Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Tecnológico de Monterrey

Implementación de robótica inteligente
TE3002B, Grupo 501

Mini Reto 01: Visión

Elaborado Por:

Emilio Rizo	A01721612
Jose Pablo Cedano	A00832019
Luis Antonio Zermeño	A01781835
Omar Flores	A01383530
Luis Mario Lozoya	A00833364
Jorge Axel Castruita	A00832843
Rodrigo Escandón	A01704287

Funcionalidad Básica del Código

Nuestro modelo de cámara y tracking de credenciales implementa un funcionamiento a base de calibración. Este modelo nos permite encontrar valores específicos de calibración a raíz de una imagen para así poder realizar un trackeo de IDs que cumplen con ciertas características. Nuestro modelo comienza por desplegar una imagen donde se encuentran tres credenciales ubicadas en diferentes planos en esta imagen el usuario primero seleccionara un punto origen y procede a seleccionar las esquinas de las credenciales.

El código de preferencia requiere de mínimo 6 puntos nosotros en este caso le estamos otorgando 8. Estos puntos son analizados y se guardan como coordenadas. Estas coordenadas se guardan en un archivo .json el cual leeremos para poder empezar a establecer relaciones entre puntos 2d y 3d en nuestros planos.

El usuario una vez esté conforme con los puntos seleccionados presiona la tecla “space” para confirmar que ya ha seleccionado los puntos. Otra ventana emergente se presentará. Esta nueva ventana es con acceso a la cámara, misma donde el usuario presenta o muestra su credencial y tras dar doble click sobre la credencial analiza y selecciona el color de la credencial para en conjunto con otros parámetros como; figura geométrica, tonalidades, número de esquinas detectadas y un algoritmo de detección de contornos procede a hacer el tracking de la credencial.

Una vez detectada una credencial que hace match con el color seleccionado analiza si la credencial cuenta con algún rostro dentro de esta si se cumplen las condiciones la detecta como una ID. Ejemplo: detectando un color azul y analizando y encontrando esta figura hace el trackeo, si detecta un rostro dentro de la credencial y cumple con las características previamente mencionadas la identificara como una “credencial TEC”.

Por último utilizando todos los parámetros obtenidos hasta el momento hacemos uso de unas funciones y algoritmos para sacar nuestra matriz de cámara, coeficientes de distorsión, vectores de rotación y vectores de traslación. Por último con estos valores obtenemos nuestros valores focales y con unas conversiones de píxeles podemos saber el valor focal para calibración. Todo esto para poder implementar los valores mencionados en un cálculo que nos dará la dirección a la que se encuentra la credencial.

