COMUNICACIONES INALÁMBRICAS

TRABAJO PRÁCTICO N° 2

COMUNICACIÓN DE DATOS

DOCENTES: MG. ING. MARTIN PICO, ING. MILTON POZZO
CHIABO FRANCO, KUHN LAUTARO, PALOMEQUE MATEO

Índice

Objetivos a cumplir

Tenemos como objetivo realizar una comunicación inalámbrica mediante un sensor infrarrojo como receptor y de emisor un control, un led emisor infrarrojo y un dispositivo programable. Esto tiene como finalidad poder intercambiar información entre dos dispositivos digitales sin la utilización de cables, y para eso necesitamos instruirnos sobre como decodificar señales ópticas provenientes de dispositivos existentes y como recrear señales ópticas para transmitir acciones a otros medios.

Decodificar la señal proveniente de un control remoto

Para esta primera instancia del trabajo, se requiere decodificar la señal recibida de 7 botones de un control remoto que cuanta con dicha cantidad de pulsador e incluso más. Cada botón cuenta con un valor especifico que le fue asigna por su fabricante, y la intención es que se puedan mostrar en el puerto serie. Para conseguir esto se empleó del diseño sencillo de circuito.

Circuito y componentes utilizados

Para la recepción y decodificación de la señal óptica enviada por el control remoto, se utilizó como dispositivo programable un Arduino Nano. Esta placa se es conecta mediante una protoboard a un sensor infrarrojo VS1838B empaquetado, que nos permitirá captar la señal infrarroja. Este sensor cuenta con tres “patitas”, la primera es la receptora de señal que se encuentra conectado al pin digital 9 del dispositivo, la del medio es el GND o también conocida “tierra”, y la última es la VCC, recibe la tención para encender.

Figura 1. Sensor Infrarrojo VS1838B.

Como emisor se utiliza un control remoto básico que se encuentre en cualquier kit de Arduino que cuenta con 17 botones. Este Mando usa el protocolo NEC que trabaja a 38KHz de frecuencia y al presionar un botón envía la señal con formato de tren de pulsos.

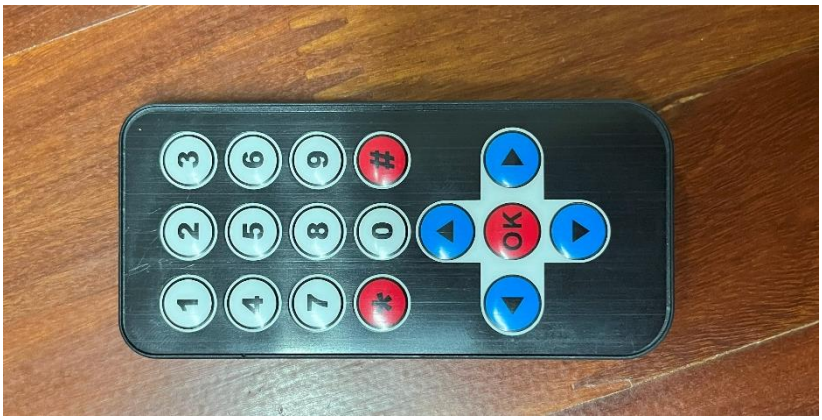

Figura 2. Control Kit Arduino.

Código programable

Si bien es fácil de llevar a cabo el circuito, no podría funcionar sin un código programado para el Arduino Nano que sea capaz de captar y decodificar las señales infrarrojas emitidas por el control remoto. Por esa razón se investigo acerca de esta situación, y nos encontramos con una librería con la capacidad de resolver esta problemática. La librería es IRremote, nos facilita todo el proceso y lo realiza de forma casi automática, pero antes de explicar el uso del código, ¿Qué hace esta librería?

La librería **IRremote** es una biblioteca de código abierto para Arduino que permite a los microcontroladores Arduino enviar y recibir señales infrarrojas (IR). Esta librería simplifica enormemente el proceso de trabajar con dispositivos infrarrojos permitiendo a un dispositivo recibir señales infrarrojas de otros dispositivos, lo que lo hace útil para proyectos donde deseas interactuar con controles remotos como es en nuestro caso puntual.

La librería incluye soporte para decodificar una amplia gama de protocolos IR comunes, como NEC, Sony, Panasonic, Philips, etc. Esto facilita la interoperabilidad con diferentes dispositivos que utilizan diferentes protocolos IR. Nosotros debido al control que usamos, el protocolo a implementar es NEC.

