UNIVERSIDAD DE GRANADA

Teoría de la Información y la Codificación

4º Grado en Ingeniería Informática

Guión de prácticas
Práctica 1

Plataforma Arduino para envío y recepción de información por láser

1. Requisitos....2
2. Contenido....2
3. Metodología de trabajo....2
4. Sesión 1: Introducción a Arduino....4
5. Sesión 2: Transmisión de datos por el canal....12
6. Sesión 3: Creación del código uniforme....20
7. Entrega y evaluación de la práctica....24
8. Anexo: Cuestionario de evaluación....25

© Prof. Manuel Pegalajar Cuéllar
Dpto. Ciencias de la Computación e I. A.
Universidad de Granada

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

Introducción a la codificación. Códigos uniformes. Plataforma Arduino para envío y recepción de información por láser.

1. Requisitos

Para la realización de esta práctica es necesario haber realizado el "*Seminario 1: Introducción a Arduino*".

2. Contenido

Este documento contiene los trabajos a realizar en las sesiones de laboratorio correspondientes a la práctica 1, junto con las preguntas que el alumno debe saber contestar tras la finalización de las sesiones. En este guión de prácticas se desarrollan habilidades relacionadas con la utilización de la plataforma Arduino Uno, construcción, compilación y envío de programas, transmisión de datos por puerto serie y construcción del hardware y la base software de la plataforma de envío y recepción de datos mediante láser.

Cada una de las secciones siguientes se corresponde con las diferentes sesiones de trabajo en el laboratorio (1 sesión= 2 horas presenciales), y guiarán al alumno en la construcción de la plataforma de envío y recepción de datos mediante láser usando Arduino. Finalmente, las últimas secciones contienen un conjunto de preguntas que el alumno deberá entregar al profesor, junto con la descripción del método de evaluación.

3. Metodología de trabajo

Las prácticas deberán ser realizadas por parejas de estudiantes. Con carácter previo a la realización de este cuaderno de prácticas en el laboratorio, cada estudiante deberá haber estudiado **antes de asistir a clase** el Seminario asociado al cuaderno de prácticas y, preferiblemente, también las explicaciones disponibles en este documento para la sesión de prácticas correspondiente. Si es posible, también es recomendable haber realizado previamente esquemas, diseños e incluso desarrollos que favorezcan realizar las prácticas de laboratorio de forma más fluida. **No es recomendable asistir a clase de prácticas sin haber estudiado previamente el presente cuaderno de prácticas ni el seminario asociado**, pues se corre el riesgo de no poder terminar a tiempo los trabajos requeridos. Con carácter general, **el estudiante debe tener todo el material preparado para poder dedicar las clases de prácticas íntegramente a implementación de los programas en la plataforma Arduino**.

Se recomienda seguir la planificación que se ilustra a continuación para asegurar el éxito en la realización de la práctica:

TAREAS A REALIZAR PARA LA ELABORACIÓN DE LA PRÁCTICA	
Antes de la Sesión 1 en el laboratorio	• Estudiar el Seminario 1, apartados 1) ¿Qué es Arduino?, 2) Instalación para las prácticas, 3) Comunicaciones en serie, 4) Dispositivos GPIO. • Estudiar la Sección 4 de este documento. • Instalar el IDE de Arduino en el PC. • Realizar pseudocódigo, diagramas o diseños y, si es posible, código fuente, de la tarea final de la sesión.
Sesión 1	• Implementar los códigos de ejemplo del seminario y familiarizarse con la plataforma Arduino. • Realizar los montajes requeridos en el trabajo final de sesión e implementar la funcionalidad requerida. Pruebas de funcionamiento.
Antes de la Sesión 2 en el laboratorio	• Estudiar el Seminario 1, apartados 5) Sistema de emisión/recepción por láser, 6) Transmisión de datos por el canal. • Estudiar la Sección 5 de este documento. • Realizar pseudocódigo, diagramas o diseños y, si es posible, código fuente, de la tarea final de la sesión. Verificarlos con el profesor.
Sesión 2	• Realizar los montajes requeridos en el trabajo final de sesión e implementar la funcionalidad requerida. Pruebas de funcionamiento.
Antes de la Sesión 3 en el laboratorio	• Estudiar el Seminario 1, apartados 5) Sistema de emisión/recepción por láser, 7) Creación del código uniforme, 8) pruebas finales. • Estudiar la Sección 6 de este documento. • Realizar pseudocódigo, diagramas o diseños y, si es posible, código fuente, de la tarea final de la sesión. Verificarlos con el profesor.
Sesion 3	• Realizar los montajes requeridos en el trabajo final de sesión e implementar la funcionalidad requerida. Pruebas de funcionamiento. Defensa ante el profesor.
Antes de la finalización de la entrega de la práctica	• Defensa ante el profesor. • Estudio de los temas 1 y 2 de teoría. • Elaborar las respuestas al cuestionario final de la práctica.

4. Sesión 1: Introducción a Arduino

4.1. Prerrequisitos

Para elaborar esta sesión es necesario que el alumno haya estudiado previamente el ***Seminario 1: Introducción a Arduino***, apartados:

1. ¿Qué es Arduino?
2. Instalación de Prácticas
3. Comunicaciones en Serie
4. Dispositivos GPIO

Se asume también que el estudiante ha instalado el entorno IDE de Arduino en su PC.

4.2. Puerto Serie

Las comunicaciones entre el PC y Arduino se realizan habitualmente mediante la interfaz de puerto serie. Las bibliotecas de Arduino contienen el objeto global **Serial**, configurado para trabajar sobre el primer puerto serie de cualquier placa Arduino.

