UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

CONTROL DE UNA MATRIZ DE LÁSERES VCSEL MEDIANTE MATLAB Y ARDUINO

CARLOS GARCÍA ROIG

Tutor: **FRANCISCO JOSÉ MARTÍNEZ ZALDIVAR**

Cotutor: **MARTÍN SANZ SABATER**

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 11 de septiembre de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO.ES-CV-000030

Resumen

Este trabajo pretende apoyar una línea de trabajo del GPOEI (*Grupo de Procesado Opto-Electrónico de Imágenes)* de la Universitat de València en colaboración con el instituto iTEAM (*Instituto de Telecomunicaciones y Aplicaciones Multimedia)* de la Universitat Politècnica de Valencia, basada en el estudio de la holografía, mediante el diseño de un sistema de control de una matriz de láseres VCSEL (*Vertical Cavity Surface Emitting Laser*) 8x8 utilizando Matlab y Arduino. El trabajo se puede dividir en dos grandes bloques:

El primero estará dedicado a la electrónica para estudiar la manera de conectar los 64 láseres de la matriz a una placa Arduino. También se diseñará un circuito para cada láser que se encargará de alimentarlo para su correcto funcionamiento.

El segundo bloque estará dedicado a la programación tanto en Matlab como en Arduino. Desde el entorno de Matlab se creará una clase que permita al usuario enviar órdenes al Arduino a través del puerto serie para que este las procese y las envíe a la matriz.

Resum

Aquest treball té com a objectiu ajudar una línia de treball del GPOEI (*Grupo de Procesado Opto-Electrónico de Imágenes)* de la Universitat de València en col·laboració amb l'institut iTEAM (*Instituto de Telecomunicaciones y Aplicaciones Multimedia)* de la Universitat Politècnica de Valencia, dedicada a estudiar el fenomen de la holografia, mitjançant el disseny d'un sistema de control d'una matriu de làsers VCSEL (*Vertical Cavity Surface Emitting Laser*) 8x8 utilitzant Matlab i Arduino. El treball es dividirà en dos grans blocs:

El primer es dedicarà a l'electrònica per a estudiar la forma de connectar aquesta matriu de 64 làsers a una placa Arduino. També es dissenyarà un circuit per a cada làser que s'encarregarà de protegir el seu correcte funcionament.

El segon bloc estarà dedicat a la programació en Matlab i Arduino. Dins del entorn de Matlab es crearà una classe que permeti al usuari final enviar ordres al Arduino utilitzant el port sèrie per a que aquest les processe i envie a la matriu.

Abstract

This work aims to support a line of work of the GPOEI (*Grupo de Procesado Opto-Electrónico de Imágenes)* of Universistat de València in collaboration with the institute iTEAM (*Instituto de Telecomunicaciones y Aplicaciones Multimedia)* of the Universistat Politècnica de València, based on the study of holography, by designing a control system of a matrix of lasers VCSEL (*Vertical Cavity Surface Emitting Laser*) 8x8 using Matlab and Arduino. The work can be divided into two blocks:

The first one will be related to electronics to study how to connect the 64 lasers of the matrix to an Arduino board. Every laser will be connected to a circuit that will preserve its correct operation.

The second block will be dedicated to programming in both Matlab and Arduino. From the Matlab environment, a class will be created to allow the user to send orders to the Arduino through the serial port so that they can be processed and sent to the matrix.

ÍNDICE

1 INTRODUCCIÓN

Los láseres que componen la matriz que se pretende controlar son conocidos como VCSEL, de ahí que utilicemos estas siglas cada vez que queramos referirnos a ella.

En nuestro caso se trata de una matriz de 8x8 y su uso representará un apoyo para una línea de trabajo del GPOEI basada en el estudio de la holografía.

La holografía fue inventada en 1948 por Dennis Gabor. La idea principal consiste en registrar el frente de ondas completo, tanto amplitud como fase, que proviene de un objeto para, posteriormente en un segundo paso, ser capaz de reconstruir dicho frente de ondas. Al resultado de este registro se le llama holograma.

Para conseguir registrar la fase del frente de onda que proviene del objeto, se le añade un haz de referencia que genera interferencias, posibilitando registrar los cambios de intensidad debido a las fases de ambos haces. En el segundo paso, utilizando el holograma registrado, y un haz de reconstrucción igual al de referencia, podemos reconstruir el objeto.

Una de las aplicaciones más importantes de la holografía es la **microscopía**. Gracias a la posibilidad de registrar los cambios en la fase es posible realizar medidas como los cambios de índice de un objeto determinado como, por ejemplo, una célula semitransparente donde dichos cambios están relacionados directamente con el grosor de la célula.

Sumado a está aplicación también podemos encontrar el **análisis de objetos en 3D,** entre otros.

Como diría Emmett Leith, uno de los pioneros de la holografía, "l*a holografía puede parecer un campo de investigación no muy amplio. Sin embargo, si la combinas con otros campos conseguirás un área lo suficientemente grande para poder dedicarle toda una vida".*

2 OBJETIVOS.

El principal objetivo de este trabajo es el de apoyar una línea de trabajo del GPOEI (Grupo de Procesado Optoelectrónico de Imagen) de la Universitat de València basada en el estudio de la holografía. Dicho apoyo consistirá en el diseño e implementación del sistema de control de una matriz de láseres VCSEL 8x8.

Se ha decidido utilizar la plataforma Arduino para la comunicación directa con la matriz de láseres a través de un circuito regulador de tensión, de bajo coste, cuyo principal objetivo es el de proteger los láseres de la matriz de cualquier sobrecarga o pico de corriente que pudiese provocar un fallo en su funcionamiento.

El control de la placa Arduino se realizará a través de Matlab, utilizando bien un interfaz gráfico de fácil entendimiento y diversas funcionalidades o bien a través de la línea de comandos.

Como objetivo final se realizará un pequeño experimento utilizando todas las herramientas mencionas anteriormente para aplicarlas sobre el control de una matriz de 2x2 Led. Dicho experimento pretende demostrar el correcto funcionamiento de todo el sistema de control que, en un futuro se expandirá para poder gestionar los 64 láseres que componen la matriz de láseres que se mencionó al principio.

A continuación, se describirá la memoria que recoge todo el proceso llevado a cabo para cumplir con estos objetivos.

3 METODOLOGÍA DE TRABAJO DEL TFG

En este apartado se describe la planificación temporal, así como las diferentes tareas en las que se ha dividido el trabajo y la forma en que éstas se han organizado y realizado.

3.1 DISTRIBUCIÓN DE TAREAS

Una vez claros los objetivos, es hora de definir las tareas a realizar durante el periodo dedicado a este trabajo. La descripción de cada una de las tareas se detallará en el apartado **4 - Desarrollo y resultados**

3.1.1 Aprendizaje

Dentro del trabajo tenemos dos áreas de trabajo bien diferenciadas: la programación y la electrónica.

- **Programación**

La programación tendrá lugar dentro de dos entornos: **Matlab** y **Arduino**. Por la parte de **Matlab** es necesario aprender el funcionamiento de los interfaces y el sistema de clases y objetos que posee. En cambio, dentro de Arduino es necesario aprender el funcionamiento, casi al completo, de dicha plataforma para poder utilizar funciones como la comunicación a través del puerto serie o el uso de la librería **TimerOne**.

La adquisición de estos conocimientos se ha conseguido a través de las asignaturas cursadas durante el Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación, la ayuda de los tutores y la investigación personal.

- **Electrónica**

El uso de la plataforma **Arduino** requiere también unos conocimientos de electrónica para poder entender y aplicar mejor todas las funciones que ofrece. Además, son necesarios los conocimientos de otros conceptos como el uso de transistores, reguladores de corriente y fuentes de alimentación, por citar unos cuantos.

La adquisición de estos conocimientos ha sido posible, en gran medida, a las asignaturas relacionadas directamente con la electrónica que he cursado durante la carrera (Dispositivos electrónicos y Circuitos electrónicos, por ejemplo) y la ayuda de los tutores.

3.1.2 Búsqueda y elección de material

El hecho de contar con una parte de electrónica dentro del proyecto ha implicado la necesidad de buscar los componentes adecuados para realizar las funciones programadas.

También es necesario encontrar una placa Arduino que pueda conectarse simultáneamente a 64 láseres sin que esto suponga un problema para su correcto funcionamiento.

3.1.3 Protocolo de comunicaciones

La placa Arduino se conectará a través de un cable USB, emulando un puerto serie, al ordenador para enviar las ordenes desde Matlab hasta la placa. El conector del extremo del cable que se conectará al Arduino será tipo b y el otro será tipo a. La existencia de esta conexión implica la necesidad de diseñar un protocolo de comunicaciones para, entre otras cosas, asegurarse de que el mensaje enviado se recibe correctamente, e informar de si el formato del mensaje no es el definido es el protocolo.

3.1.4 Circuitería

Una vez obtenidos los conocimientos necesarios llega el momento de ponerlos en práctica. Los láseres que conforman la matriz son bastante delicados y no se puede permitir que un pequeño pico de corriente o error en el circuito electrónico al que se conecta cada uno de ellos provoque un fallo en su funcionamiento. Debido a esto es importante diseñar un circuito que asegure la "salud" de estos láseres para que su vida útil sea lo más larga posible.

3.1.5 Programación

Como ya se ha mencionado en apartados anteriores, la programación está presente en gran medida dentro de este proyecto con dos funcionalidades distintas. Por un lado, Matlab se encargará de interactuar con el usuario que desee utilizar la matriz, creando un entorno de fácil entendimiento y utilización para que este pueda enviar ordenes al Arduino pulsando solamente una serie de botones dentro de un interfaz gráfico. También existirá la posibilidad de enviar todas estas órdenes utilizando la línea de comandos de Matlab.

Sintetizando, Matlab se encarga de enviar las órdenes y el Arduino de procesarlas y operar en una u otra medida.

3.1.6 Experimento

Por último, se realizará un pequeño experimento con una matriz de Led 2x2 en el cual se probará el correcto funcionamiento de todo el sistema de control conformado por todas las tareas realizadas con anterioridad.

El objetivo de este experimento es mostrar también la sencilla extrapolación que supondría pasar de una matriz 2x2 a 8x8.

3.2 DIAGRAMA TEMPORAL

De forma general se puede dividir este TFG en dos grandes bloques temporales: el primero, comprendido entre el 11 y 30 de junio, donde el objetivo principal fue la investigación y aprendizaje de los conocimientos necesarios y, el segundo, asociado al periodo del 1 al 31 de julio, centrado en la implementación del trabajo en base a los objetivos.

En la siguiente tabla se puede observar la distribución temporal seguida.

	Implementación			**Investigación y aprendizaje**		
JUNIO						
L	**M**	**X**	**J**	**V**	**S**	**D**
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	
JULIO						
L	**M**	**X**	**J**	**V**	**S**	**D**
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Tabla 1. Distribución temporal del TFG

Durante el primer periodo se realizaron las primeras dos tareas descritas en el apartado anterior: **Aprendizaje y Búsqueda y elección de material.** El resto de las tareas se han realizado durante el segundo periodo.

4 DESARROLLO Y RESULTADOS DEL TRABAJO

En esta sección se redactará con más detalle todo el desarrollo del TFG desde la parte del aprendizaje hasta el experimento realizado como prueba fiable de funcionamiento. La descripción del presupuesto se ha plasmado en la siguiente sección del documento.

Una vez concluida toda la fase de aprendizaje pasamos a la búsqueda y elección de los elementos idóneos para hacer este trabajo.

4.1 ELECCIÓN DE PLACA ARDUINO

4.1.1 Requisitos

Los requisitos que se establecieron para que una placa Arduino fuese propuesta como candidata fueron los siguientes:

- Existencia de un conector serie para facilitar la conexión entre la placa y el PC.
- Capacidad de procesado suficiente para manejar tráfico de información a través del puerto serie, modificar el estado de muchos pines al mismo tiempo.
- Inclusión de un bus I2C.

En base a los requisitos establecidos, se accedió a la lista de modelos que Arduino oferta en su tienda online y, tras un tiempo investigando, se encontraron dos placas que los cumplían. Dichas placas eran: **Arduino UNO Rev3** y **Arduino Mega 2560 Rev3**

Se ha elaborado una pequeña tabla comparativa entre ambas en la que se muestra las características principales de cada una, para ayudar a tomar la decisión.

