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

TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica, Industrial y Automática

DISEÑO Y MONTAJE DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN CUADRICÓPTERO

Anexos

Autor: Eric Vergara Samsó
Director: Manuel Andrés Manzanares Brotons

Convocatoria: Junio 2020

Índice

Código Arduino

```
//
//////////////////////////////////////////////////////////////////////////////////////////////// //
// DISEÑO Y MONTAJE DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN CUADRICÓPTERO - ERIC VERGARA SAMSÓ //
//
//////////////////////////////////////////////////////////////////////////////////////////////// //

//////////////////////////////////////////////////////////////////////////////////////////////// //

  #define usCiclo 7000  // Ciclo de ejecucion de software en microsegundos

 // MODOS DE FUNCIONAMIENTO
  int  iModoEstable = 1;       // Modo Estable
  int  iModoAcrobatico = 0;    // Modo Acrobatico

 // TEST PID
  int   iTestPID = 0;

 // LIBRERÍAS UTILIZADAS EN EL PROGRAMA
  #include <EnableInterrupt.h>
  #include <Wire.h>
  #include <LiquidCrystal_I2C.h>
  LiquidCrystal_I2C lcd(0x27, 16, 2);

 // DECLARACIÓN DE VARIABLES GLOBALES
  // TIEMPOS
    long     lExecuteTime, lLoopTimer, lLoopTimerAux, lLoopTimerESC, tiempo_ON, tiempo_1, tiempo_2;

  // BATERIA
    bool     bBatLow = false;
    float    fVoltBat, fReadBat= 0.00;
    int      iCBat, iContBatLow, iLowBat = 0;

  // GY-521
    #define   gyro_address 0x68
    float    angulo_pitch, angulo_roll, angulo_yaw, yawGyro, pitchGyro, rollGyro, angle_pitch_acc, angle_roll_acc, temperature;
    int      emerg, gx, gy, gz;
    float    acc_total_vector, ax, ay, az;
    float    gyro_X_cal, gyro_Y_cal, gyro_Z_cal, gyro_roll_input , gyro_pitch_input , gyro_yaw_input;
    float    angle_pitch_acc_cal, angle_roll_acc_cal, angle_yaw_acc_cal;
```

```
    bool     set_gyro_angles, accCalibOK = false;

  // MOTORES
    float    fESC1, fESC2, fESC3, fESC4;

  // MANDO RC
    float    calPotencia, pulsoPotencia, pulsoPotenciaAnt, PotenciaFilt = 0.00;
    float    wPitchConsigna, wRollConsigna, wYawConsigna = 0.00;
    float    calPitch, wPitch,  wPitchFilt = 0.00;
    float    calRoll, wRoll, wRollFilt = 0.00;
    float    calYaw, wYaw,  wYawlFilt = 0.00;

  // PID
    float    PIDwInPitch, pitch_w_giroscopio, pitch_w_error_prop, ITerm_pitch_w,
pitch_w_giroscopio_anterior, PID_pitch_w, DPitch_w;
    float    PIDwInRoll, roll_w_giroscopio, roll_w_error_prop, ITerm_roll_w,
roll_w_giroscopio_anterior, PID_roll_w, DRoll_w;
    float    PIDwInYaw, yaw_w_giroscopio, yaw_w_error_prop, ITerm_yaw_w,
yaw_w_giroscopio_anterior, PID_yaw_w, PID_yaw_w_ant, DYaw_w;
    float    pitch_ang_giroscopio, pitch_ang_error_prop, ITerm_pitch_ang,
pitch_ang_giroscopio_anterior, PID_pitch_ang, DPitch_ang;
    float    roll_ang_giroscopio, roll_ang_error_prop, ITerm_roll_ang, roll_ang_giroscopio_anterior,
PID_roll_ang, DRoll_ang;

  // CALIBRACION
    int      cal_int = 0;

  //LEDS
    int      iLED = 0;

  // MANDO POTENCIA
    const int PulsoMaxPotencia = 1830;
    const int PulsoMinPotencia = 930;
    const float tMaxPotencia = 1.83;
    const float tMinPotencia = 1.12;

    const float tMax = 2;
    const float tMin = 1;

    // MANDO PITCH
    const int wMaxPitch = -75;
    const int wMinPitch = 75;

  // MANDO ROLL
    const int wMaxRoll = 60;
    const int wMinRoll = -60;

  // MANDO YAW
```

```
    const int wMaxYaw = 60;
    const int wMinYaw = -60;

  // SEGURIDAD
    int cont_seguridad2;

  // INTERRUPCIONES
    volatile long contPotenciaInit; // LEER MANDO RC --> POTENCIA (Throttle)
    volatile int PulsoPotencia;

  // ADQUISICIÓN DE DATOS
    float    tiempo_ejecucion;
    int      contador, t;

 // SETUP - LLAMADA DE SUBPROGRAMAS
   void setup() {
    GENERAL();           // Iniciacion general
    BUCLE_MPU6050();      // Iniciacion MPU6050
    BUCLE_BATERIA();      // Leer tension de bateria
    BUCLE_CALIB_RC();     // Calibrar mando RC
    BUCLE_CALIB_MPU6050(); // Calibrar MPU6050

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Listo para volar");

    lLoopTimer = micros();
   }

 // LOOP PRINCIPAL
   void loop() {
    ProcesMPU();                                    // Obtener inclinacion (º)
    if (iModoEstable == 1)PID_ang();                        // Obtener salida PID inclinacion
(PIDang->PIDw->Salida)
    PID_w();                                   // Obtener salida PID velocidad
    Modulador();                                  // Modulador en base a salidas PID
    if (micros() - lLoopTimer > usCiclo + 50) digitalWrite(10, HIGH);          // Si la duración del ciclo ha
sido mayor de usCiclo+50us, encender led de aviso
                                              // Nunca hay que sobrepasar el tiempo de ciclo que se
haya estipulado
    while (micros() - lLoopTimer < usCiclo);                    // Comienzo de un nuevo ciclo
    lLoopTimer = micros();                               // Registrar instante de comienzo del ciclo
    PWM();                                    // Generar PWMs de salida
    while (micros() - lLoopTimer < 2200);                       // Esperar 2200us para leer el sensor
GY-521. La duracion de la salida PWM puede variar entre
                                              // 1000us y 2000us. Lo que se hace es esperar siempre
2200us para asegurarse de leer el sensor siempre en el mismo instante
    MPU_6050();                                   // Leer IMU
```

```
  AdquisicionDatos();                                  // Adquisición de datos
 }


// INICIALIZACIÓN DEL PROGRAMA
 void GENERAL() {
  Wire.begin();
  TWBR = 24;                              // Fijar el reloj I2C a 400kHz.
