el resultado de una o las dos fórmulas de balance superen un umbral y el sistema determine que hay un obstáculo. Mientras que la fórmula de balance horizontal puede advertir de un obstáculo en los laterales de la imagen, la fórmula de balance vertical puede advertir de obstáculos de frente, en el caso de que se presente en la ventana superior y, en ciertos casos se puede advertir de colisiones con paredes donde hubiera un cambio drástico de textura entre el piso y la pared.

2.2.7 Toma de decisiones

En base al procesamiento de los valores obtenidos de los algoritmos de flujo óptico mostrados en la sección anterior, se pueden proponer dos formas en que el robot pueda tomar decisiones durante el proceso de navegación:

- **Decisión para evadir obstáculos:** durante el recorrido del robot el sistema de navegación se irá guiando de las fórmulas de balance de las ventanas, en caso de que se supere el umbral en alguna de las dos fórmulas, se considerará que existe un obstáculo en el plano de la imagen por lo que se procederá a tomar la decisión que permita que el robot pueda evadir el obstáculo.
- **Decisión en caso de error en clasificación:** a través de la clasificación de los valores de flujo óptico se puede saber si el robot está ejecutando la acción indicada por el sistema de navegación. En caso de que el clasificador indique un estado diferente al instruido, se considera que el robot está en un estado en el que no puede llevar a cabo la instrucción (atorado por alguna causa externa) por lo que se le darán ciertas instrucciones las cuales tendrán prioridad sobre las decisiones del detector de obstáculos, esto con el fin de que el robot pueda salir de la situación en la que se encuentra y pueda volver a cumplir con las instrucciones dadas por el sistema.

2.3 Criterios de control

En esta sección se presentan las métricas utilizadas la evaluación de los algoritmos de flujo óptico, los algoritmos de clasificación y la técnica de evasión de obstáculos utilizados en este proyecto.

2.3.1 Métrica para la evaluación de los algoritmos de flujo óptico

Las medidas de evaluación tienen como objetivo revelar las propiedades de los algoritmos de flujo óptico y ayudan a comprender mejor los puntos fuertes y débiles relativos de cada uno. No sólo proporcionan información para mejorar los algoritmos, sino que también permiten seleccionar los enfoques más adecuados para hacer frente a retos especiales, como el tratamiento del ruido, los grandes desplazamientos o los límites de movimiento. La medición del rendimiento de los algoritmos de flujo óptico sigue siendo objeto de debate científico, y ha llevado a la introducción de diversas medidas de evaluación (Tu *et al.*, 2019).

La métrica seleccionada para medir el desempeño de los algoritmos de flujo óptico es el error medio del punto final (*average endpoint error*, AEE por sus siglas en inglés), mide el error absoluto entre el flujo óptico estimado y el *ground truth* (flujo óptico verdadero) (Huang & Song, 2017). La fórmula para calcular el AEE es:

$$AEE = \frac{1}{N} \sum_{i=1}^N \sqrt{(u_i - u_{itruth})^2 + (v_i - v_{itruth})^2} \quad (2.14)$$

Donde (u_i, v_i) es el flujo óptico estimado por el algoritmo de flujo óptico, (u_{itruth}, v_{itruth}) es el flujo óptico verdadero y N es el número total de píxeles de la imagen.

2.3.2 Métricas para clasificación (Barrios Arce, 2019)

Matriz de confusión

En el campo de la inteligencia artificial y el aprendizaje automático una matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real, (figura 2.7).

VALORES PREDICCIÓN	Verdaderos positivos	Falsos Positivos
	Falsos Negativos	Verdaderos Negativos
	VALORES REALES	

Figura 2.7 Matriz de confusión binaria

En el caso de la figura 2.7 se habla de una clasificación binaria por lo que existen cuatro posibilidades de resultados de parte del clasificador. Estos son los cuatro resultados posibles:

- **Verdadero positivo:** El valor real es positivo y el clasificador predijo también que el resultado es positivo.
- **Verdadero negativo:** El valor real es negativo y el clasificador predijo también que el resultado era negativo.
- **Falso negativo:** El valor real es positivo, y el clasificador predijo que el resultado es negativo. Esto es lo que en estadística se conoce como error tipo II
- **Falso positivo:** El valor real es negativo, y el clasificador predijo que el resultado es positivo. Esto es lo que en estadística se conoce como error tipo I

A partir de estas cuatro opciones surgen las métricas de la matriz de confusión: *Accuracy* (Exactitud), *Precision* (Precisión), *Recall* (Sensibilidad) y *F1 Score*.

Accuracy

La Exactitud (en inglés, *accuracy*) se refiere a lo cerca que está el resultado de una medición del valor verdadero. En términos estadísticos, la exactitud está relacionada con el sesgo de una estimación. Se representa como la proporción de resultados verdaderos dividido entre el número total de casos examinados. En forma práctica, la exactitud es la cantidad de predicciones que fueron correctas.

$$Accuracy = (VP + VN)/(VP + FP + FN + VN) \quad (2.15)$$

Donde *VP* es la cantidad de verdaderos positivos, *FP* los falsos positivos, *VN* los verdaderos negativos y *FN* falsos negativos.

Precision

La Precisión (en inglés *Precision*) se refiere a la dispersión del conjunto de valores obtenidos a partir de mediciones repetidas de una magnitud. Cuanto menor es la dispersión mayor la precisión. Se representa por la proporción de verdaderos positivos dividido entre todos los resultados positivos (tanto verdaderos positivos, como falsos positivos). En forma práctica es el porcentaje de casos positivos detectados. Se calcula como:

$$Precision = VP/(VP + FP) \quad (2.16)$$

Donde *VP* es la cantidad de verdaderos positivos y *FP* los falsos positivos.

Recall

La Sensibilidad (en inglés *Recall* o *Sensitivity*) también se conoce como Tasa de Verdaderos Positivos (*True Positive Rate*) ó TP. Es la proporción de casos positivos que fueron correctamente identificadas por el algoritmo.

$$Recall = VP/(VP + FN) \quad (2.17)$$

Donde *VP* es la cantidad de verdaderos positivos y *FN* los falsos negativos.

F1 Score

F1 Score es la media ponderada de *Precision* y *Recall*. Por lo tanto, esta puntuación tiene en cuenta tanto los falsos positivos como los falsos negativos. Intuitivamente no es tan fácil

de entender como la exactitud, pero suele ser más útil que el *Accuracy*, especialmente si se tiene una distribución de clases desigual. Se calcula:

$$F1\ Score = \frac{2 * (Recall * Precision)}{(Recall + Precision)} \quad (2.18)$$

2.3.3 Métrica para evaluación para la técnica de evasión de obstáculos

Uno de los principales objetivos del sistema de navegación propuesto es la evasión de obstáculos por lo que es necesario medir su eficacia en base a los obstáculos detectados a los obstáculos presentes en el campo de visión del robot.

$$Eficacia\ en\ evasión\ de\ obstáculos = \frac{Obstáculos\ detectados}{Obstáculos\ presentes\ en\ el\ campo\ de\ visión} \quad (2.19)$$

2.4 Conclusiones

Con la propuesta de solución planteada se obtiene que mediante el flujo óptico calculado sobre ventanas de imágenes de baja resolución un robot rodante pueda navegar y evadir obstáculos en *tiempo real* en una computadora con recursos limitados.

En teoría, con los valores de flujo óptico es posible determinar el estado del robot (adelante, atrás, derecha, izquierda o sin movimiento) por lo que se escogieron algoritmos de clasificación basados en arboles de decisión para que generaran el conjunto de reglas necesarias para determinar el estado del robot en base al flujo óptico obtenido de las cuatro ventanas de la imagen.

En base a los trabajos presentados en el estado del arte se presenta una estrategia para la evasión de obstáculos mediante el promedio de las magnitudes de los vectores de flujo óptico sobre las cuatro ventanas, dado que cuando existe un obstáculo sobre una de las ventanas, las magnitudes de flujo óptico se incrementan sobre las demás.

Con los datos obtenidos de los algoritmos de clasificación así con las fórmulas de balance las magnitudes de flujo óptico se obtiene la información necesaria para que el sistema de navegación implementado funcione correctamente.

Capítulo 3

Análisis, diseño e implementación del sistema

En este capítulo se presenta la implementación del sistema de navegación robótica, también se describe el ambiente de desarrollo que incluye el lenguaje de programación y librerías utilizadas.

3.1 Entorno de desarrollo

El sistema fue desarrollado en el lenguaje de programación Python en su versión 3.7.3, el sistema operativo *Raspbian* (actualmente llamando *Raspberry Pi OS*) en su versión 10 que es una distribución de *Linux* especial para la *Raspberry Pi*.

Para la visualización de la pantalla de la *Raspberry Pi* de manera inalámbrica se hizo uso del servicio VNC (*Virtual Network Computing*)² instalado de manera nativa en el sistema operativo.

3.1.1 Librerías utilizadas

Para el proceso de obtener los fotogramas desde el módulo de cámara de la *Raspberry Pi* se utilizó la librería *picamera*³ en su versión 1.13, para el procesamiento de los fotogramas para la transformación a escala de grises y su visualización se utilizó la librería *OpenCV*⁴ en su versión 3.2.0 y para el tratamiento de las ventanas de la imagen como matrices y el

² "VNC Connect for Raspberry Pi." <https://www.realvnc.com/en/raspberrypi/>.

³ "picamera — Picamera 1.13 Documentation." <https://picamera.readthedocs.io/en/release-1.13/>.

⁴ "OpenCV: OpenCV modules." <https://docs.opencv.org/3.2.0/>.

cálculo del flujo óptico se utilizaron las librerías *scipy*⁵ y *numpy*⁶ en sus versiones 1.1.0 y 1.21.6 respectivamente.

Para el proceso de la estimación del estado del robot mediante los algoritmos de clasificación CART, *Adaboost* y *RandomForest* se utilizó la librería *sklearn*⁷ en su versión 0.0.post1 y para cargar el modelo entrenado y realizar el proceso de clasificación del estado del robot se utilizó la librería *joblib*⁸ en su versión 1.2.0

Para la integración de la interfaz gráfica (sección 3.3) se hizo uso de la librería *tkinter*⁹ en su versión 8.6.

Para la comunicación de la *Raspberry* con el robot rodante mediante conexión de los pines GPIO se utilizó la librería *RPi.GPIO*¹⁰ en su versión 0.7.0.

3.2 Diseño del sistema de navegación robótica

La implementación del sistema de navegación que estima la posición del robot y evade obstáculos está conformada por una serie de procesos, rutinas y condiciones de control como se muestra en la figura 3.1.

Dado que el robot no cuenta con sensores que permitan tener su ubicación como un giroscopio o una IMU (Unidad de Medición Inercial) se agregó una pequeña rutina de seguimiento de luz guía para poder indicar un punto de inicio y un punto final en algunas experimentaciones. Esta rutina solo es auxiliar al sistema de navegación.

3.2.1 Parámetros iniciales

Para el funcionamiento del sistema de navegación se requieren los siguientes parámetros iniciales:

- **Método de cálculo de flujo óptico:** El sistema solo trabaja con un algoritmo el cual puede ser el de Horn-Schunck o Lucas-Kanade.
- **Tamaño de ventana de convolución** (para algoritmo de Lucas-Kanade).
- **Número de iteraciones** (para algoritmo de Horn-Schunck).
- **Resolución de los fotogramas tomados del módulo de cámara** (altura, anchura).
- **Tamaño de las ventanas** (altura, anchura).
- **Cantidad de fotogramas por segundo:** Para este proyecto se experimentó con 30 y 60 fps.
- **Umbral de balance:** Sirve para delimitar el umbral para el balance entre las magnitudes de las cuatro ventanas, es utilizado en la detección de obstáculos.
- **Límite de estados de error:** este límite sirve para la rutina del estado de error y para detener el proceso de navegación cuando este límite sea superado

⁵ “Scipy documentation.” <https://docs.scipy.org/doc/scipy/>.

⁶ “Numpy documentation.” <https://numpy.org/doc/stable/>.

⁷ “Sklearn 0.0.post1.” <https://pypi.org/project/sklearn/>.

⁸ “Joblib documentation.” <https://joblib.readthedocs.io/en/latest/>.

⁹ “Tkinter documentation.” <https://docs.python.org/es/3/library/tkinter.html>.

¹⁰ “Rpi.GPIO.” <https://pypi.org/project/RPi.GPIO/>.

- **Límite de estados de evasión de obstáculo:** este límite sirve para la rutina de la detección de obstáculos y delimita la cantidad de veces que una instrucción de giro (izquierda, derecha) o de desplazamientos hacia atrás será realizada por el robot en caso de que se detecte un obstáculo.
- **Límite de estados de *perdido*:** este límite sirve para la rutina de seguimiento de luz guía en caso de que el robot no pueda encontrar la luz guía para tomar una decisión y sirve para detener el proceso de navegación cuando este límite es superado.

Estos parámetros iniciales son ingresados por el usuario en la interfaz que se menciona en la sección 3.3.

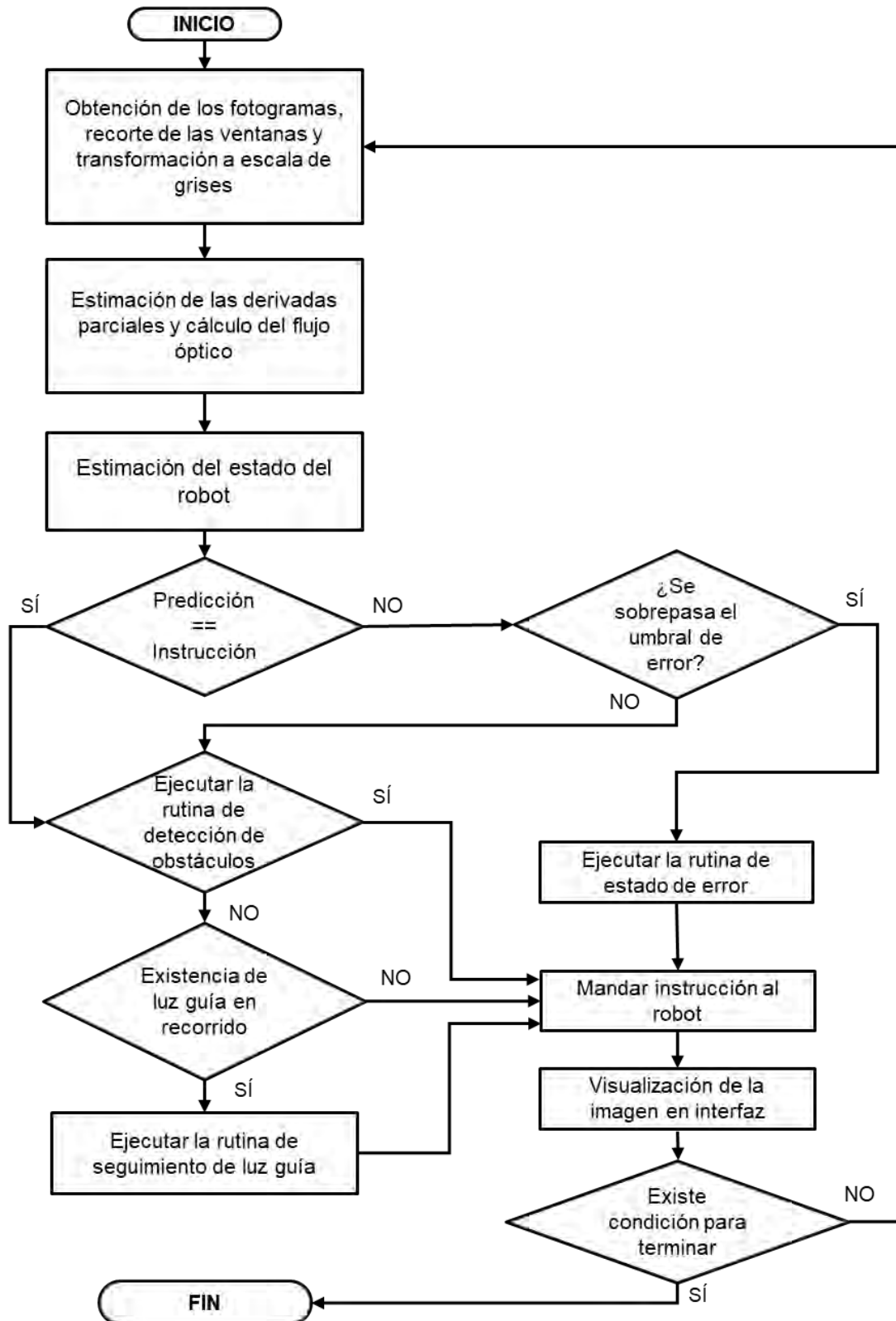


Figura 3.1 Procesos, rutinas y condiciones de control del sistema de navegación robótica propuesto

3.2.2 Descripción de los procesos y rutinas del sistema

3.2.2.1 Obtención de los fotogramas y recorte de las ventanas

Los pares de fotogramas consecutivos son obtenidos del módulo de cámara a través del método *capture_continuous* de la librería *picamera* que controla de manera nativa la captura de los fotogramas lo que asegura la velocidad de fotogramas por segundo delimitada en los parámetros iniciales.

