ISBN: 978-628-96613-1-6. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2025.1.1.521

Rubik's Cube Solver Robot with previous scanning using the HSV Model

Dávalos-Carpio, Juan-Pablo, Br.[1]; Mendoza-López, Mauricio, Br.[2] ; Portugal-Cuno, Luis, Br.[3] Quispe Ccachuco, Marcelo, Dr[4]; Siles-Nates, Fernando, Dr. [5]
[1,2,3,4,5]Universidad Católica de Santa María, Perú, *juan.davalos@ucsm.edu.pe, luis.portugal@ucsm.edu.pe , mauricio.mendoza@ucsm.edu.pe, mquispec@ucsm.edu.pe, fsiles@ucsm.edu.pe*

Abstract– ***The purpose of this work is to create a robot capable of solving the Rubik's Cube. This involves three important stages. The first stage involves detecting the state of the cube, that is, determining how it is mixing by knowing the colors on each face. This involves using artificial vision through a camera and a computer that processes the images. Artificial vision programming is based on the HSV model, where the parameters for detecting colors are: hue, saturation, and value. The second stage consists of obtaining the movements required to solve the cube using the Kociemba algorithm. In the third stage, the movements obtained in the second stage are executed, where the ATmega2560 microcontroller is responsible for rotating each face of the cube using stepper motors. The robot was designed with a configuration that allows it to rotate all six faces of the cube and was built with 3D-printed parts and acrylic sheets. For machine vision, a box was built to scan the cube so that outside lighting wouldn't affect color recognition. The robot finally solved the cube in an average time of six seconds.***

Keywords-- ***Artificial vision, algorithm, colors, cube, programming.***

Robot Solucionador De Cubo Rubik Con Previo Escaneo Mediante El Modelo HSV

Dávalos-Carpio, Juan-Pablo, Br.[1]; Mendoza-López, Mauricio, Br.[2] ; Portugal-Cuno, Luis, Br.[3] Quispe Ccachuco, Marcelo, Dr[4]; Siles-Nates, Fernando, Dr. [5]
[1,2,3,4,5]Universidad Católica de Santa María, Perú, *juan.davalos@ucsm.edu.pe, luis.portugal@ucsm.edu.pe , mauricio.mendoza@ucsm.edu.pe, mquispec@ucsm.edu.pe, fsiles@ucsm.edu.pe*

Resumen– El propósito de este trabajo es realizar un robot capaz de solucionar el cubo Rubik. Para ello se tienen tres etapas importantes. La primera etapa se trata de la detección del estado del cubo, es decir, saber cómo está mezclado sabiendo los colores que hay en cada cara. Para ello es utilizada la visión artificial por medio de una cámara y un computador que procesa las imágenes. La programación de la visión artificial se basa en el modelo HSV, en donde los parámetros para detectar los colores son: tono, saturación y brillo. La segunda etapa consta de la obtención de los movimientos que se tienen que realizar para resolver el cubo utilizando el algoritmo Kociemba. En la tercera etapa se ejercen los movimientos obtenidos en la segunda etapa, en donde el microcontrolador ATmega2560 se encarga de girar cada cara del cubo a través de motores paso a paso. Se diseñó el robot con una configuración que permite girar las seis caras del cubo y fue construido con piezas impresas en 3D y planchas de acrílico. Para la visión artificial, se fabricó una caja en donde se realiza el escaneo del cubo con el propósito de que la iluminación exterior no afecte el reconocimiento de los colores. Finalmente, el robot logra resolver el cubo con un promedio de tiempo de seis segundos.

Palabras clave—algoritmo, colores, cubo, programación, visión artificial.

I. Introducción

El cubo de Rubik es uno de los pocos rompecabezas que se ha mantenido vigente a lo largo del tiempo. Aunque existen muchas variantes como diferentes formas (triangulares, mixtas, de múltiples dimensiones, entre otras), el formato clásico de seis caras con una cuadrícula de 3x3 por lado sigue siendo el más conocido y popular.

Para armar el cubo Rubik se utilizan algoritmos que uno mismo va aprendiendo mientras practica y completa el cubo. Estos mismos algoritmos pueden ser programados en una máquina, permitiéndole resolver el cubo al indicarle específicamente qué secuencias de movimientos debe ejecutar.

La tecnología de la visión artificial permite a los robots identificar, seguir y manipular objetos de manera autónoma, lo que resulta crucial en entornos dinámicos y cambiantes. La capacidad de discernir entre objetos según sus características, como forma, color y tamaño, añade una capa adicional de inteligencia a la operación del robot [1].

Actualmente existen todo tipo de robots que tienen la capacidad de realizar movimientos complejos y ejecutar procesos de forma automatizada según las instrucciones que reciben. Entre estos, hay robots que son fabricados para solucionar el cubo Rubik, reconociendo el estado inicial del cubo (se identifica los colores de cada cara mediante distintos métodos) y a partir de esa información se determinan algoritmos que pueden ser aplicados para resolverlo. Todo el proceso culmina cuando cada cara muestra un color uniforme, de esa forma se evidencia que el cubo se ha solucionado correctamente.

A. Antecedentes

El desarrollo de sistemas automatizados capaces de resolver el cubo Rubik ha captado el interés de la comunidad científica y educativa. Empresas como MoYu han diseñado robots comerciales que combinan algoritmos de resolución optimizados con control preciso de motores, permitiendo la solución del cubo en pocos segundos. Además, como el robot de Mitsubishi desarrollaron sistemas automatizados de alta precisión, como brazos robóticos de seis grados de libertad, capaces de resolver el cubo Rubik en tiempos récord. Estos proyectos demuestran la capacidad de los robots industriales no solo para tareas repetitivas, sino también para operaciones complejas que requieren coordinación, visión artificial y procesamiento en tiempo real.