  lcd.init();
  lcd.backlight();

  // Mando RC (Declaración de iterrupciones)
   pinMode(5, INPUT_PULLUP);                  // YAW
   enableInterrupt(5, INTyaw, CHANGE);
   pinMode(6, INPUT_PULLUP);                  // POTENCIA
   enableInterrupt(6, INTpotencia, CHANGE);
   pinMode(7, INPUT_PULLUP);                  // PITCH
   enableInterrupt(7, INTpitch, CHANGE);
   pinMode(8, INPUT_PULLUP);                  // ROLL
   enableInterrupt(8, INTroll, CHANGE);

  // LEDs
   pinMode(10, OUTPUT);                    // LED rojo
   pinMode(A2, OUTPUT);                    // LED azul
   pinMode(12, OUTPUT);                    // LED verde
   pinMode(13, OUTPUT);                    // LED amarillo

  // Declaración de los motores
   pinMode(9, OUTPUT);                    // Motor 1
   pinMode(2, OUTPUT);                    // Motor 2
   pinMode(3, OUTPUT);                    // Motor 3
   pinMode(4, OUTPUT);                    // Motor 4

   digitalWrite(10, LOW);

  // Adquisición de datos
    Serial.begin(9600);
    analogWrite(A2, 255);

  // Mensaje al arrancar
   lcd.setCursor(0, 0);
   lcd.print("Encender mando");
   lcd.setCursor(0, 1);
   if (iModoEstable == 1)
   lcd.print("Modo Estable");
   else
   lcd.print("Modo Acrobatico");
  }
```

```
 // GESTIÓN DEL ACERÓMETRO MPU6050
  // Subrutina inicializacion
   void BUCLE_MPU6050() {
    Wire.beginTransmission(gyro_address);
    Wire.write(0x6B);                                   //Registro 6B hex - Activar modo configuración
del acelerómetro
    Wire.write(0x00);
    Wire.endTransmission();
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1B);                                   //Registro 1B hex - Configurar el giroscopio
    Wire.write(0x08);
    Wire.endTransmission();
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1C);                                   //Registro (1A hex) - Configurar acelerómetro
    Wire.write(0x10);
    Wire.endTransmission();

    Wire.beginTransmission(gyro_address);                       // Test MPU6050
    Wire.write(0x1B);
    Wire.endTransmission();
    Wire.requestFrom(gyro_address, 1);
    while (Wire.available() < 1);
    if (Wire.read() != 0x08) {
     lcd.backlight();
     lcd.setCursor(0, 0);
     lcd.print("IMU Error");                              // Si hay un error se enclava el programa
     while (1)delay(10);
    }

    // Se activa y configura un filtro pasa bajos que incorpora el sensor -> 0x04 <-> 10Hz(13.8ms)
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1A);
    Wire.write(0x04);
    Wire.endTransmission();
   }


  // Calibracion MPU6050
   void BUCLE_CALIB_MPU6050() {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Calibrando Giro");
    if (bBatLow == true) {
     lcd.setCursor(0, 1);
     lcd.print("Bateria Baja!");
    }
    for (cal_int = 0; cal_int < 3000 ; cal_int ++) {          // Calibrar giroscopio tomando 3000 muestras
```

```
      MPU_6050();
      gyro_X_cal += gx;
      gyro_Y_cal += gy;
      gyro_Z_cal += gz;
      delayMicroseconds(1000);
     }
     gyro_X_cal = gyro_X_cal / 3000;                    // Calcular el valor medio de las 3000 muestras
     gyro_Y_cal = gyro_Y_cal / 3000;
     gyro_Z_cal = gyro_Z_cal / 3000;

     lcd.setCursor(0, 0);
     lcd.print("Calibrando Acc.");

     if (iModoEstable == 1) {
      for (cal_int = 0; cal_int < 3000 ; cal_int ++) {          // Calibrar acelerometro tomando 3000
muestras
       MPU_6050();
       angle_pitch_acc_cal += ax;
       angle_roll_acc_cal += ay;
       angle_yaw_acc_cal += az;
      }
      angle_pitch_acc_cal = angle_pitch_acc_cal / 3000;          // Calcular el valor medio de las 3000
muestras
      angle_roll_acc_cal = angle_roll_acc_cal / 3000;
      angle_yaw_acc_cal = angle_yaw_acc_cal / 3000;
      accCalibOK = true;
     }
    }


  // Lectura MPU6050
    void MPU_6050() {
     emerg = 0;
     digitalWrite(9, LOW);
     digitalWrite(2, LOW);
     digitalWrite(3, LOW);
     digitalWrite(4, LOW);

     Wire.beginTransmission(gyro_address);     //Empezar comunicación con MPU6050
     Wire.write(0x3B);
     Wire.endTransmission();
     Wire.requestFrom(gyro_address, 14);

     lExecuteTime = (micros() - lLoopTimerAux) / 1000;
     lLoopTimerAux = micros();
     ax = Wire.read() << 8 | Wire.read();
     ay = Wire.read() << 8 | Wire.read();
     az = Wire.read() << 8 | Wire.read();
```

```
    temperature = Wire.read() << 8 | Wire.read();
    gx = Wire.read() << 8 | Wire.read();
    gy = Wire.read() << 8 | Wire.read();
    gz = Wire.read() << 8 | Wire.read();

    if (accCalibOK == true) {         // Restar valores de calibracion
     ax -= angle_pitch_acc_cal;
     ay -= angle_roll_acc_cal;
     az -= angle_yaw_acc_cal;
     az = az + 4096;
    }
   }


  // Calculo de angulo
   void ProcesMPU() {
    // Calculo w
     pitchGyro = (gx - gyro_X_cal) / 65.5;                              // 65.5 en "crudo" = 1º/s
     rollGyro = (gy - gyro_Y_cal) / 65.5;
     yawGyro = (gz - gyro_Z_cal) / 65.5;

     gyro_roll_input = rollGyro;
     gyro_pitch_input = pitchGyro;
     gyro_yaw_input = yawGyro;

    // Calculo ang
     angulo_pitch += pitchGyro * lExecuteTime / 1000 ;
     angulo_roll += rollGyro * lExecuteTime / 1000 ;
     angulo_pitch += angulo_roll * sin((gz - gyro_Z_cal) * lExecuteTime  * 0.000000266);       //