Para trabajar con el puerto serie, es necesario inicializarlo en la función **setup()**, indicando como parámetro la velocidad (en baudios) que se requiere para las comunicaciones (ver Figura 1). Típicamente usaremos un valor de **9600 baudios**. Se puede obtener más información sobre cómo inicializar el puerto serie de Arduino en la API online de la biblioteca:

https://www.arduino.cc/reference/en/language/functions/communication/serial/

```
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {}
```

Figura 1: Inicialización del puerto serie.

Las funcionalidades del puerto serie que permite Arduino se implementan como métodos del objeto **Serial**. Las más importantes son las siguientes:

4.2.1. Serial.available()

Devuelve el número de bytes que hay en el buffer de lectura del puerto serie (máximo 64). Valor 0 si no hay datos para leer desde el puerto serie. La Figura 2 muestra un ejemplo de uso de **Serial.available()**.

```
int incomingByte = 0; // for incoming serial data

void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {
  // reply only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Figura 2: Ejemplo de uso de Serial.available() y de Serial.print(x)

4.2.2. Serial.print(x) / Serial.println(x)

Envía los bytes del valor **x** por el puerto serie como salida. El valor **x** puede ser numérico, cadena de caracteres (objeto **String** o array de **char**). En el caso del método **Serial.println(x)**, se añade un byte adicional como salida por el puerto serie, correspondiente al caracter de salto de línea. La Figura 2 muestra un ejemplo de uso de **Serial.print(x)** y de **Serial.println(x)**.

4.2.3. Serial.read()

Lee un byte desde el buffer de entrada del puerto serie en Arduino, y lo devuelve como salida del método. La Figura 2 muestra un ejemplo de uso de **Serial.read()**.

4.2.4. Serial.readString()

Lee una secuencia de bytes desde el buffer de entrada del puerto serie en Arduino, transforma la secuencia a cadena de caracteres y devuelve dicha cadena como un objeto de tipo **String**. y lo devuelve como salida del método. La Figura 3 muestra un ejemplo de uso de **Serial.readString()**.

Figura 3: Ejemplo de uso de Serial.readString()

4.2.5. El monitor serie

El **monitor serie** es una opción del IDE de Arduino, para permitir que podamos comunicarnos con la placa Arduino desde PC de forma simple. Se accede desde el menú ***Herramientas->Monitor Serie***, habiendo seleccionado previamente el puerto donde se encuentra conectado Arduino en el menú ***Herramientoas->Puerto***. La Figura 4 muestra el aspecto del monitor serie. En la parte superior, podemos escribir texto a enviar a la plataforma, y que Arduino captará con las funcionalidades **Serial.read()/Serial.readString()**. En la parte inferior, podemos visualizar el texto que Arduino nos envía con las funcionalidades **Serial.print** y **Serial.println**.

IMPORTANTE: Debemos asegurarnos siempre de que la velocidad del monitor serie (baudios, parte inferior del monitor serie) se ajusta a la configuración establecida en Arduino (típicamente 9600 baudios), Y QUE NO SE AÑADA AJUSTE DE LÍNEA.

Figura 4: Ilustración del monitor serie del IDE de Arduino

4.3. La clase String

La clase **String** (con **S mayúscula**) es una clase implementada en las bibliotecas de Arduino para gestión de cadenas de caracteres. La documentación de la función se encuentra disponible online en el siguiente enlace:

https://www.arduino.cc/en/Reference/StringLibrary

En las prácticas de la asignatura, usaremos la clase String para albergar cadenas de caracteres leídas desde el puerto serie, o a enviar por el mismo. Su uso es muy similar a la clase estándar **string** de C/C++. La Figura 5 muestra un ejemplo de diferentes declaraciones de variables de tipo **String**.

```
String stringOne = "Hello String";                       // using a constant String
String stringOne =  String('a');                         // converting a constant char into a String
String stringTwo =  String("This is a string");          // converting a constant string into a String
String stringOne =  String(stringTwo + " with more");    // concatenating two strings
String stringOne =  String(13);                          // using a constant integer
```

Figura 5: Ejemplo de declaración de variables de tipo String.

Arduino también permite hacer uso de cadenas de caracteres al estilo tradicional de C (arrays de tipo char). Se podrá optar por esta opción también en las prácticas, dependiendo del diseño y de las características del programa que se realice. La documentación necesaria para gestión de cadenas de caracteres se puede encontrar en la siguiente URL:

https://www.arduino.cc/reference/en/language/variables/data-types/string/

Además de las cadenas de caracteres clásicas, también se dispone de las funciones de operación sobre arrays de tipo char (strcpy, strcmp, strcat, etc.), disponibles en la **biblioteca cstring/string.h**, realizando el **#include** correspondiente en el programa **.ino**. Un ejemplo de uso de cadenas de caracteres como arrays de tipo char en Arduino se muestra en la Figura 6.

```
char *myStrings[] = {"This is string 1", "This is string 2", "This is string 3",
                     "This is string 4", "This is string 5", "This is string 6"
                    };

