Sistema de control para rastreo facial

Isaac Arcia, Ricardo Sánchez
Área de Conocimiento de Tecnología de la Información y Comunicación, Universidad Nacional de Ingeniería
Managua, Nicaragua
isaac.arcia52u@std.uni.edu.ni
ricardo.sanchez52u@std.uni.edu.ni

***Abstract*— El presente documento es un informe acerca de nuestro proyecto final para el curso de Sistemas de Control. El proyecto consiste en un sistema de control de rastreo facial a través de procesamiento de video para la medición del error y servomotores para su corrección.**

I. INTRODUCCIÓN

El avance en la integración de tecnologías de visión por computadora y sistemas de control ha permitido el desarrollo de soluciones innovadoras en diversos campos, desde la automatización hasta la interacción humano-máquina. En este contexto, el presente proyecto propone un sistema de seguimiento facial, cuyo objetivo principal es demostrar la capacidad de integrar herramientas modernas como el procesamiento de video y control servomecánico para rastrear rostros en tiempo real.

El sistema se compone de una cámara montada en un dispositivo móvil, dos servomotores de 90 grados colocados en forma de azimut y elevación y una Raspberry Pi Pico para el control de los mismos. A través de procesamiento de video con OpenCV, se detectan rostros en el marco de la cámara, calculando el error en píxeles con respecto al centro de la imagen. Este error se traduce en ángulos de corrección que se envían a la Raspberry Pi Pico para que luego controle a los servomotores y así ajustar la orientación del dispositivo móvil.

II. JUSTIFICACIÓN

La justificación de este proyecto radica en la aplicación práctica de conceptos de control y visión artificial en el contexto del seguimiento facial, lo cual tiene múltiples implicaciones en áreas como la robótica y la seguridad. El sistema propuesto emplea una cámara de teléfono móvil junto con servomotores, lo que permite un diseño compacto y de bajo costo en comparación con soluciones profesionales.

El uso de servomotores y procesamiento de video en tiempo real para mantener la cara del objetivo centrada en el campo de visión de la cámara no solo permite demostrar conceptos técnicos, sino que también introduce oportunidades para el desarrollo de sistemas más complejos.

III. OBJETIVOS

A. General

- Desarrollar un sistema de seguimiento facial que integre visión por computadora y control servomecánico, capaz de rastrear y mantener rostros en el centro del marco de una cámara en tiempo real.

B. Específicos

- Analizar el funcionamiento de los servomotores MG995 y las condiciones mecánicas necesarias para sostener y manipular un dispositivo móvil como cámara principal en el sistema.
- Demostrar la efectividad del controlador al traducir el error en píxeles a ángulos de corrección y operar los servomotores mediante la Raspberry Pi Pico para realizar el seguimiento facial.
- Establecer el proceso de detección de rostros mediante OpenCV y explicar cómo se calcula y traduce el error en píxeles al centro de la imagen para generar comandos de corrección angular.

IV. ÓPTICA

Antes de hablar del sistema elaborado, primero debemos entender un poco de óptica y el funcionamiento de una cámara.

A. Punto y Distancia Focal

El punto focal es una característica clave en el funcionamiento de los lentes de una cámara. Se define como el punto donde los rayos de luz paralelos al eje óptico del lente convergen después de ser refractados. La distancia entre este punto y el centro óptico del lente se denomina distancia focal, y es uno de los parámetros más importantes para entender cómo un lente proyecta imágenes en el sensor de la cámara.

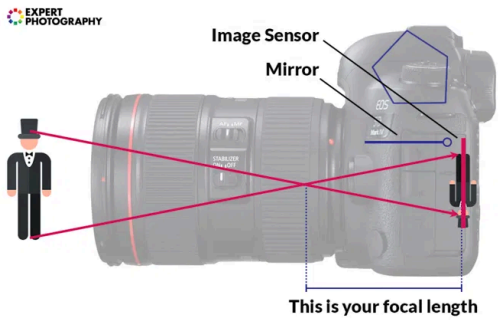

Fig. 1 Estructura simplificada de una cámara, mostrando el punto y distancia focal

De acuerdo a [2], la distancia focal determina el campo de visión (o FOV) de la cámara y, en consecuencia, la escala de los objetos proyectados en el sensor. Un lente con una distancia focal corta (gran angular) tiene un FOV amplio y proyecta los objetos más pequeños en el sensor, lo que permite incluir más elementos de la escena en el encuadre. En cambio, un lente con una distancia focal larga (teleobjetivo) tiene un FOV estrecho, proyectando los objetos más grandes en el sensor y creando la ilusión de estar más cerca (zoom).