(tiempo_ejecucion/1000) * (1/65.5) * (PI/180)
     angulo_roll -= angulo_pitch * sin((gz - gyro_Z_cal) * lExecuteTime  * 0.000000266);

     acc_total_vector = sqrt(pow(ay, 2) + pow(ax, 2) + pow(az, 2));
     angle_pitch_acc = asin((float)ay / acc_total_vector) * 57.2958;                  // 57.2958 =
Conversión de radianes a grados 180/PI
     angle_roll_acc = asin((float)ax / acc_total_vector) * -57.2958;

    if (set_gyro_angles) {
     angulo_pitch = angulo_pitch * 0.995 + angle_pitch_acc * 0.005;                   // Angulo Pitch
de inclinacion
     angulo_roll = angulo_roll * 0.995 + angle_roll_acc * 0.005;                  // Angulo Roll de
inclinacion
    }
    else {
     angulo_pitch = angle_pitch_acc;
     angulo_roll = angle_roll_acc;
     set_gyro_angles = true;
    }
```

```
    }


 // GESTIÓN DEL MANDO DE RC
    void INTpotencia() {
      if (digitalRead(6) == HIGH) contPotenciaInit = micros();
      if (digitalRead(6) == LOW)PulsoPotencia = micros() - contPotenciaInit;
    }

    volatile long contPitchInit;  // PITCH
    volatile int PulsoPitch;
    void INTpitch() {
      if (digitalRead(7) == HIGH) contPitchInit = micros();
      if (digitalRead(7) == LOW) PulsoPitch = micros() - contPitchInit;
    }

    volatile long contRollInit;   // ROLL
    volatile int PulsoRoll;
    void INTroll() {
      if (digitalRead(8) == HIGH) contRollInit = micros();
      if (digitalRead(8) == LOW) PulsoRoll = micros() - contRollInit;
    }

    volatile long contYawInit;    // YAW
    volatile int PulsoYaw;
    void INTyaw() {
      if (digitalRead(5) == HIGH) contYawInit = micros();
      if (digitalRead(5) == LOW) PulsoYaw = micros() - contYawInit;
    }


   // Calibracion mando RC
    void BUCLE_CALIB_RC() {
      // Hasta no encender el mando no salir de este bucle por seguridad
      while (pulsoPotencia > 2500 || pulsoPotencia < 500) {
        pulsoPotencia =  ((PulsoMaxPotencia - PulsoMinPotencia) / (tMinPotencia - tMaxPotencia)) *
((PulsoPotencia) / 1000.00 - tMaxPotencia) + PulsoMinPotencia;
        delay(100);
      }

      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Calibrando ");
      lcd.setCursor(0, 1);
      lcd.print("Mando RC");

      delay(100);
      // Calibracion del mando
```

```
      for (cal_int = 0; cal_int < 100 ; cal_int ++) {
        calPotencia = calPotencia + PulsoPotencia;
        calPitch = calPitch + PulsoPitch;
        calRoll = calRoll + PulsoRoll;
        calYaw = calYaw + PulsoYaw;
        delay(20);
      }

      calPotencia = calPotencia / 100 - 990;
      calPitch = calPitch / 100 - 1500;
      calRoll = calRoll / 100 - 1500;
      calYaw = calYaw / 100 - 1500;

      // Tras calibrar el mando, hacer un comprobacion de las lecturas para ver si entan dentro de
rango
      wPitch =   ((wMinPitch - wMaxPitch) / (tMax - tMin)) * ((PulsoPitch - calPitch) / 1000.00 - tMin) +
wMaxPitch;
      wRoll =   ((wMaxRoll - wMinRoll) / (tMax - tMin)) * ((PulsoRoll - calRoll) / 1000.00 - tMin) +
wMinRoll;
      wYaw =  ((wMinYaw - wMaxYaw) / (tMax - tMin)) * ((PulsoYaw - calYaw) / 1000.00 - tMin) +
wMaxYaw;
      if (wPitch < -4 || wPitch > 4 || wRoll > 4 || wRoll < -4 || wYaw > 4 || wYaw < -4 ) {          // Test
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Error Mando RC");
        lcd.setCursor(0, 1);
        lcd.print("Reset");
        while (1)delay(10);
      }
     }

   // Lectura mando RC
     void lectura_mandoRC() {
      // Procesamiento de la señal --> funcion map() de Arduino
      pulsoPotencia =  ((PulsoMaxPotencia - PulsoMinPotencia) / (tMinPotencia - tMaxPotencia)) *
((PulsoPotencia) / 1000.00 - tMaxPotencia) + PulsoMinPotencia;
      wPitch =   ((wMinPitch - wMaxPitch) / (tMax - tMin)) * ((PulsoPitch - calPitch) / 1000.00 - tMin) +
wMaxPitch;
      wRoll =   ((wMaxRoll - wMinRoll) / (tMax - tMin)) * ((PulsoRoll - calRoll) / 1000.00 - tMin) +
wMinRoll;
      wYaw =  ((wMinYaw - wMaxYaw) / (tMax - tMin)) * ((PulsoYaw - calYaw) / 1000.00 - tMin) +
wMaxYaw;

      //  Filtro de lecturas del mando
      PotenciaFilt = PotenciaFilt * 0.9 + pulsoPotencia * 0.1;
      wPitchFilt = wPitchFilt * 0.9 + wPitch * 0.1;
      wRollFilt = wRollFilt * 0.9 + wRoll * 0.1;
      wYawlFilt = wYawlFilt * 0.9 + wYaw * 0.1;
```

```
      wPitchConsigna = wPitchFilt;
      wRollConsigna = wRollFilt;
      wYawConsigna = wYawlFilt;

      // Si las lecturas son cercanas a 0, se fuerzan a 0 para evitar inclinar el dron por error
      if (wPitchFilt < 3  && wPitchFilt > -3)wPitchConsigna = 0;
      if (wRollFilt < 3 && wRollFilt > -3)wRollConsigna = 0;
      if (wYawlFilt < 3  && wYawlFilt > -3)wYawConsigna = 0;

      if (iModoAcrobatico == 1) {
        // Si se vuela en modo estable, se controla la velocidad de cada eje
        wPitchConsigna = wPitchConsigna;
        wRollConsigna = wRollConsigna;
      }
      else {
        // Si se vuela en modo estable, se controla la inclinacion de cada eje, por lo que
        // se divide la consigna entre 2.6 para suavizar los movimientos (evitar movimientos bruscos)
        wPitchConsigna = wPitchConsigna / 2.6;
        wRollConsigna = wRollConsigna / 2.6;
      }

      // Detectar perdida mando: si la lectura del Mando RC es indentica a la medida anterior durante
      // un numero definido de lecturas, se entiende que se ha perdido la conexion con el mando RC.