void setup() {
  Serial.begin(9600);
}

void loop() {
  for (int i = 0; i < 6; i++) {
    Serial.println(myStrings[i]);
    delay(500);
  }
}
```

Figura 6: Ejemplo de uso de cadenas como arrays de char

4.4. Pines GPIO

Los pines de Arduino se encuentran repartidos por toda la placa. Hay pines de diversos tipos:

- Voltaje de referencia a tierra (GND).
- Voltaje de 3.3V / 5V.
- Pines digitales de entrada/salida (GPIO).
- Pines analógicos de entrada.
- Etc.

En las prácticas de la asignatura haremos uso únicamente de los pines digitales GPIO, y de los pines de tierra (GND) y de voltaje (3.3V y 5V). Mientras que los pines de tierra y de voltaje se utilizan para alimentar eléctricamente dispositivos externos (sensores, actuadores, etc.), los pines GPIO se utilizan para comunicarnos con ellos. Pueden ser configurados como:

- **Pines de entrada:** Arduino obtiene datos de dispositivos externos (sensores).
- **Pines de salida:** Arduino escribe/envía datos a dispositivos externos (actuadores).

A continuación, mostramos las funciones de las bibliotecas de Arduino necesarias para configurar un pin como entrada o como salida, y también para enviar o recibir datos por estos pines.

4.4.1. La función *pinMode(pin, tipo)*

La función ***pinMode(pin, tipo)*** sirve para configurar un pin GPIO de la placa Arduino como entrada o como salida. Esta operación deberá realizarse al comienzo del programa, en la función **setup()**. La función tiene como entrada dos parámetros:

- **pin:** Un número entero positivo que indica el pin que vamos a configurar.
- **tipo:** Indica cómo se usará el pin. El parámetro sólo puede tener los valores **INPUT** (se usará como entrada a Arduino, se leerán datos desde el pin) y **OUTPUT** (se usará como salida desde Arduino, se escribirán datos en el pin).

Por tanto, durante las prácticas configuraremos los pines asociados a sensores (fotorreceptor) como **INPUT** y los pines asociados a actuadores (láser) como **OUTPUT**.

IMPORTANTE: No se deben escribir datos sobre pines configurados como INPUT. Se corre el riesgo de dañar la placa y también el sensor que se ha conectado al pin.

La documentación oficial de la función ***pinMode(pin, tipo)*** se encuentra en la siguiente URL:

https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/

La Figura 7 muestra un ejemplo del uso de la función, donde se establece el pin digital 13 como salida (se enviarán/escribirán datos por él).

```
void setup() {
  pinMode(13, OUTPUT);    // sets the digital pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH); // sets the digital pin 13 on
  delay(1000);            // waits for a second
  digitalWrite(13, LOW);  // sets the digital pin 13 off
  delay(1000);            // waits for a second
}
```

Figura 7: Ejemplo de uso de pinMode y digitalWrite

4.4.2. La función *digitalWrite(pin, valor)*

La función ***digitalWrite(pin, valor)*** de la biblioteca de Arduino sirve para activar (poner voltaje alto a 3.3V) o desactivar (poner voltaje bajo a 0V) pines digitales de la placa Arduino. La función tiene como entrada dos parámetros:

- **pin:** Un número entero positivo que indica el pin sobre el que vamos a escribir.
- **valor:** Indica qué valor (voltaje) se escribirá en el pin. El parámetro sólo puede tener los valores **HIGH** (voltaje alto, 3.3V) y **LOW** (voltaje bajo, 0V).

Esta función sólo debe ser usada sobre pines configurados antes como salida (OUTPUT), y nunca sobre pines configurados como entrada (INPUT).

La documentación sobre la función ***digitalWrite(pin, valor)*** se encuentra disponible en el enlace siguiente:

https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

La Figura 7 muestra un ejemplo del uso de esta función en un programa de Arduino, escribiendo valores **HIGH** y **LOW** sobre el pin **13**, **configurado previamente como salida**.

4.4.3. La función *digitalRead(pin)*

La función ***digitalRead(pin)*** de la biblioteca de Arduino sirve para muestrear el valor que un dispositivo externo (sensor, etc.) está proporcionando a Arduino como entrada. La función tiene como entrada un único parametro: Un número entero positivo que indica el pin del que se desea conocer su valor actual. Proporciona como salida un valor **HIGH** o **LOW**:

- ***digitalRead(pin)*** devolverá valor **LOW** si el pin no tiene voltaje desde el dispositivo externo conectado al pin ***pin***.
- ***digitalRead(pin)*** devolverá valor **HIGH** si el pin tiene voltaje desde el dispositivo externo conectado al pin ***pin***.

La documentación oficial de la función se encuentra disponible en el siguiente enlace:

https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/

La Figura 8 muestra un ejemplo de lectura de un pin de entrada configurado como el **pin 7**.

```
int ledPin = 13;  // LED connected to digital pin 13
int inPin = 7;    // pushbutton connected to digital pin 7
int val = 0;      // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the digital pin 13 as output
  pinMode(inPin, INPUT);    // sets the digital pin 7 as input
}