Placa	Arduino UNO	Arduino Mega 2560 Rev3
Voltaje de operación	5 V	5 V
Pines Digitales	14	54
Pines PWM	6	15
Pines Analógicos	6	16
Corriente continua por pin digital	20 mA	20 mA
Memoria Flash	32 KB	256 KB
SRAM	2 KB	8 KB
Conector serie	Sí	Sí
Bus I2C	Sí	Sí

Tabla 2. Características placas Arduino

A continuación, se definió una lista de criterios para ayudar a seleccionar la placa idónea para este proyecto.

Los criterios fueron los siguientes:

- Prioridad de la potencia de procesamiento de la placa ya que se realizarán → no sé si me convence el tiempo verbal diversas funciones al mismo tiempo y no es deseable que la placa Arduino se vea saturada y ralentice la ejecución de estas.
- La falta de importancia en la cantidad de los pines digitales de la placa debido a la adquisición de una placa auxiliar encargada de conseguir esos 64 pines para poder conectar la matriz VCSEL.
- Precio no elevado.

En base a estos criterios se decidió optar por la placa **Arduino Mega 2560.** Pese a que la placa **Arduino UNO,** a priori, podía cumplir con los requisitos establecidos, se quedaba corta en potencia de cálculo frente al modelo **Mega.** Al considerar la potencia como la característica más importante, se descartó el modelo **UNO** a pesar de que en los otros criterios le diferencia no era significativa.

Se encargó el modelo Mega 2560 Rev3, cuyo aspecto se puede observar en la siguiente imagen:

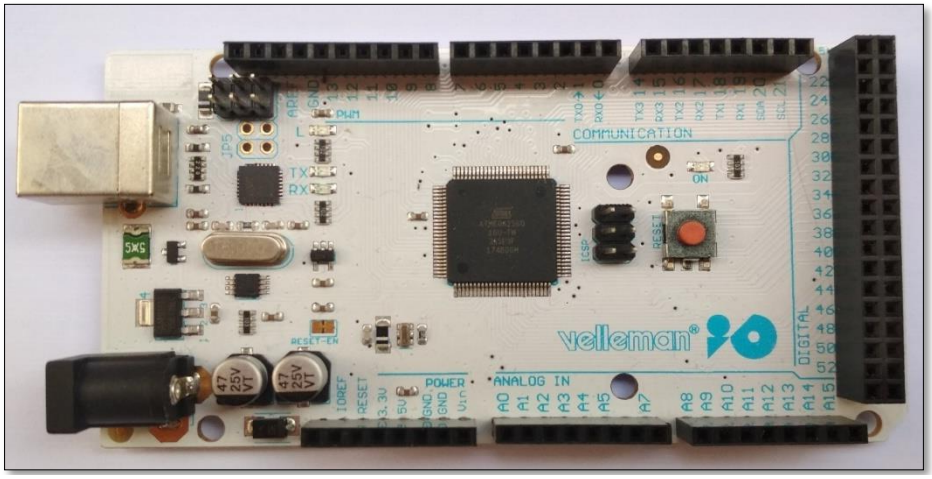

Ilustración 1 Placa Arduino Mega 2560 Rev3 de otro fabricante.

Como se puede observar en la ilustración 2, la placa no tiene el mismo aspecto que el modelo Mega 2560 ofertado en la página oficial de Arduino. La causa de esto es porque se adquirió en una tienda de confianza en la que la universidad suele realizar otras compras de componentes electrónicos, sumado al hecho que la plataforma es de hardware libre, de modo que cualquier fabricante puede coger los esquemáticos y producir y comercializar su propia versión.

Una placa Mega 2560 original tendría el siguiente aspecto:

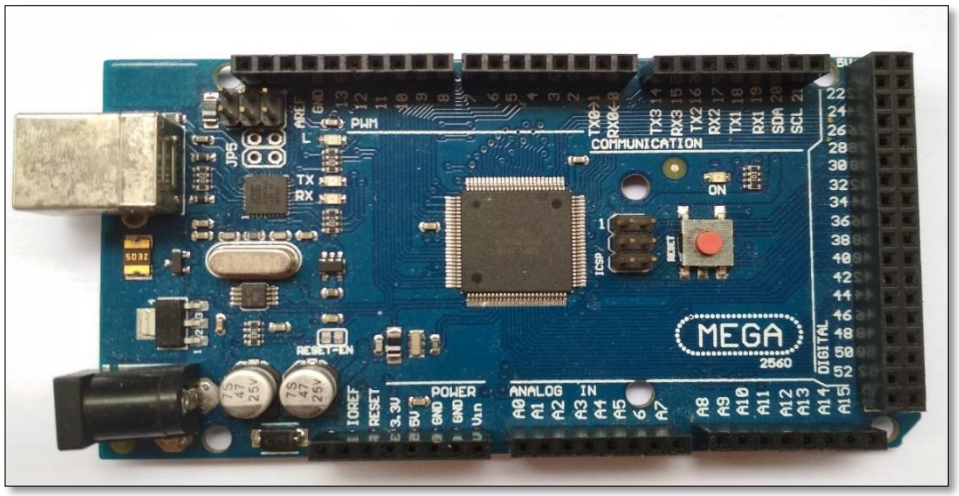

Ilustración 2. Placa Arduino Mega 2560 Rev3 Original

Aunque el aspecto sea diferente, no se puede decir lo mismo del funcionamiento ya que el IDE de Arduino reconoció sin ningún problema el modelo de otro fabricante sin la necesidad de instalar un controlador adicional.

Una vez adquirida la placa pasamos a valorar el siguiente punto importante relacionado con el hardware: **la cantidad de salidas**.

4.2 AUMENTAR LA CANTIDAD DE SALIDAS

Como ya se ha comentado anteriormente, el objetivo final de este trabajo es el de controlar una matriz compuesta por 64 láseres, lo que implica que esos 64 láseres tienen que estar conectados a la placa Arduino.

Esto suscita un problema ya que la placa adquirida cuenta solo con 54 pines digitales. De modo que impera la necesidad de poder alimentar los 10 láseres restantes.

La primera solución que se planteó fue la de utilizar los propios pines analógicos, 16 en concreto que tiene nuestra placa, como pines digitales. Aunque está solución resulta la más rápida se descartó al final. La principal razón fue el embrollo que supondría conectar los 64 circuitos directamente a la placa Arduino generando un cableado complejo que dificultaría el montaje, sumando al hecho que daría una imagen poco ordenada y caótica.

Descartada está opción, se planteó la opción de buscar alguna placa o "shield" ya diseñada que pudiese facilitarnos el conectar todos esos circuitos mencionados con algún tipo de orden. Tras una investigación, se encontró una placa/shield diseñada por **Macetech** que permitía aumentar las salidas digitales del Arduino en 64, consiguiéndose un total de 118 pines digitales si los sumásemos a los 54 iniciales.

La placa recibe el nombre de **Centipede V2 Shield** y tiene el siguiente aspecto:

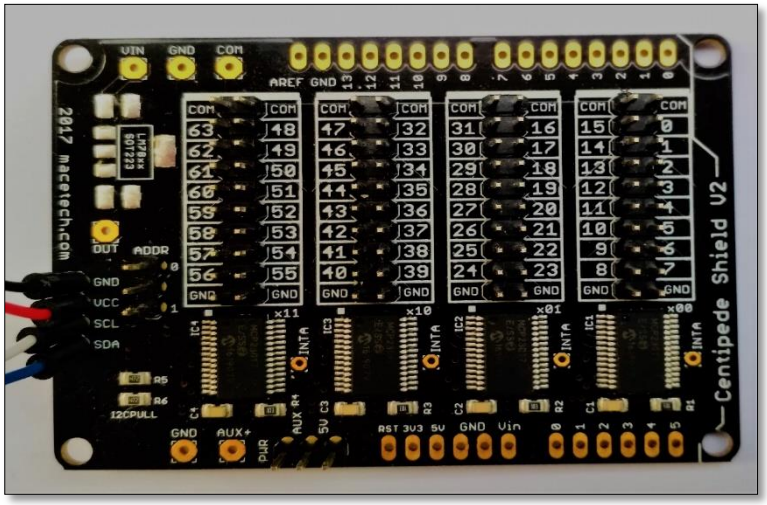

Ilustración 3 Placa Centipede v2 Shield

Atendiendo primero a la **Imagen 2,** encontramos 4 multiplexores, cada uno con 16 salidas. Conectando cada uno de los 64 circuitos mencionados con anterioridad a cada una de las salidas de la placa adquirida evitaríamos saturar el propio Arduino de muchas conexiones excepto las necesarias para alimentar y transmitir datos entre una placa y otra. Hay que mencionar también que el cableado final será más fácil de identificar al estar dispuestos todos los pines de manera muy clara y ordenada.

Una vez revisadas las características de esta placa pasamos a ver cómo funciona. Para poder alimentarla utilizando el Arduino se tiene que conectar el pin de VCC y GND, los cuales se pueden localizar en la parte izquierda de la **Imagen 2,** utilizando los cables dispuestos para ello, rojo y negro respectivamente.

Para poder comunicarse con el Arduino se utiliza el bus I2C. El estándar I2C fue definido hace mucho tiempo y su funcionamiento es realmente sencillo. Solamente requiere dos cables: un para la señal de reloj y otro para la transmisión y recepción de datos. La comunicación se define como un sistema de maestro-esclavo en el que el maestro (**Arduino**) emite una serie de datos a un esclavo (**Centipede**). En este caso no es necesario distinguir entre varios esclavos ya que solo se encuentra la placa **Centipede** conectada a este bus. Otra característica importarte es que es un bus síncrono, es decir, el maestro genera una señal de reloj que mantiene a todos los dispositivos conectados al bus sincronizados.

Cada placa Arduino tiene unos pines específicos asociados a la señal de reloj (SCL) y de datos (SDA). En el caso del modelo que estamos tratando son el 20 y 21 para datos y reloj respectivamente. Para localizar esos pines en la placa **Centipede** basta con mirar junto a los pines de GND y VCC situados en la parte izquierda. Conectando el cable blanco al pin 21 y el azul al 20 ya podríamos transmitir información entre ambas placas.

Junto a la compra de esta placa también se incluían una serie de patillas para convertirla en un shield típico de Arduino, pero se decidió prescindir de está funcionalidad para que los componentes no se juntasen mucho y dificultase el montaje final.

En el siguiente esquema se puede observar como conectar adecuadamente la placa Arduino con la Centipede.

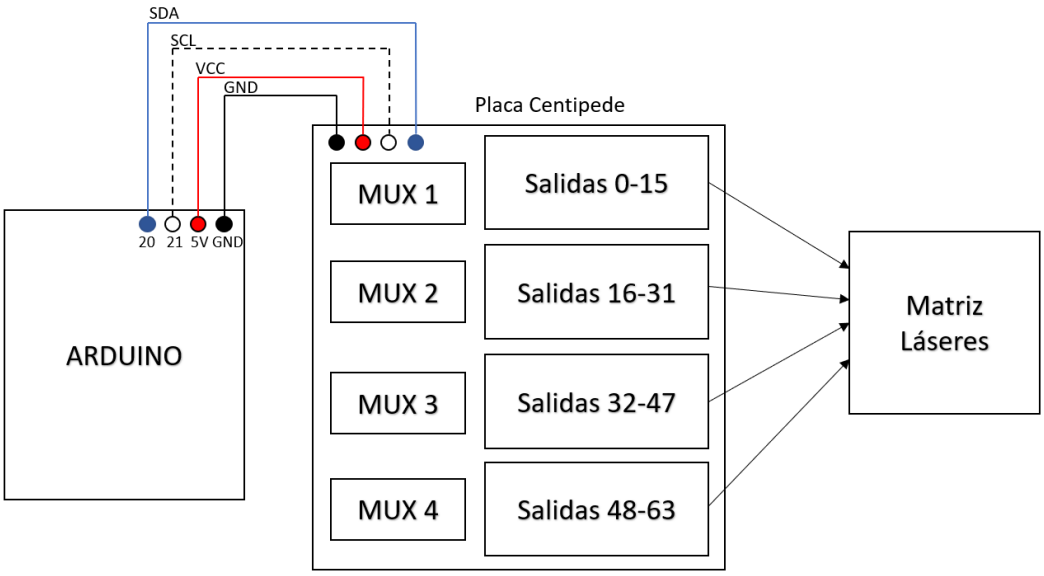

Ilustración 4. Esquema de conexión de la placa Centipede.

