UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO DE FINAL DE GRADO

Doble Grado en Ingeniería Electrónica industrial mecánica y automática e Ingeniería mecánica

DISEÑO Y FABRICACIÓN DE LA ESTRUCTURA Y DEL SISTEMA DE ADQUISICIÓN DE DATOS DE UN CUADRICÓPTERO DE APLICACIÓN GENERAL

Anexos

Autor/a: Morales Vélez, Alejandro
Director/a: Manzanares Brotons, Manuel Andrés
Codirector/a: Travieso Rodríguez, José Antonio

Convocatoria: septiembre 2023

Índice

1. Código Arduino

```
/* ------------------------------------------------------------------------
-----------

Autor:  Alejandro Morales Vélez
Programa: Adquisición de datos dron con transmisión BT para visualización
mediante
          módulo HC-06 y App Inventor
Fecha: septiembre 2023

------------------------------------------------------------------------
-----------*/
//librerias necesarias para DHT
#include <DHT.h>
#include <SoftwareSerial.h>

//librerias necesarias para BMP
#include <Wire.h>   // para comunicación I2C
#include <Adafruit_Sensor.h>  // para sensor BMP280
#include <Adafruit_BMP280.h> //para sensor BMP280

// librerias necesarias para MPU6050
#include <Simple_MPU6050.h>

// definición del pineado de DHT
#define DHTPIN 1 //pin al que se va conectar el pin de datos del sensor
#define DHTTYPE DHT22 //definición del tipo de sensosr DHT, en este caso un
DHT22 (AM2302)
DHT dht(DHTPIN, DHTTYPE);

//definicíon para MPU6050
#define MPU6050_Address 0x68  // direccion I2C con AD0 en LOW o sin
conexion, la que se utilizara en este caso, en caso de conexión seria la
0x69
#define Six_Axis_Quaternions 6 //por defecto ya esta en 6, podríamos
prescindir de esta linea de codigo
Simple_MPU6050 mpu(Six_Axis_Quaternions); // crea objeto con nombre mpu

//ofsets para MPU6050, en caso de ya tenerlos calibrados esta habilitado,
sino comentar estas lineas, solo al inicio comentadas
#define OFFSETS  4294966994,     4294965982,   1260
,      53,     4294967261,       4294967287  //
// spamtimer funcion para generar un delay a la hora de mostrar valores por
el monitor serie pero sin bloquear el flujo del programa
```

```
#define spamtimer(t) for (static uint32_t SpamTimer; (uint32_t)(millis() -
SpamTimer) >= (t); SpamTimer = millis())

//definición del pin de entrada analogico del MQ135
#define MQ135_sensor 4

// se define comunicación serial adicional a RX TX, para comunicación con el
APM
SoftwareSerial ardupilotSerial(18, 17);//RX TX

uint8_t SpamDelay=100;

// definición de los pines de comunicación Soft serial para el módulo BT
SoftwareSerial BT_Serial(12, 11);//RX TX

// definición de String para envió de datos
String datos_BT =""; // Se inicilaiza totalmente vacío

//variables para DHT
int T;
int K;
int RH;

//variables para almacenamiento de datos del BMP
  //float Temp;   // variable para almacenar valor de temperatura
  int Pressure;     // variable para almacenar valor de presion atmosferica
  int Pressure_ref; // se podrían indicar todas las variables del mismo tipo
separadas por comas en la misma linea de codigo
  int Altitud;

// Variables para MPU
    int yaw;
    int pitch;
    int roll;
    int accx;
    int accy;
    int accz;

 //variable para MQ135
    int ADC_MQ135;

//creacion de un objeto con el nombre bmp
  Adafruit_BMP280 bmp;

// Mavlink variables
```

```
unsigned long previousMillisMAVLink = 0;     // gurda el ultimo momento en
que MAVLink ha trasmitido o escuchado
unsigned long next_interval_MAVLink = 1000;  // nuevo intervalo de conteo
const int num_hbs = 60;                      // numero de hbs que pasan
antes de la activación de la comunicación
int num_hbs_pasados = num_hbs;

//variables para control comunicación serial con APM
// int request = 0;
// int receive = 0;

//declaracion de las variables de posición global de APM
  uint32_t time_boot_ms; ///< Timestamp (milliseconds since system boot)
  //status
  float Voltaje;
  float Amp;
  //global position
  int32_t lat; ///< Latitude, expressed as * 1E7
  int32_t lon; ///< Longitude, expressed as * 1E7
  int16_t vx; ///< Ground X Speed (Latitude), expressed as m/s * 100
  int16_t vy; ///< Ground Y Speed (Longitude), expressed as m/s * 100
