UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo e implementación de algoritmos de visión por computadora clásicos empleando OpenCV en sistemas embebidos.

Trabajo de graduación presentado por Hector Alejandro Klée González
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

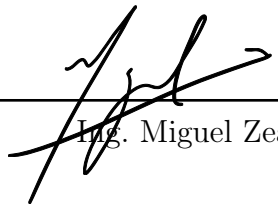
Desarrollo e implementación de algoritmos de visión por computadora clásicos empleando OpenCV en sistemas embebidos.

Trabajo de graduación presentado por Hector Alejandro Klée González
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

Vo.Bo.:

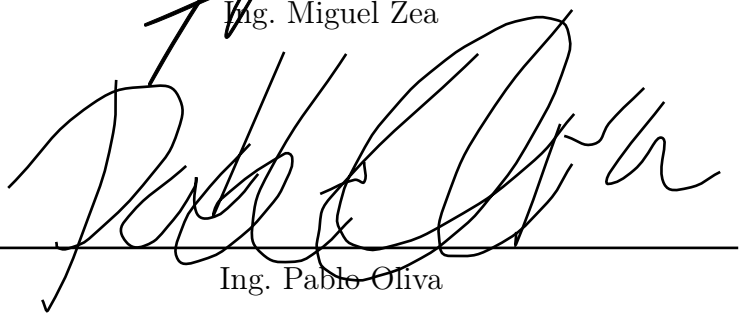
(f) 

Ing. Miguel Zea

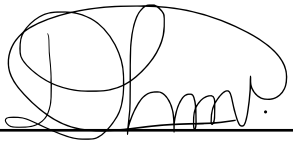
Tribunal Examinador:

(f) 

Ing. Miguel Zea

(f) 

Ing. Pablo Oliva

(f) 

Ing. Diego Alberto Morales

Fecha de aprobación: Guatemala, 06 de enero de 2022.

Prefacio

Me gustaría agradecer a mis padres, Evelyn y Héctor, quiénes han sido un gran apoyo en todos los aspectos de mi vida. Siempre conmigo en las buenas y en las malas, motivándome a tratar de ser mi mejor versión. Sin ellos, ninguno de los logros que he alcanzado pudieron haberse hecho realidad. Gracias también a mi hermana, María José, por aguantar mis traspiés y brindar siempre una cara relajada en mis días más tensos. Hago una dedicatoria especial a mi abuela María Angelina, por ti va este trabajo, gracias por todos los momentos que pasamos juntos, gracias por las experiencias que solo una persona tan valiente como tú puede llegar a brindar. A ellos, al resto de mi familia cercana y a mis amigos, les dedico mis logros académicos.

Este trabajo se logró gracias al Ingeniero Miguel Zea, por su apoyo que, a pesar de las circunstancias vividas en el último año, siempre buscó formas de brindar ayuda y motivación para lograr un trabajo que refleja los valores de la Universidad del Valle de Guatemala.

Índice

Prefacio v

Lista de figuras xii

Lista de cuadros xiii

Resumen xv

Abstract xvii

1. Introducción 1

2. Antecedentes 3
- 2.1. JeVois Smart Machine Vision Camera . . . 3
- 2.2. ESP32-CAM . . . 4
- 2.3. OpenMV Cam H7 . . . 4
- 2.4. Pixy . . . 4
- 2.5. Arduino Camera . . . 5
- 2.6. OV7670 CameraChip Sensor . . . 5
- 2.7. Raspberry Pi High Quality Camera . . . 5
- 2.8. Familia TDA3X de Texas Instruments . . . 5
- 2.9. Control basado en visión (*visual servoing*) . . . 6

3. Justificación 7

4. Objetivos 9
- 4.1. Objetivo general . . . 9
- 4.2. Objetivos específicos . . . 9

5. Alcance 11

6. Marco teórico 13
- 6.1. Visión por computadora . . . 13
- 6.2. Formación de imagen . . . 13

6.2.1. Primitivos geométricos y transformaciones . . . 13
6.3. Modelos de cámara y calibración . . . 21
6.3.1. Modelo de cámara . . . 21
6.3.2. Calibración . . . 24
6.4. Procesamiento de imágenes . . . 26
6.4.1. Suavizado . . . 26
6.4.2. Algoritmo de relleno por difusión (*Flood Fill*) . . . 27
6.4.3. Pirámide de imágenes . . . 27
6.4.4. *Threshold* . . . 28
6.5. Transformadas de imagen . . . 28
6.5.1. Convolución . . . 28
6.5.2. Gradientes y derivadas de Sobel . . . 29
6.5.3. Laplace . . . 29
6.5.4. Detector de bordes de Canny . . . 30
6.5.5. Transformada de Hough . . . 31
6.6. Detección de características y emparejamiento . . . 32
6.6.1. Segmentación . . . 33
6.7. Marcadores de referencia . . . 34
6.8. Control basado en visión . . . 35
6.8.1. Control basado en imágenes . . . 35

7. Metodología **39**
7.1. OpenCV . . . 39
7.2. *Software* . . . 40
7.2.1. *Software* para JeVois Smart Vision Camera . . . 40
7.2.2. *Software* para ESP32-CAM . . . 40
7.3. Herramienta de comparación . . . 41
7.3.1. Parámetros de estudio y pesos asignados . . . 41
7.3.2. Calificaciones . . . 41
7.4. Integración de brazo robot . . . 42
7.4.1. Estructura de brazo robot . . . 42
7.4.2. Controlador . . . 42

8. Algoritmos de visión por computadora **43**
8.1. Threshold . . . 43
8.1.1. Threshold binario . . . 43
8.1.2. Threshold binario invertido . . . 44
8.1.3. Threshold truncado . . . 44
8.1.4. Threshold a cero . . . 45
8.1.5. Threshold a cero invertido . . . 46
8.1.6. Threshold adaptativo a la media . . . 46
8.1.7. Threshold adaptativo Gaussiano . . . 47
8.2. Detector de bordes de Canny . . . 47
8.3. Detector de contornos . . . 48
8.4. Transformada de Hough para líneas . . . 49
8.5. Transformada de Hough para círculos . . . 50
8.6. Detector de marcadores ArUco . . . 51

9. Comparación de módulos de visión **53**

10.Control basado en visión **57**
10.1. Brazo robot 57
10.2. Planteamiento general 57
10.3. Controlador PD 58
10.4. Controlador basado en imágenes 59

11.Conclusiones **63**

12.Recomendaciones **65**

13.Bibliografía **67**

14.Anexos **69**
14.1. Comparación de desempeño de algoritmos 69
14.2. Control basado en visión 69

15.Glosario **71**

Lista de figuras

1. Transformaciones 2D básicas [15]. 14
2. Resumen de transformaciones 2D [15]. 15
3. Resumen de transformaciones 3D [15]. 16
4. Representación de cuaterniones unitarios [15]. 17
5. Algoritmo SLERP [15]. 18
6. Proyecciones de 3D a 2D [15]. 19
7. Modelo de cámara *pinhole* [14]. 21
8. Modelo modificado de cámara *pinhole* [14]. 22
9. Tipos de distorsiones de lente [14]. 23
10. Representación de una homografía [14]. 24
11. Ejemplos de suavizado [14]. 26
12. Ejemplos de *Flood Fill* [14]. 27
13. Representación de una pirámide de imágenes [15]. 28
14. Tipos de *threshold* en *OpenCV* y sus efectos [14]. 29
15. Comparación de *thresholds* [14]. 30
16. Derivadas de Sobel aplicadas en diferentes dimensiones [14]. 31
17. Implementación del detector de bordes de Canny [14]. 31
18. Implementación de la transformada de Hough [14]. 32
19. Segmentación por cortes normalizados [15]. 34
20. Ejemplos de marcadores de referencia [20] 35
21. Aproximaciones para el control basado en visión [22]. 36

22. Implementación de *threshold* binario. 44
23. Implementación de *threshold* binario invertido. 44
24. Implementación de *threshold* truncado. 45
25. Implementación de *threshold* a cero. 45
26. Implementación de *threshold* a cero invertido. 46
27. Implementación de *threshold* adaptativo a la media. 47
28. Implementación de *threshold* adaptativo Gaussiano. 47
29. Implementación de detección de bordes de Canny. 48
30. Implementación de detección de contornos. 48
31. Implementación de transformada de Hough para líneas. 49
32. Implementación de transformada de Hough para círculos 1. 50

33. Implementación de transformada de Hough para círculos 2. 50
34. Implementación de detección de marcadores ArUco. 51

35. Vista frontal del brazo robot. 58
36. Vistas lateral y superior del brazo robot. 58

Lista de cuadros

1. Algoritmos de detección de características [16]. 32

2. Valores de interés para la herramienta de comparación. 41
3. Valores de interés para la herramienta de comparación. 42

4. Resultados del estudio aplicado a JeVois Smart Vision Camera. 54
5. Resultados del estudio aplicado a ESP32-CAM. 55

6. Parámetros de Denavit-Hartenberg. 59

Resumen

Este proyecto consistió en la selección de un módulo capaz de dotar de visión por computadora a un sistema embebido e implementarlo en una aplicación de control basado en visión (*visual servoing*).

Dado que el mercado de módulos con estas características es vasto, se decidió realizar un estudio comparativo (*trade study*) entre dos de ellos: JeVois Smart Vision Camera y ESP32-CAM. Las capacidades de cada módulo fueron medidas por medio de la implementación de algoritmos de visión por computadora clásicos, empleando la librería OpenCV. Los desempeños de cada módulo fueron ponderados realizar una decisión.

El módulo con mejores capacidades para la aplicación de control basado en visión fue la JeVois Smart Vision Camera. Esta fue integrada en un brazo robot de dos grados de libertad que realizaba rotaciones horizontales y verticales (*pan-tilt*). La idea general fue realizar un controlador que utilizara una señal proveniente de la cámara, como realimentación, para seguir un marcador ArUco.

Abstract

This project consisted in the selection of a module capable of endowing computer vision to an embedded system and implementing it in an visual servoing application.

Given that the market of modules with this characteristics is vast, two modules were selected to make a trade study: JeVois Smart Vision Camera and ESP32-CAM. The capabilities of each module were measured by the implementation of classic computer vision algorithms, using the OpenCV library. The performance of each module was the graded to make a decision.

The best suited module for the visual servoing application ended up being the JeVois Smart Vision Camera. It was integrated into a two degrees of freedom robotic arm capable or performing a pan-tilt action. The general idea was make a controller that used a signal coming from the camera, as feedback, to follow an ArUco marker.

CAPÍTULO 1

Introducción

Esta investigación se centra en el campo del control basado en visión, específicamente,por medio de la implementación de algoritmos clásicos de visión por computadora. Principalmente, se enfoca en la selección de un módulo óptimo capaz de dotar de visión por computadora a un sistema embebido para poder realizar una aplicación de control sencilla. La investigación se realizó con el interés que la Universidad del Valle de Guatemala se mantenga a la vanguardia en el campo de la Robótica al permitir iniciar investigaciones con técnicas de control flexibles, versátiles y modernas.

Para llevar a cabo el presente proyecto se siguieron dos etapas generales, selección del módulo de visión e implementación. En la primera etapa, se consideró opciones de módulos de visión para poder realizar una comparación que permitiera la selección de la opción más viable para la implementación posterior. Luego, con el módulo de visión seleccionado, se pasó a diseñar un sistema que acoplara el módulo, actuadores y microcontrolador. Por último, se desarrolló el *software* que permitiera realizar el control basado en visión.

CAPÍTULO 2

Antecedentes

La visión por computadora detecta características e información a partir de una imagen, la cual sirve como entrada para diversos algoritmos de reconocimiento de características. Por lo tanto, al dotar un sistema con visión se obtiene la flexibilidad de poder ajustarlo al ambiente en "tiempo real" para cumplir su tarea. La cantidad de sistemas que utilizan esta tecnología ha aumentado en los últimos años. En esta sección se presenta diversos módulos que permiten dotar de visión a un sistema embebido.

2.1. JeVois Smart Machine Vision Camera

Como lo menciona la página web del fabricante [1], la JeVois consiste en una cámara *open-source* que permite dotar de visión por computadora a proyectos realizados en diferentes plataformas, siendo ejemplos, PC, Arduino y Raspberry Pi.

La característica atractiva de la JeVois es la versatilidad que se puede conseguir con ella en un empaquetado pequeño. Según las especificaciones [1], consiste en un módulo de 28 cc y 17 g que incluye un sensor de vídeo, CPU quad-core, puertos serial y de vídeo por USB. La programación se realiza por medio de cargar los algoritmos de visión a una tarjeta de memoria micro SD.

Según la información recopilada por [2], el *firmware* de JeVois tiene las librerías de OpenCV 4.0, TensorFlow Lite, Darknet deep neural networks, DLib, etc. En [2] se presenta una comparación entre el desempeño de tres cámaras que dotan de visión por computadora a un sistema con el fin de verificar la sujeción de un objeto por medio de *machine learning*. La visón por medio de *machine learning* no es objeto de estudio para este documento, sin embargo, [2] presenta una idea de los usos de la JeVois y otros módulos que se presentarán más adelante.

2.2. ESP32-CAM

De acuerdo con la hoja de datos de la ESP32-CAM [3], este es un módulo de desarrollo que se basa en el ESP32 con una cámara integrada. Es una solución para proyectos del internet de las cosas (*Internet of Things*) y prototipos. La programación del módulo se realiza por medio del IDE de Arduino, la forma de integración de módulo con el *software* se describe en [4].

El módulo integra WiFi y Bluetooth, cumpliendo con estándares 802.11b/g/n/e/i y 4.2 respectivamente, con dos CPUs de 32 bits LX6 de alto rendimiento. Adopta una arquitectura de *pipeline* de siete etapas, distintos sensores de funcionamiento y un rango de frecuencias de funcionamiento desde 80MHz hasta 240MHz [3].

Por lo tanto, la ESP32-CAM puede utilizarse en modo de maestro para construir redes de controladores independientes, o bien, como esclavo de otro microcontrolador para agregar capacidad de red a otros dispositivos [3].

2.3. OpenMV Cam H7

La OpenMV Cam H7 [5], consiste en una cámara de visión por computadora, de bajo poder, la cual permite funciones de procesamiento de imágenes y redes neuronales. Por esta última, la OpenMV Cam H7 también se considera en la comparación de módulos realizada en [2]. Asimismo, [6] la define como un microcontrolador que permite la implementación de aplicaciones de visión en el mundo real. La programación del módulo se realiza en Python por medio de MicroPython, lo cual hace sencillo el manejo de las salidas de los algoritmos de visión por computadora, sin comprometer el control sobre la pines de entrada y salida de la cámara.

La OpenMV Cam tiene instalada una librería RPC, Remote Python/Procedure Call, la cual permite la conexión del microcontrolador con computadores, Raspberry Pi, Arduino o ESP8266/32.

2.4. Pixy

Pixy consiste en un sistema de visión pequeño, de fácil uso y bajo costo. Tiene la característica de utilizar un algoritmo de filtrado basado en color para la detección de colores. Según [7] el algoritmo de filtrado de color utilizado en Pixy permite tener una velocidad de 50 fps, es eficiente y robusto en la detección. El módulo detecta el color y saturación de cada pixel RGB de la imagen detectada y la utiliza para realizar un filtrado preliminar.

Tiene la capacidad de conectarse a Arduino, Raspeberry Pi, BeagleBone, etc. Dada la habilidad de conexión a estos controladores, las librerías respectivas son proveídas. Es compatible con los lenguajes C, C++ y Python.

2.5. Arduino Camera

Arduino Camera es un módulo que era originalmente exclusivo a Arduino, sin embargo, según [8] en la actualidad tiene compatibilidad con otros controladores como Raspberry Pi. Consiste en una cámara SPI de alta resolución, la cual se utiliza para aplicaciones de Internet de las Cosas (IoT), impresión 3D y robótica.

El módulo SPI de la cámara tiene designado una velocidad entre 4Mbps a 8Mbps para que haya compatibilidad con los voltajes de alimentación aceptados por Arduino u otros controladores compatibles.

2.6. OV7670 CameraChip Sensor

La hoja de datos del OV7670 CameraChip Sensor [9], la define como un dispositivo CMOS de bajo voltaje con las funciones de una cámara VGA y procesador de imágenes en un empaquetado compacto. Permite mostrar imágenes de 8 bits en diferentes formatos, controlado por medio de una interfaz SCCB (Serial Camera Control Bus). Además, es capaz de brindar al usuario control completo de la calidad, formato y salida de imagen operando hasta 30 fps en VGA.

2.7. Raspberry Pi High Quality Camera

La Raspberry Pi High Quality Camera es un accesorio para los todos los modelos actuales de la Raspberry Pi, excluyendo los modelos Zero. El resumen del módulo [10], indica que el empaquetado está compuesto de una placa con un sensor Sony IMX477, cable FPC (razón por la que los modelos Zero no son compatibles), una montura para lente con montura para trípode y anillo de ajuste, así como adaptadores para monturas C y CS. Todo esto resulta en una cámara de 12 megapixeles de alta resolución y sensibilidad.

Las aplicaciones que se le puede dar abarcan desde equipos de seguridad de alta calidad, hasta aplicaciones de óptica.

2.8. Familia TDA3X de Texas Instruments

La familia de componetes de TDA3X de Texas Instruments consiste en sistemas sobre chip (SoC por sus siglas en inglés) altamente optimizados, diseñados para sistemas de asistencia de controladores avanzados (ADAS) [11]. El uso de esta familia de componentes permite aplicaciones seguras en el campo de la automatización, pues integra bajo voltaje, un factor de forma pequeño y procesamiento analítico de visión ADAS. Como lo menciona [11], los SoC TDA3x permiten tecnología de visión embebida bajo una sola arquitectura.

Algunos de los módulos que se encontró de la familia TDA3X, además de su área de enfoque, se presentan a continuación [11]:

- TDA3LA: Módulo de bajo voltaje con aceleración de visión para aplicaciones ADAS.
- TDA3LX: Módulo de bajo voltaje con aceleración de visión, procesamiento y representación de imágenes para aplicaciones ADAS.
- TDA3MA: Módulo de bajo voltaje con aceleración de visión y procesamiento completo para aplicaciones ADAS.

2.9. Control basado en visión (*visual servoing*)

El control basado en visión, como lo define [12], consiste en un método de control que incorpora directamente información visual al lazo de control. Esto permite tener versatilidad en el sistema, como aplicaciones robóticas, dada la actualización de datos en tiempo real.

Según [13] se tienen tres tipos de control basado en visión, categorizados según el error obtenido, ya sea en error de control en un espacio de imagen 2D, o un espacio cartesiano 3D. Los tipos de control son: control basado en visión basado en imagen, control basado en visión basado en posición y la aproximación 2 1/2 D.

Algunos estudios que utilizan aplicaciones con control basado en visión se presentan en [12] y [13], en donde se discute acerca del uso de control basado en visión para optimizar el marco de trabajo para el control de cuerpo completo de robots humanoides y un aplicación de sujeción y colocación en un matadero, respectivamente.

