Práctica final

Alfonso José Mateos Hoyos (DNI:)
Manuel Salgado de la Iglesia (DNI:)
Francisco Pinto Santos (DNI:)

Puesto de trabajo número 9

Índice

I. Innovaciones

El ejercicio final nos pedía realizar una alarma con un sensor de distancia, de forma que cuando el sensor detectase algo a cierta distancia, avisase de esto a un usuario mandando un SMS.
No obstante, nosotros hemos añadido una serie de características adicionales:

- Uso de varios elementos para mostrar si se ha activado la alarma: hemos incorporado para ello un buzzer y un LED que se activaran cuando el sistema se establezca en periodo de alerta.
- Uso de un lector RFID: Con este podremos desactivar la alarma y hacerla pasar de activa a inactiva y viceversa.
- Uso de un LCD: con él mostramos en todo momento el estado de la alarma de forma que el usuario pueda leerlo.

II. Circuito

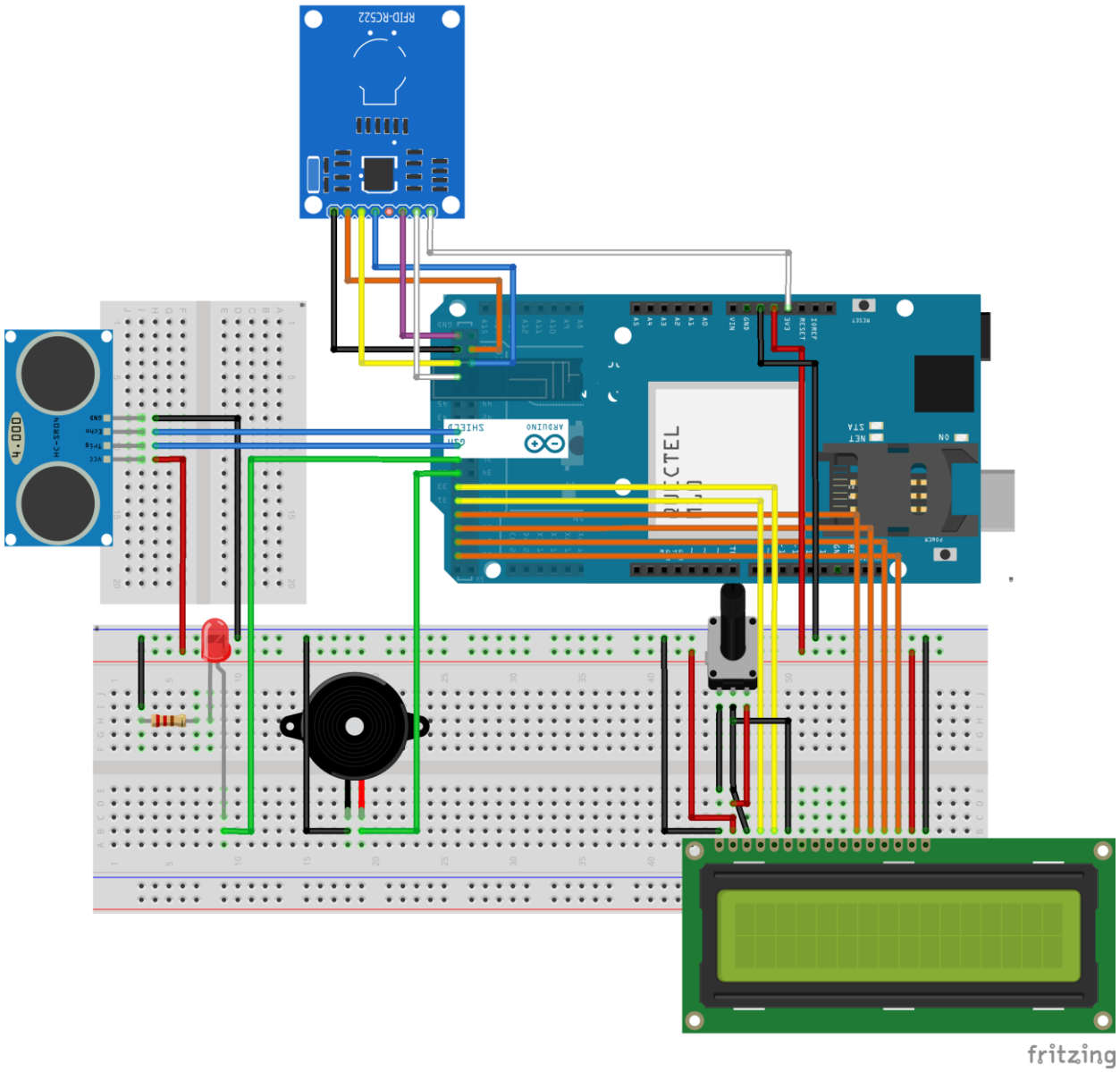

Figura 2.0.0:Diagrama de circuito completo

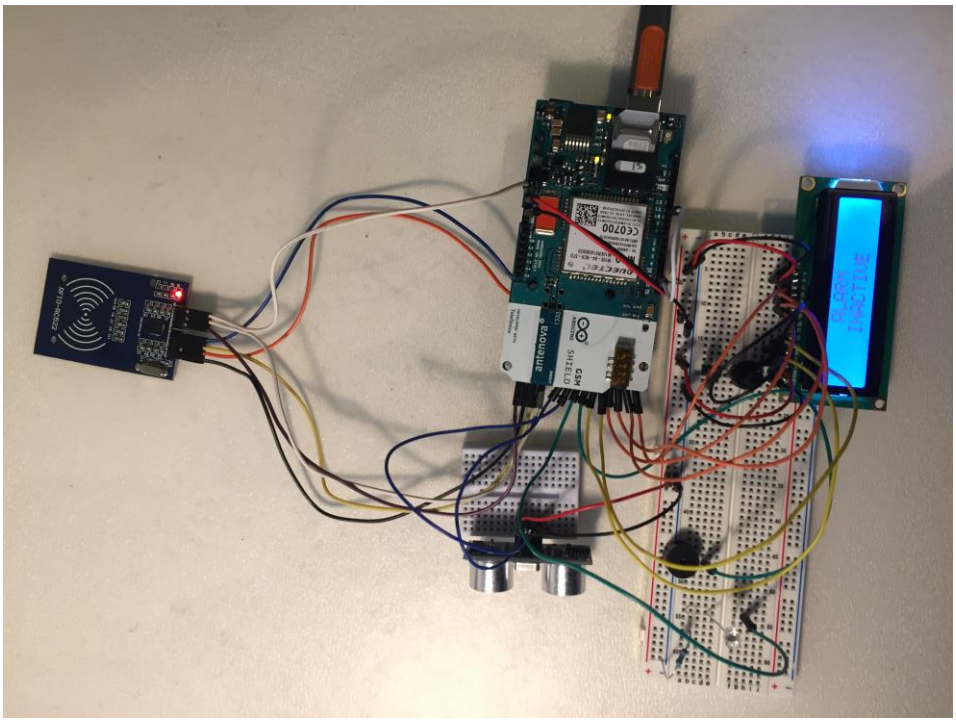

Figura 2.0.1: Montaje completo

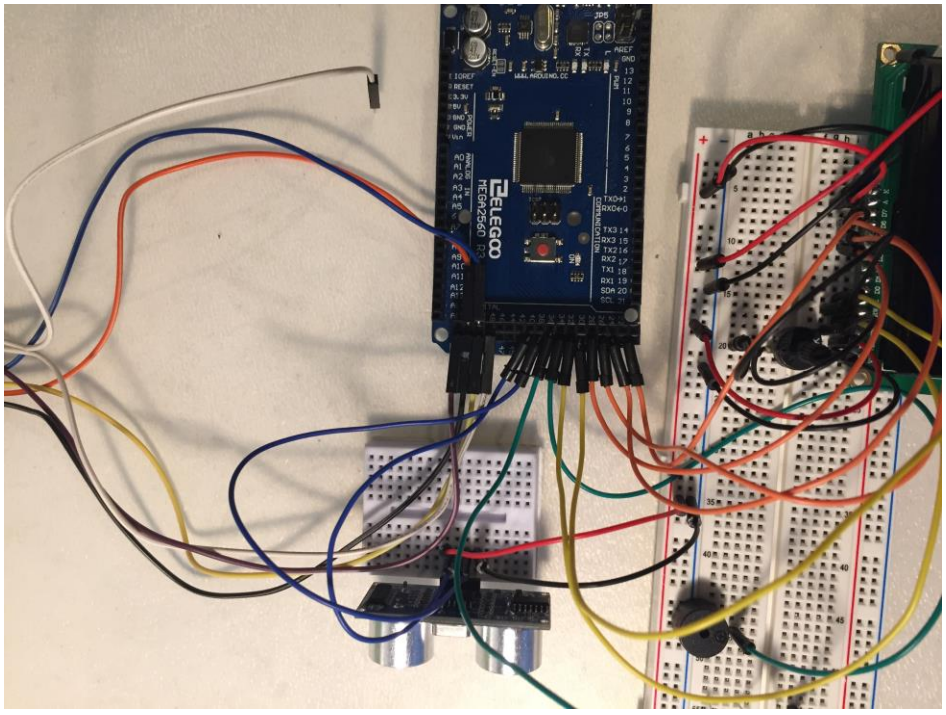

Figura 2.0.2: Montaje sin GSM para apreciar conexiones mejor

a) Sensor de distancia

El sensor que hemos usado para medir distancias es el HC-SR04, el cual ya conocíamos de prácticas anteriores realizadas en el laboratorio. El uso de este es quizás el más relevante, pues nos ayudará a detectar movimiento para activar nuestra alarma.

