UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE TECNOLOGÍA
CARRERA DE ELECTRONICA Y TELECOMUNICACIONES

"SISTEMA DE ALARMA ANTIRROBO PARA EL HOGAR UTILIZANDO LA PLATAFORMA ARDUINO"

Trabajo de aplicación – Examen de Grado presentado para obtener el Grado de Licenciatura

POR: EFRAIN PATRICIO MENDOZA ALVARADO

LA PAZ – BOLIVIA

Noviembre, 2021

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE TECNOLOGÍA
CARRERA DE ELECTRÓNICA Y TELECOMUNICACIONES

Trabajo de Aplicación – Examen de Grado

"SISTEMA DE ALARMA ANTIRROBO PARA EL HOGAR UTILIZANDO LA PLATAFORMA ARDUINO"

Presentado por: EFRAIN PATRICIO MENDOZA ALVARADO

Para optar del grado académico de Licenciado en Electrónica y Telecomunicaciones

Nota numeral: …….

Nota literal: …….

Ha sido …….

M.Sc. Luis Richard Marquez Gonzales

Director de la Carrera de Electrónica y Telecomunicaciones

Tutor:

Tribunal Lic. Julia Torrez Soria

Tribunal Ing. Luis Ramiro Velarde Chávez

Tribunal Lic. Edder Tomás Jurado Moya

DEDICATORIA

A mi Esposa, Giovana Vargas, quien siempre me brindó su apoyo en todo momento y con quien he crecido como persona madura y espiritual.

A mis padres, Juan Mendoza y Elizabeth Alvarado porque siempre me brindaron su apoyo y guía incondicional para seguir avanzando en la vida.

A mis hermanos America y Adrian, porque siempre me acompañaron me vieron crecer y me animaron a seguir en todo el proceso de estudio en la Carrera.

AGRADECIMIENTO

Principalmente a Dios, por darme un día más de vida y permitirme realizar este trabajo.

A la Facultad de Tecnología por permitirme estudiar todo lo necesario en sus predios.

A los docentes de la Carrera de Electrónica y Telecomunicaciones, por darme las herramientas necesarias para desempeñarme como tecnólogo y profesional.

A mis compañeros de la carrera, por impulsarme a seguir adelante, en especial a mi compañero Favio por no dejar que desista de seguir avanzando.

ÍNDICE

RESUMEN

En la actualidad, los robos a las viviendas no hecho sino aumentar. Los ladrones se dan modos para ingresar a los domicilios de manera no autorizada, saqueando todo lo que pueden.

Una buena solución sería contar con un Sistema de Alarma antirrobo que reducirá de manera considerable el riesgo de sufrir un robo en nuestro hogar. Con este Sistema, ya se puede prevenir de un robo o ingreso no autorizado.

En el presente Trabajo de Aplicación se plantea una solución, la implementación de un Sistema de Alarma antirrobo para el hogar utilizando la plataforma Arduino, que lo hace un Sistema económico, fácil de armar y configurar.

Este Sistema de Alarma antirrobo para el hogar contará con la plataforma Arduino UNO como central de Alarma, dos sensores de movimiento PIR que detectarán justamente el ingreso de una persona al domicilio, contará con un teclado y display LCD, que son periféricos que nos ayudarán a Configurar la Central de Alarma y ver los estados de configuración. También contará con una Sirena de Alarma para que al sonar, cumpla con la función de avisar del posible robo.

CAPITULO I
PLANTEAMIENTO DEL PROBLEMA

1.1 PLANTEAMIENTO DEL PROBLEMA

A medida que va pasando el tiempo, la calidad de vida de las personas no ha hecho sino aumentar, en todo sentido. Con el aumento de la tecnología, se cuentan con muy buenas innovaciones y modos de realizar muchas actividades buenas, pero lamentablemente también se realizan muchas actividades malas, como el robo en los hogares, que al parecer va en aumento.

A raíz de que ahora las personas siempre ocupan su tiempo en el trabajo, muchas de estas personas, incluso, familias enteras salen de sus hogares para desempeñar sus actividades, colegio, universidad, trabajo, entre otras. A causa de esta dinámica de las personas, se dejan muchas veces las casas deshabitadas, lo cual es aprovechado por los amigos de lo ajeno.

Justamente cuando una casa está deshabitada, se vuelve vulnerable ante el posible robo por los ladrones, además, como se mencionó antes, ellos cuentan con una variedad de habilidades, dándose modos de entrar a dichos hogares, entonces una vez ingresen al hogar, robarán cuanto puedan.

Aunque existen zonas que cuentan con guardias policiales o privados que cuidan las calles, y, por ende, los hogares, estos pueden ser burlados o incluso atacados por los ladrones, entonces el riesgo, aunque es menor, sigue presente. Además, en caso de que los guardias sean burlados, no existe un sistema o alguien que les avise de manera rápida ni ruidosa.

1.2 JUSTIFICACIÓN DEL TRABAJO

La necesidad de realizar este trabajo de aplicación se debe a que, con todo lo mencionado arriba, es necesario contar con un sistema que avise a las personas sobre un posible robo

o ingreso no autorizado a los domicilios, al contar con dicho sistema, las personas podrán ser avisadas si hay un ingreso no autorizado, y si es ruidoso, los vecinos y los guardias (si los hay), estarán atentos para poder actuar en consecuencia.

Con este trabajo de aplicación, podremos reducir mucho más el riesgo de que los amigos de lo ajeno ingresen a los domicilios a robar. Ni bien ingresen al domicilio, estos serán detectados por los sensores de movimiento que avisarán al sistema que han entrado, y después procederá a hacer mucho ruido con una sirena, todos escucharán e incluso, los ladrones se asustarán y lo pensarán antes de continuar.

El trabajo de aplicación es una muy buena opción que complementa con los sistemas de seguridad como las cámaras de vigilancia. Además, este trabajo es fácil de operar, pero al mismo tiempo es una buena solución casera para estar protegidos.

Este trabajo de aplicación se realiza por su fácil armado, utilizamos la plataforma Arduino, por su bajo costo, lo hace una buena opción para protegernos lo más pronto posible. Su operación es sencilla y se puede optimizar.

1.3. OBJETIVOS

1.3.1. General

Implementar un Sistema de Alarma antirrobo para el hogar utilizando la plataforma Arduino.

1.3.2. Específicos

Desarrollar el hardware del Sistema de Alarma antirrobo, utilizando como dispositivo central al Arduino UNO.

Desarrollar el software (entorno de programación) utilizando el entorno IDE de Arduino para cumplir con la función principal del Sistema de Alarma.

Configurar un teclado matricial para que pida contraseña de acceso a las personas autorizadas para activar o desactivar el Sistema de Alarma.

Configurar un display LCD para que se pueda visualizar los estados del Sistema de Alarma y datos que ingrese la persona para activar o desactivar el mismo.

CAPITULO II

FUNDAMENTO TEORICO

2.1. SISTEMA DE ALARMA

2.1.1. Breve historia de los Sistemas de Alarma

El concepto de "sistemas de alarmas" remonta su origen a principios de los años treinta a consecuencia del incremento de nuevas modalidades delictivas que afectaban a la comunidad. El incremento de robos hacia los hogares y comercios comenzó a convertirse en dicha época en uno de los problemas más urgentes a resolver por las autoridades. (Academy Xperts Bolivia (2018) Alarmas de Seguridad. Recuperado de Experto en Seguridad Electrónica Módulo 2)

El Sr. Tildesley, un inventor inglés fue el primero en construir un modelo aceptable de las alarmas antirrobo. Relacionó unas campanas con la cerradura de la puerta e hizo que el sistema sonara cuando alguien tocara o cuando se tratara de abrir la cerradura. (Academy Xperts Bolivia (2018) Alarmas de Seguridad. Recuperado de Experto en Seguridad Electrónica Módulo 2)

La innovación se dio en 1850 al hacerlo funcionar con electricidad y el uso de imanes por medio de cables que conducían la corriente eléctrica hacia dichos imanes. Al colocar los imanes en las puertas se establecía un circuito de corriente eléctrico cerrado y al ser abierta la puerta se producía el sonido de un timbre. (Academy Xperts Bolivia (2018) Alarmas de Seguridad. Recuperado de Experto en Seguridad Electrónica Módulo 2)

Después de la Segunda Guerra Mundial se convirtió en un sistema más barato y capaz de adaptarse con facilidad y rapidez a diversas funciones. Para mediados de los 90's se adoptó como un modelo en los sistemas de alarmas convencionales. Con la avanzada tecnología aparecieron los sistemas inalámbricos. (Academy Xperts Bolivia (2018) Alarmas de Seguridad. Recuperado de Experto en Seguridad Electrónica Módulo 2)

2.1.2 ¿Qué es un sistema?

Un sistema es la combinación de elementos que actúan de manera conjunta para lograr un objetivo. Dicho sistema se forma con partes que cumplen una función específica, trabajando de manera periódica y conjunta para alcanzar dicho objetivo.

2.1.3 ¿Qué es una alarma?

Podemos entender que una alarma es aquella señal o aviso que nos advierte de un posible peligro o un evento importante, y así poder actuar en consecuencia. En este caso, una alarma es un aviso contra una entrada no autorizada a nuestro domicilio.

2.1.4 Sistema de Alarma

"Un sistema de Alarma es como tener un Doctor de confianza en la familia: Es bueno tenerlo, pero esperar nunca necesitarlo." *(https://www.productosintegra.com/wp-content/uploads/2018/12/que-es-un-sistema-de-alarma.pdf)*

Un Sistema de Alarma es el conjunto de elementos de seguridad pasiva, que cumplen con el objetivo de avisar sobre un peligro o evento importante, otra vez, en este caso, dicho sistema nos avisa de una entrada no autorizada a nuestro domicilio. No evitan el peligro, pero sí que nos avisa, por lo tanto, podemos actuar en consecuencia tomando todas las previsiones.