  int16_t vz; ///< Ground Z Speed (Altitude), expressed as m/s * 100
  uint16_t hdg; ///< Compass heading in degrees * 100, 0.0..359.99 degrees.
If unknown, set to: 65535


void setup()
{
  BT_Serial.begin(9600);//inicialización de la comunicación BT y serial
  Serial.begin(115200); //se inicializa el puerto de comunicacion serie para
la visualización de los datos mediante serial monitor en el PC
  ardupilotSerial.begin(57600); //inicialización de la comunicación serie
con APM
  dht.begin(); //inicialización para la medición de temperatura
  bmp.begin();//inicialización medición bmp

//guardado de presión de referencia para calculo posterior de altura
  Pressure_ref = bmp.readPressure()/100;

//para MPU6050
  uint8_t val;

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE  // establece la velocidad
del bus I2C, ya que se esta trabajando con la libreria I2Cdev y no wire, por
tanto activa el bus I2C a 400 Khz
  Wire.begin();
```

```
  Wire.setClock(400000);
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

#ifdef OFFSETS                // solo se ejecuta en la compilación, si
exiten se carga sino no, en este caso, si existen OFFSETS
  Serial.println(F("Offsets predefinidos"));
  mpu.SetAddress(MPU6050_Address).load_DMP_Image(OFFSETS);  //
inicialización del sensor así como la carga de los offsets predefinidos
#else                     // en caso de no existir offsets
  Serial.println(F(" No se establecieron Offsets.\n"
                   " Colocar el sensor en un superficie plana y esperar unos
segundos\n"
                   " Indicar los nuevos Offsets en #define OFFSETS\n"
                   " \t\tPresionar cualquier tecla y ENTER"));

  while (Serial.available() && Serial.read());    // lectura de monitor
serie
  while (!Serial.available());        // detiene el programa en caso de no
existir comunicación serial
  while (Serial.available() && Serial.read());    // si existe comunicación
serial y se esta leyendo datos
  mpu.SetAddress(MPU6050_Address).CalibrateMPU().load_DMP_Image();  //
inicializacion comunicación I2C con la dirección definida
#endif

  mpu.on_FIFO(all_MPU6050_data);    // FIFO es la pequeña memoria que tiene
el MPU6050 (first input first output), una vez esta llena llama a la función
mostrar_valores

 Serial.println ("Inicializando adquisición "); //

}


void loop() // se irán llamando las diferentes funciones para adquisición y
posterior envió por BT
{

DHT_22();
BMP_280();
mpu.dmp_read_fifo();
MQ135();
APM_status();
Mav_Request_Data();
```

```
Send_data();
delay(2000);
}

void DHT_22 ()
{
  T = dht.readTemperature(); //muestra los valores con decimales
  K = (dht.readTemperature()+273.15); //muestra los valores con decimales
  // float F = dht.readTemperature(true);
  RH = dht.readHumidity();  // declaracion de las variables de las que se
quiere obtener la magnitud, para ello y dado qu se trata de magnitudes con
valores decimales (el sensor permite esta sensiblidad) se declara una
variable float

// solo necesario para visualizar desde monitor serial
  // Serial.print( "Temperatura (ºC): ");
  // Serial.println(T);
  // Serial.print( "Temperatura (ºK): ");
  // Serial.println(K);
  // Serial.print( "Humedad relativa (%): ");
  // Serial.println(RH);
  // delay(2000);
   //tasa de refresco minima permitida por el sensor 2 segundos, para esta
aplicación suficeinte una tasa de refresco cada 4-6 segundos

}

void BMP_280()
{

  Pressure = bmp.readPressure()/100;    // se almacena el valor de
temperatura recibido por el sensor dividido entre 100 para expresarlo en
mbar o hpa, ya que el valor original viene en Pa
  Altitud = bmp.readAltitude(Pressure_ref); // altitud es una lectura de
presión respecto de la presión de referencia, de esta manera se obtiene la
altiutd del UAV

  // Serial.print("Presion: ");     // muestra texto
  // Serial.print(Pressure);      // muestra valor de la variable
  // Serial.print(" hPa o mbar ");      // muestra texto hPa indicando
hectopascales

  // Serial.print("Altitud: ");   // muestra texto
  // Serial.print(Altitud);     // queremos saber la altitud del dron, con