4.3 CIRCUITO REGULADOR

Los láseres que componen la matriz que se desea controlar son bastante delicados, por no añadir que son insustituibles y muy caros. Sabiendo esto, no podemos permitir que, por una sobrecarga, fallo o cualquier otro tipo de suceso altere su funcionamiento. De modo que el diseño de una especie de circuito regulador/protector es totalmente necesario.

Otra razón importante para tener que diseñar este circuito tiene relación con el tema de la alimentación. Situándonos en un caso crítico en el que tuviésemos que encender todos los láseres de la matriz, nos encontraríamos con el hecho de que la placa Arduino no es capaz de suministrar la potencia necesaria; de hecho, podríamos llegar a poner en peligro la integridad de la propia placa. Sabiendo esto, se llegó a la conclusión de que se necesitaba alimentación externa para hacer funcionar los láseres de la matriz. El papel del Arduino se limitaría a las funciones de control de estos. La alimentación externa se puede obtener de varias maneras: una de ellas es utilizar una pila 9V, sólo usada durante el experimento, y la otra es utilizar una fuente de alimentación regulable como la que se puede encontrar en casi cualquier laboratorio de electrónica.

Ilustración 5. Fuente de alimentación regulable

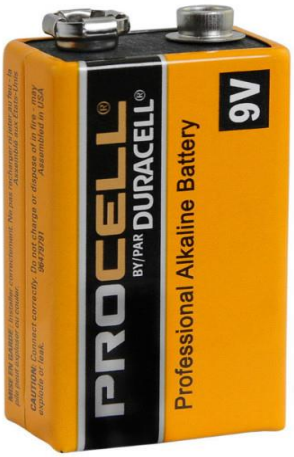

Ilustración 6 Pila 9 V

El uso de la pila de 9 V sólo se podrá ver durante el experimento realizado al final del desarrollo de este trabajo. Sin embargo, no se podría usar para alimentar los 64 láseres a la vez llegado el momento ya que ocurriría el mismo problema que con la placa Arduino. No por esto se ha dejado de mencionar en este trabajo ya que su utilización ha sido necesaria durante las diversas pruebas que se han realizado fuera del laboratorio de la universidad debido a la falta de la fuente de alimentación regulable.

Resumiendo, las características que debería tener nuestro circuito son las siguientes:

- Protección frente a picos de corriente.
- Alimentación externa.
- Control de encendido utilizando el Arduino.
- Independencia entre el Arduino y la alimentación externa.

4.3.1 Selección de componentes.

4.3.1.1 Regulador de tensión.

La principal idea por la que se decidió que se necesitaba un circuito regulador es por el hecho de proteger los láseres debido a su alto coste y elevada fragilidad. Debido a esto, es necesario centrarse primer en encontrar una forma de regular la corriente que le llega a cada uno de ellos.

Para esto se ha decidido incluir en el circuito, antes que ningún otro componente, un regulador de tensión. El modelo seleccionado es el **LM317,** un regulador de tensión ajustable que se diseñó hace bastantes años, pero no por ello ha dejado de resultar útil.

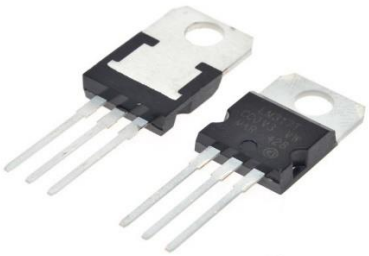

Ilustración 7. Componente LM317

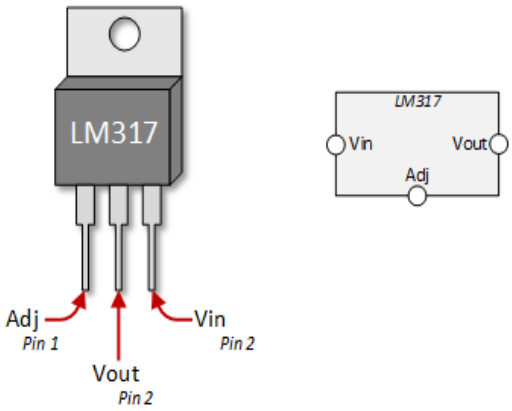

Ilustración 8. Descripción de los pines y esquemático del LM317.

El LM317 cuenta con tres pines: **Vin**, **Vout**, **Adj**. El pin **Vin** es al que se conecta la pila, fuente o cualquier otra forma de suministrar corriente eléctrica. Los otros dos pines tienen una relación muy concreta la cual define la utilidad de este componente. Dicha relación es que siempre habrá **1,25 V** entre dichos pines.

Sabiendo esta relación podemos determinar, con tan solo colocar una resistencia entre el pin **Vout** y el **Adj**, cuál será la corriente que circulará por ella. Además, se limitará la cantidad de corriente que pasará por la resistencia.

Para entender mejor su funcionamiento realizamos un pequeño ejemplo con el siguiente montaje:

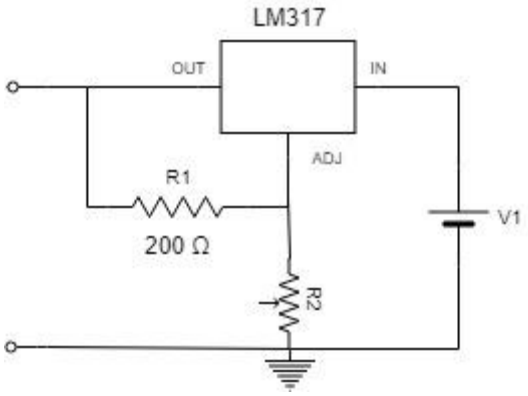

Ilustración 9. Circuito de ejemplo de uso del LM317

El circuito consta de una fuente, dos resistencias, una de 200 ohmios y la otra regulable, y el regulador de voltaje LM317.

Si aplicamos las leyes de ohm, recordando que el voltaje entre los pines de Adj y Vout es siempre de 1,25, obtenemos lo siguiente:

$$\boldsymbol{I_{R1} = \frac{V}{R_1} = \frac{1.25}{R_1} \quad A} \tag{1}$$

De esta forma podemos controlar la intensidad que queremos que circule por dicha resistencia.

La intensidad que circula por el pin de Adj es muy pequeña, del orden de µA, por lo tanto, podemos despreciarla y asumir que la intensidad que circula por R1 es la misma que por R2.

De modo que el voltaje que cae en R2 es el siguiente:

$$\boldsymbol{V_{R2} = I_{R1} * R_2 = \frac{1,25}{R_1} * R_2 \quad V} \tag{2}$$

Sumando ambos voltajes, V_{R1} y V_{R2} , obtenemos el voltaje que hay a la salida.

$$\boldsymbol{V_{out} = V_{R1} + V_{R2} = 1,25 + 1,25 * \frac{R_2}{R_1} = 1,25 * \left(1 + \frac{R_2}{R_1}\right) \quad V} \tag{3}$$

Variando R2 podemos regular el voltaje que habrá a la salida del circuito, sin afectar a la corriente que circula entre ambas resistencias ya que su valor depende únicamente de R1, valor de la cual no cambiamos en ningún momento.

Aunque el hecho de poder regular la tensión no es primordial en este proyecto, si lo es el controlar la corriente que la llega a cada láser. Esta necesidad convierte el regulador LM317 una pieza indispensable en nuestro circuito.

4.3.1.2 Transistor

Para poder independizar el control de encendido de los láseres con la alimentación de estos se ha decidido utilizar un transistor.

El modelo seleccionado ha sido el **BC547B** y sus características pueden encontrarse en la siguiente tabla:

V_{CEO} (V)	V_{CBO} Maximum (V)	I_C (A)	h_{FE} Minimum at I_C = 2mA	f_T (Typical) MHz	P_{tot} (mW)	Package	Part Number
45	50	0.1	200	300	625	TO-92	BC547B

Tabla 3 Especificaciones del BC547B.

Las características de este modelo de transistor cumplen sin problema las exigencias del circuito que pretendemos diseñar.

4.3.2 Diseño

Adquiridos ya los componentes pasamos a realizar el diseño del circuito que se encargará de proteger los láseres, el cual se puede observar en la siguiente imagen.

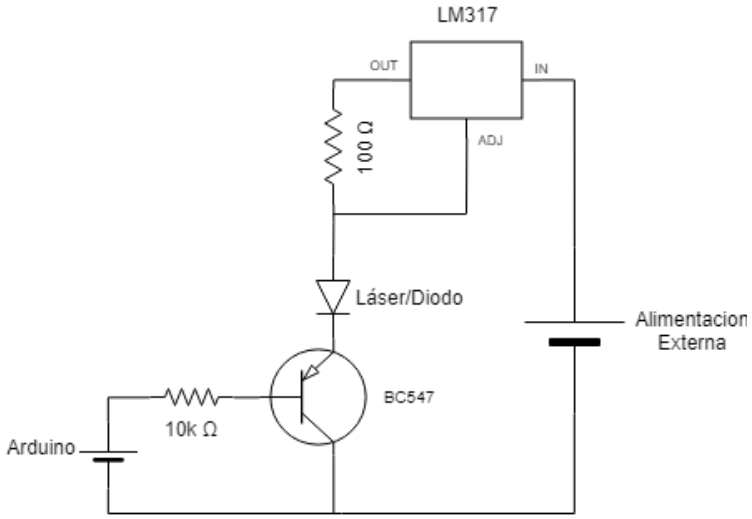

Ilustración 10. Primer diseño del circuito regulador

Se ha elegido una resistencia de 100 ohmios para colocarla entre los pines **Out** y **Adj** del regulador de tensión. Atendiendo a la fórmula (1) podemos calcular de forma inmediata la corriente que circulará por esa resistencia.

$$I_R = \frac{1,25}{100} = 12,5\ mA \qquad (4)$$

Esta corriente es la suficiente para poder encender los láseres de la matriz VCSEL, así como los leds que utilizaremos en el experimento más adelante.

El valor de la alimentación externa debería ser entre 5 y 9 V, pero ahora mismo es un tema trivial ya que el proyecto final contará con el uso de una fuente de tensión regulable de modo que se podrá cambiar entre cualquiera de los valores comprendidos en ese rango sin ningún tipo de problema.

El transistor actúa como una especie de interruptor, cuando se encuentra en corte simularía un interruptor abierto y cuando se encuentra en saturación simularía un interruptor cerrado. Para poder ir cambiando entre un estado y otro utilizamos el Arduino.

El funcionamiento el siguiente: cuando el Arduino procesa el mensaje recibido averigua que láseres debe encender y cuales mantener apagados. Para aquellos que tiene que encender, activa el pin correspondiente de la placa **Centipede** del circuito que contiene el láser que se desea encender. A efectos del circuito, es como si colocásemos una pila de 5 V a la entrada en la base del transistor, como se puede observar en la Imagen 8. La resistencia de 10k ohmios provoca que la intensidad que circula por la base del transistor sea muy pequeña, más concretamente de:

$$\boldsymbol{I_B = \frac{5 - 0,7}{10000} = 430\ \mu A} \qquad (5)$$

Aunque esta corriente sea extremadamente pequeña, es la suficiente para hacer que el transistor entre en saturación, simulando un interruptor cerrado entre emisor y colector, y permitiendo que el láser se ilumine.

4.3.3 Prueba

Para comprobar si los cálculos realizados anteriormente son correctos se ha montado este circuito regulador para tomar las siguientes medidas y contrastarlas.

Medida	Teórica	Experimental
Tensión que cae en R1 = $\boldsymbol{V_{R1}}$	1,25 V	1,24 V
Tensión que cae en el diodo led = $\boldsymbol{V_D}$	1,9 V	1,98 V
Diferencia de tensión entre Colector y emisor = $\boldsymbol{V_{CE}}$	0 V	0,08 V
Diferencia de tensión entre el pin Out y Adj del regulador = $\boldsymbol{V_{Regulador}}$	5,85 V	5,7 V
Intensidad que circula por el led = $\boldsymbol{I_D}$	12,5 mA	12,4 mA
Tensión que cae en R2 = $\boldsymbol{V_{R2}}$	4,3 V	4,29 V
Intensidad en la base del transistor = $\boldsymbol{I_B}$	430 uA	429 uA

Tabla 4. Comparación entre medidas experimentales y teóricas.