El mejor sistema de alarma es aquel que combinará la protección del perímetro y el interior del domicilio. Cada puerta o ventana deben ser protegidas con algún tipo de sensor. El interior de su propiedad se protege utilizando sensores de movimiento. Los más comunes detectan la presencia de alguna persona comparando la temperatura del cuerpo con la temperatura ambiente de la habitación.

En general podemos definir a un sistema de alarma, como el conjunto de elementos e instalaciones que son necesarias para proporcionar a las personas y a sus bienes materiales la adecuada protección en caso de que un suceso inesperado como un robo, sabotaje, incendio, etc, ocurra.

2.1.5 Partes de un Sistema de Alarma

Un sistema de alarmas está compuesto principalmente de:

- Entrada (Sensores)
- Procesamiento
- Salida

Cada elemento cumple su función específica para que el Sistema de Alarma cumpla con el objetivo de advertir de un suceso.

2.1.6 Entrada (Sensores)

Es aquel sector donde se hace la detección del peligro o evento, para lograr dicha detección, se utiliza Sensores.

2.1.7 Sensores

También llamados Detectores, son aquellos dispositivos que justamente "detectan" un fenómeno físico, dependiendo del caso, se puede detectar humo, movimiento, proximidad, luz, etc.

2.1.8 Tipos de Sensores

2.1.8.1 Sensor de Humo

Un sensor de humo aquel dispositivo que detecta la presencia de humo en el aire y emite una señal acústica para avisar un peligro de incendio. Dependiendo del método de detección, existen diversos tipos:

Detectores iónicos: Usados para la detección de gases y humos de combustión que no son visibles a simple vista.

Detectores de humo: Detectan los humos visibles mediante la absorción o difusión de la luz.

Pueden ser de dos tipos, según se detecte el humo por oscurecimiento o por dispersión del aire en un espacio:

De rayo infrarrojo, compuestos por un dispositivo emisor y otro receptor. Al oscurecerse el espacio entre ellos debido al humo sólo una fracción de la luz emitida alcanza al receptor provocando que la señal eléctrica producida por éste sea más débil y se active la alarma.

De tipo puntual, en los que emisor y receptor se ubican en la misma cámara, pero no se ven al formar sus ejes un ángulo mayor de 90º y estar separados por una pantalla, por lo tanto, el rayo emitido no alcanza el receptor. Cuando el humo entra en la cámara el haz de luz emitido, se refracta en las partículas de humo y puede alcanzar al receptor, lo cual provoca la activación de la alarma.

Figura 1: *Sensor de Humo*

Fuente: *https://prevencionrimac.com/riesgospatrimoniales/articulo/Detectores-Automaticos-De-Humo*

2.1.8.2 Sensor de Movimiento

Los sensores infrarrojos pasivos PIR (Passive Infrared) detectan el movimiento de los cuerpos que desprenden calor y rayos infrarrojos (por medio de un sensor piro eléctrico y con un lente de Fresnell que concentra los rayos infrarrojos hacia el sensor). Básicamente reciben la variación de las radiaciones infrarrojas del medio ambiente que cubre.

Figura 2: *Sensor de Movimiento PIR*
Fuente: *https://www.tecnoseguro.com/faqs/alarma/que-es-un-detector-de-movimiento-pasivo-o-pir*

2.1.8.3 Sensor Magnético

Se trata de un sensor que forma un circuito cerrado por un imán y un contacto muy sensible que, al separarse, cambia el estado (se puede programar como NC o NA) provocando un salto de alarma. Se utiliza en puertas y ventanas, colocando una parte del sensor en el marco y otra en la puerta o ventana.

Figura 3: *Detector Magnético*
Fuente: *https://www.tecnoseguro.com/faqs/alarma/que-es-un-detector-magnetico-de-apertura*

2.1.8.4 Pulsadores de asalto (Botón de Pánico)

Tan solo son pulsadores de alarma de contactos secos que permite activar el sistema sin que suene la sirena en caso de asalto. Se pueden instalar en los baños, bajo mesas, etc. También existe el receptor con llavero inalámbrico, que puede ser usado como botón de pánico o activar desactivar la alarma.

Figura 4: *Receptor con llavero inalámbrico y Botón de Pánico*
Fuente: *https://www.monografias.com/trabajos-pdf5/sistema-alarmas/sistema-alarmas.shtml*

2.1.9 Procesamiento

2.1.9.1 Central de Alarma

La central de alarma es el cerebro de todo el sistema. Esta se encarga de recibir las señales de los sensores y tomar acciones como activar una sirena, un emisor telefónico, etc.

Las centrales de alarma se caracterizan por contar en su mayoría con zonas (entradas), teclado, salida de alarma, fuente de alimentación (transformador), batería, salida PGM, códigos de usuario, etc.

2.1.9.2 Zonas de un Sistema de Alarma

Las zonas representan cada detector que se instaló, en un determinado lugar o área. Permiten el ordenamiento de la protección en su conjunto (por ejemplo, la zona uno es generalmente la entrada principal).

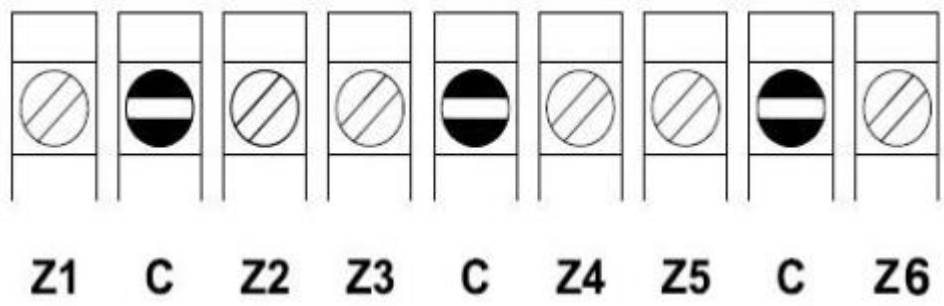

Figura 5: *Bornes de distribución para zonas en un Sistema de Alarma*
Fuente: *http://repositorio.umsa.bo/xmlui/handle/123456789/21656*

2.9.1.3 Central procesadora.

Es la CPU del sistema. En ella se albergan la placa base, la fuente y la memoria central. Esta parte del sistema es la que recibe las diferentes señales que los diferentes sensores pueden emitir, y actúa en consecuencia, disparando la alarma, comunicándose con la central por medio de un modem, etc.

Figura 6: *Central de Alarma y procesadora*
Fuente: *https://www.monografias.com/trabajos-pdf5/sistema-alarmas/sistema-alarmas.shtml*

2.9.1.4 Teclado

El teclado es aquel dispositivo que permite realizar las programaciones de la central de alarma, así como también realizar el control del sistema como activación, desactivación, cancelación de zonas, etc.

Su función principal es la de permitir a los usuarios autorizados armar (activar) y desarmar (desactivar) el sistema, todo por medio de un código de acceso.

Los teclados podrán ser con leds (luces) indicadores o con display alfanumérico que a través del mismo se visualizarán palabras con indicaciones de fácil lectura.

Un sistema podrá disponer de más de un teclado como también podrá conectársele un control remoto para realizar activaciones y desactivaciones a distancia.

Figura 7: *Teclado para alarma*
Fuente: *https://tecnosinergia.zendesk.com/hc/es/articles/115003753731--C%C3%B3mo-consultar-los-eventos-de-alarma-NX-en-teclado-LCD-NX-148-*

2.1.10 Salida

2.1.10.1 Sirena de alarma

La **sirena de alarma** es probablemente una de los primeros elementos en los sistemas de seguridad de la historia. Su función principal tiene un carácter más disuasorio que es la de alertar para prepararse ante la intrusión, con esto, se busca disuadir al ladrón de continuar con el robo.

Figura 8*: Sirena de alarma*
Fuente*: https://www.monografias.com/trabajos-pdf5/sistema-alarmas/sistema-alarmas.shtml*

2.1.11 Otros elementos

2.1.11.1 Baterías de respaldo

La batería es utilizada para darle respaldo eléctrico al sistema, en caso de fallas en el suministro de energía, este puede ser de entre 12 horas hasta de 72 horas dependiendo del tipo y estado. Esto asegura el funcionamiento normal de un Sistema de Alarma.

Figura 9: *Batería de respaldo*
Fuente: *https://www.monografias.com/trabajos-pdf5/sistema-alarmas/sistema-alarmas.shtml*

2.1.11.1 Salida PGM

Generalmente es una salida de colector abierto y que cambia de estado según la programación que se le asigne. Cuando ocurre un evento específico, la PGM puede restaurar los detectores de humo, activar las luces estroboscópicas, abrir o cerrar puertas de garajes y más opciones.

2.2 Arduino

2.2.1 ¿Qué es Arduino?

Arduino es una plataforma libre, educativa y de desarrollo. Generalmente, nos referimos a Arduino como el hardware físico (también conocido como tarjeta, placa o PCB), sin embargo, es mucho más que eso. Una definición más exacta es la de **plataforma** formada por: un hardware, un software (o entorno de programación) y un lenguaje de

programación. La función de la plataforma Arduino es facilitar el uso de un microcontrolador (MCU). A partir de aquí, abreviaremos al microcontrolador como MCU.

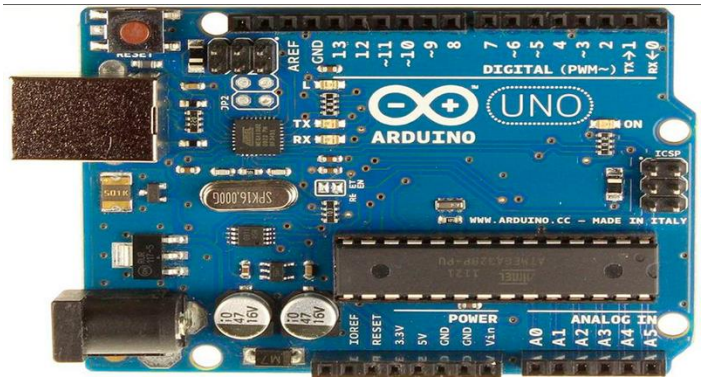

Figura 10: *Plataforma Arduino UNO*
Fuente: *https://internetpasoapaso.com/arduino-uno/*

2.2.2 Microcontrolador

El corazón de Arduino es un microcontrolador. El MCU es un pequeño ordenador en un chip. Dependiendo del tipo de MCU, contiene de un procesador, memoria **RAM** (memoria de acceso aleatorio) para contener datos, memoria **EPROM** (ROM programable borrable) o de memoria **Flash** para contener nuestros programas, y tiene pines de entrada y salida. Estos pines de entrada/salida son los que conectan el microcontrolador con el resto de componentes electrónicos.