Una vez explicado el uso y función de la librería procedemos a desarrollar el código y darlo a entender a grandes rasgos.

En primer luchar se incluye la librería IRremote y guardamos el pin digital 9 como la entrada de la señal a captar con el nombre de variable *IRpin.*

Luego definimos como entrada de señal infrarroja utilizando la sentencia *IrReceiver.begin(IRpin)* de la librería. De esta forma se compromete al pin anteriormente guardado a ser el receptor.

Para terminar, dentro del bucle principal del programa se agrega un condicional para que cuando el dispositivo decodifique un 0 entre en bucle, esto se logra con el comando *IrReceiver.decode()*. Una vez inicializado, la librería es capaz de obtener el protocolo que utiliza el dispositivo emisor mediante la sentencia *IrReceiver.decodedIRData.protocol*, y como no está incorporado el registro de valores, nos brindará un valor numérico. Tal como se aclaró antes en nuestro caso el protocolo es NEC y nos mostrará en el puerto serie el número 7.

La sentencia *IrReceiver.decodedIRData.command* decodifica la señal y la guarda en una variable que se definió anteriormente, y que será visible en el puerto serie para así terminar con el proceso limpiando y dejando en espera las sentencias de la librería utilizadas para la decodificación mediante el comando *IrReceiver.resume().*

De esta forma se dio a conocer el protocolo y valor de cada botón del control remoto utilizado para la comunicación inalámbrica. A continuación, se compartirá los resultados obtenidos en la decodificación.

Código	Botón
69	1
70	2
71	3
68	4
64	5
67	6
7	7
21	8
9	9
25	0
22	*
13	#
28	OK
90	Derecha
8	Izquierda
24	Arriba
82	Abajo

Controlador LED RGB por control remoto

Una vez decodificado todos los botones del control remoto procederemos a realizar un circuito donde podremos controlar un led RGB para que cumpla con diferentes situaciones mediante la utilización y emisión del control remoto anteriormente decodificado.

Circuito Led RGB

La conexión es sencilla ya que utilizaremos la base del circuito decodificador agregando el led RGB a los pines digitales 2, 3 y 4 del Arduino Nano. El pin 2 será utilizado por la patita del led azul, que se le añadirá una resistencia de 470 Ω, el pin 3 se usará para el led verde y también tendrá una resistencia de 470 Ω, y por último en el pin 4 se conectará con el led rojo con una resistencia de 560 Ω.

Como se explicó con anterioridad, el circuito implementado solo contará del sensor receptor de infrarrojo, utilizado de la misma forma que para la decodificación, y del led RGB con sus respectivas resistencias y conexiones. No se complejiza en ningún punto en particular.

Programa controlador del Led RGB

Para poder manipular el led a través del control remoto vamos a necesitar más que solo recibir la señal óptica recibida, por eso se emplea un código capaz de capturar, decodificar y mediante los valores obtenidos indicar acciones para el led RGB.

Empezamos incluyendo la librería IRremote, la misma empleada para la decodificación que se mencionó antes, y definiendo los pines del led RGB y del sensor infrarrojo VS1838B.

```
// Definir pines para el LED RGB y llegada del infrarojo
#define PIN_LED_RED 4
#define PIN_LED_GREEN 3
#define PIN_LED_BLUE 2
#define IRpin 9
```

Figura 3. Variables definidas.

Luego procedemos a definir los códigos de las acciones que cada botón va a realizar. Para esto debemos adjudicarle a cada acción un valor que se obtuvo en la decodificación para así indicar que botón ejecutará el código.

```
// Definir códigos de los botones del control remoto
#define CODIGO_BOTON_1 69   //prender y apagar
#define CODIGO_BOTON_2 70   //aumentar nivel del rojo
#define CODIGO_BOTON_3 71   //disminuir el nivel de rojo
#define CODIGO_BOTON_4 68   //aumentar nivel de verde
#define CODIGO_BOTON_5 64   //disminuir nivel de verde
#define CODIGO_BOTON_6 67   //aumentar nivel de azul
#define CODIGO_BOTON_7 7    //disminuir nivel de azul
```

Figura 4. Botones definidos por Función.

Continuamos definiendo variables que se utilizarán a lo largo del programa, tales como una variable para almacenar el valor decodificado del control remoto, una variable que determina el estado del led RGB, encendido o apagado, y por último las variables de intensidad inicial de los leds de diferentes colores, en este caso de los tres iniciarán en 255, su máximo.