Los resultados son casi idénticos por lo que podemos concluir que el circuito cumple con los objetivos propuestos. Se podría sustituir sin problema alguno el diodo led por un láser de la matriz VCSEL ya que no habría riesgo de sobrecarga al estar limitada por el regulador de tensión LM317.

De los 9 V que se están suministrando a la entrada del circuito 5,7 V caen en el regulador de tensión. Esto no supone tampoco un riesgo para el funcionamiento de este ya que es capaz de soportar hasta 40 voltios de diferencia entre entrada y salida.

La corriente mínima necesaria para que el transistor entrase en saturación se puede obtener de la siguiente formula teniendo en cuenta que la IC_sat es de 12,5 mA.

$$\boldsymbol{I_B = 5 * \frac{I_{C_{sat}}}{\beta} = 5 * 12,5 \frac{12,5 * 10^{-3}}{200} = 312,5\ \mu A} \qquad (6)$$

Con todo esto, podemos concluir que el circuito regulador se encargará perfectamente de proteger los láseres de la matriz VCSEL evitando que su funcionamiento se vea comprometido en cualquier momento.

4.4 PROTOCOLO DE COMUNICACIONES

Dejando un poco de lado toda la parte relacionada con la electrónica, pasamos a hablar de la comunicación entre el Arduino y Matlab.

El Arduino está conectado al PC a través de un cable USB y será a través de este que se producirá todo el envío de información entre ambos dispositivos. La naturaleza de la conexión que tenemos en esta situación no nos permite saber sin un mensaje ha llegado correctamente y, en caso de que llegase, saber si ha llegado con algún error, ya sea uno o varios bits cambiados de signo o la falta de un parte del mensaje que el emisor envió.

Debido a esta situación se ha decidido crear un protocolo de comunicaciones personalizado para poder realizar las siguientes funciones:

- Respuesta por parte del receptor (ACK).
- Comprobación de la estructura del mensaje.
- Determinar cuándo se está recibiendo un mensaje.
- Diferenciar entre tipos de mensajes.

Empezamos definiendo la estructura de los mensajes que se intercambiarán entre el Arduino y el PC ya que cada uno enviará un mensaje distinto. El PC se encargará de enviar órdenes y el Arduino de procesarlas y generar los mensajes de respuesta pertinentes.

La estructura de los mensajes del PC es la siguiente:

Carácter Inicio	Tipo de mensaje	Patrón de encendido	Carácter de fin
1 byte	1 byte	0 - 64 bytes	1 byte

Valores de los campos:

- **Carácter de inicio**: letra "i", con valor 105 según el código ASCII.
- **Tipo de mensaje**: actualmente hay dos tipos de mensajes: Uno asociado a letra "a" y otro a la letra "b". El asociado a la letra "a" se encuentra en desuso actualmente y el asociado a la letra "b" indica que se enviará un **patrón de encendido** para que el Arduino lo pueda procesar.
- **Patrón de encendido**: Este campo solo se usa cuando el mensaje es de tipo b. Su tamaño puede oscilar entre 0 y 64 bytes. La razón para definir como variable el tamaño de este campo es para incluir la posibilidad de controlar matrices de distintos tamaños siempre y cuando sean cuadradas. Esto puede resultar útil para realizar diversos experimentos como el que tendrá lugar después de explicar el funcionamiento de todo el sistema de control. Cada uno de los bytes se asocia a uno de los láseres de modo que si el valor del valor de ese byte es "1" se encenderá y si es "0" se apagará o mantendrá apagado en caso de que ya lo esté.
- **Carácter de fin**: El valor de este carácter será siempre la letra "f", con valor 102 según el código ASCII.

Más adelante, cuando afrontemos la parte de programación en Matlab, se explicará cómo definir un tamaño para el campo **Patrón de encendido** mediante un constructor de una clase personalizada.

Todos los mensajes enviados desde el PC tienen un carácter de inicio y otro de fin para que el Arduino sepa cuando tiene que empezar y dejar de leer información del puerto serie y pasar a procesar el mensaje en sí.

Se ha decidido que los mensajes sean una cadena de caracteres que el Arduino va leyendo de uno en uno a través de puerto serie. Las razones para tomar esta decisión son las siguientes:

- Sencillez a la hora de crear el mensaje utilizando Matlab.
- Simplificar la tarea de decodificación por parte del Arduino.
- Falta de criticidad en el tiempo de procesado de los mensajes, así como el tiempo de encendido de los láseres.

Una vez recibido cualquiera de los dos mensajes, a o b, el **Arduino** genera una respuesta. Cada uno de los mensajes genera una respuesta distinta.

En el caso del mensaje tipo a, el Arduino contesta, actualmente, la frase "Informando". Esto se debe a que el uso del primer mensaje ha pasado a segundo plano tras un cambio que se produjo en el programa encargado de procesar y actuar sobre los pines del Arduino. Dicho cambio se realizó después de haber definido este protocolo de modo que este tipo mensaje pasó a un segundo plano. Eso no descarta que se le puede añadir un nuevo tipo de funcionalidad en el futuro.

Por otro lado, si se recibe un mensaje tipo b, el Arduino recoge el patrón de encendido que ha sido enviado desde el PC y lo envía de vuelta a través del puerto serie para que este se muestre en la ventana de comandos de Matlab. De esta forma es posible visualizar que patrón ha procesado el Arduino y comprobar si coincide con el que se envió.

Observando las siguientes dos imágenes podemos entender mejor el funcionamiento de este protocolo.

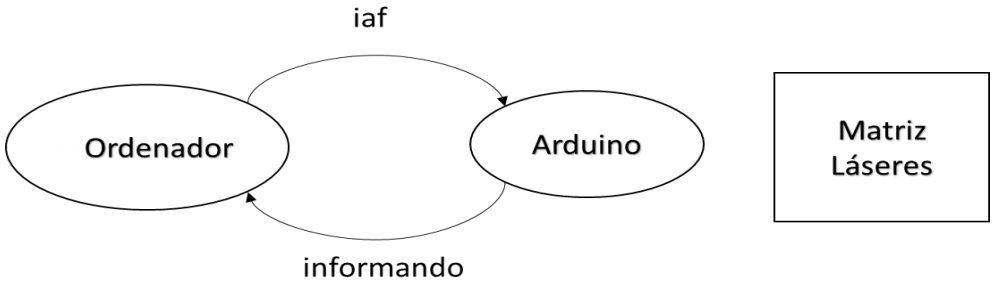

Ilustración 11. Ejemplo de mensaje tipo a.

En la primera imagen podemos ver como el mensaje está compuesto por 3 bytes: carácter de inicio, tipo de mensaje a y carácter de fin. El Arduino lo procesa, no envía nada a la matriz de láseres, y contesta enviando el mensaje "informando" al PC.

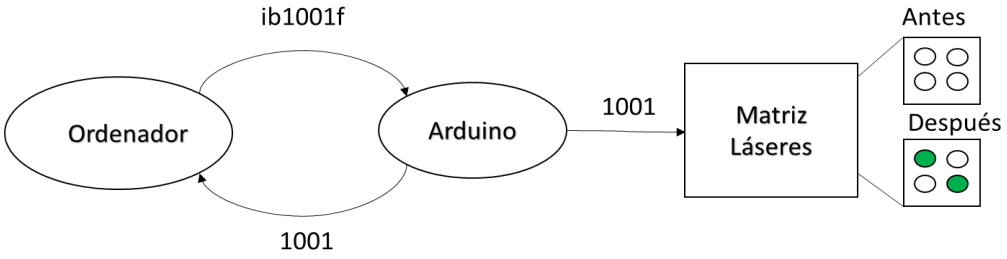

Ilustración 12. Ejemplo de mensaje tipo b.

La segunda imagen muestra el funcionamiento habitual de este protocolo. En este caso se trata de una matriz de 2x2 como la que veremos en el experimento. El mensaje está formado por 7 bytes: carácter de inicio, tipo de mensaje b, patrón de encendido y carácter de fin. Una vez que el Arduino procesa el mensaje, envía la secuencia a la matriz de láseres para que encienda y apague los láseres que corresponde. Partiendo de la situación en el que todos los láseres estén apagados, se encenderán el primero y el cuarto puesto que sus bytes correspondientes tienen el valor de "1".

Una vez hecho esto, el Arduino contesta enviando el patrón que ha recibido, en este caso 1001, al PC.

4.4.1 Protección frente a errores

Debemos tener en cuenta que existe la posibilidad de que aparezcan errores durante el proceso de envío y recepción del mensaje. Para poder prevenir malas interpretaciones por parte del Arduino se ha diseñado un método dentro de la parte de programación en Arduino que se encarga de analizar el cuerpo del mensaje después del recibirlo y antes de procesarlo. Antes de ejecutar este método, el Arduino se ha encargado de recibir el mensaje enviado desde Matlab. Para realizar esta tarea ha ido leyendo constantemente los datos que llegaban a través del puerto serie hasta que ha identificado los caracteres de inicio y fin. Una vez almacenado el cuerpo del mensaje en un vector de caracteres, pasa a analizar su contenido.

Primero comprueba si el primer carácter de ese vector coincide con los caracteres asociados a los tipos de mensajes definidos dentro del protocolo, en este caso a o b. En caso de que no cumpla está condición, informa de ello enviando el mensaje "Tipo_de_mensaje_no_existente" a través del puerto serie para que Matlab lo muestre por la línea de comandos.

Una vez hecha la primera comprobación sin detectar un error, pasa a analizar el patrón de encendido mediante un bucle for. En cada iteración del bucle comprobará el carácter correspondiente del patrón y si su valor es distinto de "0" o "1", entonces, enviará a través del puerto serie el mensaje "Patron_erroneo" para que Matlab lo muestre por la línea de comandos.

El bucle for anterior también comprueba que la longitud del patrón coincida con el tamaño de la matriz que se desea controlar, informando con el mensaje "Longitud_ de_patron_incorrecta" si no lo hace.

Si el método no detecta ningún error en el formato de la trama, entonces pasa a procesar el mensaje y enviar el patrón de encendido a la matriz.

En el caso que se diese cualquiera de los tres errores, después de enviar los pertinentes mensajes de error, finalizaría la ejecución del método sin permitir que el mensaje fuese procesado. Tras esto, el método **recepcionMensaje** volvería a ejecutarse en la próxima iteración del loop para leer los datos recibidos a través del puerto serie, descartando todos hasta que detectase el carácter de inicio. El mensaje recibido anteriormente que había sido guardado en un vector de caracteres, empieza a ser sustituido por el nuevo.

Habiendo hablado ya de la parte relacionada con la electrónica y explicado el protocolo que se empleará para poder comunicar de forma correcta y coherente todos los elementos incluidos en este proyecto, pasamos al siguiente gran bloque dentro de este TFG: **Programación**.

4.5 PROGRAMACIÓN EN MATLAB

La parte de programación que se ha realizado en Matlab tiene por objetivo diseñar un sistema que permita comunicarse con la placa Arduino a través del puerto serie y enviar órdenes a través de mensajes que sigan la estructura establecida en el apartado anterior.

Para poder lograr estos objetivos se ha acudido al sistema de clases y objetos que posee Matlab y así diseñar una clase con ciertos atributos y métodos, de sencillo uso y entendimiento, que permitan realizar las tareas designadas a través de la línea de comandos de Matlab. Sumado a esto, también se ha diseñado un interfaz utilizando las herramientas que Matlab pone a disposición para ello con el fin de proporcionar una forma de comunicarse con la placa Arduino a un nivel más alto que la línea de comandos. Este interfaz utiliza una serie de botones que, internamente, ejecuta métodos de la clase definida con anterioridad. La explicación completa del funcionamiento de este interfaz se puede encontrar en el **Anexo 1**.

4.5.1 Definición de la clase vcsel.

El primer paso, antes que nada, es construir la clase y todos sus componentes. El nombre seleccionado para la misma es VCSEL en referencia a los láseres que componen la matriz que se desea controlar.