Las entradas pueden leer señales digitales (¿el interruptor está abierto o cerrado?) y analógicas (¿cuál es la tensión en un pin?). Esto nos permite conectar innumerables tipos de sensores de luz, temperatura, sonido, etc.

Las salidas también pueden ser analógicas o digitales. Así, se puede establecer que un pin esté activado o desactivado (5 V o 0 V) y esto puede encender o apagar los LED directamente, o se puede usar la salida para controlar dispositivos más potentes, como motores. También pueden proporcionar tensión de salida analógica. Es decir, se puede fijar la salida de un pin a una determinada tensión, lo que permite controlar la velocidad

de un motor o el brillo de una bombilla, por ejemplo, en lugar de solo encenderlo o apagarlo.

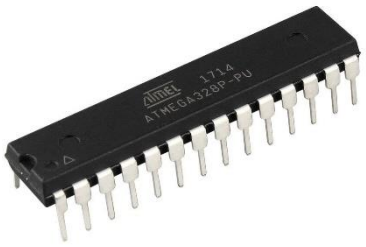

Figura 11*: Microcontrolador AT Mega 328p*
Fuente: *https://conmicrocontroladores.com/microcontroladores-arduino/*

2.2.3 Ventajas y Desventajas de Arduino

2.2.3.1 Ventajas

- Toda la plataforma viene lista para utilizarse.
- Entorno de programación compatible con Windows, Linux y Mac OS.
- Facilita la realización de proyectos electrónicos.
- Configuración de software y hardware de manera fácil e inmediata.
- Cuenta con librerías y shields para realizar gran cantidad de proyectos.

2.2.3.2 Desventajas

- Poder de procesamiento limitado, frente a otros MCU con más prestaciones.
- A raíz de que el Arduino ya viene armado, tiene una flexibilidad limitada porque debemos adecuarnos a los pines, shields y librerías para desarrollar un proyecto.
- Dependiendo del tipo de Arduino y como mencionamos anteriormente, las limitaciones de Arduino impedirían con la realización otros proyectos donde, ciertos parámetros solo pueden cubrirse utilizando MCU.

- El costo de los shields puede hacer el proyecto más costoso.

2.2.3 Arduino UNO

2.2.3.1 Características

A continuación, hablemos de las características que tiene este Arduino UNO.

Microcontrolador Atmega328P: MCU tipo AVR con CPU de hasta 16 MHz, 32 KB de memoria Flash, 2KB de Memoria SRAM y 1 KB de memoria EEPROM.

Voltaje de Operación 5V: tensión máxima con la que puede trabajar el MCU del Arduino, por lo tanto, es la tensión máxima con la que contamos para realizar nuestro proyecto.

Voltaje de entrada (recomendado) 7-12V: como ya se mencionó, en la mayoría de las placas de Arduino han incorporado un conector Jack para alimentar un Arduino a través de una pila, batería o cargador. Siempre que se respete los límites de voltaje, esta opción nos puede ser muy útil.

Voltaje de entrada (límite) 6-20V: estos límites definen la tensión máxima y mínima a la que podemos alimentar nuestro Arduino, debemos evitar llegar estos límites para que nuestra placa funcione sin sobrecalentamientos o anomalías.

Pines Digitales E/S 14 (de los cuales 6 dan salidas PWM): el Arduino UNO dispone de 14 pines digitales que pueden comportarse como entradas o salidas, por ser digitales los pines pueden estar “encendidos” o “apagados”. Con esto, nosotros podemos prender o apagar leds (en caso de estar configurado como salida) o detectar un pulso de un pulsador (en caso de estar configurado como entrada). Además, con las 6 salidas que son PWM podemos regular un motor o una luz.

Pines de Entrada Analógica 6: con estos pines podemos tomar lecturas de señales que varían entre 0 y 5V. Por ejemplo, cuando usamos sensores que devuelven un valor de tensión en función de la cantidad de magnitud física medida.

Pin de Corriente DC por E/S 20mA: limitación de intensidad que puede entrar o salir por cada pin de nuestro Arduino.

Pin de Corriente DC para 3,3V 50mA: todos los pines de un Arduino cuentan con una etiqueta, que nos indicará su función. En la parte de potencia del hardware de nuestro Arduino UNO tenemos un pin de 3,3V que ofrece esa tensión constantemente y a diferencia del resto de pines permite hasta 50mA de corriente DC.

Memoria Flash 32 KB (ATmega328P) del cual 0,5 KB se usa para el bootloader: La memoria FLASH es en donde se almacena el programa que carguemos en nuestro Arduino, que en el caso del Arduino UNO tiene un límite de grabaciones de 10.000 veces. En cuanto los 0,5 KB para el bootloader, es un gestor de arranque que permite que los MCU de Arduino funcionen en la plataforma Arduino.

SRAM 2 KB (ATmega328P): esta memoria es la que utiliza el MCU para trabajar con los datos temporales, es decir, se las necesita en cada momento para realizar las operaciones de programación. Si se apaga el Arduino, los datos desaparecen. Esta memoria no tiene límite de escrituras.

EEPROM 1 KB (ATmega328P): esta memoria permite almacenar datos y aunque se apague el Arduino podemos recuperarlos una vez se encienda, tiene un límite de 100.000 escrituras.

Velocidad de Reloj 16 MHz: define “la rapidez” con la que opera un Arduino y también interviene en temporizaciones que realizamos con Arduino.

LED_BUILTIN 13: Nuestro Arduino cuenta con un led en la placa asociado al estado del pin 13.

Longitud 68,6 mm: el largo del Arduino UNO.

Ancho 53,4 mm: el ancho del Arduino UNO.

Peso 25g: peso del Arduino UNO.

2.2.4 Otros tipos de Arduino

Ahora, hablemos de otros tipos de Arduino y sus características principales con respecto a Arduino UNO:

Genuino 101: Voltaje de trabajo 3,3V, pero permite 5V por los pines, 4 pines PWM, dispone de acelerómetro, giroscopio y bluetooth.

2560: 54 pines I/O digitales, 15 pines PWM, 16 entradas analógicas.

Micro: 20 pines I/O digitales, 7 pines PWM, 12 entradas analógicas.

MKR1000: voltaje de trabajo 3,3V ,8 pines I/O digitales, 12 pines PWM, 7 entradas analógicas, 1 salida analógica, conexión por WIFI y 7mA por pin.

PRO: voltaje de trabajo 3,3V o 5V y 40mA de limitación por pin.

PRO MINI: voltaje de trabajo 3,3V o 5V 40mA de limitación por pin, no incluye programador.

Genuino ZERO: voltaje de trabajo 3,3V ,20 pines I/O digitales, 10 pines PWM,1 salida analógica y 7mA de limitación por pin.

DUE: voltaje de trabajo 3,3V ,54 pines I/O digitales, 12 pines PWM, 12 entradas analógicas, 2 salidas analógicas, 130mA de limitación por todos los pines, salvo 800mA por 3,3V y 5V (cada uno).

ETHERNET: 14 pines I/O digitales, 4 pines PWM, 6 entradas analógicas, 40mA de limitación
por pin, conexión a internet a través de cable de red y posibilidad de conectar una tarjeta microSD.

LEONARDO 10: 20 pines I/O digitales, 7 pines PWM, 12 entradas analógicas y 40mA de limitación por pin.

MEGA ADK: 54 pines I/O digitales, 15 pines PWM, 16 entradas analógicas, 40mA de limitación por pin un conector USB para comunicación con dispositivos Android.

MINI:8 entradas analógicas y hasta 40 mA por pin.

NANO: 22 pines I/O digitales, 12 pines PWM, 8 entradas analógicas y hasta 40mA de limitación por pin.

YÚN: 20 pines I/O digitales, 7 pines PWM, 12 entradas analógicas, hasta 40mA de limitación
por pin y conexión a internet por WIFI o cable.

2.2.5 Shields de Arduino

Podríamos decir que los Shields de Arduino son hardware extra, que, en combinación de las librerías, sirven para ayudarnos con los proyectos electrónicos que queramos realizar. Hay una amplia variedad de Shields a día de hoy, sin embargo, hablaremos de los más comunes:

El Arduino Motor Shield permite conectar motores paso a paso, relés y solenoides a un Arduino.

Arduino USB Host Shield ofrece un interfaz para poder conectar dispositivos USB.

El Arduino 4 Relays Shield incoprota 4 salidas a relé que nos permite aumentar la potencia de salida de ciertos pines del Arduino.

ARDUINO WIRELESS SD SHIELD: integra una tarjeta SD y la posibilidad de conectar un módulo XBEE.

Arduino Ethernet Shield 2 conecta un Arduino a Internet.

El Arduino GSM Shield V2 permite conectar un Arduino a Internet mediante una red de telefonía o envío y recepción de llamas y mensajes.