Los fotogramas obtenidos de la librería *picamera* son en el espacio de color RGB (Rojo, Verde y Azul), por lo que posteriormente del recorte de las ventanas se hace la transformación de las ventanas a escala de grises mediante el método *cvtColor* de la librería *OpenCV* (figura 3.2). Posteriormente, con el método *astype* de la librería *numpy* se hace la transformación de las ventanas de valores de tipo *uint8* (enteros sin signo de 8 bits) a valores de tipo *float* para el proceso del cálculo del flujo óptico.

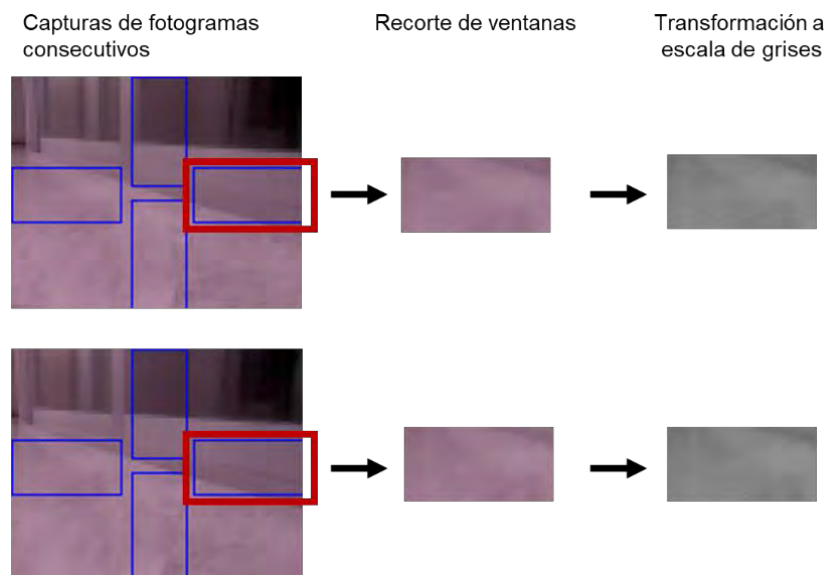


Figura 3.2 Proceso de captura de los fotogramas, recorte de ventanas y transformación escala de grises

Para el proceso de la rutina de seguimiento de luz guía se guarda una copia de uno de los fotogramas capturados y es transformado al sistema de color HSV (Matiz, Saturación, Valor, por sus siglas en inglés) con el mismo método *cvtColor* de la librería *OpenCV*, posteriormente esta imagen es binarizada con el método *inRange* de la librería *OpenCV* con los valores mínimos de (64, 85, 145) y valores máximos de (113, 255, 255) para poder identificar la luz guía de color verde.

3.2.2.2 Cálculo del flujo óptico

Una vez que se tienen las ventanas del par de fotogramas en escala de grises con valores de tipo *float*, se procede a calcular las derivadas parciales mediante la convolución de las ventanas de los dos fotogramas capturados con las máscaras mostradas en la sección 1.1.1.4 (figura 1.2). Con esto se obtienen tres matrices con los valores de las derivadas parciales con respecto a x , y y t , para cada una de las ventanas. Posteriormente, con los

valores obtenidos de las derivadas parciales se realiza el cálculo del flujo óptico para cada ventana. Con este cálculo se obtienen dos matrices con los valores de flujo óptico en los componentes $x(u)$ y $y(v)$.

3.2.2.3 Estimación del estado del robot

Una vez teniendo los valores del cálculo del flujo óptico se procede a crear los vectores de características (figura 2.5) mediante el promedio de los valores de flujo óptico en u y v con las ecuaciones 2.9 y 2.10. En este caso, como es para clasificación el vector no contiene la etiqueta del estado real.

Previamente se debe de contar con un modelo entrenado por alguno de los algoritmos de clasificación utilizados en este proyecto (*CART*, *Adaboost* y *RandomForest*) los cuales fueron implementados con la librería *sklearn*.

Este vector generado es introducido al método *test* de la librería *joblib* al modelo entrenado para obtener como resultado la predicción del estado del robot. Este resultado es un valor numérico entre 1 y 5, la representación de este resultado se muestra en la tabla 3.1.

Tabla 3.1 Representación del resultado obtenido de la clasificación

Resultado del clasificador	Estado detectado
1	Sin movimiento
2	Adelante
3	Atrás
4	Derecha
5	Izquierda

3.2.2.4 Rutina de estado de error

Esta rutina es la primera en ejecutarse, posterior a obtener el resultado obtenido del clasificador dado que se pueden presentar errores de clasificación por motivos que el robot esté en una situación en la que no puede llevar a cabo la instrucción dada por el mismo sistema (en caso de que el robot se encuentre atorado. Sin embargo, también se pueden presentar errores ocasionales de parte del clasificador por causa de los valores de flujo óptico calculados o por factores externos del robot, por esta situación se desarrolló esta rutina donde mediante el límite de estado de error (parámetro inicial descrita en la sección 3.2.1) se mandan instrucciones al robot a fin de que de manera autónoma pueda volver a un estado en el que pueda realizar las instrucciones dadas por el sistema.

Se recomienda que el límite de estado de error sea múltiplo de cinco, dado que esta rutina está dividida en cinco partes donde cada vez que se supere una quinta parte del límite del error se cambia la instrucción. Adicionalmente, también se consideró la velocidad de procesamiento de cada par de fotogramas para determinar este umbral, para que el robot tuviera el tiempo suficiente para realizar las acciones deseadas. Esta rutina es desglosada en el algoritmo 3.1.

Algoritmo 3.1 Rutina de estado de error

Datos: Contador de estado de error y límite del estado de error (parámetro inicial).

Resultado: Instrucción para el robot y estado de control de error

Inicio

1. **SI** el estado del robot no coincide con la respuesta obtenida de la clasificación:
 - 1.1 Se aumenta en 1 el contador de estado de error.
 - 1.2. **SI** el contador de error es mayor o igual a $1/5$ del límite de error:
 - 1.2.1 Se activa el estado de error.
 - 1.2.2 **SI** el contador de error es mayor o igual a $1/5$ y menor a $2/5$ del límite de error:
 - 1.2.2.1 Se envía la instrucción para que el robot vaya para atrás.
 - 1.2.3 **SI** el contador de error es mayor o igual a $2/5$ y menor a $3/5$ del límite de error:
 - 1.2.3.1 Se envía la instrucción para que el robot vaya hacia la derecha.
 - 1.2.4 **SI** el contador de error es mayor o igual a $3/5$ y menor a $4/5$ del límite de error:
 - 1.2.4.1 Se envía la instrucción para que el robot vaya hacia la izquierda.
 - 1.2.5 **SI** el contador de error es mayor o igual a $4/5$ del límite de error y menor al límite de error:
 - 1.2.5.1 Se envía la instrucción para que el robot vaya hacia adelante.
 - 1.2.6 **SI** el contador de error es igual al límite de error:
 - 1.2.6.1 Se envía la instrucción para que el robot se detenga.
 - 1.3 **SI NO:**
 - 1.3.1 Se obtiene instrucción de parte de la rutina de detección de obstáculos o de la rutina de seguimiento de luz guía.
 2. **SI NO:**
 - 2.1 Se desactiva el estado de error.
 - 2.2 Se reinicia contador de estado de error a 0.
 - 2.3 Se obtiene instrucción de parte de la rutina de detección de obstáculos o de la rutina de seguimiento de luz guía.

Fin

3.2.2.5 Rutina de detección de obstáculos

Esta segunda rutina se ejecuta cuando el estado del robot, obtenido del clasificador, coincide con el estado real del robot. En esta rutina se hacen uso de las ecuaciones 2.11 y 2.12 donde a base del balance de las ventanas se detecta si existe algún obstáculo frente al robot, también es requerido un umbral que delimitará qué tan sensible será el balance en las cuatro ventanas. Si el umbral es demasiado bajo, el sistema tomará como obstáculos los pequeños cambios de textura, por el contrario, un umbral demasiado alto haría que el robot no detecte los obstáculos con textura semejantes al entorno provocando colisiones. Esta rutina se desglosa en el algoritmo 3.2.

Algoritmo 3.2 Rutina de detección de obstáculos

Datos: Promedio de las magnitudes de las cuatro ventanas y umbral de balance

Resultado: Instrucción para el robot y estado de control de obstáculo

Inicio

1. **SI** el resultado de la ecuación 2.11 es mayor al umbral:
 - 1.1 Se activa el estado de obstáculo.
 - 1.2 **SI** el resultado de la ecuación 2.12 es mayor al umbral:
 - 1.2.1 Se envía la instrucción para que el robot vaya para atrás (Obstáculo de frente).
 - 1.3. **SI NO:**
 - 1.3.1 **SI** el promedio de la ventana 1 es mayor o igual al promedio de la ventana 2:
 - 1.3.1.1 Se envía la instrucción para que el robot vaya hacia la derecha (Obstáculo a la izquierda).
 - 1.3.2 **SÍ** el promedio de la ventana 2 es mayor o igual al promedio de la ventana 2:
 - 1.3.2.1 Se envía la instrucción para que el robot vaya hacia la izquierda (Obstáculo a la derecha).
2. **SI NO:**
 - 2.1 **SI** el resultado de la ecuación 2.12 es mayor al umbral:
 - 2.1.1 Se activa el estado de obstáculo.
 - 2.1.2 **SI** el promedio de la ventana 4 es mayor o igual al promedio de la ventana 3:
 - 2.1.2.1 Se envía la instrucción para que el robot vaya para atrás (Obstáculo de frente).
 - 2.1.3 **SI** el promedio de la ventana 1 es mayor o igual al promedio de la ventana 2:
 - 2.1.3.1 Se envía la instrucción para que el robot vaya hacia la derecha (Obstáculo a la izquierda).
 - 2.1.4 **SI** el promedio de la ventana 2 es mayor o igual al promedio de la ventana 2:
 - 2.1.4.1 Se envía la instrucción para que el robot vaya hacia la izquierda (Obstáculo a la derecha).

Fin

3.2.2.6 Rutina de seguimiento de luz guía

Como se mencionó al inicio de esta sección se desarrolló un sistema auxiliar de búsqueda de una luz guía. En este caso se hace uso de la imagen binaria mencionada en la sección 3.2.2.1 para guiar al robot hacia un punto final durante las experimentaciones. Esta imagen es dividida en dos partes en una proporción aproximada de 62% para la parte superior y 38% para la parte inferior (figura 3.3).

Para saber si esta la luz guía dentro del campo de visión se cuenta el número de píxeles que no sean 0 mediante el método `count_nonzero` de la librería `numpy`.

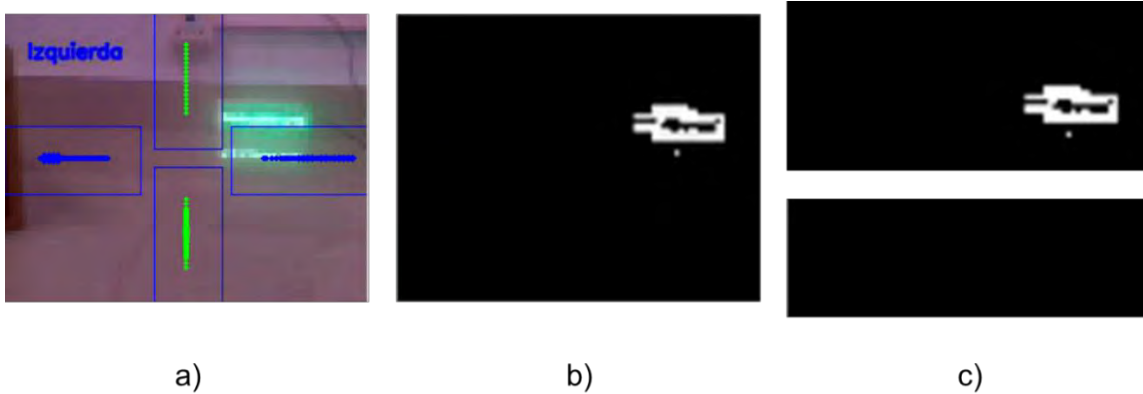


Figura 3.3. Procesamiento de la imagen para la rutina de seguimiento de luz guía: a) Imagen obtenida de la cámara para el sistema de navegación, b) Imagen binarizada con los umbrales HSV, c) Imagen dividida horizontalmente en proporción 62% y 38%.

En esta rutina se utiliza el límite de estado de *perdido* para cuando el robot no encuentre la luz guía. Se recomienda que el límite de estado de *perdido* sea múltiplo de tres dado que esta rutina está dividida en tres partes donde cada vez que se supere $1/3$ del límite del estado de *perdido* se cambia de instrucción. Adicionalmente también se consideró la velocidad de procesamiento de cada par de fotogramas para determinar este umbral, para que el robot tuviera el tiempo suficiente para realizar las acciones deseadas. Esta rutina se desglosa en el algoritmo 3.3.

Algoritmo 3.3 Rutina de seguimiento de luz guía

Datos: Imagen binaria, contador de estado de *perdido* y límite de estado de *perdido*

Resultado: Instrucción para el robot y estado de *perdido*

Inicio

1. Se divide la imagen de manera horizontal
 2. Se cuentan el número de pixeles que no tengan valores de 0 en las dos partes de la imagen binaria
 3. **SI** el contador de pixeles de alguna de las dos partes de la imagen es mayor a 0:
 - 3.1 Se desactiva el estado de *perdido*
 - 3.2 Se reinicia el contador de estado de *perdido* a 0.
 - 3.3 **SI** el contado de la parte superior de la imagen es mayor al contador de la parte inferior:
 - 3.3.1 Se manda la instrucción de ir hacia adelante.
 - 3.4 **SI** el contado de la parte inferior de la imagen es mayor al contador de la parte superior:
 - 3.4.1 Se manda la instrucción de detener al robot.
 4. **SI NO:**
 - 4.1 Se aumenta en uno el contador de estado de *perdido*
 - 4.2 **SI** el contador de estado de *perdido* es igual o mayor a $1/3$ y menor a $2/3$ del límite de estado de *perdido*:
 - 4.2.1 Se envía la instrucción para que el robot vaya para derecha.
-

Algoritmo 3.3 Rutina de seguimiento de luz guía (Continuación)

4.3 SI el contador de estado de *perdido* es igual o mayor a $2/3$ del y menor del límite de estado de *perdido*:

4.3.1 Se envía la instrucción para que el robot vaya a la izquierda.

4.4 SI el contador de estado de *perdido* es igual al límite de estado de *perdido*:

4.4.1 Se envía la instrucción para que el robot se detenga

Fin

3.2.2.7 Condiciones para terminar el recorrido

Para que el robot pudiera detener su recorrido de manera automática ya sea que haya llegado a un punto destino mediante la rutina de seguimiento de luz guía o haya superado el límite de estados de error de la rutina de estado de error, se hace la comparativa a si alguna de las dos rutinas envía la instrucción de detenerse. Si se presenta esta condición se detiene el robot y se da por terminado el recorrido.