4.5.1.1 Atributos

Los atributos de la clase vcsel son los siguientes:

- **arduino**

Se utiliza para poder identificar la conexión serie que se establece con la placa Arduino.

- **tamaño**

Indica el tamaño de la matriz de láseres que se quiere controlar. Cuando se crea un objeta de esta clase se genera un matriz cuyo tamaño es el valor de este atributo. Por lo tanto, las matrices que se quieran controlar deben ser cuadradas. El hecho de poder decidir el tamaño de la matriz permite realizar experimentos con diferentes tamaños sin tener que crear una clase particular para cada uno.

- **matriz**

Matriz que almacena el estado de la matriz de láseres que se pretende controlar.

En el momento que se crea un objeto de esta clase, se asigna a este atributo una matriz de ceros cuadrada cuyo tamaño es indicado por el valor del atributo **tamaño.**

- **conectado**

Este atributo permite identificar si el objeto tiene una conexión abierta a través del puerto serie con una placa Arduino. Solo se puede tener una conexión abierta al mismo tiempo de modo que, si tenemos varios objetos creados en nuestro entorno de Matlab, solo uno de ellos podría tener su atributo Conectado con valor "verdadero".

- **puerto**

Identifica el puerto COM en el que se encuentra nuestra placa Arduino. El valor de este atributo es importante ya que sin él no sabríamos con qué puerto COM hay que establecer una conexión serie.

El puerto COM que se asocia al Arduino varía en cada PC ya que los va asignando según estén disponibles en el momento que se conecta por primera vez el dispositivo. Para poder averiguar cuál es el puerto COM asignado se puede acceder al IDE de Arduino y en la pestaña Herramientas > Procesador aparece el puerto COM asignado a cada placa Arduino que haya conectada al PC. Otra forma de localizar dicho puerto es a través del administrador de dispositivos de Windows, en la sección Puertos (COM y LPT).

4.5.1.2 **Métodos**

Antes de pasar a describir cada uno de los métodos empecemos por describir el constructor de la clase. Por defecto recibe tres argumentos: el primero de ellos indica el puerto COM al que se encuentra conectado el Arduino, el segundo el tamaño de la matriz de láseres y el tercero la tasa de baudios (baud rate) a la que se envían los datos a través del puerto serie.

En caso de que alguno de los argumentos no tuviese el formato establecido, ya sea por un error al escribirlo, diferente orden de los argumentos o cualquier otro factor; se mostraría entonces un mensaje por la línea de comandos de Matlab indicando donde se ha producido el error.

Si todos los argumentos han sido introducidos correctamente cambia el valor de los atributos por los valores que les corresponden y crea el objeto.

También existe la posibilidad de escribir un constructor utilizando sólo el primer argumento. En este caso se definen los siguientes valores predeterminados para los dos argumentos:

- Tamaño = 8
- Baud Rate = 9600

Por lo tanto, existen dos formas de crear objetos de clase vcsel. El constructor para un solo argumento se ha diseñado para controlar la matriz que se encuentra en la UV ya que su tamaño y tasa de baudios nunca cambiarán.

Pasamos ahora a describir los diferentes métodos que componen esta clase.

- **conectar**

Intenta crear una conexión serie con el puerto COM indicado por el valor del atributo **puerto** y guarda el resultado en el atributo **arduino.**

Se pueden dar casos para que este método genere una excepción y, en consecuencia, un mensaje de error: la primera es que el objeto ya esté conectado y la segunda hace referencia a un error al intentar conectar. La causa de este segundo error puede deberse a que la conexión esté intentando realizarse sobre un puerto COM erróneo o que dicho puerto ya tiene una conexión establecida.

Si la conexión se realiza correctamente cambia el valor del atributo **conectado** a true y muestra un mensaje por pantalla informando que la conexión ha sido establecida con éxito.

- **desconectar**

Intenta cerrar la conexión serie con el puerto COM indicado por el valor del atributo **puerto.**

En el caso que la conexión ya estuviese cerrada, genera una excepción e informa de ello a través de la línea de comandos.

Si la conexión se cierra satisfactoriamente muestra un mensaje en la línea de comandos y cambia el valor del atributo **conectado** a false.

- **borrar**

Elimina la reserva del puerto COM que se realizó en el momento que se creó el objeto para que este puedo ser utilizado por otro objeto.

También elimina el objeto del entorno de Matlab.

- **encenderLaser (posx , posy)**

Recibe dos argumentos: el primero indica la fila de la matriz que almacena el atributo **matriz** y el segundo la columna, seleccionando así una posición de dicha matriz. Tras esto, cambia el valor del número almacenado en esa posición a 1.

Si la posición indicada por los argumentos no existe dentro de la matriz, salta una excepción y, en consecuencia, un mensaje de error por la línea de comandos.

- **apagarLaser (posx , posy)**

Misma funcionalidad que el método **encenderLaser,** pero cambiando el valor del número seleccionado por 0.

- **encenderFila (posx)**

Recibe un solo argumento para poder determinar una fila de la matriz que almacena el atributo **Matriz.** Utilizando un bucle for, recorre toda la fila seleccionada ejecutando el método **Encender Laser** en cada interacción para poder cambiar todos los números de la fila a 1.

Si la fila seleccionada no existe dentro de la matriz, salta una excepción y, en consecuencia, un mensaje de error por la línea de comandos.

- **apagarFila (posx)**

Misma funcionalidad que el método **encenderFila,** pero en vez de cambiar todos los números de la fila seleccionada por 1, los cambia por 0.

- **encenderColumna(posy)**

Misma funcionalidad que el método **encenderFila,** pero seleccionando una columna en vez de una fila.

- **apagarColumna (posy)**

Misma funcionalidad que el método **apagarFila,** pero seleccionando una columna en vez de una fila.

- **apagarMatriz**

Cambia a el valor del atributo **matriz** a una nueva matriz de ceros cuyo viene indicado por el atributo **Tamaño.**

- **extraerPatron**

Este método se encarga de convertir la matriz almacenada por el atributo **matriz** en una cadena de caracteres para su posterior envío a la placa Arduino.

Para poder realizar esta tarea se utiliza un bucle for realiza tantas iteraciones como filas tenga la matriz, es decir, según el valor del atributo **tamaño.**

En cada iteración obtiene la fila que corresponda de la matriz, la convierte a string utilizando la función *int2str* incluida en Matlab. A continuación, elimina los espacios en blanco entre cada carácter creados a partir de la función anterior y concatena el resultado con otra variable que va almacenando las filas anteriores ya procesadas y convertidas.

Por último, una vez finalizado el bucle for, devuelve el valor de dicha variable.

- **construirMensaje**

Este método recibe como argumento una cadena de caracteres cuyo valor siempre será "**ib**". A continuación, concatena esa cadena con el resultado del método **extraerPatron.** Por último, concatena el resultado anterior con el carácter "f" y devuelve el resultado final.

El objetivo de este método es crear un mensaje que cumpla la estructura que hemos visto en el apartado de **Protocolo de comunicaciones.** Debido a que el mensaje tipo a no precisa enviar ningún tipo de información que modifique el estado de los láseres no es necesario que el valor del argumento que recibe este método sea "**ia**". Pese a que el valor del argumento es siempre el mismo se ha decidido mantenerlo ya que no se descarta la posibilidad de incluir otros tipos de mensajes en el futuro.

- **enviar (tipoMensaje)**

La función de este método es la enviar el mensaje pertinente y bien formado a través de la conexión serie que se ha establecido previamente con la placa **Arduino.**

Recibe como argumento un número que puede tomar el valor de 1 o 2, asociados a los mensajes tipo a y b respectivamente. En el caso que se trate de un mensaje tipo a. Se construye un vector de caracteres cuyo valor sea "**iaf**" y se envía directamente a través de la conexión serie. En el caso de que se trate de un mensaje tipo b, se concatena la cadena de caracteres "**ib**" con el resultado del método **construirMensaje** y se envía a la placa Arduino**.**

Para enviar el mensaje a través del puerto serie se utiliza la función **fprintf** cuyos argumentos son el atributo **arduino,** el parámetro %s para indicar que se trata de una cadena de caracteres y la cadena de caracteres resultante de la ejecución de todo este método. La función **fprintf** puede fallar por diversos factores y, por eso, es conveniente capturarlos. Si el objeto no se encuentra conectado con ninguna placa Arduino, se producirá un error y se informará con un mensaje a través del puerto serie de Matlab. Por otro lado, también puede haber un fallo en el envío del mensaje ya sea porque el puerto ya está siendo usado, entre otros. En dicho caso se producirá, de nuevo, un error y se informará de ello con un mensaje.

Si el envío se ha realizado correctamente, entonces el ordenador deja de comportarse como un emisor y asume el papel de receptor. Durante esta fase intenta recibir una respuesta por parte de la placa Arduino. Si dentro del intervalo de espera, definido internamente por Matlab, se recibe respuesta por el puerto serie, entonces la muestra en la línea de comandos. Si el mensaje enviado era tipo a se debería recibir como respuesta la palabra "Informando" y, si el mensaje enviado era tipo b, se debería recibir como respuesta el **patrón de encendido** que el Arduino ha procesado y transmitido a la placa **Centipede V2 Shield.**

4.5.1.3 **Diagramas de secuencia.**

Para poder explicar y entender mejor el funcionamiento de todos estos métodos se ha creado una serie de diagramas de secuencias que complementan las explicaciones mostradas en la sección anterior.

En la siguiente imagen se pueden los diferentes estados por los que puede pasar un objeto desde su creación hasta su eliminación del entorno de Matlab. El estado inicial siempre será uno base, también llamado idle, en el que no existe conexión serie con ninguna placa Arduino. Si existe una placa conectada al PC se puede intentar conectar con ella para pasar al estado **conectado.** A partir de ahí, se puede eliminar el objeto en cualquier momento, ya sea desde el estado **Idle** o **conectado,** utilizando el método **borrar.** También existe la posibilidad de cerrar la conexión y volver al estado **Idle.**

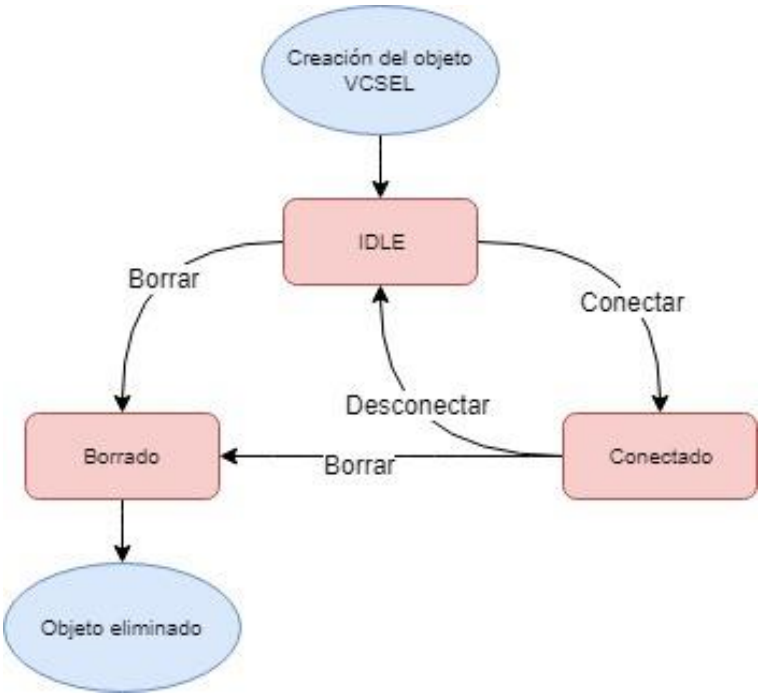

Ilustración 13. Diagrama de estados de un objeto VCSEL.