B. Algoritmo de los Cubos Rubik

El cubo de Rubik, además de ser un rompecabezas ampliamente conocido, ha servido como base para el desarrollo de algoritmos aplicados en distintos campos de la ingeniería. Inicialmente, se propusieron diversos métodos para resolver el cubo de forma secuencial, lo cual sentó las bases para algoritmos sistemáticos y programables. Con el avance tecnológico, estas ideas evolucionaron hacia aplicaciones más complejas, como el procesamiento y codificación de imágenes. Tomando como ejemplo, Renusree Varma Mudduluri et al. desarrollaron un algoritmo de cifrado de imágenes inspirado en la estructura del cubo de Rubik. En su propuesta, utilizan dos vectores generados aleatoriamente para reordenar los píxeles de la imagen, emulando los movimientos de rotación del cubo [2]. Este enfoque introduce una capa de seguridad al dificultar la recuperación del contenido original sin la clave adecuada. De manera complementaria, Adrian-Viorel Diaconu et al. propusieron un sistema de encriptación digital caótico por bloques, que también se basa en el principio del cubo de Rubik [3]. Su método inicia codificando la imagen mediante desplazamientos alternos de filas y columnas, similares a los giros del cubo, y posteriormente aplica un operador XOR sobre la imagen resultante, empleando claves irregulares. Este

algoritmo combina la estructura del cubo con teorías de caos para robustecer la encriptación.

C. Visión Artificial mediante el uso del modelo de color HSV

De acuerdo con la capacidad de resolución visual humana, los tres componentes del espacio de color HSV se cuantifican respectivamente, y los niveles de cuantificación de los componentes H, S y V son 8, 3 y 3, respectivamente [4]. Tal como es usado en el área del reconocimiento de objetos móviles, fijos, etc. Como tal es explicado en el mismo identificador de gestos en una persona mediante un modelo HSV, se usa similarmente el mismo principio para el cubo de Rubik convirtiendo la imagen RGB a un modelo de color HSV, de esta forma obteniendo información que se puede utilizar a la posterioridad [5].

Aunque para este mismo reconocimiento existen algunos limitantes para su uso o configuración por ejemplo en la cantidad de iluminación que existe por la zona con esto recibiendo valores incorrectos de acuerdo con la visión por computadora [6], existiendo formas de arreglar esta desventaja como son los algoritmos de mejora de la toma de imagen con baja luminosidad [7], de esta forma se puede utilizar para asignar a valores que puedan usarse como un cubo virtual como es mostrado en la siguiente figura, significando por ejemplo la letra "F" de Front, "U" para Up, "L" de Left para cada cara y creando así el cubo antes dicho todo esto explicado se puede visualizar en la figura 1 que esta mostrado a continuación para poder entender de una manera sin demasiadas complicaciones.

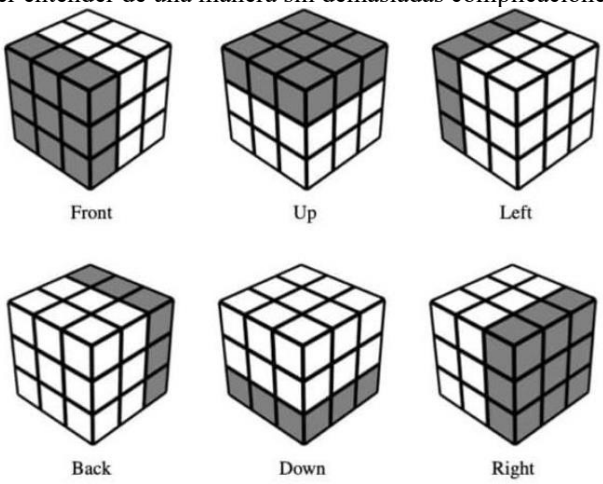

Figura 1. Vistas de las caras del cubo de Rubik

D. Lenguajes de programación

En la época que estamos se utilizan códigos para poder controlar secuencia de funciones mediante estas líneas de programación el cual mediante lenguajes digitales como es dicho, teniendo software para el mismo fin teniendo ejemplos a los entornos de desarrollo integrado (IDE) como Python, Arduino, C++, etc.

De esta forma obtener procesos que puedan realizar la secuencia de pasos como movimiento de los actuadores (usando Arduino IDE), comprensión de los datos en computadora y obtenidos por la visión artificial (Python) dando estos tipos de ejemplos para que existan desarrollos de algoritmos que permiten solucionar el rompecabezas. Basándose en la teoría de grupos, el algoritmo de Kociemba, permite obtener soluciones subóptimas con un breve tiempo de computación, partiendo de una descripción del estado inicial del cubo [8].

E. Microcontroladores

Cuando se requiera utilizar Arduino como componente principal para el funcionamiento de un proyecto, esta plataforma permite el procesamiento de valores provenientes de sensores u otros componentes que entregan datos numéricos, facilitando así la automatización de procesos [9][10].

Además, existen diversas versiones de microcontroladores compatibles con Arduino que pueden emplearse en distintos proyectos. Un ejemplo es el Arduino Nano, un microcontrolador compacto y de fácil implementación [11], también podemos incluir al Arduino Mega con mejoras características y velocidades a la hora de utilizar en los procesos que se necesitaran, aunque con estos tienen un mayor costo tienen capacidades de emplearlo hasta en los PLCs con su control correctamente validado [12] y también otro de los más conocidos como el Arduino UNO un hardware para crear procesos con bajo costos siendo uno de las placas más básicas para comenzar en el control de los lenguajes de programación [13] siendo así que podemos tener varias opciones a la hora de pensar en usar un microcontrolador para el mismo fin o modificar el uso del mismo para poder tener una mejor flexibilidad [14], sabiendo del concepto básico de los microcontroladores, tenemos a continuación en la tabla 1, la comparación entre estos mismos microcontroladores Arduino que se pudieron haber escogido para el trabajo actual, además de algunos adicionales no mencionados con anterioridad pero que fueran una posibilidad de poder utilizarlos ya que tienen la misma tecnología para que se puedan usar en el proyecto actual. Para el trabajo se ha utilizado el arduino MEGA debido a la cantidad de pines que posee.