void loop() {
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);  // sets the LED to the button's value
}
```

Figura 8: Ejemplo de uso de digitalRead y digitalWrite

4.5. Trabajo final de sesión

4.5.1. Material necesario

- PC con IDE Arduino instalado y puerto USB.
- 1 placa Arduino Uno.
- 1 cable USB de conexión Arduino – PC.
- 1 Módulo receptor 2PCS láser no modulador sensor Tubo.
- 2 diodos LED (preferiblemente de color diferente).
- 7 Cables conectores macho/hembra.

4.5.2. Montaje

1. Conecte el polo positivo de un LED al PIN DIGITAL 8 de Arduino, y el polo negativo a

algún PIN GND.
2. Conecte el polo positivo del otro LED al PIN DIGITAL 9 de Arduino, y el polo negativo a algún PIN GND.
3. Conecte el pin VCC del sensor fotorreceptor al pin de salida de 3.3V de Arduino, el pin GND del sensor a algún PIN GND de Arduino, y el pin OUT del sensor al PIN DIGITAL 7 de Arduino.

4.5.3. Comportamiento esperado

Escriba un programa para Arduino que implemente el siguiente comportamiento:

- El usuario podrá enviar mensajes "ON" u "OFF" a Arduino a través del monitor serie. Por defecto, se asume que el sistema está a ON.
- Si el sistema está en estado ON:
  - Por defecto, el LED del PIN 8 deberá estar apagado y el LED del PIN 9 encendido.
  - Si el sensor fotorreceptor detecta luz, entonces se apagará el LED del PIN 9 y se encenderá el LED del PIN 8.
  - Si el sensor fotorreceptor no detecta luz, se volverá al comportamiento por defecto.
- Si el sistema está en estado OFF: No se encenderá ningún LED independientemente de qué valor lea el fotorreceptor.
- Este comportamiento debe poder ser realizado por Arduino múltiples veces sin necesidad de reiniciar la placa.

5. Sesión 2: Transmisión de datos por el canal

5.1. Prerrequisitos

Para elaborar esta sesión es necesario que el alumno haya estudiado previamente el ***Seminario 1: Introducción a Arduino***, apartados:

1. Sistema de emisión/recepción por láser
2. Transmisión de datos por el canal

Se asume también que el estudiante ha finalizado con éxito la sesión 1 de prácticas.

5.2. Material necesario

- 2 PCs con IDE Arduino instalado y puerto USB (PC emisor y PC receptor).
- 2 placas Arduino Uno (1 placa para el PC emisor, 1 placa para el PC receptor).
- 2 cables USB de conexión Arduino – PC (1 cable para cada Arduino, emisor y receptor).
- 1 Módulo receptor 2PCS láser no modulador sensor Tubo.
- 1 Módulo emisor láser KY-008.
- 6 Cables conectores macho/hembra.
- 1 kit de soporte para el emisor y receptor.

5.3. Montaje del soporte para el emisor y receptor

Al finalizar la práctica, se deberá poder enviar información a través de láser entre 2 plataformas Arduino diferentes. El sistema de comunicaciones deberá seguir el esquema se que muestra en la Figura 9: Un Arduino emisor codificará información recibida por puerto serie desde el PC emisor, y transmitirá los datos por láser. Por otra parte, otro Arduino receptor muestreará el canal con el sensor fotorreceptor, y captará la secuencia de palabras del código generado, las decodificará, y las enviará a PC por el puerto serie.

Para poder construir con éxito todo este sistema, en primer lugar es necesario disponer de un método fiable para transferir y recibir datos por el canal. Esto conlleva que:

- El láser emisor y el fotorreceptor estén correctamente alineados.

- Exista un protocolo de comunicaciones válido implementado coherentemente tanto en el Arduino emisor como en el Arduino receptor.

En esta sesión de prácticas nos centramos en estos aspectos del sistema, implementando el método para transmitir datos por el canal.

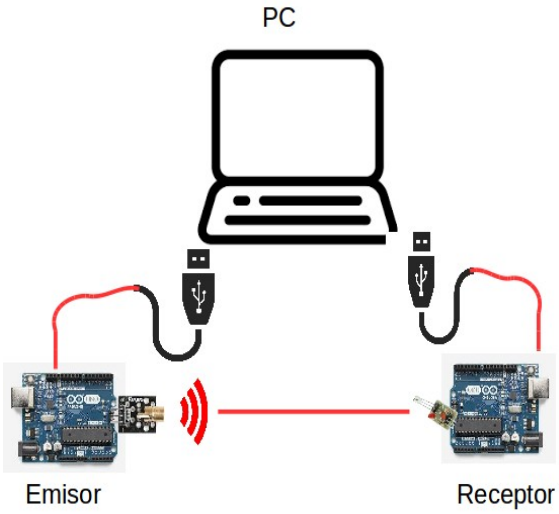

Figura 9: Esquema de montaje del emisor y del receptor

El soporte del montaje para el envío de datos entre el Arduino emisor y el Arduino receptor permitirá que el láser y el fotorreceptor estén correctamente alineados. Este soporte está compuesto por 3 piezas:

- **Base** (Figura 10). Es una placa de plástico con pestañas a cada lado, para sujetar los soportes del emisor y del receptor de modo que estén alineados.
- **Soporte emisor** (Figura 11). Es una pieza de plástico con hendiduras para introducir el emisor láser, permitiendo que la luz pueda ser enviada a través del orificio de salida.