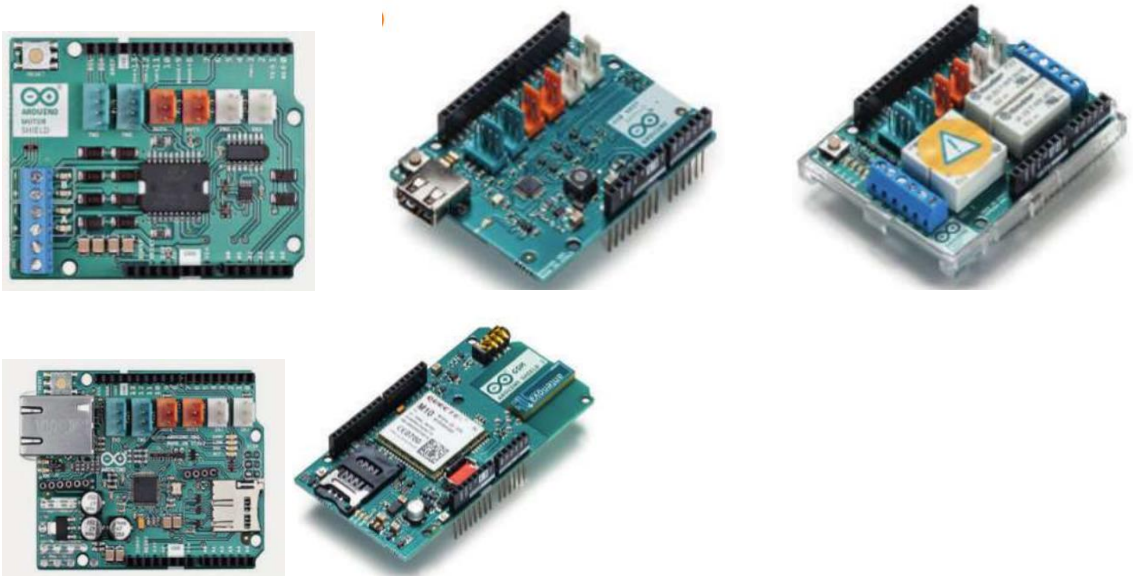

Figura 12: *Shields de Arduino (Motor Shield, USB Host Shield, Shield 4 Relés, Ethernet Shield 2, GSM Shield V2)*
Fuente: *Beiroa, Rubén. (2019). Aprender Arduino, electrónica y programación. Ciudad de México Alfaomega Grupo Editor*

2.2.6 Entorno de programación

2.2.6.1 Lenguaje Máquina

Hablemos del entorno de programación (idioma artificial diseñado para expresar instrucciones que pueden ser llevadas a cabo por una máquina), podemos diferenciar varios niveles. El lenguaje a más bajo nivel sería el lenguaje máquina o assembler (ensamblador): es el lenguaje capaz de almacenar e interpretar una máquina, en nuestro caso, un MCU.

Lo malo es que, usando lenguaje máquina nos puede dar ciertos inconvenientes. Por ejemplo, debemos memorizar cada instrucción para un MCU, que puede ser diferente de un modelo a otro, lo cual nos dificulta en su memorización, además, si hay errores, debemos buscar la instrucción incorrecta, lo cual nos llevaría a buscar y examinar cada instrucción, demorando tiempo considerable.

2.2.6.2 Lenguaje Estructurado

A causa de todo esto, se empezó a evolucionar en la programación hasta llegar a lo que conocemos a día de hoy como lenguaje a alto nivel o lenguaje estructurado. Existe una variedad de lenguajes a alto nivel: C, Python, C++, Java, etc. El lenguaje a alto nivel se compone de instrucciones más complejas que el lenguaje máquina, lo que significa que una instrucción en este lenguaje puede equivaler a varias en lenguaje máquina. Eso solo puede representar en un trabajo simplificado y menos instrucciones para un proyecto en comparación con el lenguaje máquina.

2.2.6.3 Lenguaje IDE de Arduino

El lenguaje a alto nivel que emplea el IDE de Arduino está basado en el lenguaje C, sin embargo, está enfocado para la plataforma Arduino ya que cuenta con instrucciones propias para este. La carga de un programa al MCU del Arduino conlleva una serie de pasos que se conoce como compilación. La compilación se puede considerar como una

traducción del programa que hemos desarrollado a un lenguaje que pueda almacenar e interpretar el MCU.

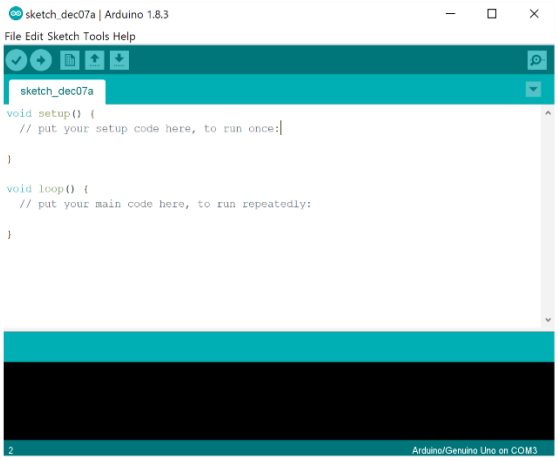

Figura 13: *Entorno Arduino IDE*
Fuente: *https://support.microsoft.com/es-es/topic/cargar-c%C3%B3digo-de-placa-e-ide-de-arduino-a9723765-1314-49e0-a69b-bb5c3e1f628d*

2.2.7 Librerías de Arduino

Las librerías nos sirven para ampliar las funcionalidades de un Arduino e incluso facilitan la programación a la hora de trabajar con módulos (GSM, ETHERNET, WIFI, etc.). Hablemos de las librerías más comunes:

EEPROM: Permite trabajar con la memoria EEPROM del Arduino, es una memoria que almacena los datos aún después de apagarse el Arduino. Puede almacenarse datos como si fuera una memoria SD.

ETHERNET: Incorpora todas las funciones necesarias para poder conectarnos a internet mediante un módulo ETHERNET, configurando el Arduino como un servidor o un cliente.

GSM: Desarrollada para utilizar módulos GSM/GPRS con los cuales podemos conectar nuestro Arduino a una red móvil y conectarnos a internet o simplemente el envío y recepción de mensajes y llamadas.

LiquidCrystal: Dispone de las funciones necesarias para controlar una pantalla LCD, muchos de estos módulos integran un hardware que permite la comunicación con el protocolo I2C con lo que nos ahorramos una gran cantidad de conexiones, pero en este caso la librería sería **LiquidCrystal_I2C**.

SD: Esta librería nos permite almacenar y leer datos en una memoria SD. Incluyendo todas las funciones para crear, buscar eliminar archivos y carpetas con diferentes extensiones.

SERVO: Con ella se pueden controlar servomotores.

SoftwareSerial: Nos permite crear nuevos puertos serie para ampliar el número de dispositivos con los que se puede comunicar a través de este protocolo.

Stepper: Diseñada para trabajar con motores paso a paso.

TFT: Permite dibujar texto, imágenes y detectar la interacción de usuario con una pantalla TFT.

WiFi: Incluye toda la programación que incorpora la librería ETHERNET, solo que trabaja con un módulo WiFi.

Wire: Librería que trabaja con el protocolo de comunicación I2C. Una gran cantidad de sensores y módulos trabajan con este protocolo.

Hay muchas más librerías, las no oficiales, las muy nuevas o incluso, nosotros podemos crearnos nuestras propias librerías.

CAPITULO III

DESARROLLO DEL TRABAJO

3.1 DESCRIPCIÓN DEL TRABAJO

En el presente trabajo se realizará un Sistema de Alarma antirrobo utilizando la plataforma Arduino UNO, contará con 2 sensores de movimiento, teclado para activar o desactivar el sistema de alarma, display LCD para visualizar los estados y condiciones del sistema y una sirena de alarma que sonará una vez la alarma se active.

Recordemos que cada componente del Sistema de Alarma tiene su función, de modo que se pueda cumplir el objetivo de proteger un hogar. Aquí iremos explicando cómo funciona el proyecto, cómo trabaja cada componente, el código de programación y el resultado del proyecto.

3.2 DIAGRAMA DE BLOQUES DEL TRABAJO

A continuación, se muestra el diagrama de bloque por el que está conformado el Sistema de Alarma antirrobo para el hogar.

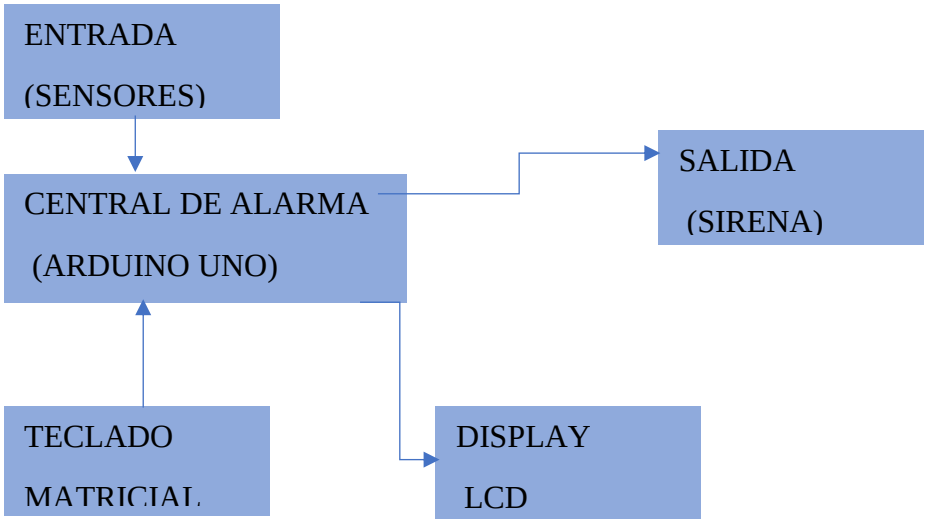

Figura 14: *Diagrama de Bloques del Proyecto*
Fuente: *Elaboración Propia*

En el diagrama se puede apreciar cómo está compuesto el proyecto a grandes rasgos, iremos explicándolos uno a uno.

