Proyecto final. Sistemas de visión de industrial

El objetivo principal de este proyecto es desarrollar una aplicación que utilice una cámara web para capturar imágenes y procesarlas para detectar un gesto de la mano. En función del gesto, el sistema enviará una señal a un microcontrolador Arduino, el cual controlará el encendido de un arreglo de LEDs.

Introducción

En el ámbito de los sistemas de visión industrial, la integración de técnicas de procesamiento de imágenes con el control de hardware ofrece soluciones innovadoras y prácticas para diversas aplicaciones. Este proyecto tiene como objetivo desarrollar un sistema que utilice una cámara web para capturar imágenes en tiempo real, procesarlas y detectar gestos de la mano del usuario. Basado en los gestos reconocidos, el sistema interactúa con un microcontrolador Arduino para controlar el encendido de un arreglo de LEDs.

La principal motivación de este proyecto es demostrar cómo la visión por computadora puede combinarse con sistemas embebidos para crear interfaces naturales y eficientes. La implementación utiliza Python y OpenCV para el procesamiento de imágenes, mientras que la comunicación con Arduino se realiza mediante la biblioteca pySerial. Este enfoque permite una interacción intuitiva entre el usuario y el sistema, donde gestos simples como una mano cerrada, dedos levantados o movimientos se traducen en señales para controlar los LEDs. El proyecto busca no solo implementar esta funcionalidad, sino también explorar los desafíos asociados con la detección de gestos, las condiciones de iluminación y la optimización de la comunicación hardware-software.

Descripción del proyecto

1. Captura de Imágenes: Utilizarás una webcam para adquirir imágenes en tiempo real de la mano del usuario.
1. Procesamiento de Imágenes: Usarás técnicas de procesamiento de imágenes para detectar el gesto de la mano mediante OpenCV y Python.
1. Detección de Gesto: El sistema debe ser capaz de identificar el gesto de la mano y, dependiendo de este, realizar la siguiente acción: o Si se detecta un gesto de una mano cerrada, o puño, no se encenderá ningún LED. o Si se detecta el gesto del dedo índice levantado, se encenderán un LED. o Si se detecta el gesto del dedo índice y el dedo medio levantados, se encenderán dos LED. o Si se detecta el gesto del de la mano moviéndose, se encenderán todos los LEDs.
1. Interacción con Arduino: Utilizarás un Arduino (cualquier modelo compatible, como Arduino Uno) para controlar los LEDs. El Arduino recibirá las señales desde la computadora (donde se está procesando la imagen) a través de una conexión serie (puerto COM). El código de Arduino debe estar configurado para encender los LEDs según la señal recibida.
1. Interfaz: La comunicación entre el software que procesa las imágenes y Arduino será a través del puerto serie (usualmente con la librería Serial en Arduino y funciones de comunicación en Python, como pySerial o similares).

1.0.1 1. Importar bibliotecas y crear variables

En esta parte del código además de crear variables también se importó la biblioteca del Arduino y también se captó desde el puerto COM5 el Arduino, de esta forma se conectó Python con el Arduino para hacer el encendido de los LEDs.

```
import numpy as np
import cv2
import serial
import random
arduino = serial.Serial('COM5', 9600, timeout=2)  # Timeout más largo

# Mantener el background frame para quitarlo posteriormente
background = None
# Guarda los datos de la mano para que todos sus detalles estén en un
solo␣lugar.
hand = None
# Variables para contar cuántos fotogramas han pasado y para
establecer el␣tamaño de la ventana.
frames_elapsed = 0
FRAME_HEIGHT = 300
FRAME_WIDTH = 400
# Prueba a editarlas si tu programa tiene problemas para reconocer tu
tono de␣piel.
CALIBRATION_TIME = 30
BG_WEIGHT = 0.5
OBJ_THRESHOLD = 50
```

1.0.2 2. Clase HandData: Una clase para guardar todos los detalles de la mano y las banderas

```
class HandData:
    # Atributos de la clase
    top = (0, 0)
    bottom = (0, 0)
    left = (0, 0)
    right = (0, 0)
    centerX = 0
    prevCenterX = 0
    isInFrame = False
    isWaving = False
    fingers = None
    gestureList = []

    # Constructor
    def __init__(self, top, bottom, left, right, centerX):
        self.top = top
        self.bottom = bottom
        self.left = left
        self.right = right
        self.centerX = centerX
        self.prevCenterX = 0
```

```
        self.isInFrame = False
        self.isWaving = False

    # Método para actualizar los valores
    def update(self, top, bottom, left, right):
        self.top = top
        self.bottom = bottom
        self.left = left
        self.right = right

    # Método para verificar si hay un movimiento de "saludo"
    def check_for_waving(self, centerX):
        self.prevCenterX = self.centerX
        self.centerX = centerX
        if abs(self.centerX - self.prevCenterX) > 3:
            self.isWaving = True
        else:
            self.isWaving = False
```

1.0.3 3. write_on_image(): Escribir información relacionada con el gesto de la mano y delimitar la región de interés