3.2.3 Estados de control

Dado que existen tres rutinas las cuales pueden mandar instrucciones al robot rodante, se hace uso de variables booleanas de control que permiten jerarquizar las rutinas y decidir qué rutina puede enviar instrucciones al robot móvil.

- **Estado de error:** Cuando se activa este estado desde la rutina de estado de error, la instrucción dada tiene prioridad sobre las rutinas de detección de obstáculos y búsqueda de luz guía.
- **Estado de obstáculo:** Cuando se activa este estado desde la rutina de detección de obstáculos, la instrucción dada tiene prioridad sobre la rutina de búsqueda de luz guía.
- **Estado de búsqueda de luz guía:** Cuando se activa este estado desde la rutina de búsqueda de luz guía, la instrucción dada tiene prioridad sobre la rutina de detección de obstáculos, dado que esta rutina se lleva a cabo cuando la rutina del detector de obstáculos determina que el camino está libre.

3.3 Interfaz del sistema

Para el proceso de experimentación se hizo uso de una interfaz sencilla conformado por una ventana del tamaño de la imagen escalada (sección 3.2.2.1). Dentro de la ventana de la imagen se muestran en color azul las ventanas 1 y 2 mientras que las ventanas 3 y 4 se muestran de color verde.

En el lado superior izquierda de la ventana se pueden ver tres textos los cuales representan mensajes para el usuario con respecto a la respuesta recibida del clasificador, la presencia de obstáculos y el mensaje de error en caso de que el robot entre dentro de la rutina de estado de error (figura 3.4).

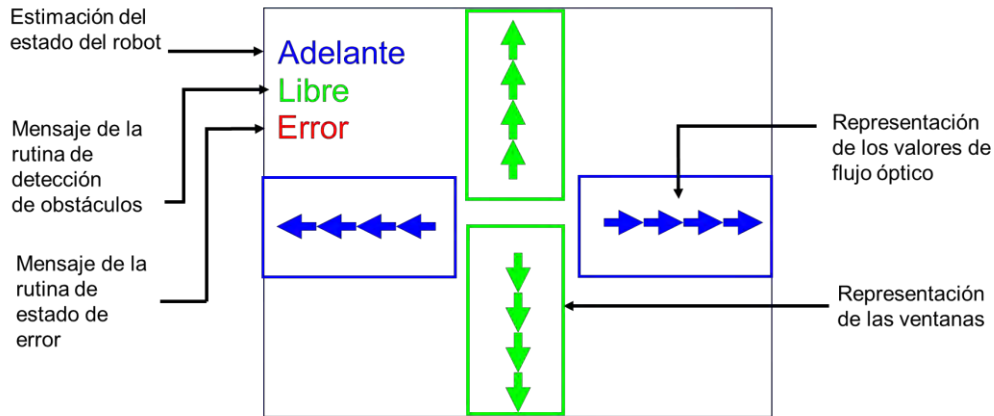


Figura 3.4 Elementos de la interfaz del sistema de navegación

Posteriormente, para facilitar el proceso en la última fase de experimentación bajo diferentes parámetros iniciales, se diseñó una interfaz gráfica con la librería *tkinter* donde se pueden ingresar los parámetros iniciales y poder ver el estado del robot, si hay presencia de obstáculos o si ha ingresado a la rutina de estado de error (figura 3.5).

Figura 3.5 Estructura de la interfaz del sistema de navegación

3.4 Conclusiones

En este capítulo se presentó la implementación del sistema de navegación propuesto en base al concepto teórico presentado en el capítulo dos, con respecto al uso de los valores de flujo óptico obtenido de ventanas de baja resolución para determinar el estado del robot, así como para evadir obstáculos en *tiempo real* sobre una computadora con recursos limitados como lo es la *Raspberry Pi*.

Para desarrollar el sistema se hizo uso del lenguaje de programación Python con el uso de librerías como *OpenCV*, *picamera*, *scipy*, *numpy*, *joblib* y *tkinter*.

Con los datos obtenidos de flujo óptico y los algoritmos de clasificación se implementaron tres rutinas para la evasión de obstáculos, para cuando el robot se encuentre imposibilitado para realizar la acción instruida a causa de factores externos y para el seguimiento de luz guía. Estas tres rutinas permiten que un robot rodante pueda navegar de manera autónoma en un entorno en *tiempo real*. Para el control durante la navegación, se crearon tres estados que indican que rutina mandará una instrucción al robot rodante.

Por último, se diseñó e implementó una interfaz gráfica para poder realizar experimentaciones bajo diferentes parámetros iniciales requeridos por el sistema de navegación.

Capítulo 4

Experimentación y análisis de los resultados

En este capítulo se presentan los recursos que se utilizaron durante la experimentación, las pruebas realizadas divididas en tres fases y por último se presenta un análisis de los resultados obtenidos.

4.1 Recursos utilizados

4.1.1 Robot rodante

Con el fin de realizar pruebas para determinar el funcionamiento de los algoritmos investigados se realizó el diseño y desarrollo de un robot rodante de cuatro llantas que puede realizar desplazamientos hacia adelante y hacia atrás, así también puede hacer giros sobre su propio eje a la derecha y a la izquierda.

Dentro de sus principales componentes se encuentra el uso de la placa de desarrollo ESP32 que recibe las señales de un control remoto para la toma de los *datasets*, y a su vez puede recibir las señales de la *Raspberry Pi* enviadas por el sistema de navegación robótica propuesto. Una vez que el ESP32 recibe las señales y a su vez son enviadas al módulo L298N el cual controla los cuatro motorreductores. en la figura 4.1 se muestran los componentes y principales conexiones, en la figura 4.2 se muestra al robot rodante, y en el anexo B se presenta de manera detallada la construcción y funcionamiento del mismo.

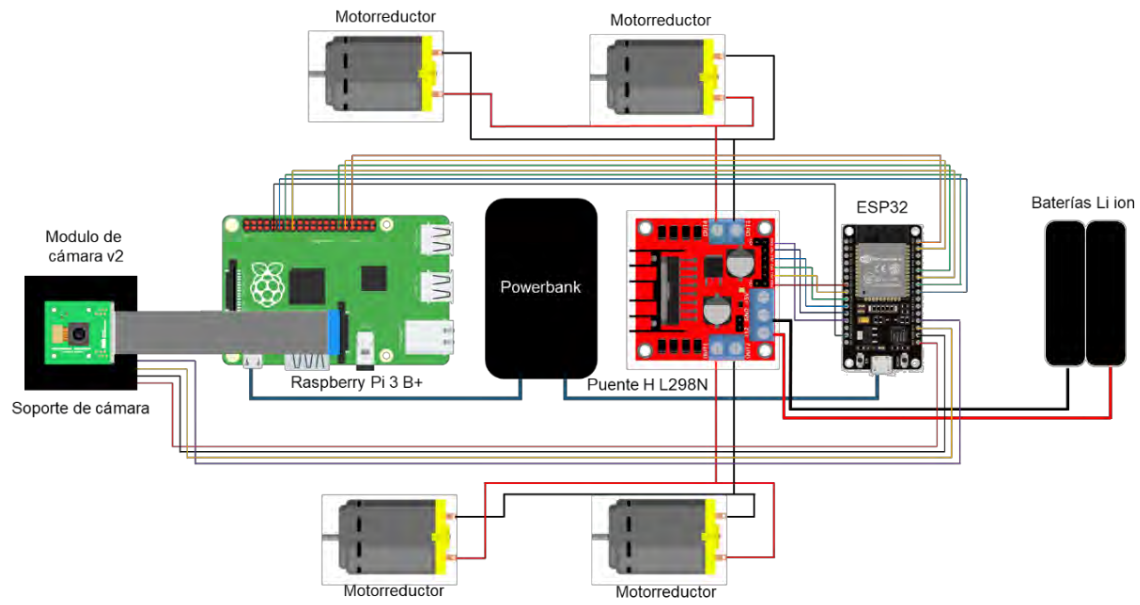


Figura 4.1 Conexiones entre los componentes del robot rodante



Figura 4.2 Robot rodante utilizado para la experimentación



4.1.2 Datasets

Para poder hacer comparaciones de los algoritmos de flujo óptico se hizo búsqueda de *datasets* utilizados en la literatura. Sin embargo, para la detección del estado del robot se realizó un conjunto de *datasets* propio dado que no existen dentro del estado del arte *datasets* que cumpla con las características para probar los algoritmos con ventanas de la imagen.

4.1.2.1 Datasets para experimento de la fase 1

Para la fase 1 se seleccionaron dos *datasets* en base al trabajo de (Huang & Song, 2017) los cuales se muestran en la tabla 4.1, los cuales constan cada uno de dos imágenes consecutivas con su respectivo *ground truth* (flujo óptico verdadero).

Tabla 4.1 Datasets para experimento de la fase 1

Nombre del <i>dataset</i>	Muestra de imágenes del <i>dataset</i>	Características
<i>Rubber Whale</i> (Baker et al., 2011)		Cantidad de fotogramas: 2 Resolución: 388×584 Campos de flujo óptico: 1 Máximo valor de desplazamiento en <i>ground truth</i>: 5 píxeles
<i>Urban2</i> (Baker et al., 2011)		Cantidad de fotogramas: 2 Resolución: 480×640 Campos de flujo óptico: 1 Máximo valor de desplazamiento en <i>ground truth</i>: 22 píxeles



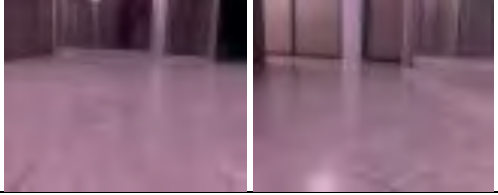
4.1.2.2 Datasets propios para los experimentos de la fase 2 y 3

Para las pruebas de detección del estado del robot se capturaron *datasets* de recorridos en diferentes condiciones en cuanto a luminosidad (interiores y exteriores), resolución utilizada y velocidad de fotogramas por segundo (tabla 4.2). Estos *datasets* están conformados por secuencias de fotogramas capturados junto con la instrucción realizada. Los lugares utilizados para la captura de los fotogramas fueron áreas exteriores (como el estacionamiento y la cancha) y el edificio del Departamento de Ciencias Computacionales (DCC) dentro del TecNM/CENIDET. Para las capturas de los *datasets* se procuró mantener la misma velocidad aproximada de 15 cm/s en desplazamientos y 9 cm/s en giros.

Tabla 4.2 Datasets para los experimentos de la fase 2 y 3

Nombre del <i>dataset</i>	Muestra de imágenes del <i>dataset</i>	Características
<i>Dataset 1</i>		Ángulo de inclinación de la cámara: 95° aprox. Cantidad de fotogramas: 6670 Resolución: 160 x 128 Fotogramas por segundo: 30 fps Campos de flujo óptico: 3335

Tabla 4.2 Datasets para los experimentos de la fase 2 y 3 (Continuación)

Nombre del dataset	Muestra de imágenes del dataset	Características
Dataset 2		Ángulo de inclinación de la cámara: 95° aprox. Cantidad de fotogramas: 16590 Resolución: 160 x 128 Fotogramas por segundo: 60 fps Campos de flujo óptico: 8295
Dataset 3		Ángulo de inclinación de la cámara: 95° aprox. Cantidad de fotogramas: 16590 Resolución: 80 x 64 (reescalado) Fotogramas por segundo: 60 fps Campos de flujo óptico: 8295
Dataset 4		Ángulo de inclinación de la cámara: 95° aprox. Cantidad de fotogramas: 7290 Resolución: 80 x 64 Fotogramas por segundo: 60 fps Campos de flujo óptico: 3645

4.2 Experimentación

4.2.1 Fase 1. Pruebas de funcionamiento (robot, algoritmos de flujo óptico)

4.2.1.1 Experimento 1 - Prueba de funcionamiento del robot bajo instrucciones de control remoto

Este experimento consistió en hacer pruebas con el robot rodante para verificar su capacidad para hacer los desplazamientos y giros. En este caso el robot fue controlado de manera remota mediante *bluetooth* entre el ESP32 y una aplicación de celular llamada *Arduino Bluetooth Robot Car*.

Los resultados obtenidos demuestran el correcto funcionamiento del robot rodante ante los diferentes desplazamientos y giros bajo las instrucciones del control remoto. El video de este experimento se encuentra disponible en https://drive.google.com/file/d/1rL2irVQxuWdXR5asrr9tUtcwE9ZHDyGS/view?usp=share_link.

4.2.1.2 Experimento 2 -Prueba de los algoritmos de flujo óptico

Este experimento consistió en evaluar los algoritmos de flujo óptico implementados, por lo que en base al trabajo de Huang & Song (2017) se seleccionaron los *datasets Rubber Whale* y *Urban2* para medir en base al error medio del punto final (métrica seleccionada en la sección 2.3.1) el correcto funcionamiento de los algoritmos de Horn-Schunck y Lucas-Kanade.

En este caso el flujo óptico fue calculado utilizando todos los píxeles de la imagen bajo diferentes configuraciones (En el caso de Horn-Schunck diferente número de iteraciones y en el caso de Lucas-Kanade diferentes tamaños de ventana de convolución). En la tabla

4.3 se muestra el resultado de las pruebas de los algoritmos de flujo óptico con los *datasets* seleccionados.

Tabla 4.3 Resultados del experimento 2

Dataset utilizado	Algoritmo utilizado	Condiciones adicionales	AEE	AEE en el trabajo de Huang & Song (2017)
<i>Rubber Whale</i>	Lucas-Kanade	Máscara de convolución de 3×3	1.2567	0.44
		Máscara de convolución de 5×5	0.4740	
		Máscara de convolución de 7×7	0.4735	
		Máscara de convolución de 15×15	0.3317	
	Horn-Schunck	1 iteración	1.0373	1.24
		3 iteraciones	0.8007	
		5 iteraciones	0.6819	
		10 iteraciones	0.5509	
<i>Urban2</i>	Lucas-Kanade	Máscara de convolución de 3×3	8.3940	8.04
		Máscara de convolución de 5×5	8.8736	
		Máscara de convolución de 7×7	8.8767	
		Máscara de convolución de 15×15	8.0383	
	Horn-Schunck	1 iteración	8.3955	8.15
		3 iteraciones	8.3170	
		5 iteraciones	8.2826	
		10 iteraciones	8.2449	

Como resultado de este experimento se pudo ver que el AEE (error medio de punto final) se ve reducido al realizar el proceso de cálculo de flujo óptico de Horn-Schunck con un mayor número de iteraciones, así también en el caso de Lucas-Kanade al utilizar una ventana. Por lo tanto, se consideró para los siguientes experimentos de este proyecto el uso de máscaras de convolución de 7×7 y 15×15 en el caso de Lucas-Kanade y en cuanto al algoritmo de Horn-Schunck se decidió realizar los siguientes experimentos con 5 y 10 iteraciones.

También se obtuvo un resultado similar al trabajo de Huang & Song (2017) con respecto al desempeño de los algoritmos de flujo óptico seleccionados. Con el *dataset Rubber Whale* que tiene pequeños desplazamientos de las intensidades de los píxeles se obtiene un menor AEE con respecto a los mismos experimentos hechos con el *dataset Urban2* donde hay presencia de grandes desplazamientos. Por lo tanto, se consideró para este proyecto utilizar una velocidad baja en el robot móvil para que los algoritmos de flujo óptico tengan un mejor desempeño.