- **Soporte receptor** (Figura 12). Es una pieza de plástico con hendiduras para introducir el fotorreceptor, permitiendo que la luz direccional del láser pueda entrar al interior e incidir en el fotorreceptor.

Figura 10: Base del soporte

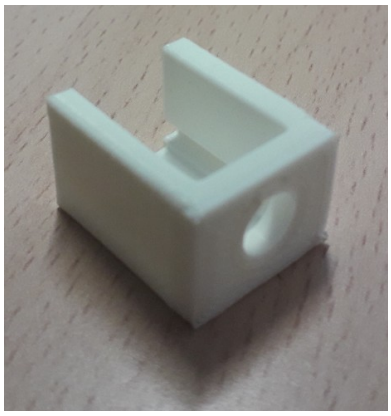

Figura 11: Soporte del emisor

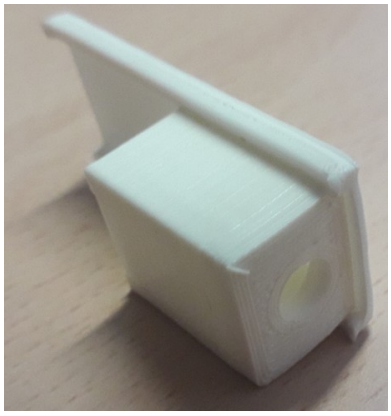

Figura 12: Soporte del receptor

Se deberá montar el soporte, junto con los módulos emisor y receptor de láser según se indica en la Figura 13. **IMPORTANTE: Debemos asegurarnos de que la luz del láser incide correctamente en el cristal del módulo fotorreceptor**.

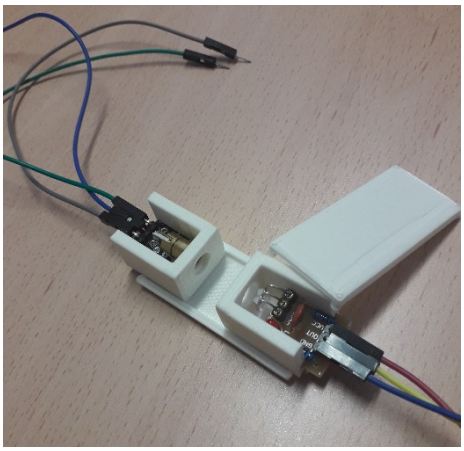

Figura 13: Montaje del soporte

5.4. Conexiones

Una vez realizado el montaje del soporte para la transmisión, se realizarán las conexiones del emisor láser y del fotorreceptor según se indica a continuación:

- **Emisor láser:**
  - Pin **S** del láser al PIN DIGITAL 8 del Arduino emisor.
  - Pin **–** del láser a algún PIN GND del Arduino emisor.
  - El pin restante del láser al PIN de 5V de salida del Arduino emisor.
- **Fotorreceptor:**
  - Pin **GND** del fotorreceptor a algún PIN GND del Arduino receptor.
  - Pin **VCC** del fotorreceptor al PIN de 3.3V de salida del Arduino receptor.
  - Pin **OUT** del fotorreceptor al PIN DIGITAL 7 del Arduino receptor.

El esquema de las conexiones a realizar se muestra en la Figura 14.

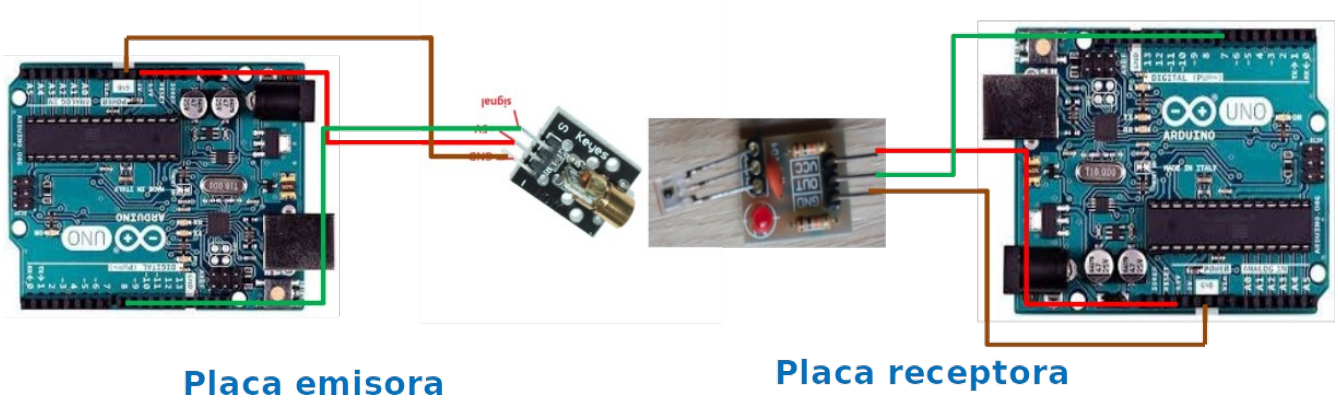

Figura 14: Conexiones de las placas emisora y receptora

5.5. Envío y recepción de bits

5.5.1. Arduino emisor

El envío de bits "0" o "1" por el canal deberá realizarse mediante una función ***void sendBit(int value)***, que tenga como entrada el valor a enviar "0" o "1" y que envíe por láser los pulsos necesarios para transmitir el "bit 0" o el "bit 1". Previamente, deberá haberse definido una constante global (o **#define** en su defecto) de nombre **TAU**, que tenga como valor el número de milisegundos de duración de un pulso de láser activo o inactivo.

Tras definir el valor **TAU**, un bit "0" o "1" se enviará mediante el esquema siguiente:

- **Bit 0**: Se enviará un pulso **HIGH** seguido de un pulso **LOW** por el láser (ver Figura 15).
- **Bit 1**: Se enviarán dos pulsos **HIGH** seguido de un pulso **LOW** por el láser (ver Figura 16).

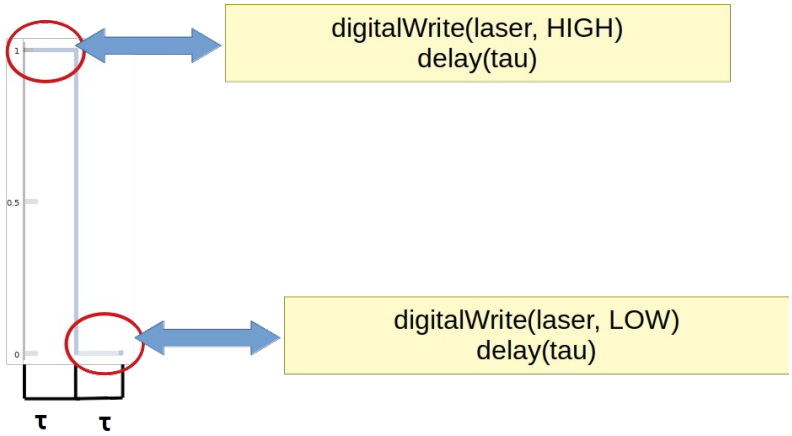

Figura 15: Envío de "bit 0" por el canal

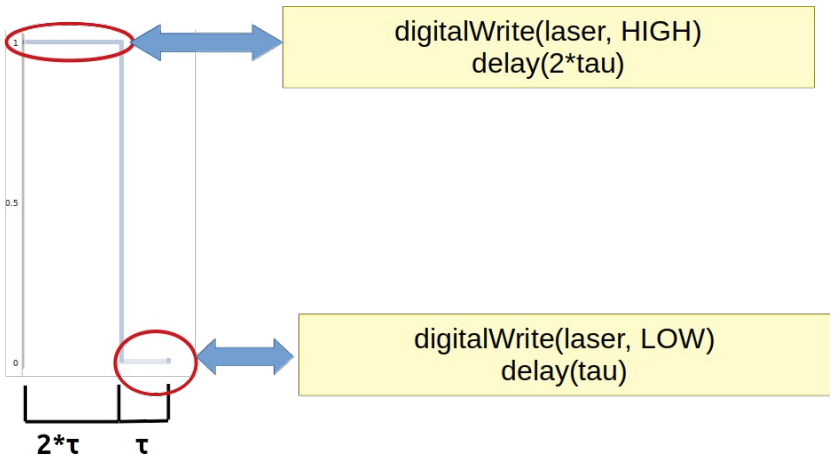

Figura 16: Envío de "bit 1" por el canal

Por ejemplo, para enviar la secuencia de bits "00101" se debería establecer la secuencia de pulsos que se muestra en la Figura 17. De este modo, la implementación de la función ***void sendBit(int value)*** deberá seguir el siguiente esquema:

1. Establecer a HIGH el pin del láser
2. Si value=0, esperar un tiempo **TAU** ms.
3. En otro caso, esperar un tiempo **2*TAU** ms.
4. Establecer a LOW el pin del láser
5. Esperar un tiempo **TAU** ms.

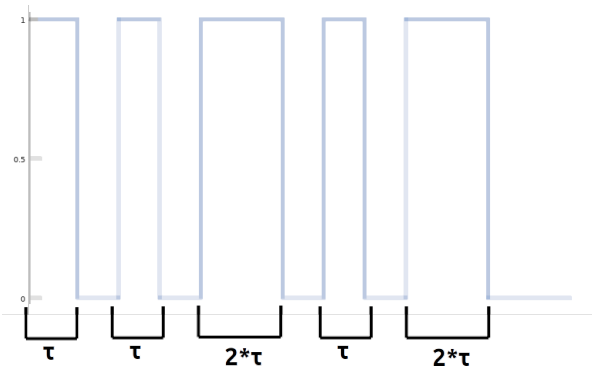

Figura 17: Secuencia de pulsos de activación del láser para enviar 00101

5.5.2. Arduino receptor

La recepción de un bit "0" o "1" en el Arduino receptor, atendiendo al modelo de transmisión implementado en el emisor, requiere que el canal se muestree a intervalos periódicos de **TAU/n** ms, donde **n>=2** para que se cumplan las condiciones del Teorema de Muestreo de Nyquist. **El valor de TAU escogido deberá ser el mismo que el valor de TAU en el emisor**.

Por ejemplo, para un valor **n=2**, si el emisor emite la información de la Figura 17, un posible muestreo sería el que se ilustra en la Figura 18.

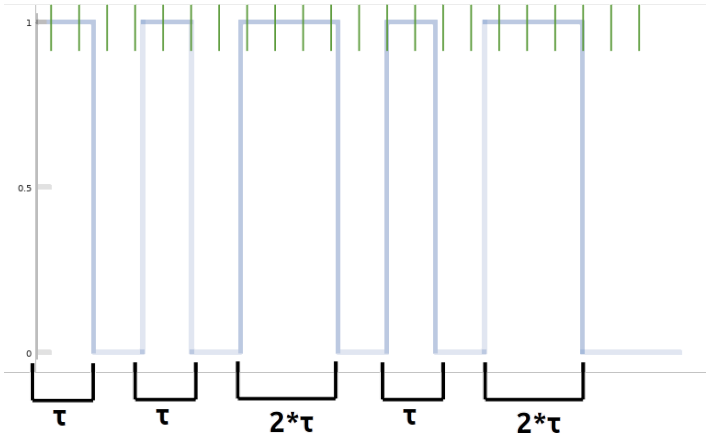

Figura 18: Ejemplo de muestreo con periodo TAU/2 en el receptor

De este modo, para recibir en el receptor el bit "0" o "1" enviado por el emisor, será necesario declarar dos constantes globales **TAU** y **n** (o **#define** en su defecto), con el valor de **TAU** establecido en el emisor y un valor de **n** igual o superior a 2. La recepción del bit se implementará en una función ***int recvBit()***, que devolverá:

- **Valor 0** si el emisor se encuentra emitiendo y ha enviado el bit 0. Se sabrá que el emisor está enviando el bit 0 si el sensor de luz detecta entre 1 y **n** valores consecutivos a **HIGH**, muestreados con un periodo de **TAU/n**.