lo que indica presión de referencia cuando se encuentra en tierra, como es
un valor de referencia se indica con la declaración de variables
```

```
  // Serial.println(" m ");       // muestra letra C indicando grados
centigrados
  delay(1000);
}

void all_MPU6050_data (int16_t *gyro, int16_t *accel, int32_t *quat) {

  Quaternion q;         // variable necesaria para recibir info del sensor y
realizar calculos posteriores
  VectorFloat gravity;        // variable necesaria para recibir info del
sensor y realizar calculos posteriores
    // array para almacenar valores convertidos a grados de yaw, pitch, roll
  float ypr[3] = { 0, 0, 0 };  //  para almacenar valores de yaw, pitch,
roll; inicializado en 0
  float xyz[3] = { 0, 0, 0 }; // para almacenar valores como referencia a
los ejes

  spamtimer(SpamDelay) { // aplica un delay que no detiene el flujo del
programa, solo el que tiene contenido dentro {}

    mpu.GetQuaternion(&q, quat);    // funcion para obtener valor para
calculo de los angulos de yaw, pitch y roll
    mpu.GetGravity(&gravity, &q);   // funcion para obtener valor para
calculo de los angulos de yaw, pitch y roll
    mpu.GetYawPitchRoll(ypr, &q, &gravity); // con esta fucnión se almacena
el valor de q y gravity en la variable ypr anteriormente creada para la
obtención de valores de yaw, ptich, roll
    mpu.ConvertToDegrees(ypr, xyz);   // se convierte a grados en radianes a
grados sexagesimales los valores de ypr y los almacena en el array xyz

    yaw = xyz[0];
    pitch = xyz[1];
    roll = xyz[2];
    accx = accel[0]/16384.0; //división para cambio de unidades de LSB a
m/s^2
    accy = accel[1]/16384.0; //accel ya definido en libreria no neceario
definir
    accz = accel[2]/16384.0;

  }
}

void MQ135()
{
```

```
    ADC_MQ135 = analogRead(5); // declaración de un int (números enteros
positivos o negativos sin decimales) igual a la lectura recibida desde la
salida analógica del MQ135
}


void APM_status()
{
  // MAVLink
  //cabecera por defecto para MCU
  int sysid = 1;                   ///< ID 20 de la aeronave. 1 PX, 255
ground station
  int compid = 158;                ///< Componente que envía el message
  int type = MAV_TYPE_FIXED_WING;    ///< Tipo de sistema

  // Define el tipo de sistema
  uint8_t system_type = MAV_TYPE_GENERIC;
  uint8_t autopilot_type = MAV_AUTOPILOT_INVALID;

  uint8_t system_mode = MAV_MODE_PREFLIGHT; ///< Booting up
  uint32_t custom_mode = 0;                 ///< Custom mode, can be defined
by user/adopter
  uint8_t system_state = MAV_STATE_STANDBY; ///< System ready for flight

  // Inicialización de los buffers necesarios
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  // Empaquetado del mensaje
  //mavlink_msg_heartbeat_pack(sysid,compid, &msg, type, autopilot_type,
system_mode, custom_mode, system_state);
  mavlink_msg_heartbeat_pack(1, 0, &msg, type, autopilot_type, system_mode,
custom_mode, system_state);

  // Se copia el mensaje en el buffer
  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);

  // Se envía el mensaje por puerto serial bajo el contador en cuestión
  unsigned long currentMillisMAVLink = millis(); // función para conteo de
los microsecundos transcurridos
  if (currentMillisMAVLink - previousMillisMAVLink >= next_interval_MAVLink)
  {
    previousMillisMAVLink = currentMillisMAVLink;
    ardupilotSerial.write(buf, len); //se envía el mensaje hacia APM
```

```
    num_hbs_pasados++; //se incrementa el valor de hbs para inicializar la
recibida de datos
    if (num_hbs_pasados >= num_hbs)
    {
      comm_status_receive();
    }

  }
}

void comm_status_receive()
{
  while (ardupilotSerial.available() > 0)
  { //mientras se encuentre disponible la comunicación serial con ardupilot
    uint8_t c = ardupilotSerial.read(); //lectura del puerto serie

    // Se trata de parsear el nuevo mensaje para la interpretación de los
datos recibidos
    if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status))
    {

      //manipulación del mensaje para obtención de datos
      switch (msg.msgid)
      {
        case MAVLINK_MSG_ID_SYS_STATUS:  // #1: SYS_STATUS
          {