En el void setup() inicializamos el receptor infrarrojo mediante la sentencia IrReceiver.begin(IRpin), donde definimos al sensor como entrada de la señal emitida por el control remoto. En esta instancia también se configuran los pines del led RGB como salidas, OUTPUT.

En el bucle principal, void loop(), se procede a implementar el funcionamiento general del programa. Empezamos con un condicional para que el código corra solo si se detecta una señal emisora, para esto usamos la sentencia IrReceiver.decode(). Si es detectada empieza el bucle, donde lo primera que hace es decodificar y almacenar la señal recibida en la variable que se definió anteriormente, se consigue mediante igualar la variable de almacenamiento con el comando IrReceiver.decodedIRData.command.

Ahora con la decodificación completa, se sabe de qué botón proviene la señal, por lo cual se sabrá qué acciones debe cumplir en base a la configuración planteada al principio del código. Aquí se mapea las acciones de los botones y ejecutará la correspondiente al valor obtenido.

Después decidimos mostrar los valores obtenidos en el puerto serie para tener un control e información del estado del led RGB.

Y al final del bucle contamos con una sentencia que reinicia al receptor para que así pueda volver a decodificar y almacenar otro valor al recibir nuevamente una señal, implementamos IrReceiver.resume() para llevarlo a cabo.

Estuvimos hablando de las acciones que deben cumplir los botones y que se ejecutarán en el bucle, pero no se explicó cómo se realizan, para la primera acción se emplea una función en donde el led RGB se encenderá o apagará, para esto se implementa la variable booleana que se definió anteriormente sobre el estado inicial del

led, en caso de que sea verdadera (true), las variables de las intensidades de los colores será la misma que se especificó al inicio del programa. En caso de que el estado sea falso (false), las intensidades serán 0, esto hace que el led este totalmente apagado.

```
// Función para cambiar el estado del LED (encender/apagar)
void cambiarEstadoLED() {

  estado = !estado;
  if (estado == true) {
    analogWrite(PIN_LED_RED, intensidadR);
    analogWrite(PIN_LED_GREEN, intensidadG);
    analogWrite(PIN_LED_BLUE, intensidadB);
  }

  if (estado == false) {
    analogWrite(PIN_LED_RED, 0);
    analogWrite(PIN_LED_GREEN, 0);
    analogWrite(PIN_LED_BLUE, 0);
  }
}
```

Figura 5. Función implementada para el cambio de estado del LED RGB.

Para finalizar con el programa solo nos queda explicar las últimas dos funciones que son para el incremento y disminución de la intensidad de los colores del led RGB. Estas funciones si bien son dos distintas, su configuración y funcionamiento son exactamente iguales. Se inicia con un condicional para saber si el led se encuentra encendido, se verifica el estado, si lo está se procede a definir otros tres condicionales, una por cada color para así saber en cual led se desea aplicar la acción. Por último se utiliza la sentencia constrain() que nos permite trabajar el valor deseado entre un límite máximo y mínimo. Este comando requiere de tres parámetros, el primera es la variable inicial y la resto o suma que se desea aplicar en cada accionar, el segundo es el mínimo del valor, en este caso 0, y el tercero es el máximo que sería 255.

```
// Función para aumentar la intensidad de un color
void aumentarIntensidadColor(int pinLED) {
  if (estado == true) {
    if (pinLED == PIN_LED_RED) {
      intensidadR = constrain(intensidadR + 20, 0, 255); /
      analogWrite(pinLED, intensidadR);
    }
    if (pinLED == PIN_LED_GREEN) {
      intensidadG = constrain(intensidadG + 20, 0, 255);
      analogWrite(pinLED, intensidadG);
    }
    if (pinLED == PIN_LED_BLUE) {
      intensidadB = constrain(intensidadB + 20, 0, 255);
      analogWrite(pinLED, intensidadB);
    }
  }
}
```

Figura 6. Función implementada para el incremento de la intensidad del LED RGB.

```
// Función para disminuir la intensidad de un color
void disminuirIntensidadColor(int pinLED) {
  if (estado == true) {
    if (pinLED == PIN_LED_RED) {
      intensidadR = constrain(intensidadR - 20, 0, 255);
      analogWrite(pinLED, intensidadR);
    }
    if (pinLED == PIN_LED_GREEN) {
      intensidadG = constrain(intensidadG - 20, 0, 255);
      analogWrite(pinLED, intensidadG);
    }
    if (pinLED == PIN_LED_BLUE) {
      intensidadB = constrain(intensidadB - 20, 0, 255);
      analogWrite(pinLED, intensidadB);
    }
  }
}
```

Figura 7. Función implementada para disminuir la intensidad del LED RGB.

