Diseño de Sistemas de Internet de las Cosas

Tarea 2.

Profesor: Luciano Radrigan F.

Cada grupo tendrá una Raspberry pi y dos microcontroladores ESP32, con estos elementos debe realizar las siguientes tareas:

- Crear una aplicación de escritorio con Python QT que permita realizar una configuración inicial con los ESP32 y las gráficas de los datos recepcionados en operación. Para realizar esta tarea el modo de operación de esp32 debe estar en status 0 (ver tabla 1). En este modo la comunicación entre el ESP32 y la Raspberry debe ser a través de una conexión Bluetooth punto a punto. Las figuras 1 y 2 muestran una propuesta de interfaz.

Tabla 1- Modos de operación ESP32.

status	Descripción	Compatibilidad con ID_protocol
0	Configuración por Bluetooth	1
20	Configuración vía TCP en BD	1
21	Conexión TCP continua	1-2-3-4-5
22	Conexión TCP discontinua	1-2-3-4-5
23	Conexión UDP	1-2-3-4-5
30	BLE continua	1-2-3-4
31	BLE discontinua	1-2-3-4

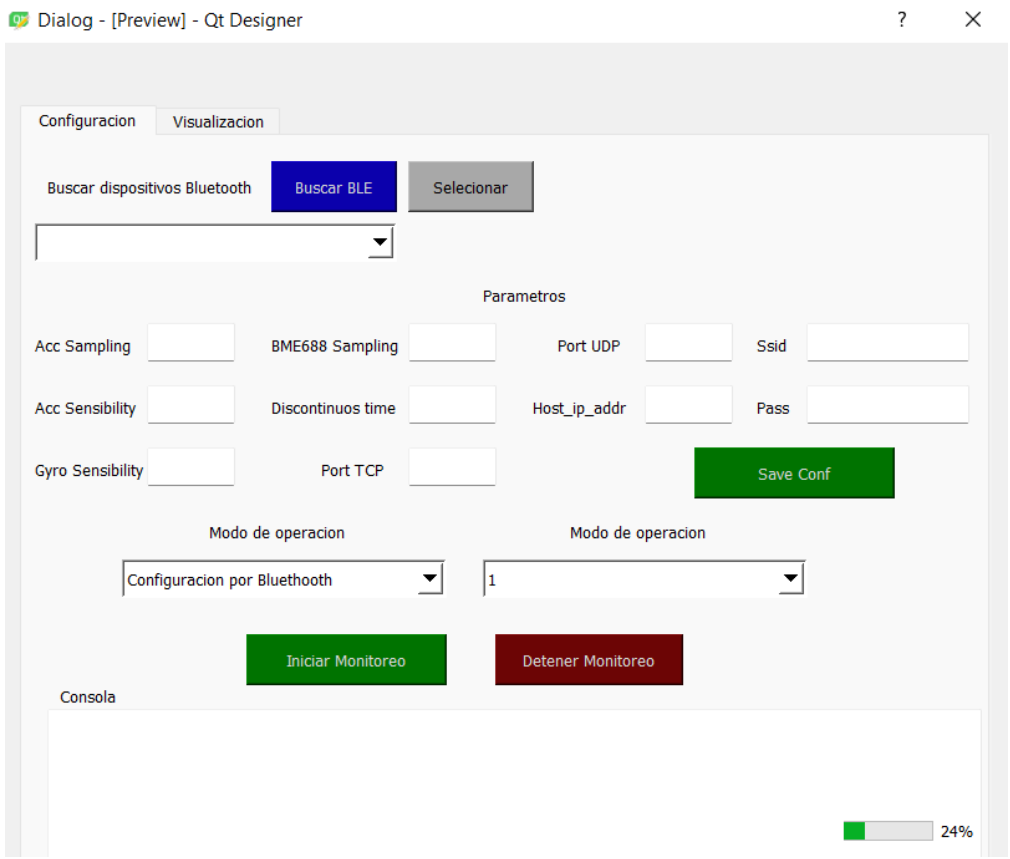

Figura 1. Ejemplo de pantalla de configuración.