Código/Documentación Técnica

Python

```python
import cv2
import json
import os
import numpy as np

# Variables globales para valores HSV seleccionados
lower_color = np.array([110, 50, 50])
upper_color = np.array([130, 255, 255])

color_selection = {
    "blue": {
        "lower": np.array([110, 50, 50]),
        "upper": np.array([130, 255, 255]),
        "message": "Estudiante Tec"
    },
    "yellow": {
        "lower": np.array([20, 100, 100]),
        "upper": np.array([50, 255, 255]),
        "message": "Estudiante UDEM"
    },
}

def click_event(event, x, y, flags, params):
    """
    Handle mouse click events.

    Parameters:
    - event: The type of mouse event.
    - x: The x-coordinate of the mouse click.
    - y: The y-coordinate of the mouse click.
    - flags: Additional flags for the mouse event.
    - params: Additional parameters for the mouse event.

    Returns:
    None
    """
    if event == cv2.EVENT_LBUTTONDOWN:
        # Añadir punto a la lista global
        puntos_ref[img_name].append((x, y))
        # Dibujar el punto seleccionado en la imagen
        cv2.circle(img, (x, y), 5, (255, 0, 0), -1)
        cv2.imshow(img_name, img)

# Función para detectar objetos del color seleccionado
def detect_color(frame):
```

```
    """
    Detects and extracts colors from an input frame.

    Parameters:
    - frame: The input frame to detect colors from.

    Returns:
    - res: The result of bitwise-AND operation between the frame and the
color mask.
    - mask: The color mask obtained from thresholding the frame.

    """
    # Convertir de BGR a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Threshold la imagen HSV para obtener solo los colores en el rango
seleccionado
    mask = cv2.inRange(hsv, lower_color, upper_color)

    # Convertir la máscara de un solo canal a tres canales
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    # Asegurarse de que la máscara sea un entero sin signo de 8 bits
    mask = np.uint8(mask)

    # Bitwise-AND entre la máscara y la imagen original
    res = cv2.bitwise_and(frame, frame, mask=mask[:,:,0])

    return res, mask

# Función para seleccionar el color HSV haciendo doble clic
def on_double_click(event, x, y, flags, param):
    """
    Event handler for double-click events.

    Args:
        event: The type of event triggered.
        x: The x-coordinate of the event.
        y: The y-coordinate of the event.
        flags: Additional flags passed by OpenCV.
        param: Additional parameters passed to the event handler.

    Returns:
        None
    """
    global lower_color, upper_color
    if event == cv2.EVENT_LBUTTONDBLCLK:
        frame = param  # Obtener el frame actual
```

```
        # Convertir el píxel seleccionado a HSV
        hsv_pixel = cv2.cvtColor(np.uint8([[frame[y, x]]]),
cv2.COLOR_BGR2HSV)[0][0]
        print("Color seleccionado/Selected HSV:", hsv_pixel)
        # Actualizar los valores de color
        hue_adjustment = 20  # Ajuste de valor H, puedes ajustar este valor
según tu necesidad
        lower_color = np.array([max(hsv_pixel[0] - hue_adjustment, 0), 50,
50])
        upper_color = np.array([min(hsv_pixel[0] + hue_adjustment, 179),
255, 255])

# Función para calcular la distancia basada en el tamaño aparente
def calculate_distance(apparent_size, real_size_width, real_size_height,
focal_length_width, focal_length_height):
    """
    Calculates the distance between an object and a camera based on its
apparent size, real size, and focal length.

    Parameters:
    apparent_size (tuple): The apparent size of the object in pixels (width,
height).
    real_size_width (float): The real width of the object in a known unit of
measurement.
    real_size_height (float): The real height of the object in a known unit
of measurement.
    focal_length_width (float): The focal length of the camera for width in
pixels.
    focal_length_height (float): The focal length of the camera for height
in pixels.

    Returns:
    float: The calculated distance between the object and the camera in the
same unit of measurement as the real size.

    """
    distance_width = (real_size_width * focal_length_width) /
apparent_size[0]
    distance_height = (real_size_height * focal_length_height) /
apparent_size[1]
    distance = (distance_width + distance_height) / 2  # Distancia promedio
entre ancho y alto
    return distance

def main():
    """
    Main function for object detection and distance calculation.
```

```
    This function captures frames from a camera, detects objects of a
selected color,
    calculates their distance from the camera, and detects faces within the
objects.

    Returns:
        None
    """
    # Calibrate the camera and get focal lengths
    #focal_length_width, focal_length_height = click_event(event, x, y,
flags, params)

    cap = cv2.VideoCapture(1)

    # Real-world size of the object (in inches)
    real_object_width = 3.3  # Example: 3.3 inches
    real_object_height = 2  # Example: 2 inches

    # Focal length of the camera (in pixels) - You need to calibrate your
camera to obtain this value
    #focal_length_width = 539 #Numeros obtenidos de calibracion con ajedrez
# Example: 1000 pixels
    #focal_length_height = 549 #Numeros obtenidos de calibracion con ajedrez
# Example: 1000 pixels

    # Load face detection classifier
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    while True:
        ret, frame = cap.read()

        # Mostrar la imagen en la ventana para seleccionar el color
        cv2.imshow('Select Color', frame)

        # Esperar a que el usuario seleccione el color y ajustar los valores
        cv2.setMouseCallback('Select Color', on_double_click, frame)

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

    # Segunda parte: Deteción de credenciales y caras
    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()

        # Detectar objetos del color seleccionado
        color_detected, color_mask = detect_color(frame)
```

```
        # Encontrar contornos
        contours, _ = cv2.findContours(cv2.cvtColor(color_mask,
cv2.COLOR_BGR2GRAY), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Filtrar contornos basados en el área
        filtered_contours = []
        for contour in contours:
            area = cv2.contourArea(contour)