TABLA 1
COMPARACIÓN DE LOS MICROCONTROLADORES ARDUINO

Nombre	Microcontrolador	Memoria	Velocidad	Pines I/O
UNO	ATmega328P	32kb	16 MHz	14
MEGA	**ATmega2560**	**256kb**	**16MHz**	**54**
ESP32	ESP32	512KB	240 MHz	34
ESP8266	ESP8266	128KB	80 MHz	17

II. Materiales y Métodos

A. Diseño del Robot

A inicios del proyecto, se consideró el diseño del robot compacto que maximice la eficiencia en términos de espacio y funcionalidad como se puede ver en la figura 2. Dicha arquitectura de este robot abarcaría la sincronización de 6 actuadores y acompañada del uso de 2 cámaras que permitan el escaneo de 3 caras del cubo a la vez. Asimismo, permitiendo la captura de datos más eficientes optimizando los tiempos muertos asociados al reposicionamiento del cubo, logrando así un diseño eficiente y tecnológicamente avanzado.

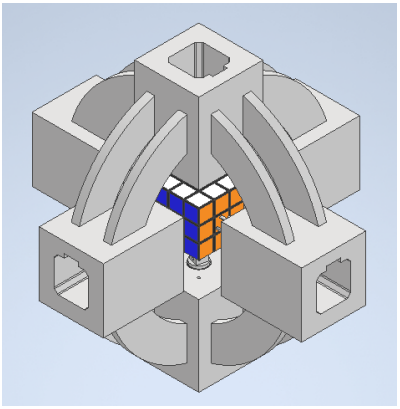

Figura 2. Boceto 1

Si bien este boceto, en comparación con los prototipos desarrollados por empresas de alto renombre como se ve en la figura 10, realizados en base a materiales industriales, presenta una composición optimizada para la tarea propuesta, su complejidad se origina en el procesamiento de colores, debido a su demanda de precisión como la interpretación de las tonalidades capturadas. Esto planteó un desafío en términos de rendimiento computación y la consistencia de la iluminación, ya que se entiende que la iluminación incide mucho en el procesamiento de los colores por lo que se requiere una calibración más avanzada. Siguiendo este contexto se optó por realizar otra alternativa menos compleja y que sea accesible, priorizando calidad – precio como se ve en la figura 3.

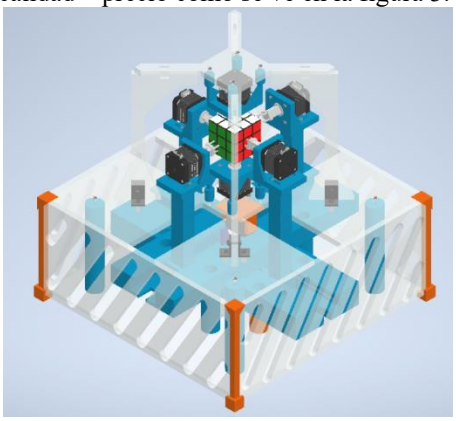

Figura 3. Boceto Final

El concepto del Robot en la Figura 3 fue realizado mediante el software Autodesk Inventor donde se comparó medidas reales como virtuales como ejemplo se tiene los modelos 3D de los motores paso a paso, como también los físicos para poder realizar diseños que no perjudiquen en insertos o ajustes en el momento de su ensamblaje. Este concepto final a su vez presenta el desarrollo de engranajes y correas las cuales fueron calculadas en base a lo que se requería que era desarrollar un mecanismo de autoajuste mediante el mecanismo piñón – cremallera impulsados por un servomotor MGR966R. Ahora comparándolo con el boceto inicial presenta tener un tamaño considerable comparándolo con la Figura 2 teniendo un carácter minimalista y funcional mediante técnicas automatizadas impidiendo la ejecución humana. La estructura presenta planchas de acrílico de 5 y 10 mm de espesor como también abarca piezas complejas diseñadas de cero e impresas por medio de una Impresora 3D utilizando tolerancias con el material PLA.

B. Componentes Físicos

A manera de introducción se tiene presente que el cubo se compone de 6 caras por lo que se hizo uso de 6 actuadores que sean capaces de ejercer el torque necesario para poder mover una de las caras de una manera secuencial.

Debido a la necesidad de realizar movimientos angulares precisos para rotar las diferentes caras del cubo con un ángulo de 90 grados o 180 grados, se propuso la idea de usar los motores paso a paso. Son precisos y pueden manejarse mediante múltiples señales simultáneamente, por lo que se presenta en la Tabla 2 una relación de diferentes modelos de Nema 17 con los diferentes Torques y corrientes que utiliza.

