Project 2: Remote management of robotics arm

Alberto Vázquez Martínez

Alberto.Vazquez1@alu.uclm.es

Curso 2021/22 – Diseño de Sistemas Basado en Microprocesador

Universidad de Castilla-La Mancha

Escuela Superior de Informática

Contenido

1. Circuito diseñado

Para este proyecto he optado por utilizar dos placas ESP32. Una de la ellas la utilizaré para implementar el control con el joystick analógico y la otra mandará las ordenes a los dos servomotores. Para la comunicación entre las placas, utilizaré Bluetooth y configuraré una placa como maestro (la del joystick) y la otra como esclavo (la de los servomotores).

El circuito diseñado quedaría tal y como se muestra en la imagen.

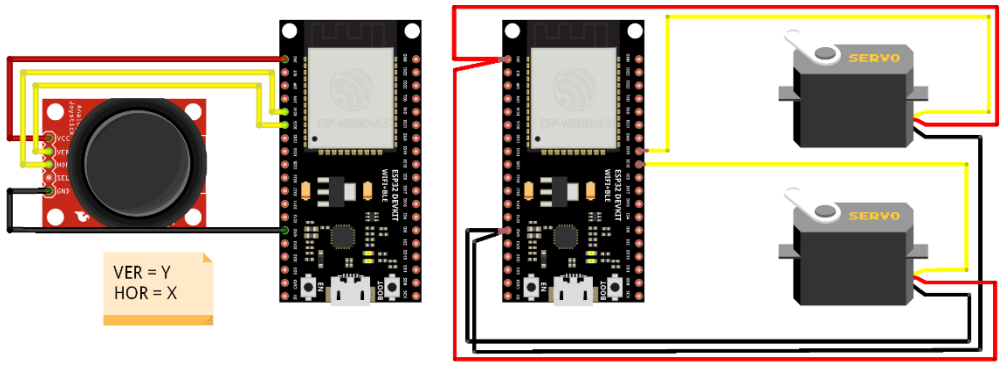

Fig. 1 – Diseño del circuito

2. Código desarrollado placa ESP32 (maestro)

Esta placa se encargará de enviar la información obtenida por el joystick a la otra placa ESP32, mediante una conexión Bluetooth.

Primero, configuro el ESP32 como maestro para la conexión Bluetooth. Para ello utilizo la función *bluetoothConnect()* y utilizando un objeto *BluetoothSerial* intento conectarme al otro ESP32 repetidamente.

```
/* Configure and connect bluetooth */
void bluetoothConnect() {
  BT.begin("ESP32_client", true); /* True indicates that is configured as a master */
  Serial.println("El dispositivo Bluetooth está en modo maestro. Conectando con el anfitrión ...");
  connected = BT.connect(clientName);
  if(connected) {
    Serial.println("¡Conectado exitosamente!");
  } else {
    while(!BT.connected(10000)) {
      Serial.println("No se pudo conectar. Asegúrese de que el dispositivo remoto esté disponible y
    }
  }
}
```

Fig. 2 – Configuración conexión Bluetooth maestro

Una vez que se establece la conexión con el esclavo, entonces se llama a la función *app_ main()* la cual inicializa un semáforo *mutex* y crea dos tareas.

```
void app_main() {
  xMutex = xSemaphoreCreateMutex();

  if( xMutex != NULL )
  {
    xTaskCreate(readPinTask, "XPin", 4096, (void *) XPin, 10, NULL);
    xTaskCreate(readPinTask, "YPin", 4096, (void *) YPin, 10, NULL);
  }
}
```

Fig. 3 – Crear tareas

Las tareas que se crean se asocian a la misma función, llamada *readPinTask()*, que recibe como parámetro el pin que se desea leer del joystick analógico, es decir, X o Y. Una vez obtiene el valor, mapea ese valor para que el resultado sea entre un número de 0 a 180, ya que el servomotor tiene un rango de movimiento de 0 a 180 grados.

```
/* Read a given position (X or Y) of the joystick */
void readPinTask(void *pvParameters) {
  int val = 0;
  int pin = ( int ) pvParameters;
  for(;;) {
    val = analogRead(p int ADC_Max
    val = map(val, 0, ADC_Max, 0, 180); /* Map analog reading to 0 to 180 degrees */
    sendInfoToSlave(val, pin);
    vTaskDelay(250/portTICK_PERIOD_MS);
  }
  vTaskDelete(NULL);
}
```