Su funcionamiento es el siguiente:

- Envía un pulso de alta frecuencia no audible por el ser humano.
- Este pulso rebota en objetos cercanos y es reflejado hacia el sensor, el cual dispone de un micrófono para esta frecuencia.
- Se mide el tiempo entre pulsos, conociendo la velocidad del sonido, y estimamos la distancia del objeto contra el que ha rebotado.

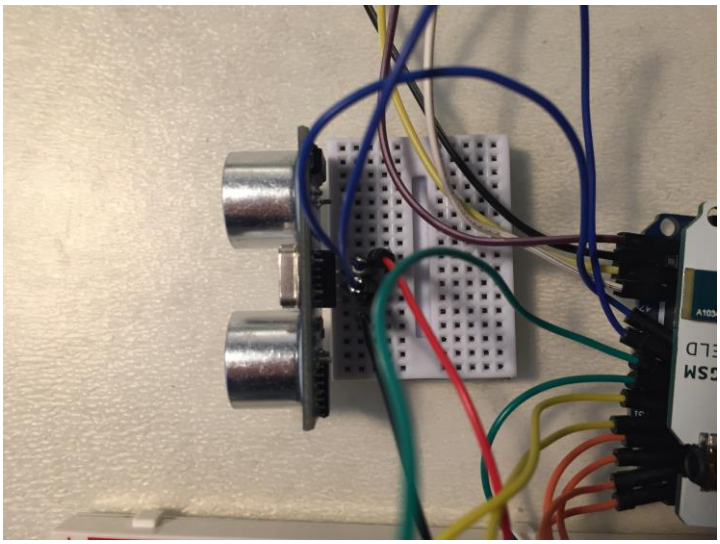

Figura 2.b.1: Sensor HC-SR04

b) Lector RFID

El lector RFID, es un dispositivo que emite una señal de radio, la cual es recibida por los RFID Tag cercanos, los cuales la procesan y devuelven una información u otra en función de dicha señal (Figura 2.c.2).

En concreto, hemos usado el módulo MRFC522 (Figura 2.c.1). Utiliza 3.3V como voltaje de alimentación y las comunicaciones con este se realizan a través del SPI*. Utiliza un sistema de modulación/demodulación para todo tipo de

dispositivos pasivos de 13.56Mhz. Lo usaremos para activar/desactivar la alarma con su correspondiente tarjeta.

Por otro lado, la tarjeta, es un RFID TAG cuenta con 64 bloques de memoria (0-63), donde se hace lectura y/o escritura. Cada uno de estos bloques tiene una capacidad máxima de 16 Bytes. El número serie de la tarjeta consiste en cinco valores hexadecimales, los cuales usamos como código de desbloqueo.

*SPI: El Serial Peripherical Interface, es un estándar de bus ideado en los años 80, el cual incorporan gran cantidad de dispositivos hoy en día. Este lo incorpora Arduino, y consiste en una interfaz de comunicación para múltiples periféricos con una arquitectura maestro-esclavo.
Esto nos permite especificar un pin de selección (SS) y uno de reset (RR) del dispositivo, permitiendo que el resto de la comunicación se realice a partir del resto de pines, los cuales pueden compartir varios dispositivos.

