

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey



**Tecnológico
de Monterrey**

Implementación de robótica inteligente
TE3002B, Grupo 501

State of the Art Movement Replication with Computer Vision

Elaborado Por:

Emilio Rizo De la Mora	A01721612
Jose Pablo Cedano Serna	A00832019
Luis Antonio Zermeno De Gorordo	A01781835
Omar Flores Sánchez	A01383530
Luis Mario Lozoya Chairez	A00833364
Jorge Axel Castruita Bretado	A00832843
Rodrigo Escandón López Guerrero	A01704287

Dr. Luis Alberto Muñoz

Objetivo

El objetivo de este proyecto es controlar un brazo robótico simulado mediante la detección de color en tiempo real. El control se realiza detectando un objeto de color específico a través de una cámara, lo que permite manipular la simulación del brazo robótico en un entorno Pygame.

Solución

Elección del software y herramientas:

- OpenCV: Se utiliza para el procesamiento y análisis de las imágenes capturadas por la cámara. Permite detectar y seguir un objeto de color específico.
- Pygame: Se usa para crear y manipular una interfaz gráfica donde se simula el movimiento del brazo robótico.

Detalles de implementación:

1. Inicialización de Pygame y OpenCV:
 - Configuración de la ventana de Pygame.
 - Inicio de la captura de vídeo con OpenCV.
2. Detección de color:
 - Conversión de la imagen a espacio de color HSV.
 - Creación de una máscara para aislar el rango de color deseado.
 - Detección de contornos y selección del contorno con mayor área.
3. Simulación de movimientos:
 - Actualización de la posición del objeto detectado en la interfaz Pygame.
 - Representación visual de la orientación y posición del objeto.

Código relevante

- El script incluye funciones como `detect_color` para identificar la posición y orientación del objeto de color.
- `draw_pygame_scene` actualiza la interfaz gráfica basada en los datos de posición del objeto.

```
import cv2
import numpy as np
import pygame
import sys

# Inicialización de Pygame
pygame.init()
pygame_screen = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Simulated Robotic Arm')

# Inicialización de OpenCV
cap = cv2.VideoCapture(0)
lower_color = np.array([110, 50, 50]) # Supuesto rango bajo para azul
upper_color = np.array([130, 255, 255]) # Supuesto rango alto para azul

def detect_color(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower_color, upper_color)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        max_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(max_contour)
        orientation = 'horizontal' if w > h else 'vertical'
```

```
        return (x, y, w, h), (x + w // 2, y + h // 2), orientation
    return None, None, None

def draw_pygame_scene(x, y, orientation):
    pygame_screen.fill((0, 0, 0))
    # Mesa y hueco
    pygame.draw.rect(pygame_screen, (0, 0, 255), (0, 400, 640, 80))
    # Mesa
    pygame.draw.rect(pygame_screen, (100, 100, 100), (295, 400, 50,
80)) # Hueco
    # Restricción de altura para la pinza
    pinza_y = y if 295 <= x <= 345 else min(y, 370) # Limita la
altura para que no descienda sobre la mesa
    # Brazo y pinza
    pygame.draw.line(pygame_screen, (0, 255, 0), (320, 0), (320,
pinza_y), 5) # Brazo vertical
    pygame.draw.line(pygame_screen, (0, 255, 0), (320, pinza_y), (x,
pinza_y), 5) # Brazo horizontal
    if orientation == 'horizontal':
        pygame.draw.rect(pygame_screen, (0, 255, 0), (x - 25, pinza_y
- 10, 50, 20)) # Pinza horizontal
    else:
        pygame.draw.rect(pygame_screen, (0, 255, 0), (x - 10, pinza_y
- 25, 20, 50)) # Pinza vertical
    pygame.display.flip()

def main():
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        tracked_rect, tracked_pos, orientation = detect_color(frame)
        if tracked_rect:
            cv2.rectangle(frame, (tracked_rect[0], tracked_rect[1]),
(tracked_rect[0] + tracked_rect[2], tracked_rect[1] +
tracked_rect[3]), (0, 255, 0), 2)
            mapped_x = int(np.interp(tracked_pos[0], [0,
frame.shape[1]], [0, 640]))
```

```
        mapped_y = int(np.interp(tracked_pos[1], [0,
frame.shape[0]], [0, 480]))
        draw_pygame_scene(mapped_x, mapped_y, orientation)
    else:
        pygame_screen.fill((0, 0, 0))
        pygame.display.flip()

    cv2.imshow("Tracking", frame)
    for event in pygame.event.get():
        if event.type is pygame.QUIT:
            pygame.quit()
            cap.release()
            cv2.destroyAllWindows()
            sys.exit()

    if cv2.waitKey(1) & 0xFF == ord(' '):
        break

    pygame.quit()
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Pasos para la ejecución

python codigointegrado_1.3.py

Este comando inicia la simulación, mostrando una ventana donde se reflejan los movimientos del brazo robótico simulado en respuesta a la posición.

Video Demostrativo

📁 minireto3_vision.mp4

Repositorio de Github

<https://github.com/jpcedano/State-of-the-Art-Movement-Replication-with-Computer-Vision>

Conclusión

El proyecto demuestra la viabilidad de controlar un brazo robótico mediante la detección de objetos de color, usando herramientas accesibles como OpenCV y Pygame. Esta solución puede extenderse para controlar robots físicos en aplicaciones industriales o educativas.