//              // Message decoding: PRIMITIVE  información directamente
obtenida de las librerias mavlink
//                   mavlink_msg_sys_status_decode(const mavlink_message_t*
msg, mavlink_sys_status_t* sys_status)

            mavlink_sys_status_t sys_status;
            mavlink_msg_sys_status_decode(&msg, &sys_status); // se
decodifica el mensaje
            // de aqui se obtendrán los datos de batería y amperaje
            Voltaje = sys_status.voltage_battery;
            Amp = sys_status.current_battery;
          }
          break;
      }
    }
  }
}

void Mav_Request_Data()
{
```

```
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN]; //variable equivalente a 1 byte (0-
255), definida en la libreria de mavlink--> mavlink_types.h visitarla para
ver el contenido exacto (MAVLINK_MAX_PAYLOAD_LEN +
MAVLINK_NUM_NON_PAYLOAD_BYTES) ///< Maximum packet length
  //mavlink_status_t status;
  mavlink_msg_global_position_int_pack(1, 200, &msg, millis(), lat, lon,
alt, relative_alt, vx, vy, vz, hdg);
  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
  unsigned long currentMillisMAVLink = millis(); // función para conteo de
los microsecundos transcurridos
  if (currentMillisMAVLink - previousMillisMAVLink >= next_interval_MAVLink)
  {
    previousMillisMAVLink = currentMillisMAVLink;
    ardupilotSerial.write(buf, len); //se envía el mensaje hacia APM

    num_hbs_pasados++; //se incrementa el valor de hbs para inicializar la
recibida de datos
    if (num_hbs_pasados >= num_hbs)
    {
    comm_global_position_recive();
    }

  }
}


void comm_global_position_recive()
{
 while (ardupilotSerial.available() > 0)
  { //mientras se encuentre disponible la comunicación serial con ardupilot
    uint8_t c = ardupilotSerial.read(); //lectura del puerto serie

 // Try to get a new message, se realiza el parseo del mensaje para la
obtención de los datos requeridos .
    if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status))
    {
      //Manipulación del mensaje para la obtención de los datos requeridos
      switch (msg.msgid)
      {
        case MAVLINK_MSG_ID_GLOBAL_POSITION_INT:  // #33: SYS_STATUS
          {
            /* Message decoding: PRIMITIVE información directamente obtenida
desde la libreria mavlink
```

```
                mavlink_msg_global_position_int_decode(const
mavlink_message_t* msg, mavlink_global_position_int_t*
global_position_int)*/
            //mavlink_message_t* msg;
            mavlink_global_position_int_t global_position_int;
            mavlink_msg_global_position_int_decode( &msg,
&global_position_int);

            lat = global_position_int.lat; //latitud geografica en 1E7
            lon = global_position_int.lon; //longitud geografica en 1E7
            vx = global_position_int.vx;
            vy = global_position_int.vy;
            vz = global_position_int.vz;
            hdg = global_position_int.hdg/100; //orientación brujula
          }
          break;
      }
    }
  }
}


void Send_data()
{
 //envío de datos por HC-06 a teléfono móvil para visualización
   BT_Serial.print(T);
   BT_Serial.print(',');
   BT_Serial.print(K);
   BT_Serial.print(',');
   BT_Serial.print(RH);
   BT_Serial.print(',');
   BT_Serial.print(Pressure);
   BT_Serial.print(',');
   BT_Serial.print(Altitud);
   BT_Serial.print(',');
   BT_Serial.print(yaw);
   BT_Serial.print(',');
   BT_Serial.print(pitch);
   BT_Serial.print(',');
   BT_Serial.print(roll);
   BT_Serial.print(',');
   BT_Serial.print(accx);
   BT_Serial.print(',');
   BT_Serial.print(accy);
   BT_Serial.print(',');
   BT_Serial.print(accz);
```

```
   BT_Serial.print(',');
   BT_Serial.print(ADC_MQ135);
   BT_Serial.print(',');
   BT_Serial.print(Voltaje);
   BT_Serial.print(',');
   BT_Serial.print(Amp);
   BT_Serial.print(',');
   BT_Serial.print(lat);
   BT_Serial.print(',');
   BT_Serial.print(lon);
   BT_Serial.print(',');
   BT_Serial.print(vx);
   BT_Serial.print(',');
   BT_Serial.print(vy);
   BT_Serial.print(',');
   BT_Serial.print(vz);
   BT_Serial.print(',');
   BT_Serial.print(hdg);
   BT_Serial.println(); //indica que el mensaje o la transmisión ha
finalizado

}
```