Recreación de la señal óptica

Para finalizar con este práctico se requiere la recreación de una señal óptica mediante otro dispositivo digital programable, este debe contar con un total de 8 pulsadores, 7 para realizar las acciones que se describieron en el programa anterior y 1 pulsador extra para poder guardar el color del led RGB para cuando esta comunicación se finalice, intencionalmente o no, al reanudarse continue desde los últimos valores almacenados.

Circuito Receptor y Emisor Infrarrojo

Para este ejercicio el receptor lo tuvimos que cambiar a otro dispositivo, en este caso un Arduino MEGA. Los pines digitales que se definieron para el LED RGB fueron el 6 para el azul (B), el 7 para el verde (G) y el 8 para el rojo (R), todos conectados a sus respectivas resistencias. También se eligió el pin 2 para la entrada de la señal del sensor infrarrojo.

En la recepción se complejiza más que el cambio de dispositivo, pero por el resto, el circuito se mantiene de la misma forma, básico y sencillo.

Ahora el emisor pasa a estar en el Arduino NANO, en el cual se le adjuntará una un led emisor infrarrojo, conectado a una resistencia de 100 Ω, asociado al pin 2 del dispositivo. Para el envió de las señales, los pulsadores que se usaron parten de una placa matricial 4x4 que cumple con la función de un teclado, a las columnas se les asocia con los pines 8, 9, 10 y 11, y para las filas los pines 4, 5, 6 y 7.

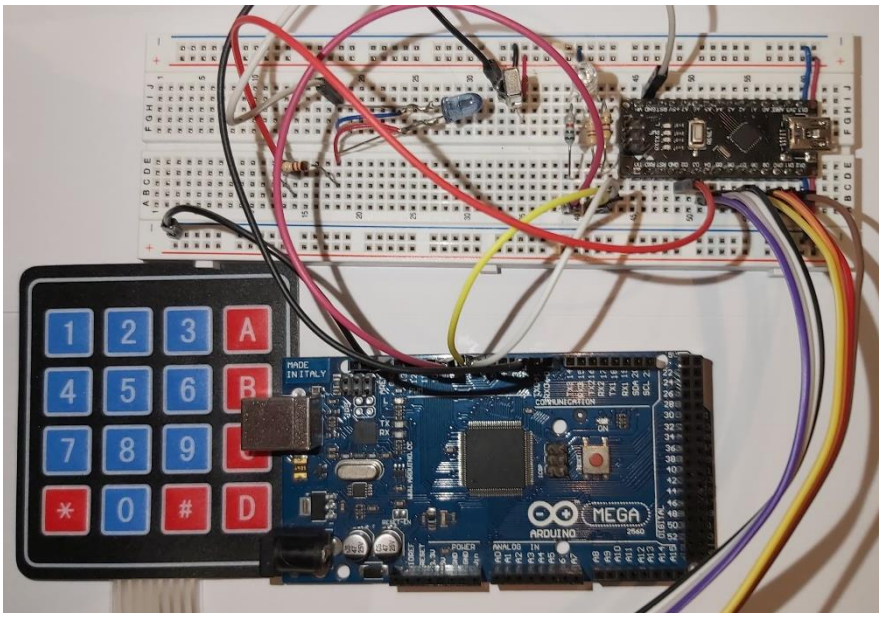

Figura 8. Vista superior de la comunicación inalámbrica.