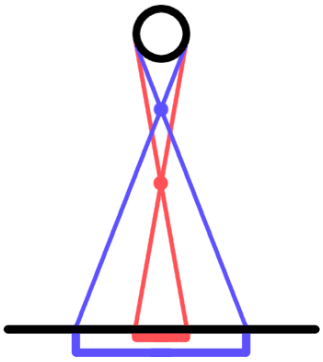

Fig. 2 Comparación del tamaño de la proyección de un objeto según la distancia focal.

En el cálculo del error angular utilizado en nuestro sistema esta relación es crucial, ya que la escala de los objetos es también la escala del error en distancia. Por tanto, tener en cuenta la distancia focal en nuestros cálculos es esencial para un correcto funcionamiento del sistema.

B. Sensor

El sensor de una cámara es el encargado de reaccionar a los rayos de luz provenientes del lente. Estos sensores son componentes extremadamente complejos, sin embargo, para nuestro proyecto nos interesa sólamente saber qué rol tienen sus dimensiones físicas.

Cuando se observa una imagen a través de una cámara digital, esta está compuesta por **píxeles**, los cuales podrían considerarse la unidad digital de distancia en una imagen. Para conseguir la magnitud real de una distancia en píxeles se debe efectuar una regla de tres usando las dimensiones digitales y reales del diámetro total de la cámara.

$$d_{\text{real}} = \left(\frac{d_{\text{pixeles}}}{d_{\text{pixeles\_total}}} \right) \times d_{\text{real\_total}}$$

C. Especificaciones del Sistema

TABLA I
Especificaciones de la cámara utilizada (teléfono móvil Redmi Note 12S)

FOV	60°
Diámetro del sensor	6.35 mm
Distancia focal	7.1 mm

V. SISTEMA DE CONTROL

A. Diagrama de Comunicación

La comunicación en nuestro sistema comienza por el dispositivo móvil cuando este le envía un *stream* de video a través del *Android Debug Bridge* (ADB) a una computadora que corre el programa escrito en Python. La computadora, luego de calcular el error, procede a enviárselo a una Raspberry Pi Pico (RPP) como un json a través de un puerto serial. Como último paso, la RPP se encarga de mover los servomotores según la información recibida

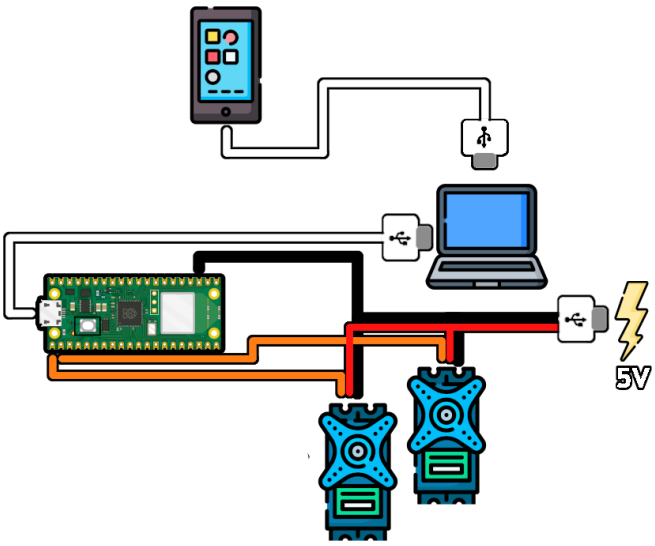

Fig. 3 Diagrama de comunicación

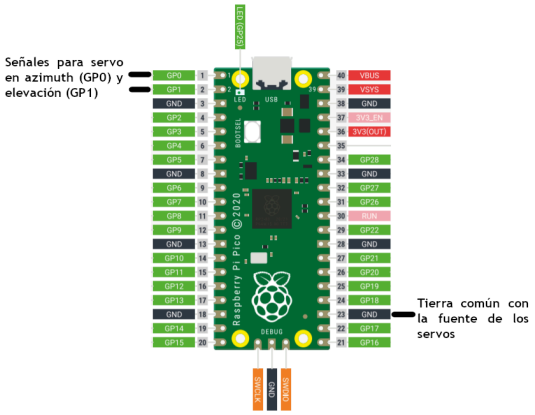

Fig. 4 Conexiones a la Raspberry Pi Pico

B. Diagrama de Bloques

Nuestro sistema es uno de lazo cerrado con retroalimentación positiva.