4.2.2 Fase 2. Pruebas de algoritmos de flujo óptico para la detección del estado del robot

En esta fase de experimentación, se hicieron pruebas de los algoritmos de clasificación seleccionados con los datos obtenidos del cálculo de flujo óptico aplicado a los *datasets* propios generados de fotogramas de diferentes desplazamientos del robot rodante bajo instrucciones de un control remoto. Estas capturas fueron realizadas en diferentes entornos (interiores y exteriores) bajo diferentes velocidades de fotogramas por segundo.

Para esta experimentación se utilizó la técnica de validación cruzada (*cross validation*) con cinco particiones.

4.2.2.1 Experimento 3

En este experimento se hizo uso del *Dataset 1*, el cual tiene como característica el uso de ambientes exteriores, así como la velocidad de captura de 30 fps.

Primeramente, con el *Dataset 1* se hicieron pruebas de los algoritmos de flujo óptico bajo diferentes condiciones para observar los tiempos de procesamiento de las ventanas, como resultado se obtuvieron los tiempos de procesamiento que se muestran en la tabla 4.4.

Tabla 4.4 Tiempos de procesamiento del experimento 3

Algoritmo empleado	Tamaño de ventana	Puntos de flujo óptico por ventana	Condiciones adicionales	Tiempos de procesamiento con cuatro ventanas	
				Mínimo	Máximo
Lucas-Kanade	7×30	24	Máscara de convolución de 7×7	0.0778s	0.1868s
	15×30	15	Máscara de convolución de 15×15	0.0653s	0.1280s
Horn-Schunck	7×30	140	5 iteraciones	0.1219s	0.1582s
			10 iteraciones	0.1237s	0.2027s
	15×30	364	5 iteraciones	0.2363s	0.3929s
			10 iteraciones	0.2355s	0.4074s

Como se puede observar en la tabla 4.4 el algoritmo de Lucas-Kanade es el que presenta menores tiempos de procesamiento incluso con una mayor cantidad de información proporcionada por el tamaño de ventana más grande (15×30 píxeles).

Con los datos obtenidos del flujo óptico se realizaron las pruebas de clasificación para determinar el estado del robot mediante los algoritmos *CART*, *Adaboost* y *RandomForest* los cuales fueron evaluadas con las métricas mencionadas en la sección 2.3.2, en las tablas 4.5 y 4.6.

Tabla 4.5 Resultados de clasificación con datos del algoritmo de Lucas-Kanade

Condiciones	Algoritmo de clasificación	Árboles de decisión (CART)		AdaBoost		RandomForest	
	Métrica	Train	Test	Train	Test	Train	Test
Máscara de convolución de 7×7 y ventanas de 7×30	Accuracy	1.0	0.6611	0.7109	0.7586	1.0	0.826
	Recall	1.0	0.6611	0.7375	0.7586	1.0	0.829
	Presicion	1.0	0.6156	0.7109	0.7807	1.0	0.8698
	F1 Score	1.0	0.6365	0.7756	0.768	1.0	0.8698
Máscara de convolución de 15×15 y ventanas de 15×30	Accuracy	1.0	0.6467	0.7019	0.7601	1.0	0.8305
	Recall	1.0	0.6467	0.7328	0.7601	1.0	0.8305
	Presicion	1.0	0.6013	0.7019	0.8348	1.0	0.9337
	F1 Score	1.0	0.6209	0.7776	0.7938	1.0	0.8662

Tabla 4.6 Resultados de clasificación con datos del algoritmo de Horn-Schunck

Condiciones	Algoritmo de clasificación	Árboles de decisión (CART)		AdaBoost		RandomForest	
	Métrica	Train	Test	Train	Test	Train	Test
Ventanas de 7×30 y 5 iteraciones	Accuracy	1.0	0.6836	0.7736	0.7946	1.0	0.8275
	Recall	1.0	0.6836	0.7736	0.7946	1.0	0.8275
	Presicion	1.0	0.6605	0.8404	0.7756	1.0	0.933
	F1 Score	1.0	0.6671	0.8029	0.8176	1.0	0.8757
Ventanas de 7×30 y 10 iteraciones	Accuracy	1.0	0.6806	0.6962	0.7946	1.0	0.8185
	Recall	1.0	0.6806	0.6962	0.7946	1.0	0.8185
	Presicion	1.0	0.6412	0.7756	0.8459	1.0	0.9205
	F1 Score	1.0	0.6594	0.7298	0.8176	1.0	0.8658
Ventanas de 15×30 y 5 iteraciones	Accuracy	1.0	0.6791	0.6962	0.7946	1.0	0.7946
	Recall	1.0	0.6791	0.6962	0.7946	1.0	0.7946
	Presicion	1.0	0.6206	0.7756	0.8459	1.0	0.8459
	F1 Score	1.0	0.6462	0.7298	0.8176	1.0	0.8176
Ventanas de 15×30 y 10 iteraciones	Accuracy	1.0	0.7121	0.697	0.7886	1.0	0.8305
	Recall	1.0	0.7121	0.7316	0.7886	1.0	0.8305
	Presicion	1.0	0.7142	0.7316	0.8527	1.0	0.9485
	F1 Score	1.0	0.7097	0.7776	0.8187	1.0	0.8845

Con los resultados de los algoritmos de clasificación, se obtuvo que las condiciones que dan mejores resultados, en cuanto a la estimación del estado del robot, son el uso de máscara de convolución de 15×15 y ventanas de 15×30 en el algoritmo de Lucas Kanade, mientras que con el algoritmo de Horn-Schunck la estimación del estado del robot dio mejor resultado al realizar el proceso 10 veces con ventanas de 15×30 . Sin embargo, y dados los tiempos de procesamiento en la tabla 4.4, se decidió realizar el proceso durante 5 iteraciones para los siguientes experimentos.

Un factor importante observado desde los resultados obtenidos es que la velocidad de captura de 30 fps hace que en ciertas ocasiones exista confusión al momento de la clasificación. Por lo tanto, se decidió aumentar la velocidad de captura de los fotogramas de 30 a 60 fotogramas por segundo.

4.2.2.2 Experimento 4

Este experimento tuvo como propósito reducir la resolución utilizada con el fin de que con el mismo tamaño de ventanas se pueda cubrir más partes del área de visión del robot para el proceso de evasión de obstáculos.

Para este experimento se utilizaron los *datasets* 2 y 3 los cuales consisten en el mismo conjunto de imágenes, con la diferencia de que tienen fotogramas de diferentes resoluciones (160×128 y 80×64 respectivamente). Los datos obtenidos del cálculo del flujo óptico fueron ingresados a los algoritmos de clasificación seleccionados para comparar el desempeño de la estimación del estado del robot con los dos *datasets*. Los resultados de la clasificación se muestran en la tabla 4.7.

Tabla 4.7 Resultados de clasificación del experimento 4

Algoritmo de Flujo Óptico y condiciones	Algoritmo de clasificación	Árboles de decisión (CART)		AdaBoost		RandomForest	
	Métrica	Train	Test	Train	Test	Train	Test
<i>Dataset 2</i> (resolución de 160×128 pixeles)							
Lucas-Kanade con ventanas de 15×30 y con máscara de convolución de 15×15	Accuracy	1.0	0.5085	0.5409	0.6809	1.0	0.6774
	Recall	1.0	0.5085	0.5409	0.6809	1.0	0.6774
	Presicion	1.0	0.5014	0.6284	0.5681	1.0	0.7985
	F1 Score	1.0	0.503	0.5681	0.7187	1.0	0.7164
Horn-Schunck con ventanas de 15×30 y 5 iteraciones	Accuracy	1.0	0.4797	0.5357	0.6498	1.0	0.6463
	Recall	1.0	0.4797	0.5357	0.6498	1.0	0.6463
	Presicion	1.0	0.457	0.6362	0.7335	1.0	0.7246
	F1 Score	1.0	0.4627	0.5652	0.7335	1.0	0.6692

Tabla 4.7 Resultados de clasificación del experimento 4 (Continuación)

Algoritmo de Flujo Óptico y condiciones	Algoritmo de clasificación	Árboles de decisión (CART)		AdaBoost		RandomForest	
	Métrica	Train	Test	Train	Test	Train	Test
<i>Dataset 3</i> (resolución de 80×64 píxeles)							
Lucas-Kanade con ventanas de 15×30 y con máscara de convolución de 15×15	Accuracy	1.0	0.6023	0.6086	0.6615	1.0	0.7038
	Recall	1.0	0.6023	0.6086	0.6615	1.0	0.7038
	Presicion	1.0	0.5878	0.6692	0.6905	1.0	0.7475
	F1 Score	1.0	0.5925	0.6285	0.6711	1.0	0.7138
Horn-Schunck con ventanas de 15×30 y 5 iteraciones	Accuracy	1.0	0.5812	0.5661	0.6639	1.0	0.7038
	Recall	1.0	0.5812	0.5661	0.6639	1.0	0.7038
	Presicion	1.0	0.5663	0.6258	0.6849	1.0	0.7475
	F1 Score	1.0	0.5695	0.5858	0.6642	1.0	0.7138

Con los resultados obtenidos se puede observar que el uso de una menor resolución mejora la estimación del estado del robot en el uso de los cuatro clasificadores. Sin embargo, se obtuvieron resultados bajos en las métricas en el uso de los clasificadores, esto se puede deber a factores externos como los pisos de los entornos utilizados que producen cambios de intensidades de las imágenes capturadas con la cámara del robot rodante que alteran los valores de flujo óptico para la clasificación. Por lo tanto, se decidió realizar los siguientes experimentos en entornos cerrados donde las condiciones del piso y luminosidad permitan generar mejores patrones de los valores de flujo óptico para la estimación del estado del robot. Asimismo, se decidió el uso de la resolución de 80×64 píxeles para los siguientes experimentos.

4.2.2.4 Experimento 5

El propósito de este experimento es probar la estimación del estado del robot con los valores de flujo óptico obtenidos de secuencias de fotogramas en ambientes cerrados. Para este experimento se utilizó el *Dataset 4* el cual consta de un conjunto de fotogramas de una resolución de 80×64 píxeles los cuales fueron tomados dentro del edificio del Departamento de Ciencias Computacionales del TecNM/CENIDET.

Las condiciones de tamaño de ventanas, máscara de convolución (Lucas-Kanade) y número de iteraciones (Horn-Schunck) fueron las seleccionadas en el experimento anterior. Los resultados de la clasificación del estado del robot rodante se muestran en la tabla 4.8.

Tabla 4.8 Resultados de clasificación del experimento 5

Algoritmo de Flujo Óptico y condiciones	Algoritmo de clasificación	Árboles de decisión (CART)		AdaBoost		RandomForest	
	Métrica	Train	Test	Train	Test	Train	Test
Lucas-Kanade con ventanas de 15×30 y con máscara de convolución de 15×15	Accuracy	1.0	0.8571	0.7974	0.9135	1.0	0.9210
	Recall	1.0	0.8571	0.7974	0.9135	1.0	0.9210
	Presicion	1.0	0.8376	0.8213	0.9191	1.0	0.9255
	F1 Score	1.0	0.8441	0.8015	0.9142	1.0	0.9206
Horn-Schunck con ventanas de 15×30 y 5 iteraciones	Accuracy	1.0	0.8007	0.7179	0.8308	1.0	0.8834
	Recall	1.0	0.8007	0.7179	0.8308	1.0	0.8834
	Presicion	1.0	0.7889	0.7683	0.859	1.0	0.9016
	F1 Score	1.0	0.7919	0.7359	0.8313	1.0	0.8905

De acuerdo con los resultados de la calificación de los valores de flujo óptico obtenidos de fotogramas capturados en un entorno se nota una mejor estimación del estado del robot rodante. Asimismo, de los dos algoritmos de flujo óptico se puede observar que el algoritmo de Lucas-Kanade junto con el algoritmo de clasificación *RandomForest* son los que mejores resultados dan en cuanto a la estimación del estado del robot. Por lo que se escogió el interior del edificio del DCC como ambiente para la siguiente fase de experimentación y también el uso de un modelo entrenado con el algoritmo de *RandomForest* para los siguientes experimentos.

4.2.3 Fase 3. Pruebas de funcionamiento del sistema navegación completo (detección de estado y evasión de obstáculos)

En esta fase de experimentación se hicieron pruebas de la integración del sistema completo de navegación. Como ambiente de experimentación se hizo uso de áreas interiores del edificio del DCC. Para la captura de fotogramas se utilizó la resolución de 80×64 pixeles con una velocidad de captura de 60 fps. Para esta fase de experimentaciones se mantuvo la misma velocidad a la que fue capturado el *dataset 4* que es de 15 cm/s en desplazamientos y 9 cm/s en giros aproximadamente. Por último, para la estimación del robot rodante se utilizaron los modelos entrenados con el algoritmo de *RandomForest* obtenidos del experimento cinco.

4.2.3.1 Experimento 6

Este experimento tuvo como propósito evaluar el sistema de navegación con la estimación del estado del robot y evasión de obstáculos. Para este experimento solo se utilizó la ecuación de balance horizontal (ecuación 2.11) para la detección de obstáculos, poniendo como umbral de balance 0.5 de acuerdo al comportamiento de los valores de flujo óptico en experimentos anteriores. Como área de experimentación se utilizaron los pasillos del segundo piso del edificio del DCC. Para este experimento se realizó el cálculo de flujo óptico

de Lucas-Kanade con ventanas de 15×30 y con una máscara de convolución de 15×15 . De las rutinas mencionadas en el capítulo anterior solo se utilizaron las rutinas de estado de error y detección de obstáculos, delimitando un umbral de 20 y 1 respectivamente en base a los tiempos de procesamiento de los experimentos anteriores, con esto se buscó darle tiempo al robot para poder ejecutar las instrucciones dadas por estas rutinas.

Como resultado se obtuvo que el robot logró evitar la colisión en 22 de 29 ocasiones sobre paredes u obstáculos (eficacia del 75.86%). Sin embargo, en los 7 casos donde se presentó una colisión el robot logró ejecutar el estado de error para poder continuar el recorrido. Las fallas presentadas en la evasión de obstáculos se debieron a que el robot estaba de frente a paredes o columnas con ausencia de texturas, asimismo no había reglas de balance establecidas para las ventanas verticales (ventanas 3 y 4). Por lo tanto, se decidió incluir la ecuación de balance vertical para los siguientes experimentos.

En cuanto a la estimación del estado del robot este tuvo un buen funcionamiento con el modelo entrenado, dado que el robot no tuvo que entrar en el estado de error.

En cuanto a los tiempos de procesamiento el tiempo del procesamiento del sistema incluyendo el cálculo del flujo óptico, la estimación del estado del robot y la detección de obstáculos con el algoritmo de Lucas-Kanade se realizó el proceso en un tiempo entre 0.1239 y 0.2043 segundos logrando un procesamiento de 3 a 8 pares de fotogramas por segundo, permitiendo que el robot respondiera correctamente a la presencia de obstáculos.

4.2.3.2 Experimento 7

Este experimento tuvo como propósito evaluar el sistema de navegación con la estimación del estado del robot y evasión de obstáculos. Para este experimento se utilizaron las ecuaciones de balance horizontal y vertical (ecuación 2.11 y 2.12) para la detección de obstáculos, en este caso se incrementó el umbral de balance en busca de tener un mejor comportamiento. Como área de experimentación se utilizaron los pasillos del segundo piso del edificio del DCC. Para este experimento se realizó el cálculo de flujo óptico de Lucas-Kanade con ventanas de 15×30 y con una máscara de convolución de 15×15 . De las rutinas mencionadas en el capítulo anterior solo se utilizaron las rutinas del estado de error y detección de obstáculos, delimitando un umbral de 20 y 3 respectivamente en base a los tiempos de procesamiento del experimento anterior, con esto se buscó darle más tiempo al robot para ejecutar las instrucciones dadas por estas rutinas.

Como resultado se tuvo que el robot evadió en 9 de 12 ocasiones la presencia de paredes u obstáculos (eficacia del 75%) y a comparación del experimento anterior se presentó un mejor comportamiento ante la presencia de obstáculos de colores solidos como paredes y columnas. Sin embargo, hubo ocasiones donde el robot se vio atorado por diversos factores por lo que entró en dos ocasiones a la rutina de estado de error logrando que el robot pudiera continuar su recorrido.