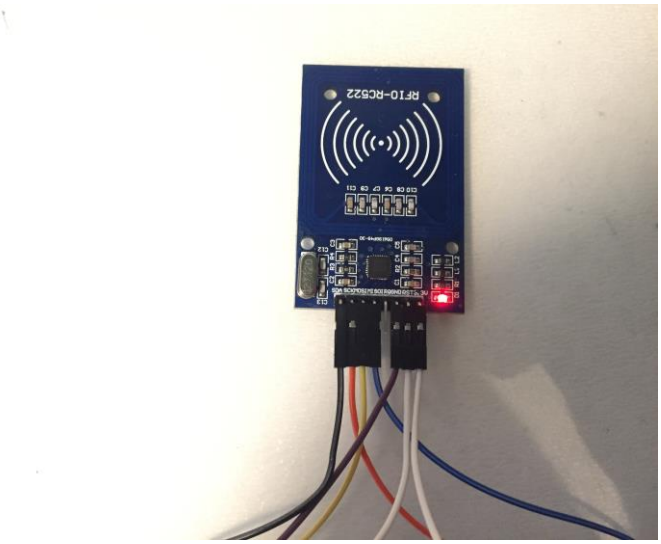

Figura 2.c.1: Módulo MRFC522

Figura 2.c.2: Tag RFID

c) GSM

GSM es un estándar internacional para teléfonos móviles, es el acrónimo de **Global System for Mobile Communications**.

En nuestro caso, hemos utilizado un GSM Shield que permite a una placa Arduino conectarse a internet, enviar y recibir SMS y hacer llamadas de voz, usando la biblioteca de GSM.
En nuestro caso concretamente, si la alarma no se desactiva en un minuto, se enviará un SMS al teléfono indicado.

Cabe destacar que al igual que el modulo MRFC522, este se comunica mediante el SPI. Debido a esto, nos surgió un problema de compatibilidad del GSM Shield con la placa, por lo que ha sido necesario no conectar el pin numero 12, y puentear el pin 10 y el 12, porque este Shield ha sido diseñado para el Arduino Uno y, al cambiar el modelo de placa, es necesario hacer estos pequeños ajustes, si no la comunicación no se establece de forma apropiada.

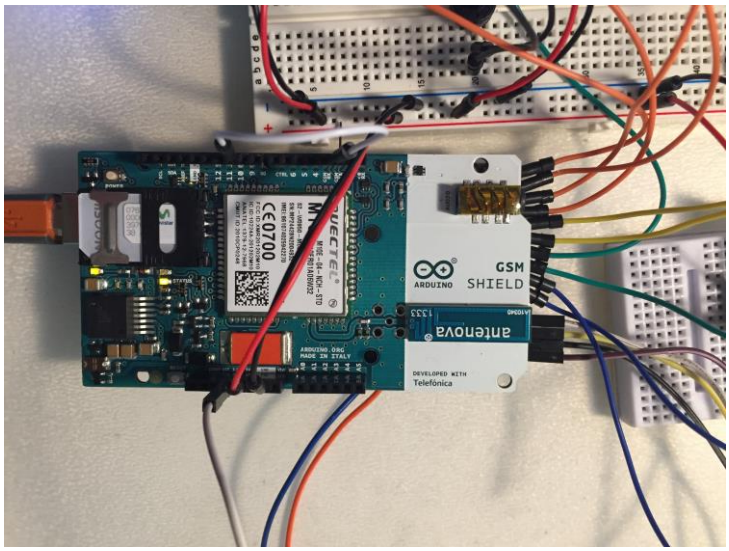

Figura 2.d.1: GSM

d) LCD Display

LCD es el acrónimo de **LiquidCrystal Display**, utiliza las propiedades de la luz polarizada para mostrarnos la información en una pantalla.
Esto lo consigue haciendo que, en cada pixel, haya dos filtros polarizadores, uno rotado 90 grados respecto al otro, con cristal líquido en medio. Al aplicarle corriente a este material, hace que las propiedades de este cambien, afectando a cómo pasa la luz a través de este, haciendo que el píxel se torne oscuro o claro.
Este mecanismo necesita una fuente de luz externa para funcionar, debido a lo cual, hay un luz a la parte derecha de este.

Cabe destacar que, en este LCD, el circuito respectivo al LED y el LCD va por separado, debido a lo cual hay que establecer muchas conexiones. Además, es necesario usar un potenciómetro para regular el contraste del LCD.

Este componente se encarga de convertir las señales eléctricas de la placa en información visual entendible por los seres humanos. Gracias a esto, podremos ir mostrando el estado actualizado de la alarma, además de otros mensajes relevantes.

Figura2.e.1: LCD

e) LED y Buzzer

Por último, estos dos componentes serán otra forma de comunicar el estado de la alarma. Cuando esta esté activa y detecte movimiento, el LED se encenderá y el Buzzer comenzará a sonar hasta que logremos desactivarla.

Ambos son componentes que ya conocemos de las anteriores prácticas. Por un lado, el LED, es un diodo emisor de luz. Por otro lado, el buzzer o zumbador, es un dispositivo que genera un sonido de una frecuencia determinada y fija cuando es conectado a tensión.