3.4 DIAGRAMA DE FUJO DEL PROYECTO

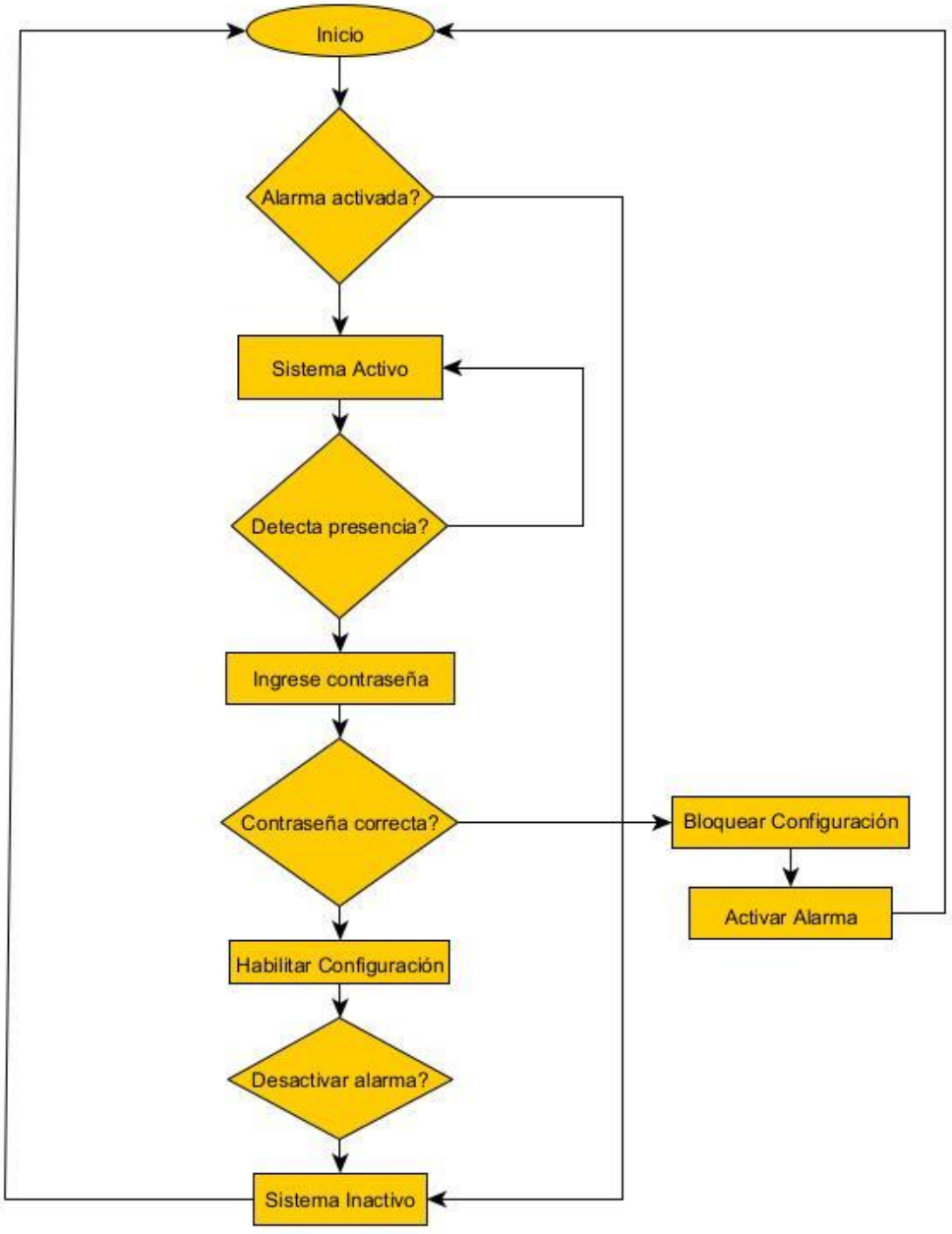

Figura 15*: Diagrama de Flujo del proyecto*
Fuente: *Elaboración propia*

3.5 DIAGRAMA DE CIRCUITO DE SIMULACIÓN EN FRITZING

A continuación, veremos el esquema de circuito del proyecto utilizando el entorno de simulación Fritzing, el cual es muy útil para ver cómo se arma en protoboard los sistemas electrónicos.

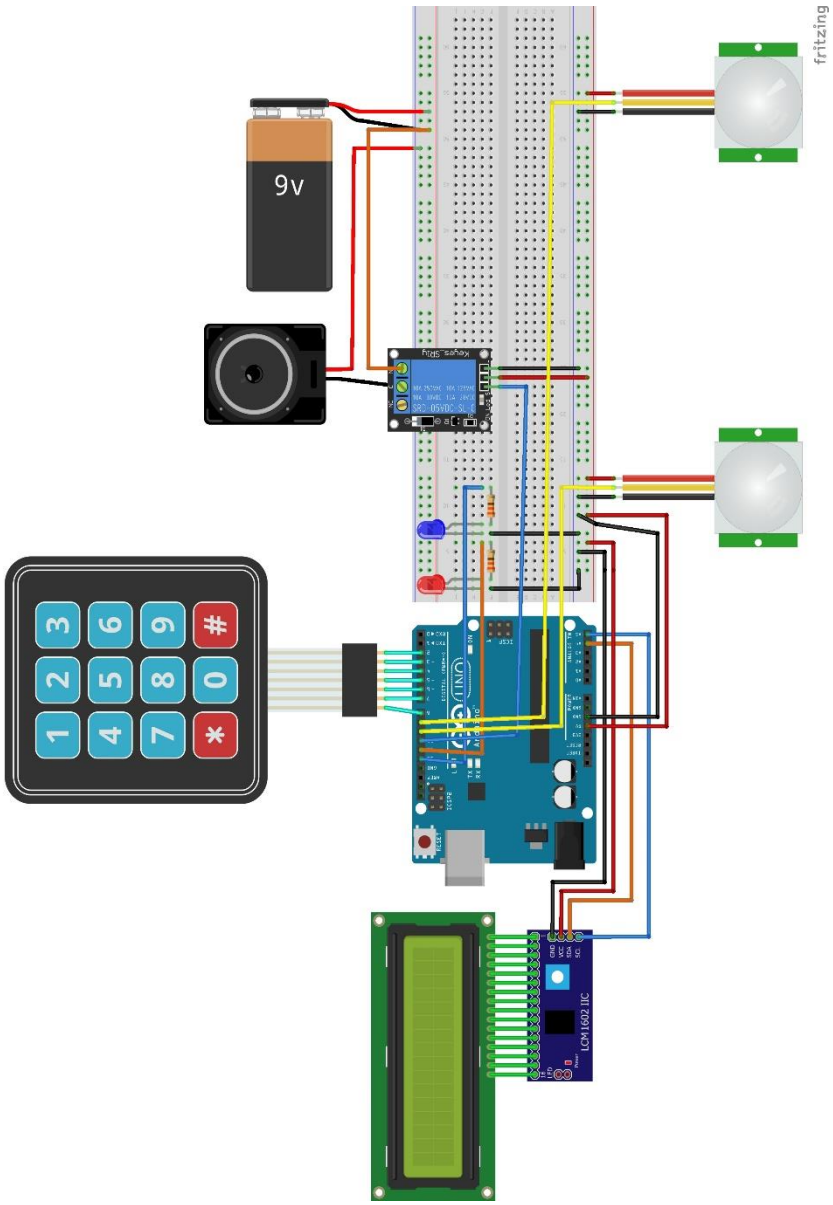

Figura 16: *Circuito del Sistema de Alarmas antirrobo en Simulador Fritzing*
Fuente: *Elaboración propia*

Aclaremos que, en el parlante, el cable negro en Fritzing representa a Vcc y el cable rojo a GND.

3.6 FUNCIONAMIENTO DE CADA COMPONENTE DEL TRABAJO

Sensor PIR para Arduino: El modelo HC-SR501 puede detectar movimiento de 3 hasta 7 metros de distancia, lo cual lo hace suficientemente útil para operar en el hogar, sin nada que envidiar a los sensores PIR industriales. Este sensor de movimiento PIR tiene 3 pines, VCC, OUTPUT y GND, 2 potenciómetros para ajustar la sensibilidad y la demora.

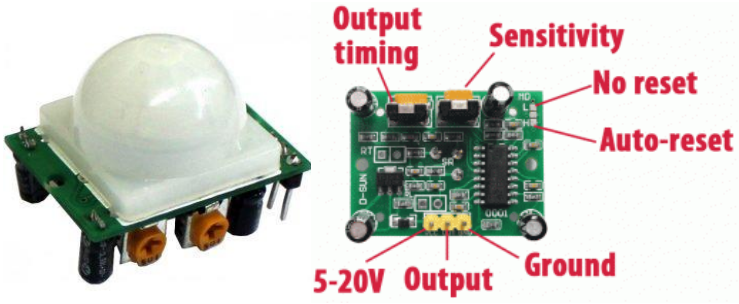

Figura 17: *Sensor PIR HC-SR501 y sus pines.*
Fuente*: https://www.diarioelectronicohoy.com/blog/sensor-hc-sr501-con-arduino*

Arduino UNO: Con la plataforma de Arduino UNO tenemos la Central de Alarma. Todo el Sistema estará basado en dicha plataforma. Ya explicaremos su funcionamiento en el código de programación.

Se utilizan los pines A4 y A5 para el Display LCD, pines del 2 a 8 para el Teclado Matricial, pines 9 y 10 para los sensores PIR, pin 11 para el shield relé (señal) y pines 12 y 13 para los leds.

Teclado Matricial 4x3: Este dispositivo nos sirve para colocar el código de acceso en la configuración del Sistema de Alarma, cuenta con filas y columnas, para poder escribir los diferentes caracteres disponibles.

Display LCD 16x2: Con este dispositivo podremos ver la salida de lo que ingresemos con el teclado y así poder saber si el código de acceso es correcto.

Sirena de Alarma: Este dispositivo prácticamente avisa del peligro de ingreso no autorizado. El modelo que usamos es el ESP 30, muy utilizado en alarmas. Trabaja a 12 V DC, 30 watts de potencia. Cuenta con 3 pines. Común (blanco), Tono 1 (rojo) y Tono 2 (amarillo). Cada pin de Tono se conecta a Vcc.