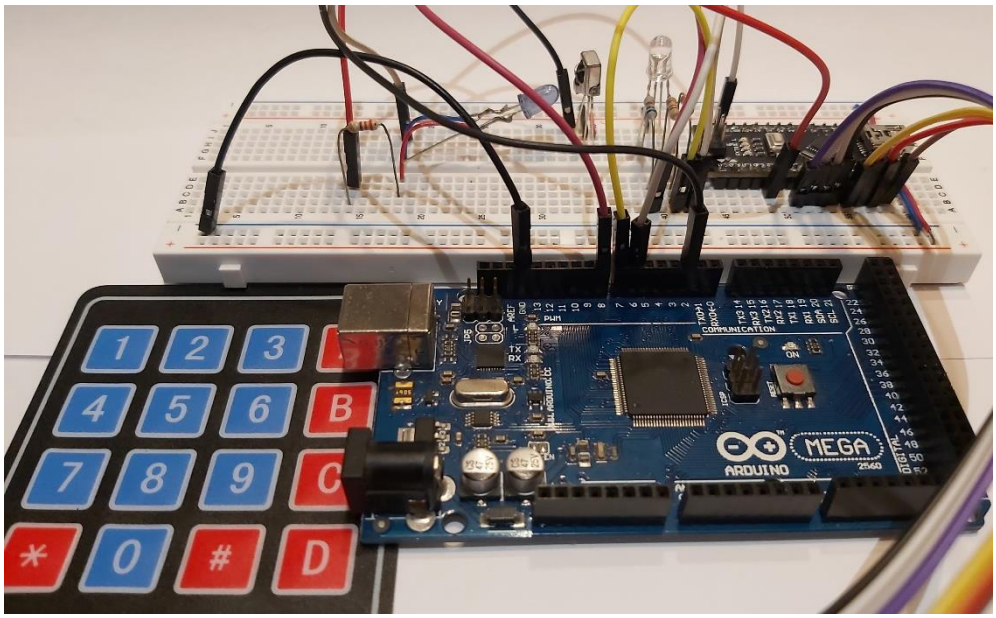

Figura 9. Vista lateral de la comunicación inalámbrica

Como se puede percibir, su elaboración física no es dificultosa ni nos proporcionó ningún error relacionado a la composición de componentes.

Programa Emisor

Para poder recrear la señal infrarroja y transmitirla por medio del led específico, implementamos la librería IRremote, la misma utiliza para el receptor.

Continuamos estableciendo los valores para las variables de la intensidad de los colores del LED RGB, todas inicializan en 255. Luego se definen tres variables similares, pero ahora con la intención de usarlas para almacenar la intensidad de los

colores para cuando el led es apagado intencionalmente o no, se reanude con las últimas intensidades guardadas por el usuario.

Para poder utilizar el teclado, primero lo tenemos que configurar, definiéndolas filas, columnas, los pines a lo que están conectadas e incluir una función con un bucle para configurar la función de los pulsadore. La forma de hacerlo no varía tanto al lado de un teclado matricial de Arduino, primero definimos los pines a los que se conectan las filas y columnas, las mismas que se mencionaron con anterioridad, seguido de la cantidad de filas y columnas que se utilizaran, estas definidas en variables *const byte.* Después se realiza una matriz para definir los valores del teclado.

```
byte pinesColumnas[] = {8, 9, 10, 11};
byte pinesFilas[] = {7, 6, 5, 4};
const byte numeroFilas = 4;
const byte numeroColumnas = 4;
int teclas[4][4] = {{1, 2, 3, 12},
  {4, 5, 6, 13},
  {7, 8, 9, 14},
  {11, 0, 10, 15}
};
```

Figura 10. Definir teclado matricial.

Para su uso eficiente se realiza una función, teclado_config(), la cual configura, mediante dos bucles *for,* las sentencias de las teclas del teclado. El primer bucle recorre los índices de las filas y define los pines de las filas como salidas (OUTPUT) en estado alto (HIGH). Mantener las filas en un estado alto permite que, cuando una tecla se presione y conecte una fila y una columna, el estado de la columna cambie a bajo (LOW). Posteriormente, el microcontrolador puede cambiar una fila a estado bajo (LOW) y verificar si alguna columna cambia de estado, lo que indicaría que se ha presionado una tecla en esa fila.

```
void teclado_config() {
  for (int nF = 0; nF < numeroFilas; nF++) {
    pinMode(pinesFilas[nF], OUTPUT);
    digitalWrite(pinesFilas[nF], HIGH);
  }

  for (int nC = 0; nC < numeroColumnas; nC++) {
    pinMode(pinesColumnas[nC], INPUT_PULLUP);
  }
}
```

Figura 11. Función para configurar teclado.

El segundo bucle recorre los índices de las columnas y establece como entradas con resistencia pull-up. Configurar las columnas como entradas con resistencia

pull-up permite detectar cuándo una columna cambia a estado bajo (LOW) al presionar una tecla, facilitando la lectura de la matriz del teclado.