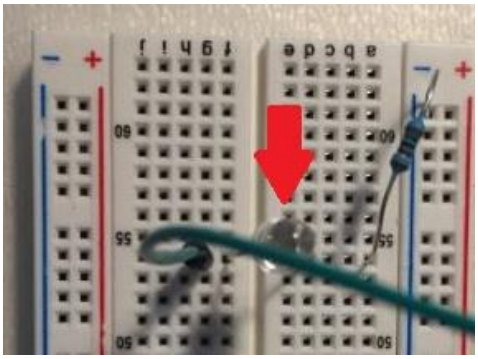

Figura2.f.1: LED

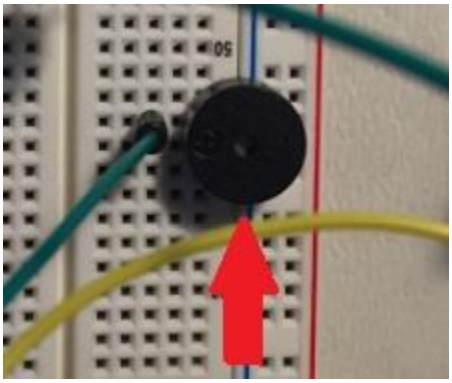

Figura2.f.2: Buzzer

III. Código

a) Concepto y estructura

Para modelar este sistema, propusimos 3 estados, los cuales se pueden apreciar en la Figura 3.a.1:

- Desactivado: El sistema no realiza la detección de intrusos, solo escanea el lector RFID para que cuando se introduzca la tarjeta, pase a modo activado.
- Activado: El sistema detecta intrusos y escanea el lector RFID para que cuando se introduzca la tarjeta se pase a modo desactivado.
  En el caso de detectar un intruso se pasa a modo alerta.
- Alerta: Se enciende el LED, se emite sonido por el buzzer y en el LCD se muestra el tiempo restante para introducir la tarjeta.
  Si no se introduce la tarjeta se envía un SMS, se paran el buzzer y el LED y se finaliza.

Cada estado está representado mediante una función, la cual se llama desde la función loop, como se puede ver en la Figura 3.a.2.
El estado se guarda en la variable state, que es una variable tipo AlarmState, el cual es un enum con 3 valores cuyos nombres coinciden con los 3 estados posibles de la alarma.
Como se puede apreciar, en cada iteración de la función loop, aparte de llamar a la función correspondiente al estado, se esperan DELAY segundos.

Respecto a la inicialización, esta se realiza en la función setup, como podemos ver en la Figura 3.a.3, donde inicializamos el LCD, SPI (Serial Pheriferical Interface, utilizado para comunicar el Arduino con el MFRC522 y el GSM), MFRC522, GSM, establecemos los pines del buzzer y LED como salida y, por último, establecemos el estado inicial como inactivo e imprimimos el estado.
Cabe destacar que esta función puede durar bastante tiempo, porque la función de inicialización del GSM es bastante lenta.

En cuanto a las 3 funciones correspondientes a cada estado:

- alarmInactiveBehaviour: cómo podemos ver en la Figura 3.a.4, en el estado inactivo se mira el lector de tarjetas.
  En el caso de que se haya introducido la tarjeta correcta, se pone el estado a activo y se imprime este, tras lo cual en la siguiente iteración de loop se ejecutara la función alarmActiveBehaviour.

- alarmActiveBehaviour: en esta función se mira el sensor de distancia, y en el caso de que haya variado lo suficiente (esto se explicara más a fondo adelante), se establece e imprime por el LCD el estado de alerta, además de guardar el tiempo actual, lo cual se utilizara para calcular el tiempo restante para enviar el SMS. En este caso también se enciende el LED y se activa el buzzer.
  También se escanea el lector de tarjetas, pues si se acerca la tarjeta correcta, se establece el estado inactivo, para ello se notifica por el LCD que la alarma se ha desactivado, tras lo cual se reinicia el Arduino, lo que hará que se vuelva al estado inactivo.

- alarmAlertBehaviour: en esta función lo primero que se hace es calcular el tiempo restante. En el caso de que sea menor que cero se apagan el LED y buzzer y se envía el SMS. Tras esto se reinicia el Arduino, lo que hará que se pase a estado inactivo.
  Después de esto, se mira la tarjeta para ver si se ha conseguido desactivar la alarma, en cuyo caso, se paran el LED y buzzer, se notifica y, al igual que en el caso anterior, se reinicia el Arduino, lo que nos llevará a modo inactivo.