Shield Relé: En este caso usamos un Shield Relé de 2 canales, pero solo necesitamos uno, por eso se puso uno simple en Fritzing, se usa los pines Vcc, GND y señal, común y NC para la sirena de alarma. ¿Por qué? Porque se necesita 12V para operar con la sirena adecuadamente. Sin embargo, Arduino UNO solo suministra o puede soportar hasta 5V en sus pines.

Por eso, el Shield Relé, está basado en el Relé Songle, que trabaja hasta los 30 V DC y corriente de 10 A.

Módulo I2C para Display LCD: Este módulo es bastante útil para reducir el número de pines que deben entrar al Arduino, solo se necesitan 2 pines analógicos (A4 y A5).

Leds para el aviso: sirven para señalizar que la alarma está activa o inactiva.

Todo el Sistema general trabaja con 5V DC, sin embargo, Para el parlante necesitaremos 12V DC, ya que a diferencia de un buzzer pequeño, suena mucho más fuerte y con el Shield Relé se puede potenciar los 5 voltios de Arduino.

3.7 PROGRAMACIÓN DEL TRABAJO CON ARDUINO IDE

```
/*
  Examen de Grado
  Trabajo de Aplicación

  Sistema de Alarma antirrobo
  utilizando la plataforma
  Arduino UNO

  Autor: Efrain Patricio Mendoza Alvarado
*/

#include <Keypad.h> //http://playground.arduino.cc/Code/Keypad
#include <Password.h> //http://playground.arduino.cc/Code/Password
#include <LiquidCrystal_I2C.h> //https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
#include <Wire.h>

//Declaración de las variables de tiempo de espera entre funciones.
const uint16_t delayFunctionT = 1000, intervalT = 400, delayLedT = 50;
unsigned long previousMillisT;

//Ellos comienzan bajo un estado bajo las variables que controlan el sistema.
uint8_t passPositionT = 0, speakerStatusT = LOW, ledStatusT = LOW;
bool alarmStatusT = false, alarmActiveT = false, motionDetectionT = false;
```

```
//Declaración de los pines del teclado, LEDs, sensores PIR y la Sirena.
const uint8_t colsT = 3, rowsT = 4;
const uint8_t colPinsT[colsT] = {8, 7, 6}, rowPinsT[rowsT] = {5, 4, 3, 2}, pirPin1T = 9,
pirPin2T = 10, speakerPinT = 11, ledRedPinT = 12, ledBluePinT = 13;
const char keysT[rowsT][colsT] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

//Instancias del LCD, contraseña y objetos del teclado. La contraseña es "1111", se
puede
//cambiar esta por cualquier combinación de 4 dígitos.
LiquidCrystal_I2C lcd(0x3f, 16, 2);
Password passwordT = Password("1105");
Keypad keypad = Keypad( makeKeymap(keysT), rowPinsT, colPinsT, rowsT, colsT );

void setup() {
  lcd.begin();
  //Configuración de Pines.
  pinMode(speakerPinT, OUTPUT);
  pinMode(ledRedPinT, OUTPUT);
  pinMode(ledBluePinT, OUTPUT);
  pinMode(pirPin1T, INPUT);
  pinMode(pirPin2T, INPUT);
  digitalWrite(ledBluePinT, HIGH);
```

```
  //Empieza el mensaje con 3 segundos de espera.
  lcd.setCursor(0, 0);
  lcd.print("Sistema de Alarma");
  lcd.setCursor(0, 1);
  delay(3000);
  previousMillisT = millis();
  keypad.addEventListener(keypadEventT);
  blockingMessageT();
}

void loop() {
  //Si la variable alarmActiveT está en estado alto y cualquiera de los sensores detecten movimiento,
  //llamará a la función motionDetectedT. La fución motionDetectionT hace
  //sonar la Sirena.
  keypad.getKey();
  if (alarmActiveT == true && digitalRead(pirPin1T) == HIGH)
    motionDetectedT();
  if (motionDetectionT)
    digitalWrite(speakerPinT, speakerAlarmT());
  if (alarmActiveT == true && digitalRead(pirPin2T) == HIGH)
    motionDetectedT();
  if (motionDetectionT)
    digitalWrite(speakerPinT,HIGH);
  else if (motionDetectionT)
    digitalWrite(speakerPinT,LOW);
```

```
}

//keypadEventT registra los eventos del teclado.
void keypadEventT(KeypadEvent keyT) {
  switch (keypad.getState()) {
    //Cuando se teclea "#" o al ingresar más de 4 caracteres, la función será
    //llamada checkPasswordT.
    case PRESSED:
      if (passPositionT >= 5)
        checkPasswordT();
      lcd.setCursor((passPositionT++), 2);
      switch (keyT) {
        case '#':
          checkPasswordT();
          break;
        default:
          passwordT.append(keyT);
          lcd.print("*");         //Para enmascarar lo que se teclee, así el password no se verá
en texto plano.
      }
  }
}

//Si la contraseña es correcta y la alarma está desactivada, entonces se llamará
//a la función blockedSystemT, si la contraseña es correcta y la alarma está activada,
//entoces se llamará a la función unlockSystemT, al contrario de todo esto, se llamará
//a la función incorrectPasswordT, el cual se presenta ante la contraseña incorrecta.
```

```
void checkPasswordT() {
  if (passwordT.evaluate()) {
    if (alarmActiveT == false && alarmStatusT == false)
      blockedSystemT();
    else if (alarmActiveT == true || alarmStatusT == true)
      unlockSystemT();
  }
  else
    incorrectPasswordT();
}

//La función que indica que se ingresó una contraseña incorrecta.
void incorrectPasswordT() {
  passwordT.reset();
  passPositionT = 0;
  digitalWrite(ledBluePinT, LOW);
  //Un mensaje de contraseña incorrecta es enviado.
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Clave incorrecta");
  //Usando el operador ternario se prende o se apaga un led.
  for (uint8_t iT = 1; iT <= 8; iT++) {
    ledStatusT = (ledStatusT) ? LOW : HIGH;
    digitalWrite(ledRedPinT, ledStatusT);
    delay(delayLedT);
  }
  //Las siguientes condiciones servirán para retornar al estado del led.
```

```
  if (alarmActiveT == false && alarmStatusT == false) {
    digitalWrite(ledRedPinT, LOW);
    digitalWrite(ledBluePinT, HIGH);
    delay(delayFunctionT);
    blockingMessageT();  //Si la alarma está desactivada, el led azul prende y el rojo está apagado.
  }
  else if (alarmActiveT == true || alarmStatusT == true) {
    digitalWrite(ledRedPinT, HIGH);
    digitalWrite(ledBluePinT, LOW);
    delay(delayFunctionT);
    unlockMessageT();  ////Si la alarma está activada, el led rojo prende y el azul está apagado.
  }
}

//La función que muestra detección de movimiento.
void motionDetectedT() {
  //Activa la alarma y reinicia la contraseña.
  motionDetectionT = true;
  alarmStatusT = true;
  passwordT.reset();
  passPositionT = 0;
  //Envía un mensaje de que el movimiento ha sido detectado, también se indica por los leds.
  lcd.clear();
  lcd.setCursor(0, 0);
```

```
  lcd.print("Movimiento detectado");
  for (uint8_t iT = 1; iT <= 8; iT++) {
    ledStatusT = (ledStatusT) ? LOW : HIGH;
    digitalWrite(ledRedPinT, ledStatusT);
    delay(delayLedT);
  }
  for (uint8_t iT = 1; iT <= 8; iT++) {
    ledStatusT = (ledStatusT) ? LOW : HIGH;
    digitalWrite(ledBluePinT, ledStatusT);
    delay(delayLedT);
  }
  digitalWrite(ledRedPinT, HIGH);
  digitalWrite(ledBluePinT, LOW);
  delay(delayFunctionT);
  unlockMessageT();
}

//La función que bloquea el Sistema.
void blockedSystemT() {
  passwordT.reset();
  passPositionT = 0;
  digitalWrite(ledRedPinT, HIGH);
  digitalWrite(ledBluePinT, LOW);
  //Se dan 10 segundos de tolerancia antes que el Sistema se active.
  for (uint8_t iT = 10; iT > 0; iT--) {
    lcd.clear();
    lcd.setCursor(0, 0);
```

```
    lcd.print("Quedan ");
    lcd.setCursor(7, 0);
    lcd.print(iT);
    lcd.setCursor(9, 0);
    lcd.print(" segundos");
    lcd.setCursor(0, 1);
    lcd.print("para que se active");
    lcd.setCursor(0, 2);
    lcd.print("el sistema");
    delay(delayFunctionT);
  }

  alarmActiveT = true;
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Sistema Activo");
  delay(delayFunctionT);
  unlockMessageT();
}

//La función que desbloquea el Sistema.
void unlockSystemT() {
  //El estado de alarma cacmbia a estado bajo.
  alarmStatusT = false;
  alarmActiveT = false;
  motionDetectionT = false;
  passwordT.reset();
```

```
  passPositionT = 0;
  digitalWrite(ledRedPinT, LOW);
  digitalWrite(ledBluePinT, HIGH);
  digitalWrite(speakerPinT, LOW);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Sistema desactivado");
  delay(delayFunctionT);
  blockingMessageT();
}

//Las siguientes dos funciones muestran mensajes en la pantalla.
void blockingMessageT() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Ingrese Clave para");
  lcd.setCursor(0, 1);
  lcd.print("activar sistema");
}

void unlockMessageT() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Ingrese Clave");
  lcd.setCursor(0, 1);
  lcd.print("desactivar alarma");
}
```

```
//Esta función retorna a estado alto o bajo, que es asignado a la Sirena.
uint8_t speakerAlarmT() {
  //Utilizando la función millis podemos ejecutar las funciones en "background".
  //background = segundo plano
  unsigned long currentMillisT = millis();
  if ((unsigned long)(currentMillisT - previousMillisT) >= intervalT) {
    previousMillisT = millis();
    return speakerStatusT = (speakerStatusT) ? LOW : HIGH;
  }
}
```

3.8 FUNCIONAMIENTO DEL TRABAJO

3.8.1 Funcionamiento General Del Sistema De Alarma Antirrobo

El Sistema de Alarmas antirrobo funciona de la siguiente manera:

Primero, la Central de Alarma con Arduino UNO si bien puede estar inactivo, no está apagado. Entonces los sensores ya estarán listos para detectar movimiento. El usuario es responsable para la activación o desactivación del Sistema. El sistema debe estar alimentado permanentemente por el Vcc de 5 V y 12 V para la Sirena.

