Funcionamiento de mostrar_video_udp

El método mostrar_video_udp recibe y muestra un flujo de video codificado enviado por UDP. Este proceso se usa típicamente en aplicaciones de visión en robótica (por ejemplo, robots móviles transmitiendo video) o cámaras IP que envían fotogramas a un servidor. La función realiza los siguientes pasos:

1. Recepción de video por UDP

- El socket UDP se configura con un tamaño de buffer (CHUNK_SIZE) para recibir fragmentos de datos. En un bucle while True, se llama a sock.recvfrom(CHUNK_SIZE) para obtener cada paquete de datos. Cada fragmento recibido contiene parte de un fotograma codificado (normalmente en JPEG) y, posiblemente, encabezados (por ejemplo, la longitud del fotograma).
- Se extrae la longitud del fotograma (por ejemplo, de los primeros bytes del paquete) y se acumula la carga útil en un buffer frame_data. Se lleva también un contador frame_length de la cantidad total de bytes esperados.
- Se repite la llamada recvfrom hasta que el buffer frame_data alcance la longitud necesaria para un fotograma completo. Este proceso de fragmentación y reconstrucción se ejemplifica en código: se recibe cada paquete con recvfrom, se acumula en frame_data y se suma su longitud (como en un ejemplo típico de streaming UDP)papaproxy.net. Cuando len(frame_data) >= frame_length, significa que tenemos los datos completos del fotograma.

2. Decodificación y preprocesamiento

- Una vez reunidos los bytes completos de un fotograma, se decodifican a imagen utilizando OpenCV. Es común usar np.frombuffer(frame_data, dtype=np.uint8) para convertir los bytes a un arreglo NumPy y luego cv2.imdecode(..., cv2.IMREAD_COLOR) para obtener el frame en color (espacio BGR). Así, frame = cv2.imdecode(np.frombuffer(frame_data, dtype=np.uint8), cv2.IMREAD_COLOR) reconstruye la imagen rawpapaproxy.net.
- A continuación se ajusta la orientación y tamaño del frame. Por ejemplo, si la cámara envía imágenes rotadas, se puede usar cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE) para rotar 90° a la derechageeksforgeeks.org. Luego se puede redimensionar con cv2.resize; por ejemplo, frame = cv2.resize(frame, (nuevo_ancho, nueva_altura)) o usando factores de escala (fx, fy)geeksforgeeks.org. Estas operaciones aseguran un tamaño de imagen apropiado para el procesamiento posterior.

3. Actualización de la variable global frame_actual

- La función mantiene una variable global (o compartida) frame_actual que guarda el último fotograma procesado para otras partes del programa. Al ser compartida entre hilos (por ejemplo, uno que recibe video y otro que actualiza la interfaz), se usa un **mutex** (threading.Lock) para sincronizar su escritura.
- En la práctica se haría algo como with lock: frame_actual = frame, de modo que la asignación sea atómica. La documentación de Python confirma que **Lock** crea un bloqueo primitivo: cuando un hilo adquiere el lock, otros hilos quedan bloqueados hasta que se liberadocs.python.org. De este modo se evita una condición de carrera al actualizar frame_actual.

4. Procesamiento de imagen

- El frame BGR se convierte al espacio de color HSV con cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)omes-va.com. El espacio HSV facilita la segmentación por color (por ejemplo, identificar un objeto amarillo).
- Se crea una **máscara** para el color amarillo usando cv2.inRange. Por ejemplo, con rangos adecuados de matiz (H), saturación (S) y valor (V) para amarillo:

python

CopiarEditar

mask = cv2.inRange(frameHSV, (H_min, S_min, V_min), (H_max, S_max, V_max))

La función inRange devuelve una imagen binaria (máscara) con 255 donde el píxel está dentro del rango de color y 0 fueramedium.com. Luego se aplica cv2.bitwise_and(frame, frame, mask=mask) para filtrar el frame original con la máscara, aislando las regiones amarillas. Como explica [17], “inRange devuelve una máscara con valores 1 donde detectó el color elegido y 0 donde no; bitwise_and filtra la imagen original con esa máscara”medium.com.

- A la máscara binaria se le aplican operaciones **morfológicas** para reducir ruido. Primero se define un *kernel* (p.ej. cv2.getStructuringElement) y se aplica cv2.erode(mask, kernel, iterations=1) para eliminar píxeles aislados de ruido (erosiona los bordes del objeto) y luego cv2.dilate(..., iterations=1) para restaurar el tamaño original del objeto. Este proceso (apertura/cierre) elimina manchas pequeñas: “la erosión elimina ruidos blancos pequeños (quitando partes exteriores), y luego la dilatación recupera la forma del objeto principal”geeksforgeeks.org.