CAPÍTULO 3

Justificación

Dadas las ventajas que presenta la visión por computadora, se han desarrollado diferentes disciplinas dentro del campo. El alcance de este estudio se enfocará en visión de robot, rama de la visión por computadora la cual permite la investigación acerca de temas de control basado en visión.

Se pretende realizar una comparación clara entre los módulos con visión por computadora disponibles en el mercado para, al finalizar, desarrollar una aplicación de control basado en visión. Como se aprecia en la sección de Antecedentes, existen variedad de módulos capaces de incorporar aplicaciones de control mediante visión en sistemas embebidos, sin embargo, las capacidades de cada uno son distintas. Por esta razón, la iniciativa de una comparativa es necesaria para determinar el componente que mejor se adapte a las necesidades actuales respecto al desarrollo del laboratorio de Robótica de la Universidad del Valle de Guatemala, específicamente en el campo de los robots móviles. Para realizar las comparaciones indicadas, se pretende estudiar algoritmos clásicos de visión por computadora en dos módulos distintos, la JeVois Smart Machine Vision Camera y la ESP32-CAM, realizando una serie de experimentos que demuestren las capacidades de cada módulo para, finalmente, implementarlo como realimentación en una aplicación de control basado en visión. Asimismo se desea demostrar que este método de control es implementable y puede ser más efectivo a comparación de los métodos de localización implementados a sistemas ensamblados en años anteriores.

Dado que el laboratorio dedicado a Robótica de la Universidad del Valle de Guatemala está empezando, la importancia de este estudio radica en permitir iniciar las investigaciones que se llevarán a cabo en este recinto con técnicas de control flexibles, versátiles y, sobre todo, modernas para mantenerse a la vanguardia en el campo. Por otro lado, con este estudio se incentiva a iniciar una línea de investigación basada en el control basado en visión.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar e implementar algoritmos de visión por computadora clásicos para la selección del componente más óptimo a utilizar en una aplicación de control basado en visión (*visual servoing*).

4.2. Objetivos específicos

- Diseñar e implementar experimentos y algoritmos, con la librería OpenCV, que permitan dotar de visión por computadora a un sistema embebido.
- Investigar, evaluar y seleccionar el módulo de visión por computadora más útil para una aplicación de control basado en visión.
- Implementar un sistema de control representativo que emplee el sistema de visión por computadora como realimentación (*feedback*).

CAPÍTULO 5

Alcance

Este proyecto se enfocó en la selección de un módulo capaz de dotar de visión por computadora a un sistema embebido y, con este, realizar una aplicación básica de control basado en visión. En el mercado se tiene distintos módulos que cumplen con características básicas de visión por computadora, sin embargo, para una aplicación como la deseada se necesitó de un módulo capaz de procesar imagenes en tiempo real y con capacidad de detectar marcadores de referencia. Las opciones iniciales para este estudio fueron la JeVois Smart Vision Camera y la OpenMV Cam H7. Estos dos fueron seleccionados dadas sus características similares, por lo que una comparación era justificada.

Sin embargo, debido al a pandemia del COVID-19 y la escasez de chips que se presentó en 2021, la obtención del módulo OpenMV Cam H7 no fue posible a tiempo para el estudio. Por motivos del proyecto se tomó la decisión de utilizar un módulo que cumpliera con las características básicas de dotar de visión por computadora a un sistema, aunque la comparación subsecuente fuera sesgada hacia la JeVois Smart Vision Camera. El módulo seleccionado fue la ESP32-CAM, un componente menos poderoso que la OpenMV Cam H7 pero lo suficientemente capaz para realizar la comparación deseada.

Luego de la selección del módulo de visión la idea fue integrarlo en un brazo robot sencillo y que siguiera un marcador de referencia ArUco por medio de movimientos horizontales y verticales. La señal proveída por el módulo funcionó como realimentación en un controlador, encargado de mover los motores a las posiciones necesarias para mantener centrado en la cámara el objeto.

CAPÍTULO 6

Marco teórico

6.1. Visión por computadora

La visión por computadora es la transformación de datos obtenidos por medio de una cámara, en una decisión o nueva representación [14]. Este consiste en un problema inverso en el que se busca recuperar incógnitas dado que no hay suficiente información para una solución completamente especificada [15].

6.2. Formación de imagen

6.2.1. Primitivos geométricos y transformaciones

Geometría 2D

Los puntos 2D, o bien, coordenadas de píxel en imagen, se denotan como pares ordenados de la forma $\mathbf{x} = (x, y) \in \mathcal{R}^2$, representado también de forma homogénea como $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$. Un vector homogéneo se puede volver no homogéneo al dividir sus elementos entre $\tilde{w}$, teniendo esto, se llega a (1), donde $\bar{\mathbf{x}} = (x, y, 1)$ y es conocido como el vector aumentado.

$$\tilde{x} = \tilde{w}\bar{x}. \tag{1}$$

Los puntos homogéneos cuyo $\tilde{w} = 0$ se conocen como puntos ideales, o puntos en el infinito [15].

A partir de lo anterior, se puede generar una línea 2D utilizando coordenadas homogéneas $\tilde{\mathbf{l}} = (a, b, c)$, llegando a (2).

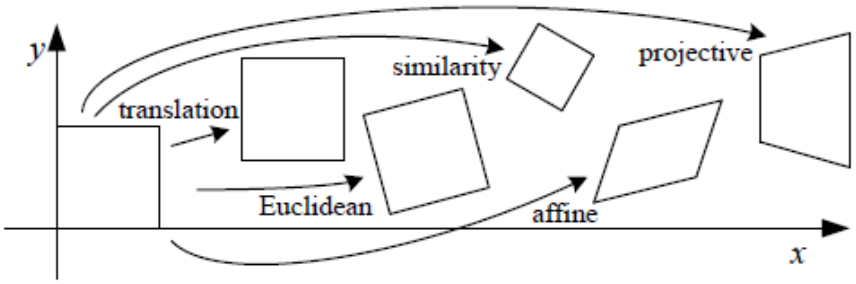


Figura 1: Transformaciones 2D básicas [15].

$$\bar{x} \cdot \tilde{I} = ax + by + c = 0. \tag{2}$$

Las secciones cónicas 2D son útiles en el estudio de geometría multivista y calibración de cámara, estas se describen con (3).

$$\tilde{x}^T Q \tilde{x} = 0. \tag{3}$$

Geometría 3D

Los puntos 3D se denotan de la forma $x = (x, y, z) \in \mathcal{R}^3$, también representados de forma homogénea como $\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$. El vector aumentado $\bar{x}$ también es utilizado.

El análogo a una línea 2D en el espacio 3D es un plano, el cual se describe con las coordenadas homogéneas $\tilde{m} = (a, b, c, d)$, llegando a (4).

$$\bar{x} \cdot \tilde{m} = ax + by + cz + d = 0. \tag{4}$$

El caso de las secciones cónicas en 3D siguen a (3) respetando el espacio de los vectores. Además del estudio de geometrías multivista, en 3D las secciones cónicas son útiles para el modelado de primitivos, tales como esferas, elipsoides o cilindros [15].

Transformaciones 2D

Las transformaciones 2D posibles las presenta [15] y se muestran en la Figura 1. Estas se logran por medio de una combinación de traslaciones y/o rotaciones.

Las traslaciones en 2D están definidas por (5)

$$x' = \begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix}. \tag{5}$$

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix}\mathbf{I} & \mathbf{t}\end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix}s\mathbf{R} & \mathbf{t}\end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix}\mathbf{A}\end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix}\tilde{\mathbf{H}}\end{bmatrix}_{3\times3}$	8	straight lines	

Figura 2: Resumen de transformaciones 2D [15].

El vector aumentado en (5) puede ser sustituido por un vector homogéneo $\tilde{x}$.

La combinación de una rotación con una traslación es conocido como un movimiento de cuerpo rígido 2D, o bien, transformación Euclidiana 2D, cuya expresión matemática está dada por (6).

$$x' = \begin{bmatrix} R & t \end{bmatrix}. \tag{6}$$

Donde R es la matriz de rotación (7), la cual puede ser escalada con un factor de escala s, como se puede observar en la Figura 2:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{7}$$

Algunas otras transformaciones son: la transformación afín, en la que un vector aumentado $\bar{x}$ es multiplicado por una matriz A de 2×3 y la transformación de perspectiva, u homográfica, la cual se discute más adelante. Los efectos de estas transformaciones se presentan en la Figura 2.

Transformaciones 3D

Tanto las traslaciones, como la combinación de rotación y traslación siguen las expresiones (5) y (6) con las modificaciones necesarias para el caso 3D. Esto quiere decir que la matriz identidad de (5) y la matriz de rotación de (6) son de 3×3. El efecto de cada una de las transformaciones se muestra en la Figura 3.

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{3\times4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{3\times4}$	6	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{3\times4}$	7	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{3\times4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{4\times4}$	15	straight lines	

Figura 3: Resumen de transformaciones 3D [15].

Rotaciones 3D

Los ángulos de Euler consisten en el producto de rotaciones en los tres ejes coordenados, cambiando el resultado según el orden de la rotación. Dada esta característica y que, en ocasiones, no es posible realizar movimientos suaves en el espacio de parámetros, no se recomienda el uso de esta convención. En esta sección se presenta dos alternativas: la convención eje/ángulo y los cuaterniones unitarios [15].

La convención eje/ángulo indica que una rotación puede ser descrita como la rotación en el eje $\hat{n}$ por un ángulo θ.

$$\omega = \theta\hat{n}. \tag{8}$$

La representación dada por (8) es minimalista para una rotación 3D y es útil para pequeñas rotaciones. La matriz de rotación correspondiente a una rotación θ alrededor de un un eje $\hat{n}$ se encuentra por medio de la Fórmula de Rodrigues (9), la derivación de esta se puede encontrar en [15].

$$R(\hat{n}, \theta) = I + \sin\theta[\hat{n}]_x + (1 - \cos\theta)[\hat{n}]_x^2. \tag{9}$$

De (9) se nota el término $[\hat{n}]_x$ el cual es una forma matricial de un producto cruz con un vector $\hat{n} = [\hat{n}_x\hat{n}_y\hat{n}_z]$ representado como (10):

$$[\hat{n}]_x = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}. \tag{10}$$

El uso de (8) simplifica la fórmula de Rodrigues (9), como se muestra en (11). Esta da

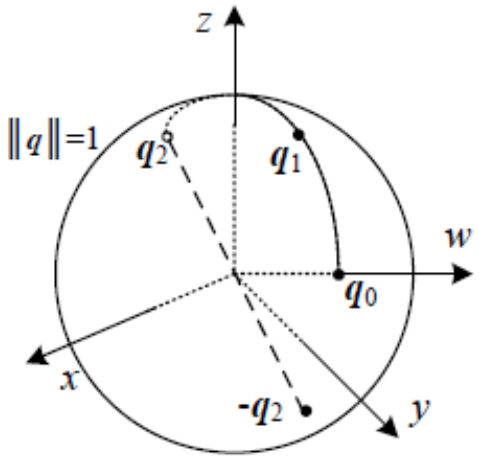


Figura 4: Representación de cuaterniones unitarios [15].

una relación linealizada de la rotación entre los parámetros ω y R,

$$R(\omega) = I + \sin\theta[\hat{n}]_x \approx I + [\theta\hat{n}]_x = \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix}. \tag{11}$$

Los cuaterniones unitarios son vectores de cuatro elementos de la forma $q = (q_x, q_y, q_z, q_\omega)$. Se presentan en una esfera unitaria, como se muestra en la Figura 4. Los cuaterniones de signo negativo $(-q)$ representan la misma rotación que q, fuera de esta dualidad, cada cuaternión expresa una única rotación.

Los cuaterniones unitarios dan una representación continua, es decir, al haber matrices de rotación que varían de forma continua, se puede encontrar una representación en cuaterniones a pesar que el recorrido en la esfera unitaria se cierre antes de regresar al origen dado por $q = (0, 0, 0, 1)$. Por esta y otras razones son una representación popular de pose e interpolación de pose en el campo de gráficos de computadora, [15].

Un cuaternión unitario se puede derivar de la representación eje/ángulo utilizando una igualdad para v y ω, así como dos identidades trigonométricas:

$$\begin{aligned} q = (v, \omega) &= \left(\sin\frac{\theta}{2}\hat{n}, \cos\frac{\theta}{2}\right), \\ \sin\theta &= 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}, \\ (1 - \cos\theta) &= 2\sin^2\frac{\theta}{2}. \end{aligned}$$

Utilizando las expresiones previas, la fórmula de Rodrigues se representa por (12).

procedure *slerp*$(\mathbf{q}_0, \mathbf{q}_1, \alpha)$:

1. $\mathbf{q}_r = \mathbf{q}_1/\mathbf{q}_0 = (\mathbf{v}_r, w_r)$
2. **if** $w_r < 0$ **then** $\mathbf{q}_r \leftarrow -\mathbf{q}_r$
3. $\theta_r = 2\tan^{-1}(\|\mathbf{v}_r\|/w_r)$
4. $\hat{\mathbf{n}}_r = \mathcal{N}(\mathbf{v}_r) = \mathbf{v}_r/\|\mathbf{v}_r\|$
5. $\theta_\alpha = \alpha\,\theta_r$
6. $\mathbf{q}_\alpha = (\sin\frac{\theta_\alpha}{2}\hat{\mathbf{n}}_r, \cos\frac{\theta_\alpha}{2})$
7. **return** $\mathbf{q}_2 = \mathbf{q}_\alpha\mathbf{q}_0$

Figura 5: Algoritmo SLERP [15].

$$R(v,\omega) = I + 2\omega[\hat{v}]_x + 2[\hat{v}]_x^2 = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-z\omega) & 2(xy-y\omega) \\ 2(xy+z\omega) & 1-2(x^2+z^2) & 2(yz-x\omega) \\ 2(xz-y\omega) & 2(yz+x\omega) & 1-2(x^2+y^2) \end{bmatrix}. \quad (12)$$

Si se desea determinar la rotación entre dos rotaciones dadas, se calcula la rotación creciente tomando una fracción del ángulo y calculando la nueva rotación. Este proceso es conocido como el algoritmo SLERP (*Spherical Lineal Interpolation*. El pseudocódigo del algoritmo se presenta en la Figura 5.

Proyecciones de 3D a 2D

Existen diversos modelos de proyección, estos se muestran en la Figura 6. La aproximación más sencilla es la proyección ortográfica, la que quita la componente z de un vector tridimensional p para obtener uno bidimensional x como se muestra en (13).

$$x = \begin{bmatrix} I_{2\times 2} & 0 \end{bmatrix} p. \quad (13)$$

En la práctica es usual escalar la proyección ortogonal por medio de un factor de escalamiento s, multiplicado a la matriz identidad. La proyección ortográfica escalada es popular en la reconstrucción de modelos 3D alejados de la cámara [15].

Otro modelo de proyección es la para-perspectiva. En este, los puntos del objeto son proyectados a una referencia local de forma paralela a la línea de visión del centro del objeto [15].

La proyección más utilizada en la visión por computadora, así como en los gráficos de computadora, es la perspectiva 3D. Los puntos en un plano son divididos entre su componente z (14) expresa este tipo de proyección de forma matemática.

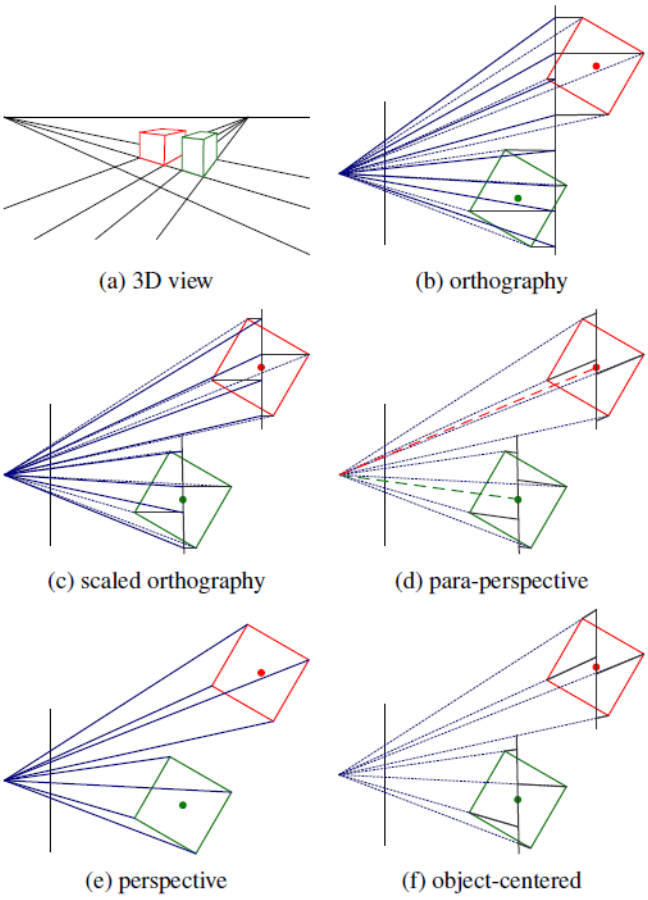


Figura 6: Proyecciones de 3D a 2D [15].

$$\bar{x} = \mathcal{P}_z(p) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}. \tag{14}$$

Utilizando coordenadas homogéneas, la proyección perspectiva se representa con (15). Nótese que se pierde la componente ω del p, esto corresponde a la distancia del punto 3D de la imagen [15].

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{p}. \tag{15}$$

Intrínsecos de la cámara

La combinación de proyecciones 2D a 3D la expresa (16):

$$p = \begin{bmatrix} R_s & c_s \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = M_s \bar{x}_s, \tag{16}$$

donde R_s es una rotación 3D y c_s son las coordenadas del origen del centro de proyección de la cámara. Por lo tanto, observando (16), se nota que M es una matriz 3×3 cuyas primeras dos columnas tienen los datos de los pasos unitarios en el arreglo de la imagen en las direcciones de x_s y y_s, mientras que la tercera consiste en el arreglo de origen c_s.

La matriz M está parametrizada por ocho incógnitas, tres de rotación, dos de factores de escala. En la práctica solo se puede estimar siete de estos ochos parámetros (grados de libertad), pues la distancia entre el sensor y el origen no se puede comprobar. Es por esta razón que estimar un modelo de cámara M_s con siete grados de libertad es poco práctico, por lo que se asume una forma general de matriz 3×3 homogénea [15].

Una proyección completa entre un punto p_c y las coordenadas homogéneas de un píxel está descrita por (17).

$$\tilde{x}_s = \alpha M_s^{-1} p_c = K p_c. \tag{17}$$

En (17), K es una matriz de 3×3 conocida como matriz de calibración, la cual describe los intrínsecos de la cámara.