TABLA 2
MODELOS DE MOTORES PASO A PASO

MODELO	CORRIENTE (A)	TORQUE (N-CM)	PESO DE MOTOR (G)
17HS2408	0.6	12	150
17HS3401	1.3	28	220
17HS3410	1.7	28	220
17HS3630	0.4	21	220
17HS4401	**1.7**	**40**	**280**
17HS4402	1.3	40	280
17HS4602	1.2	28	280
17HS8401	.17	52	350
17HS8402	1.3	52	350
17HS8403	2.3	46	350
17HS8630	0.4	34	350
17HS9403	2.3	70	480
17HS6403	2.3	80	500

Se seleccionó el Nema 17HS4401, capaz de realizar trabajos de movimiento sin necesidad de sobrecalentarse ni trabarse. Para manejar este motor paso a paso se hará uso del controlador A4988, este circuito nos permitirá un control uniforme en los motores paso a paso con una precisión y movimientos coordinados para cada una de las caras del cubo. En la figura 4 se puede apreciar los terminales que tiene el motor paso a paso y en la figura 5 se ve el conexionado del arduino MEGA con los controladores A4988 y los motores seleccionados.

Figura 4. Motor Paso a Paso en software Wokwi

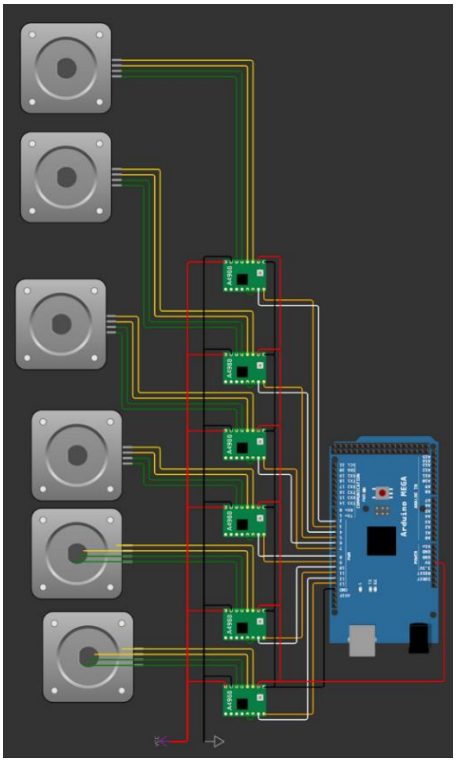

Figura 5. Circuito Electrónico

Para el mecanismo de autoajuste, que se menciona en el diseño del robot, se empleará el servomotor MG996R. De esta forma se moverán las torres en donde están los motores paso a paso.

Figura 6. Servomotor MG966R

C. Diagrama de Flujo

El programa comienza leyendo los datos que obtiene de la cámara para luego crear las variables de cada cara del cubo de Rubik, esto visualizado en la figura 7.

Cuando se tenga iniciado el programa por completo correctamente, podremos tener la opción de presionar una tecla en específico con el mismo nombre de las variables cuales son F, B, R, L, U y D, destacándose que estos son las iniciales de las caras del cubo de Rubik al presionar uno de estos reemplazara en la ventana del cubo virtual una de sus caras por lo que ve la cámara en su momento, realizando este proceso hasta que no se presione ninguna de estos botones y yendo por el "ENTER" lo cual realizara la solución de las caras que tenga actualmente entregando la solución paso a paso que hay que seguir para solucionar el cubo de Rubik posteriormente nos lo mostrara en el cubo virtual para que este también sea mandado al software de Arduino y pueda controlar los actuadores, armando el cubo presente que este en el robot.

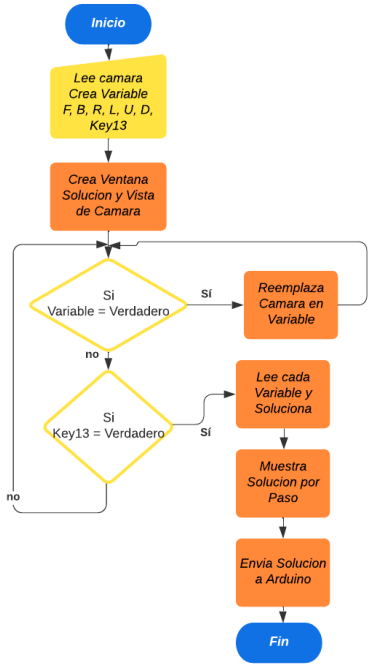

Figura 7. Diagrama de Flujo para el programa en Python

D. Programa del Funcionamiento Completo

En primer lugar, se plantea el código en partes relevantes que se utilizó en el software de Python para que pueda realizar todos los procesos antes ya explicados desde detectar el cubo en las posiciones correctas hasta comunicarse con el Arduino para que pueda mover los actuadores propuestos, donde en primer lugar tenemos el inicio del programa para que pueda utilizar las librerías, la comunicación con el Arduino MEGA y la asignación de variables.

```
import cv2
import numpy as np
import kociemba as Cube
import time
import colorama
import serial
import time
#ASIGNACION DEL PUERTO PARA EL ARDUINO
ardu = serial.Serial("COM3", 9600, timeout=1)
time.sleep(1)
```

Luego de obtener las asignaciones de las variables o librerías que se van a utilizar, se coloca la función para el cual comenzara cada cara del cubo, en una matriz con los colores escritos.

```
state= {
                'up':['white','white','white','white','white','white',
'white','white','white',],
        'right':['white','white','white' ,'white','red','white' ,'white'
,'white','white',],
        'front':['white','white','white' ,'white','green','white' ,'whi
te','white','white',],
        'down':['white','white','white' ,'white','yellow','white' ,'w
hite','white','white',],
        'left':['white','white','white' ,'white','orange','white' ,'whi
te','white','white',],
        'back':['white','white','white' ,'white','blue','white' ,'whit
e','white','white',]
    }
```

Ya que tenemos las posiciones de colores al inicio del cubo virtual en la computadora, además del software, utilizaremos las siguientes líneas para que cada centro con el que se comience tenga las iniciales de la cara.

```
sign_conv={
        'green'  : 'F',
        'white'  : 'U',
        'blue'   : 'B',
        'red'    : 'R',
        'orange' : 'L',
        'yellow' : 'D' }
```