            if area > 1000:  # Ajustar este umbral según el tamaño esperado
de la credencial
                filtered_contours.append(contour)

        # Dibujar rectángulos alrededor de los objetos y calcular la
distancia
        for contour in filtered_contours:
            # Aproximar el contorno para verificar si es un rectángulo
            epsilon = .1 * cv2.arcLength(contour, True) #epsilon de 0.04
            approx = cv2.approxPolyDP(contour, epsilon, True)

            # Verificar si el contorno aproximado tiene cuatro esquinas
(rectángulo)
            if len(approx) == 4:
                x, y, w, h = cv2.boundingRect(contour)
                cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

                # Calcular el tamaño aparente
                apparent_size = (w, h)

                # Calcular la distancia
                distance = calculate_distance(apparent_size,
real_object_width, real_object_height, focal_length_width,
focal_length_height)

                # Mostrar distancia
                cv2.putText(frame, f"Distancia: {distance:.2f} pulgadas",
(x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

                # Detectar caras dentro del rectángulo
                roi_frame = frame[y:y + h, x:x + w]
                gray_roi = cv2.cvtColor(roi_frame, cv2.COLOR_BGR2GRAY)
                faces = face_cascade.detectMultiScale(gray_roi, 1.3, 5)
                for (fx, fy, fw, fh) in faces:
                    cv2.rectangle(roi_frame, (fx, fy), (fx + fw, fy + fh),
(0, 255, 0), 2)

                    # Verificar si se detecta una cara dentro del rectángulo
                if len(faces) > 0:
```

```
                    best_match_color = None
                    min_difference = float('inf')

                    # Iterar sobre cada color en color_selection y encontrar
el que se asimila mejor a los valores actuales
                    for color, values in color_selection.items():
                        lower_diff = np.abs(lower_color - values["lower"])
                        upper_diff = np.abs(upper_color - values["upper"])
                        total_diff = np.sum(lower_diff) + np.sum(upper_diff)

                        # Actualizar el mejor candidato si encontramos una
mejor coincidencia
                        if total_diff < min_difference:
                            min_difference = total_diff
                            best_match_color = color

                    # Mostrar el mensaje del mejor color coincidente
                    if best_match_color is not None:
                        cv2.putText(frame,
color_selection[best_match_color]["message"], (x, y - 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Mostrar el marco resultante
        cv2.imshow('Object Detection', frame)

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

    # Liberar la captura cuando se termine
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":

    # Diccionario para almacenar puntos de todas las imágenes
    puntos_ref = {}

    # Dirección de la imagen
    img_path = 'C:/Users/jpced/OneDrive/Documentos/ID7.jpg'
    img_name = os.path.basename(img_path)

    # Leer la imagen
    img = cv2.imread(img_path, 1)
    img = cv2.resize(img, (500, 400)) #500,400
    puntos_ref[img_name] = []  # Inicializar lista de puntos para esta
imagen

    cv2.imshow(img_name, img)
```

```
    cv2.setMouseCallback(img_name, click_event)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar los puntos en un archivo
    with open('puntos_referencia.json', 'w') as fp:
        json.dump(puntos_ref, fp)

    print("Puntos de referencia guardados en 'puntos_referencia.json'")

    img_path = 'C:/Users/jpced/OneDrive/Documentos/ID7.jpg'
    img = cv2.imread(img_path, 1)
    #img = cv2.resize(img, (500, 400))

    # Leer los puntos de referencia desde el archivo JSON
    with open('puntos_referencia.json', 'r') as fp:
        puntos_ref = json.load(fp)

    # Definir puntos de imagen y puntos de objeto (esquinas conocidas de la
credencial)
    puntos_imagen = []
    puntos_objeto = []

    credencial_ancho = 3.3  # Ancho de la credencial en pulgadas
    credencial_alto = 2.0  # Alto de la credencial en pulgadas

    for img_name, puntos in puntos_ref.items():
        for punto in puntos:
            puntos_imagen.append([punto])
            # Calcular puntos de objeto basados en el tamaño de las
credenciales
            x, y = punto
            puntos_objeto.append([(float(x / img.shape[1]) *
credencial_ancho, float(y / img.shape[0]) * credencial_alto, 0.0)])

    # Convertir puntos de imagen a NumPy array
    puntos_imagen = np.array(puntos_imagen, dtype=np.float32)

    # Convertir puntos de objeto a estructura requerida por
cv2.calibrateCamera (using cv2.Point3d)
    puntos_objeto_nested = []
    for punto in puntos_objeto:
        punto_3d = tuple(punto)  # Convert each point to a tuple
        puntos_objeto_nested.append(punto_3d)
    puntos_objeto = np.array(puntos_objeto_nested, dtype=np.float32)

    # Calcular parámetros intrínsecos y extrínsecos de la cámara
```

```
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera([puntos_objeto],
[puntos_imagen], (img.shape[1], img.shape[0]), None, None)

    # Imprimir resultados
    print("Matriz de la cámara (parámetros intrínsecos):")
    print(mtx)
    print("\nCoeficientes de distorsión:")
    print(dist)
    print("\nVectores de rotación:")
    print(rvecs)
    print("\nVectores de traslación:")
    print(tvecs)

    img = cv2.imread('C:/Users/jpced/OneDrive/Documentos/ID7.jpg')
    h, w = img.shape[:2]
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1,
(w,h))

    # undistort
    dst = cv2.undistort(img, mtx, dist, None, newcameramtx)
    # crop the image
    x, y, w, h = roi
    dst = dst[y:y+h, x:x+w]
    cv2.imwrite('calibresult.png', dst)

    tamano_pixel_mm = 0.0004

    focal_length_width = (mtx[0][0])*(tamano_pixel_mm)
    focal_length_height = (mtx[1][1])*(tamano_pixel_mm)

    print ("\nfocal width:")
    print (focal_length_width)
    print ("\nfocal height:")
    print (focal_length_height)

    main()

# los puntos se seleccionan de origen a credencial de la izquierda, luego
derecha, ultima credencial y de arriba a abajo
```

Video demostrativo

https://youtu.be/Jj_vVO3f80M