Por convención, la matriz K se escribe de forma triangular superior, una de las posibles maneras de denotarla se presenta en (18).

$$K = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{18}$$

donde f es la distancia focal, a la relación de aspecto, s el posible sesgo (*skew*) y los términos c_x y c_y representan el centro de la imagen, conocido en la literatura de visión por computadora como punto principal [15].

Por lo general, el punto (c_x, c_y) se toma alrededor del centro de la imagen, es decir:

$$(c_x, c_y) = \left(\frac{W}{2}, \frac{H}{2} \right),$$

siendo W y H el ancho y alto de la imagen respectivamente. Esto da un modelo aceptable de la cámara, teniendo únicamente la distancia focal como incógnita.

Al combinar los intrínsecos K y extrínsecos R y t de la cámara, se obtiene una matriz de 3×4 llamada P. Esta es la matriz de cámara y (19) la denota.

$$P = K \begin{bmatrix} R & t \end{bmatrix}. \tag{19}$$

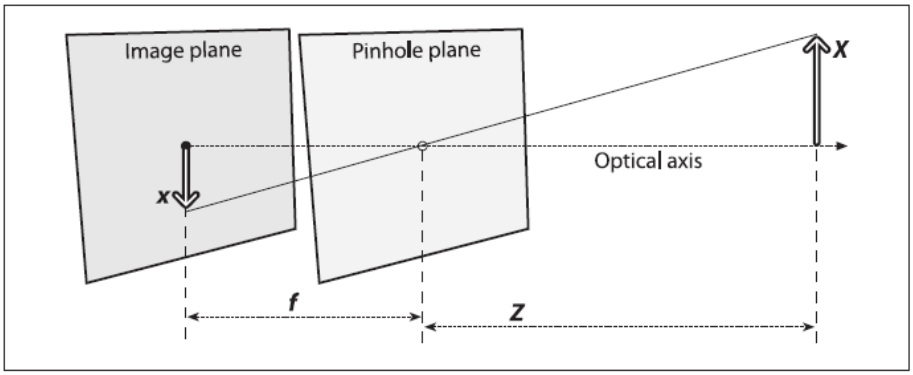


Figura 7: Modelo de cámara *pinhole* [14].

En ocasiones es preferible utilizar una matriz 4×4, dado que se puede invertir. Esta es obtenida al no desechar la última fila de P, por lo tanto, si se toma (19) y se reescribe se obtiene (20).

$$\tilde{P} = \begin{bmatrix} K & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} = \tilde{K}E. \tag{20}$$

Donde E es una transformación Euclidiana de cuerpo rígido y $\tilde{K}$ es una calibración de rango completo. La matriz $\tilde{P}$, (20), es útil para mapear desde coordenadas 3D $\bar{p}_\omega = (x_\omega, y_\omega, z_\omega, 1)$ a un plano de imagen $x_s = (x_s, y_s, 1, d)$ más disparidad, esto se logra por medio de (21).

$$x_s \sim \tilde{P}\bar{p}_\omega. \tag{21}$$

En donde $\sim$ denota igualdad a escala [15].

6.3. Modelos de cámara y calibración

6.3.1. Modelo de cámara

El modelo de cámara más sencillo es el modelo *pinhole*. En este se tiene que un único rayo de luz entra desde la escena, u objeto distante, por un punto en particular, el *pinhole*. Un *pinhole* es un plano imaginario con un pequeño agujero, generalmente en el centro, que bloquea todos los rayos de luz excepto uno [14].

En una cámara *pinhole* física este punto es proyectado en el plano de imagen. Como resultado, la imagen en el plano siempre está enfocada y el tamaño respecto al objeto capturado está dado por la distancia focal. Para cámaras *pinhole* idealizadas, la distancia entre el *pinhole* y la pantalla (plano de imagen) es la distancia focal.

Observando la Figura 7 se nota la relación mostrada en (22), en la cual, x es la imagen

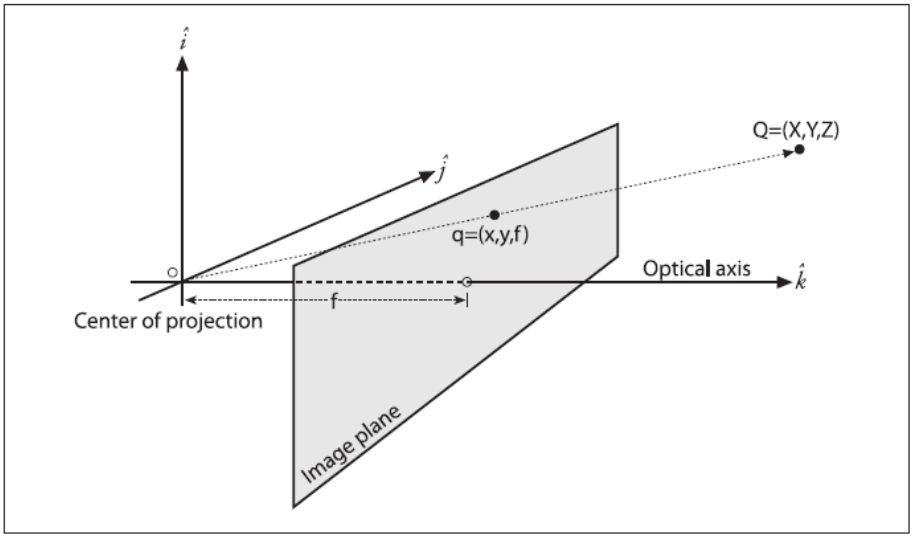


Figura 8: Modelo modificado de cámara *pinhole* [14].

del objeto en el plano de imagen, f es la distancia focal, X el tamaño del objeto y Z la distancia entre la cámara y el objeto.

$$-x = f\frac{X}{Z}. \tag{22}$$

El negativo en x de (22) se debe a que la proyección obtenida está de cabeza. Para arreglar esto se puede modificar el modelo *pinhole* como se muestra en la Figura 8, notando que el punto en el *pinhole* es renombrado como centro de proyección [14].

Físicamente, el modelo presentado en la Figura 8 no se puede armar, sin embargo, simplifica el análisis matemático para llegar a las expresiones (23) y (24).

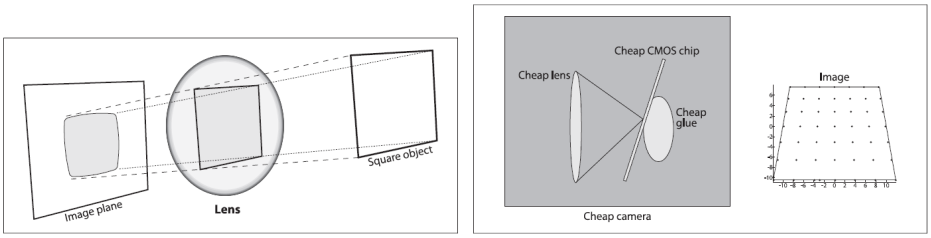
$$x_{screen} = f_x\left(\frac{X}{Z}\right) + c_x, \tag{23}$$

$$y_{screen} = f_y\left(\frac{Y}{Z}\right) + c_y. \tag{24}$$

Donde c_x y c_y son parámetros de desplazamiento del centro de coordenadas del plano de imagen. Realizando una observación, se tiene dos distancias focales, una en cada eje pues con cámaras de bajo costo los píxeles son rectangulares de lado s_x y s_y. El valor real de la distancia focal se obtiene con (25) o (26) [14].

$$F = \frac{f_x}{s_x}, \tag{25}$$

$$F = \frac{f_y}{s_y}. \tag{26}$$



(a) Distorsión radial. (b) Distorsión tangencial.

Figura 9: Tipos de distorsiones de lente [14].

Proyecciones geométricas básicas

La relación que mapea puntos del mundo físico, especificados por un vector Q a coordenadas en el plano de imagen se conoce como trasformación proyectiva, la cual está dada por (27).

$$q = MQ, \tag{27}$$

en donde las variables tienen la siguiente forma:

$$q = \begin{bmatrix} x & y & \omega \end{bmatrix}^T, \tag{28}$$

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{29}$$

$$Q = \begin{bmatrix} X & Y & Z \end{bmatrix}^T. \tag{30}$$

Distorsión de lente

No existe lentes perfectamente parabólicas, por lo que se debe considerar distorsiones en la imagen proyectada. Se tiene dos tipos principales de distorsiones: radiales y tangenciales, [14].

Las distorsiones radiales se dan por la forma del lente. En estas, la localización de los píxeles en la periferia de la imagen se ven distorsionados mientras que los píxeles del centro no presentan ninguna. La Figura 9a ejemplifica una distorsión radial. Las distorsiones tangenciales se dan por defectos de manufactura, en los que el lente no está en paralelo al plano de imagen, esto se puede observar de mejor manera en la Figura 9b.

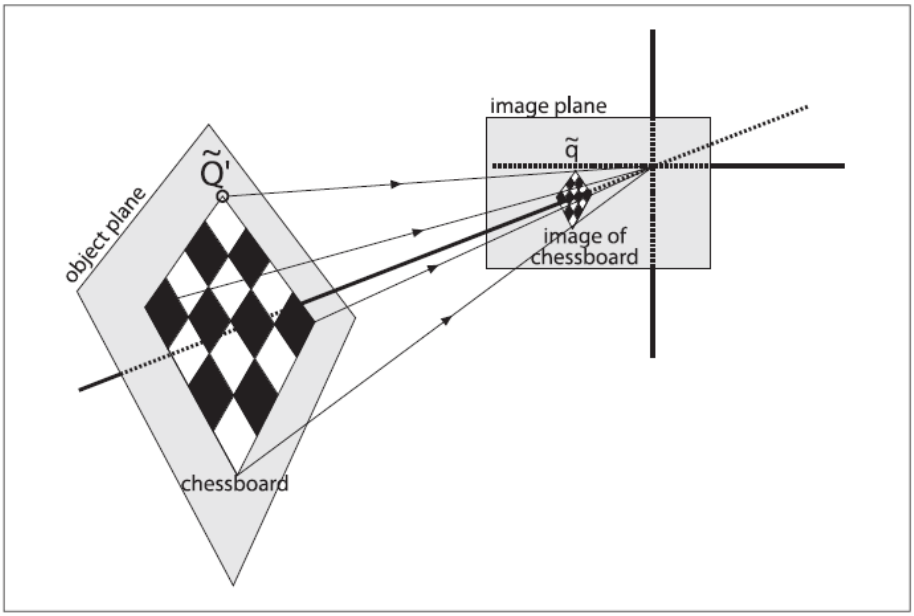


Figura 10: Representación de una homografía [14].

6.3.2. Calibración

Tablero de ajedrez

En principio cualquier objeto con características definidas puede ser utilizado como objeto de calibración. Sin embrago, la elección práctica es un tablero de ajedrez, el cual corresponde de un patrón alternante de blancos y negros. Este se utiliza para que no haya sesgo en las lecturas, además, las esquinas de la grilla son útiles para localización de subpíxeles [14].

Homografía

Una homografía planar consiste en un mapeo proyectivo desde una superficie planar a otra. En términos de multiplicación de matrices, (31) describe una homografía. En este se sabe que los vectores $\tilde{q}$ y $\tilde{Q}$ son coordenadas homogéneas en el plano y mundo real respectivamente, por otro lado, s es un factor de escala y H es la matriz de homografía

$$\tilde{q} = sH\tilde{Q}. \tag{31}$$

Observando la Figura 10, se nota que H tiene dos partes; una transformada física, la cual localiza el plano del objeto, y una proyección que introduce los intrínsecos de la cámara.

Se sabe que $\tilde{Q}$ está definido en todo el espacio, por lo que el cálculo matemático se puede simplificar para el plano de objeto, $\tilde{Q}'$, eligiéndolo para que Z sea igual a cero. Esto se logra separando la matriz de rotación en vectores columna de 3×1, es decir:

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}.$$

Teniendo estas condiciones nuevas y aplicándolas a (31), se tiene el siguiente procedimiento el cual concluye en (32), que da una expresión para la matriz de homografía incluyendo el factor de escala s en ella [14].

$$\begin{aligned} \tilde{q} &= sH\tilde{Q}, \\ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} &= sM \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}, \\ &= sM \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, \end{aligned}$$

$$H = sM \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}. \tag{32}$$

La matriz de homografía relaciona las posiciones de los puntos en un plano de imagen fuente a un plano de imagen destino, sabiendo que:

$$\begin{aligned} p_{dst} &= \begin{bmatrix} x_{dst} & y_{dst} & 1 \end{bmatrix}^T, \\ p_{src} &= \begin{bmatrix} x_{src} & y_{src} & 1 \end{bmatrix}^T. \end{aligned}$$

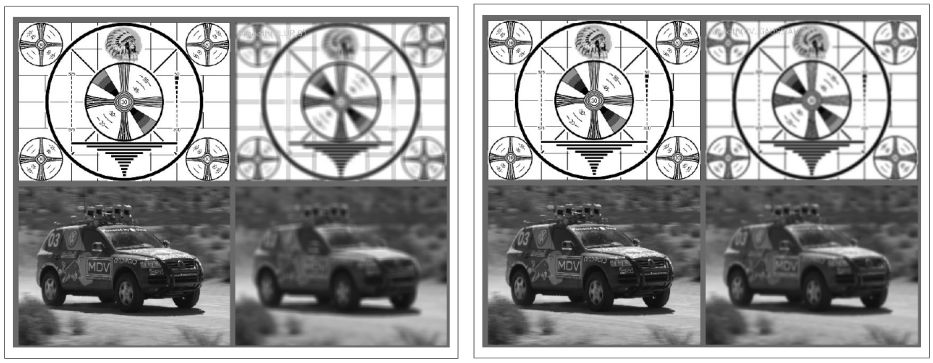
Utilizando estos datos se puede llegar a las expresiones (33) y (34), las cuales muestra que, a pesar que se tiene (32), se puede calcular la matriz de homografía H sin necesidad de conocer los intrínsecos de la cámara [14].

$$p_{dst} = Hp_{src}, \tag{33}$$
$$p_{src} = H^{-1}p_{dst}. \tag{34}$$

Calibración de cámara

La expresión matemática para describir la calibración de cámara se deduce en [14, p. 389]. La ecuación a la que se llega es (35).

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = (1 + k_1r^2 + k_2r^4 + k_3r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1x_dy_d + p_2(r^2 + 2x_d^2) \\ p_1(r^2 + 2y_d^2) + 2p_2x_dy_d \end{bmatrix}. \tag{35}$$



(a) Suavizado simple. (b) Suavizado Gaussiano.

Figura 11: Ejemplos de suavizado [14].

Explicando (35), se sabe que el punto (x_p, y_p) es la localización del píxel del objeto si la cámara *pinhole* fuera perfecta, mientras que el punto (x_d, y_d) corresponde a las coordenadas distorsionadas. Respecto al resto de términos, en la práctica, la distorsión es pequeña por lo que se puede expresar por medio de los primeros términos de una expansión por series de Taylor alrededor de $r = 0$; (k_1, k_2, k_3, p_1, p_2) son los coeficientes de distorsión obtenidos por la expansión [14].

6.4. Procesamiento de imágenes

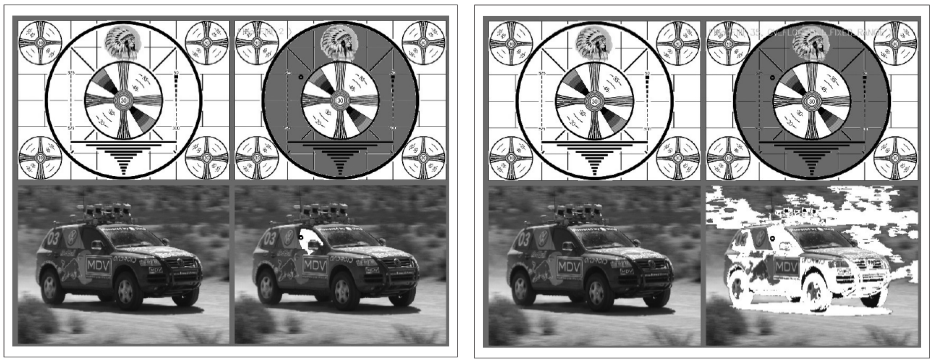
6.4.1. Suavizado

El suavizado o difuminado, es una operación de procesamiento de imágenes donde se reduce el ruido o artefactos de cámara. También es utilizado para reducir la resolución de una imagen. Se tiene distintos tipos de suavizado, logrados a partir de diferentes técnicas, la Figura 11 muestra dos casos: suavizado simple y Gaussiano.

El suavizado simple consiste en una operación en la que todos los píxeles de salida son la media de los píxeles en una ventana alrededor del píxel correspondiente en la entrada, un ejemplo de este tipo de suavizado se muestra en la Figura 11a. Por otro lado, el suavizado medio utiliza el valor de la mediana, en lugar de la media, esto lo hace con los datos de un cuadrado con el píxel de entrada correspondiente en el centro.

El filtro Gaussiano se realiza al convolucionar cada punto en el arreglo de entrada con un kernel Gaussiano y luego sumar los resultados para obtener el arreglo de salida. Es el más utilizado en la práctica, más no el más rápido. Se puede observar un ejemplo de un suavizado Gaussiano en la Figura 11b.

Finalmente, el suavizado bilateral consiste en una operación similar al filtro Gaussiano explicado previamente, sin embargo, este le da un mayor peso a los píxeles similares que a los píxeles menos similares. El efecto que se obtiene al aplicar un suavizado de este tipo es el de una pintura de acuarela [14].



(a) Ejemplo 1 de *Flood Fill*.

(b) Ejemplo 2 de *Flood Fill*.

Figura 12: Ejemplos de *Flood Fill* [14].

6.4.2. Algoritmo de relleno por difusión (*Flood Fill*)

Este es un algoritmo utilizado para marcar o aislar porciones de una imagen. Asimismo, se utiliza para derivar máscaras a partir de una imagen de entrada con las que se puede acelerar o restringir el procesamiento de píxeles a los se les haya aplicado [14]. Se puede observar dos ejemplos de este este algoritmo en la Figura 12.

En la Figura 12a se aplicó el algoritmo de relleno (*flood fill*) alrededor del punto negro, presentado en ambas imágenes. La imagen superior fue llenada con gris y se nota como las zonas que comparten características fueron rellenadas, mientras que en la imagen inferior se aplicó un llenado blanco. La Figura 12b muestra los mismos resultados de manera más drástica, de allí es que se nota más zonas rellenas de gris en la imagen superior y un relleno más amplio de blanco en la imagen inferior.

6.4.3. Pirámide de imágenes

Una pirámide consiste en una colección de imágenes que surgen a partir de una original. Cada una de estas imágenes es una reducción de muestra, esto quiere decir que las imágenes fueron muestreadas de forma que se ampliara la imagen hasta llegar a un punto deseado. La Figura 13 muestra como, a partir de una imagen base, conforme se sube de nivel la imagen muestreada es cada vez más específica respecto a un punto. Según [15], cada nivel de la pirámide de la Figura 13 tiene la mitad de resolución (en largo y ancho), por ende un cuarto de los píxeles del nivel previo.