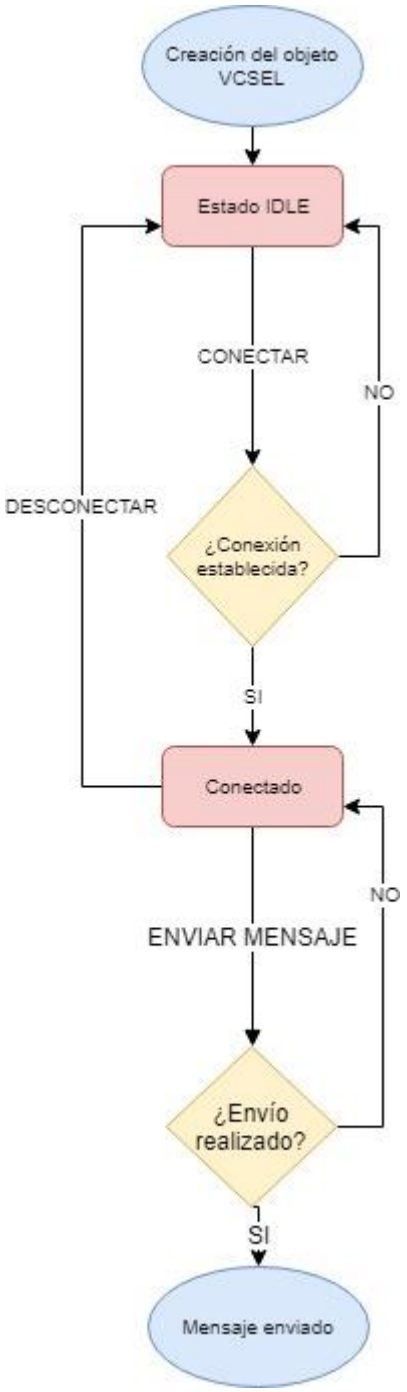

Ilustración 14. Diagrama de conexión y envío de mensaje

En la imagen anterior se pueden observar los pasos que va siguiendo un objeto cualquiera de la clase vcsel desde su creación hasta que se envía un mensaje a la placa Arduino.

4.6 PROGRAMACIÓN EN ARDUINO

La función que realiza el Arduino dentro de este proyecto es la de recibir y procesar los mensajes enviados a través de la conexión serie establecida con el PC. Una vez realizada esta tarea, en el caso que se trate de un mensaje tipo b, la placa Arduino envía los datos correspondientes al patrón de encendido utilizando el protocolo I2C a la placa **Centipede.**

Todas las funciones descritas en el párrafo anterior son totalmente transparentes para el usuario final, es decir, no es necesaria interacción alguna con el Arduino por parte del usuario. Todo lo contrario, a lo que ocurrió en la parte de programación realizada en Matlab en la que se interactuaba a través de la línea de comandas o utilizando un interfaz gráfico.

4.6.1 Estructura del programa

Los programas en Arduino reciben el nombre de sketches y su ejecución consta de dos partes: la primera recibe el nombre de **setup** y es la primera en ejecutarse después de compilar el programa. La segunda se llama **loop**, se ejecuta a continuación del **setup** y continúa ejecutándose de forma indefinida hasta que el Arduino deja de ser alimentado.

Siguiendo esta estructura se ha definido la siguiente distribución de acciones en cada una de las partes que componen un sketch.

- **setup**

En esta parte inicializamos tanto la conexión serie que posee el Arduino como la conexión con la placa **Centipede.** La conexión serie debe abrirse previamente en el Arduino para que el PC pueda conectarse a la placa y enviarle mensajes.

También se activan en modo OUTPUT tantos pines de la placa **Centipede** como láseres haya en la matriz que se desee controlar. En el experimento que realizaremos más adelante solo será necesario activar los pines 0,1,2 y 3 ya que sólo necesitaremos controlar 4 láseres. Pero en el caso de la matriz VCSEL tendremos que activar los 64 pines que forman la placa **Centipede.**

Por último, también se asigna una rutina de interrupción cuya función se explicará en una sección aparte.

- **loop**

En esta parte se encarga de ir leyendo constantemente los datos que llegan a través de la conexión serie. En el momento que se detecte la estructura definida en el **protocolo de comunicaciones y** se comprueba que no hay ningún error en la misma, el Arduino pausa la lectura del puerto serie para poder procesar este mensaje. A continuación, analiza si se trata de un mensaje tipo a o tipo b. En el primer caso envía a través del puerto serie la palabra “Informando” para que Matlab la muestre en la línea de comandos. En el segundo caso extrae el patrón de encendido y lo envía a la placa **Centipede** para encender y apagar los láseres correspondientes.

Al cabo de un intervalo de tiempo todos los láseres de la matriz se apagarán para preservar su vida útil y evitar que, por un descuido, los láseres queden encendidos durante un periodo de tiempo demasiado largo.

Realizado todo proceso, el Arduino reanuda la lectura del puerto serie hasta que detecte de nuevo un mensaje con la estructura correcta.

4.6.2 Rutina de interrupción

Antes de pasar a describir los métodos que conforman el sketch programado es necesario explicar en qué consiste una rutina de interrupción y la función que cumple dentro del programa.

Las placas Arduino UNO y Mega cuentan con un temporizador cuya frecuencia es de 16 MHz. Este temporizador se puede utilizar para programar rutina de interrupción que se ejecuten cada cierto intervalo de tiempo. Para poder realizar esto es necesario descargar una librería llamada **Timer1.** Una vez descargada e importada al IDE de Arduino hay que seguir los siguientes pasos para crear y adjuntar una rutina de interrupción al temporizador de la placa:

1. Crear un método que contenga el trozo de código que deseemos que se ejecute cada vez que salte la rutina.
2. Utilizar el método **initialize** de la librería **Timer1** que recibe como argumento el valor frecuencia de interrupción de la rutina en microsegundos.
3. Utilizar el método **attachInterrupt** de la librería **Timer1** que recibe como argumento el método que queremos que el Arduino ejecute cada vez que salta la rutina de interrupción.

Tras realizar estos pasos, el intervalo de interrupción empezará a contar y, cada vez que se cumpla este, ejecutará el método asociado a la rutina. Tras esto, volverá a la parte del sketch en la que se quedó cuando saltó la interrupción.

Como ya se ha mencionado anteriormente, la función que desempeña la rutina de interrupción en nuestro sketch es la de controlar el apagado de los láseres transcurrido un intervalo de tiempo tras encenderlos. Por tanto, no podemos dejar que la rutina de interrupción vaya ejecutándose con total libertad y se hace necesario controlar cuándo se inicia el intervalo de tiempo para evitar que, por ejemplo, los láseres se apaguen demasiado pronto.

Para lograr esto, la librería **Timer1** cuenta con una serie de métodos para detener, reanudar y reiniciar el intervalo de interrupción. En nuestro caso utilizaremos los siguientes:

- **stop:** Detiene el temporizador.
- **resume:** Reanuda el temporizador.

Por un lado, el método **Stop** se utilizará en dos ocasiones: tras ejecutar el paso 3 y después de apagar los láseres. Por otro lado, el método **Resume** se utilizará después de enviar el patrón de encendido a la placa **Centipede** para asegurar así que el tiempo que los láseres pasan encendidos siempre sea el mismo.

En el diagrama mostrado a continuación se puede ver con más claridad la funcionalidad de la rutina de interrupción.

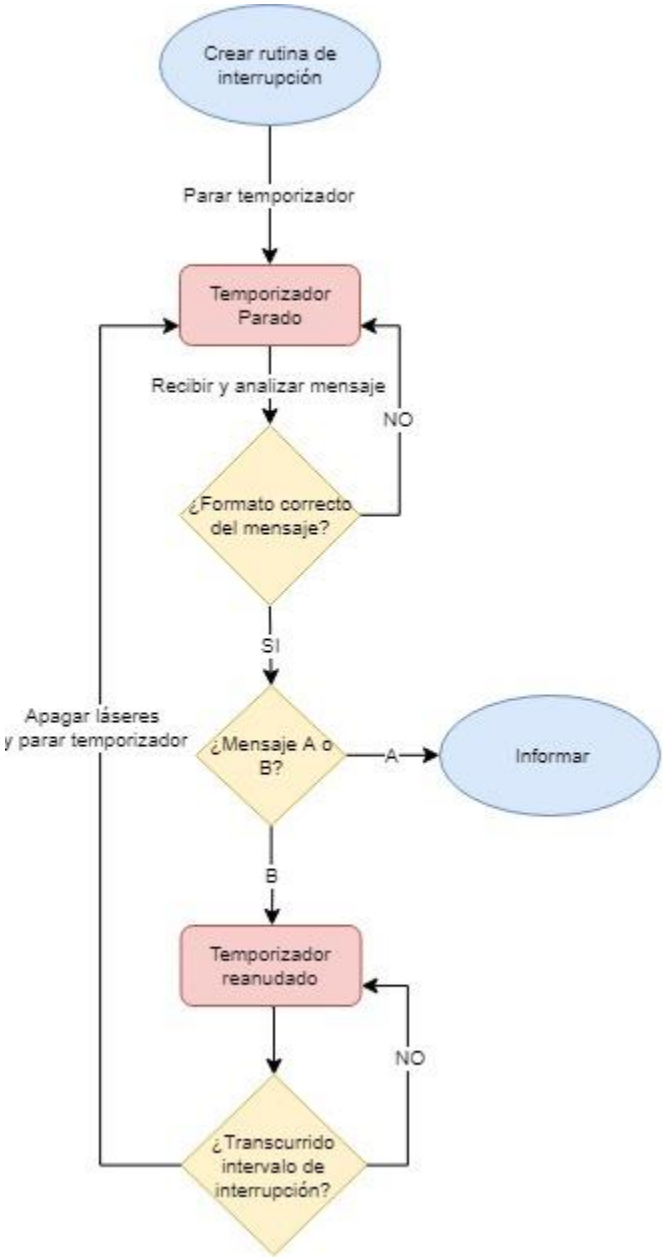

Ilustración 15. Diagrama de funcionamiento de la rutina de interrupción.

4.6.3 Métodos

Las dos partes descritas en el apartado anterior, setup y loop, también son consideradas métodos, pero se ha creído conveniente explicarlas antes para entender mejor el funcionamiento del resto.

Dicho esto, los métodos desarrollados dentro del sketch de Arduino son los siguientes:

- **encenderPines**

Se ejecuta durante el método **setup** y se encarga de activar en modo OUTPUT los pines de la placa **Centipede** que estén conectados a los diversos láseres de la matriz que se desea controlar.

- **recepciónMensaje**

Se ejecuta durante el método **loop** y su función consiste en leer los datos que se van recibiendo a través de la conexión serie con el PC y determinar si se está recibiendo un mensaje. A partir de recibir el carácter de inicio, empieza a almacenar en un vector todos los datos que va leyendo del puerto serie hasta que detecta el carácter de fin. Una vez detectado todo con éxito el mensaje enviado desde el PC, lo envía a otro método encargado de analizarlo para comprobar que cumple con la estructura definida en el protocolo de comunicaciones.

En el siguiente diagrama se puede ver de forma clara su funcionamiento. Se han utilizado cajas verdes para simbolizar las acciones que realiza el Arduino.

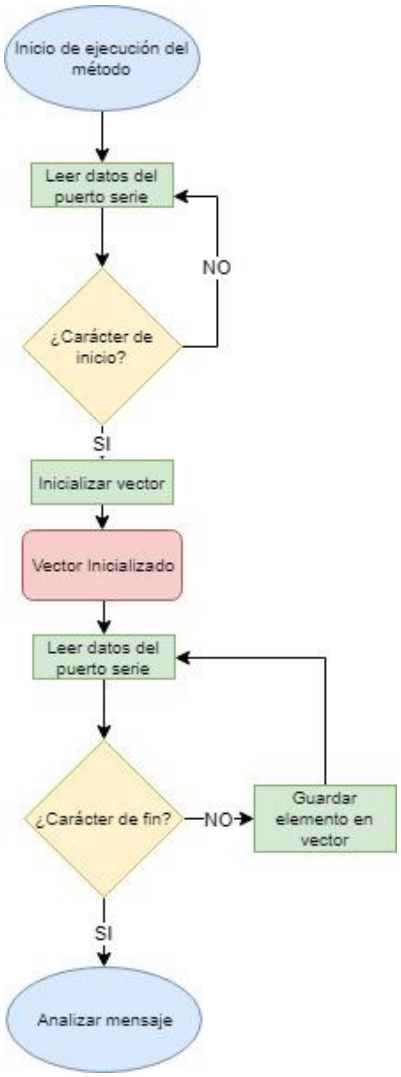

Ilustración 16. Diagrama del método Recepción mensaje

- **analizarMensaje**

Se encarga de analizar el cuerpo del mensaje recibido por el método anterior. Primero comprueba si el primer carácter del vector coincide con los asociados a cada tipo de mensaje dentro del protocolo de comunicaciones. Luego analiza el patrón de encendido mediante un bucle for. En cada iteración comprueba si el carácter correspondiente es un "0" o "1". En caso de que se descubra un error en cada una de las partes de este método informa de ello enviando un mensaje a través del puerto serie para que Matlab lo muestre por la línea de comandos. También evita que el sketch ejecute el método **procesarMensaje.** Por otro lado, si el mensaje no contiene ningún error, continua con la ejecución normal del sketch disponiendo el mensaje para su procesado por parte del método siguiente.