Segundo, el Usuario puede ser cualquiera de las personas que viven el hogar, o el ladrón que ingresa sin autorización o cualquier persona. Entonces, los responsables serían las personas que viven ahí.

La Configuración del Sistema se enfoca principalmente a la introducción de la contraseña, y la activación o desactivación del mismo.

Dicho todo esto, explicamos cómo funciona el Sistema.

Si el Sistema de Alarma está inactivo y no hay alarma, entonces no pasa nada. El Usuario puede activar el Sistema en el momento que sea necesario. Por ejemplo, cuando no hay nadie en el hogar.

Una vez el Usuario active el Sistema de Alarma, tendrá 10 segundos para abandonar el hogar antes que el sistema se active.

Una vez que el Sistema de Alarma esté activo, la Central de Alarma empezará a activar la alarma en caso de que los sensores hayan detectado movimiento.

Una vez se detecte movimiento, la Sirena comenzará a sonar de manera intermitente, cumpliendo con la función de avisar de un posible robo al hogar.

El Sistema de Alarma permanecerá activo y con la sirena sonando (si detectó movimiento), a menos que se introduzca la contraseña correcta para desactivar el mismo.

Si la contraseña es correcta, el Sistema de Alarma se desactiva y la sirena deja de sonar.

Pero, si la contraseña es incorrecta, el Sistema de Alarma sigue activo y la Sirena seguirá sonando de forma intermitente.

3.8.2 Explicación De La Programación De La Central De Alarma

Una vez se alimente al Sistema de Alarma, este estará listo para programarse, el display LCD se enciende y el led azul también, indicando que el Sistema está inactivo.

Todos los mensajes que aparezcan entre comillas (“”), se verán en el display.

Primero Aparecerá “Sistema de Alarma” y después “Ingrese Clave para activar sistema”.

Ingrese en el teclado los números 1105 (clave) y luego # para activar el Sistema.

Si ingresó otros números, presionó más de 4 dígitos o presionó directamente #, entonces le aparecerá el mensaje “Clave incorrecta”.

Si ingresó la clave correcta, le aparecerá el mensaje “Quedan iT segundos”, donde iT es una variable que empieza en 10 y se descuenta hasta 0, entonces abandone el hogar dentro de 10 segundos.

Pasado los 10 segundos, aparece el mensaje “Sistema Activo”, el led rojo se enciende y el Sistema ya empieza a funcionar.

Si se detecta movimiento, aparecerá el mensaje “Movimiento detectado”, y la sirena empezará a sonar.

Para desactivar el Sistema, ingrese 1105 y luego #.

Si ingresó otros números, presionó más de 4 dígitos o presionó directamente #, entonces le aparecerá el mensaje “Clave incorrecta”.

Si ingresó la clave correcta, aparecerá el mensaje “Sistema desactivado” y el led azul se enciende de nuevo.

3.9 COSTOS DEL TRABAJO

He aquí se muestran los costos del Trabajo de Aplicación:

- Arduino UNO 70 Bs.
- 2 sensores PIR HC-SR501 40 Bs. (20 Bs. Cada uno)
- Shield Relé de 5 V, 2 canales 18 Bs.
- Sirena de Alarma ESP 30 de 12 V 85 Bs.
- Teclado matricial 4x3 15 Bs.
- Display LCD 16x2 50 Bs.
- Módulo I2C para Display LCD 18 Bs.
- Protoboard 28 Bs.
- TOTAL 324 Bs.

CAPITULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

Se logró implementar un Sistema de Alarma antirrobo para el hogar utilizando la plataforma Arduino.

Se desarrolló el hardware del Sistema de Alarma antirrobo, utilizando como dispositivo central al Arduino UNO.

Se desarrolló el software (entorno de programación) utilizando el entorno IDE de Arduino para cumplir con la función principal del Sistema de Alarma.

Se configuró un teclado matricial para que pida contraseña de acceso a las personas autorizadas para activar o desactivar el Sistema de Alarma.

Se configuró un display LCD para que se pueda visualizar los estados del Sistema de Alarma y datos que ingrese la persona para activar o desactivar el mismo.

Se observó que el Sistema de Alarma funciona de manera correcta, haciéndolo efectivamente un Sistema económico, fácil de armar y configurar, sin tener nada que envidiar a Sistemas más robustos.

4.2 RECOMENDACIONES

El Sistema de Alarmas de este trabajo se puede mejorar aumentando más sensores en las entradas analógicas de Arduino UNO restantes, como ser Detector Magnético para puertas o Detector de Ruptura de Vidrio en las ventanas.

Se podría mejorar la configuración de la Central de Alarma con, por ejemplo, colocando líneas de código para que el Usuario tenga hasta 3 oportunidades para ingresar la contraseña correcta, pero al mismo tiempo solo dar 10 segundos para que escriba, pasado ese tiempo, la alarma haga la cuenta regresiva para activarse.

Ver la forma de colocar una batería de respaldo al Arduino, que lo haría más completo el Sistema de Alarma, en caso de corte de energía, o ver la opción de colocarle a la fuente un UPS.

BIBLIOGRAFÍA

Academy Xperts Bolivia. (2018). Experto en Seguridad Electrónica. Módulo 2. La Paz, Bolivia. Academy Xperts.

Arduino Team. (16 de noviembre de 2021). Arduino UNO datasheet. Recuperado de: https://docs.arduino.cc/static/1574b4e36f0be25c48c299221e2cad95/A000066-datasheet.pdf.

Beiroa Mosquera Rubén. (2019). Aprender Arduino, electrónica y programación con 100 ejercicios práctico. Ciudad de México, México. Alfaomega.

Blanco Quiroga Héctor. (14 de abril de 2021). Sirena de alarma: qué es y para qué sirve. Recuperado de: https://queadslcontratar.com/alarmas-casa/componentes/sirena.

Edgardosilvi. (29 de febrero de 2016). Acamica: Ventajas y desventajas de Arduino. Recuperado de: https://edgardosilvi.wordpress.com/2016/02/29/acamica-ventajas-y-desventajas-de-arduino/

García Vicente. (7 de noviembre de 2017). Sensor HC-SR501 Con Arduino. Recuperado de: https://www.diarioelectronicohoy.com/blog/sensor-hc-sr501-con-arduino.

Mamani Sanga Eddy Robert. (noviembre 2018). Diseño de un sistema electrónico de alarma antirrobo para la protección del hogar. Recuperado de: http://repositorio.umsa.bo/xmlui/handle/123456789/21656.

Monk Simon. (2012). 30 proyectos con Arduino. Madrid, España. Estribor.

Proyecto TEOS. (11 de junio de 2019). Sistemas de Seguridad con Arduino. Recuperado de: https://github.com/proyectoTEOS/Sistema-de-seguridad-con-Arduino.

Pulgarin Juan Carlos. (s.f.). Funcionamiento de un Sistema de Alarma. Recuperado de: https://www.monografias.com/trabajos-pdf5/sistema-alarmas/sistema-alarmas.shtml.

TECNOSeguro (s.f.). ¿Qué es un Sistema de Alarma? Recuperado de: https://www.tecnoseguro.com/faqs/alarma/que-es-un-sistema-de-alarma.

ANEXOS

ANEXO 1 – CARACTERÍSTICAS DE LA SIRENA ESP 30

dLux

ESP-30

Descripción

Sirena de exterior para sistema de alarma.

Especificaciones técnicas

Alimentación 12 Vdc
Potencia de salida 30W
Nivel de presión sonora 112dB
Dos tonos audibles
Incluye soporte para montaje