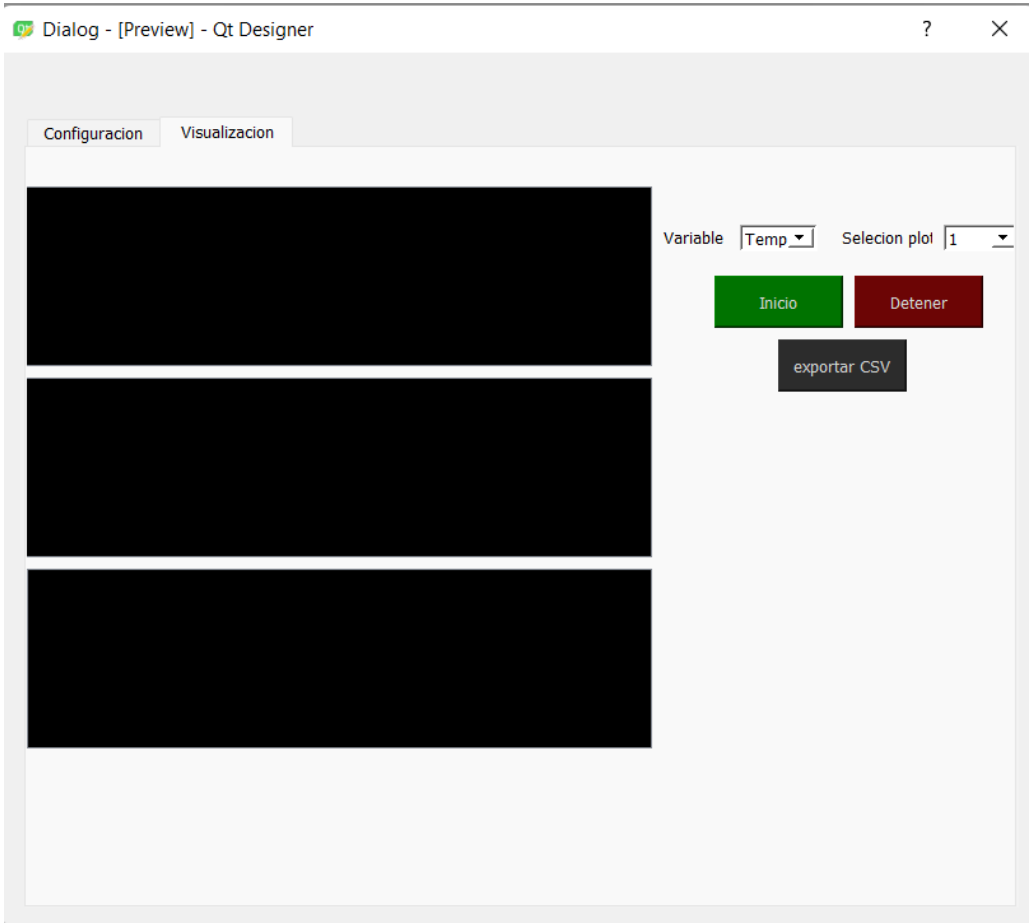

Figura 2. Ejemplo de pantalla de gráfico.

Descripción de Status:

Configuración por Bluetooth (status=0): El esp32 se configura con los datos seteados por la interfaz QT desde la raspberry a través de una comunicación bluetooth punto a punto (se recomienda utilizar la librería pygatt en la raspberry). Los parámetros para configurar deben guardarse en la Raspberry en una tabla de la base de datos Mongo_db (llamada *config*) y en el esp32 en memoria no volátil. Los parámetros para configurar son los siguientes:

Tabla 2- Parámetros de configuración

Parámetros de configuración	
int8_t	Status
int8	ID_Protocol
int32_t	BMI270_sampling (valores posibles:10,100,400,1000)
int32_t	BMI270_Acc_Sensibility (valores posibles:2,4,8,16)
int32_t	BMI270_Gyro_Sensibility (valores posibles:200,250,500)
int32_t	BME688_Sampling (1,2,3,4)
int32_t	Discontinuous_Time
int32_t	Port_TCP
int32_t	Port_UDP
int32_t	Host_Ip_Addr
Char	Ssid
Char	Pass

Configuración vía TCP en BD (status = 20): El *Esp32* tendrá un Cliente TCP y la Raspberry un Servidor TCP. (el "*Ssid", "Pass" y "Port_TCP"* se toman de los valores configurados por la interfaz). En este modo el esp32 puede actualizar cualquiera de los valores de la tabla "*Parámetros de configuración" a* través de una conexión TCP. Los valores se adquieren de la tabla *config* de la DB.

Conexión TCP continua (status= 21): El *Esp32* tendrá un Cliente TCP y la Raspberry un Servidor TCP. (el "*Ssid", "Pass"* y "*Port_TCP"* se toman de los valores configurados por la interfaz). Según el valor de "*ID_Protocol"* es el paquete de datos que se transferirá. (protocolos de datos se observan en la Tabla 3). El esp32 deberá enviar este paquete de forma continua hasta que desde la raspberry (desde la interfaz gráfica).

Conexión TCP discontinua (status= 22): El *Esp32* tendrá un Cliente TCP y la Raspberry un Servidor TCP. (el "*Ssid", "Pass" y "Port_TCP"* se toman de los valores configurados por la interfaz). Según el valor de "*ID_Protocol"* es el paquete de datos que se transferirá. (protocolos de datos se observan en la Tabla 3). Luego de enviar los datos según el valor de "*Discontinuous_Time*" el esp32 entrara por ese tiempo en modo "*Deep_sleep"*. Se recomienda que el "*Discontinuous_Time"* tenga como unidad minutos y que su valor mínimo sea 1. Este modo se deberá poder detener desde la raspberry se detenga el envió (desde la interfaz gráfica).