Existen dos tipos de pirámides: Gaussianas y Laplacianas [14]. En las pirámides Gaussianas se realiza una reducción de muestra, relacionando a la Figura 13, es una pirámide que va desde niveles bajos a niveles altos. Por el otro lado, las pirámides Laplacianas consisten en un aumento de muestra, esto implica una reconstrucción de la imagen a partir un nivel con más detalle. Relacionando con la Figura 13, se empieza en los niveles más altos y se desea bajar a niveles más cercanos a la base.

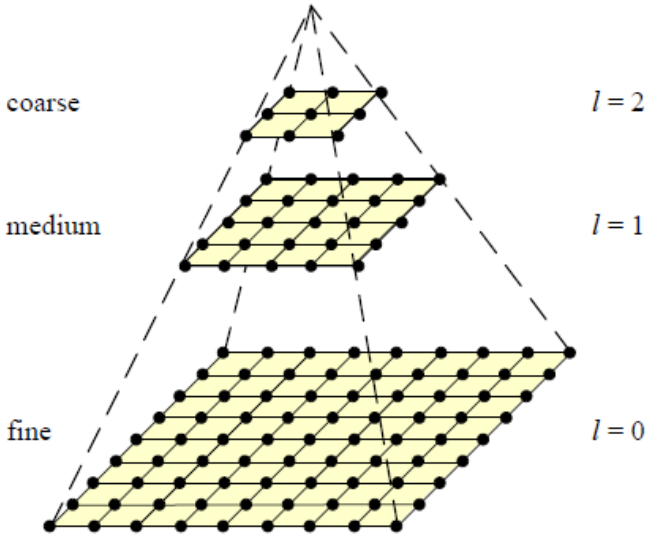


Figura 13: Representación de una pirámide de imágenes [15].

6.4.4. *Threshold*

El *threshold* es una operación de comparación y categorización, en la que a cada píxel se le asigna un valor dependiendo de su valor respecto a un umbral propuesto. La idea es proveer un arreglo junto con un umbral (*threshold*) y obtener una salida en cada elemento del arreglo según sus características.

La librería *OpenCV* presenta distintos tipos de *threshold*, se puede observar cuales son y el tipo de categorización que realizan en la Figura 14. En esta misma figura se puede notar que los valores de umbral al ser fijos, pueden llegar a presentar problemas en caso parámetros de iluminación o reflectividad no sean cumplidos. Esto pues los resultados tienden a ser extremos, como el caso del *threshold* binario. Para solucionar esto, se desarrolló una técnica de *threshold* adaptativo en el cual el nivel del umbral es variable. Una comparativa entre un *threshold* binario y uno adaptativo se presenta en la Figura 15.

El *threshold* adaptativo es útil cuando se presentan gradientes de iluminación y reflectividad grandes [14].

6.5. Transformadas de imagen

6.5.1. Convolución

La base para muchas de las transformadas en el campo de visión por computadora es la convolución. El resultado de esta está determinado por el kernel utilizado. El kernel consisten en un arreglo de tamaño fijo con coeficientes numéricos y un punto ancla, normalmente localizado en el centro [14].

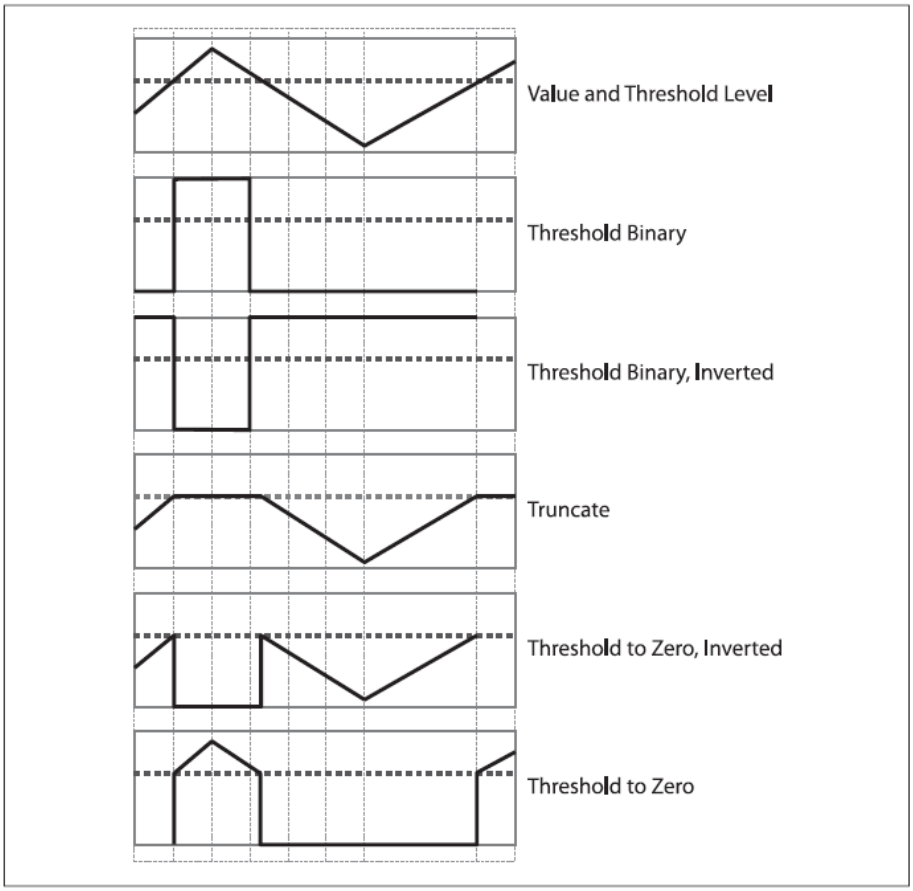


Figura 14: Tipos de *threshold* en *OpenCV* y sus efectos [14].

6.5.2. Gradientes y derivadas de Sobel

El cálculo de derivadas se obtiene por medio de convolución. El operador más común para representar diferenciación se conoce como derivada de Sobel la cual, en realidad, no es una derivada sino un ajuste a un polinomio [14].

Los operadores de Sobel pueden ser de cualquier orden de derivada así como para derivadas parciales mixtas. Estos tienen la característica de poder ser definidos por kernels de cualquier dimensión.

La Figura 16 muestra ejemplos de los resultados que obtiene al aplicar derivadas de Sobel respecto a las dimensiones x y y. Se observa que el resultado en las derivadas aplicadas en x resaltan las líneas verticales de las imágenes propuestas (Figura 16a), caso contrario en las derivadas aplicadas en y donde resaltan las líneas horizontales (Figura 16b).

6.5.3. Laplace

El operador de Laplace consiste en una sumatoria de derivadas de segundo orden a lo largo de los ejes x y y. Se utiliza en diferentes contextos, como detección de regiones (*blob*

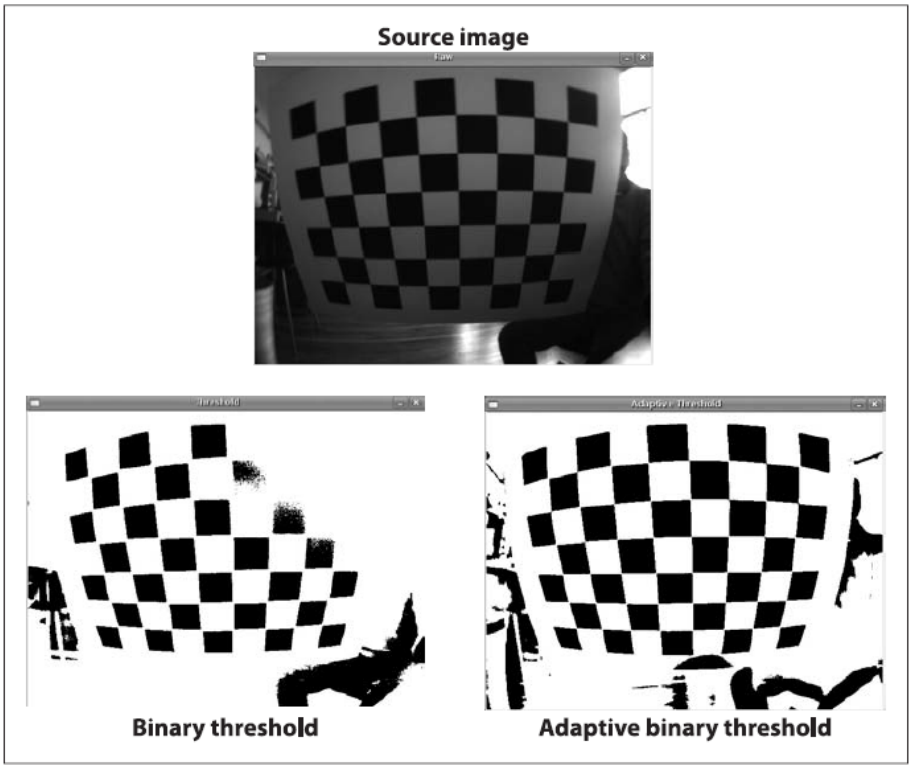


Figura 15: Comparación de *thresholds* [14].

detection) pues cualquier punto que esté rodeado por valores mayores tenderá a maximizar la función [14].

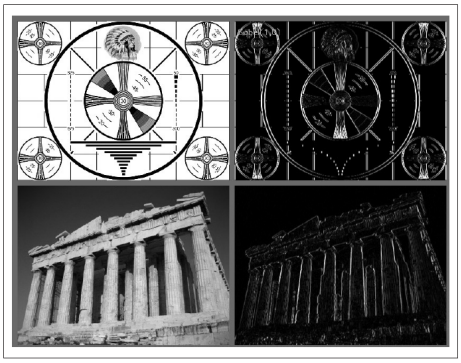
6.5.4. Detector de bordes de Canny

El operador de Laplace puede utilizarse para detección de bordes. Para esto se considera la primera derivada, la cual será grande en lugares donde haya cambios rápidos en la función. De igual forma, crecerá rápidamente al acercarse a discontinuidades parecidas a bordes y disminuirá al alejarse. Por lo tanto, la derivada estará en un máximo local en el rango del borde [14].

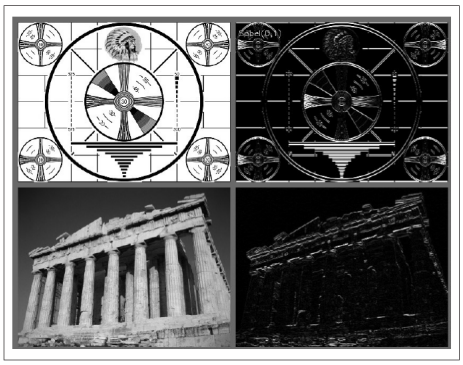
John Canny refinó la idea del operador de Laplace al calcular las primeras derivadas tanto en x como en y y luego combinarlas en derivadas de cuatro direcciones. Los puntos en los que estas derivadas direccionales son máximos locales, son candidatos para ser bordes.

La adición más significativa del algoritmo de Canny con respecto al operador de Laplace es que el algoritmo de Canny trata de ensamblar cada candidato de borde en contornos (lista de puntos que representan una curva en una imagen). Estos contornos se forman al aplicar un umbral de histéresis (*hysteresis threshold*) a los píxeles, esto significa que hay dos umbrales, uno superior y otro inferior [14].

Si el gradiente de un píxel es mayor que el umbral superior es aceptado como un borde; si está por debajo del umbral inferior, es rechazado y, si se encuentra entre los umbrales, es

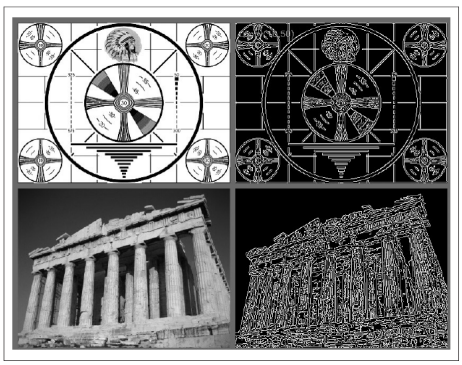


(a) Derivadas de Sobel aplicadas en la dimensión x.

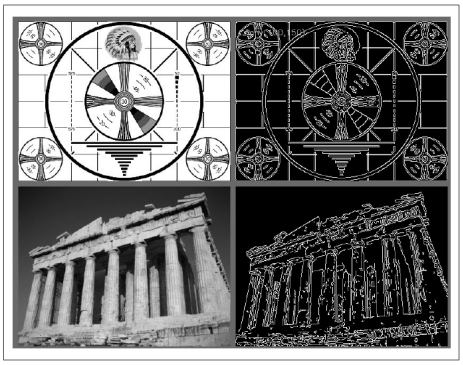


(b) Derivadas de Sobel aplicadas en la dimensión y.

Figura 16: Derivadas de Sobel aplicadas en diferentes dimensiones [14].



(a) Umbrales 50:10.



(b) Umbrales 150:100.

Figura 17: Implementación del detector de bordes de Canny [14].

aceptado solo si está conectado a un píxel que esté por encima del umbral superior.

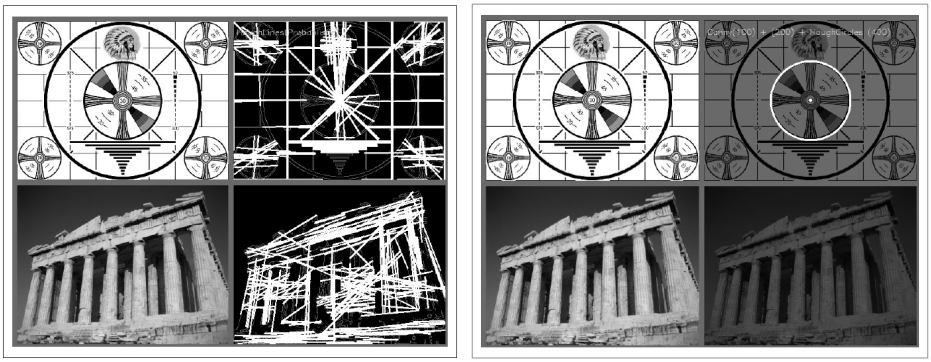
Canny recomendó umbrales, con nomenclatura superior:inferior, entre 2:1 y 3:1 [14].

La Figura 17 muestra ejemplos de la aplicación del algoritmo detector de bordes de Canny. Se nota como, en la Figura 17a, hay mayor cantidad de bordes especialmente en la imagen inferior, por los umbrales aplicados. Al hacer los umbrales más grandes, como en la Figura 17b, se nota una imagen más limpia con una detección de bordes definida.

6.5.5. Transformada de Hough

Consiste en un método para encontrar líneas, círculos, u otras geometrías básicas. La teoría de esta transformada indica que cualquier punto de una imagen binaria puede ser parte de algún set posible de líneas. Se tiene dos tipos de transformadas de Hough, uno es específico para líneas mientras el otro para círculos [14].

En la Figura 18 se muestra ejemplos de la aplicación de cada uno de los tipos de transformada de Hough. Para ambas subfiguras, primero se aplicó el detector de bordes de Canny y luego la transformada de Hough, donde los resultados de ambos algoritmos se intersectan se



(a) Transformada de Hough para líneas. (b) Transformada de Hough para círculos.

Figura 18: Implementación de la transformada de Hough [14].

Algoritmo	**Utilidad**
Harris	Detección de esquinas.
SIFT	Detección de regiones.
SURF	Detección de regiones.
FAST	Detección de esquinas.
BRIEF	Detección de regiones.
ORB	Detección de regiones y esquinas.

Cuadro 1: Algoritmos de detección de características [16].

muestra líneas blancas. Se observa que, en la Figura 18a, las líneas descritas logran coincidir, en su mayoría, con los bordes de las imágenes presentadas. Por otro lado, en la Figura 18b se nota como se tiene buena coincidencia en un uno de los círculos de la imagen superior y no detecta ninguno en la imagen inferior.

6.6. Detección de características y emparejamiento

La detección de características en un punto es útil para encontrar su localización correspondiente en diferentes imágenes [15].

Existen dos aproximaciones para encontrar las características de un punto y sus correspondencias. La primera consiste en realizar un rastreo preciso utilizado técnicas de búsqueda locales, como correlación o mínimos cuadrados. Este tipo de aproximación es adecuado cuando las imágenes son tomadas desde puntos de vista cercanos, o bien, bajo una rápida sucesión como en secuencias de vídeo. La segunda aproximación consiste en detectar las características de forma independiente en todas las imágenes bajo consideración, y luego realizar en emparejamiento basado en apariencias locales. Esta aproximación es adecuada cuando se tenga gran cantidad de movimiento o cambios de apariencia esperados, por ejemplo, al juntar panoramas o realizar reconocimiento de objetos [15].

Algunos algoritmos de detección y sus usos respectivos se muestran en el Cuadro 1.

6.6.1. Segmentación

La segmentación de imagen es la tarea de buscar grupos de píxeles que van juntos. Esto se logra por medio de marcadores espaciales y restricciones en la imagen [15]. En esta sección se presentan algunas técnicas de segmentación.

Separación de región

Esta técnica calcula el histograma de toda la imagen, luego busca un *threshold* que separe los picos grandes en el histograma. Este proceso se repite hasta que las regiones se hayan separado de manera uniforme o estén por debajo de un tamaño específico [15].

Combinación de región

Consiste en la combinación de regiones adyacentes cuya diferencia de color promedio esté por debajo de un umbral o bien, cuyas regiones sean muy pequeñas [15].

Cálculo de cuenca (*Watershed)*

Técnica relacionada al *thresholding*, en esta, se segmenta una imagen en diversas regiones interpretadas como valles y crestas. Las regiones son descubiertas al inundar los valles en todos los mínimos locales y etiquetar riscos en donde la evolución de la función cambia [15].

Segmentación basada en gráficas

Consiste en un algoritmo de combinación que utiliza disimilitudes relativas entre regiones para determinar las que deben ser combinadas. Se inicia con una medida de disimilitud píxel a píxel, la cual mide alguna característica, como por ejemplo, la diferencia de intensidad entre $\mathcal{N}_8$ vecinos [15].

Cambios de media

Esta técnica modela los vectores de características asociados con cada píxel como muestras de una función probabilística de densidad desconocida y luego busca grupos en la distribución [15].

Cortes normalizados

Técnica que examina las afinidades entre píxeles cercanos y trata de separar grupos que estén conectados por afinidades débiles [15]. Un ejemplo de segmentación por medio de esta técnica se muestra en la Figura 19.