En cuanto a la estimación del estado del robot este tuvo un buen funcionamiento con el modelo entrenado, dado que incluso en ocasiones en las que el robot llegó a atorarse la estimación del robot permitió que pudiera ejecutarse la rutina de estado de error y mandar instrucciones al robot para poder salir de aquella situación.

En cuanto a los tiempos de procesamiento del sistema donde se contempla el cálculo del flujo óptico, la estimación del estado del robot y la detección de obstáculos, se obtuvo que con el algoritmo de Lucas-Kanade se realizó el proceso en un tiempo entre 0.1224 y 0.2575 segundos logrando procesar de 3 a 8 pares de fotogramas por segundo, permitiendo que el robot respondiera correctamente a la presencia de obstáculos.

4.2.3.3 Experimento 8

Este experimento tuvo como objetivo comprobar el funcionamiento y los tiempos de procesamiento del sistema completo con dos de las rutinas implementadas (estado de error y detección de obstáculos), con el fin de corroborar el correcto funcionamiento del sistema de navegación, en el cual las instrucciones al robot son dadas por la implementación de los algoritmos de flujo óptico (detección del estado del robot y evasión de obstáculos) y que el sistema de seguimiento de luz guía (utilizado en el siguiente experimento) es solo un apoyo para darle trayectoria al robot rodante pero no interviene en la tarea de evasión de obstáculos.

En este experimento solo se utilizaron las rutinas del estado de error y detección de obstáculos ya que no había existencia de luz guía, delimitando un umbral de 20 y 3 respetivamente en base a los resultados del experimento anterior. Para las fórmulas de balance la para detección de obstáculos se dejó 1 como umbral dado el buen funcionamiento en el experimento anterior.

Para este experimento se utilizó como escenario el aula 3104 del edificio del DCC. y se colocaron como obstáculos objetos encontrados dentro del laboratorio de Ciencias Computacionales como se muestra en la figura 4.3.

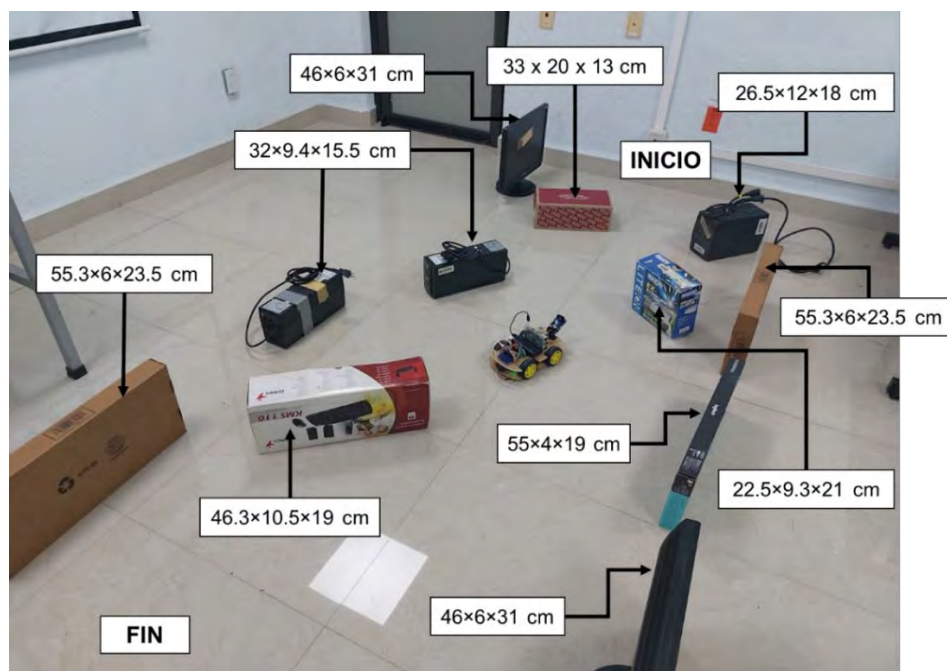


Figura 4.3 Área de navegación para el experimento 8

Para este experimento se realizaron dos pruebas: una con el algoritmo de Lucas-Kanade y otra con el algoritmo de Horn-Schunck. Como resultado de este experimento se obtuvo que cuando se utilizó el algoritmo de Lucas-Kanade para el cálculo del flujo óptico el robot reaccionó mejor ante la presencia de obstáculos logrando evadir 3 de 3 obstáculos presentes en el campo de visión (eficacia del 100%) a comparación del algoritmo de Horn-Schunck en las mismas condiciones donde colisiono con 3 obstáculos presentes en el campo de visión (eficacia del 0%). En cuanto a los tiempos de procesamiento los resultados se muestran en la tabla 4.9 donde se logran procesar de 5 a 7 pares de fotogramas por segundo con el algoritmo de Lucas-Kanade y de 3 a 4 pares de fotogramas por segundo con el algoritmo de Horn-Schunck.

Tabla 4.9 Tiempos de procesamiento del experimento 8

Algoritmo empleado	Condiciones	Tiempos de procesamiento del sistema de navegación por cada dos fotogramas	
		Mínima	Máxima
Lucas-Kanade	Ventanas de 15×30 y máscara de convolución de 15×15	0.1338s	0.1737s
Horn-Schunck	Ventanas de 15×30 y 5 iteraciones	0.2553s	0.2870s

4.2.3.4 Experimento 9

Este experimento tuvo como objetivo comprobar el funcionamiento y los tiempos de procesamiento del sistema completo con todas las rutinas implementadas (estado de error, detección de obstáculos y seguimiento de luz guía), con el fin de corroborar el correcto funcionamiento del sistema de navegación.

En este experimento se utilizaron las rutinas de estado de error, de evasión de obstáculos y de seguimiento de luz guía, delimitando un umbral de 20, 3 y 40 respetivamente en base a los tiempos de procesamiento y los resultados obtenidos en el experimento anterior. Para las fórmulas de balance para la detección de obstáculos se dejó 1 como umbral dado el buen funcionamiento en el experimento anterior.

Para este experimento se utilizó como escenario el aula 3104 del edificio del DCC y se colocaron como obstáculos objetos encontrados dentro del laboratorio de Ciencias Computacionales, así también se marcó un punto final para el recorrido del robot como se muestra en la figura 4.4.

Para este experimento se realizaron dos pruebas: una con el algoritmo de Lucas-Kanade y otra con el algoritmo de Horn-Schunck. Como resultado de este experimento se obtuvo que cuando se utilizó el algoritmo de Lucas-Kanade para el cálculo del flujo óptico el robot reaccionó mejor ante la presencia de obstáculos evadiendo 6 de los 6 obstáculos presentes en el campo de visión (eficacia del 100%) en comparación del algoritmo de Horn-Schunck

en las mismas condiciones (figura 4.5) evadiendo 2 de 4 obstáculos presentes en el campo de visión (eficacia del 50%).

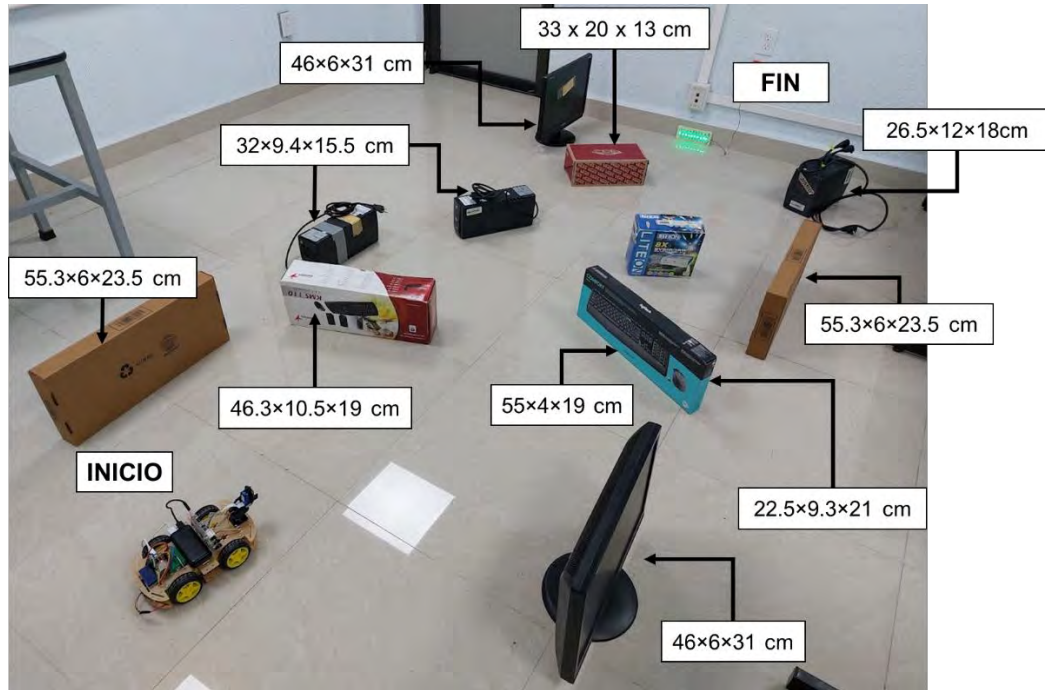


Figura 4. 4 Ruta para el experimento 9

El sistema de seguimiento de la luz guía le permitió al robot llegar al punto final de manera exitosa en ambos casos. En cuanto a los tiempos de procesamiento los resultados se muestran en la tabla 4.10 donde se logran procesar de 5 a 7 pares de fotogramas por segundo con el algoritmo de Lucas-Kanade y de 3 a 4 pares de fotogramas por segundo con el algoritmo de Horn-Schunck.

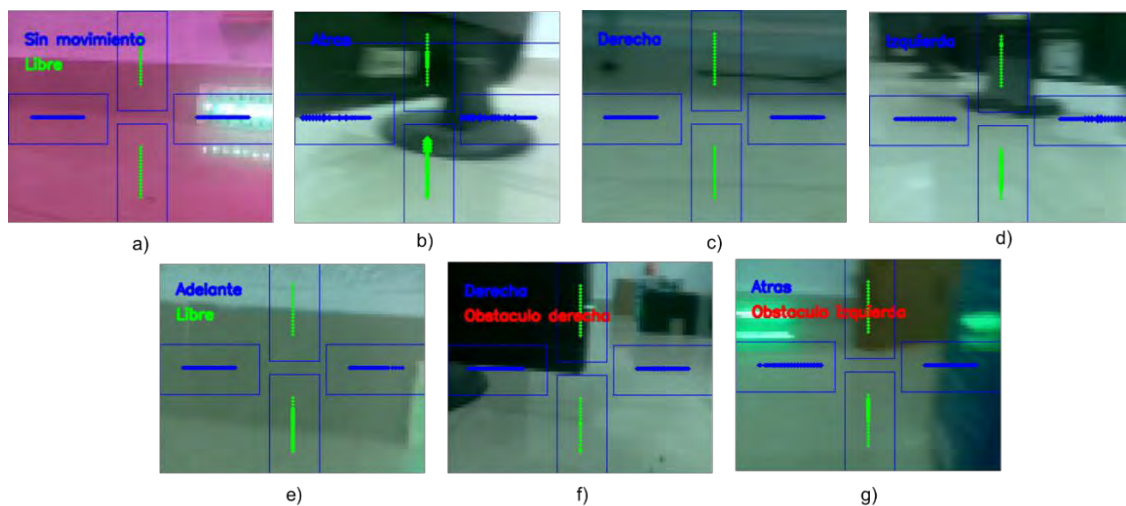


Figura 4.5 Funcionamiento del robot para la detección del estado del robot: a) sin movimiento, b) atrás, c) derecha, d) izquierda y la detección de la presencia de obstáculos e) libre f) obstáculo a la derecha y g) obstáculo a la izquierda.

obstáculos logrando esquivar 3 de los 3 obstáculos presentes en el campo de visión (eficacia del 100%) a comparación del algoritmo de Horn-Schunck en las mismas condiciones esquivando 3 de 5 obstáculos presentes en el campo de visión (eficacia del 60%). El sistema de seguimiento de la luz guía le permitió al robot llegar al punto final de manera exitosa en ambos casos. En cuanto a los tiempos de procesamiento los resultados se muestran en la tabla 4.11 donde se logran procesar de 5 a 7 pares de fotogramas por segundo con el algoritmo de Lucas-Kanade y de 3 a 4 pares de fotogramas por segundo con el algoritmo de Horn-Schunck.

Tabla 4.11 Tiempos de procesamiento del experimento 10

Algoritmo empleado	Condiciones	Tiempos de procesamiento del sistema de navegación por cada dos fotogramas	
		Mínima	Máxima
Lucas-Kanade	Ventanas de 15×30 y máscara de convolución de 15×15	0.1372s	0.1832s
Horn-Schunck	Ventanas de 15×30 y 5 iteraciones	0.2220s	0.2723s

4.2.3.5 Experimento 11

Este experimento tuvo como objetivo comprobar el funcionamiento y los tiempos de procesamiento del sistema completo con todas las rutinas implementadas (estado de error, detección de obstáculos y seguimiento de luz guía), con el fin de corroborar el correcto funcionamiento del sistema de navegación. Asimismo, se hicieron otras tomas en video para capturar el comportamiento del robot ante un nuevo circuito de obstáculos y se hizo uso de la interfaz gráfica presentada en la sección 3.3.

En este experimento se utilizaron los mismos umbrales para las rutinas de estado de error, de evasión de obstáculos y de seguimiento de luz guía utilizados en el experimento anterior en base a los resultados obtenidos. Asimismo, se dejó 3 como umbral para las fórmulas de balance para la detección de obstáculos.

Para este experimento se utilizó como escenario el pasillo principal del edificio del Departamento de Ciencias Computacionales del TecNM/CENIDET y se colocaron como obstáculos objetos encontrados dentro del laboratorio de Ciencias Computacionales, así también se marcó un punto final para el recorrido del robot como se muestra en la figura 4.7. El algoritmo de cálculo de flujo óptico utilizado fue el de Lucas-Kanade por los buenos resultados obtenidos en los experimentos anteriores.

4.2.4 Tiempos de procesamiento y rendimiento del sistema de navegación

De los tiempos de procesamiento de los experimentos realizados se midió el tiempo empelado en cada uno de los pasos del sistema de navegación, estos tiempos fueron promediados para determinar el porcentaje que toma cada bloque en el tiempo total de procesamiento, los porcentajes obtenidos se muestran en la figura 4.8.

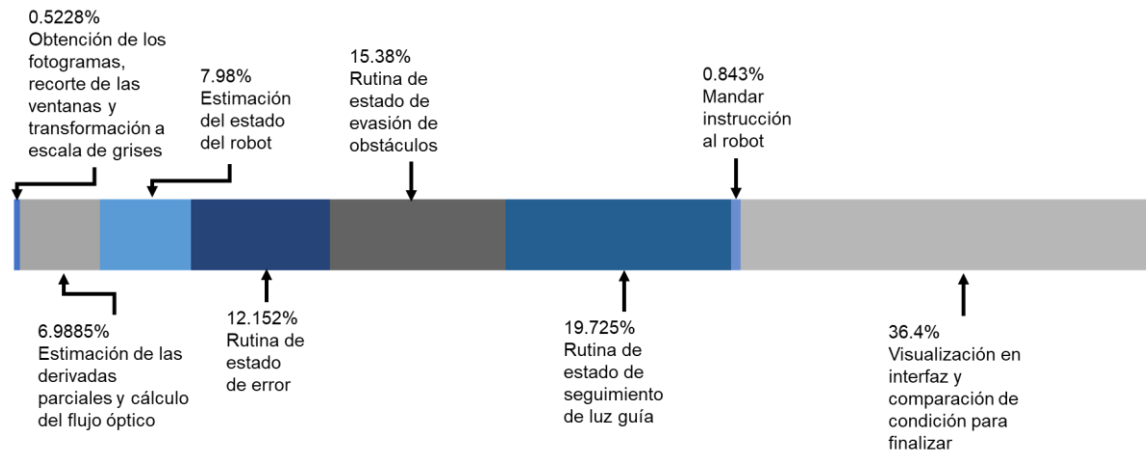


Figura 4.8 Distribución del tiempo de procesamiento

En cuanto al rendimiento de la *Rapsberry Pi*, de los experimentos realizados se obtuvo que el rendimiento del CPU fue constante haciendo un uso menor del 20% y de la memoria RAM se hizo un uso menor a los 150 MB (figura 4.9).