- Sobre la máscara refinada se buscan **contornos** con contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE). En visión por computador, los contornos representan los bordes de los objetos en la imagengeeksforgeeks.org. Se examinan los contornos detectados (por ejemplo, el más grande) para identificar el cono amarillo. Con x,y,w,h = cv2.boundingRect(contour) se obtiene el rectángulo mínimo que encierra el objeto. El valor h (altura en píxeles del cono en la imagen) se usa para estimar la distancia.
- **Cálculo de distancia:** si se conoce la altura real del cono H_real y la longitud focal f de la cámara (calibrada previamente), la distancia al objeto se aproxima por la fórmula de semejanza:

ini

CopiarEditar

distancia = (H_real * f) / h_pixels

donde h_pixels es la altura en píxeles del cono. Este cálculo sigue la relación clásica focal/distancia usada en visión computacionalgeeksforgeeks.org. En [27] se presenta cómo, dado el ancho real y de referencia en píxeles, se obtiene la distancia: distance = (real_width * Focal_Length) / face_width_in_framegeeksforgeeks.org (aplicable igualmente usando alturas para objetos de perfil).

5. Visualización en Tkinter

- El frame resultante (posiblemente con anotaciones, como el rectángulo del cono y la distancia) se debe mostrar en la interfaz gráfica (Tkinter). Primero se convierte la imagen OpenCV (formato NumPy BGR) a formato compatible con Tkinter. Se suele convertir a RGB y luego a objeto PhotoImage:

python

CopiarEditar

import cv2

from PIL import Image, ImageTk

image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

im_pil = Image.fromarray(image_rgb)

imgtk = ImageTk.PhotoImage(image=im_pil)

El uso de Image.fromarray y ImageTk.PhotoImage es estándar (tal como se muestra en ejemplos, convirtiendo BGR a RGB)stackoverflow.comtutorialspoint.com. Finalmente, se actualiza el widget donde va el video: video_frame.config(image=imgtk) y se suele guardar video_frame.image = imgtk para evitar que Python elimine la imagen por garbage collection. Así, el widget video_frame muestra el último fotograma procesado en la ventana de Tkinterstackoverflow.comtutorialspoint.com.

6. Manejo de errores

- El código de recepción incorpora manejo de excepciones para robustez. Por ejemplo, se establece un timeout al socket (sock.settimeout(tiempo)) para que recvfrom no bloquee indefinidamente. Se envuelve la recepción en un bloque try/except, capturando socket.timeout y socket.error. Al ocurrir una excepción se puede abortar el bucle de recepción y cerrar el socket, como ejemplifica un código similar:

python

CopiarEditar

```
try:
  data, addr = sock.recvfrom(CHUNK_SIZE)
  # procesar datos...
except (socket.timeout, socket.error) as e:
  print("Error de red:", e)
  sock.close()
  break
```

En un ejemplo de streaming TCP/UDP se muestra exactamente este patrón: al capturar (socket.error, socket.timeout), se imprime el error y se cierra la conexiónstackoverflow.com.

- También se contempla errores de decodificación. Si cv2.imdecode falla (por datos corruptos o incompletos), la función retorna None. En tal caso se debe saltar ese fotograma en el procesamiento para evitar excepciones. En resumen, se usan bloqueos try/except para capturar errores de red o de lectura, cerrando recursos apropiadamentestackoverflow.com y asegurando que la aplicación siga estable aun ante pérdida de paquetes o fotogramas inválidos.

Resumen: en conjunto, mostrar_video_udp arma los fotogramas a partir de paquetes UDP (recvfrom), los decodifica y ajusta (rotación, escala), actualiza la vista global protegida por un mutex, procesa la imagen (espacio HSV, máscara de color amarillo, operaciones morfológicas, contornos) para detectar el cono y medir su distancia por fórmula focal, convierte el resultado a PhotoImage y actualiza la interfaz Tkinter. Cada etapa incluye controles de error para mantener el sistema robusto durante la transmisión en tiempo realpapaproxy.netomes-va.commedium.comgeeksforgeeks.orggeeksforgeeks.orgstackoverflow.comstackoverflow.com.

Fuentes: Documentación de OpenCV y ejemplos de streaming en Python (por ejemplo, uso de cv2.imdecode para reconstruir imágenes desde bufferspapaproxy.net, uso de cv2.cvtColor y máscaras HSVomes-va.commedium.com, operaciones morfológicasgeeksforgeeks.org, detección de contornosgeeksforgeeks.org), así como guías de uso de Lock en Pythondocs.python.org y de conversión de imágenes a Tkinterstackoverflow.comtutorialspoint.com. Estos conceptos, combinados, explican detalladamente el funcionamiento interno de la función.