En esta sección del código se agregaron 2 puntos. El primero que se agregó un tercer led y el segundo punto se agrega una función “random” para que los LEDs se enciendan aleatoriamente.

```
# Aquí tomamos el fotograma actual, el número de fotogramas
transcurridos
# y cuántos dedos hemos detectado para poder imprimir en la pantalla
# qué gesto se está produciendo (o si la cámara se está calibrando).
# Inicializar la conexión con el Arduino (ajusta el puerto y la
velocidad de baudios según tu configuración)
def write_on_image(frame):
    global arduino, previous_message
    text = "Buscando..."
    message = "0"  # Por defecto, ningún LED encendido

    if frames_elapsed < CALIBRATION_TIME:
        text = "Calibrando..."
    elif hand is None or not hand.isInFrame:
        text = "Mano no detectada"
    else:
        if hand.isWaving:
            text = "Moviendo"
            message = str(random.randint(1, 3))  # Generar
aleatoriamente "1", "2" o "3"
        elif hand.fingers == 0:
            text = "Cero"
            message = "0"
        elif hand.fingers == 1:
```

```
            text = "Uno"
            message = "1"
        elif hand.fingers == 2:
            text = "Dos"
            message = "2"
        elif hand.fingers == 3:
            text = "Tres"
            message = "3"

    # Enviar el mensaje al Arduino
    arduino.write(f"{message}\n".encode())  # Agregar salto de línea
para delimitar

    # Mostrar el texto en la imagen
    cv2.putText(frame, text, (10, 20), cv2.FONT_HERSHEY_COMPLEX, 0.4,
(0, 0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, text, (10, 20), cv2.FONT_HERSHEY_COMPLEX, 0.4,
(255, 255, 255), 1, cv2.LINE_AA)

    # Resaltar la región de interés con un recuadro
    cv2.rectangle(frame, (region_left, region_top), (region_right,
region_bottom), (255, 255, 255), 2)
```

0.1.4 4. get_region(): Separa la región de interés y la prepara para la detección de bordes

```
def get_region(frame):
    # Separar la región de interés del resto del fotograma
    region = frame[region_top:region_bottom, region_left:region_right]

    # Transforma a escala de grises para que podamos detectar los
bordes más fácilmente.
    region = cv2.cvtColor(region, cv2.COLOR_BGR2GRAY)

    # Utiliza un desenfoque gaussiano para evitar que el ruido del
fotograma se etiquete como borde.
    region = cv2.GaussianBlur(region, (5, 5), 0)

    return region
```

0.1.5 5. get_average(): Crear una media ponderada del fondo para la diferenciación de imágens

```
def get_average(region):
    # Tenemos que utilizar la palabra clave global porque queremos
editar la variable global.
    global background

    # Si aún no hemos capturado el fondo, haz que la región actual sea
el fondo.
```

```
    if background is None:
        background = region.copy().astype("float")
        return

    # De lo contrario, añade este fotograma capturado a la media de
los fondos.
    cv2.accumulateWeighted(region, background, BG_WEIGHT)
```

0.1.6 6. segment(): Utilizar la diferenciación de imágenes para separar la mano del fond

```
# Aquí utilizamos la diferenciación para separar el fondo del objeto
de interés.
def segment(region):
    global hand

    # Encuentra la diferencia absoluta entre el fondo y el fotograma
actual.
    diff = cv2.absdiff(background.astype(np.uint8), region)

    # Umbral de esa región con un estricto 0 o 1, por lo que sólo el
primer plano se mantiene.
    thresholded_region = cv2.threshold(diff, OBJ_THRESHOLD, 255,
cv2.THRESH_BINARY)[1]

    # Obtener los contornos de la región, que devolverá un contorno de
la mano.
    (contours, _) = cv2.findContours(thresholded_region.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Si no conseguimos nada, no hay mano.
    if len(contours) == 0:
        if hand is not None:
            hand.isInFrame = False
        return None  # Devuelve explícitamente None si no se encuentra
una mano.

    # En caso contrario, devuelve una tupla de la mano rellenada
(thresholded_region),
    # junto con el contorno (segmented_region).
    else:
        if hand is not None:
            hand.isInFrame = True
        segmented_region = max(contours, key=cv2.contourArea)
        return (thresholded_region, segmented_region)
```