2. Interfaz MIT App Inventor

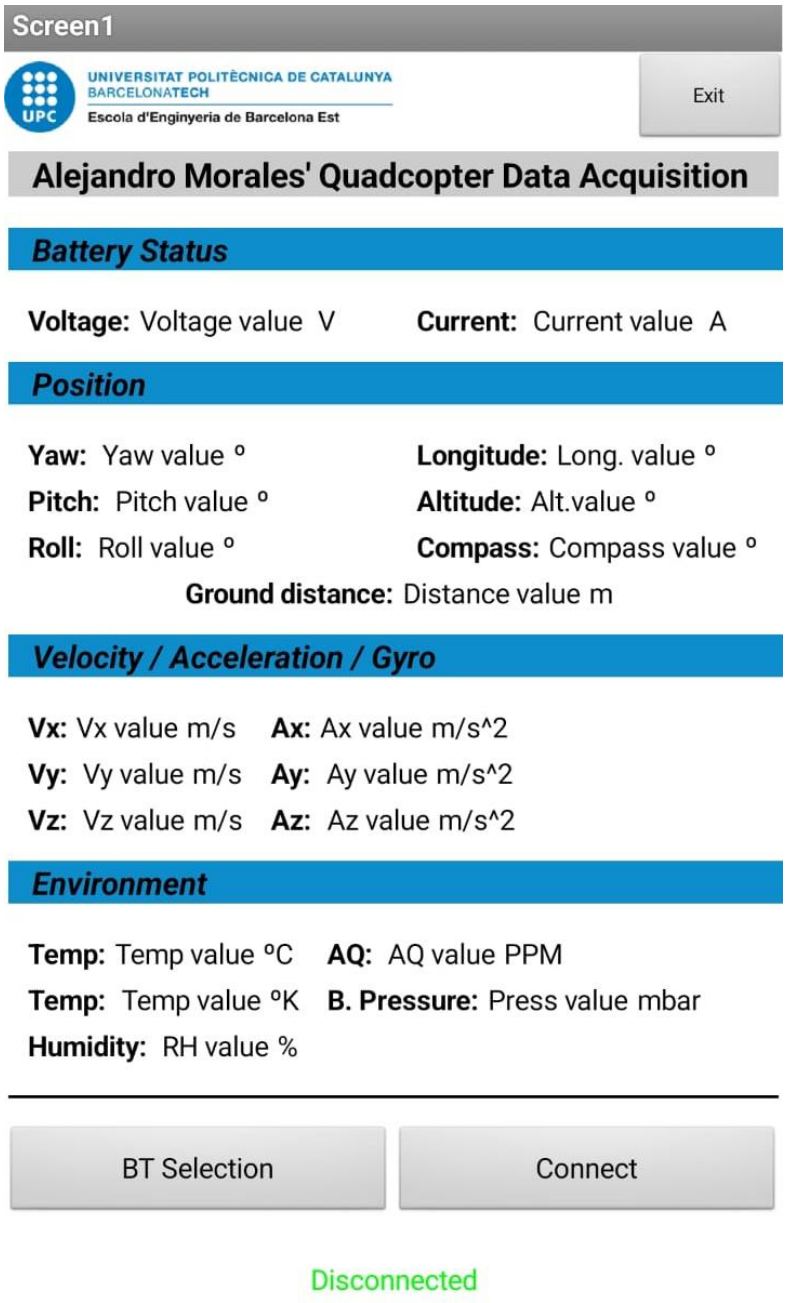
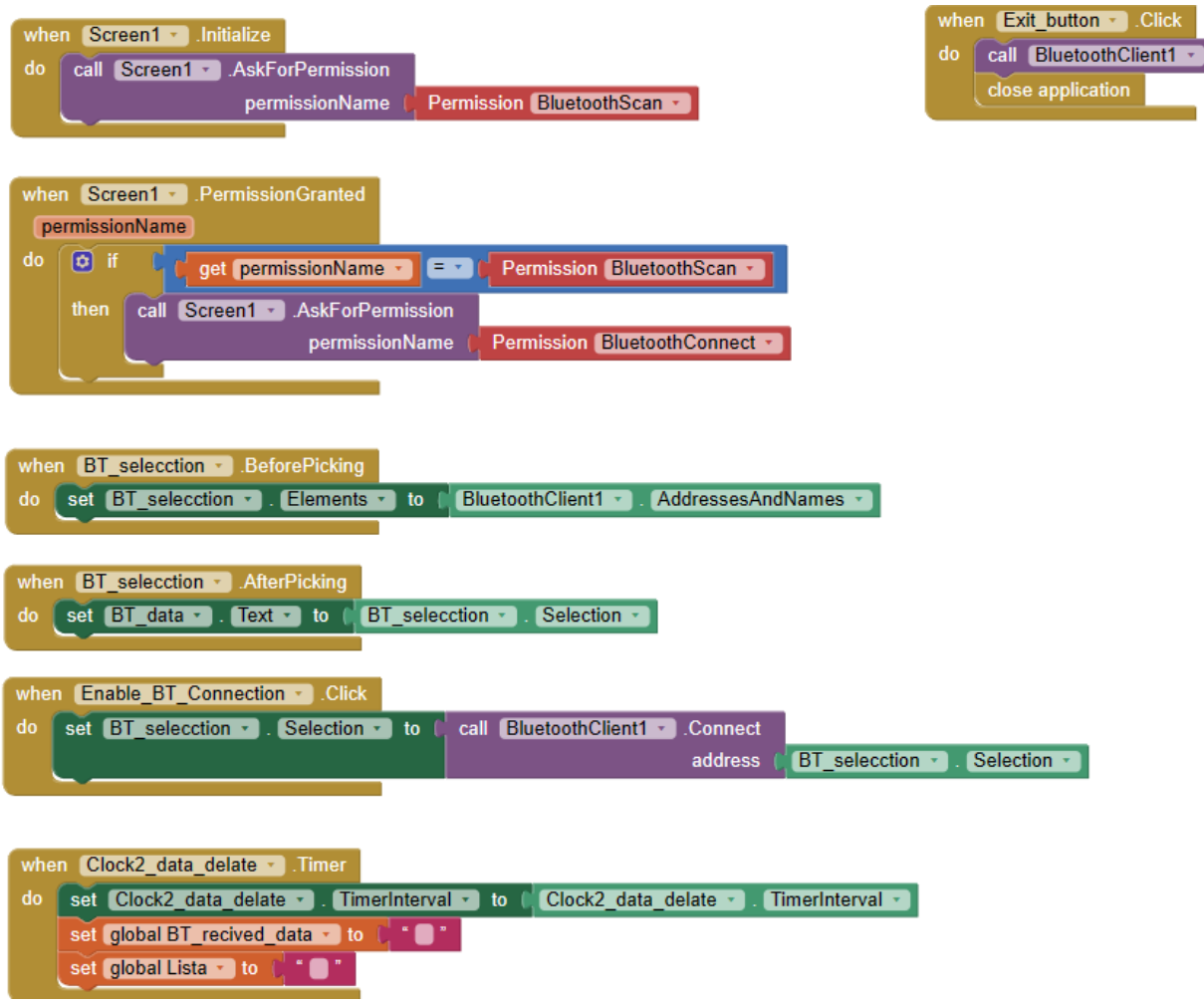