      if (pulsoPotenciaAnt == pulsoPotencia)cont_seguridad2++;
      else cont_seguridad2 = 0;
      if (cont_seguridad2 > 150) { // Perdida conexion con mando RC
        // Tomar alguna medida si se desea
      }
      pulsoPotenciaAnt = pulsoPotencia;
    }

// GESTIÓN DE PWM DE LOS MOTORES
  bool aM1, aM2, aM3, aM4 = false;
  float accion_m1, accion_m2, accion_m3, accion_m4;

  // Modulador --> Calcular consignas para los motores

  void Modulador () {

    // Si el throttle es menos a 1300us, el control de estabilidad se desactiva. La parte integral
    // de los controladores PID se fuerza a 0.

    if (pulsoPotencia <= 1300) {
      ITerm_pitch_w = 0;
      ITerm_roll_w = 0;
      ITerm_yaw_w = 0;
      ITerm_pitch_ang = 0;
```

```
    lTerm_roll_ang = 0;
    fESC1 = PotenciaFilt;
    fESC2 = PotenciaFilt;
    fESC3 = PotenciaFilt;
    fESC4 = PotenciaFilt;
    if (fESC1 < 1000) fESC1 = 950;      // Si lo motores giran con el Stick de throttle al mínimo, recudir
este valor
    if (fESC2 < 1000) fESC2 = 950;
    if (fESC3 < 1000) fESC3 = 950;
    if (fESC4 < 1000) fESC4 = 950;
   }

   // Si el throttle es mayor a 1300us, el control de estabilidad se activa.
   if (pulsoPotencia > 1300) {
    if (pulsoPotencia > 1800)pulsoPotencia = 1800;          // Limitar throttle a 1800 para dejar
margen a los PID
    fESC1 = PotenciaFilt + PID_pitch_w - PID_roll_w - PID_yaw_w; // Motor 1
    fESC2 = PotenciaFilt + PID_pitch_w + PID_roll_w + PID_yaw_w; // Motor 2
    fESC3 = PotenciaFilt - PID_pitch_w + PID_roll_w - PID_yaw_w; // Motor 3
    fESC4 = PotenciaFilt - PID_pitch_w - PID_roll_w + PID_yaw_w; // Motor 4

    if (fESC1 < 1100) fESC1 = 1100;   // Evitar que alguno de los motores se detenga completamente
en pleno vuelo
    if (fESC2 < 1100) fESC2 = 1100;
    if (fESC3 < 1100) fESC3 = 1100;
    if (fESC4 < 1100) fESC4 = 1100;
    if (fESC1 > 2000) fESC1 = 2000;   // Evitar mandar consignas mayores a 2000us a los motores
    if (fESC2 > 2000) fESC2 = 2000;
    if (fESC3 > 2000) fESC3 = 2000;
    if (fESC4 > 2000) fESC4 = 2000;
   }
  }

  // Salidas PWM --> Motores
   void PWM() {
    digitalWrite(9, HIGH); //Motor 1 HIGH
    digitalWrite(2, HIGH); //Motor 2 HIGH
    digitalWrite(3, HIGH); //Motor 3 HIGH
    digitalWrite(4, HIGH); //Motor 4 HIGH

    aM1 = true;
    aM2 = true;
    aM3 = true;
    aM4 = true;

    // 1ms max
    tiempo_1 = micros();
    accion_m1 = fESC1 + lLoopTimer;
```

```
    accion_m2 = fESC2 + lLoopTimer;
    accion_m3 = fESC3 + lLoopTimer;
    accion_m4 = fESC4 + lLoopTimer;

    lectura_mandoRC(); // Leer mando RC
    BUCLE_BATERIA();   // Leer Bateria

    // Si la duracion entre tiempo_1 y tiempo_2 ha sido mayor de 900us, encender led de aviso.
    // Nunca hay que sobrepasar 1ms de tiempo en estado HIGH.
    tiempo_2 = micros();
    tiempo_ON = tiempo_2 - tiempo_1;
    if (tiempo_ON > 900) digitalWrite(10, HIGH);   // Tiempo excedido
    // 1ms max!!!!!!!