- **Valor 1** si el emisor se encuentra emitiendo y ha enviado el bit 1. Se sabrá que el emisor está enviando el bit 0 si el sensor de luz detecta entre 1 y **n** valores consecutivos a **HIGH**, muestreados con un periodo de **TAU/n**.
- **Valor -1** si el emisor no se encuentra emitiendo. Se sabrá si el emisor no está enviando datos si se muestrean más de **n** valores LOW consecutivos del sensor de luz con un periodo **TAU/n**.

La implementación de la función ***int recvBit()*** deberá seguir el siguiente esquema:

1. Establecer contador=0
2. Mientras (contador <= n y lectura del sensor = LOW)
    1. Incrementar el contador en 1 unidad
    2. Esperar **TAU/n** ms.
3. Si (contador > n), devolver -1
4. En otro caso:
    1. Establecer contador=0
    2. Mientras (lectura del sensor = HIGH)
        1. Incrementar el contador en 1 unidad
        2. Esperar **TAU/n** ms.
    3. Si (contador >n) devolver 1; en otro caso devolver 0

5.6. Trabajo final de sesión

5.6.1. Prerrequisitos

Para realizar el trabajo final de la sesión es necesario tener implementadas sin errores las funciones ***void sendBit(int)*** e ***int recvBit()*** en código fuente de sendos programas **Emisor** y **Receptor**. También es requisito tener realizado el montaje y las conexiones de las placas Arduino emisor y receptor según se indica en la Figura 14, siendo estas funcionales y estando conectadas a un PC emisor y otro PC receptor, respectivamente.

5.6.2. Comportamiento esperado

Se asume el esquema de emisión/recepción de datos por láser explicado en la Figura 9, Sección 5.3.

Con respecto al emisor: Se debe desarrollar un programa y enviarlo a Arduino, con la siguiente funcionalidad:

- El usuario podrá introducir una cadena de caracteres de longitud 10 en el monitor serie. Esta cadena sólo podrá contener caracteres '0' o '1'. Ejemplo: "1001111101".
- Arduino deberá comprobar si hay algún dato en el buffer de entrada por el puerto serie. Si lo hay, recogerá una cadena de caracteres con el formato especificado en el punto anterior. A continuación, cada caracter '1' o '0' se enviará por láser con los correspondientes bits 1 ó 0, respectivamente, usando la función implementada ***sendBit***.
- Este comportamiento deberá poder ser repetido múltiples veces sin necesidad de reinicializar la placa Arduino.

Con respecto al receptor: Se debe desarrollar un programa y enviarlo a Arduino, con la siguiente funcionalidad:

- El programa Arduino tendrá un buffer de caracteres, inicializado a vacío.
- El programa Arduino estará comprobando constantemente si se están recibiendo datos por el láser, usando la función implementada ***recvBit***. Mientras no se reciban datos, no hará nada.
- Cuando el programa Arduino detecte que el emisor está enviando datos por láser, recogerá los bits existentes en el canal mediante la función ***recvBit*** implementada, y guardará los datos el el buffer.
- Cuando finalmente el programa Arduino detecte que el emisor ha parado de enviar datos por láser, enviará los bits del buffer por el puerto serie a PC, como una cadena de caracteres conteniendo "0"s y "1"s.
- El usuario podrá visualizar el monitor serie de Arduino, y comprobará que la cadena recibida es la misma que la introducida por el usuario en el PC emisor.
- Este comportamiento deberá poder ser repetido múltiples veces sin necesidad de reinicializar la placa Arduino.

6. Sesión 3: Creación del código uniforme

6.1. Prerrequisitos

Para elaborar esta sesión es necesario que el alumno haya estudiado previamente el ***Seminario 1: Introducción a Arduino***, apartados:

1. Creación del código uniforme
2. Pruebas finales

Se asume también que el estudiante ha finalizado con éxito la sesión 2 de prácticas, y que posee la implementación correcta de las funciones ***sendBit*** y ***recvBit*** requeridas en dicha sesión.

6.2. Material necesario

- 2 PCs con IDE Arduino instalado y puerto USB (PC emisor y PC receptor).
- 2 placas Arduino Uno (1 placa para el PC emisor, 1 placa para el PC receptor).
- 2 cables USB de conexión Arduino – PC (1 cable para cada Arduino, emisor y receptor).
- 1 Módulo receptor 2PCS láser no modulador sensor Tubo.
- 1 Módulo emisor láser KY-008.