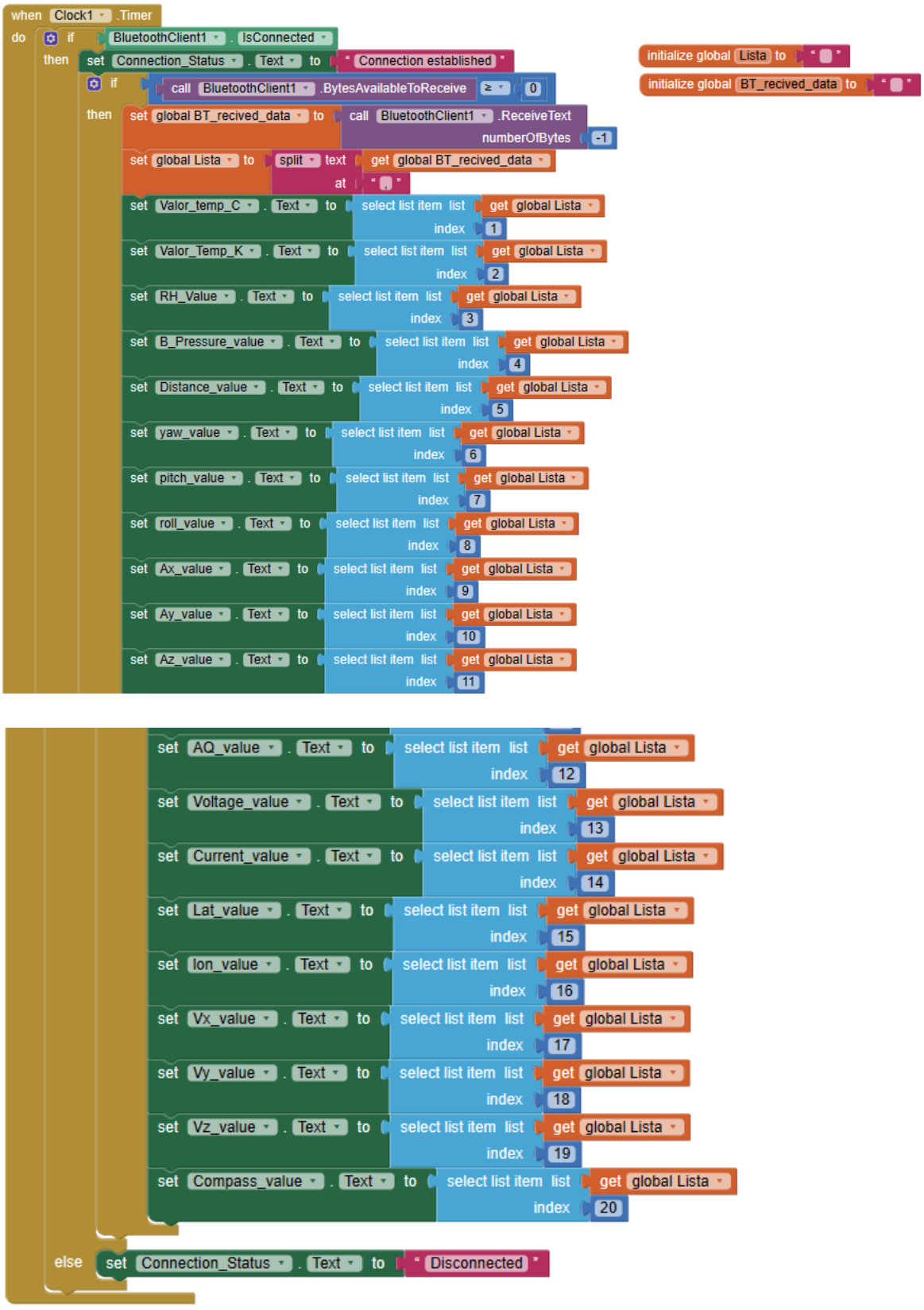
Figura 2.1. Interfaz para visualización de datos MIT App Inventor (Fuente: Propia).