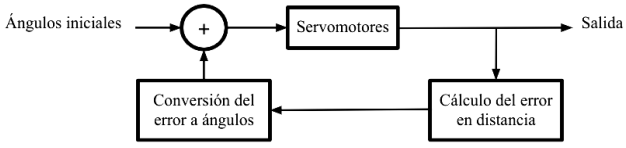

Fig. 5 Diagrama de bloques

B. Conversión del Error

El cálculo del error en distancia física (mm) es trivial usando la fórmula mostrada anteriormente, la verdadera complejidad del cálculo recae en su conversión a error angular.

A grandes rasgos, lo que se debe hacer es calcular el v**ector central** y el **vector objetivo** para poder obtener los ángulos entre ellos.

El vector central se define como el vector normalizado que va desde el centro del sensor de la cámara hasta el punto focal, es decir, el vector que va directamente hacia el frente de la cámara. El vector objetivo es, en cambio, el vector que va desde el centro del

sensor de la cámara hasta el centro del objetivo.

Para empezar, debemos pensar que nuestro sistema está en un espacio tridimensional. Se utilizó la convención mostrada en la siguiente figura.

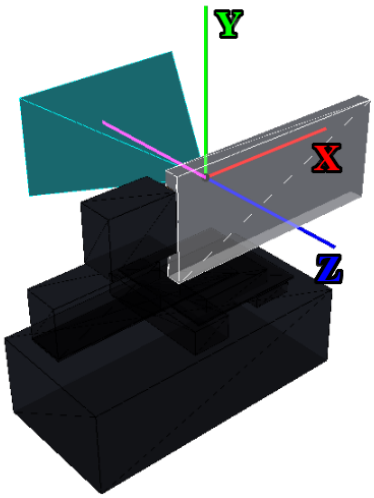

Fig. 6 Convención de los ejes

Para calcular el vector central se toma en cuenta que cuando los ángulos de los servomotores se encuentran en 0°, el vector central es [0 0 -1]. Entonces se emplean las siguientes matrices de rotación (asumiendo que α es azimuth y β es elevación), tomando el vector central como $\mathbf{v_c}$:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

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

$$\mathbf{v_c} = R_y(\beta) \cdot R_x(\alpha) \cdot \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

El próximo paso es obtener una base vectorial que genere el subespacio comprendido por el sensor de nuestra cámara con las rotaciones de los servomotores. para esto decimos que el vector central es normal al plano del sensor, y a partir de ahí se efectúa el siguiente producto vectorial normalizado para generar el componente $\hat{x}_s$ el cual debe ser paralelo al plano XZ.

$$\hat{x}_s = \frac{\mathbf{v_c} \times [0, 1, 0]}{|\mathbf{v_c} \times [0, 1, 0]|}$$

Ya teniendo este vector, se repite el proceso una vez más para encontrar el componente $\hat{y}_s$:

$$\hat{y}_s = \frac{\mathbf{v_c} \times \hat{x}_s}{|\mathbf{v_c} \times \hat{x}_s|}$$

Con nuestra base vectorial calculada podemos escalar los componentes $\hat{x}_s$ y $\hat{y}_s$ por el error en x y el error en y respectivamente para así obtener una representación vectorial tridimensional del error. Entonces nuestro vector objetivo (denotado como $\mathbf{v_o}$) es obtenido con la siguiente fórmula:

$$\mathbf{v_o} = \mathbf{v_c} + \text{error}_x \cdot \hat{x}_s + \text{error}_y \cdot \hat{y}_s$$