- 6 Cables conectores macho/hembra.
- 1 kit de soporte para el emisor y receptor.

6.3. Montaje del soporte para el emisor y receptor

Se realizará el montaje del sistema de emisión/recepción de datos por láser explicado en la Sección 5.3.

6.4. Codificación del alfabeto de la fuente

Asumimos que la fuente será el usuario que introduce datos por el monitor serie del sistema emisor de datos. El alfabeto de la fuente estará compuesto por:

- Todas las letras del alfabeto (salvo la ñ), en mayúsculas.
- Los signos de puntuación punto (.), coma (,), punto y coma (;), espacio ( ), y un símbolo adicional no visible que se usará como símbolo de fin de mensaje.

En total, el alfabeto de la fuente estará formado por 31 símbolos.

6.4.1. Trabajo a realizar

En primer lugar, **el trabajo a realizar en este apartado** requiere diseñar un código uniforme capaz de codificar en un alfabeto binario formado por "0"s y "1"s el alfabeto de la fuente. Se deberá crear una tabla con dos columnas: **alfabeto de la fuente** y **palabra del código uniforme**, que contenga cada símbolo del alfabeto de la fuente y la palabra del código correspondiente, en binario.

En segundo lugar, **se requiere escoger o diseñar una estructura de datos** que permita, en el emisor, codificar de forma eficiente un símbolo del alfabeto de la fuente en una palabra del código uniforme requerido.

En tercer lugar, **se requiere escoger o diseñar una estructura de datos** que permita, en el receptor, decodificar de forma eficiente una palabra del código uniforme en un símbolo del alfabeto de la fuente.

Como estructuras de datos básicas, se puede pensar en un array estático de pares de datos, una matriz estática de 2 columnas, una tabla Hash, etc.

6.5. Implementación del código

6.5.1. Trabajo a realizar en el emisor

En el sistema emisor, se deberá incorporar al código del programa emisor de la sesión anterior una función ***codifica***, que tenga como entrada una secuencia de símbolos de la fuente. La salida de esta función deberá ser una cadena de “0”s y “1”s, finalizada con los bits de la palabra del código asociada con el símbolo de fin de mensaje.

RECOMENDACIÓN: Puede resultar más cómodo, de cara a posible depuración de errores, no implementar la función *codifica* directamente en el IDE de Arduino. En su lugar, se recomienda implementarla primero en el IDE de programación de C/C++ habitual y comprobar su correcto funcionamiento. Una vez realizadas estas comprobaciones, se podrá incorporar al IDE de Arduino para el desarrollo de la práctica.

6.5.2. Trabajo a realizar en el receptor

En el sistema receptor, se deberá incorporar al código del programa receptor de la sesión anterior una función ***decodifica***, que tenga como entrada una secuencia de bits leídos desde el canal mediante la función **recvBit**. La salida de esta función deberá ser una cadena de caracteres conteniendo los símbolos de la fuente decodificados, sustituyendo el caracter de fin de mensaje por el caracter '**\0**'.

RECOMENDACIÓN: Puede resultar más cómodo, de cara a posible depuración de errores, no implementar la función *decodifica* directamente en el IDE de Arduino. En su lugar, se recomienda implementarla primero en el IDE de programación de C/C++ habitual y comprobar su correcto funcionamiento. Una vez realizadas estas comprobaciones, se podrá incorporar al IDE de Arduino para el desarrollo de la práctica.

6.6. Trabajo final de sesión.

6.6.1. Prerrequisitos

Para realizar el trabajo final de la sesión es necesario tener implementadas sin errores las funciones ***sendBit y codifica*** en el programa emisor, y las funciones ***recvBit y decodifica*** en el programa receptor. También es requisito tener realizado el montaje y las conexiones de las placas Arduino emisor y receptor según se indica en la Figura 14, siendo estas funcionales y estando conectadas a un PC emisor y otro PC receptor, respectivamente.

6.6.2. Comportamiento esperado

Se asume el esquema de emisión/recepción de datos por láser explicado en la Figura 9, Sección 5.3.

Con respecto al emisor: Se debe desarrollar un programa y enviarlo a Arduino, con la siguiente funcionalidad:

- El usuario podrá introducir una cadena de caracteres de longitud máxima 60 caracteres, conteniendo símbolos del alfabeto castellano (salvo la ñ) en mayúscula, junto con los símbolos de puntación *punto* (.), *coma* (,), *punto y coma* (;) y *espacio* ( ).
- El programa del Arduino emisor deberá comprobar si hay algún dato en el buffer de entrada por el puerto serie. Si lo hay, recogerá una cadena de caracteres con el formato especificado en el punto anterior. A continuación, codificará los símbolos de la cadena en una secuencia de bits utilizando la función ***codifica***, que añadirá los bits de la palabra del código de fin de mensaje al final de la secuencia. Seguidamente, el programa recorrerá la secuencia de bits resultante, y enviará por láser los bits 1 ó 0 codificados, usando la

función implementada ***sendBit***.

- Este comportamiento deberá poder ser repetido múltiples veces sin necesidad de reinicializar la placa Arduino.

Con respecto al receptor: Se debe desarrollar un programa y enviarlo a Arduino, con la siguiente funcionalidad:

- El programa Arduino tendrá un buffer de caracteres, inicializado a vacío.
- El programa Arduino estará comprobando constantemente si se están recibiendo datos por el láser, usando la función implementada ***recvBit***. Mientras no se reciban datos, no hará nada.
- Cuando el programa Arduino detecte que el emisor está enviando datos por láser, recogerá los bits existentes en el canal mediante la función ***recvBit*** implementada, y guardará los datos el el buffer.
- Cuando finalmente el programa Arduino detecte que el emisor ha parado de enviar datos por láser, decodificará la secuencia de bits en una secuencia de símbolos con la función ***decodifica***. Los datos decodificados serán enviados por el puerto serie a PC, como una cadena de caracteres.
- El usuario podrá visualizar el monitor serie de Arduino, y comprobará que la cadena recibida es la misma que la introducida por el usuario en el PC emisor.
- Este comportamiento deberá poder ser repetido múltiples veces sin necesidad de reinicializar la placa Arduino.

7. Entrega y evaluación de la práctica

La práctica debe ser resuelta **por parejas, y contribuirá con 1.5 puntos sobre 5 (prácticas) a la calificación global de la asignatura.** La evaluación principal será continua, y se realizará por sesiones o al finalizar la práctica. Se evaluarán los siguientes items:

- Funcionamiento del ejercicio final de la sesión 1: 1 punto.
- Funcionamiento del ejercicio final de la sesión 2: 3 puntos.
- Funcionamiento del ejercicio final de la sesión 3: 3 puntos.

El funcionamiento de los ejercicios anteriores se evaluará en el laboratorio. Consistirá en una inspección y entrevista del profesor con los estudiantes, donde se tratarán y se deberán resolver las siguientes cuestiones:

- Prueba del correcto funcionamiento esperado del sistema (mitad de la puntuación).
- Explicación de cómo se ha implementado cada parte (mitad de la puntuación).

Adicionalmente, al finalizar la práctica se deberá presentar al profesor, mediante la entrega habilitada en la plataforma docente web de la asignatura, la siguiente documentación:

- Código fuente final de cada una de las sesiones de la práctica. El código fuente se organizará por carpetas, denominadas "S1" (código de la sesión 1), "S2" (código de la sesión 2), "S3" (código de la sesión 3). **La no entrega del código supondrá la calificación numérica 0 en el apartado correspondiente, independientemente de la defensa realizada en clase de laboratorio.**
- **Entrega del cuestionario del anexo de este documento**, en formato PDF. Todas las preguntas del cuestionario se valorarán con la misma calificación. La calificación total del cuestionario en la práctica es de 3 puntos.

8. Anexo: Cuestionario de evaluación

Nombre, apellidos y email del estudiante 1:	
Nombre, apellidos y email del estudiante 2:	

CUESTIONARIO
Calcule la capacidad del canal creado en la práctica.
Asumiendo que todos los símbolos del alfabeto de la fuente son equiprobables, calcule la entropía de la fuente.
¿Qué es un código uniforme?

¿Cuánto se tardaría, en promedio, en enviarse 4 símbolos cualesquiera por el canal? ¿Y como mínimo? ¿Y como máximo? Justifique la equivalencia de los valores (si existe), o su variabilidad en caso contrario.