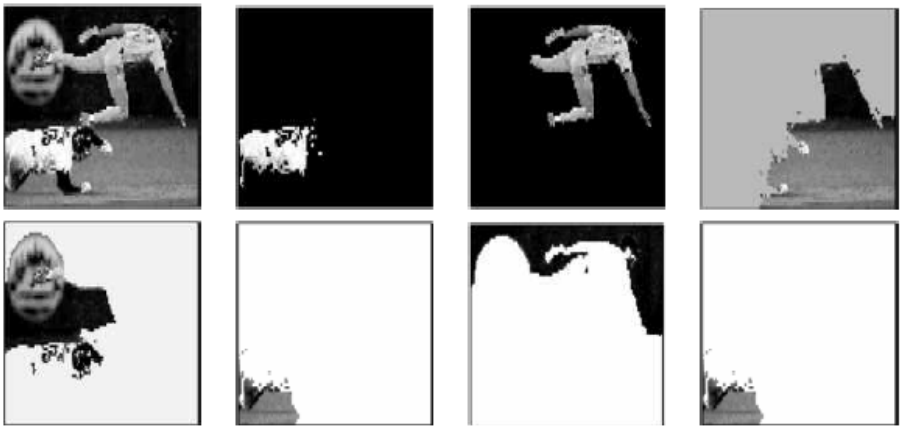


Figura 19: Segmentación por cortes normalizados [15].

6.7. Marcadores de referencia

Los marcadores de referencia, también conocidos como *fiducial markers*, son puntos de referencia artificiales añadidos a una escena utilizados para facilitar la localización de puntos correspondientes entre imágenes y un modelo conocido [17].

Son útiles en situaciones en las que el reconocimiento de un objeto o determinación de su pose es necesaria con alta fiabilidad o bien en situaciones en las que las características naturales no están presentes en suficiente cantidad y unicidad. Algunas aplicaciones de ejemplo incluyen realidad aumentada en interiores, mensajes de referencia para activar un comportamiento o determinación genérica de pose en escenarios industriales [17].

La Figura 20 presenta tres de tipos de marcadores de referencia.

AprilTags utiliza segmentación basada en gráficos con gradientes locales para estimar líneas e introduce un método de detección cuádruple diseñado para manejar oclusiones. Al haber detectado las líneas, los AprilTags dependen de un *threshold* adaptativo utilizando valores conocidos de píxeles blancos y negros para decodificar el peso. Esto hace el sistema de detección robusto a variaciones de iluminación [18].

ARToolKit utiliza un *threshold* global para crear una imagen binaria y un método de correlación de imagen en el que el peso es comparado con una base de datos predefinida. Esto permite crear marcadores con patrones intuitivos pero también incrementa el esfuerzo computacional cuando se tiene bases de datos grandes pues cada marcador potencial debe ser verificado contra la base de datos entera [18].

ArUco es una librería para aplicaciones de realidad aumentada basadas exclusivamente en OpenCV. Depende de marcadores en blanco y negro con códigos detectados al llamar una única función [19]. ArUco propone un método para crear diccionarios con números configurables de marcadores y número de bits. Este método maximiza las transiciones de bit y la diferencia entre marcadores para reducir falsos positivos y confusiones en índices respectivamente. Esta librería también presenta un método de corrección de error. El proceso consiste en la aplicación de un *threshold* adaptativo a una imagen en escala de grises y

Figura 20: Ejemplos de marcadores de referencia [20]

encontrar los candidatos del marcador al descartar los contornos que no se aproximan a un rectángulo [18]. Para este estudio se utilizó marcadores ArUco.

6.8. Control basado en visión

También conocido en la literatura como *visual servoing*, es un método de control el cual incorpora de forma directa información visual dentro del lazo de control [12]. La tarea en el control basado en visión consta de controlar la pose de un robot utilizando información visual, es decir, características extraídas de una imagen [21].

Existen dos aproximaciones al control basado en visión [22]:

- PBVS (*Position Based Visual Servoing*): el control basado en posición utiliza características visuales obtenidas, una cámara calibrada y un modelo geométrico conocido del objetivo para estimar su pose respecto a la cámara.
- IBVS (*Image Based Visual Servoing*): el control basado en imagen no estima la pose, dado que esta ya está implícita en los valores de las características de la imagen.

Para ejemplificar de mejor forma cada una de las aproximaciones se presenta la Figura 21.

6.8.1. Control basado en imágenes

Este estudio se enfoca en el uso de IBVS por lo que es pertinente profundizar un poco más en el tema.

El *visual servoing* basado en imágenes ignora la naturaleza tridimensional del objeto a rastrear, considerando únicamente la información medida por la cámara, es decir, la proyección de la geometría tridimensional al plano de imagen. La idea es plantear un problema de control que relacione la dinámica de las características de la imagen (*image features*) en el plano de imagen con la capacidad de movimiento de la cámara y, por ende, del robot.

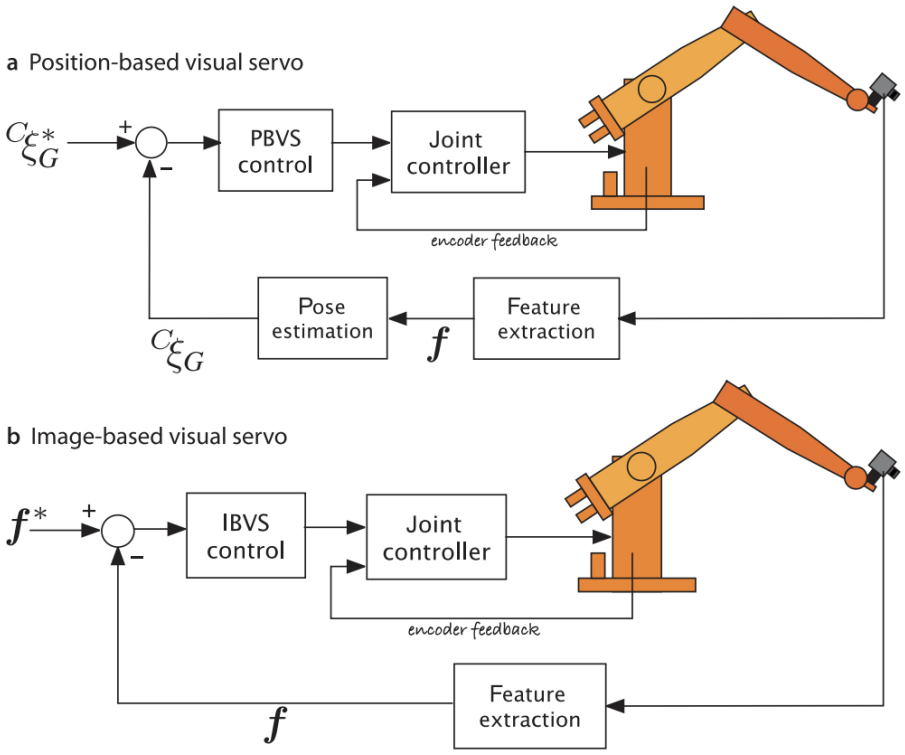


Figura 21: Aproximaciones para el control basado en visión [22].

Las características de imagen son almacenadas en un vector $\mathbf{s} = \begin{bmatrix} \mathbf{s}_1 & \mathbf{s}_2 & \dots & \mathbf{s}_N \end{bmatrix}^\top$ llamado vector de características (*feature vector*) [22].

Se considera el caso más sencillo de un punto en el plano de imagen como única característica del vector de características, es decir, las coordenadas horizontal y vertical respecto al plano de imagen:

$$\mathbf{s} = \begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} = \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix}.$$

La dinámica de la característica se puede determinar siguiendo la deducción de [23]:

$$u - u_0 = \bar{u} = \frac{f\,{}^C x}{\rho_w\,{}^C z}, \qquad \Rightarrow \dot{\bar{u}} = \frac{f\rho_w\,{}^C\dot{x}\,{}^C z - f\rho_w\,{}^C x\,{}^C\dot{z}}{\rho_w^2\,{}^C z^2},$$

$$v - v_0 = \bar{v} = \frac{f\,{}^C y}{\rho_h\,{}^C z}, \qquad \Rightarrow \dot{\bar{v}} = \frac{f\rho_h\,{}^C\dot{y}\,{}^C z - f\rho_h\,{}^C y\,{}^C\dot{z}}{\rho_h^2\,{}^C z^2},$$

con lo cual se obtiene:

$$\begin{bmatrix}\dot{\bar{u}}\\ \dot{\bar{v}}\end{bmatrix} = \begin{bmatrix}\frac{f}{\rho_w {}^C z} & 0 & -\frac{\bar{u}}{{}^C z}\\ 0 & \frac{f}{\rho_h {}^C z} & -\frac{\bar{v}}{{}^C z}\end{bmatrix} \underbrace{\begin{bmatrix}{}^C\dot{x}\\ {}^C\dot{y}\\ {}^C\dot{z}\end{bmatrix}}_{{}^C\dot{\mathbf{p}}}.$$

De cinemática diferencial de cuerpos rígidos se sabe que:

$${}^C\dot{\mathbf{p}} = {}^C\mathbf{R}_B\, {}^B\dot{\mathbf{p}} - \mathcal{S}({}^C\boldsymbol{\omega}_{BC}){}^C\mathbf{p} - \underbrace{{}^C\mathbf{R}_B\, {}^B\dot{\mathbf{o}}_C}_{{}^C\mathbf{v}_C},$$

dado que $\mathcal{S}({}^C\boldsymbol{\omega}_{BC}){}^C\mathbf{p}$ representa un producto cruz, el cual es anti-conmutativo, es equivalente a $-\mathcal{S}({}^C\mathbf{p}){}^C\boldsymbol{\omega}_{BC}$ Asimismo nótese que ${}^C\mathbf{v}_C$ representa la velocidad lineal de la cámara con respecto de la base pero expresada en el marco de referencia de la cámara [23]. Finalmente, si se considera que el punto a controlar en el plano de imagen no se mueve con respecto de la base del robot entonces ${}^B\dot{\mathbf{p}} = \mathbf{0}$ y se obtiene:

$${}^C\dot{\mathbf{p}} = \mathcal{S}({}^C\mathbf{p}){}^C\boldsymbol{\omega}_{BC} - {}^C\mathbf{v}_C = \begin{bmatrix}-\mathbf{I} & \mathcal{S}({}^C\mathbf{p})\end{bmatrix}\begin{bmatrix}{}^C\mathbf{v}_C\\ {}^C\boldsymbol{\omega}_{BC}\end{bmatrix}.$$

Antes de continuar, se define una expresión para las coordenadas normalizadas:

$$u = \frac{f\,{}^C x}{\rho_w {}^C z} - u_0, \qquad\qquad v = \frac{f\,{}^C y}{\rho_h {}^C z} - v_0. \tag{36}$$

Por lo tanto la expresión queda de la siguiente forma:

$$\begin{bmatrix}\dot{\bar{u}}\\ \dot{\bar{v}}\end{bmatrix} = \begin{bmatrix}\frac{f}{\rho_w {}^C z} & 0 & -\frac{\bar{u}}{{}^C z}\\ 0 & \frac{f}{\rho_h {}^C z} & -\frac{\bar{v}}{{}^C z}\end{bmatrix}\begin{bmatrix}-\mathbf{I} & \mathcal{S}({}^C\mathbf{p})\end{bmatrix}\begin{bmatrix}{}^C\mathbf{v}_C\\ {}^C\boldsymbol{\omega}_{BC}\end{bmatrix},$$

y aplicando 36 a esto se obtiene:

$$\begin{bmatrix}\dot{\bar{u}}\\ \dot{\bar{v}}\end{bmatrix} = \begin{bmatrix}-\frac{f}{\rho_w {}^C z} & 0 & \frac{\bar{u}}{{}^C z} & \frac{\rho_h \bar{u}\bar{v}}{f} & -\frac{f^2+\rho_w^2\bar{u}^2}{f\rho_w} & \frac{\rho_h \bar{v}}{\rho_w}\\ 0 & -\frac{f}{\rho_h {}^C z} & \frac{\bar{v}}{{}^C z} & \frac{f^2+\rho_h^2\bar{v}^2}{f\rho_h} & -\frac{\rho_w \bar{u}\bar{v}}{f} & -\frac{\rho_w \bar{u}}{\rho_h}\end{bmatrix}\begin{bmatrix}{}^C\mathbf{v}_C\\ {}^C\boldsymbol{\omega}_{BC}\end{bmatrix}.$$

En la práctica, usualmente se tiene $\rho_w = \rho_h = \rho$ y la longitud focal en píxeles puede ser definida como $f' = f/\rho$. Adicionalmente, se asume que el marco de referencia de la cámara coincide con el del efector final, es decir, $\{C\} = \{E\}$ [23]. Por lo que finalmente se obtiene:

$$\dot{\mathbf{s}} = \begin{bmatrix}\dot{\bar{u}}\\ \dot{\bar{v}}\end{bmatrix} = \underbrace{\begin{bmatrix}-\frac{f'}{{}^C z} & 0 & \frac{\bar{u}}{{}^C z} & \frac{\bar{u}\bar{v}}{f'} & -\frac{f'^2+\bar{u}^2}{f'} & \bar{v}\\ 0 & -\frac{f'}{{}^C z} & \frac{\bar{v}}{{}^C z} & \frac{f'^2+\bar{v}^2}{f'} & -\frac{\bar{u}\bar{v}}{f'} & -\bar{u}\end{bmatrix}}_{\mathbf{L}_p(\mathbf{s},{}^C z)}\begin{bmatrix}{}^E\mathbf{v}_E\\ {}^E\boldsymbol{\omega}_{BE}\end{bmatrix}, \tag{37}$$

en donde $\mathbf{L}_p(\mathbf{s},{}^C z)$ es conocida como la matriz de interacción correspondiente a una característica (de imagen) puntual. Esta matriz brinda una relación entre la velocidad de las características de imagen con la velocidad de la cámara. Por otro lado, el hecho de hacer coincidir $\{C\}$ con $\{E\}$ permite observar que la velocidad de las características en el plano de imagen puede relacionarse con la velocidad de la configuración del manipulador [23], esto se logra de la siguiente manera:

$$\dot{\mathbf{s}} = \mathbf{L}_p(\mathbf{s},{}^C z) \begin{bmatrix} {}^B\mathbf{R}_E^\top & \mathbf{0} \\ \mathbf{0} & {}^B\mathbf{R}_E^\top \end{bmatrix} = \mathbf{L}_p(\mathbf{s},{}^C z) \underbrace{\begin{bmatrix} {}^B\mathbf{R}_E^T\top(\mathbf{q}) & \mathbf{0} \\ \mathbf{0} & {}^B\mathbf{R}_E^\top(\mathbf{q}) \end{bmatrix} \mathbf{J}(\mathbf{q})}_{{}^E\mathbf{J}(\mathbf{q})} \dot{\mathbf{q}},$$

$$\dot{\mathbf{s}} = \mathbf{J}_L(\mathbf{q}, \mathbf{s},{}^C z)\dot{\mathbf{q}}, \tag{38}$$

en donde $\mathbf{J}_L(\mathbf{q}, \mathbf{s},{}^C z)$ es llamado el Jacobiano de imagen. Nótese que este jacobiano reúne el comportamiento cinemático de la cámara como del manipulador. Esta relación se puede explotar para plantear de manera sencilla el problema de *visual servoing* basado en imágenes al suponer que se quiere hacer que las características de imagen $\mathbf{s}$ presenten un comportamiento $\mathbf{s}_d$ fijo [23], de lo cual su error y dinámica es:

$$\mathbf{e}_\mathbf{s} = \mathbf{s}_d - \mathbf{s}.$$

Si se asume que se puede controlar al manipulador para que ejecute cualquier velocidad $\dot{\mathbf{q}}_{ref}$ en su configuración, se puede utilizar el vector $\dot{\mathbf{q}}$ como entrada para plantear el controlador:

$$\dot{\mathbf{q}} = \mathbf{J}_L^\dagger(\mathbf{q}, \mathbf{s},{}^C z)\mathbf{K}\mathbf{e}_\mathbf{s},$$

que hace que el error presente en lazo cerrado la dinámica LTI:

$$\dot{\mathbf{e}}_\mathbf{s} = -\mathbf{K}\mathbf{e}_\mathbf{s},$$

la cual es globalmente asintóticamente estable siempre y cuando $\mathbf{K} \succ 0$. Este controlador también puede implementarse mediante un algoritmo iterativo para la configuración del robot:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}_L^\dagger(\mathbf{q}_k, \mathbf{s}[k],{}^C z[k])\mathbf{K}\mathbf{e}_\mathbf{s}[k], \tag{39}$$

asumiendo que $\mathbf{K}$ absorbe el período de muestro de la discretización [23].

CAPÍTULO 7

Metodología

7.1. OpenCV

Las características de los módulos de visión a comparar, JeVois Smart Vision Camera y ESP32-CAM, fueron estudiadas al realizar una serie de experimentos de procesamiento de imágenes básicos.

Como base de los experimentos se utilizó la librería OpenCV. Esta fue seleccionada por la facilidad de aplicación de algoritmos de procesamiento de imagen y visión por computadora, además de su versatilidad para poder ser utilizado en C++ y Python.

Los experimentos realizados, o que se pretendió realizar, en cada módulo fueron:

1. Operaciones de *thresholding*.
2. Detección de bordes.
3. Detección de contornos.
4. Extracción de características:
   a) Aplicación de transformada de Hough.
   b) Aplicación de transformada de Hough para círculos.
5. Marcadores de referencia (*fiducial markers*):
   a) Detección de marcadores ArUco.

La mayoría de los algoritmos fue evaluado con una escena compuesta por tres objetos de enfoque principales: lentes, personaje y cubo de rompecabezas. Se procuró capturar escenas

similares con ambos módulos, procurando mantener la mayor cantidad de detalle igual; la posición, distancia, iluminación, etc. Se seleccionó estos sujetos dada su topología, ya que presentan características distintas con las que se puede extraer variedad de información al capturar la escena.

Los algoritmos que no utilizaron esta escena (transformadas de Hough y detección de marcadores) fueron evaluados presentado imágenes con las geometrías explícitas, es decir, líneas, círculos y marcadores. Esto para facilitar la detección de las características deseadas.

7.2. *Software*

Según el módulo a estudiar, se utilizó aplicaciones diferentes para programar los algoritmos.

7.2.1. *Software* para JeVois Smart Vision Camera

En el caso de JeVois, se utilizó el *software* provisto por el fabricante: JeVois Inventor [24].

JeVois Inventor es una plataforma que cuenta con diversos módulos de visión provistos por el fabricante, además de dar la facilidad de programar módulos personalizados y probarlos de forma instantánea.

JeVois Smart Vision Camera tiene la característica de poder variar el módulo de visión a implementar por medio del cambio de la resolución de la cámara. Por lo tanto, en caso se deseara utilizar alguna otra aplicación para visualizar el resultado de mejor forma se podía hacer, únicamente se tenía que memorizar la resolución de módulo. En ocasiones, para verificar la calidad de la imagen generada, se utilizó *Open Broadcast Software* (OBS) [25].