    while (aM1 || aM2 || aM3 || aM4 == true) {
     lLoopTimerESC = micros();
     if (accion_m1 <= lLoopTimerESC) { // Motor 1 LOW
       aM1 = false;
       digitalWrite(9, LOW);
     }
     lLoopTimerESC = micros();
     if (accion_m2 <= lLoopTimerESC) { // Motor 2 LOW
       aM2 = false;
       digitalWrite(2, LOW);
     }
     lLoopTimerESC = micros();
     if (accion_m3 <= lLoopTimerESC) { // Motor 3 LOW
       aM3 = false;
       digitalWrite(3, LOW);
     }
     lLoopTimerESC = micros();
     if (accion_m4 <= lLoopTimerESC) { // Motor 4 LOW
       aM4 = false;
       digitalWrite(4, LOW);
     }
    }
   }

// LECTURA DE LA TENSIÓN DE LA BATERIA
  void BUCLE_BATERIA() {
    fReadBat = analogRead(A3);      // Leer entrada analogica
    fVoltBat = 2.5 * (fReadBat * 5  / 1023);

    // Si la tension es inferior a 10.5V, se activa la señal de bateria baja
    if (fVoltBat < 10.5) {
      iContBatLow++;
      bBatLow = true;
    }
```

```
  }

// CONTROL PID
 // Parametros PID PITCH W (lazo velocidad)
  float Kp_pitch_w = 1.9, Ki_pitch_w = 0.07, Kd_pitch_w = 12;
  int salidaPI_pitch_w_max1 = 380;  // Limitar parte integral (anti-wind-up)
  int salidaPI_pitch_w_min1 = -380; // Limitar parte integral (anti-wind-up)
  int salidaPI_pitch_w_max2 = 380;  // Limitar salida del PID
  int salidaPI_pitch_w_min2 = -380; // Limitar salida del PID

 // Parametros PID ROLL W (lazo velocidad)
  float Kp_roll_w = 1.9, Ki_roll_w = 0.07, Kd_roll_w = 12;
  int salidaPI_roll_w_max1 = 380;  // Limitar parte integral (anti-wind-up)
  int salidaPI_roll_w_min1 = -380; // Limitar parte integral (anti-wind-up)
  int salidaPI_roll_w_max2 = 380;  // Limitar salida del PID
  int salidaPI_roll_w_min2 = -380; // Limitar salida del PID

 // Parametros PID YAW W (lazo velocidad)
  float Kp_yaw_w = 1.5, Ki_yaw_w = 0.05, Kd_yaw_w = 0;
  int salidaPI_yaw_w_max1 = 380;  // Limitar parte integral (anti-wind-up)
  int salidaPI_yaw_w_min1 = -380; // Limitar parte integral (anti-wind-up)
  int salidaPI_yaw_w_max2 = 380;  // Limitar salida del PID
  int salidaPI_yaw_w_min2 = -380; // Limitar salida del PID

 // Parametros PID PITCH ang (lazo inclinacion)
  float Kp_pitch_ang = 2.2, Ki_pitch_ang = 0.06, Kd_pitch_ang = 15 ;
  int salidaPI_pitch_ang_max1 = 130;  // Limitar parte integral (anti-wind-up)
  int salidaPI_pitch_ang_min1 = -130; // Limitar parte integral (anti-wind-up)
  int salidaPI_pitch_ang_max2 = 130;  // Limitar salida del PID
  int salidaPI_pitch_ang_min2 = -130; // Limitar salida del PID

 // Parametros PID ROLL ang (lazo inclinacion)
  float Kp_roll_ang = 2.2, Ki_roll_ang =  0.06, Kd_roll_ang = 15;