Esta función inicializa los pines del Arduino necesarios para interactuar con los pulsadores, configurando las filas como salidas en estado alto y las columnas como entradas con resistencias pull-up. Esto permite detectar las pulsaciones de teclas mediante la lectura de cambios en el estado de las columnas cuando se activan las filas correspondientes.

Dicha función, llamada teclado(), emplea una función anti rebote, variable definida con anterioridad, para evitar una mala detección del pulsador. Dentro usa dos bucles, en el primero recorre todos los índices de la fila y establece el pin de la fila en estado bajo (LOW). En el segundo bucle recorre los índices de las columnas y contiene un condicional que, si el pin correspondiente a la columna está en estado bajo, se almacena el valor de la tecla presionada en una variable. Al finalizar establece en estado alto (HIGH) los pines de las filas.

```
void teclado() {
  if ( (millis() - antirebote) > 500) {
    for (int nF = 0; nF < numeroFilas; nF++) {
      digitalWrite(pinesFilas[nF], LOW);
      for (int nC = 0; nC < numeroColumnas; nC++) {
        if (digitalRead(pinesColumnas[nC]) == LOW) {
          valor = teclas[nF][nC];
          antirebote = millis();
          Serial.print("Tecla: ");
          Serial.println(teclas[nF][nC]);
        }
      }
      digitalWrite(pinesFilas[nF], HIGH);
    }
  }
}
```

Figura 12. Función teclado con función anti rebote para la detección de tecla.

Se declara una función llamada envio() la cual será la encargada de enviar el valor de la señal, que en este caso será en formato LG pero podría ser SONY o NEC. La función se elabora en base de condiciones que, al coincidir con un pulsador del teclado, realizará alguna acción. Para poder diferenciar si el led este encendido al momento de presionar la tecla se usa una variable booleana llamada *flag*, sus estados pueden ser *true* al estar encendido o *false* cuando esta apagado. La tecla 13 es en encargado de almacenar las intensidades de los colores con las variables ya mencionadas. Las teclas 1, 2 y 3 se encargan de incrementar la intensidad de un color determinado, el 1 rojo, el 2 verde y el 3 azul. Las teclas 4, 5 y 6 realizan la disminución

en la intensidad de los colores, 4 rojo, 5 verde y 6 azul. Las teclas 7, 8, 9, 10, 11, 14 y 15 no realizan ninguna acción. La tecla 12 será la encargada de encender y apagar el LED RGB.

```
if (valor == 13) {
  if (flag == true) {
    irsend.sendLG(0x17, 16, 1);
    registroR = intensidadR;
    registroG = intensidadG;
    registroB = intensidadB;
  }
}
```

Figura 13. Condicional para guardar color del LED RGB.

```
if (valor == 1) {
  if (flag == true) {
    irsend.sendLG(0xB, 16, 1);
    intensidadR = constrain(intensidadR + 20, 0, 255);
  }
}

if (valor == 2) {
  if (flag == true) {
    irsend.sendLG(0xC, 16, 1);
    intensidadG = constrain(intensidadG + 20, 0, 255);
  }
}

if (valor == 3) {
  if (flag == true) {
    irsend.sendLG(0xD, 16, 1);
    intensidadB = constrain(intensidadB + 20, 0, 255);
  }
}
```

Figura 14. Incremento de la intensidad de colores.

```
if (valor == 4) {
  if (flag == true) {
    irsend.sendLG(0xE, 16, 1);
    intensidadR = constrain(intensidadR - 20, 0, 255);
  }
}

if (valor == 5) {
  if (flag == true) {
    irsend.sendLG(0xF, 16, 1);
    intensidadG = constrain(intensidadG - 20, 0, 255);
  }
}

if (valor == 6) {
  if (flag == true) {
    irsend.sendLG(0x10, 16, 1);
    intensidadB = constrain(intensidadB - 20, 0, 255);
  }
}
```

Figura 15. Disminución de la intensidad de colores.

```
if (valor == 12) {
  if (flag == false) {
    irsend.sendLG(0x16, 16, 1);


    irsend.sendLG(0x28, registroR, 1);


    irsend.sendLG(0x32, registroG, 1);


    irsend.sendLG(0x3C, registroB, 1);
    flag = true;
  }
  else {
    irsend.sendLG(0x5C, 16, 1);
    flag = false;
  }
}
```

Figura 16. Condicionales para encender y apagar LED RGB.