Se seleccionó OBS como *software* auxiliar dada su facilidad para agregar dispositivos de captura de vídeo y la resolución en la que se desea visualizar.

7.2.2. *Software* para ESP32-CAM

Para la ESP32-CAM se utilizó, como medio de programación del módulo, el IDE de Arduino [26], junto con la librería de ESP32 [27]. Estas herramientas fueron utilizadas únicamente para obtener captura de vídeo.

ESP32-CAM tiene la característica que transmite vídeo a internet, por lo tanto, el vídeo debe ser recuperado desde la dirección de transmisión para poder ser procesado. Tanto el módulo como la computadora en donde se desea hacer el procesamiento deben estar conectadas a la misma red. El procesamiento se realizó utilizando la extensión de Python en Visual Studio Code.

Parámetro	**Peso**
Calidad de imagen	5 %
Imagen en tiempo real	10 %
Desempeño en *threshold* binario	2 %
Desempeño en *threshold* binario invertido	2 %
Desempeño en *threshold* truncado	2 %
Desempeño en *threshold* a cero	2 %
Desempeño en *threshold* a cero invertido	2 %
Desempeño en *threshold* adaptativo a la media	2 %
Desempeño en *threshold* adaptativo Gaussiano	2 %
Desempeño en detección de bordes de Canny	14 %
Desempeño en detección de contornos	14 %
Desempeño en transformada de Hough	14 %
Desempeño en transformada de Hough para círculos	14 %
Desempeño en detección de marcadores ArUco	15 %

Cuadro 2: Valores de interés para la herramienta de comparación.

7.3. Herramienta de comparación

Una de la actividades principales de este proyecto fue la comparación de módulos capaces de dotar de visión por computadora a un sistema embebido. Para esto, se tomó en consideración distintos parámetros de estudio y realizó un cuadro de estudio (*trade study*).

7.3.1. Parámetros de estudio y pesos asignados

El Cuadro 2 muestra los campos que se estudiaron en cada uno de los módulos, así como los pesos asignados a cada uno. La idea de esta herramienta fue asignar una calificación al desempeño de los módulos y ponderarlos para realizar una selección objetiva.

Los pesos presentados en el Cuadro 2 fueron seleccionados según la importancia de cada parámetro en el proyecto general. La calidad de imagen no es la característica que más se busca en el estudio, sin embargo, fue útil para estudiar el desempeño de los algoritmos, por ello se asignó un peso bajo respecto al resto de parámetros a evaluar. Obtener una imagen en tiempo real fue un parámetro importante pues se buscó implementar un controlador capaz de actuar de manera rápida, esto se logró al seleccionar un módulo que pudiera actualizar la imagen proporcionada a una velocidad adecuada. Luego se puede observar una distribución equitativa en todos los algoritmos, excepto el último, al cual se le asignó más peso pues la detección de marcadores ArUco fue importante para la concretar la aplicación de control basado en visión.

7.3.2. Calificaciones

El Cuadro 3 muestra las calificaciones posibles al desempeño de cada uno de los parámetros mostrados en el Cuadro 2.

Adjetivo(s)	Calificación
No aceptable / Nulo	0
Muy malo / Muy bajo	1
Malo / Bajo	2
Moderado / Normal	3
Bueno / Alto	4
Muy bueno / Muy alto	5

Cuadro 3: Valores de interés para la herramienta de comparación.

7.4. Integración de brazo robot

Con la selección de un módulo de visión, se pudo avanzar en el proyecto a la fase de implementación de un controlador basado en visión. Este etapa del proyecto consistió en armar un brazo robot e integrar un controlador que permitiera el movimiento del mismo a partir de una señal enviada por la cámara.

7.4.1. Estructura de brazo robot

Se realizó la estructura física del brazo robot. Este se diseñó para contar con dos grados de libertad. Se tiene un motor inferior, el cual permite una movimiento horizontal, paralelo a la mesa de trabajo y otro que realiza movimientos verticales, perpendiculares a la mesa.

El prototipo del brazo se diseño en Autodesk Inventor [28] y se mandó a materializar por medio de corte láser. Teniendo las partes físicas se pasó a el ensamblaje, las uniones se hicieron por medio de tornillos, silicona fría y entradas a presión.

7.4.2. Controlador

El último paso del proyecto fue implementar un controlador para poder recibir una señal del módulo de visión y que lo motores reaccionaran a la posición de un marcador ArUco.

El microcontrolador seleccionado para la aplicación fue un Arduino Pro Micro. Este es de tamaño pequeño, cuenta con una cantidad suficiente de pines de entrada y salida y sólo se precisó de un puerto serial, por lo que cumplió con necesidades básicas de *hardware* y espacio para permitir un movimiento fluido del brazo. Además, Arduino cuenta con una amplia librería de ejemplos que facilitaron la implementación del controlador deseado.

CAPÍTULO 8

Algoritmos de visión por computadora

En este capítulo se presentan los resultados de la aplicación de algoritmos de visión por computadora clásicos, empleando OpenCV. En cada uno de los algoritmos se presenta una comparación entre las salidas de cada módulo de visión estudiado (JeVois Smart Vision Camera y ESP32-CAM). Asimismo, se presenta un análisis del desempeño del módulo en la prueba.

En Anexos se tiene un enlace que lleva a un vídeo de comparación directo entre los dos módulos.

8.1. Threshold

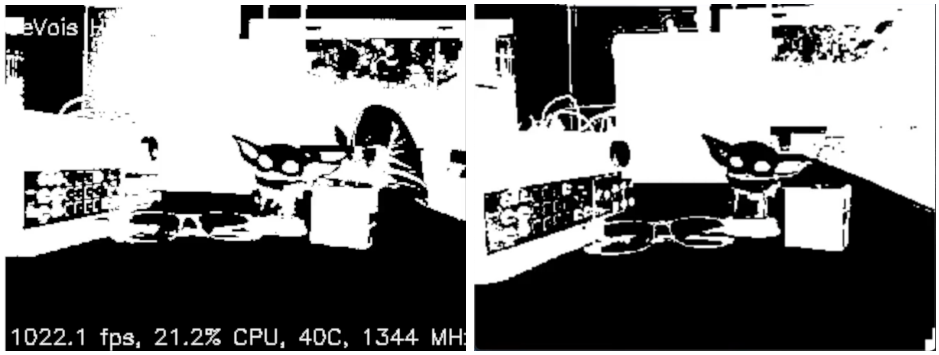
8.1.1. Threshold binario

La Figura 22 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* binario a la imagen.

El algoritmo presenta un desempeño similar en ambos módulos, se tiene una separación de colores aceptable para los dos casos. La escena capturada por JeVois, Figura 22a, presenta más información en el área central, por lo que se nota más detalle en la oreja del personaje. La escena capturada por ESP32-CAM, Figura 22b, muestra mejor definición en los bordes, ya que se nota mayor detalle en el generador de funciones del lado izquierdo de la imagen.

(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 22: Implementación de *threshold* binario.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 23: Implementación de *threshold* binario invertido.

8.1.2. Threshold binario invertido

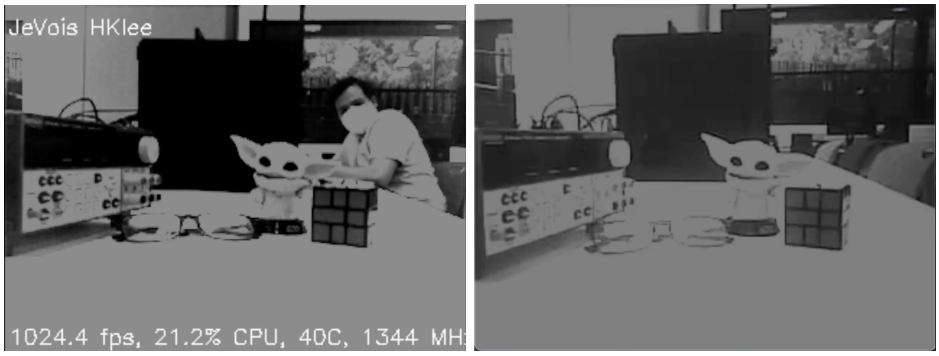
La Figura 23 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* binario invertido a la imagen.

Al observar la Figura 23 se notan varias similitudes. La Figura 23b muestra mayor detalle en la zona del generador de funciones, mientras que la Figura 23a presenta una respuesta más detallada en la zona central. La forma del interior de las orejas del personaje y el detalle de los *stickers* del cubo de rompecabezas es considerablemente mayor que en en esta captura. La parte superior del fondo presenta mayor detalle en la Figura 23a, en esta se puede apreciar el marco de la ventana trasera, a diferencia de la Figura 23b en la que no hay una forma definida del marco.

8.1.3. Threshold truncado

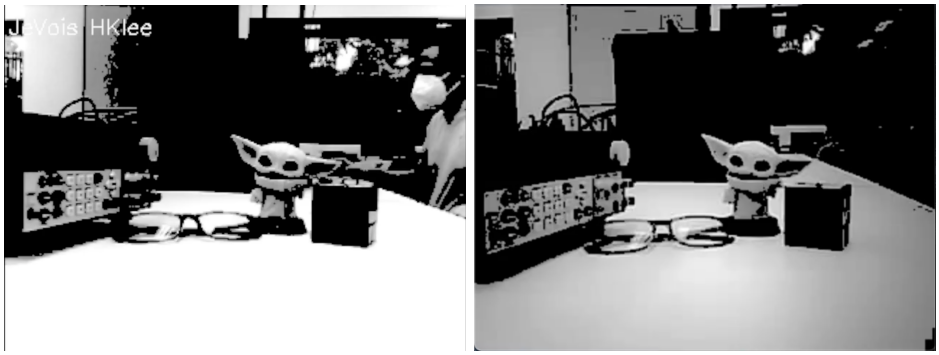
La Figura 24 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* truncado a la imagen.

La primera característica a comparar es la saturación de la imagen. Se puede observar cómo la Figura 24a tiene colores más saturados, esto se puede observar en el negro del CPU



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 24: Implementación de *threshold* truncado.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 25: Implementación de *threshold* a cero.

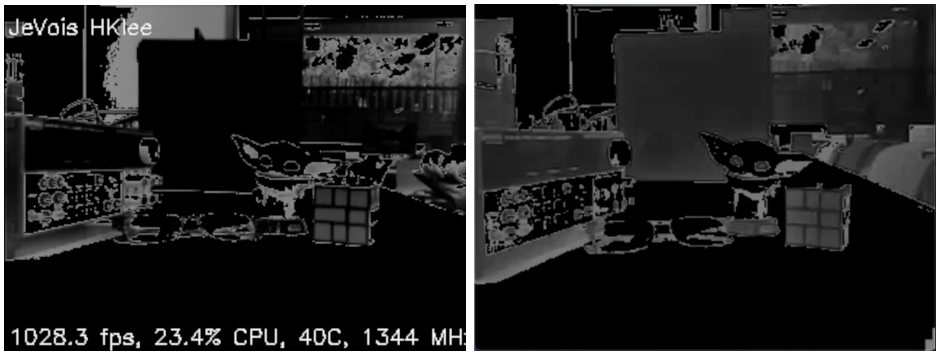
en el fondo, la base del personaje y las separaciones del cubo de rompecabezas. Es por esto que se puede apreciar más detalle en la captura de JeVois. A comparación, la captura de ESP32-CAM, Figura 24b, pareciera que se le aplicó un filtro de suavizado, pues los detalles en se encuentran difuminados.

8.1.4. Threshold a cero

La Figura 25 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* a cero a la imagen.

La saturación de colores entre las capturas de la Figura 25 es la primera característica a considerar. La Figura 25a presenta un negro más vibrante, esto le beneficia para obtener detalle en la zona baja de la imagen. Se nota mayor detalle en el interior de las orejas del personaje, además de un detalle fino en los *stickers* del cubo de rompecabezas. Por último, la forma del generador de funciones y sus botones, a la izquierda de la captura, están más definidos.

Sin embargo, se nota que no hay buena definición en el contorno del CPU al fondo, detalle que logró capturar bien el ESP32-CAM, Figura 25b. Esto da la pauta que mayor saturación en los colores no es una ventaja en todos los sentidos, se logra capturar mayor



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 26: Implementación de *threshold* a cero invertido.

cantidad de detalle en una zona al precio de bajar la calidad en otra.

8.1.5. Threshold a cero invertido

La Figura 26 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* a cero invertido a la imagen.

La diferencia más grande que se encuentra en la comparación de la Figura 26 es la definición de contornos. Se nota como en la Figura 26b el fondo fue llenado de un tono gris para lograr definir mejor a los sujetos centrales, mientras que en la Figura 26a los tonos son más oscuros y la definición entre objetos está definida por líneas de un tono más claro.

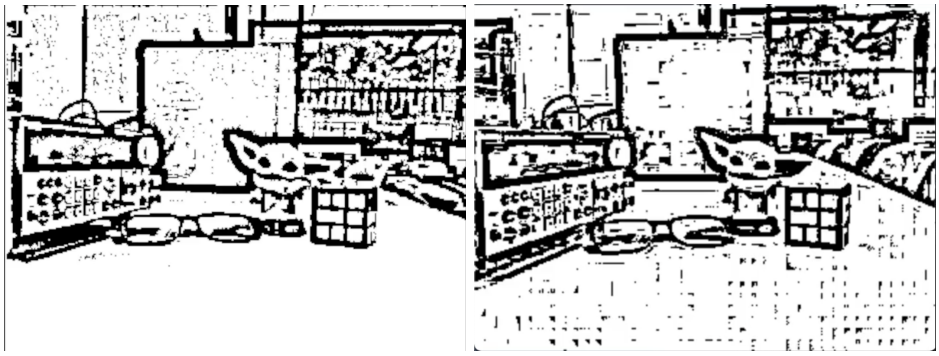
La definición en general de la captura de JeVois fue considerablemente mejor al aplicar este algoritmo. Los detalles están mejor definidos y no hay gradientes en los tonos de color para que sobresalgan otros sujetos por encima de otros.

8.1.6. Threshold adaptativo a la media

La Figura 27 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* adaptativo a la media a la imagen.

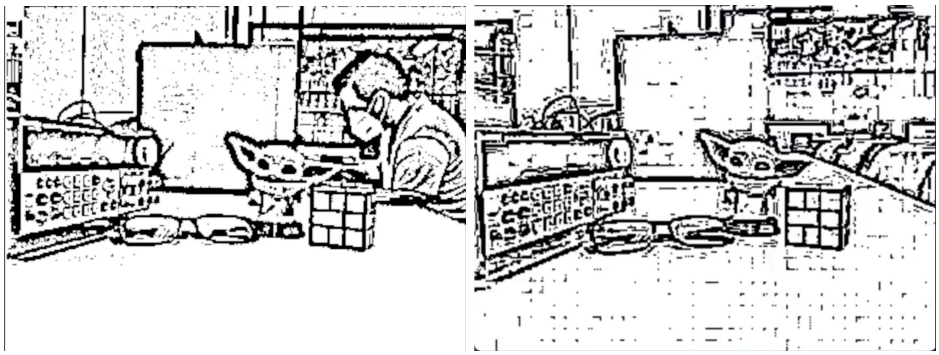
Al aplicar este algoritmo se nota una toma más limpia en la captura de JeVois, Figura 27a. La segunda característica a notar es la variación en el grosor de las líneas. La Figura 27b presenta líneas en su mayoría del mismo grosor, esto se nota en los detalles del la ropa del personaje, las divisiones del cubo de rompecabezas y el marco de los lentes. Este grosor, sin embargo, ayuda a capturar de mejor manera los botones del generador de funciones.

La Figura 27a presenta más líneas de distintos grosores, esto le ayuda a capturar los detalles de la baranda en el fondo de la imagen además de dar un detalle más fino en zonas más pequeñas como en la ropa del personaje, las divisiones del cubo de rompecabezas y el marco de los lentes.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 27: Implementación de *threshold* adaptativo a la media.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 28: Implementación de *threshold* adaptativo Gaussiano.

8.1.7. Threshold adaptativo Gaussiano

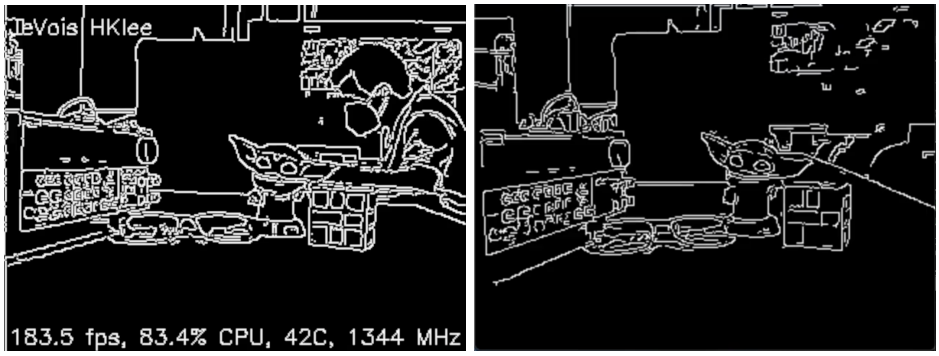
La Figura 28 muestra una comparación del desempeño de cada módulo al aplicar una operación de *threshold* adaptativo Gaussiano a la imagen.

Se nota una toma más limpia en la Figura 28a. El grosor de línea en ambas presenta variaciones que le ayudan a capturar las geometrías de buena manera. Una característica que se puede notar es que JeVois 28a utiliza líneas más finas para capturar detalles, se puede apreciar en el grueso del marco de los lentes y las divisiones del cubo de rompecabezas. Por otro lado, la forma en la que se obtiene los detalles en ESP32-CAM (Figura 28b) permite que, a pesar de ser una toma más sucia, se capturen de mejor forma los detalles del generador de funciones.

8.2. Detector de bordes de Canny

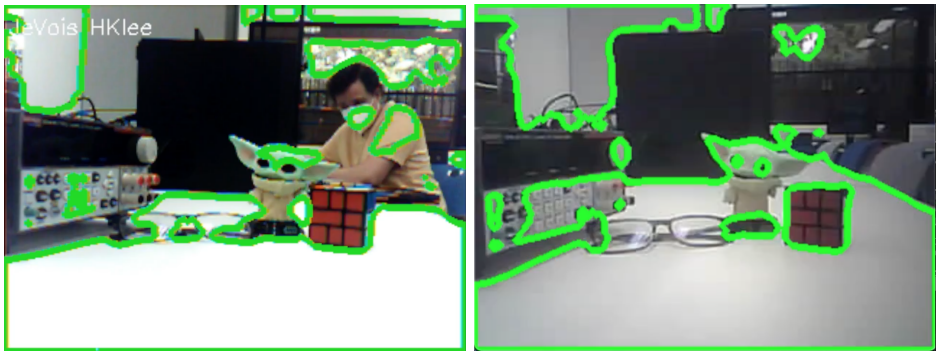
La Figura 29 muestra una comparación del desempeño de cada módulo al aplicar una operación de detección de bordes por medio del algoritmo de Canny a la imagen.