  int salidaPI_roll_ang_max1 = 130;  // Limitar parte integral (anti-wind-up)
  int salidaPI_roll_ang_min1 = -130; // Limitar parte integral (anti-wind-up)
  int salidaPI_roll_ang_max2 = 130;  // Limitar salida del PID
  int salidaPI_roll_ang_min2 = -130; // Limitar salida del PID

 // Parametros PID yaw ang (lazo inclinacion)
 //float Kp_yaw_ang = 1.6, Ki_yaw_ang =  0.022, Kd_yaw_ang = 0 * 3;
 //float  yaw_ang_giroscopio, yaw_ang_error_prop, ITerm_yaw_ang, yaw_ang_giroscopio_anterior,
PID_yaw_ang, PID_yaw_ang_ant, Dyaw_ang;
 //int salidaPI_yaw_ang_max1 = 100;  // Limitar parte integral (anti-wind-up)
 //int salidaPI_yaw_ang_min1 = -100; // Limitar parte integral (anti-wind-up)
 //int salidaPI_yaw_ang_max2 = 100;  // Limitar salida del PID
 //int salidaPI_yaw_ang_min2 = -100; // Limitar salida del PID
```

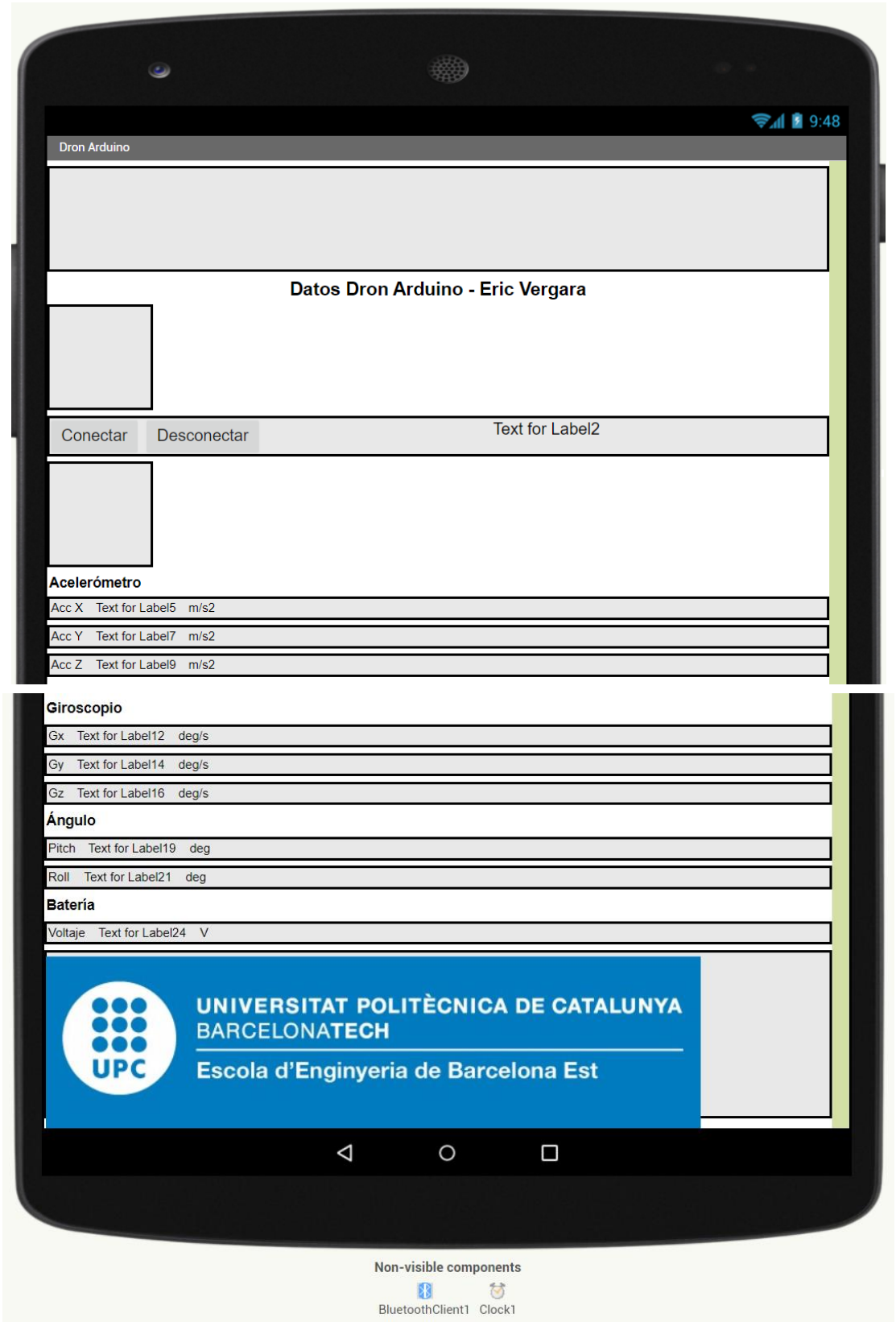
```
 // PID angulo (inclinacion) --> Lazo exterior
  void PID_ang() {

    //PITCH GYRO ang
    pitch_ang_error_prop = wPitchConsigna - angulo_pitch;                          // Error entre
lectura y consigna
    ITerm_pitch_ang += (Ki_pitch_ang * pitch_ang_error_prop);                      // Parte integral
(sumatorio del error en el tiempo)
    ITerm_pitch_ang = constrain(ITerm_pitch_ang, salidaPI_pitch_ang_min1,
salidaPI_pitch_ang_max1); // Limitar parte integral
    DPitch_ang = Kd_pitch_ang * (angulo_pitch - pitch_ang_giroscopio_anterior);    // Parte
Derivativa (diferencia entre el error actual y el anterior)

    PID_pitch_ang = Kp_pitch_ang * pitch_ang_error_prop + ITerm_pitch_ang - DPitch_ang;  //
Salida PID
    PID_pitch_ang = constrain(PID_pitch_ang, salidaPI_pitch_ang_min2, salidaPI_pitch_ang_max2);
// Limitar salida del PID
    pitch_ang_giroscopio_anterior = angulo_pitch;                                  // Error enterior

    // ROLL GYRO ang
    roll_ang_error_prop = wRollConsigna - angulo_roll;
    ITerm_roll_ang += (Ki_roll_ang * roll_ang_error_prop);
    ITerm_roll_ang = constrain(ITerm_roll_ang, salidaPI_roll_ang_min1, salidaPI_roll_ang_max1);
    DRoll_ang = Kd_roll_ang * (angulo_roll - roll_ang_giroscopio_anterior);

    PID_roll_ang = Kp_roll_ang * roll_ang_error_prop + ITerm_roll_ang - DRoll_ang; // salida PID
    PID_roll_ang = constrain(PID_roll_ang, salidaPI_roll_ang_min2, salidaPI_roll_ang_max2);
    roll_ang_giroscopio_anterior =  angulo_roll;

    //  //YAW GYRO ang
    //  yaw_ang_error_prop = wYawConsigna;
    //  ITerm_yaw_ang += (Ki_yaw_ang * yaw_ang_error_prop);
    //  ITerm_yaw_ang = constrain(ITerm_yaw_ang, salidaPI_yaw_ang_min1,
salidaPI_yaw_ang_max1);
    //  Dyaw_ang = Kd_yaw_ang * (yaw_ang_error_prop - yaw_ang_giroscopio_anterior);
    //