Conexión UDP (status=23): El Esp32 tendrá un Cliente UDP y la Raspberry un Servidor UDP. (el "*Ssid", "Pass"* y "*Port_UDP* "se toman de los valores configurados por la interfaz). Según el valor de "*ID_Protocol"* es el paquete de datos que se transferirá. (protocolos de datos se observan en la Tabla 3). El esp32 deberá enviar este paquete de forma continua hasta que desde la raspberry se detenga el envío (desde la interfaz gráfica).

BLE continua (status=30): El *Esp32* tendrá un Server BLE y la Raspberry un Servidor Cliente. (Se recomienda utilizar la librería pygatt). Según el valor de "*ID_Protocol"* es el paquete de datos que se transferirá. (protocolos de datos se observan en la Tabla 3 y este modo no es compatible con "*ID_protocol*"=5). El esp32 deberá enviar este paquete de forma continua hasta que desde la raspberry se detenga el envió (desde la interfaz gráfica).

BLE discontinua (status=31): El *Esp32* tendrá un Server BLE y la Raspberry un Servidor Cliente. (Se recomienda utilizar la librería pygatt). Según el valor de "*ID_Protocol"* es el paquete de datos que se transferirá. (protocolos de datos se observan en la Tabla 3 y este modo no es compatible con "*ID_protocol*"=5). Luego de enviar los datos según el valor de "*Discontinuous_Time*" el esp32 entrara por ese tiempo en modo "*Deep_sleep"*. Se recomienda que el "*Discontinuous_Time"* tenga como unidad minutos y que su valor mínimo sea 1. Este modo se deberá poder detener desde la raspberry se detenga el envió (desde la interfaz gráfica).

- Los datos enviados desde los microcontroladores deberán ser programados, para esto se deberán implementar funciones que emulen el funcionamiento de los sensores, dentro de la que destacan:
    - Aceloremeter_sensor: Genera un vector de 1600 datos por eje (x,y,z). Los datos son flotantes dados por la siguiente formula
        - Acc_x= Valores aletarorios entre el -8000 y 8000
        - Acc_y= Valores aletarorios entre el -8000 y 8000
        - Acc_z= Valores aletarorios entre el -8000 y 8000
    - THPC_sensor:
        - Temp = Valor aleatorio entre 5 a 30
        - hum = Valor aleatorio entre 30 a 80.
        - pres = Valor aleatorio entre 1000 y 1200
        - Co = Valor aleatorio entre 30 a 200
    - Batt_sensor:
        - Value = Valor aleatorio entre 1 y 100
    - Aceloremeter_kpi
        - RMS = ((Amp_x)^2+ (Amp_y)^2+(Amp_z)^2)^1/2
        - Amp_x = Valor aleatorio entre 0.0059 y 0.12
        - Frec_x = Valor aleatorio entre 29.0 y 31.0
        - Amp_y = Valor aleatorio entre 0.0041 y 0.11
        - Frec_y = Valor aleatorio entre 59.0 y 61.0
        - Amp_z = Valor aleatorio entre 0.008 y 0.15
        - Frec_z = Valor aleatorio entre 89.0 y 91.0

Tabla 3- Id_protocols

				1 bytes	4 bytes	1 bytes	4 bytes	1 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes
	ID Protocol	Status	leng msg	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10	Data 11	Data 12	Data 13
Cloud	1		6	Batt_level	Timestamp											
	2		16	Batt_level	Timestamp	Temp	Press	Hum	Co							
	3		20	Batt_level	Timestamp	Temp	Press	Hum	Co	RMS						
	4		44	Batt_level	Timestamp	Temp	Press	Hum	Co	RMS	Amp x	Frec x	Amp y	Frec y	Amp z	Frec z
				1 bytes	4 bytes	1 bytes	4 bytes	1 bytes	4 bytes							
										Data 8	Data 9	Data 10	Data 11	Data 12	Data 13	Data 14
	5			Batt_level	Timestamp	Temp	Press	Hum	Co	Acc_X	Acc_y	Acc z				
				1 bytes	4 bytes	1 bytes	4 bytes	1 bytes	4 bytes	Arreglo de 2000 datos del tipo Int16_t	Arreglo de 2000 datos del tipo Int16_t	Arreglo de 2000 datos del tipo Int16_t				

*Se adjunta base de datos propuesta. (desde la interfaz se guardan las opciones de configuración en la tabla *configuration*, y desde ahí el microcontrolador puede ir reconfigurándose). Solo para términos de depuración la variable *Conf_peripheral* se compone como la suma de string de los valores de configuración de *Acc_samplig, Acc_sensibility Gyro_sensibility, BME688_sampling y discontinuos_time.*