La aplicación de este algoritmo presenta una diferencia considerable. Se nota cómo la



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 29: Implementación de detección de bordes de Canny.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 30: Implementación de detección de contornos.

captura de JeVois (Figura 29a) presenta más detalle que la de ESP32-CAM (Figura 29b).

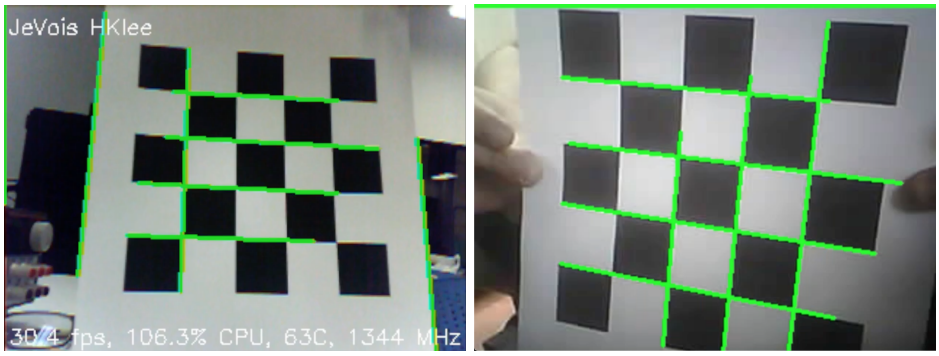
En la Figura 29a) se nota que existe detalle en la ropa del personaje, la cantidad de *stickers* del cubo de rompecabezas es mayor y más detallada y los botones del generador de funciones se distinguen sin problema. Para finalizar, la saturación en el color es considerablemente mayor que en la Figura 29b.

8.3. Detector de contornos

La Figura 30 muestra una comparación del desempeño de cada módulo al aplicar una operación de detección de contornos a la imagen.

La aplicación de este algoritmo presentó respuestas distintas en cada módulo. Se nota cómo en la Figura 30a, el algoritmo se dedicó a detectar el contorno blanco de la mesa de trabajo. La separación que realizó el algoritmo en este caso es acertada, se aprecia como procura seguir únicamente el color blanco. En zonas en las que encuentre un color diferente, se excluye de la sección contorneada y no trata de hacer una nueva.

Por otro lado, en la Figura 30b, se nota cómo trató de separar la mayor cantidad de colores, se aprecia cómo se contorneó el negro de los ojos del personaje y su base, y realizó



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 31: Implementación de transformada de Hough para líneas.

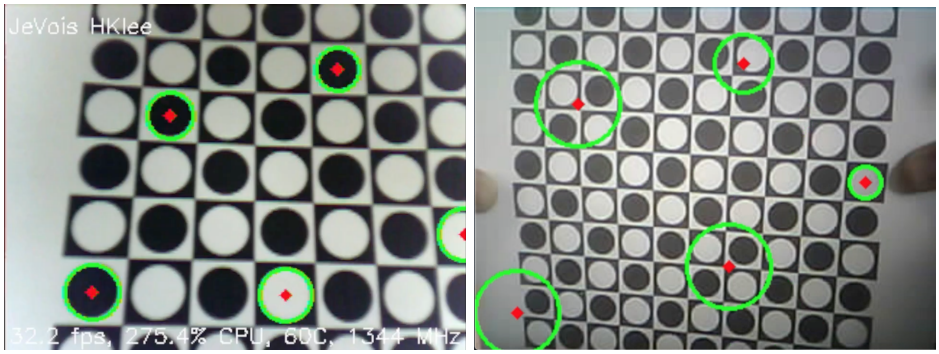
una nueva división al detectar el cambio de color en el cubo de rompecabezas. Sin embargo, el contorno específico de la mesa no es del todo correcto pues detecta parte del generador de funciones como parte de de la mesa, esto se puede explicar por la iluminación y saturación de color de la toma. El cambio de color entre la mesa y el generador no es lo suficientemente grande como para que el algoritmo detecte un cambio de región, esto se puede apreciar de mejor manera en la pared trasera. La captura de la Figura 30a presenta una cambio notable de blanco a gris, por lo que, siguiendo la lógica de detección de blancos, sólo realiza el contorno de una región. La captura de la Figura 30b no muestra un cambio de color, por lo que se detecta una única región.

8.4. Transformada de Hough para líneas

La Figura 31 muestra una comparación del desempeño de cada módulo al aplicar la transformada de Hough para líneas a la imagen.

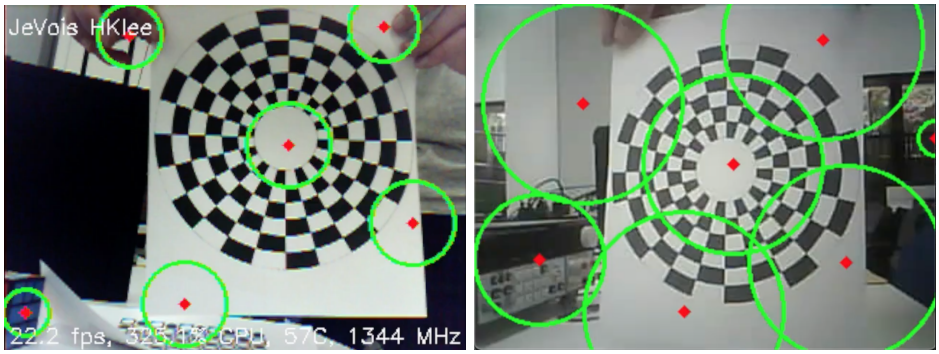
Al observar la aplicación de este algoritmo, se nota un desempeño bueno en ambos módulos. Fuera de las distinciones de saturación de color la respuesta es muy similar. Una diferencia interesante se encuentra en la detección del borde de la hoja por parte de JeVois en la Figura 31a, esta se puede deber a la iluminación de la imagen o bien a la orientación de la misma. El último detalle a mencionar, respecto al desempeño del algoritmo, es el largo de la línea detectada. Se observa que la Figura 31b presenta mayor cantidad de líneas detectadas además de ser más largas.

Una observación respecto a la inicialización del módulo para la aplicación del algoritmo es que se utilizó la misma base de código en ambos casos, sin embargo, la plataforma de desarrollo de JeVois presentó mayor estabilidad al momento de no detectar líneas en la marco inicial. En el caso de JeVois, la compilación de código era pausada hasta que se mostrara alguna figura a la que se le pudiera detectar líneas. En el caso de la plataforma de programación para la ESP32-CAM, la compilación del código finalizaba. Este comportamiento dio la noción que, en algoritmos más complejos, la JeVois Smart Vision Camera tiene hardware más capaz para continuar la captura de vídeo normal (sin ningún tipo de procesamiento) en caso el módulo de visión programado presentara errores temporales.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 32: Implementación de transformada de Hough para círculos 1.



(a) Algoritmo aplicado en JeVois. (b) Algoritmo aplicado en ESP32-CAM.

Figura 33: Implementación de transformada de Hough para círculos 2.

8.5. Transformada de Hough para círculos

Las Figuras 32 y 33 muestran una comparación del desempeño de cada módulo al aplicar la transformada de Hough para círculos a dos imagenes con círculos de distintos tamaños.

Como se puede observar, para el estudio de este algoritmo se utilizaron dos imágenes con características distintas para lograr capturar diferente información. La Figura 32 sirvió para verificar el desempeño de los módulos para detectar geometrías pequeñas, la Figura 33 se utilizó para determinar la capacidad de detectar círculos en una imagen circular.

Como se puede observar en la 32a, JeVois tuvo un mejor desempeño al poder detectar cinco círculos en la imagen, de hecho, todos los círculos generados por el algoritmo concuerdan con círculos reales en la figura. Por otro lado, ESP32-CAM detectó correctamente solo un círculo, las geometrías extra generadas por el algoritmo no concuerdan con círculos reales.

El desempeño de los módulos en la Figura 33 muestra que ambos módulos tienen dificultad detectando círculos. Se nota, de la Figura 33a, que JeVois detectó parcialmente el círculo central, mientras que, de la Figura 33b, ESP32-CAM detectó parcialmente un círculo en el centro de la imagen.

(a) Aplicado a múltiples marcadores. (b) Aplicado a un solo marcador

Figura 34: Implementación de detección de marcadores ArUco.

Se hace la aclaración que en el caso de JeVois los parámetros de detección para la Figura 33a fueron los mismos que los utilizados en de la Figura 32. En el caso del resultado mostrado en la Figura 33b, se cambió los parámetros para poder identificar círculos de mayor radio. Esto da una pauta de la poca versatilidad que ofrece ESP32-CAM en la aplicación de este algoritmo.

Para finalizar, el problema de la detección inicial mencionada en la aplicación de la transformada de Hough para líneas se aplicó también en este algoritmo, reafirmando la comparación de capacidad de manejo de entre los módulos siendo mejor JeVois Smart Vision Camera.

8.6. Detector de marcadores ArUco

Esta prueba se logró realizar únicamente con el módulo JeVois Smart Vision Camera. Dadas las limitaciones de la ESP32-CAM no se pudo obtener un resultado que permitiera realizar una comparación.

La Figura 34 muestra el desempeño de la JeVois Smart Vision Camera al aplicar una detección de marcadores ArUco a una imagen con un solo marcador, Figura 34b, o múltiples Figura 34a.

Observando los resultados de la Figura 34, puede notarse que la JeVois Smart Vision Camera es un módulo capaz de identificar de manera satisfactoria marcadores ArUco sin importar la cantidad que se le presente. El resultado de la detección en la Figura 34a no varía con el de la Figura 34b, por lo que da una pauta de la versatilidad del módulo al realizar detección de marcadores de referencia.

CAPÍTULO 9

Comparación de módulos de visión

En este capítulo se presenta el resultado del estudio comparativo propuesto. Se presentan los cuadros 4 y 5 que muestran las calificaciones otorgadas a cada parámetro de JeVois y ESP32-CAM respectivamente.

Observando el Cuadro 4 se nota que las calificaciones de JeVois fueron mayormente positivas, siendo la calificación más baja obtenida un 4 que, según el Cuadro 3 representa una respuesta buena o alta. Los parámetros en los que este módulo no obtuvo la mejor calificación fueron:

1. *Threshold* a cero.
2. *Threshold* adaptativo a la media.
3. Aplicación de transformada de Hough.
4. Aplicación de transformada de Hough para círculos.

La justificación de la calificación para el *threshold* a cero es que hubo pérdida de información en algunas zonas de la imagen. Se nota en la Figura 25a que se tiene definición aceptable en la mayoría de la imagen, sin embargo, en la zona superior no se distingue el contorno del CPU y este se mezcla con el fondo de la imagen. El *threshold* adaptativo a la media no fue calificado de la mejor manera dada la comparación realizada en la Figura 27, en la que se nota que algunos detalles son capturados de mejor manera en la ESP32-CAM. A pesar de ser una imagen más sucia, se nota mayor detalle en el generador de funciones.

En el caso de la transformada de Hough para líneas, no se dio una calificación completa por la comparación de desempeño presentada en la Figura 31, en donde se nota que el resultado de la ESP32-CAM (Figura 31b) es levemente mejor que el de JeVois (Figura 31a)

Estudio a JeVois Smart Vision Camera		
Parámetro	**Peso**	**Calificación**
Calidad de imagen	5 %	5
Imagen en tiempo real	10 %	5
Desempeño en *threshold* binario	2 %	5
Desempeño en *threshold* binario invertido	2 %	5
Desempeño en *threshold* truncado	2 %	5
Desempeño en *threshold* a cero	2 %	4
Desempeño en *threshold* a cero invertido	2 %	5
Desempeño en *threshold* adaptativo a la media	2 %	4
Desempeño en *threshold* adaptativo Gaussiano	2 %	5
Desempeño en detección de bordes de Canny	14 %	5
Desempeño en detección de contornos	14 %	5
Desempeño en transformada de Hough	14 %	4
Desempeño en transformada de Hough para círculos	14 %	4
Desempeño en detección de marcadores ArUco	15 %	5
Calificación promediada		4.71
Calificación pesada		**4.68**

Cuadro 4: Resultados del estudio aplicado a JeVois Smart Vision Camera.

dado que se detectó mayor cantidad de líneas verticales y la longitud de las líneas, tanto verticales como horizontales también fue mayor. Para finalizar, la calificación respectiva a la transformada de Hough para círculos se debe al bajo desempeño en la prueba realizada en la Figura 33a, en donde se nota que se trató de detectar un círculo pero los centros no coincidieron del todo.

Observando el Cuadro 5 se nota que las calificaciones de ESP32-CAM fueron bastante fluctuantes, siendo la calificación más llamativa un 0 que, según el Cuadro 3 representa una respuesta no aceptable o nula. Los parámetros en los que este módulo no obtuvo la mejor calificación fueron:

1. Calidad de imagen.
2. Imagen en tiempo real.
3. *Threshold* truncado.
4. *Threshold* a cero invertido.
5. Detección de bordes por medio del algoritmo de Canny.
6. Aplicación de transformada de Hough.
7. Aplicación de transformada de Hough para círculos.
8. Detección de marcadores ArUco.

La calificación que se dio a la calidad de imagen se justifica principalmente por comparación con la imagen que presenta JeVois, en donde se percibe que la imagen de la ESP32-CAM

Estudio a ESP32-CAM		
Parámetro	**Peso**	**Calificación**
Calidad de imagen	5 %	4
Imagen en tiempo real	10 %	2
Desempeño en *threshold* binario	2 %	5
Desempeño en *threshold* binario invertido	2 %	5
Desempeño en *threshold* truncado	2 %	4
Desempeño en *threshold* a cero	2 %	5
Desempeño en *threshold* a cero invertido	2 %	3
Desempeño en *threshold* adaptativo a la media	2 %	5
Desempeño en *threshold* adaptativo Gaussiano	2 %	5
Desempeño en detección de bordes de Canny	14 %	3
Desempeño en detección de contornos	14 %	5
Desempeño en transformada de Hough	14 %	4
Desempeño en transformada de Hough para círculos	14 %	3
Desempeño en detección de marcadores ArUco	15 %	0
Calificación promediada		3.79
Calificación pesada		**3.14**

Cuadro 5: Resultados del estudio aplicado a ESP32-CAM.

es muy difuminada, aparentando que se le aplicó un filtro de suavizado por lo que el estudio en imágenes complejas que requieren de mayor detalle no es recomendable. Por otro lado, una de las calificaciones más bajas otorgadas a este módulo fue la imagen en tiempo real. Dadas las limitaciones del módulo, la imagen primero debe ser transmitida por internet y luego recuperada en Python marco por marco. En el caso del código realizado, la actualización de marco se hacía cada milisegundo, pero la velocidad de transmisión no fue suficiente para capturar todos los movimientos que se le presentaban a la cámara en tiempo real. Esta es una desventaja pues, si se desea realizar control basado en visión, el tiempo de procesamiento de imagen debe ser rápido para que la reacción del resto del sistema también lo sea.

Pasando a las calificaciones de los algoritmos, se dio la calificación mostrada en el Cuadro 5 al *threshold* truncado dada la baja definición de la imagen presentada a comparación del módulo JeVois. La saturación del color es baja, por lo que no se logra capturar de manera satisfactoria algunos detalles a comparación de JeVois. En el caso del *threshold* a cero invertido, se nota que la definición de contornos no es la mejor y se aprecia cómo se aplicó una máscara de color gris para que resalten más los sujetos centrales. Al hacer esto, se perdió detalle en elementos como la pantalla del generador de funciones y la base del personaje.

La calificación que se le dio a la detección de bordes se justifica en el detalle que presenta a comparación del módulo JeVois. Observando la Figura 29 se puede apreciar como los detalles finos, como los *stickers* del cubo de rompecabezas, la ropa del personaje y los botones del generador de funciones son mejores en la Figura 29a que en la Figura 29b.

Con respecto a la aplicación de las transformadas de Hough, el criterio principal para la calificación se basó en el desempeño al inicializar el módulo de visión. Este simplemente

se detenía si no se encontraba desde el inicio la geometría deseada, por lo que demuestra poca amigabilidad con el usuario. Sumando a esto, la transformada de Hough para círculos no presentó resultados satisfactorios. En las Figuras 32b y 33b se aprecia que, de todos los intentos de detección de círculos, muy pocos son reales. En la Figura 32b únicamente una de las propuestas fue correcta, mientras que en la Figura 33b la detección es parcialmente correcta pues el centro propuesto no coincide del todo con el real. A esto se le agrega que para la obtención de esta última figura, se tuvo que realizar modificaciones a los parámetros de detección, lo que demuestra poca versatilidad en la aplicación del algoritmo.

Para finalizar, dadas la limitaciones del módulo no se pudo realizar la detección de marcadores ArUco. La principal limitación siendo que el módulo utilizado cuenta únicamente con un puerto serial, por lo que aun si se pudiera realizar la detección de marcadores no serviría para avanzar en el proyecto pues no se podría establecer comunicación serial entre el microcontrolador, el módulo y la computadora con un único puerto serial, se necesitaría al menos dos: uno que reciba la imagen y la mande a internet (conexión entre módulo y computadora) y otro que tomara la imagen procesada y la mande al microcontrolador (conexión entre módulo y microcontrolador). Para poder establecer este sistema, se debería utilizar algún componente extra, no únicamente la ESP32-CAM.

CAPÍTULO 10

Control basado en visión

10.1. Brazo robot

En las Figuras 35 y 36 se muestra el brazo robot de dos grados de libertad construido para realzar las pruebas de control basado en visión. En la Figura 36b se nota el circuito armado con un Arduino Micro Pro, el sistema embebido que funciona como controlador.

10.2. Planteamiento general

El brazo robot presentado en las Figuras 35 y 36 consiste en dos motores servo, uno que realiza movimientos en paralelo a la mesa de trabajo y otro perpendicular a esta. Los motores están conectados a dos pines del Arduino Micro Pro, instalado en la parte superior del robot. El microcontrolador recibe una señal, por medio de comunicación serial, por parte de la JeVois Smart Vision Camera.

El objetivo del robot es que el módulo de visión detecte un marcador de referencia, en este caso un marcador ArUco, y lo mantenga centrado en su plano de imagen. Esto se logró por medio de dos métodos de control: una combinación de controladores PD y un controlador basado en imágenes (IBVS). Los valores de los ángulos se actualizan según el movimiento propuesto por el marcador ArUco.