Para reiniciar el Arduino, se ha utilizado la función RESET_ARDUINO(), la cual es un puntero a 0, como se ven en la Figura 3.a.7, la cual lo que hace es que se ejecute desde un principio la función setup y luego la función loop.
También cabe destacar que el reiniciar el Arduino se hace porque como estamos utilizando un contador de 60 segundos para permitir poner la tarjeta y desactivar la alarma, daba problemas con la medición de tiempo.

Por último, en este punto, queríamos destacar que hemos utilizado un fichero de cabecera llamado constants.h, en el cual hemos definido todos los pines, tiempos, parámetros, … que necesitamos, para facilitar el cambio de cualquiera de estos.

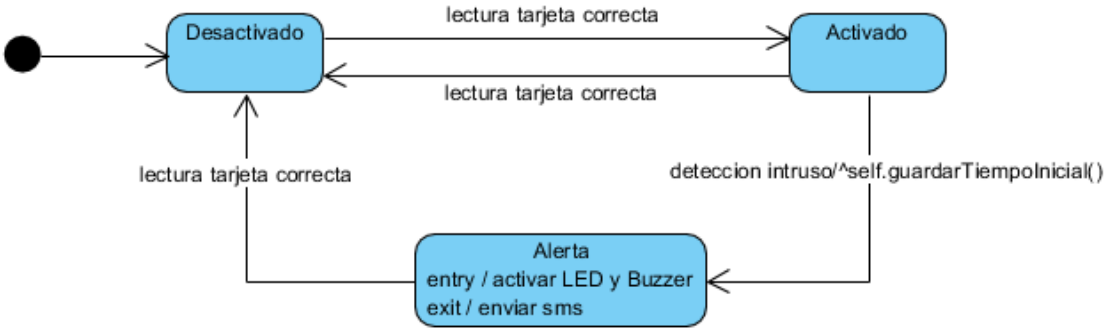

Figura 3.a.1: Diagrama de estados

```
void loop() {
  switch(state){
    case ACTIVE:    alarmActiveBehaviour();   break;
    case INACTIVE:  alarmInactiveBehaviour(); break;
    case ALERT:     alarmAlertBehaviour();    break;
  }

  delay(DELAY);
}
```

Figura 3.a.2: función loop

```
void setup() {
  //start serial port
  Serial.begin(9600);

  //start lcd and print first message
  lcd.begin(LCD_NCOL, LCD_NROW);
  lcd.print("  Initializing");

  //start card reader
  SPI.begin();
  mfrc522.PCD_Init();

  //start gsm
  gsmAccess.begin("");

  //define buffer and led pin
  pinMode(LED_PIN,OUTPUT);
  pinMode(BUZZER_PIN,OUTPUT);

  //set the satate of the alarm inactive
  state = INACTIVE;
  printState();
}
```

Figura 3.a.3: Función setup

```
void alarmInactiveBehaviour(){
  if (checkCardReader()){
    state = ACTIVE;
    printState();
  }
}
```

Figura 3.a.4: función de alarma inactiva

```
void alarmActiveBehaviour(){
    //check if its necessary to enter in alert mode
    if( checkDistance() ){
        //set the alert mode
        state = ALERT;
        printState();
        //start the sound and led
        tone(BUZZER_PIN,BUZZER_TONE);
        digitalWrite(LED_PIN,HIGH);
        //save current time
        alertStartTime = now();
    }

    //chek if alarm deactivation is required
    if ( checkCardReader()){
     lcd.clear();
     lcd.print("     ALARM");
     lcd.setCursor(0,1);
     lcd.print("   DEACTIVATED");
     delay(NOTIFY_DELAY);
     //reset arduino
     RESET_ARDUINO();
    }
}
```

Figura 3.a.5: función de alarma activa

```
void alarmAlertBehaviour(){
  time_t currentTime = now() - alertStartTime;
  long remainingTime= MAX_ALERT_TIME - currentTime;

  //check if the state
  if( remainingTime < 0){
     //stop sound and led
     noTone(BUZZER_PIN);
     digitalWrite(LED_PIN,LOW);
     //send a SMS
     sendSMS();
     //notify sms sent
     lcd.clear();
     lcd.print("TIME LIMIT REACHED");
     lcd.setCursor(0,1);
     lcd.print("     SMS SENT     ");
     delay(NOTIFY_DELAY);
     //reset arduino
     RESET_ARDUINO();
  }

  //check if deactivation is required
  if( checkCardReader()){
     //stop sound and led
     noTone(BUZZER_PIN);
     digitalWrite(LED_PIN,LOW);
     //print
     lcd.clear();
     lcd.print("     ALARM");
     lcd.setCursor(0,1);
     lcd.print("   DEACTIVATED");
     delay(NOTIFY_DELAY);
     //reset arduino
     RESET_ARDUINO();
  }

  printState();
}
```

Figura 3.a.6: función de alarma alerta

```
void(* RESET_ARDUINO) (void) = 0;  // declare reset fuction at address 0
```

Figura 3.a.7: función reset

b) Sensor de distancia

Para manejar el sensor de distancia, al igual que hemos hecho en otras prácticas, hemos utilizado un objeto manejador (Figura 3.b.1) proporcionado por la biblioteca HCSR04.
Este nos permite medir la distancia en cm con una simple función encapsulando la medición y el cálculo de transformación.