- **procesarMensaje**

Se ejecuta al final del método anterior y se encarga de analizar el vector donde se ha guardado el mensaje enviado desde el PC y determinar si se trata de un mensaje tipo a o tipo b. Si se trata de un mensaje tipo a, ejecuta el método **informar** y si se trata de un mensaje tipo b ejecuta el método **encenderLaseres.**

- **informar**

Se encarga de enviar la palabra "Informando" a través del puerto serie para que Matlab la reciba y la muestre en la línea de comandos y pueda confirmar que el Arduino ha recibido el mensaje tipo a.

- **encenderLáseres**

Se encarga de analizar el patrón de encendido contenido en el mensaje que el método **recepcionMensaje** guardó en un vector. Este análisis consiste en un bucle for que va recorriendo los elementos del vector mencionado. Si el elemento del vector es un 1 enciende el láser de la matriz correspondiente y si es un 0 lo apaga.

Concluida esta tarea, reanuda el temporizador de la rutina de interrupción para que, transcurrido el intervalo definido con anterioridad, apague los láseres de la matriz.

Por último, cambia el valor de una variable de tipo boolean llamada **mensajeProcesado** a true. Esta variable la utilizará el siguiente método para determinar si se tienen que apagar los láseres o no.

- **controlApagado**

Contiene la parte de código que se ejecutará cada vez que salte la rutina de interrupción. Primero comprueba si el valor de la variable de tipo boolean **mensajeProcesado** es true o false. En caso de que sea true, cambia el valor de otra variable de tipo boolean llamada **apagar** a true y el **mensajeProcesado** a false.

- **apagarLaseres**

Se ejecuta en el método **loop** y se encarga de apagar los láseres de la matriz. Antes de realizar este apagado comprueba si la variable **apagar** tiene valor true. Si es así, apaga los láseres, cambia el valor de la variable **apagar** a false y detiene el temporizador de la rutina de interrupción para evitar que salte de nuevo cuando transcurra el intervalo de tiempo de interrupción.

El cambio del valor de la variable **apagar** de true a false se debe principalmente a que este método se ejecuta en cada iteración del método **loop** y no queremos que los láseres se apaguen cada vez ya que el tiempo que pasarían encendidos no sería mayor que un pestañeo.

La idea inicial era que las funciones que realiza el método **apagarLaseres** las realizase la propia rutina de interrupción, pero surgieron problemas al intentar apagar o encender láseres dentro de la rutina y se optó por utilizar el sistema de dos métodos separados y dos variables booleanas.

4.7 EXPERIMENTO

Llegados a este punto se podría decir que el proyecto está casi finalizado. Pero antes queda comprobar si todas las piezas que forman este trabajo encajan de forma correcta y siguiendo todos los criterios que se han ido enumerando y explicando en los apartados anteriores.

Como ya se ha mencionado con anterioridad, no se posible probar directamente todo el sistema de control sobre la matriz VCSEL. La principal razón es que falta decidir cómo se van a distribuir los 64 circuitos reguladores necesarios para poder controlar sin riesgo alguno todos los láseres de la matriz. Para ello, se está estudiando esto de diseñar una PCB que incluya toda la circuitería necesaria para proteger los láseres de la matriz.

Este pequeño inconveniente no evita la posibilidad de comprobar si todo el sistema diseñado funciona correctamente ya que, como se ha explicado en la sección de programación, la clase vcsel diseñada en Matlab permite la creación de objetos para controlar matrices de distintos tamaños siempre y cuando sean cuadradas.

Siguiendo esta línea se ha propuesto un experimento en el que se intentará controlar una matriz de 2 x 2 leds. Cada led contará con su propio circuito regulador para controlar la corriente que circula por ellos. Los leds son baratos, fácilmente reemplazables y cumplen con el principal objetivo de este experimento: demostrar que el sistema funciona.

Los cuatro circuitos estarán alimentados por la misma fuente la cual puede ser o bien una pila o bien una fuente regulable de laboratorio y cada uno estará conectado a un pin de la placa **Centipede,** en este caso del 0 al 3. De modo que, en el momento que uno de ellos se active, provocará que aparezca una pequeña cantidad de corriente en la base de transistor del circuito correspondiente forzando su funcionamiento en modo saturación, comportándose como un simple cable entre colector y emisor, y cerrando el circuito para hacer que circule una cantidad de corriente, totalmente controlada, por el diodo led.

En la siguiente imagen se muestra cómo quedaría el esquemático del circuito que se pretende analizar.

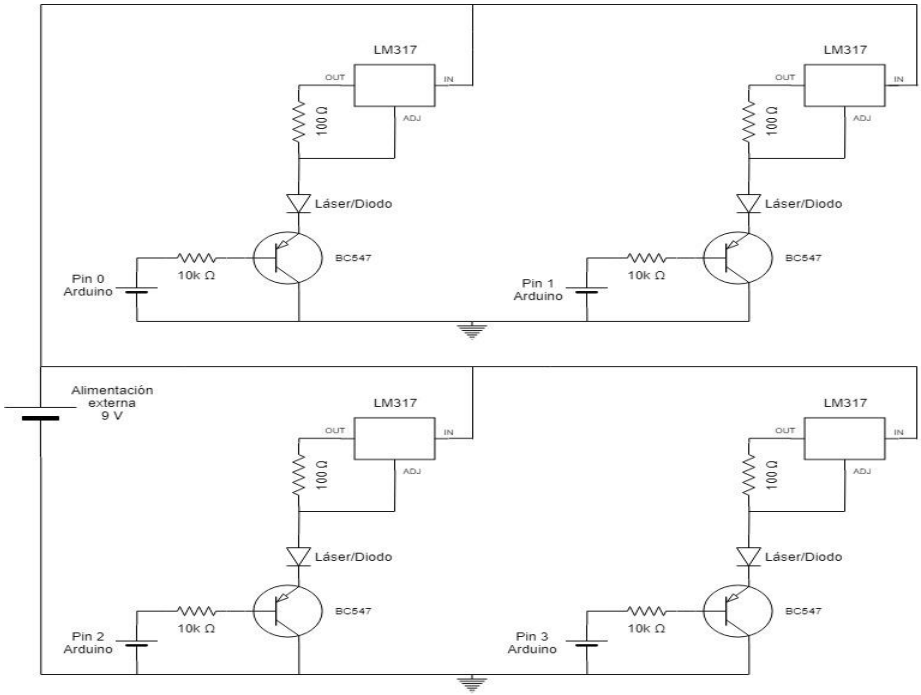

Ilustración 17. Esquema del experimento.

Se han representado los pines del Arduino como fuentes continuas de tensión para mostrar una situación crítica en la que todos los láseres estuviesen encendidos, forzando a la fuente de alimentación regulable a generar más corriente a la salida.

El montaje del circuito quedaría de la siguiente manera.

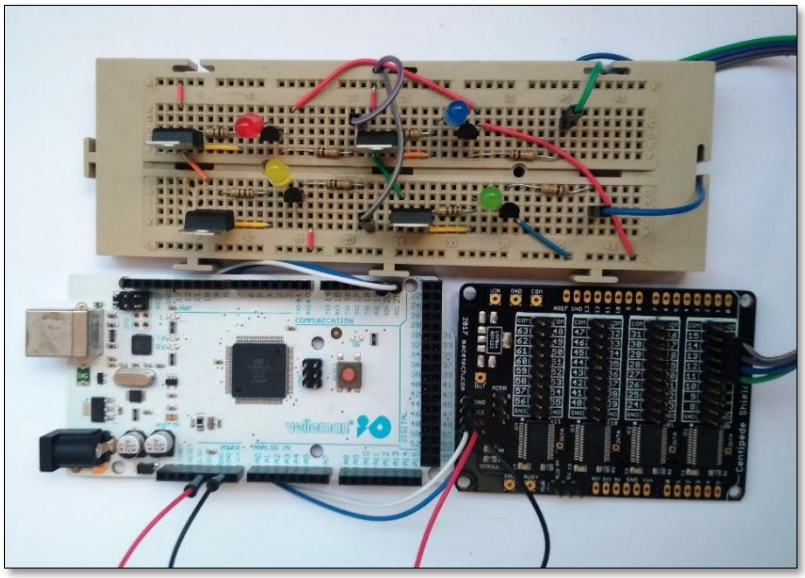

Ilustración 18. Circuito del experimento.

4.7.1 Esquema global

Realizado ya el montaje del circuito del experimento, se conecta al PC utilizando un cable USB, ejecutamos Matlab y utilizamos una de las dos formas establecidas para enviar ordenes al Arduino: bien línea de comandos, bien interfaz gráfico.

En la siguiente imagen se puede observar un esquema global de cómo se conectan todos los elementos que forman este proyecto con el fin de probar su correcto funcionamiento.

.

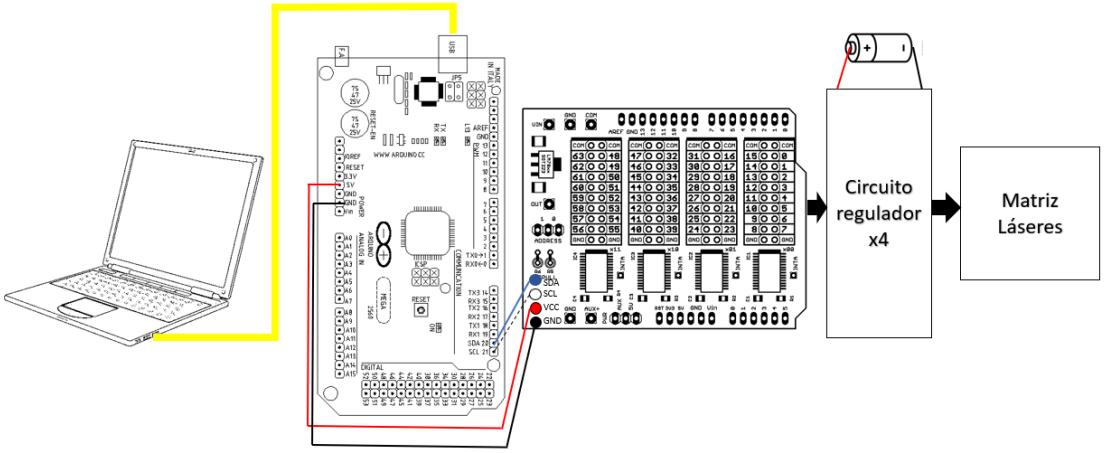

Ilustración 19. Esquema global de conexión.

4.7.2 Ejemplo de uso

Conectamos el Arduino al PC mediante un cable serie y cargamos el sketch, programado anteriormente, en la placa. A continuación, iniciamos Matlab y creamos un objeto de la clase vcsel usando el siguiente comando para indicar que queremos controlar una matriz de 2x2 conectado al puerto **COM5,** a una tasa de 9600 baudios por segundo.

```
>> tfg = vcsel('COM5',2,9600)
```

En este caso hemos usado el constructor con tres argumentos ya que no estamos controlando la matriz de 8x8. Por lo tanto, es necesario proporcionar el tamaño de la matriz.

Una vez creado el objeto podríamos empezar a utilizar los métodos para controlar el estado de la matriz. Pero se ha decidido crear una conexión serie a través del método conectar de la clase vcsel.

```
>> tfg.conectar
Conexión establecida
```

Como la conexión se ha abierto correctamente, se ha generado un mensaje informando de ello.

Llegados a este punto ya podemos enviar órdenes al Arduino para que este las procese.

