Para empezar, importa las librerías necesarias. La línea `import serial` trae la librería PySerial para la comunicación por puerto serie. Luego, `import serial.tools.list_ports` se usa para encontrar los puertos COM disponibles. Con `import time` se manejan pausas y tiempos de espera, mientras que `import threading` es crucial para ejecutar la comunicación en un hilo separado y no congelar la aplicación. Finalmente, con `from tkinter import messagebox` se pueden mostrar ventanas de error al usuario.

A continuación, se definen unas variables globales que mantendrán el estado de la comunicación. La variable `arduino_serial = None` guardará el objeto de la conexión una vez que se establezca. La variable `arduino_conectado = False` es una bandera que nos dirá en todo momento si la conexión está activa o no. Para evitar problemas de concurrencia entre hilos, se crea un cerrojo con `lock_datos_hardware = threading.Lock()`.

Se define un diccionario llamado `datos_hardware` que servirá como un almacén central para todos los datos recibidos del hardware. Se inicializa con valores por defecto como `"sp1_distancia": 999.0` para la distancia del sensor 1, `"s1_estado": 1` para el estado del sensor de carrera (donde 1 significa no presionado y 0 significa presionado), `"e_estado": 1` para el botón de emergencia, y `"rfid_uid": "NADA"` para la última tarjeta leída. También incluye `"ultimo_rfid_procesado_para_acceso": "NADA"` que es una variable de control para la lógica de la aplicación.

Se preparan las variables para el hilo de escucha: `hilo_listener_arduino = None` guardará el objeto del hilo y `hilo_listener_arduino_activo = False` será la bandera para mantenerlo corriendo o detenerlo. La variable `app_gui_ref = None` se usará para guardar una referencia a la interfaz gráfica y poder actualizarla.

Se establecen dos constantes: `VELOCIDAD_ARDUINO = 115200` define la velocidad de comunicación, y `TIMEOUT_SERIAL = 1` el tiempo de espera en las lecturas.

La función `asignar_app_gui_referencia(gui_instance)` es una función simple que permite a la aplicación principal pasar una referencia de sí misma para que este módulo pueda interactuar con la GUI.

La función principal de conexión es `conectar_a_arduino(puerto_seleccionado_str)`. Primero, si ya existe una conexión, la cierra usando `arduino_serial.close()`. Luego, verifica que el puerto seleccionado sea válido. La conexión se intenta dentro de un bloque `try...except` para capturar cualquier error. La línea clave es `arduino_serial = serial.Serial(...)` que abre el puerto. Después, hace una pausa de 2 segundos con `time.sleep(2)` para dar tiempo a que el Arduino se reinicie. Para confirmar que el Arduino está listo, entra en un bucle `while` que dura 5 segundos como máximo, esperando recibir el mensaje "ARDUINO_LISTO". Si lo recibe, establece `arduino_conectado = True` y devuelve `True`. Si no, cierra el puerto y devuelve `False`.

Para enviar datos al Arduino, se usa la función `enviar_comando_a_arduino(comando_str)`. Esta función revisa si la conexión está activa y, de ser así, formatea el mensaje con el prefijo "COMANDO:" y un salto de línea final, como en `mensaje_completo = f"COMANDO:{comando_str}\n"`. Finalmente, lo envía usando `arduino_serial.write(mensaje_completo.encode('utf-8'))`, convirtiendo el texto a bytes.

La función más importante es `escuchar_datos_arduino()`, diseñada para correr en su propio hilo. Se ejecuta en un bucle infinito controlado por la bandera `while hilo_listener_arduino_activo:`. Dentro del bucle, primero comprueba si hay datos disponibles con `if arduino_serial.in_waiting > 0:`. Si los hay, lee todos los bytes con `bytes_recibidos = arduino_serial.read(...)` y los decodifica a texto. Como los datos pueden llegar en trozos, los va acumulando y los procesa línea por línea. Si una línea comienza con "DATOS;", como se comprueba con `if linea_str.startswith("DATOS;")`, sabe que es un paquete de sensores. Entonces, divide la línea por el carácter ';' con `partes = linea_str.split(';')`. Para actualizar el diccionario `datos_hardware` de forma segura, usa un cerrojo con la instrucción `with lock_datos_hardware:`. Dentro de este bloque seguro, extrae cada valor (distancias, estados de los sensores, etc.) y lo guarda en el diccionario, convirtiéndolo al tipo de dato correcto (float o int). Si no hay datos que leer, hace una pequeña pausa con `time.sleep(0.01)` para no consumir CPU innecesariamente. Todo este proceso está envuelto en un `try...except` para manejar desconexiones inesperadas.

Si ocurre una desconexión (por ejemplo, se desenchufa el cable), se llama a la función `desconectar_arduino_emergencia()`. Esta función cierra el puerto con `arduino_serial.close()`, pone la bandera `arduino_conectado` en `False`, y actualiza la interfaz gráfica para que el usuario vea el estado de "Desconectado". Para actualizar la GUI de forma segura desde el hilo, usa el método `app_gui_ref.after(0, ...)`.

Finalmente, hay dos funciones de ayuda. La función `get_datos_hardware_copia()` permite a otras partes del programa obtener una copia segura de los datos de los sensores. Usa `with lock_datos_hardware:` para garantizar que no se lean datos a medio escribir, y devuelve una copia con `return datos_hardware.copy()`. La función `is_arduino_conectado()` simplemente devuelve el valor de la bandera `arduino_conectado`.