En cuanto a la función en sí, devuelve verdadero en el caso de que se detecte una perturbación y falso en caso contrario.
Su funcionamiento es muy básico pues en una variable estática guarda el resultado de la medición anterior y si este presenta una diferencia en cm mayor que una MAX_PERTURBACION, devuelve verdadero porque esto se considera perturbación.

El que se haya definido MAX_PERTURBATION como distancia mínima para considerarlo una perturbación, es porque al no tener el sensor mucha precisión, entre medida y medida presentaba una pequeña desviación, que si no se tenía en cuenta a la hora de programar el funcionamiento, haría que saltara la alarma nada más activarla.

```
boolean checkDistance(){
  static double lastDistance = -2;
  double distance = distanceSensor.measureDistanceCm();
  boolean result = ( abs(distance - lastDistance) > MAXPERTURBATION );
  lastDistance = distance;
  return result;
}
```

Figura 3.b.1:Función de medición de distancia

```
UltraSonicDistanceSensor distanceSensor(TRIGGERPIN, ECHOPIN);
```

Figura 3.b.2: Objeto controlador del sensor de distancia HCSR04

c) Lector RFID

Para realizar las lecturas de este sensor, hemos utilizado la bibliotecas MRFC522 y SPI, que nos provee el objeto manejador con el que podemos trabajar.

A pesar de que el sensor tiene varias conexiones, al manejador solo hace falta darle 2, que son cuál es el pin de selección y cuál es el pin de reset del sensor, puesto que el resto de las conexiones se realizan con los respectivos pines

correspondientes al Serial Peripherical Interface, y la biblioteca sabe internamente cuales son dependiendo del modelo de tarjeta Arduino.
En cuanto a la manera de trabajar con el lector, hemos encapsulado su funcionamiento en checkCardReader, una función que comprueba si hay una nueva tarjeta, en cuyo caso la lee, y compara el UID con el UID de desbloqueo. En caso de que sea correcto devuelve verdadero, y en cualquier otro caso devuelve falso.

```
MFRC522 mfrc522(SS_PIN, RST_PIN);
```

Figura 3.c.1: manejador de lector RFID

```
boolean checkCardReader(){
  int i;
  boolean isCodeOK = false;
  if ( mfrc522.PICC_IsNewCardPresent() ){
    if( mfrc522.PICC_ReadCardSerial()){
      isCodeOK = true;
      isCodeOK = isCodeOK && (mfrc522.uid.size == CODE_SIZE);
      for(i=0; i<CODE_SIZE && i<mfrc522.uid.size && isCodeOK; i++)
        isCodeOK = isCodeOK && (cardCode[i] == mfrc522.uid.uidByte[i]);
      mfrc522.PICC_HaltA();
    }
  }
  return isCodeOK;
}
```

Figura 3.c.2: lectura de tarjeta RFID

d) GSM

En cuanto a el GSM, hemos utilizado también una biblioteca, GSM.h. Además de SPI.h, que encapsula el uso del SPI, bus que también usa el GSM.

Estas bibliotecas nos proveen de manejadoras para realizar llamadas, mandar SMS, acceder a la SIM, …
En nuestro caso, solo hemos necesitado el manejador principal del GSM (GSM) y el manejador de SMS (GSM_SMS), con los cuales hemos podido desbloquear la SIM e iniciar el GSM, y mandar un SMS, respectivamente.

En cuanto a la inicialización, como se puede ver en la Figura 3.a.6, basta con llamar a la función begin y pasarle como parámetro el código de desbloqueo de la SIM, en este caso ninguno.

En cuanto a mandar un SMS, como se ve en la Figura 3.d.2, hay que iniciar el envio indicando el número de móvil del receptor. Después se pueden escribir

hasta 140 caracteres mediante funciones print, y, por último, se envía el SMS haciendo un end.
Cabe destacar que esta última función tarda un poco en ejecutarse, debido a la naturaleza de las comunicaciones.