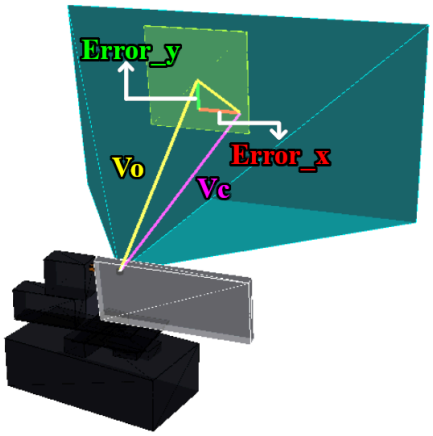

Fig. 7 Representación tridimensional de los vectores

Como último paso, calculamos los errores en azimuth y elevación usando las siguientes fórmulas:

$$\text{error}_{azimuth} = -\cos^{-1}\left(\frac{v_{c_x}v_{o_x} + v_{c_z}v_{o_z}}{\sqrt{v_{c_x}^2 + v_{c_z}^2}\sqrt{v_{o_x}^2 + v_{o_z}^2}}\right)$$

$$\text{error}_{elevation} = \cos^{-1}\left(\frac{\sqrt{v_{c_x}^2 + v_{c_z}^2}v_{c_y} + \sqrt{v_{o_x}^2 + v_{o_z}^2}v_{o_y}}{\sqrt{(v_{c_x}^2 + v_{c_z}^2 + v_{c_y}^2)}\sqrt{(v_{o_x}^2 + v_{o_z}^2 + v_{o_y}^2)}}\right)$$

Estos son los errores que serán enviados en un json a través del puerto serial a la Raspberry Pi Pico.

VI. HARDWARE

El sistema está montado sobre una base fabricada con una caja de cartón, sobre la cual se encuentran los componentes principales. Se utilizan dos servomotores MG995: el primero, encargado del azimut, está fijado a la base y orientado hacia arriba, permitiendo el movimiento de una plataforma en la que se encuentra el segundo servomotor, también MG995, responsable de la elevación. El mástil de este último está adherido a un soporte diseñado para estabilizar el dispositivo móvil.

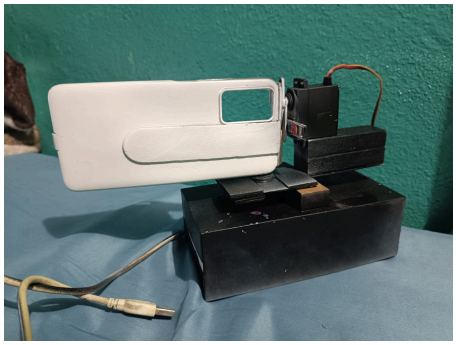

Fig. 8 Foto del sistema

Este sistema requiere de tres conexiones USB para funcionar:

- Dispositivo Android → PC (Comunicación por ADB)
- Raspberry Pi Pico → PC (Comunicación serial)
- Servomotores → Adaptador de corriente (Fuente)

VII. SOFTWARE

Además de la implementación del cálculo del error, en la parte de software se tiene el procesamiento de la imagen, detección facial y una aplicación de escritorio. Todo esto puede ser encontrado en nuestro repositorio de github en [3].

A. Detección Facial

Todo el software fue escrito en Python, donde se empleó la librería OpenCV para la detección facial. OpenCV utiliza un método de detección facial llamado cascada multi-escala basado en el algoritmo de Haar Cascade descrito en [4], que consiste en entrenar una serie de clasificadores en diferentes escalas de la imagen para identificar las características faciales. Este método funciona analizando la imagen en varias resoluciones, permitiendo detectar rostros en diferentes tamaños, lo que mejora la precisión y eficiencia de la detección. La cascada se utiliza para identificar patrones característicos del rostro humano en distintas etapas, desde la forma general hasta los detalles más específicos, como los ojos, la nariz y la boca.

La detección de la cámara se lleva a cabo en su propia thread para no frenar el procesamiento del stream de video proveniente del dispositivo android.

A. Aplicación de Escritorio

Se desarrolló una aplicación de escritorio a forma de interfaz para que el usuario pueda ver fácilmente información en tiempo real sobre el sistema. Para ello se utilizó el framework Qt con bindings en Python mediante el módulo PyQt. También se utilizó el módulo PyQtGraph para mostrar al usuario un gráfico en 3D en tiempo del sistema.