ANEXO 2 – LIBRERÍA DEL TECLADO

```
/*
||
|| @file Keypad.h
|| @version 3.1
|| @author Mark Stanley, Alexander Brevig
|| @contact mstanley@technologist.com, alexanderbrevig@gmail.com
||
|| @description
|| | This library provides a simple interface for using matrix
|| | keypads. It supports multiple keypresses while maintaining
|| | backwards compatibility with the old single key library.
|| | It also supports user selectable pins and definable keymaps.
|| #
||
|| @license
|| | This library is free software; you can redistribute it and/or
|| | modify it under the terms of the GNU Lesser General Public
|| | License as published by the Free Software Foundation; version
|| | 2.1 of the License.
|| |
|| | This library is distributed in the hope that it will be useful,
|| | but WITHOUT ANY WARRANTY; without even the implied warranty of
|| | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
|| | Lesser General Public License for more details.
|| |
|| | You should have received a copy of the GNU Lesser General Public
```

```
|| | License along with this library; if not, write to the Free Software
|| | Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
|| #
||
*/

#ifndef KEYPAD_H
#define KEYPAD_H

#include "utility/Key.h"

// Arduino versioning.
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

// bperrybap - Thanks for a well reasoned argument and the following macro(s).
// See http://arduino.cc/forum/index.php/topic,142041.msg1069480.html#msg1069480
#ifndef INPUT_PULLUP
#warning "Using  pinMode() INPUT_PULLUP AVR emulation"
#define INPUT_PULLUP 0x2
#define pinMode(_pin, _mode) _mypinMode(_pin, _mode)
#define _mypinMode(_pin, _mode)  \
do {                                                    \
        if(_mode == INPUT_PULLUP)       \
                pinMode(_pin, INPUT);       \
```

```
                digitalWrite(_pin, 1);  \
        if(_mode != INPUT_PULLUP)       \
                pinMode(_pin, _mode);       \
}while(0)
#endif


#define OPEN LOW
#define CLOSED HIGH

typedef char KeypadEvent;
typedef unsigned int uint;
typedef unsigned long ulong;

// Made changes according to this post http://arduino.cc/forum/index.php?topic=58337.0
// by Nick Gammon. Thanks for the input Nick. It actually saved 78 bytes for me. :)
typedef struct {
   byte rows;
   byte columns;
} KeypadSize;

#define LIST_MAX 10                 // Max number of keys on the active list.
#define MAPSIZE 10          // MAPSIZE is the number of rows (times 16 columns)
#define makeKeymap(x) ((char*)x)


//class Keypad : public Key, public HAL_obj {
class Keypad : public Key {
```

```
public:

	Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte
numCols);


	virtual void pin_mode(byte pinNum, byte mode) { pinMode(pinNum, mode); }
	virtual void pin_write(byte pinNum, boolean level) { digitalWrite(pinNum,
level); }
	virtual int  pin_read(byte pinNum) { return digitalRead(pinNum); }


	uint bitMap[MAPSIZE];	// 10 row x 16 column array of bits. Except Due
which has 32 columns.
	Key key[LIST_MAX];
	unsigned long holdTimer;


	char getKey();
	bool getKeys();
	KeyState getState();
	void begin(char *userKeymap);
	bool isPressed(char keyChar);
	void setDebounceTime(uint);
	void setHoldTime(uint);
	void addEventListener(void (*listener)(char));
	int findInList(char keyChar);
	int findInList(int keyCode);
	char waitForKey();
	bool keyStateChanged();
	byte numKeys();
```

```
private:
        unsigned long startTime;
        char *keymap;
   byte *rowPins;
   byte *columnPins;
        KeypadSize sizeKpd;
        uint debounceTime;
        uint holdTime;
        bool single_key;

        void scanKeys();
        bool updateList();
        void nextKeyState(byte n, boolean button);
        void transitionTo(byte n, KeyState nextState);
        void (*keypadEventListener)(char);
};

#endif

/*
|| @changelog
|| | 3.1 2013-01-15 - Mark Stanley     : Fixed missing RELEASED & IDLE status when using a single key.
|| | 3.0 2012-07-12 - Mark Stanley     : Made library multi-keypress by default. (Backwards compatible)
|| | 3.0 2012-07-12 - Mark Stanley     : Modified pin functions to support Keypad_I2C
|| | 3.0 2012-07-12 - Stanley & Young  : Removed static variables. Fix for multiple keypad objects.
```

|| | 3.0 2012-07-12 - Mark Stanley : Fixed bug that caused shorted pins when pressing multiple keys.

|| | 2.0 2011-12-29 - Mark Stanley : Added waitForKey().

|| | 2.0 2011-12-23 - Mark Stanley : Added the public function keyStateChanged().

|| | 2.0 2011-12-23 - Mark Stanley : Added the private function scanKeys().

|| | 2.0 2011-12-23 - Mark Stanley : Moved the Finite State Machine into the function getKeyState().

|| | 2.0 2011-12-23 - Mark Stanley : Removed the member variable lastUdate. Not needed after rewrite.

|| | 1.8 2011-11-21 - Mark Stanley : Added test to determine which header file to compile,

|| | WProgram.h or Arduino.h.

|| | 1.8 2009-07-08 - Alexander Brevig : No longer uses arrays

|| | 1.7 2009-06-18 - Alexander Brevig : This library is a Finite State Machine every time a state changes

|| | the keypadEventListener will trigger, if set

|| | 1.7 2009-06-18 - Alexander Brevig : Added setDebounceTime setHoldTime specifies the amount of

|| | microseconds before a HOLD state triggers

|| | 1.7 2009-06-18 - Alexander Brevig : Added transitionTo

|| | 1.6 2009-06-15 - Alexander Brevig : Added getState() and state variable

|| | 1.5 2009-05-19 - Alexander Brevig : Added setHoldTime()

|| | 1.4 2009-05-15 - Alexander Brevig : Added addEventListener

|| | 1.3 2009-05-12 - Alexander Brevig : Added lastUdate, in order to do simple debouncing

|| | 1.2 2009-05-09 - Alexander Brevig : Changed getKey()

|| | 1.1 2009-04-28 - Alexander Brevig : Modified API, and made variables private

|| | 1.0 2007-XX-XX - Mark Stanley : Initial Release

|| #

*/

ANEXO 3 – LIBRERÍA DE LA CONTRESEÑA

```
/*
||
|| @file Password.h
|| @version 1.2
|| @author Alexander Brevig
|| @contact alexanderbrevig@gmail.com
||
||  4/5/2012 Updates Nathan Sobieck: Nathan@Sobisource.com
||   Now v1.2 Arduino IDE v1.0 With BAckwards compatibility
||
||
||
|| @description
|| | Handle passwords easily
|| #
||
|| @license
|| | This library is free software; you can redistribute it and/or
|| | modify it under the terms of the GNU Lesser General Public
|| | License as published by the Free Software Foundation; version
|| | 2.1 of the License.
|| |
|| | This library is distributed in the hope that it will be useful,
|| | but WITHOUT ANY WARRANTY; without even the implied warranty of
|| | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
|| | Lesser General Public License for more details.
```

```
|| |
|| | You should have received a copy of the GNU Lesser General Public
|| | License along with this library; if not, write to the Free Software
|| | Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
|| #
||
*/

#ifndef PASSWORD_H
#define PASSWORD_H

// Arduino versioning.
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h" // for digitalRead, digitalWrite, etc
#else
#include "WProgram.h"
#endif

#define MAX_PASSWORD_LENGTH 20

#define STRING_TERMINATOR '\0'

class Password {
public:
        Password(char* pass);

        void set(char* pass);
        bool is(char* pass);
```

```
        bool append(char character);
        void reset();
        bool evaluate();

        //char* getPassword();
        //char* getGuess();

        //operators
        Password &operator=(char* pass);
        bool operator==(char* pass);
        bool operator!=(char* pass);
        Password &operator<<(char character);

private:
        char* target;
        char guess[ MAX_PASSWORD_LENGTH ];
        byte currentIndex;
};

#endif

/*
|| @changelog
|| | 1.1 2009-06-17 - Alexander Brevig : Added assignment operator =, equality operators == != and insertion operator <<
|| | 1.0 2009-06-17 - Alexander Brevig : Initial Release
|| #
*/
```

ANEXO 4 – LIBRERÍA DE DISPLAY LCD CON MÓDULO I2C

```
#ifndef FDB_LIQUID_CRYSTAL_I2C_H
#define FDB_LIQUID_CRYSTAL_I2C_H

#include <inttypes.h>
#include <Print.h>

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
```

```
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

// flags for backlight control
#define LCD_BACKLIGHT 0x08
#define LCD_NOBACKLIGHT 0x00

#define En B00000100  // Enable bit
#define Rw B00000010  // Read/Write bit
#define Rs B00000001  // Register select bit

/**
 * This is the driver for the Liquid Crystal LCD displays that use the I2C bus.
 *
```

```
 * After creating an instance of this class, first call begin() before anything else.
 * The backlight is on by default, since that is the most likely operating mode in
 * most cases.
 */
class LiquidCrystal_I2C : public Print {
public:
        /**
         * Constructor
         *
         * @param lcd_addr   I2C slave address of the LCD display. Most likely printed on the
         *                   LCD circuit board, or look in the supplied LCD documentation.
         * @param lcd_cols   Number of columns your LCD display has.
         * @param lcd_rows   Number of rows your LCD display has.
         * @param charsize   The size in dots that the display has, use LCD_5x10DOTS or LCD_5x8DOTS.
         */
        LiquidCrystal_I2C(uint8_t lcd_addr, uint8_t lcd_cols, uint8_t lcd_rows, uint8_t charsize = LCD_5x8DOTS);


        /**
         * Set the LCD display in the correct begin state, must be called before anything else is done.
         */
        void begin();


        /**
         * Remove all the characters currently shown. Next print/write operation will start
```

```
     * from the first position on LCD display.
     */
    void clear();

    /**
     * Next print/write operation will will start from the first position on the LCD display.
     */
    void home();

     /**
      * Do not show any characters on the LCD display. Backlight state will remain unchanged.
      * Also all characters written on the display will return, when the display in enabled again.
      */
    void noDisplay();

    /**
     * Show the characters on the LCD display, this is the normal behaviour. This method should
     * only be used after noDisplay() has been used.
     */
    void display();

    /**
     * Do not blink the cursor indicator.
     */
    void noBlink();
```

```
/**
 * Start blinking the cursor indicator.
 */
void blink();

/**
 * Do not show a cursor indicator.
 */
void noCursor();

/**
 * Show a cursor indicator, cursor can blink on not blink. Use the
 * methods blink() and noBlink() for changing cursor blink.
 */
void cursor();

void scrollDisplayLeft();
void scrollDisplayRight();
void printLeft();
void printRight();
void leftToRight();
void rightToLeft();
void shiftIncrement();
void shiftDecrement();
void noBacklight();
void backlight();
bool getBacklight();
```

```
        void autoscroll();
        void noAutoscroll();
        void createChar(uint8_t, uint8_t[]);
        void setCursor(uint8_t, uint8_t);
        virtual size_t write(uint8_t);
        void command(uint8_t);

        inline void blink_on() { blink(); }
        inline void blink_off() { noBlink(); }
        inline void cursor_on() { cursor(); }
        inline void cursor_off() { noCursor(); }

// Compatibility API function aliases
        void setBacklight(uint8_t new_val);                 // alias for backlight()
and nobacklight()
        void load_custom_character(uint8_t char_num, uint8_t *rows);     // alias for
createChar()
        void printstr(const char[]);

private:
        void send(uint8_t, uint8_t);
        void write4bits(uint8_t);
        void expanderWrite(uint8_t);
        void pulseEnable(uint8_t);
        uint8_t _addr;
        uint8_t _displayfunction;
        uint8_t _displaycontrol;
        uint8_t _displaymode;
```

```
        uint8_t _cols;
        uint8_t _rows;
        uint8_t _charsize;
        uint8_t _backlightval;
};

#endif // FDB_LIQUID_CRYSTAL_I2C_H
```