Con las líneas anteriores, se asignará a cada texto el color respectivo en un código de color RGB, para que el programa pueda reconocerlo.

```
color = {
    'red'    : (0,0,255),
    'orange' : (0,165,255),
    'blue'   : (255,0,0),
    'green'  : (0,255,0),
    'white'  : (255,255,255),
    'yellow' : (0,255,255) }
```

Posteriormente se tendrá el código principal donde se encuentra el creador de las ventanas, la lectura de la cámara, las funciones que realiza al presionar un botón en específico para que pueda reemplazar los valores de la cámara al cubo de Rubik virtual.

```
if __name__=='__main__':
  preview = np.zeros((700,800,3), np.uint8)
  while True:
    hsv=[]
    current_state=[]
    ret,img=cap.read()
    # Obtener dimensiones de la imagen capturada
    height, width = img.shape[:2]
```

Ahora tenemos un ajustador de la cámara, solo es modificado y utilizado solo si la cámara esta fuera del centro que debería captar regularmente.

```
# Definir un área central (ajusta estos valores para centrar más
si es necesario)
    left_offset = int(width * 0.25)
    right_offset = int(width * 0.85)
    top_offset = int(height * 0.1)
    bottom_offset = int(height * 0.85)
    # Extraer la región central
        centered_img = img[top_offset:bottom_offset,
left_offset:right_offset]
```

La imagen que ya tenemos centrada y capturada la convertimos en el modelo de color HSV, debido a que, la visión artificial lo está detectando de acuerdo con ese modelo.

```
# Convertir la imagen a espacio de color HSV para detección
de color
    frame = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    mask = np.zeros(frame.shape, dtype=np.uint8)
    # Dibujar los stickers y llenar la vista previa
    draw_stickers(img, stickers, 'main')
    draw_stickers(img, stickers, 'current')
    draw_preview_stickers(preview, stickers)
    fill_stickers(preview, stickers, state)
    texton_preview_stickers(preview, stickers)
```

Con lo demás reconocido comenzamos con los reemplazos en el cubo virtual, cuando comenzamos utilizamos el estado inicial que se dio con anterioridad.

```
for i in range(9):
hsv.append(frame[stickers['main'][i][1]+10][stickers['main'][i][0]+10])
    a = 0
for idx, (x, y) in enumerate(stickers['current']):
color_name = color_detect(hsv[a][0], hsv[a][1], hsv[a][2])
cv2.rectangle(img, (x, y), (x+30, y+30), color[color_name], -1)
        a += 1
current_state.append(color_name)
```

Ahora con los botones y sus funciones por lo que tenemos a la letra “k” el cual significa terminar todo proceso.

```
k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
```

Si la letra es “u” y no es “k” reemplazara cada posición menos la central con lo visto en la cámara.

```
elif k == ord('u'):
        check_state.append('u')
        state['up'][0:4] = current_state[0:4]
        state['up'][5:9] = current_state[5:9]
```

Para el siguiente sigue el mismo proceso solo que la letra es la “r” para cambiar el lado derecho del cubo.

```
    elif k == ord('r'):
        check_state.append('r')
        state['right'][0:4] = current_state[0:4]
        state['right'][5:9] = current_state[5:9]
```

Ahora con la letra “l” para que reemplace el lado izquierdo del cubo.

```
    elif k == ord('l'):
        check_state.append('l')
        state['left'][0:4] = current_state[0:4]
        state['left'][5:9] = current_state[5:9]
```

Ahora con la letra “d” para que reemplace el lado inferior del cubo.

```
    elif k == ord('d'):
        check_state.append('d')
        state['down'][0:4] = current_state[0:4]
        state['down'][5:9] = current_state[5:9]
```

Aun seguimos con la letra “f” para poder reemplazar la parte frontal del cubo.

```
    elif k == ord('f'):
        check_state.append('f')
        state['front'][0:4] = current_state[0:4]
        state['front'][5:9] = current_state[5:9]
```

Finalmente, con la letra “b” para que podamos reemplazar el estado de la parte trasera del cubo virtual.

```
    elif k == ord('b'):
        check_state.append('b')
        state['back'][0:4] = current_state[0:4]
        state['back'][5:9] = current_state[5:9]
```

Con cada reemplaza del estado del cubo realizado, se procede con la resolución del cubo de Rubik además de mostrar cada paso para que pueda ser utilizado por lo que se da las siguientes líneas.

```
    elif k == ord('\r'):
        if len(set(check_state))==6:
            try:
                solved=solve(state)
                if solved:
                    operation=solved.split(' ')
                    send_solution(state)
                    process(operation)
            except:
                print("Error en la detección de lados. Es posible que no hayas seguido la secuencia correctamente o que algún color no se haya detectado bien. Inténtalo de nuevo.")
```

Pero si hubiera algún problema con algún lado de la cara mal escaneada, o no se hubieran escaneado todas las caras al dar el “ENTER” del teclado, nos saldrá un aviso.

```
    else:
            print("No se han escaneado todos los lados. Verifica en la otra ventana cuál falta por escanear.")
            print("Falta por escanear:",6-len(set(check_state)))
        cv2.imshow('Vista Previa',preview)
        cv2.imshow('Imagen de Camara',img[0:600,0:600])
          cv2.destroyAllWindows()
```

Para lo cual también se llegó a utilizar unas funciones específicas para que pueda solucionar de forma más precisa o correcta optando por las siguientes líneas de código, que nos sirve para el uso del modelo de colores HSV.

```
def color_detect(h,s,v):
    # print(h,s,v)
    if h<30 and s>90 and v<160:        return 'red'
    elif s>150 and h<30 and v>160:        return 'orange'
    elif h<50 and h>15 and s>125:        return 'yellow'
    elif h>50 and h<80 and s>70:        return 'green'
    elif h>80 and h<130:        return 'blue'
    elif s<80 and v>140:        return 'white'
    return 'white'
```