0.1.7 7. get_hand_data(): Busca las extremidades de la mano y colócalas en el objeto global man

```
def get_hand_data(thresholded_image, segmented_image):
    global hand
```

```
    # Encierra el área alrededor de las extremidades en un "convex
hull" para␣conectar todos los afloramientos.
    convexHull = cv2.convexHull(segmented_image)
    # Encuentra las extremidades del "convex hull" y guárdalas como
puntos.
    top = tuple(convexHull[convexHull[:, :, 1].argmin()][0])
    bottom = tuple(convexHull[convexHull[:, :, 1].argmax()][0])
    left = tuple(convexHull[convexHull[:, :, 0].argmin()][0])
    right = tuple(convexHull[convexHull[:, :, 0].argmax()][0])
    # Obtiene el centro de la palma, así podremos comprobar si ondea
y␣encontrar los dedos.
    centerX = int((left[0] + right[0]) / 2)
    # Ponemos toda la información en un objeto para extraerla
facilmente
    if hand == None:
        hand = HandData(top, bottom, left, right, centerX)
    else:
        hand.update(top, bottom, left, right)
    # Sólo comprueba la ondulación cada 6 fotogramas.
    if frames_elapsed % 6 == 0:
        hand.check_for_waving(centerX)
    # Contamos el número de dedos cada frame,
    # para no equivocarse, espera que pasen 12 frames
    hand.gestureList.append(count_fingers(thresholded_image))
    if frames_elapsed % 12 == 0:
        hand.fingers = most_frequent(hand.gestureList)
        hand.gestureList.clear()
```

0.1.8 8. count_fingers(): Cuenta el número de dedos utilizando una línea que cruce las puntas de los dedo

```
def count_fingers(thresholded_image):
    # Encuentra la altura a la que trazaremos la línea para contar los
dedos.
    line_height = int(hand.top[1] + (0.2 * (hand.bottom[1] -
hand.top[1])))

    # Obtiene la región lineal de interés a lo largo de donde estarían
los␣dedos.
    line = np.zeros(thresholded_image.shape[:2], dtype=int)

    # Traza una línea a través de esta región de interés, donde
deberían estar␣los dedos.
    cv2.line(line, (thresholded_image.shape[1], line_height), (0,
line_height),255, 1)

    # Hace una operación AND a nivel de bit para encontrar el punto en
el que la línea cruza la mano: aquí es donde están los dedos.
    line = cv2.bitwise_and(thresholded_image, thresholded_image, mask
```

```
= line.astype(np.uint8))

    # Obtiene los nuevos contornos de la línea. Los contornos son
básicamente pequeñas líneas formadas por huecos
    # en la línea grande a través de los dedos, por lo que cada uno
sería un¸dedo a menos que sea muy ancho.
    (contours, _) = cv2.findContours(line.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

    fingers = 0

    # Cuente los dedos asegurándose de que las líneas de contorno son
del «tamaño de un dedo», es decir, no demasiado anchas.
    # Así se evita que un gesto de «mano cerrada» se confunda con un
dedo.
    for curr in contours:
        width = len(curr)
        if width < 3 * abs(hand.right[0] - hand.left[0]) / 4 and width
> 5:
            fingers += 1
    return fingers
```

0.1.9 9. most_frequent(); Regresa el valor que aparece más frecuentemente en una list

```
def most_frequent(input_list):
    dict = {}
    count = 0
    most_freq = 0

    for item in reversed(input_list):
        dict[item] = dict.get(item, 0) + 1
        if dict[item] >= count :
            count, most_freq = dict[item], item
    return most_freq
```

0.1.10 10. Main function: Obtener datos de la cámara y llamar a funciones para comprenderlos

```
# Nuestra región de interés será la parte superior derecha del frame.
region_top = 0
region_bottom = int(2 * FRAME_HEIGHT / 3)
region_left = int(FRAME_WIDTH / 2)
region_right = FRAME_WIDTH
frames_elapsed = 0

capture = cv2.VideoCapture(0)

while True:
    # Guarda el fotograma de la captura de vídeo y redimensiónalo al
tamaño de la ventana.
```

```
    ret, frame = capture.read()
    frame = cv2.resize(frame, (FRAME_WIDTH, FRAME_HEIGHT))

    # Voltea el marco sobre el eje vertical para que funcione como un
espejo, lo que resulta más intuitivo para el usuario.
    frame = cv2.flip(frame, 1)

    # Separa la región de interés y la prepara para la detección de
bordes.
    region = get_region(frame)
    if frames_elapsed < CALIBRATION_TIME:
        get_average(region)
    else:
        region_pair = segment(region)
        if region_pair is not None:
            # Si tenemos las regiones segmentadas con éxito,
mostrarlas en otra ventana para el usuario.
            (thresholded_region, segmented_region) = region_pair
            cv2.drawContours(region, [segmented_region], -1, (255,
255, 255))
            cv2.imshow("Segmented Image", region)
            get_hand_data(thresholded_region, segmented_region)

    # Escribe en la pantalla la acción que realiza la mano y dibuja la
región de interés.
    write_on_image(frame)

    # Muestra el fotograma capturado anteriormente.
    cv2.imshow("Camera Input", frame)
    frames_elapsed += 1

    # Comprueba si el usuario desea salir.
    key = cv2.waitKey(1) & 0xFF
    if key == ord('x'):  # Salir con 'x'
        break
    elif key == ord('c'):  # Cerrar cámara con 'c'
        print("Cerrando la cámara...")
        capture.release()
        cv2.destroyAllWindows()
        exit()  # Salimos inmediatamente del programa

# Cuando salimos del bucle, también detenemos la captura.
capture.release()
cv2.destroyAllWindows()

# Cerrar conexión serial
arduino.close()
```