3. Código MIT App Inventor

1.1. Código conexión y adquisición para “Classic Bluetooth”



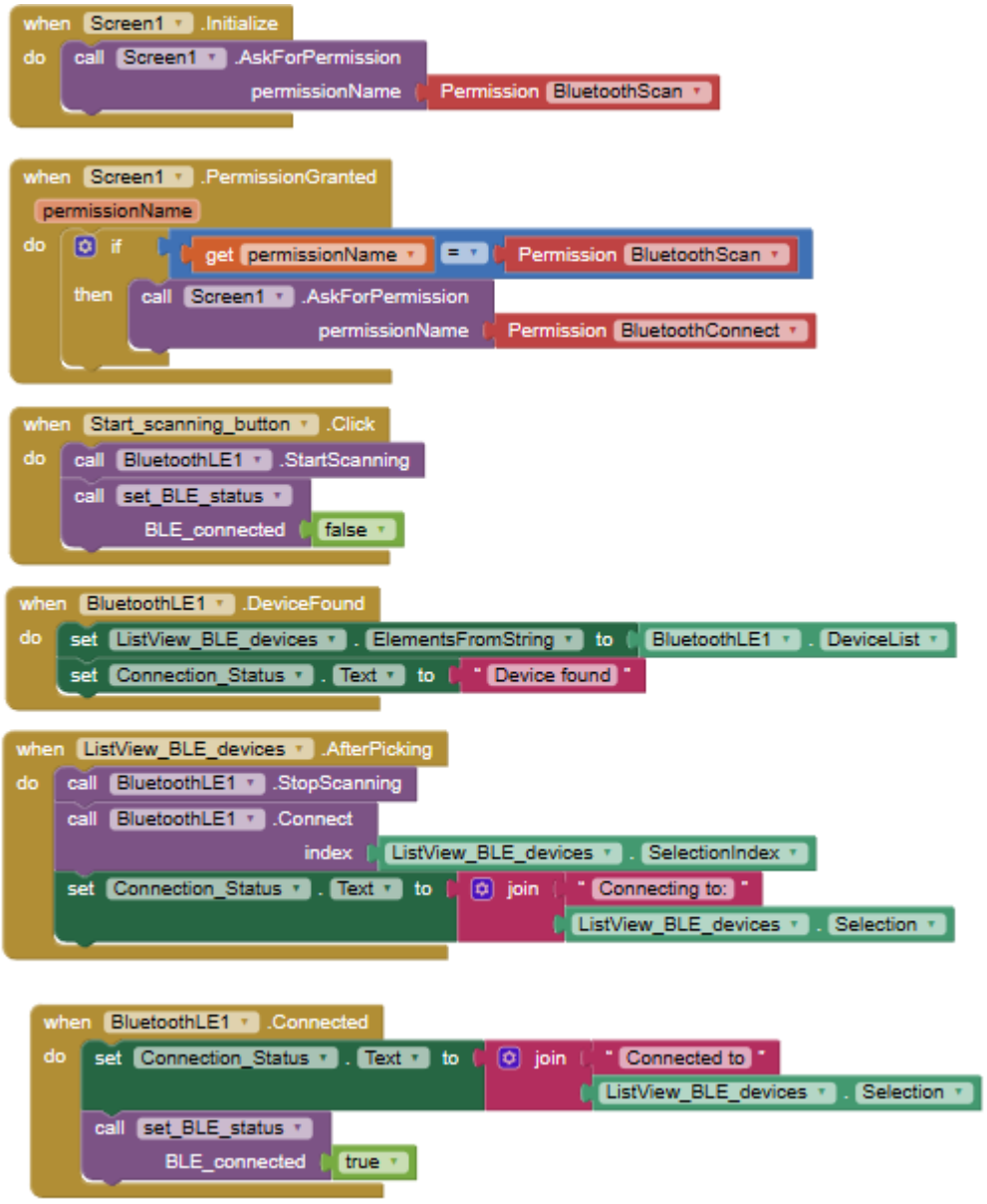
En estos bloques se inicializan los procesos y la solicitud de permisos, hasta Android 10 no necesarios, pero a partir de dicha actualización si no se incluyen los dos primeros bloques, no se inicializa el escaneo de dispositivos y aparece una lista en blanco, con lo que no se hace posible la conexión con ningún dispositivo.