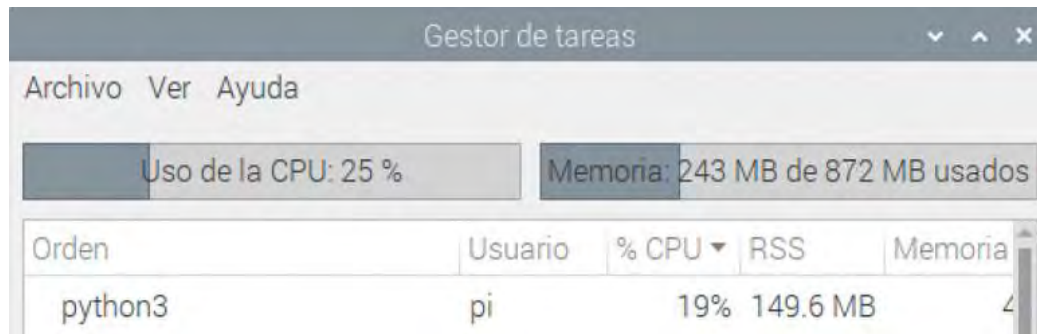


Figura 4.9 Uso del CPU y de la memoria RAM durante las experimentaciones

4.3 Análisis de los resultados obtenidos

De acuerdo con la experimentación, se obtuvieron los resultados esperados con respecto al objetivo que tenía cada experimento. En las tablas 4.13, 4.14 y 4.15 se presenta un resumen de los resultados obtenidos con respecto a los objetivos esperados en las tres fases de experimentación.

Tabla 4.13 Resultados de los experimentos de la fase 1

Experimento	Objetivo esperado	Resultado obtenido
1	Tener funcionamiento correcto del robot rodante para los desplazamiento y giros.	El robot funciona de manera correcta para los desplazamientos hacia atrás y hacia delante, así como los giros a la derecha y a la izquierda. El robot actúa de manera correcta ante las instrucciones de control remoto.
2	Tener una correcta implementación de algoritmos de flujo óptico.	Los resultados con respecto al AEE son similares a las pruebas realizadas en el trabajo de Huang & Song (2017), por lo que las implementaciones de los algoritmos de flujo óptico son las correctas.

Tabla 4.14 Resultados de los experimentos de la fase 2

Experimento	Estimación del estado del robot	Procesamiento en tiempo real
3	Con el uso de ventanas de 15×30 se obtuvieron mejores resultados que con otras configuraciones de los algoritmos de flujo óptico.	El algoritmo de Lucas-Kanade presenta mejores tiempos de procesamiento logrando calcular el flujo óptico en 0.0653s con el uso de ventanas de 15×30 píxeles.
4	Se obtuvieron bajos resultados en clasificación, pero se notó una mejoría al utilizar la resolución de 80×64 píxeles.	<i>No se midió en este experimento</i>
5	Los datos del flujo óptico con el algoritmo de Lucas-Kanade permiten que el algoritmo de <i>RandomForest</i> pueda detectar el estado del robot con un <i>F1 Score</i> del 92%.	<i>No se midió en este experimento</i>

Tabla 4.15 Resultados de los experimentos de la fase 3

Experimento	Estimación del estado del robot	Evasión de obstáculos	Procesamiento en tiempo real
6	Se logró una correcta estimación del estado del robot, permitiendo que robot desarrolle su recorrido sin anomalías. Se ejecutó en 7 ocasiones la rutina de estado de error de manera exitosa.	El robot logró evadir 22 de 29 de los obstáculos presentes durante su trayecto (eficacia del 75.86%).	Los tiempos de procesamiento fueron entre 0.1224 y 0.2575 segundos logrando procesar de 3 a 8 pares de fotogramas por segundo.
7	Se logró una correcta estimación del estado del robot, permitiendo que robot desarrolle su recorrido sin anomalías. Se ejecutó en 2 ocasiones la rutina de estado de error de manera exitosa.	El robot logró evadir 9 de 12 de los obstáculos presentes durante su trayecto (eficacia del 75%). Hubo mejora en presencia de áreas con ausencia de textura.	Los tiempos de procesamiento fueron entre 0.1239 y 0.2043 segundos logrando procesar de 3 a 8 pares de fotogramas por segundo.
8	Se logró una correcta estimación del estado del robot, permitiendo que robot desarrolle su recorrido sin anomalías.	Los datos obtenidos por el algoritmo de Lucas-Kanade permitieron detectar 3 de 3 obstáculos durante el trayecto (eficacia del 100%).	Los mejores tiempos de procesamiento fueron entre 0.1338 y 0.1737 segundos con el algoritmo de Lucas-Kanade procesando de 5 a 7 pares de fotogramas por segundo.
9	Se logró una correcta estimación del estado del robot, permitiendo que robot desarrolle su recorrido sin anomalías. La presencia de la luz guía permitió definir un trayecto para la experimentación.	Los datos obtenidos por el algoritmo de Lucas-Kanade permitieron detectar 6 de 6 obstáculos durante el trayecto (eficacia de 100%).	Los mejores tiempos de procesamiento fueron entre 0.1361 y 0.1860 segundos con el algoritmo de Lucas-Kanade procesando de 5 a 7 pares de fotogramas por segundo.
10	Se logró una correcta estimación del estado del robot, permitiendo que robot desarrolle su recorrido sin anomalías. La presencia de la luz guía permitió definir un trayecto para la experimentación	Los datos obtenidos por el algoritmo de Lucas-Kanade permitieron detectar 3 de 3 obstáculos durante el trayecto (eficacia del 100%).	Los mejores tiempos de procesamiento fueron entre 0.1372 y 0.1832 segundos con el algoritmo de Lucas-Kanade procesando de 5 a 7 pares de fotogramas por segundo.
11	Se logró una correcta estimación del estado del robot, permitiendo que robot desarrolle su recorrido sin anomalías.	El robot logró evadir 4 de 5 de los obstáculos presentes durante su trayecto (eficacia del 80%).	Los tiempos de procesamiento fueron entre 0.1428 y 0.1843 segundos con el algoritmo de Lucas-Kanade procesando de 5 a 7 pares de fotogramas por segundo.

Con los resultados obtenidos de las experimentaciones se puede concluir que se obtuvieron mejores resultados en la estimación del estado del robot mediante el algoritmo de *RandomForest* obteniendo un *accuracy* del 92.1% y un *F1 Score* de 92% con los datos de flujo óptico del algoritmo de Lucas-Kanade sobre pares de fotogramas de 80×64 píxeles tomados a una velocidad de 60 fps.

En cuanto a los parámetros para el sistema de navegación se obtuvieron mejores resultados con el algoritmo de Lucas-Kanade, un tamaño de ventanas de 15×30, un umbral de balance de 1, un límite de estados de error de 20, un límite de estados de evasión de obstáculos de 3 y un límite de estados de *perdido* de 40 en los experimentos en los que hubo presencia de luz guía.

En cuanto a los tiempos de procesamiento, los mejores resultados se obtuvieron con el algoritmo de Lucas-Kanade logrando tiempos de procesamiento de 5 a 7 pares de fotogramas por segundo, por lo que con el robot rodante con una velocidad aproximada de 15 cm/s en desplazamientos y 9 cm/s en giros se obtuvo un buen comportamiento logrando esquivar del 75% al 100% de los obstáculos presentes (promedio de 88.47%), por lo tanto, se logra que todo el proceso se haga en *tiempo real*.

Capítulo 5

Conclusiones y trabajo futuro

En esta última sección se presentan las conclusiones, los objetivos alcanzados, las aportaciones, trabajos futuros y las actividades académicas adicionales.

5.1 Conclusiones generales

El sistema de navegación desarrollado permite estimar el estado de un robot móvil y evadir obstáculos en *tiempo real* utilizando las técnicas de flujo óptico y usando solo la información de 1,800 píxeles de la imagen (uso de ventanas de 15×30) que entra dentro del rango de lo definido previamente como baja resolución.

Los resultados obtenidos de la experimentación muestran que con los valores de flujo óptico calculados con el algoritmo de Lucas-Kanade se obtienen mejores resultados en cuanto a la estimación del estado del robot, detección de obstáculos y tiempos de procesamiento en comparación con el algoritmo de Horn-Schunck.

Para la estimación del estado del robot el algoritmo de clasificación *RandomForest* obtuvo mejores resultados logrando tener en pruebas un *accuracy* del 92.1% y un *F1 Score* de 92% con los datos de flujo óptico del algoritmo de Lucas-Kanade.

Los tiempos de proceso obtenidos de la experimentación dieron como resultado que el uso de una computadora de pocos recursos de procesamiento como la *Raspberry Pi 3B+* permite realizar el proceso de navegación robótica en *tiempo real*, permitiendo procesar de 5 a 7 pares de fotogramas por segundo (con Lucas-Kanade) y de 3 a 4 pares de fotogramas por segundo (con Horn-Schunck).

A pesar de que los algoritmos de flujo óptico son dependientes de la textura del entorno, los resultados de la experimentación muestran que el sistema de navegación funciona incluso en condiciones no favorables como la presencia de paredes con ausencia de textura o pisos que reflejan la luz en interiores.

5.2 Objetivos alcanzados

El objetivo general de este trabajo definido como “desarrollar un algoritmo de visión robótica inspirado en la visión de insectos voladores capaz de evitar obstáculos en *tiempo real* en espacios reducidos, implementado con visión de baja resolución y técnicas de flujo óptico” se desarrolló de manera exitosa y los cuatro objetivos específicos se cumplieron de acuerdo a lo establecido. En la tabla 5.1 se muestran los comentarios correspondientes a cada uno de los objetivos específicos.

Tabla 5. 1 Comentarios de los objetivos específicos

Objetivo específico	Comentarios
Estudiar las técnicas de aplicación de los métodos directos y flujo óptico.	Se estudiaron artículos del estado del arte sobre sistemas de navegación robótica que utilizan las técnicas de flujo óptico como métodos directos, se revisaron artículos donde se evalúan los algoritmos de flujo óptico más utilizados y se estudió la implantación de los algoritmos de flujo óptico de Horn-Schunck y Lucas-Kanade en este proyecto.
Desarrollar un algoritmo que pueda evitar obstáculos en <i>tiempo real</i>	Se desarrolló un sistema de navegación robótica capaz de evitar del 75% al 100% de obstáculos en <i>tiempo real</i> , dado que el cálculo de flujo óptico solo se aplica a ventanas de la imagen reduciendo su costo computacional. Todo del proceso de navegación robótica se puede realizar de 5 a 7 veces por segundo.
Implementar el algoritmo en un robot móvil rodante.	Se implementó el sistema de navegación de forma embebida en un robot rodante.
Evaluar y mejorar, de ser necesario, el algoritmo propuesto	Se realizó la evaluación del sistema de navegación ante diferentes circunstancias y asimismo se agregaron diferentes mejoras al sistema de navegación.

Con respecto a los trabajos estudiados del estado del arte, el sistema de navegación desarrollado se puede destacar por las características que se muestran en la tabla 5.2.

Tabla 5. 2 Resumen del sistema de navegación robótica propuesto en base a las características resaltadas de los trabajos estado del arte

Robot utilizado	Cámara utilizada	Resolución utilizada	Algoritmo	Propósito
Robot rodante de cuatro motores	Módulo de cámara v2 para Raspberry	2 ventanas de 15×30 y 2 ventanas de 30×15 pixeles (1,800 pixeles)	Lucas-Kanade (LK)	Estimación del estado del robot y evasión de obstáculos con información obtenida de ventanas de imágenes de baja resolución.

5.3 Aportaciones

- Se presenta un sistema de navegación robótica capaz de reconocer el estado del robot, así como evadir obstáculos en *tiempo real* usando solamente el cálculo del flujo óptico en fotogramas de baja resolución obtenidos de la cámara, sin hacer uso de sensores adicionales.
- Se presenta una metodología para la estimación del estado del robot usando la información del flujo óptico calculado sobre ventanas de la imagen para reducir el costo computacional que generaría aplicar el mismo cálculo sobre la imagen completa.
- Se presenta una estrategia para la evasión de obstáculos mediante el balance de las magnitudes del flujo óptico entre las ventanas de la imagen.
- Se implementó el sistema de navegación dentro de un robot rodante controlado por una computadora con capacidades de procesamiento limitadas como la *Raspberry Pi* lo que permite su fácil replicación.
- Se diseñó una interfaz gráfica que permita realizar futuras experimentaciones bajo diferentes parámetros iniciales de una manera más sencilla.

5.4 Trabajos futuros

- El sistema de navegación opera en dos dimensiones, por lo que es aplicable a robots móviles terrestres con ruedas. Sería interesante trabajar la tercera dimensión para tener control sobre el eje fuera del plano (z), lo que permitiría al sistema ser aplicable en robots aéreos y acuáticos.
- El sistema de navegación se ve limitado a trabajar el proceso de navegación hasta 7 veces por segundo. Se podría aumentar el número de veces que se realiza el proceso mediante el cómputo en paralelo de dos o más *Raspberry Pi* u otra placa de desarrollo. Asimismo, se podrían optimizar otras tareas del sistema de navegación como el manejo de los elementos de la interfaz gráfica lo cual podría aumentar la velocidad de procesamiento de cada par de fotogramas.
- El sistema de navegación funciona con una cámara RGB. Por lo que como trabajo futuro se podría hacer uso de otro tipo de sensores cámaras de visión nocturna, cámara gran angular, cámaras IR o cámaras binoculares y poder evaluar el sistema bajo entornos diferentes a los utilizados en este proyecto.

5.5 Actividades académicas adicionales

Como parte de la difusión de este proyecto se realizaron las siguientes actividades académicas:

- Se presentó el artículo “Algoritmos de flujo óptico en imágenes de baja resolución para la navegación robótica” en la 8ª Jornada de Ciencia y Tecnología Aplicada llevada a cabo de manera virtual de los días 25 al 27 de mayo de 2022 organizado por el TecNM/CENIDET.

- Se presentó el poster “Visión robótica de superbaja resolución” en la Escuela de Inteligencia Computacional y Robótica 2022 llevada a cabo de los días 16 al 20 de agosto en la Universidad Tecnológica Emiliano Zapata en Emiliano Zapata, Morelos.
- Se presentó el poster “Visión robótica de superbaja resolución” en la 9ª Jornada de Ciencia y Tecnología Aplicada llevada a cabo de los días 16 al 18 de noviembre de 2022 en las instalaciones del TecNM/CENIDET.

Las constancias de las participaciones realizadas se muestran en el anexo C.