Fig. 4 – Obtención del valor correspondiente al joystick analógico

Además, en esta función se llama a la función *sendInfoToSlave()*, la cual se encarga de enviar el dato leído a la placa ESP32 esclava por Bluetooth.

```
/* Send values of joystick to the other ESP32 */
void sendInfoToSlave(int val, int pin) {
  xSemaphoreTake( xMutex, 500/portTICK_PERIOD_MS  ); /* Use a mutex to avoid multiple tasks in this section of code */
  {
    /* Send two data in order to identify if the data is from X or Y position (determined by pin variable) */
    BT.write(val);
    BT.write(pin);
  }
}
```

Fig. 5 – Enviar información a la placa ESP32 esclava

Dentro de esa función se utiliza el semáforo *mutex* definido anteriormente. Esto se hace ya que se envían dos mensajes por Bluetooth, primero el valor obtenido del joystick y después el pin que se ha leído. Esto se hace para que la placa ESP32 esclava pueda distinguir que servomotor tiene que mover dependiendo del pin X o el pin Y. Por lo que para que exista un orden en los mensajes se necesita definir una zona crítica que asegure que solo una tarea/proceso pueda acceder a este código.

3. Código diseñado placa ESP32 (esclavo)

Primero en la placa ESP32 esclava se conecta a la placa ESP32 maestra para establecer la conexión Bluetooth.

```
void setup() {
  Serial.begin(9600); // Inicialización de la conexión en serie para la depuración
  BT.begin("ESP32_Servomotor"); // Nombre de su dispositivo Bluetooth y en modo esclavo
  Serial.println("El dispositivo Bluetooth está listo para emparejarse");
  initServo();

  app_main();
}
```

Fig. 6 – Función setup de ESP32 esclavo

Además, en esta función inicializo los servomotores con la función *initServo()*, estableciendo los *timers*, configuración de los hercios y de los pines por los que recibirá la información los servomotores.

```
void initServo() {
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  myServo1.setPeriodHertz(50);
  myServo1.attach(pinServo1, 500, 2400);
  myServo2.setPeriodHertz(50);
  myServo2.attach(pinServo2, 500, 2400);

  myServo1.write(90);
  myServo2.write(90);
}
```

Fig. 7 – Inicialización de servomotores

Defino una función, que se ejecutará como una tarea de manera continua, llamada *readIncomingData()* y se utilizará para recibir la información por parte de la placa ESP32 maestra. Dependiendo del pin, se moverá un servomotor u otro, utilizando la función *write()*.

```
void readIncomingData(void *pvParameters) {
  int incoming = 0;
  int previousData = 0;
  for(;;) {
    if (BT.available()) {
      incoming = BT.read();
      if (incoming == XPin) {
        Serial.print("Recibido (X): ");
        Serial.println(previousData);
        myServo1.write(previousData);
      } else if (incoming == YPin) {
        Serial.print("Recibido (Y): ");
        Serial.println(previousData);
        myServo2.write(previousData);
      } else {
        previousData = incoming;
      }
    }
    vTaskDelay(250/portTICK_PERIOD_MS);
  }
  vTaskDelete(NULL);
}
```

Fig. 8 – Recepción de información utilizando Bluetooth

El circuito real con todo montado quedaría tal que así:

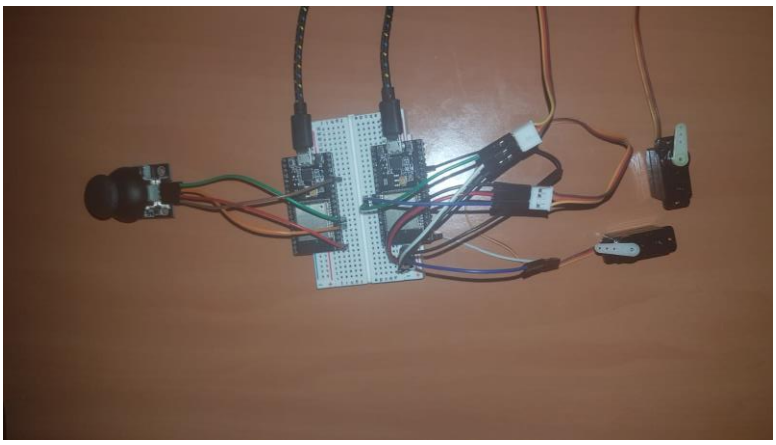

Fig. 9 – Imagen del circuito