En el void setup() colocaremos la función teclado_config() para configurar los pulsadores y la función irsend.begin(2) que pertenece a la librería IRremote para definir el pin donde se conecta el emisor.

Para terminar con el código del programa, en el bucle principal void loop(), solo se hace el llamado a las funciones teclado() y envio().

Programa Receptor

A diferencia de la instancia anterior, el receptor no solo cambio de dispositivo a un Arduino MEGA, sino que también cambio el código de programa.

Se empieza incluyendo la librería <IRremote.h> y definiendo los pines del LED RGB y del sensor infrarrojo.

```
#define PIN_LED_RED 8
#define PIN_LED_GREEN 7
#define PIN_LED_BLUE 6
#define IRpin 2
```

Figura 17. Definir pines.

Luego definimos cuales son los botones que se presionaron mediante los valores recibidos. Se adjunta una imagen para mejor perspectiva y clara compresión.

```
#define CODIGO_BOTON_AON 22   //prender
#define CODIGO_BOTON_AOFF 92  //apagar
#define CODIGO_RED 40         //codigo que identifica el color rojo
#define CODIGO_GREEN 50       //codigo que identifica el color verde
#define CODIGO_BLUE 60        //codigo que identifica el color azul
#define CODIGO_BOTON_1 11     //aumentar nivel del rojo
#define CODIGO_BOTON_4 14     //disminuir el nivel de rojo
#define CODIGO_BOTON_2 12     //aumentar nivel de verde
#define CODIGO_BOTON_5 15     //disminuir nivel de verde
#define CODIGO_BOTON_3 13     //aumentar nivel de azul
#define CODIGO_BOTON_6 16     //disminuir nivel de azul
```

Figura 18. Botones definidos por valor obtenido.

Después se crean variables para poder almacenar los valores codificados, controlar los tiempos, se define una variable booleana para el control del encendido y apagado del led, y variables con valores iniciales de intensidad de los colores.

```
int state = 0;
int control = 0;
float reset = 0;
bool estado = false;
int intensidadR = 0;
int intensidadG = 0;
int intensidadB = 0;
```

Figura 19. Variables utilizadas.

En el void setup() usamos la sentencia IrReceiver.begin(IRpin), perteneciente de la librería usada, para poder establecer como entrada de señal infrarroja al pin definido al principio. También configuramos los pines del LED RGB como salidas (OUTPUT).

En void loop() realizamos la tarea de control para verificar si recibimos una señal. Empezamos con un condicional *if* con la sentencia IrReceiver.decode() para llevar

a cabo lo antes mencionado. En caso de ser positivo almacenaremos en la variables "state" el valor de la señal recibida codificada mediante IrReceiver.decodedIRData.address. También se usa IrReceiver.decodedIRData.command para decodifica el tamaño de la señal mandada y la guarda en la variable "control".

```
if (IrReceiver.decode()) {
  state = IrReceiver.decodedIRData.address; //
  control = IrReceiver.decodedIRData.command;
```

Figura 20. Sentencias para decodificar la señal recibida.

A siguiente se usa otro condicional con la variable del botón de encendido para que, si la variable booleana se encuentre en *true*, envié 0 al LED RGB y de esta forma apagarlo cambiando a su vez el estado a *false.*

```
if (state == CODIGO_BOTON_AON) {
  estado = true;
}

if ( state == CODIGO_BOTON_AOFF) {
  analogWrite(PIN_LED_RED, 0);
  analogWrite(PIN_LED_GREEN, 0);
  analogWrite(PIN_LED_BLUE, 0);
  estado = false;
}
```

Figura 21. Condicional para pagar el LED RGB.

Mediante una combinación de condicionales se realiza una función para guardar la señal recibida desde el emisor para prender el control en el valor guardado. También se usa otros condicionales donde dependiendo del valor recibido se llama a una u otra función.

```
if (estado == true) {
  if (state == CODIGO_RED) {
    intensidadR = control;
    analogWrite(PIN_LED_RED, intensidadR);
  }

  if (state == CODIGO_GREEN) {
    intensidadG = control;
    analogWrite(PIN_LED_GREEN, intensidadG);
  }

  if (state == CODIGO_BLUE) {
    intensidadB = control;
    analogWrite(PIN_LED_BLUE, intensidadB);
  }
}
```