when Clock1 .Timer
do if BluetoothClient1 . IsConnected
then set Connection_Status . Text to " Connection established "
if call BluetoothClient1 .BytesAvailableToReceive ≥ 0
then set global BT_recived_data to call BluetoothClient1 .ReceiveText numberOfBytes -1
set global Lista to split text get global BT_recived_data at " , "
set Valor_temp_C . Text to select list item list get global Lista index 1
set Valor_Temp_K . Text to select list item list get global Lista index 2
set RH_Value . Text to select list item list get global Lista index 3
set B_Pressure_value . Text to select list item list get global Lista index 4
set Distance_value . Text to select list item list get global Lista index 5
set yaw_value . Text to select list item list get global Lista index 6
set pitch_value . Text to select list item list get global Lista index 7
set roll_value . Text to select list item list get global Lista index 8
set Ax_value . Text to select list item list get global Lista index 9
set Ay_value . Text to select list item list get global Lista index 10
set Az_value . Text to select list item list get global Lista index 11
initialize global Lista to " "
initialize global BT_recived_data to " "
set AQ_value . Text to select list item list get global Lista index 12
set Voltage_value . Text to select list item list get global Lista index 13
set Current_value . Text to select list item list get global Lista index 14
set Lat_value . Text to select list item list get global Lista index 15
set lon_value . Text to select list item list get global Lista index 16
set Vx_value . Text to select list item list get global Lista index 17
set Vy_value . Text to select list item list get global Lista index 18
set Vz_value . Text to select list item list get global Lista index 19
set Compass_value . Text to select list item list get global Lista index 20
else set Connection_Status . Text to " Disconnected "

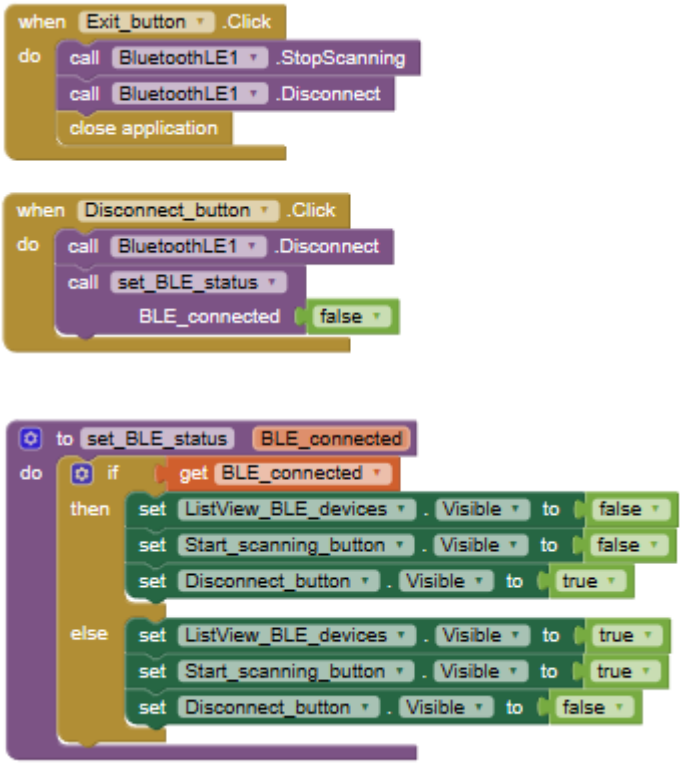
En estos bloques se inicializan variables de lista, que es la forma en la que se reciben los datos y posteriormente son separados mediante la instrucción “Split” y mostrados en los diferentes componentes o bloques de texto de la interfaz HMI.

Se inicializan dos relojes, uno que marca el proceso de recibir datos y mostrarlos por pantalla y, otro que se encarga de limpiar la lista, para asegurar que los datos recibidos están siempre actualizados y no se quedan variables pasadas almacenadas en esta. Es recomendable que el intervalo de tiempo del “clock” de limpieza sea menos que el recibir datos/ escritura.

1.2. Código conexión para “Bluetooth Low Energy”



En este caso, también es necesario inicializar el gestor de permisos para permitir el escaneo y la conexión de dispositivos bluetooth dado que también se utiliza versión de Android superior a 10, más concretamente Andorid 13 y, por otro lado, se está utilizando la extensión “BluetoothLE1” en la que se adicionan bloques únicamente disponibles para esta extensión y hace posible la conexión con dispositivos con este tipo de bluetooth.



De la misma manera, se adicionan controles en los botones del HMI y separa del resto del código el proceso a realizar una vez conectado el dispositivo bluetooth.