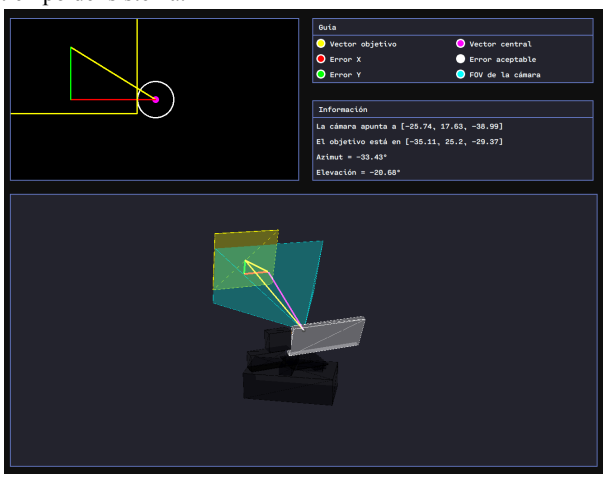

Fig. 9 Captura de la aplicación de escritorio

VIII. CAPACIDADES DEL SISTEMA

Nuestro sistema tiene las siguientes capacidades acorde a sus componentes:

- FOV: 60° horizontalmente y 33.75° verticalmente
- Tiempo de respuesta (transferencia y procesamiento de video + cálculo del error): ~370 ms
- Servomotores: 90° de giro a 450°/s

A partir de esto, podemos calcular las siguientes características:

- FOV total:

$$FOV_{\text{total}} = FOV_{\text{servo}} + FOV_{\text{cámara}}$$
$$\Rightarrow \mathbf{FOV}_{\text{total}} = \mathbf{150}^\circ$$

- Velocidad máxima del objetivo:

$$v_{\text{máx}} = \frac{FOV_{\text{cámara}}}{t_{\text{respuesta}}}$$
$$\Rightarrow \mathbf{v}_{\text{máx}} \approx \mathbf{162.16}^\circ/\text{s}$$

- Frecuencia máxima de seguimiento:

$$f_{\text{máx}} = \frac{1}{t_{\text{respuesta}}}$$
$$\Rightarrow \mathbf{f}_{\text{máx}} \approx \mathbf{2.7}\ \text{Hz}$$

En cuanto el diseño digital, se considera cualquier error en distancia igual o menor a 40 píxeles (equivalente a ~0.4mm en el sensor de la cámara) como aceptable por lo que no requiere corrección.

Por otro lado, la detección facial por cascada multi-escala de OpenCV ofrece múltiples parámetros ajustables. Obtuvimos un buen balance entre distancia máxima de detección y cantidad de falsos negativos con:

- scaleFactor = 1.3
- minNeighbors = 6
- minSize = (30, 30)

Estos parámetros nos permiten filtrar la mayoría de falsos negativos y aún así detectar caras a unos 2 metros.

IX. OBSERVACIONES

Originalmente, nuestro proyecto apuntaba a proveer reconocimiento facial y una interfaz más completa para la selección del objetivo en caso de detectar muchos rostros. Esto, a pesar de ser posible, hubiese impactado significativamente el tiempo de respuesta del proyecto. No obstante, de tener un poder computacional mayor, se podría agregar esa utilidad al sistema.

X. CONCLUSIONES

El proyecto alcanzó su objetivo general de integrar visión por computadora y control servomecánico para desarrollar un sistema de seguimiento facial capaz de mantener rostros centrados en el marco de la cámara en tiempo real. Este logro refleja el potencial de combinar tecnologías accesibles con algoritmos eficientes para aplicaciones prácticas en control y automatización.

En relación con los objetivos específicos, se identificaron y analizaron las características técnicas de los servomotores MG995, lo que permitió diseñar una estructura mecánica funcional y estable para soportar y manipular un dispositivo móvil como cámara principal. Además, se demostró la efectividad del controlador desarrollado al traducir el error en píxeles a correcciones angulares precisas, operando de manera confiable a través de una Raspberry Pi Pico.

Finalmente, se estableció un flujo claro para la detección de

rostros utilizando OpenCV y se explicó el cálculo del error en píxeles, mostrando cómo este se traduce en comandos angulares que garantizan un seguimiento preciso. Estas conclusiones destacan la capacidad del sistema para aplicar conceptos de visión por computadora y control de manera efectiva en entornos reales, superando desafíos como la latencia y la sensibilidad a las condiciones de iluminación.

REFERENCIAS

[1] *Great Big Photography World, (2023)* Understanding Focal Length and Angle of View. [Online]
Available: https://greatbigphotographyworld.com.

[2] *GSMArena, (2023)* Xiaomi Redmi Note 12S - Full phone specifications. [Online]
Available:
https://www.gsmarena.com/xiaomi_redmi_note_12s-12218.php

[3] *ikz87, (2024)* phone-facial-tracking - Final project for Control Systems course. [Online]
Available: https://github.com/ikz87/phone-facial-tracking

[4] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, pp. I-511-I-518, doi: 10.1109/CVPR.2001.990517.