Figura 22. Función para guardar los valores de recibidos.

```
if (state == CODIGO_BOTON_1) {
  aumentarIntensidadColor(PIN_LED_RED);
}
if (state == CODIGO_BOTON_4) {
  disminuirIntensidadColor(PIN_LED_RED);
}
if (state == CODIGO_BOTON_2) {
  aumentarIntensidadColor(PIN_LED_GREEN);
}
if (state == CODIGO_BOTON_5) {
  disminuirIntensidadColor(PIN_LED_GREEN);
}
if (state == CODIGO_BOTON_3) {
  aumentarIntensidadColor(PIN_LED_BLUE);
}
if (state == CODIGO_BOTON_6) {
  disminuirIntensidadColor(PIN_LED_BLUE);
}
```

Figura 23. Acciones a realizar dependiendo del botón.

Para terminar, fuera del bucle principal, se definen las funciones para incrementar y disminuir las intensidades de los colores. Estas no serán explicadas debido a que son las mismas que se usaron en la segunda parte de este trabajo.

Problemas a tener en cuenta

En ocasiones puede que la comunicación no sea óptima debido a que no llega la señal del emisor infrarrojo, por eso, es necesario tener en cuenta el alcance máximo de la comunicación inalámbrica. Esta puede variar significativamente según varios factores (potencia del emisor, sensibilidad del receptor, condiciones ambientales, obstáculos, etc.), pero típicamente se encuentra en el rango de 5 a 10 metros en condiciones ideales. Para optimizar el alcance, es importante usar componentes de alta calidad, minimizar las interferencias y asegurar una línea de visión clara entre el emisor y el receptor.

Otra situación que puede ocurrir es que el control y el dispositivo emisor transmitan a la vez, si ambos comparten el mismo protocolo entre sí y con el receptor, este último podría tomar la señal de cualquiera de los dos. En nuestro caso, el dispositivo emisor y receptor trabajan con protocolo LG, y el control de Arduino lo hace con protocolo NEC, por lo que nunca interferirá en la comunicación.

En caso de que compartan el mismo protocolo, el control remoto podría cambiar los colores, y encender y apagar el LED RGB. Es una situación molesta pero el control no podrá guardar ningún color, por lo que, si la comunicación se reinicia o se apaga, volverá al último color que el dispositivo emisor ha guardado.

Otro escenario que se puede plantear es que la comunicación sea interrumpida o se corte durante el envió de los datos. La señal transmitida no será recibida correctamente por el receptor, lo que puede resultar en datos incompletos o incorrectos, por eso se puede implementar diferentes métodos para verificar o reenviar los datos. Se puede implementar un mecanismo de reintento en el software para enviar nuevamente la señal si no se recibe una confirmación de recepción. Utilizar códigos de verificación (como CRC) para asegurar que los datos recibidos sean correctos, si no lo son, pedir la retransmisión. Implementar un protocolo de comunicación con acuse de recibo, el receptor envía un mensaje de confirmación al emisor al recibir correctamente la señal.

Conclusión y agradecimiento

Como conclusión podemos decir que las primeras dos instancias de este práctico no nos brindar mayores dificultades, eran sencillas y se encontraban mucho material de como armar y programar un receptor de señales infrarrojas, pero la parte de emisor fue compleja de realizar, el material no es tan accesible en internet y no muchos lo explican de tal manera como hacen con el receptor. Esto causo que nos atrasemos por demás y tengamos que realizar un cambio de dispositivos como se aclaró en el informe.

Agradecemos de antemano cualquier intervención que puedan ayudar a mejorar el escrito y/o el práctico y también extiendo unas disculpas por la demora a la fecha de entrega.

Bibliografía

Johann Perez E. (03 de julio 2021). Usando el Sensor Infrarrojo con Arduino - Capítulo #48. https://www.youtube.com/watch?v=w7sYxH1W1GE

Johann Perez E. (21 de julio 2021). Controlando LED RGB con el Sensor Infrarrojo y Arduino - Capítulo #50. https://www.youtube.com/watch?v=qgYrGFI-3EY&list=PLyLh25DppBIe40j3VBAslnVfs4Pz-B3ZB&index=52

Programarfacil. Control remoto IR con Arduino. https://programarfacil.com/blog/arduino-blog/mando-a-distancia-arduino/

Repositorio de Github Librería Arduino-IRremote. https://github.com/Arduino-IRremote/Arduino-IRremote/blob/master/README.md

Repositorio de GitHub con los códigos de Arduino: https://github.com/equipoPI/ComunicacionDatosTP2.git