Por ejemplo, si quisiésemos encender la primera fila y el primer led de la segunda habría que hacer lo siguiente.

```
>> tfg.encenderFila(1)
     1     1
     0     0

>> tfg.encenderLaser(2,1)
```

Tras ejecutar el método **encenderFila** muestra el estado actualizado de la matriz. Sin embargo, no lo hace con el método **encenderLaser.** Si quisiésemos visualizarlo bastaría con acceder al atributo **Matriz** del objeto que hemos creado.

```
>> tfg.matriz

ans =

     1     1
     1     0
```

Y, efectivamente, el estado de la matriz es el que buscábamos.

Ahora vamos a probar a enviar este patrón a la placa para que encienda los leds pertinentes. Utilizando el método **enviar** e indicando entre paréntesis que se trata de un mensaje tipo b (mediante un 2) podemos enviar el mensaje al Arduino.

```
>> tfg.enviar(2)
b1110
```

El Arduino procesa con éxito la orden, la transmite a la placa **Centipede** y contesta adjuntando el tipo de mensaje que ha recibido seguido del patrón de encendido procesado.

Si ahora quisiésemos enviar un patrón totalmente distinto, no sería necesario apagar los láseres utilizando los métodos de **apagarFila** o **apagarColumna** ya que la clase cuenta con otro más directo llamado **apagarMatriz.** También es totalmente cierto que, en la situación actual, no resulta del todo imprescindible al contar sólo con 4 leds. Pero en la situación final, donde habrá 64 láseres, si lo será.

4.7.3 Posibles errores

El primer error que se puede cometer es al crear el objeto de la clase vcsel, bien por colocar los argumentos en un orden distinto al establecido o bien por introducirlos en un formato erróneo. En la siguiente situación se cometerán ambos fallos para que el sistema informe de ello.

```
>> tfg2 = vcsel(2,'COM5',9600)
Warning: Error en el formato de los argumentos
> In vcsel (line 28)
Warning: El segundo argumento debe ser un numero entero, indica el tamaño de la matriz
> In vcsel (line 30)
Warning: El primer argumento debe ser un string de chars. Ej: COM1,COM2,COM3, etc..
> In vcsel (line 33)
```

Tras colocar el tamaño de la matriz se ha incumplido el formato establecido, en consecuencia, ocurre lo mismo con el puerto COM de modo que el sistema informa de ello por la línea de comandos.

Otro error que tiene grandes probabilidades de aparecer tiene relación con el encendido y apagado de los láseres. Cuando ejecutamos cualquier método que altere el estado de la matriz hay que tener en cuenta las dimensiones de esta para no seleccionar un láser, fila o columna que no exista.

Si intentásemos apagar el láser de la tercera fila y primera columna del objeto **tfg** el sistema nos informaría de que no es posible.

```
>> tfg.encenderLaser(3,1)
Warning: El led seleccionado no existe
```

En general, estos serían los dos errores que vigilar. Es cierto que existe la posibilidad de realizar fallos a la hora de conectar o enviar las órdenes, pero estos suelen deberse a errores o excepciones en la conexión del PC con el Arduino. Algo que no depende directamente de introducir comandos en Matlab y, por tanto, menos susceptible de provocarse debido a un fallo humano.

5 CONCLUSIONES Y PROPUESTA DE TRABAJO FUTURO.

Las conclusiones del trabajo son:

- **Correcto funcionamiento del sistema de control de la matriz de láseres**

El objetivo principal era diseñar todo un sistema que combinase la sencillez de Matlab con la gran versatilidad que Arduino tiene al contar con gran cantidad de funcionalidades, librerías y shields complementarios. Dicho objetivo se ha cumplido y se ha logrado crear un sistema sencillo de usar, pero con una complejidad en su interior notable.

- **Demostración mediante un experimento**

Una vez creado todo el sistema de control se ha realizado un experimento en el que se ponían todos los elementos del sistema en conjunto para demostrar su correcto funcionamiento.

Por otro lado, una propuesta de continuación de trabajo podría consistir en estudiar una forma de poder agrupar los 64 circuitos necesarios para cubrir todos lo láseres de la matriz. Una solución posible sería realizar una placa personalizada por parte propia o de un tercero. El problema con esta solución es el elevado coste que supone, así que, por el momento, esta solución queda apartada para analizar con más detalle otras posibles.

6 BIBLIOGRAFÍA

6.1 Arduino y Shields

- https://store.arduino.cc/arduino-uno-rev3
- https://store.arduino.cc/arduino-mega-2560-rev3
- http://macetech.com/store/index.php?main_page=product_info&products_id=23

6.2 Componentes electrónicos.

- http://www.ti.com/lit/ds/symlink/lm317.pdf
- https://www.farnell.com/datasheets/410427.pdf

6.3 Programación en Matlab

- https://es.mathworks.com/help/
- http://ocw.uniovi.es/pluginfile.php/650/mod_resource/content/1/1C_C11666_0910/practicas/curso09_10/Practica6.pdf
- https://www.youtube.com/playlist?list=PLjApqg2Zsw31HHJONnV9IQkxTQfUrW_9F.

6.4 Programación en Arduino

- https://www.arduino.cc/reference/en/
- https://forum.arduino.cc/index.php?topic=327480.0
- http://docs.macetech.com/doku.php/centipede_shield
- https://playground.arduino.cc/code/timer1
- https://geekytheory.com/matlab-arduino-serial-port-communication
- https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Estructuras_y_Uniones
- https://stackoverflow.com/questions/6160963/how-can-i-digitalread-a-pin-that-is-in-pinmode-output

6.5 Maquetación

- https://wordexperto.com/2016/01/31/crear-un-contenedor-para-numerar-ecuaciones-numbering-equations/

7 ANEXOS

7.1 ANEXO 1 - FUNCIONAMIENTO DEL INTERFAZ CREADO EN MATLAB.

Durante la sección de programación en Matlab se ha mencionado que, junto a la programación de la clase vcsel, también se ha creado un interfaz gráfico que utiliza una serie de botones para ejecutar internamente métodos de esta clase y lograr enviar órdenes a la matriz de láseres que se desea controlar sin necesidad de pasar por la línea de comandos de Matlab.

Sumado a los métodos programadas en la clase vcsel, también se han incluido en este interfaz diversas funcionalidades extra como, por ejemplo: Guardar y Cargar patrones de encendido o Programar secuencias de encendido.

Tras esta pequeña introducción pasamos a describir con más detalle el funcionamiento de cada parte que compone el interfaz.

7.1.1 Pantalla del interfaz

Una vez ejecutado el archivo .m que contiene todo el código del interfaz se nos presenta la siguiente ventana.

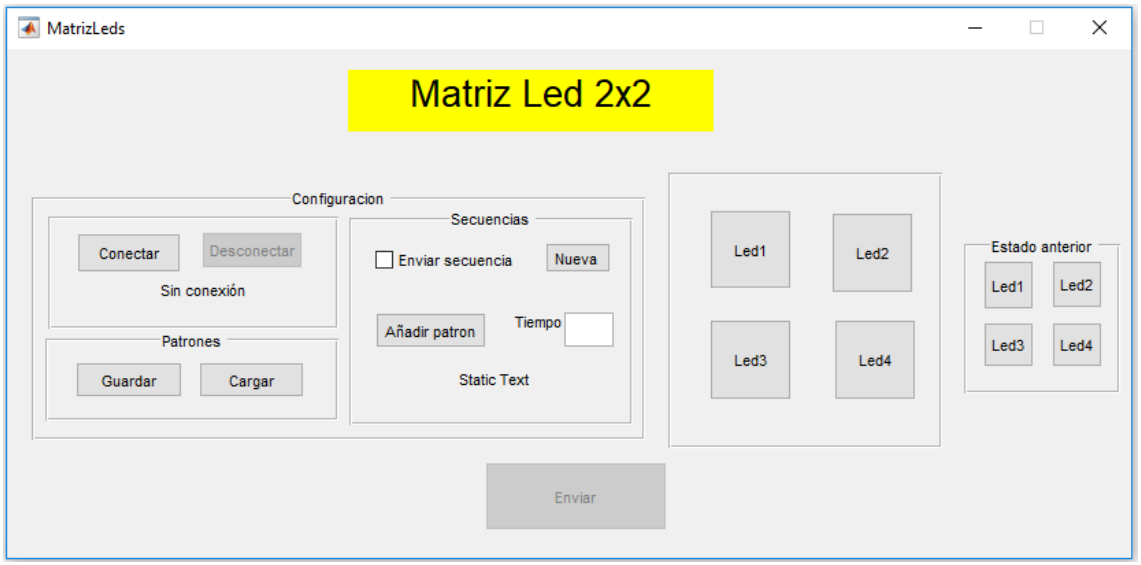

Ilustración 18. Ventana del interfaz

El interfaz se divide en dos partes: una dedicada a la configuración y otra dedicada al patrón de encendido que se desea enviar a la matriz.

En la parte de configuración podremos realizar diversas funciones como conectarnos a la placa Arduino y desconectarnos, guardar y cargar patrones o programar secuencias. Nótese que los botones de **desconectar** y **enviar** no están disponibles. Esto se debe a que el sistema no permite desconectarse ni enviar órdenes al Arduino sin antes conectarse a él. Debajo de los botones de

desconectar y conectar se encuentra un indicador de estado de la conexión entre la placa y el PC.

Una vez establecida la conexión, el botón **conectar** se bloqueará y se desbloquearán los de **desconectar y enviar.**

7.1.2 Patrones

Es posible guardar y cargar patrones de encendido mediante los botones destinados para ello.

Por un lado, al pulsar el botón de **Guardar Patrón** se abre un explorador de archivos para seleccionar la ruta y el nombre del archivo en el que se desea guardar el patrón que tiene la matriz en ese momento. Una vez hecho esto, el programa crea un archivo txt con el nombre escrito anteriormente y guardar en su interior el patrón de encendido.

Por otro lado, al pulsar el botón **Cargar Patrón** también se abre el explorador de archivos en el que se puede seleccionar un archivo txt que contenga un patrón guardado con anterior. Tras seleccionar un archivo, el programa lo abre, analiza el patrón incluido en él y actualiza el estado de la matriz.

En el caso del interfaz que se muestra en la Ilustración 18 resultan menos prácticos estos botones debido a la escasa cantidad de leds que componen la matriz. No obstante, con la matriz de 64 leds que se pretende controlar en última instancia, sí que resulta realmente útil.

7.1.3 Secuencias

Otra de las nuevas funcionalidades que incluye este interfaz consiste en programar secuencias de encendido.

Dentro de su propia sección encontramos 5 elementos:

1. **Checkbox**: Al marcarla indica al sistema que enviaremos una secuencia cuando pulsemos el botón **enviar** más adelante
2. Botón **Nueva:** Indica que se inicia una secuencia nueva.
3. Botón **Añadir patrón:** Añade el patrón actual de la matriz a la secuencia.
4. **Tiempo**: Indica el tiempo que tiene que esperar el sistema antes de enviar el siguiente patrón de la secuencia.
5. **Texto informativo:** Informa del patrón añadido a la secuencia cada vez que se presiona el botón **Añadir patrón.**

Se pueden añadir hasta 64 patrones por cada secuencia.

7.1.4 Patrón de encendido

A la derecha de la parte de configuración encontramos dos matrices 2x2, una moderadamente más grande que la otra. A pesar de que los botones que forman cada una de ellas se pueden pulsar, sólo los de la matriz grande generan una respuesta ya que es la que utilizaremos para definir los

patrones que se mencionan en las funcionalidades de los botones **Guardar Patrón, Cargar Patrón y Añadir Patrón.** Cada vez que se pulsa un botón esta matriz pueden ocurrir dos cosas:

1. Si el botón estaba de color de gris, cambia su color a verde y ejecuta internamente el método **encenderLaser(posx,posy).**
2. Si el botón estaba de color verde, cambia su color a gris y ejecuta internamente el método **apagarLaser(posx,posy).**

La función de la pequeña matriz es meramente informativa. Cuando pulsamos el botón **enviar** esta matriz se actualiza y muestra el patrón enviado. De esta forma podemos modificar la matriz grande para enviar otro patrón teniendo como referencia el ultimo enviado.