Análisis Lineal del Código: `state_machine_logic.py`

El archivo `state_machine_logic.py` es el cerebro del sistema de control de acceso. Implementa una "máquina de estados finitos" (FSM), que es un modelo de comportamiento que define cómo el sistema reacciona a diferentes eventos (como la detección de una persona o la lectura de una tarjeta), pasando de un estado a otro de manera ordenada y predecible.

El script comienza importando librerías clave. `import time` y `import datetime` se usan para gestionar timeouts y registrar marcas de tiempo. Las librerías de visión por computadora son fundamentales aquí: `import cv2` para el manejo de la cámara, `import face_recognition` para la lógica de reconocimiento facial, y `from pyzbar.pyzbar import decode as decode_qr` para decodificar códigos QR. También se importan los módulos propios del proyecto, como `arduino_comms` para interactuar con el hardware y `db_manager` para consultar la base de datos.

Una de las estructuras más importantes es la clase `EstadoSistema(Enum)`. Un `Enum` (enumeración) es una forma de crear un conjunto de constantes con nombre. Aquí, define todos los posibles estados en los que puede estar el sistema, como `REPOSO`, `ESPERANDO_VALIDACION_RFID`, `ABRIENDO_PUERTA` o `EMERGENCIA_ACTIVA`. Usar un `Enum` hace que el código sea mucho más legible y menos propenso a errores que usar simples strings o números.

Se declaran varias variables globales para mantener el estado de la lógica. `estado_actual_sistema` almacena el estado actual, inicializado en `EstadoSistema.REPOSO`. `puerta_logicamente_abierta` es una bandera para saber si la puerta debería estar abierta o cerrada. Se usan variables de tiempo como `tiempo_inicio_estado_actual_s` para controlar los timeouts en cada estado. `cap_camara` guardará el objeto de la cámara cuando esté en uso. `protocolo_seleccionado_actual` es un diccionario que define qué métodos de validación (RFID, QR, facial) están activos, y `estado_validacion_secuencial` almacena los resultados de validaciones intermedias (por ejemplo, si el RFID fue exitoso y ahora se espera el reconocimiento facial).

La función `determinar_protocolo_activo` es muy interesante, ya que lee el estado de dos interruptores físicos del Arduino (obtenidos de `s1_estado_hw` y `s2_estado_hw`) para configurar dinámicamente el protocolo de seguridad. Dependiendo de qué interruptores estén activados, el sistema puede requerir "Solo RFID", "RFID + Facial", etc. Esto permite cambiar el nivel de seguridad del sistema físicamente.

La función `cambiar_estado(nuevo_estado, mensaje_gui)` es la responsable de gestionar las transiciones. Cuando se llama, actualiza la variable `estado_actual_sistema`, reinicia el temporizador de estado y, crucialmente, gestiona los recursos. Por ejemplo, si el sistema sale de un estado que usaba la cámara (como `ESPERANDO_VALIDACION_FACIAL`), esta función se encarga de liberarla con `cap_camara.release()` para que no quede activa innecesariamente. También actualiza un `Label` en la interfaz gráfica con mensajes para el usuario, cambiando el color del texto a verde para éxito o rojo para errores.

La función principal es `logica_maquina_estados()`, que se ejecuta en un hilo continuo. Al iniciar, carga datos importantes como los "encodings" faciales de la base de datos con `facial_recognition_utils.cargar_encodings_faciales_al_inicio()`. El corazón de esta función es un gran bucle `while hilo_maquina_estados_activo:`.

Dentro del bucle, lo primero que se hace son comprobaciones de alta prioridad. La más importante es el manejo de emergencia: si el interruptor de emergencia del Arduino está activado, el sistema ignora todo lo demás, entra en el estado `EMERGENCIA_ACTIVA`, envía comandos al Arduino para hacer parpadear un LED rojo, abre o cierra la puerta según sea necesario, y comienza a grabar video con la cámara como evidencia de seguridad. Cuando el interruptor se desactiva, finaliza la grabación y el sistema intenta volver a un estado seguro.

Después de la emergencia, el bucle entra en una serie de bloques `if/elif` que corresponden a cada estado del sistema.

- Si el estado es `EstadoSistema.REPOSO`, el sistema está inactivo esperando. Monitorea el sensor de presencia SP1. Si detecta a una persona (`0 < dist_sp1 < constants.UMBRAL_DETECCION_SP1_CM`), el sistema cambia al primer estado de validación requerido por el protocolo activo (por ejemplo, `ESPERANDO_VALIDACION_RFID`).
- En los estados de validación como `ESPERANDO_VALIDACION_RFID`, `ESPERANDO_VALIDACION_QR_REAL` o `ESPERANDO_VALIDACION_FACIAL`, la lógica es similar: tiene un temporizador de timeout. Si el usuario no presenta la credencial a tiempo, el acceso se deniega. Si se presenta una credencial, se usan los módulos `db_manager` y `validation_logic` para verificar si el usuario existe, está bloqueado, está dentro de su horario, etc. Para el reconocimiento facial, activa la cámara, busca rostros y los compara con los "encodings" almacenados. Si una validación es exitosa, puede pasar a la siguiente (en un protocolo de múltiples pasos) o directamente al estado `ABRIENDO_PUERTA`. Si falla, pasa a un estado de `ACCESO_DENEGADO_TEMPORAL`.
- En el estado `ABRIENDO_PUERTA`, se envía el comando `arduino_comms.enviar_comando_a_arduino("ABRIR_PUERTA")` y se inicia un temporizador.
- En el estado `PERSONA_CRUZANDO`, el sistema monitorea los dos sensores de presencia (SP1 y SP2) para asegurarse de que la persona cruza la puerta en la dirección correcta y que no hay anomalías (como ambas personas bloqueadas a la vez, lo que activaría el estado `ALERTA_ERROR_CRUCE`).
- Una vez que la persona ha cruzado (detectado por la secuencia de activación y desactivación de los sensores), el sistema pasa a `CERRANDO_PUERTA`, envía el comando de cierre al Arduino, y finalmente, tras una breve pausa, vuelve a `EstadoSistema.REPOSO` para esperar al siguiente usuario.

El bloque final `if __name__ == "__main__":` permite ejecutar este archivo de forma independiente para pruebas, creando una instancia de la GUI y arrancando la lógica, lo que facilita la depuración del cerebro del sistema sin necesidad de ejecutar el proyecto completo.