Referencias

- Agrawal, P., Ratnoo, A., & Ghose, D. (2017). Inverse optical flow based guidance for UAV navigation through urban canyons. *Aerospace Science and Technology*, 68(May), 163–178. <https://doi.org/10.1016/j.ast.2017.05.012>
- Back, S., Cho, G., Oh, J., Tran, X. T., & Oh, H. (2020). Autonomous UAV Trail Navigation with Obstacle Avoidance Using Deep Neural Networks. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 100(3–4), 1195–1211. <https://doi.org/10.1007/s10846-020-01254-5>
- Badrloo, S., Varshosaz, M., Pirasteh, S., & Li, J. (2022). Image-Based Obstacle Detection Methods for the Safe Navigation of Unmanned Vehicles: A Review. *Remote Sensing*, 14(15), 1–26. <https://doi.org/10.3390/rs14153824>
- Baillieul, J., & Kang, F. (2021). *Visual Navigation with a 2-pixel Camera---Possibilities and Limitations*. <http://arxiv.org/abs/2103.00285>
- Baker, S., Scharstein, D., Lewis, J. P., Roth, S., Black, M. J., & Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1), 1–31. <https://doi.org/10.1007/s11263-010-0390-2>
- Barrios Arce, J. I. (2019, July 26). *La matriz de confusión y sus métricas – Inteligencia Artificial* –. <https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/>
- Barron, J. L., & Thacker, N. a. (2005). Tutorial: Computing 2D and 3D optical flow. *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester, 2004*, 1–12. <http://www.tina-vision.net/docs/memos/2004-012.pdf>
- Barrows, G. L., Chahl, J. S., & Srinivasan, M. V. (2002). Biomimetic Visual Sensing and Flight Control Australian Defence Science and Technology Organization Author Biographies: *Bristol UAV Conference*, 1–15. <https://pdfs.semanticscholar.org/8588/928d4bc30c57d390c3107d672e924ffeed06.pdf>
- Basak, K., Manjunatha, M., & Kumar Dutta, P. (2012). Pyramidal Refinement of Lucas Kanade Optical Flow based Tracking of Peripheral Air Embolism in OCT Contrast Imaging. *International Journal of Computer Applications*, 52(12), 7–12. <https://doi.org/10.5120/8252-1777>
- Bermudez, F. G., & Fearing, R. (2009). Optical flow on a flapping wing robot. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, 5027–5032. <https://doi.org/10.1109/IROS.2009.5354337>
- Bernardo, C., Augusto, Z., & Pedro, M. (2019). Caracterización de sensor de flujo óptico para aplicación en navegación integrada. *V Jornadas de Investigación, Transferencia y Extensión de La Facultad de Ingeniería (La Plata, 2019)*., 50–58.
- Bonin-Font, F., Ortiz, A., & Oliver, G. (2008). Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 53(3), 263–296. <https://doi.org/10.1007/s10846-008-9235-4>
- Chirarattananon, P. (2018). A direct optic flow-based strategy for inverse flight altitude estimation with monocular vision and IMU measurements. *Bioinspiration and Biomimetics*, 13(3), aaa2be. <https://doi.org/10.1088/1748-3190/aaa2be>
- Conradt, J. (2015). On-board real-time optic-flow for miniature event-based vision sensors. *2015 IEEE International Conference on Robotics and Biomimetics, IEEE-ROBIO 2015*, 1858–1863. <https://doi.org/10.1109/ROBIO.2015.7419043>
- Conroy, J., Gremillion, G., Ranganathan, B., & Humbert, J. S. (2009). Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. *Autonomous Robots*, 27(3), 189–198. <https://doi.org/10.1007/s10514-009-9140-0>
- Dai, B., & Li, W. (2015). Obstacle Avoidance Based on Optical Flow for Mobile Robots in Unknown Environment. *Proceedings - 2014 7th International Symposium on*

- Computational Intelligence and Design, ISCID 2014*, 2, 599–602. <https://doi.org/10.1109/ISCID.2014.286>
- de la Rosa, R., Guerrero, J. M., & Lenero, J. A. (2020). A real-time bio-inspired optical flow detection system based on a low resolution image sensor. *White Paper*, 2–7.
- de la Rosa, R., Guerrero, J. M., & Leñero, J. A. (2020). Live demonstration: A tracking system based on a real-time bio-inspired optical flow sensor. *Proceedings - IEEE International Symposium on Circuits and Systems, 2020-October*, 1–1. <https://doi.org/10.1109/iscas45731.2020.9181258>
- Duhamel, P. E. J., Pérez-Arancibia, N. O., Barrows, G. L., & Wood, R. J. (2013). Biologically inspired optical-flow sensing for altitude control of flapping-wing microrobots. *IEEE/ASME Transactions on Mechatronics*, 18(2), 556–568. <https://doi.org/10.1109/TMECH.2012.2225635>
- Editya, A. S., Ahmad, T., & Studiawan, H. (2022). Direction Estimation of Drone Collision Using Optical Flow for Forensic Investigation. *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*, 1–6.
- Granillo, O. D. M., & Beltran, Z. Z. (2018). Real-Time Drone (UAV) Trajectory Generation and Tracking by Optical Flow. *Proceedings - 2018 International Conference on Mechatronics, Electronics and Automotive Engineering, ICMEAE 2018*, 38–43. <https://doi.org/10.1109/ICMEAE.2018.00014>
- Green, W. E., Oh, P. Y., & Barrows, G. (2004). Flying insect inspired vision for autonomous aerial robot maneuvers in near-earth environments. *Proceedings - IEEE International Conference on Robotics and Automation, 2004(3)*, 2347–2352. <https://doi.org/10.1109/robot.2004.1307412>
- Horn, B., & Schunck, B. (1981). Determining optical flow. *Computer Vision*, 17(572), 185–203. <https://doi.org/10.7551/mitpress/1413.003.0014>
- Huang, L., & Song, J. (2017). Selection of the optical flow algorithms for flying vehicles with high speed. *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017*, 5414–5418. <https://doi.org/10.1109/CCDC.2017.7979459>
- Irani, M., & Anandan, P. (1999). About direct methods. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1883, 267–277. https://doi.org/10.1007/3-540-44480-7_18
- Karoly, A. I., Elek, R. N., Haidegger, T., Szell, K., & Galambos, P. (2019). Optical flow-based segmentation of moving objects for mobile robot navigation using pre-trained deep learning models. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics, 2019-October*, 3080–3086. <https://doi.org/10.1109/SMC.2019.8914359>
- Li, H., Hu, Z., & Chen, X. (2016). Robust monocular visual odometry using optical flows for mobile robots. *Chinese Control Conference, CCC, 2016-Augus*, 6003–6007. <https://doi.org/10.1109/ChiCC.2016.7554300>
- Liyanage, D. K., & Perera, M. U. S. (2012). Optical flow based obstacle avoidance for the visually impaired. *BEIAC 2012 - 2012 IEEE Business, Engineering and Industrial Applications Colloquium*, 284–289. <https://doi.org/10.1109/BEIAC.2012.6226068>
- Lu, Y., Xue, Z., Xia, G. S., & Zhang, L. (2018). A survey on vision-based UAV navigation. *Geo-Spatial Information Science*, 21(1), 21–32. <https://doi.org/10.1080/10095020.2017.1420509>
- Lucas, B. D., & Kanade, T. (1981). *Iterative Image Registration Technique With an Application To Stereo Vision*. 2, 674–679.
- McGuire, K., De Croon, G., De Wagter, C., Tuyls, K., & Kappen, H. (2017). Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone. *IEEE Robotics and Automation Letters*, 2(2), 1070–1076.

- <https://doi.org/10.1109/LRA.2017.2658940>
- Meneses, M. C., Matos, L. N., & Prado, B. O. (2018). *Low-cost Autonomous Navigation System Based on Optical Flow Classification*. <http://arxiv.org/abs/1803.03966>
- Moya-Albor, E., Coronel, S. L., Ponce, H., Brieva, J., Chavez-Dominguez, R., & Guadarrama-Munoz, A. E. (2019). Bio-inspired optical flow-based autonomous obstacle avoidance control. *Proceedings - 2019 International Conference on Mechatronics, Electronics and Automotive Engineering, ICMEAE 2019*, 18–23. <https://doi.org/10.1109/ICMEAE.2019.00011>
- Nadour, M., Boumehraz, M., Cherroun, L., & Puig, V. (2019). Mobile robot visual navigation based on fuzzy logic and optical flow approaches. *International Journal of Systems Assurance Engineering and Management*, 10(6), 1654–1667. <https://doi.org/10.1007/s13198-019-00918-2>
- Navarrete, L. O. (2019). *Sistema de Navegación Inercial Asistido por Visión para Robots Móviles Terrestres*. [Tesis de Maestría, Centro Nacional de Investigación y Desarrollo Tecnológico].
- Park, S., Lee, K., Song, H., Cho, J., Park, S. Y., & Yoon, E. (2019). Low-power, bio-inspired time-stamp-based 2-d optic flow sensor for artificial compound eyes of micro air vehicles. *IEEE Sensors Journal*, 19(24), 12059–12068. <https://doi.org/10.1109/JSEN.2019.2938559>
- Ponce, H., Brieva, J., & Moya-Albor, E. (2018). Distance Estimation Using a Bio-Inspired Optical Flow Strategy Applied to Neuro-Robotics. *Proceedings of the International Joint Conference on Neural Networks, 2018-July*, 1–7. <https://doi.org/10.1109/IJCNN.2018.8489597>
- Rahmani, B., Putra, A. E., Harjoko, A., & Priyambodo, T. K. (2015). Review of Vision-Based Robot Navigation Method. *IAES International Journal of Robotics and Automation (IJRA)*, 4(4), 254. <https://doi.org/10.11591/ijra.v4i4.pp254-261>
- Raudies, F. (2013). Optic flow. *Scholarpedia*, 8(7), 30724. <https://doi.org/10.4249/SCHOLARPEDIA.30724>
- Sabo, C., Chisholm, R., Petterson, A., & Cope, A. (2017). A lightweight, inexpensive robotic system for insect vision. *Arthropod Structure and Development*, 46(5), 689–702. <https://doi.org/10.1016/j.asd.2017.08.001>
- Sanahuja, G., Valera, A., Sánchez, A. J., Ricolfe-Viala, C., Vallés, M., & Marín, L. (2011). Control embebido de robots móviles con recursos limitados basado en flujo óptico. *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, 8(3), 250–257. <https://doi.org/10.1016/j.riai.2011.06.012>
- Sanchez, A. J., Rios, H. V., Marin, A., & Contreras, G. (2015). Decision making for obstacle avoidance in autonomous mobile robots by time to contact and optical flow. *25th International Conference on Electronics, Communications and Computers, CONIELECOMP 2015*, 130–134. <https://doi.org/10.1109/CONIELECOMP.2015.7086939>
- Serres, J. R., & Ruffier, F. (2017). Optic flow-based collision-free strategies: From insects to robots. *Arthropod Structure and Development*, 46(5), 703–717. <https://doi.org/10.1016/j.asd.2017.06.003>
- Serres, J. R., & Viollet, S. (2018). Insect-inspired vision for autonomous vehicles. *Current Opinion in Insect Science*. <https://doi.org/10.1016/j.cois.2018.09.005>
- Skelton, P. S. M., Finn, A., & Brinkworth, R. S. A. (2019). Consistent estimation of rotational optical flow in real environments using a biologically-inspired vision algorithm on embedded hardware. *Image and Vision Computing*, 92, 103814. <https://doi.org/10.1016/j.imavis.2019.09.005>
- Soria, C., & Carelli, R. (2006). Control de un robot móvil utilizando el flujo óptico obtenido

- través de un sistema omnidireccional catadióptrico. *IV Jornadas Argentinas de Robótica*, 1109, 25–36.
- Souhila, K., & Karim, A. (2007). Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1), 13–16. <https://doi.org/10.5772/5715>
- Stankovic, J. A., & Ramamritham, K. (1990). What is predictability for real-time systems? *Real-Time Systems*, 2(4), 247–254. <https://doi.org/10.1007/BF01995673>
- Steinberg, D. (2015). *CART: Classification and Regression Trees*. December.
- Tai, L., Li, S., & Liu, M. (2016). A deep-network solution towards model-less obstacle avoidance. *IEEE International Conference on Intelligent Robots and Systems, 2016-Novem*, 2759–2764. <https://doi.org/10.1109/IROS.2016.7759428>
- Tu, Z., Xie, W., Zhang, D., Poppe, R., Veltkamp, R. C., Li, B., & Yuan, J. (2019). A survey of variational and CNN-based optical flow techniques. *Signal Processing: Image Communication*, 72(October 2018), 9–24. <https://doi.org/10.1016/j.image.2018.12.002>
- Vela, V. V. (2017). *Experimentación con Odometría Visual por Métodos Directos*. [Tesis de Maestría, Centro Nacional de Investigación y Desarrollo Tecnológico].
- Velázquez, C. F. D. (2019). *Odometría mediante visión artificial usando métodos directos*. [Tesis de Maestría, Centro Nacional de Investigación y Desarrollo Tecnológico].
- Wang, Z., Wang, B., Tang, C., & Xu, G. (2020). Pose and Velocity Estimation Algorithm for UAV in Visual Landing. *Chinese Control Conference, CCC, 2020-July(1)*, 3713–3718. <https://doi.org/10.23919/CCC50068.2020.9188491>
- Yadav, N. D., & Khandekar, S. (2018). Overview of Optical Flow Technique for Mobile Robot Obstacle Avoidance. *Proceedings of the 3rd International Conference on Communication and Electronics Systems, ICCES 2018, Icces*, 619–622. <https://doi.org/10.1109/CESYS.2018.8723883>

Anexo A. Algoritmos de clasificación

En esta sección se describen a detalle los algoritmos de clasificación utilizados para la estimación del estado del robot.

Para este trabajo se seleccionaron los algoritmos relacionados con los árboles de decisión dado que se utilizan valores que representan el desplazamiento de los píxeles en la imagen con respecto al movimiento de un robot rodante, se vio la necesidad obtener reglas a partir de los datos del cálculo del flujo óptico para definir el estado del robot, entre los que están el CART, *AdaBoost* y *Random Forest*).

Arboles de decisión ¹¹

Los árboles de decisión (*Decision Trees*) son un método de aprendizaje supervisado no paramétrico que se utiliza para la clasificación y la regresión. El objetivo es crear un modelo que prediga el valor de una variable objetivo mediante el aprendizaje de reglas de decisión simples inferidas a partir de las características de los datos. Un árbol puede verse como una aproximación constante a trozos.

Un árbol de decisión consiste en nodos que forman un árbol enraizado, lo que significa que es un árbol dirigido con un nodo llamado "raíz" que no tiene aristas entrantes. Todos los demás nodos tienen exactamente una arista entrante. Un nodo con aristas salientes se llama nodo interno o de prueba. Todos los demás nodos se denominan hojas (también conocidos como nodos terminales o de decisión). En un árbol de decisión, cada nodo interno divide el espacio de la instancia en dos o más subespacios según una determinada función discreta de los valores de los atributos de entrada. En el caso más sencillo y frecuente, cada prueba considera un único atributo, de modo que el espacio de instancias se divide en función del valor del atributo. En el caso de los atributos numéricos, la condición se refiere a un rango. Cada hoja se asigna a una clase que representa el valor objetivo más adecuado. Alternativamente, la hoja puede contener un vector de probabilidad que indica la probabilidad de que el atributo objetivo tenga un determinado valor. Las instancias se clasifican de la raíz del árbol a una hoja, en función del resultado de las pruebas a lo largo del camino.

CART ¹²

El árbol de decisión CART es un procedimiento de partición recursiva binaria capaz de procesar atributos continuos y nominales como objetivos y predictores. Los datos se manejan en su forma bruta; no se requiere ni se recomienda ningún tipo de clasificación. A partir del nodo raíz, los datos se dividen en dos hijos, y cada uno de los hijos se divide a su vez en nietos. Los árboles crecen hasta un tamaño máximo sin utilizar una regla de parada;

¹¹ "Decision Trees — scikit-learn 1.1.1 documentation." <https://scikit-learn.org/stable/modules/tree.html>

¹² "Decision Trees — scikit-learn 1.1.1 documentation." <https://scikit-learn.org/stable/modules/tree.html>

el proceso de crecimiento del árbol se detiene cuando no es posible realizar más divisiones debido a la falta de datos (Steinberg, 2015).