Para finalizar tenemos un programa aparte que utiliza la misma lógica que los primeros códigos principales con lo que se empezara con las asignaciones de valores, funciones y librerías.

```
import cv2
import numpy as np
def nothing(x):
    pass
# Crea una ventana
cv2.namedWindow('image')
```

Crearemos unas barras deslizantes para poder modificar los valores de forma más dinámica y fácil cuando quisieras calibrar los colores del modelo de color HSV.

```
# Crea sliders para los valores HSV
cv2.createTrackbar('HMin','image',0,255, nothing)
cv2.createTrackbar('SMin','image',0,255, nothing)
cv2.createTrackbar('VMin','image',0,255, nothing)
cv2.createTrackbar('HMax','image',0,179, nothing)
cv2.createTrackbar('SMax','image',0,255, nothing)
cv2.createTrackbar('VMax','image',0,255, nothing)
# Selecciona los valores maximos para el HSV
cv2.setTrackbarPos('HMax', 'image', 179)
cv2.setTrackbarPos('SMax', 'image', 255)
cv2.setTrackbarPos('VMax', 'image', 255)
```

Se iniciará el programa con los valores mínimos o máximos en las barras deslizantes ya que se quiere dejar por predeterminado como nosotros realmente los vemos.

```
# Inicializa el programa con los valores en su minimo primeramente
hMin = sMin = vMin = hMax = sMax = vMax = 0
phMin = psMin = pvMin = phMax = psMax = pvMax = 0
wait_time = 33
cap=cv2.VideoCapture(0)
```

Mientras que tengamos la cámara encendida y el programa lo detecta, leerá los datos para que pueda mostrar la imagen, en tiempo real lo cual se podrá modificar de acuerdo con las barras deslizantes.

```
while(1):
    ret,image=cap.read()
    image=cv2.flip(image,1)
    output = image
    # Extrae las posiciones de las sliders cuando se modifique
    hMin = cv2.getTrackbarPos('HMin','image')
    sMin = cv2.getTrackbarPos('SMin','image')
    vMin = cv2.getTrackbarPos('VMin','image')
    hMax = cv2.getTrackbarPos('HMax','image')
    sMax = cv2.getTrackbarPos('SMax','image')
    vMax = cv2.getTrackbarPos('VMax','image')
```

Cada vez que se modifique un valor mediante la barra, este se mostrara en un texto al lado de este, además de crear la imagen de la cámara que cambiara de acuerdo con los valores.

```
    # Muestra los valores minimos y maximos en pantalla
    lower = np.array([hMin, sMin, vMin])
    upper = np.array([hMax, sMax, vMax])
    # Crea la imagen HSV en camara
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    output = cv2.bitwise_and(image,image, mask= mask)
```

Cada modificación realizada se manda a la ventana de comandos del Python para su revisión posterior.

```
    # Manda al python los valores si existe alguna modificacion
    if( (phMin != hMin) | (psMin != sMin) | (pvMin != vMin) | (phMax != hMax) | (psMax != sMax) | (pvMax != vMax) ):
        print("[%d,%d,%d],[%d,%d,%d]" % (hMin , sMin , vMin, hMax, sMax , vMax))
        phMin = hMin
        psMin = sMin
        pvMin = vMin
        phMax = hMax
        psMax = sMax
        pvMax = vMax
```

Mostraremos de forma simple lo que se vea en la cámara en primer lugar y sin algún retraso.

```
    # Muestra la imagen    cv2.imshow('image',output)
    # Espera un tiempo para evitar congelamientos en el video
    if cv2.waitKey(wait_time) & 0xFF == 27:        break
    cv2.destroyAllWindows()
```

III. Resultados y Discusiones

Al finalizar con el proyecto, podemos obtener algunos resultados. En el área del funcionamiento es posible comparar otros prototipos y/o productos comerciales que son posibles de encontrar en internet. Se demuestra las comparaciones categorizadas por los siguientes ámbitos: ensamblaje, reconocimiento y la velocidad.

A. Ensamblaje

Se puede destacar con la figura 8 que el robot realizado tiene un gran tamaño para ser un solucionador de cubo Rubik comerciales. De esta forma se obtiene mejor estabilidad de agarre y una mejor organización del espacio de trabajo para los actuadores y servomotores que se incluyeron en la parte inferior del robot para poder sujetar los soportes del cubo Rubik.

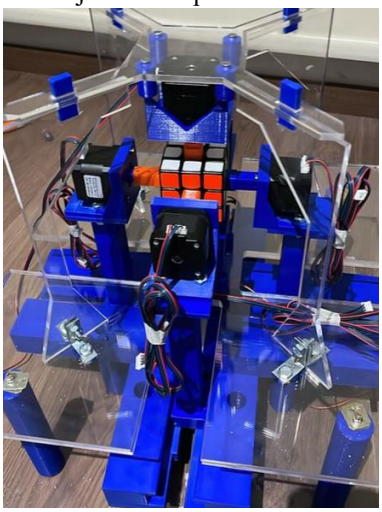

Figura 8. Robot solucionador completo del proyecto

B. Escaneo del Cubo

En la etapa del escaneo con el modelo de color HSV se utiliza visión artificial por computador, de esta forma se tiene una facilidad de visualización, velocidad y entendimiento. Sin embargo, pueden existir otros métodos como la implementación del programa completo en el mismo Arduino con pantallas integradas LCD. De todas formas, se decidió utilizar un computador y el proceso de detección de colores se puede apreciar en la figura 9. Se puede ver que el cubo se encuentra dentro de una caja y la cámara está instalada adentro.