    //  PID_yaw_ang = Kp_yaw_ang * yaw_ang_error_prop + ITerm_yaw_ang + Dyaw_ang; // salida
PID
    //  PID_yaw_ang = constrain(PID_yaw_ang, salidaPI_yaw_ang_min2, salidaPI_yaw_ang_max2);
    //  yaw_ang_giroscopio_anterior =  yaw_ang_error_prop;
  }

 //PID w (velocidad) --> Lazo interior

  void PID_w() {
    // En funcion del modo de vuelo que hayamos seleccionado, las consignas de los PID serán
diferentes
```

```
    // En modo Acro solo controlamos la velocidad de cada eje (un PID por eje). La consigna  del PID se da en º/s
    if (iModoAcrobatico == 1) {
      PIDwInPitch = wPitchConsigna;
      PIDwInRoll = wRollConsigna;
    }

    // En modo Estable controlamos la velocidad y la inclinacion de cada eje (dos PID por eje). La consigna del primer PID (inclinacion) se da en º
    // La salida del PID de inclinacion será la consigna de velocidad para el PID de velocidad
    else {
      PIDwInPitch = PID_pitch_ang;
      PIDwInRoll = PID_roll_ang;
    }
    //PITCH GYRO w
     pitch_w_error_prop = PIDwInPitch - gyro_pitch_input;
    ITerm_pitch_w += (Ki_pitch_w * pitch_w_error_prop);
    ITerm_pitch_w = constrain(ITerm_pitch_w, salidaPI_pitch_w_min1, salidaPI_pitch_w_max1);
    DPitch_w = Kd_pitch_w * (gyro_pitch_input - pitch_w_giroscopio_anterior);

    PID_pitch_w = Kp_pitch_w * pitch_w_error_prop + ITerm_pitch_w - DPitch_w; // salida PID
    PID_pitch_w = constrain(PID_pitch_w, salidaPI_pitch_w_min2, salidaPI_pitch_w_max2);
    pitch_w_giroscopio_anterior = gyro_pitch_input;

    //ROLL GYRO w
    roll_w_error_prop = PIDwInRoll - gyro_roll_input;
    ITerm_roll_w += (Ki_roll_w * roll_w_error_prop);
    ITerm_roll_w = constrain(ITerm_roll_w, salidaPI_roll_w_min1, salidaPI_roll_w_max1);
    DRoll_w = Kd_roll_w * (gyro_roll_input - roll_w_giroscopio_anterior);

    PID_roll_w = Kp_roll_w * roll_w_error_prop + ITerm_roll_w - DRoll_w; // salida PID
    PID_roll_w = constrain(PID_roll_w, salidaPI_roll_w_min2, salidaPI_roll_w_max2);
    roll_w_giroscopio_anterior =  gyro_roll_input;

    //YAW GYRO w
    yaw_w_error_prop = wYawConsigna - gyro_yaw_input;
    ITerm_yaw_w += (Ki_yaw_w * yaw_w_error_prop);
    ITerm_yaw_w = constrain(ITerm_yaw_w, salidaPI_yaw_w_min1, salidaPI_yaw_w_max1);
    DYaw_w = Kd_yaw_w * (gyro_yaw_input - yaw_w_giroscopio_anterior);

    PID_yaw_w = Kp_yaw_w * yaw_w_error_prop + ITerm_yaw_w - DYaw_w; // salida PID
    PID_yaw_w = constrain(PID_yaw_w, salidaPI_yaw_w_min2, salidaPI_yaw_w_max2);
    yaw_w_giroscopio_anterior =  gyro_yaw_input;
   }

//ADQUISICIÓN DE DATOS
  void AdquisicionDatos(){
```

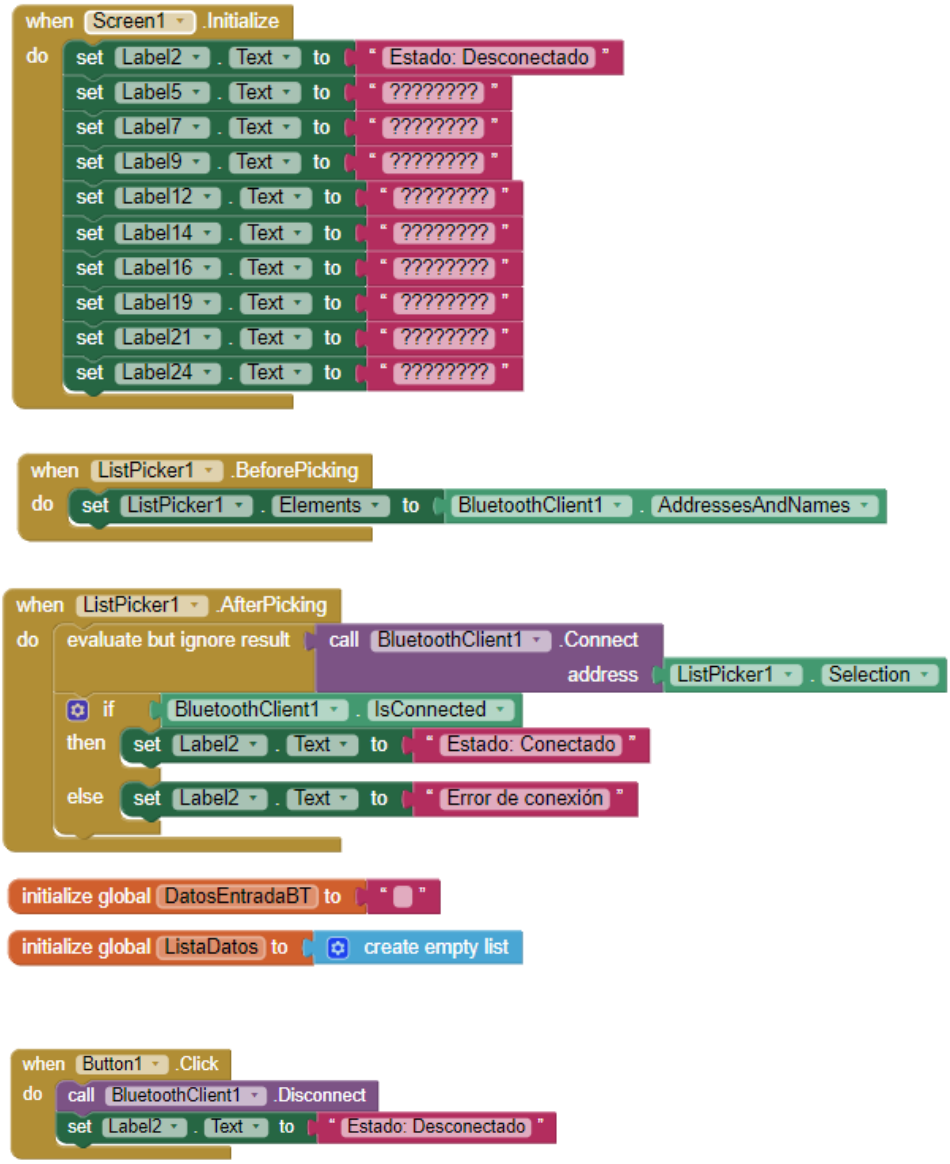
```
while (micros() - tiempo_ejecucion < 5000);  // Mandar los datos cada 5ms