CART es muy similar a algoritmo C4.5, pero difiere en que admite variables objetivo numéricas (regresión) y no calcula conjuntos de reglas. CART construye árboles binarios utilizando la característica y el umbral que producen la mayor ganancia de información en cada nodo. La librería *scikit-learn* utiliza una versión optimizada del algoritmo CART; sin embargo, la implementación de *scikit-learn* no admite variables categóricas. Los parámetros utilizados para el entrenamiento fueron los definidos por defecto (criterio de división “*gini*”, estrategia para división “*best*” y máxima profundidad “*None*”).

Adaboost¹³

Un clasificador AdaBoost es un meta-estimador que comienza ajustando un clasificador en el conjunto de datos original y luego ajusta copias adicionales del clasificador en el mismo conjunto de datos, pero donde los pesos de las instancias clasificadas incorrectamente se ajustan de manera que los clasificadores posteriores se centran más en los casos difíciles. Los parámetros utilizados para el entrenamiento fueron los definidos por defecto (número máximo de estimadores 50, algoritmo “*SAMME.R*” y tasa de aprendizaje 1).

Random Forest¹⁴

Es un metaestimador que ajusta un número de clasificadores de árboles de decisión en varias submuestras del conjunto de datos y utiliza el promedio para mejorar la precisión predictiva y controlar el sobreajuste.

Cada árbol es construido usando el siguiente algoritmo:

- Sea N el número de casos de prueba, M es el número de variables en el clasificador.
- Sea m el número de variables de entrada a ser usado para determinar la decisión en un nodo dado; m debe ser mucho menor que M .
- Elegir un conjunto de entrenamiento para este árbol y usar el resto de los casos de prueba para estimar el error.
- Para cada nodo del árbol, elegir aleatoriamente m variables en las cuales basar la decisión. Calcular la mejor partición del conjunto de entrenamiento a partir de las m variables.

Para la predicción un nuevo caso es empujado hacia abajo por el árbol. Luego se le asigna la etiqueta del nodo terminal donde termina. Este proceso es iterado por todos los árboles en el ensamblado, y la etiqueta que obtenga la mayor cantidad de incidencias es reportada como la predicción. Los parámetros utilizados para el entrenamiento fueron los definidos por defecto (número de árboles 100, criterio de división “*gini*” y máxima profundidad “*None*”).

¹³ “sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.1.1 documentation.” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

¹⁴ “sklearn.ensemble.RandomForestClassifier — scikit-learn 1.1.1 documentation.” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

Anexo B. Construcción del robot rodante

En este anexo se explica a detalle la construcción del robot rodante para las experimentaciones de este proyecto.

El propósito de la construcción de este robot es el poder realizar las pruebas de funcionamiento del sistema de navegación robótica propuesto. La elección de la construcción fue por su fácil construcción y su capacidad de soportar el peso de la *Raspberry Pi* junto con la *PowerBank* que la alimenta con el voltaje adecuado para su funcionamiento.

Componentes utilizados

A continuación, se presenta la lista de componentes del robot rodante:

Raspberry Pi 3B +: Computadora de placa simple (SBC por sus siglas en inglés) de bajo costo desarrollado en el Reino Unido por la *Raspberry Pi Foundation*. Por su pequeño tamaño y su bajo costo se seleccionó para realizar el proceso de obtención del flujo óptico a través del módulo de cámara v2. Sus especificaciones son: CPU + GPU: *Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz*. RAM: 1GB LPDDR2 SDRAM. Tiene *Wi-Fi* y *Bluetooth: 2.4GHz y 5GHz IEEE 802.11*.

Módulo cámara v2 para Raspberry: Consta de un sensor OV5647 de 5MP (2592×1944) capaz de grabar video en hasta 1920×1080 (Full HD) con un ángulo de visión 54 ° en horizontal y 44 ° en vertical. Está conectada a la *Raspberry* por medio de un cable de 15 pines al puerto dedicado para la cámara.

ESP32: Es un microcontrolador de una familia de chips de bajo costo y consumo de energía. Fue seleccionado para el control del puente H L298N y a su vez a los motorreductores de las llantas del robot móvil por su pequeño tamaño y su bajo costo.

Puente H (L298N): Es un dispositivo que proporciona la corriente necesaria para el control de motores para aplicaciones de robótica. Se utiliza para el control de los motorreductores de las llantas del robot móvil.

1 pila de respaldo (Powerbank) de 5000mah con dos salidas de 5v: Para la alimentación de la *Raspberry Pi* y el ESP32.

2 pilas conectadas en serie de Li ion (3.7v descarga y 4.2v cargada) con 2200mah: Para la alimentación del puente H que a su vez alimenta los motorreductores. Los motorreductores son alimentados aparte dado que el voltaje proporcionado por la *powerbank* (5v) no es suficiente para que los motorreductores funcionen con el peso del robot.

4 motorreductores: cuenta con engranajes de plástico y opera con un voltaje entre 3 y 12V.

4 llantas: utilizados para proyectos de *Arduino* con un diámetro de 66 mm y un ancho de 26 mm.

Soporte para la cámara: Conta de una base de plástico y dos servomotores los cuales permiten de la cámara gire de 0° a 180° sobre el eje vertical y horizontal. En este caso fueron utilizados para permitir realizar pruebas del funcionamiento con la cámara en diferentes ángulos en el eje horizontal y para poder replicar dicho ángulo en los diferentes experimentos.

Ubicación y conexión de los componentes

El robot consta de dos placas de acrílico donde los componentes utilizados se ubican en la parte superior o inferior de robot. En la figura B.1 se muestra la ubicación de los componentes:

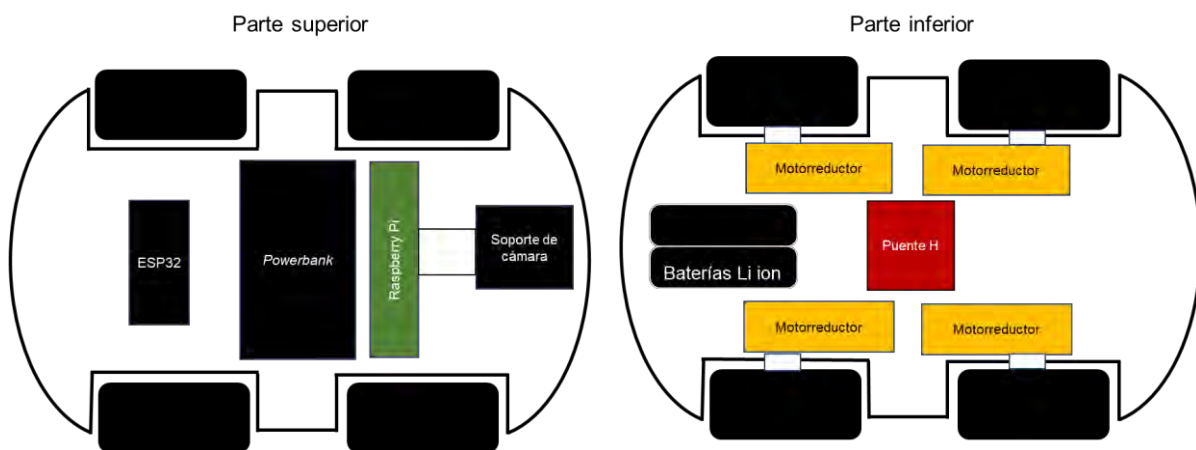


Figura B.1 Ubicación de los componentes del robot rodante

El módulo de cámara está conectado mediante un cable de 15 pines al puerto dedicado para la cámara. La *Raspberry Pi* que tiene todo el sistema de navegación está conectado de los pines 33,35 y 37 a los pines 5,18 y 19 del ESP32 respectivamente para mandar instrucciones al robot rodante y para la creación de los *datasets* propios se tiene una conexión de los pines 11, 13 y 15 de la *Raspberry Pi* a los pines 21,22 y 23 del ESP32 respectivamente.

Para la creación de los *datasets* se hizo uso de la comunicación *Bluetooth* del ESP32 a la aplicación móvil *Arduino Bluetooth Robot Car* (figura B.2) el cual enviaba instrucciones al puente H de acuerdo movimiento seleccionado o activaba el sistema de navegación mediante una señal a la *Raspberry Pi*.

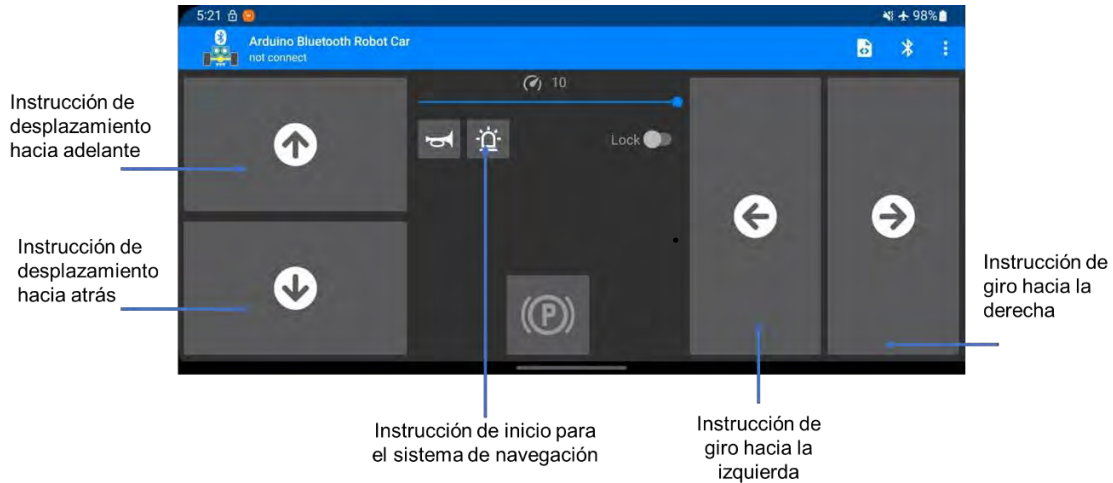


Figura B.2 Interfaz de la aplicación Arduino Bluetooth Robot Car

Una vez que ESP32 recibe una instrucción, esta es enviada al puente H el cual controla los motorreductores, esta conexión se hace desde los pines 25, 26, 27 y 14 del ESP32 a las entradas 1, 2, 3 y 4 del puente H respectivamente. Adicionalmente existe una conexión de los pines 14 y 33 a las entradas EN1 y EN2 respectivamente donde mediante una señal analógica entre 0 y 255 se envía el valor de modulación por ancho de pulsos (PWM) que controla la velocidad del robot.

Comunicación entre ESP32 y Raspberry Pi

Con las conexiones mencionadas en el punto anterior se implementó una clave de comunicación mediante números en el sistema binario para enviar o recibir instrucciones. La interpretación de estas señales se muestra en la tabla B.1.

Tabla B.1 Interpretación de las señales de comunicación de entrada y salida de la Raspberry Pi al ESP32

Señales de comunicación de salida				Señales de comunicación de entrada			
Pin 37	Pin 35	Pin 33	Instrucción enviada	Pin 15	Pin 13	Pin 11	Instrucción recibida
0	0	1	Detenerse	0	0	1	Sin movimiento
0	1	0	Ir hacia adelante	0	1	0	Desplazamiento hacia adelante
0	1	1	Ir hacia atrás	0	1	1	Desplazamiento hacia atrás
1	0	0	Girar a la derecha	1	0	0	Giro a la derecha
1	0	1	Girar a la izquierda	1	0	1	Giro a la izquierda
				1	1	0	Inicio del sistema de navegación

Funcionamiento del robot

Una vez que el ESP32 recibe instrucciones ya sea por medio del sistema de navegación o del control remoto se ejecuta las siguientes instrucciones por los motorreductores:

- **Detenerse:** Los cuatro motorreductores no reciben ninguna instrucción y se asigna un valor PWM de 0.
- **Desplazamiento hacia adelante:** Los cuatro motorreductores van hacia adelante con un PWM de 180.
- **Desplazamiento hacia atrás:** Los cuatro motorreductores van hacia atrás con un PWM de 180.
- **Giro a la derecha:** Los dos motorreductores del lado izquierdo van hacia adelante mientras que los dos motorreductores del lado derecho van en sentido contrario. Los cuatro motorreductores van con un PWM de 190.
- **Giro a la izquierda:** Los dos motorreductores del lado derecho van hacia adelante mientras que los dos motorreductores del lado izquierdo van en sentido contrario. Los cuatro motorreductores van con un PWM de 190.

Estas instrucciones fueron programadas en C++ adaptado para Arduino y la ESP32.

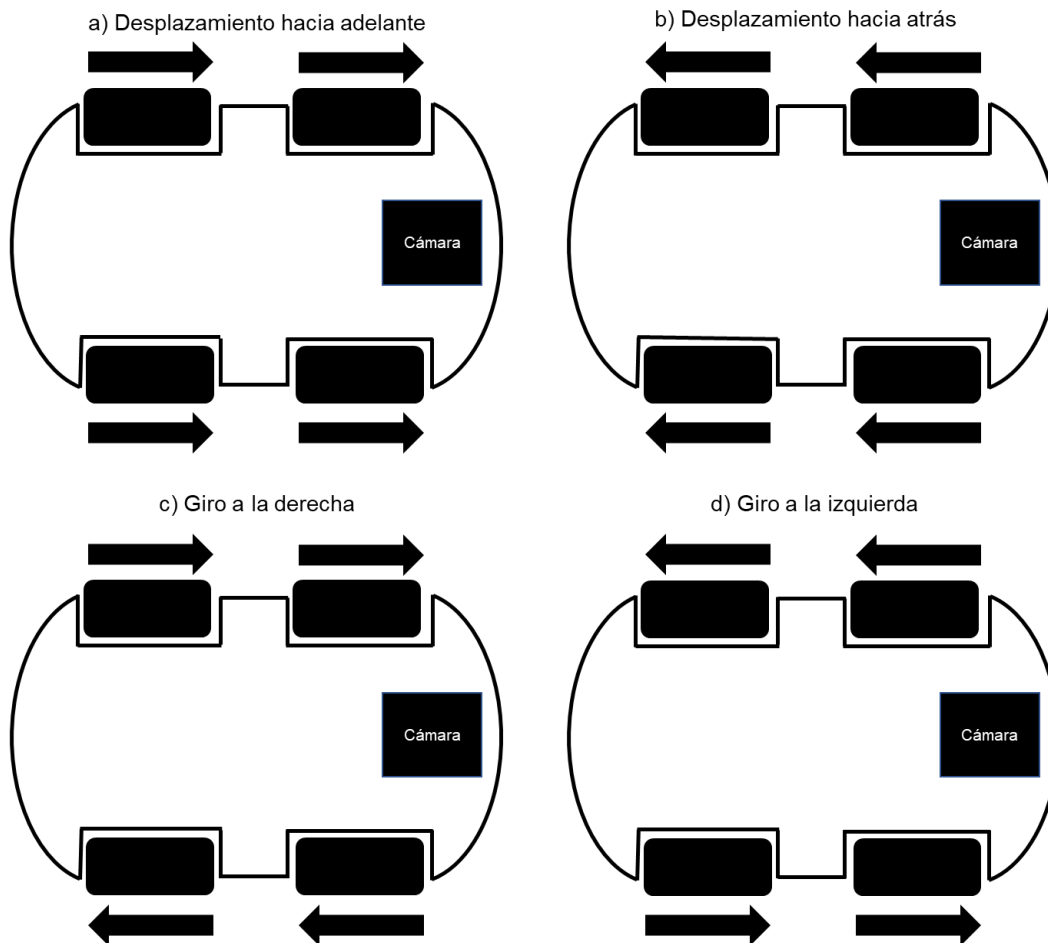


Figura B.3 Comportamiento de los motorreductores del robot en las diferentes instrucciones

Anexo C. Constancias de participaciones



Figura C.1 Constancia de presentación de artículo en la 8ª Jornada de Ciencia y Tecnología Aplicada



Figura C.2 Constancia de presentación de poster en la Escuela de Inteligencia Computacional y Robótica 2022



Figura C.3 Constancia de presentación de poster en la 9ª Jornada de Ciencia y Tecnología Aplicada