En Anexos se puede encontrar un enlace a un vídeo demostrativo del comportamiento del brazo robot realizado. A partir de este punto, la discusión se basa en el comportamiento que se puede observar en ese vídeo. Una observación previa: esta etapa del proyecto se realizó en conjunto con Luis Fernando de León García como parte de la línea de investigación de control basado en visión que se inició en la Universidad del Valle de Guatemala, es por esta

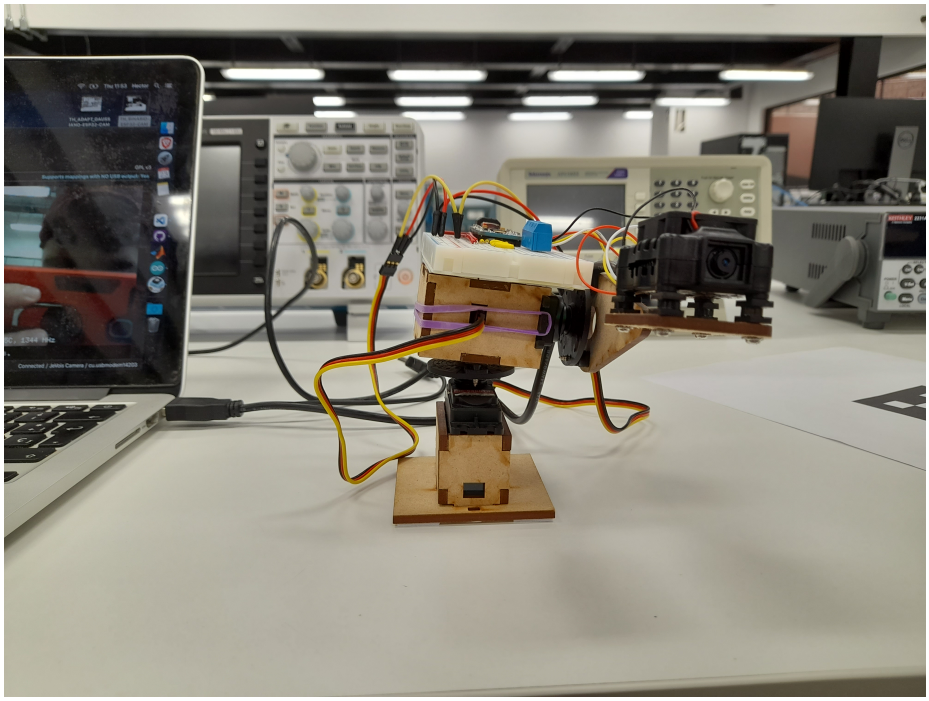


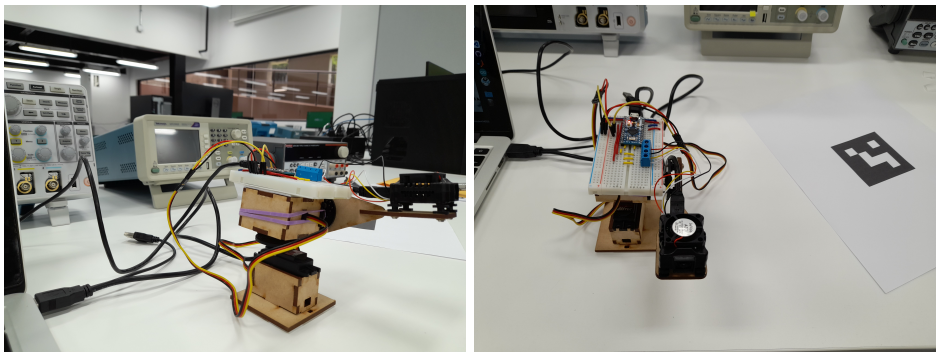
Figura 35: Vista frontal del brazo robot.

razón que el vídeo porta el nombre de ambos integrantes.

10.3. Controlador PD

El primer tipo de control planteado fue un controlador PD para cada uno de los motores. El comportamiento de este se puede observar a partir del segundo 7 hasta el segundo 48.

Se nota que la característica principal del brazo robot es la capacidad de poder seguir el marcador ArUco, centrándolo en el plano de imagen. Se puede observar cómo el movimiento de la persona que sujeta el marcador es relativamente lento, permitiendo al controlador



(a) Vista lateral del brazo robot. (b) Vista superior del brazo robot.

Figura 36: Vistas lateral y superior del brazo robot.

$\boldsymbol{\theta}$	**d**	**a**	$\boldsymbol{\alpha}$
q_1	-0.0997	0	1.5708
q_2	0	-0.0981	0

Cuadro 6: Parámetros de Denavit-Hartenberg.

obtener el dato y actualizarse constantemente.

Como observación del comportamiento, se logra apreciar que los movimientos que involucran acción en un único eje, ya sea solamente en el eje x o y, son suaves. Mientras que los movimientos que requieren de una coordenada combinada, es decir, movimiento tanto en x como en y, tienden a tener vibraciones. Esto se debe a que, en esencia se tiene una combinación de dos controladores PD correspondientes a cada uno de los motores del sistema. Por lo tanto, al realizar movimientos en un solo eje, únicamente uno de los controladores está actuando de forma activa, permitiendo un comportamiento fluido, mientras que en el caso de tener una coordenada combinada, se tiene dos controladores actuando de forma activa buscando predominar, resultando en oscilaciones en el movimiento y comportamientos bruscos.

10.4. Controlador basado en imágenes

El segundo tipo de control planteado fue un controlador aplicando el algoritmo de IBVS el cual toma en consideración la cinemática del sistema por lo que, a diferencia del controlador PD, solo se planteó un controlador para el sistema completo. El comportamiento de este se puede observar a partir del segundo 49 hasta el tiempo 1:33 (un minuto y treinta y tres segundos) Esta prueba constó de dos intentos, el primero en el que se utilizó el algoritmo descrito en (39) y el segundo con una modificación encontrada en la práctica. Ambos intentos fueron realizados utilizando el método de Levenberg-Marquardt para calcular la pseudoinversa $\mathbf{J}_L^{\dagger}(\mathbf{q}, \mathbf{s}, {}^{C}z)$. Este método se utilizó para procurar que las configuraciones propuestas del robot no fueran singularidades.

Dado que este tipo de controlador toma en consideración la cinemática del sistema, fue necesario plantear los parámetros de Denavit-Hartenberg, presentados en el Cuadro 6, plantear una transformación de base como una rotación en el eje x de $180°$ y una transformación de efector final como una rotación en y de $-90°$ para utilizarlos en la derivación del jacobiano de imagen.

La implementación del algoritmo IBVS descrito en (39) resultó no exitosa. Esto se puede apreciar a partir del segundo 49 del vídeo, hasta el tiempo 1:05. Se observa como al mostrar el marcador el brazo robot se satura en uno de sus límites horizontales y se limita a movimientos leves en el eje vertical dando la impresión que el brazo está huyendo del marcador.

Esto se puede explicar al observar (39) y entender las operaciones que suceden al aplicarlo. Desarrollando (39) se llega a:

$$\begin{bmatrix} q_{1new} \\ q_{2new} \end{bmatrix} = \begin{bmatrix} q_{1old} \\ q_{2old} \end{bmatrix} + \begin{bmatrix} J_{L1}^{\dagger} & J_{L2}^{\dagger} \\ J_{L3}^{\dagger} & J_{L4}^{\dagger} \end{bmatrix} \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \begin{bmatrix} e_{s1} \\ e_{s2} \end{bmatrix},$$
$$= \begin{bmatrix} q_{1old} \\ q_{2old} \end{bmatrix} + \begin{bmatrix} J_{L1}^{\dagger} K_1 e_{s1} + J_{L2}^{\dagger} K_2 e_{s2} \\ J_{L3}^{\dagger} K_1 e_{s1} + J_{L4}^{\dagger} K_2 e_{s2} \end{bmatrix},$$

en donde claramente al concluir la operación, el valor del ángulo de movimiento horizontal depende del valor de la constante de **K** establecida para el movimiento vertical, y viceversa. Esto representa un problema debido la diferencia de orden de magnitud entre ambas constantes seleccionadas para el controlador, siendo estas:

$$\mathbf{K} = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 * 10^{10} \end{bmatrix},$$

se asume que esta diferencia de magnitud es la que causa problemas al momento de intentar la implementación adecuada del algoritmo.

Por lo tanto, la propuesta encontrada para obtener un funcionamiento adecuado fue reorganizar el orden de los elementos que conforman el algoritmo IBVS para que los ángulos dependieran únicamente de sus constantes respectivas en la matriz **K**, la modificación propuesta fue:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{K}\mathbf{J}_L^{\dagger}(\mathbf{q}_k, \mathbf{s}[k], {}^{C}z[k])\mathbf{e}_{\mathbf{s}}[k], \tag{40}$$

esta modificación permite desarrollar el algoritmo de la siguiente manera:

$$\begin{bmatrix} q_{1new} \\ q_{2new} \end{bmatrix} = \begin{bmatrix} q_{1old} \\ q_{2old} \end{bmatrix} + \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \begin{bmatrix} J_{L1}^{\dagger} & J_{L2}^{\dagger} \\ J_{L3}^{\dagger} & J_{L4}^{\dagger} \end{bmatrix} \begin{bmatrix} e_{s1} \\ e_{s2} \end{bmatrix},$$
$$= \begin{bmatrix} q_{1old} \\ q_{2old} \end{bmatrix} + \begin{bmatrix} J_{L1}^{\dagger} K_1 e_{s1} + J_{L2}^{\dagger} K_1 e_{s2} \\ J_{L3}^{\dagger} K_2 e_{s1} + J_{L4}^{\dagger} K_2 e_{s2} \end{bmatrix},$$

esta modificación permite que el valor de la matriz **K** correspondiente al primer ángulo afecte únicamente a ese ángulo, y viceversa.

Los resultados del algoritmo al aplicar esta modificación se pueden apreciar en el vídeo a partir del tiempo 1:06 hasta el final. Se puede observar como se presenta un movimiento suave en todo momento sin importar el tipo de coordenada que esté recibiendo, ya sea únicamente en un eje, o combinada, lo cual lo hace superior a la combinación de controladores PD. Se nota al final un error, este se atribuye a la modificación realizada para el funcionamiento. A pesar que funcionó bien en un rango, no se tiene certeza que el error presente en lazo cerrado una dinámica LTI.

La razón de ser del orden propuesto en el algoritmo de IBVS es para que el error presente en lazo cerrado la dinámica LTI, por lo tanto, al aplicar (39), se obtiene lo siguiente:

$$\begin{aligned}\dot{\mathbf{e}}_\mathbf{s} &= -\mathbf{J}_L\mathbf{J}_L^\dagger\mathbf{K}\mathbf{e}_\mathbf{s},\\ &= -\mathbf{K}\mathbf{e}_\mathbf{s}.\end{aligned}$$

Al momento de realizar un cambio de orden, como el planteado sucede los siguiente:

$$\dot{\mathbf{e}}_\mathbf{s} = -\mathbf{J}_L\mathbf{K}\mathbf{J}_L^\dagger\mathbf{e}_\mathbf{s},$$

como se puede observar, en este caso no se cancela la dinámica del sistema, por lo que no se garantiza que se tenga un sistema globalmente asintóticamente estable.

Se sospecha, por la pruebas realizadas que el sistema es asintóticamente estable, pero confirmarlo, se debería plantear una función de Lyapunov para determinar la estabilidad y profundizar en ello con el principio de invariancia de LaSalle. Sin embargo, esto es tema que se sale del alcance de este estudio.

CAPÍTULO 11

Conclusiones

1. Entre los módulos comparados, JeVois Smart Vision Camera y ESP32-CAM, el desempeño de los algoritmos aplicados llegó a ser similar en varias instancias, sin embargo los factores que hicieron que se seleccionara el primero sobre el segundo fueron la velocidad de procesamiento de vídeo en tiempo real y la capacidad de detectar marcadores de referencia con los componentes que se tenía disponibles al momento de realizar el estudio.

2. El módulo de visión JeVois Smart Vision Camera presenta versatilidad considerable al momento de su aplicación, tanto para la programación como para la integración en sistemas más complejos. Esta característica se considera importante pues demuestra amigabilidad con el usuario, lo que permite trabajo más fluido.

3. El módulo de visión ESP32-CAM se considera capaz para su aplicación en proyectos sencillos. El desempeño que este mostró al momento de la comparación con el otro módulo resultó ser ilustrativa de su capacidad general y posible consideración en proyectos de menor complejidad, utilizando las herramientas adecuadas.

4. La integración de JeVois Smart Vision Camera con un Arduino Micro Pro es una combinación aceptable para la realización de una aplicación de control basado en visión. La velocidad de actualización del microcontrolador resultó ser satisfactoria para recibir la imagen en tiempo real y transmitir la posición deseada a los motores del brazo robot construido.

CAPÍTULO 12

Recomendaciones

1. Se recomienda ampliar a este estudio comparativo considerando módulos que presenten características similares a JeVois. Dada la situación de escasez de chips en 2021 no se pudo realizar un estudio más equitativo, pero esto abrió la posibilidad de considerar módulos de bajo costo con capacidad moderada. Ampliar en este estudio se considera una actividad interesante para incrementar el repertorio de posibilidades con las que cuente la Universidad para proyectos futuros, dependiendo del presupuesto propuesto.

2. La iluminación es uno de los factores más importantes al momento de realizar estudios de esta índole, por lo que se recomienda que para estudios futuros se procure trabajar en zonas bien iluminadas para obtener resultados satisfactorios.

3. Se recomienda dar oportunidad al módulo de visión ESP32-CAM en proyectos futuros. Las pruebas realizadas en este estudio demostraron una capacidad aceptable únicamente con el módulo base. Si se quisiera implementar en proyectos más complejos se recomienda buscar el ESP32-CAM MB que cuenta con una placa de adicional al módulo base que facilita la implementación del mismo al permitir conexión directa por medio de un cable micro USB.

CAPÍTULO 13

Bibliografía

[1] JeVois, *JeVois Smart Machine Vision*, 2017. dirección: https://www.jevoisinc.com/ (visitado 21-04-2021).

[2] D. Nair, A. Pakdaman y P. G. Plöger, "Performance Evaluation of Low-Cost Machine Vision Cameras for Image-Based Grasp Verification," *arXiv preprint arXiv:2003.10167*, 2020.

[3] DFRobot, "ESP32-CAM Development Board," *DFRobot*, págs. 1-5, 2019. dirección: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf.

[4] OLIMEX Ltd., "Using ESP32-CAM with Arduino IDE Tutorial," 2020. dirección: https://www.olimex.com/Products/IoT/ESP32/ESP32-CAM/resources/Using-ESP32-CAM-with-Arduino-IDE.pdf.

[5] OpenMV, "OpenMV-H7 A Python programmable machine vision camera," inf. téc., 2018. dirección: https://openmv.io.

[6] ——, *OpenMV Cam H7*, 2020. dirección: https://openmv.io/collections/products/products/openmv-cam-h7 (visitado 21-04-2021).

[7] PixyCam, *Pixy Documentation*, 2020. dirección: https://docs.pixycam.com/wiki/doku.php?id=wiki:v1:overview (visitado 21-04-2021).

[8] ArduCam, *Arduino Camera: SPI Camera Module from Arducam - Arducam*, 2019. dirección: https://www.arducam.com/spi-arduino-camera/ (visitado 21-04-2021).

[9] OmniVision, "Advanced Information Preliminary Datasheet OV7670/OV7171 CMOS VGA (640x480) CAMERACHIP TM Sensor O mni ision® with OmniPixel ® Technology General Description," inf. téc., 2006.

[10] Raspberry Pi Foundation, "Raspberry Pi High Quality Camera," inf. téc., 2020. dirección: www.raspberrypi.org.

[11] Texas Instruments Incorporated, "TDA3x SoC for Advanced Driver Assistance Systems (ADAS)," inf. téc., 2016. dirección: www.ti.com.

[12] D. J. Agravante, G. Claudio, F. Spindler y F. Chaumette, "Visual Servoing in an Optimization Framework for the Whole-Body Control of Humanoid Robots," *IEEE Robotics and Automation Letters*, vol. 2, n.º 2, págs. 608-615, 2017, ISSN: 23773766. DOI: 10.1109/LRA.2016.2645512.

[13] H. Wu, T. T. Andersen, N. A. Andersen y O. Ravn, "These Three Hardware Components Is Required , Which Will Be," págs. 457-462, 2017. dirección: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7942738.

[14] G. Bradski y A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st. O'Reilly Media, 2008.

[15] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[16] J. Howse y J. Minichino, *Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning, 3rd Edition*. Packt Publishing Ltd, 2020.

[17] M. Fiala, "Designing highly reliable fiducial markers," *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 32, n.º 7, págs. 1317-1324, 2009.

[18] D. B. dos Santos Cesar, C. Gaudig, M. Fritsche, M. A. dos Reis y F. Kirchner, "An evaluation of artificial fiducial markers in underwater environments," en *OCEANS 2015-Genova*, IEEE, 2015, págs. 1-6.

[19] R. Muñoz-Salinas y S. Garrido-Jurado, *ArUco Library*, 2013.

[20] A. Botta y G. Quaglia, "Performance analysis of Low-cost tracking system for mobile robots," *Machines*, vol. 8, n.º 2, pág. 29, 2020.

[21] P. I. Corke, "Visual Control Of Robot Manipulators – A Review," en *Visual Servoing*, World Scientific, 1994, págs. 1-31.

[22] P. Corke, *Robotics, Vision and Control. Fundamental Algorithms In MATLAB*, 2nd. Springer, 2017.

[23] M. Zea, "Control basado en visión," págs. 1-14, 2020.

[24] JeVois, *JeVois-A33: JeVois Inventor graphical user interface*, 2018.

[25] OBS Project, *OBS - Open Broadcaster Software*, 2012. dirección: https://obsproject.com/.

[26] Arduino, *Arduino Software (IDE)*, 2020. dirección: https://www.arduino.cc/en/software.

[27] P. Minatel, *Arduino-ESP32*, 2017. dirección: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json.

[28] Autodesk, *Autodesk Inventor*, 2021. dirección: https://www.autodesk.com/products/inventor/overview?term=1-YEAR&tab=subscription.

CAPÍTULO 14

Anexos

14.1. Comparación de desempeño de algoritmos

Este enlace lleva a un vídeo en el que se presenta una comparación directa, en vídeo, del desempeño de los módulos comparados en este estudio: https://youtu.be/WG2dupT1ajU

14.2. Control basado en visión

Este enlace lleva a un vídeo en el que se presenta el funcionamiento de un brazo robot de dos grados de libertad controlado por visión de computadora: https://youtu.be/eOT65mZi808

CAPÍTULO 15

Glosario

OpenCV: Librería libre de visión por computadora. Su nombre completo en inglés es *Open Computer Vision*, traduciendo, Visión por Computadora Abierta.. 3, 9, 28, 39, 43

PD: Controlador proporcional derivativo. Es un tipo de controlador en un sistema de control cuya salida varía proporcionalmente al error de la señal, así como con la derivada de la señal de error.. 58

Trade study: Estudio comparativo realizado para encontrar la opción más viable de una serie de candidatos. Se asigna una calificación y peso a los parámetros de estudio, luego se calcula una calificación pesada. Finalmente, se selecciona la opción que presente mayor calificación al estudio.. 41