tiempo_ejecucion = micros();

 if (contador == 0)Serial.print(123456789);
 if (contador == 1)Serial.print(F("|"));

//ACELERÓMETRO + GIROSCOPIO
 if (contador == 2)Serial.print(ax / 4096);
 if (contador == 3)Serial.print(F("|"));
 if (contador == 4)Serial.print(ay / 4096);
 if (contador == 5)Serial.print(F("|"));
 if (contador == 6)Serial.print(az / 4096);
 if (contador == 7)Serial.print(F("|"));

 if (contador == 8)Serial.print((gx - gyro_X_cal) / 16.4);
 if (contador == 9)Serial.print(F("|"));
 if (contador == 10) Serial.print((gy - gyro_Y_cal) / 16.4);
 if (contador == 11) Serial.print(F("|"));
 if (contador == 12)Serial.print((gz - gyro_Z_cal) / 16.4);
 if (contador == 13)Serial.print(F("|"));

 //ÁNGULO
 if (contador == 14)Serial.print(angulo_pitch);
 if (contador == 15)Serial.print(F("|"));
 if (contador == 16)Serial.print(angulo_roll);
 if (contador == 17)Serial.print(F("|"));

 //BATERIA
 if (contador == 18)Serial.print(fVoltBat);
 if (contador == 19)Serial.print(F("|"));

contador ++;
if (contador == 20)contador = 0;

t++;
if (t == 10)t = 0;

//ENSEÑAR PID
if (iTestPID == 1) {
  Ki_pitch_w = 0;
  Ki_roll_w = 0;
  Ki_yaw_w = 0;
  Ki_pitch_ang = 0;
  Ki_roll_ang = 0;

  if (contador == 0)Serial.print(fESC1);
  if (contador == 1)Serial.print("\t");
  if (contador == 2)Serial.print(fESC2);
```

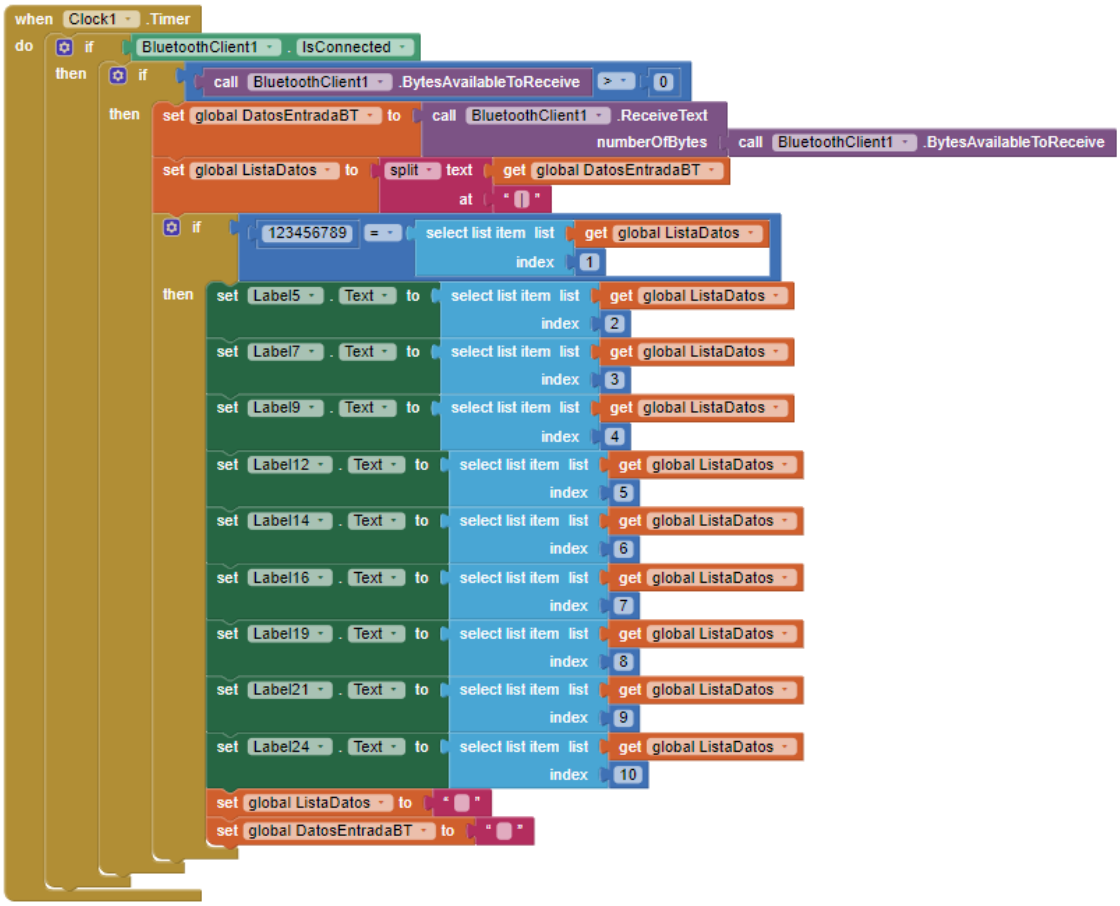
```
    if (contador == 3)Serial.print("\t");
    if (contador == 4)Serial.print(fESC3);
    if (contador == 5)Serial.print("\t");
    if (contador == 6)Serial.println(fESC4);

    contador ++;
    if (contador == 7)contador = 0;
   }
}
```

Interfaz MIT App Inventor



Código MIT App Inventor




when Clock1 .Timer
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global DatosEntradaBT to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global ListaDatos to split text get global DatosEntradaBT
at " "
if 123456789 = select list item list get global ListaDatos
index 1
then set Label5 . Text to select list item list get global ListaDatos
index 2
set Label7 . Text to select list item list get global ListaDatos
index 3
set Label9 . Text to select list item list get global ListaDatos
index 4
set Label12 . Text to select list item list get global ListaDatos
index 5
set Label14 . Text to select list item list get global ListaDatos
index 6
set Label16 . Text to select list item list get global ListaDatos
index 7
set Label19 . Text to select list item list get global ListaDatos
index 8
set Label21 . Text to select list item list get global ListaDatos
index 9
set Label24 . Text to select list item list get global ListaDatos
index 10
set global ListaDatos to " "
set global DatosEntradaBT to " "