Figura 9. Interfaz de escaneo del cubo virtual del proyecto actual

C. Velocidades y Tiempos

Las pruebas que se llegaron a realizar fueron simples y rápidas ya que solo constaron en reconocer el cubo, y que el mismo robot pueda solucionarlo apenas se mande el comando de "Iniciar". Los tiempos tomados por un cronometro se calculan el tiempo de armado mediante los valores que se les da en la programación del Arduino IDE, teniendo con cada instrucción aproximadamente, por lo que al girar un ángulo de 90 grados se usa 50 pasos, de esta forma pudiendo calcular las velocidades y tiempos aproximados del robot solucionador del cubo.

D. Comparación

El diseño presentado por Miguel Rubio propone la automatización del robot mediante el uso de cámaras capaces de escanear las 3 caras, en conjunto con un Raspberry. En comparación, nuestra propuesta no alcanza un nivel completo de automatización, ya que depende del uso de una computadora monitoreada directamente por nosotros. En cambio, el enfoque de Rubio minimiza significativamente la intervención humana al utilizar el microcontrolador Raspberry.[1] A pesar de tener estas mejoras en el robot su tiempo de resolución llega a ser de 7s. Sin embargo, nuestro robot alcanza una resolución del cubo con un tiempo promedio de 6s lo que indica una optimización destacable.

En el trabajo de investigación realizado por Daniel y Frederik realizaron diferentes algoritmos nombrándolos lentos y rápidos apoyados de varias pruebas realizadas teniendo como resultado tiempos de 3.99s y 9.88s respectivamente. Sin embargo, en nuestro robot al realizar un solo algoritmo el cual corresponde al "Kociemba" solo se pudo realizar pruebas de dicho algoritmo [16].

Para llegar a esta comparación con los robots anteriores, destacamos como nuestro producto final usa la visión artificial mediante computadora lo cual puede ser poco convencional pero nos entrega una precisión alta, y con una interfaz sencilla como es vista en la figura 8. En cambio, el robot Moyu, mostrado en la figura 10, presenta un brillo en la parte inferior, lo cual indica que está detectando constantemente esa cara del cubo mientras rota. Esto se debe a que el robot cumple funciones adicionales más allá de simplemente resolverlo, aunque para ello utiliza únicamente sensores de color.

Finalmente, con la última comparación del robot de Mitsubishi, podemos aclarar que estos tienen componentes y materiales de muy alta calidad por lo que usan cámaras en ambas diagonales como es mostrada en acercamiento en la figura 11, para un escaneo exacto y rápido en el momento, concluyendo que nuestro proyecto puede abarcar una mejor precisión que el Moyu pero no tan alta como el Mitsubishi.

Figura 10. Robot Moyu

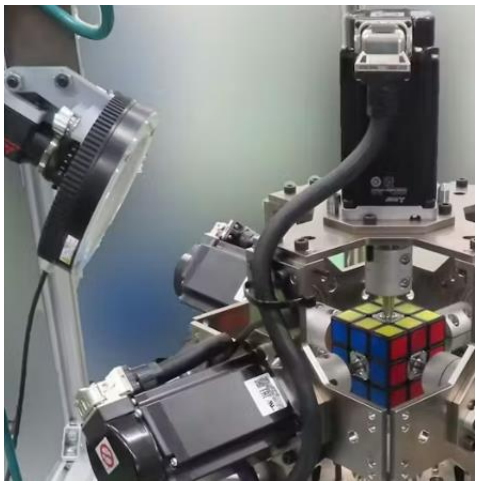

Figura 11. Robot Mitsubishi

TABLA 3
TIEMPO DE RESOLUCION DE CUBO PARA CONFIGURACION RAPIDA Y LENTA

Prueba	Movimiento	Promedio lento [s]	Promedio Rápido [s]
1	18	11.91	4.64
2	19	12.03	5
3	20	12.9	4.95
4	19	11.56	4.67
5	20	13.54	5.29
6	17	12.09	4.72
7	18	12.32	4.7
8	19	12.29	4.85
9	18	11.39	5.3
10	17	10.59	4.14
11	19	12.68	4.94
12	19	9.88	3.99
13	20	13.95	5.34
14	19	11.35	4.48
15	18	9.88	3.99
16	19	12.61	4.86
17	20	11.54	4.44
18	16	12.25	4.7
19	20	12.28	4.75
20	20	11.56	4.54

En la Tabla 3, se muestran los tiempos promedios obtenidos durante la resolución de cubo Rubik, según los resultados reportados por Daniel Gronas y Fredrik Mazur. En sí la categoría “Promedio Rápido” registra un tiempo promedio de 4.74 segundos, estos valores se tomaron como referencia para poder evaluar tanto el rendimiento como el control de nuestro robot. Si bien nuestro robot alcanza un tiempo promedio de 6 segundos, se encuentra dentro de un rango funcional comparable. Esta proximidad es significativa, ya que nos indica el desempeño realizado por parte del prototipo. De igual forma esta comparación nos permite validar la eficacia del diseño mecatrónico propuesto. [16]

IV. Conclusiones

Hablando en términos estructurales se hizo por medio del programa Inventor el cual fue el que mejor se adaptó a nuestro proyecto por la experiencia realizada en anteriores proyectos como rutas fáciles en el diseño adaptando de una manera eficaz para que pueda mantener los motores y servomotores, además del cubo de Rubik para que se pueda manipular rápidamente y con suavidad. Llegando a una estructura y ensamblada de forma que funcione de acuerdo como se planteó.