Cabe destacar que el código del GSM lo tuvimos que reescribir dos veces, pues al principio utilizábamos una biblioteca llamada SIM900.h, la cual abstraía todavía más el funcionamiento del GSM, pero tenia muy mal rendimiento y funcionaba muy mal.

```
GSM_SMS sms;
GSM gsmAccess;
```

Figura 3.d.1: manejadores GSM

```
void sendSMS(){
  sms.beginSMS(PHONENUMBER);
  sms.print(SMSCONTENT);
  sms.endSMS();
}
```

Figura 3.d.2: mandar un SMS

e) LCD Display

Para manejar el LCD, utilizamos la biblioteca LiquidCristal, con la cual ya hemos trabajado en clase, y nos permite abstraer el escribir en pantalla a funciones del tipo print y println a las que estamos acostumbrados, además de cambiar la posición del cursor con funciones setcursor.

Al manejador del LCD, hay que proporcionarles los pines de reset, enable y los cuatro pines por donde se van a enviar los datos.

Por último en este apartado, cabe destacar que el LCD lo hemos utilizado en varias partes del programa, pero que quizás su uso más destacado es en la función printState, en la cual según el estado se imprimen distintas cosas por pantalla.

```
LiquidCrystal lcd(LCD_RSPIN, LCD_ENPIN, LCD_DATA1, LCD_DATA2, LCD_DATA3, LCD_DATA4);
```

Figura 3.e.1: manejador del LCD

```
void printState(){
    lcd.clear();
    switch(state){
      case ACTIVE:
          lcd.print("     ALARM");
          lcd.setCursor(0,1);
          lcd.print("   ACTIVATED");
          break;
      case INACTIVE:
          lcd.print("     ALARM");
          lcd.setCursor(0,1);
          lcd.print("    INACTIVE");
          break;
      case ALERT:
          time_t currentTime = now() - alertStartTime;
          long remainingTime= MAX_ALERT_TIME - currentTime;
          lcd.print("     ALERT ");
          lcd.setCursor(0,1);
          lcd.print("      ");
          lcd.print(remainingTime);
          lcd.print("s");
      break;
  }
}
```

Figura 3.e.2: función principal en la que se imprime por el LCD según el estado

f) LED y Buzzer

El uso de estos dos fue muy sencillo, e igual a como lo que hemos hecho en clase.
Únicamente como ya hemos comentado en punto 3.a, declaramos los pines donde los conectamos, como pines de salida.
Más adelante en el código los iniciamos cuando se entra en estado de alerta con las funciones digitalWrite (LED) y tone(Buzzer), y los apagamos cuando se desactiva la alarma o se pasa el periodo establecido para introducir la tarjeta.

g) Gestión de temporizador

Para la medición del tiempo en la función alarmAlertBehaviour, hemos utilizado una biblioteca llamada Time. Esto lo decidimos así, puesto a que el realizar mediciones de tiempo mayores a 32 segundos, no nos funcionaba bien. Esto lo achacamos al reloj del propio Arduino, que no tiene capacidad suficiente para contar un tiempo grande, por eso al buscar por Internet

descubrimos esta biblioteca que abstraía el uso del tiempo a la función now, de forma que nos daba un timestamp y no el número de milisegundos transcurridos desde que se encendió el microcontrolador, permitiéndonos así manejar tiempos todo lo grandes que queramos.

IV. Referencias

- https://hetpro-store.com/TUTORIALES/modulo-lector-rfid-rc522-rf-con-arduino/
- https://telectronica.com/que-es-un-lector-rfid/
- https://programarfacil.com/tutoriales/fragmentos/arduino/texto-en-movimiento-en-un-lcd-con-arduino/
- https://www.luisllamas.es/arduino-buzzer-activo/
- https://www.arduino.cc/en/Tutorial/LiquidCrystalDisplay
- https://github.com/Martinsos/arduino-lib-hc-sr04
- https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/
- https://www.geekfactory.mx/tutoriales/tutoriales-arduino/pantalla-lcd-16x2-con-arduino/
- https://github.com/PaulStoffregen/Time
- https://playground.arduino.cc/Learning/MFRC522/
- https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/
- https://www.prometec.net/buzzers/
- https://es.wikipedia.org/wiki/RFID
- https://en.wikipedia.org/wiki/Liquid-crystal_display
- https://es.wikipedia.org/wiki/Sistema_global_para_las_comunicaciones_m%C3%B3viles
- https://www.arduino.cc/en/Reference/GSM
- https://forum.arduino.cc/index.php?topic=364386.0
- https://github.com/Seeed-Studio/GPRS_SIM900