De esta forma se logró diseñar un robot que resuelve el cubo Rubik mediante el uso de comandos serial que son enviados por el software Python con el soporte de una computadora, donde este mismo obtiene la solución con lo reconocido en la cámara y los valores que se convierte en este mismo.

Con lo destacado anteriormente se consigue realizar la programación con el que se pudo hacer la detección de los colores en cada cara del cubo mediante las máscaras HSV, de esta forma se controlaron los parámetros de tono, brillo y saturación. Para ello se tuvo que adaptar la iluminación por medio de una caja de escaneado debido a que los colores naranja y rojo presentaban inconvenientes que se llegaron a solucionar.

Fue posible la conexión entre Python y Arduino, esto mediante la búsqueda de opciones e información sobre cómo se realizaron en otros trabajos y/o proyectos por lo que se optó por el comando serial, teniendo una facilidad de programación de esta manera teniendo una forma de poder comunicarse con el microcontrolador de Arduino que pueda manejar de una mejor eficiencia los actuadores conectados a la misma.

Finalmente, se destaca el tiempo obtenido de resolución el cual abarca un tiempo aproximado de 6s siendo un indicador clave de la eficiencia del sistema implementado. Este resultado refleja la configuración como la optimización de los parámetros, así como la estructura previamente discutidos.

Referencias

[1] J. César Llerena Sánchez, M. Jaime Quispe Ccachuco, y C. Pio Castillo Cáceres, “Design and Implementation of a Delta-type Parallel Robot with Artificial Vision to for object tracking”, en *Proceedings of the LACCEI international Multi-conference for Engineering, Education and Technology*, Latin American and Caribbean Consortium of Engineering Institutions, 2024. doi: 10.18687/LACCEI2024.1.1.858.

[2] A. V. Diaconu, A. Costea, y M. A. Costea, “Color image scrambling technique based on transposition of pixels between RGB channels using Knight’s moving rules and digital chaotic map”, *Math Probl Eng*, vol. 2014, 2014, doi: 10.1155/2014/932875.

[3] A. Nair, D. Dalal, y R. Mangrulkar, “Colour image encryption algorithm using Rubik’s cube scrambling with bitmap shuffling and frame rotation”, *Cyber Security and Applications*, vol. 2, ene. 2024, doi: 10.1016/j.csa.2023.100030.

[4] B. Ambrus, A. Nyéki, G. Teschner, y L. Moldvai, “The Effect of Illumination on HSV Colour Segmentation for Ripe Tomatoes based on Machine

Vision", *Chemical Engineering Transactions* , vol. 114, pp. 829–834, 2024, doi: 10.3303/CET24114139.
[5] M. Li, H. Xia, P. Li, y H. Lu, "Gesture Recognition Algorithm Based on Computer Vision", en *Journal of Physics: Conference Series*, Institute of Physics, 2023. doi: 10.1088/1742-6596/2508/1/012036.
[6] S. Liu, W. Long, L. He, Y. Li, y W. Ding, "Retinex-based fast algorithm for low-light image enhancement", *Entropy*, vol. 23, jun. 2021, doi: 10.3390/e23060746.
[7] T. Zhang y Y. Ma, "Artificial Intelligence Vision Based on Computer Digital Technology in 3D Image Colour Processing", en *Journal of Physics: Conference Series*, IOP Publishing Ltd, jun. 2021. doi: 10.1088/1742-6596/1952/2/022008.
[8] Jorge González Aparicio, "Fabricación de prototipo de robot solucionador del cubo de Rubik", 2022, Consultado: el 9 de enero de 2025. [En línea]. Disponible en: https://biblus.us.es/bibing/proyectos/abreproy/94274/descargar_fichero/TFG-4274+GONZ%C3%81LEZ+APARICIO%2C+JORGE.pdf
[9] D. Sugiura, S. Mitsuya, H. Takahashi, R. Yamamoto, y Y. Miyazawa, "Microcontroller-based water control system for evaluating crop water use characteristics", *Plant Methods*, vol. 20, dic. 2024, doi: 10.1186/s13007-024-01305-0.
[10] H. F. A. Kusuma, I. R. Mutiarso, y P. Megantoro, "Design of Automatic and Smart Disinfectant Sprayer Using Microcontroller Atmega328", en *AIP Conference Proceedings*, American Institute of Physics, oct. 2024. doi: 10.1063/5.0235327. m
[11] H. Zhang, Y. Zhao, F. Yao, L. Xu, P. Shang, y G. Li, "An adaptation strategy of using LDA classifier for EMG pattern recognition", en *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 2013, pp. 4267–4270. doi: 10.1109/EMBC.2013.6610488.
[12] M. Bathre y P. K. Das, "Design & implementation of smart power management system for self-powered wireless sensor nodes based on fuzzy logic controller using Proteus & Arduino Mega 2560 microcontroller", *J Energy Storage*, vol. 97, sep. 2024, doi: 10.1016/j.est.2024.112961.
[13] M. Zouheir, M. Zniber, S. Qudsia, y T. P. Huynh, "Real-time humidity sensing by integration of copper sulfide nanocomposite with low-cost and wireless Arduino platform", *Sens Actuators A Phys*, vol. 319, mar. 2021, doi: 10.1016/j.sna.2021.112541.
[14] A. Elassal, M. Abdelaal, M. Osama, y H. Elhnidy, "Low-cost parallel delta robot for a pick-and-place application with the support of the vision system", *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 8, jun. 2024, doi: 10.1016/j.prime.2024.100518.
[15] M. Rubio Arroyo, "Robot para resolver el cubo de rubik rubok.", https://gredos.usal.es/handle/10366/150070.
[16] D. GRÖNÅS y F. MAZUR, "Robotic Cuber : A Rubik´s Cube solving robot", https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1462066&dswid=-7191.