UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

SISTEMA DE DETECCIÓN DE OBSTÁCULOS CON LUCES AUTOMÁTICAS EN UN MICROCONTROLADOR AT89C5131A-UM

Oscar Ares
Arnau González
Àlex Giménez

Informática Industrial IIEIA
Curso 2024-2025
Grupo M2
13/06/2024

ÍNDICE

1. INTRODUCCIÓN

1.1 Introducción

Los robots autónomos son máquinas programadas para realizar tareas específicas sin la necesidad de que un humano intervenga. Estos están equipados con una gran variedad de sensores que gracias a ellos pueden percibir su entorno, tomar decisiones y actuar en función de cómo este programado. Uno de los ejemplos más comunes de estos robots es aquel diseñado para detectar objetos en su camino y modificar su trayectoria para evitarlos. El funcionamiento de estos robots que esquivan obstáculos está basado en sensores como cámaras, ultrasonidos, y LIDAR (un sensor que permite al robot crear un mapa de su entorno y detectar objetos a su alrededor). Si combinamos estos sensores con el conocimiento humano para programar, se podrán crear los denominados robots autónomos.

Figura 1: *Modelo Roomba*

Estos robots encuentran aplicaciones en numerosos campos debido a su capacidad de operar de manera independiente y eficiente. En entornos industriales, los robots autónomos se utilizan para la logística y el transporte de materiales dentro de almacenes y fábricas, aumentando la productividad y reduciendo los riesgos laborales. Pero no solo están destinados para las industrias, en el ámbito doméstico, los robots aspiradores como la famosa marca Roomba, son un ejemplo popular que, al detectar obstáculos como muebles, modifican su ruta para limpiar eficientemente el hogar.

Debido a nuestro interés por este tema, hemos decidido replicar a escala pequeña y adaptándolo a nuestros conocimientos a través de esta actividad dirigida. Esta memoria se trata de la memoria técnica de la actividad dirigida para la asignatura de informática industrial. Se pretende explicar todo el proceso de diseño y programación de nuestro robot autónomo basándonos en nuestros objetivos y comentando si estos han sido un éxito o un fracaso.

1.2 Objetivos

Los principales objetivos de estudio son:

- Creación de la placa principal la cual compondrá el microcontrolador y será la base de nuestro proyecto y comprobar mediante las entregas iniciales e intermedias su correcto funcionamiento.
- Detección de la distancia gracias al ultrasónico y muestrearlo en la pantalla LCD
- Mediante un LDR y un ADC, controlar la intensidad de los leds que simularán los faros de nuestro robot autónomo proporcionando más luz o menos en función de la luminosidad del ambiente que le rodee.
- Investigar sobre el funcionamiento de los motores e intentar implementarlos con el robot
- Comprobar si todas las funciones programadas funcionan a la vez creando nuestro robot autónomo que gira cuando detecta una presencia.

Debido a que algunos de nuestros objetivos son bastante ambiciosos, al final de esta memoria y tras la presentación realizada al profesorado comentaremos cuales de estos objetivos han sido logrados y cuáles no.

1.3 Planteamiento del proyecto

Tras revisar los aspectos comentados en los apartados anteriores, el principal objetivo de este proyecto es la creación de un robot autónomo capaz de esquivar todos los objetos que se interpongan en su trayectoria. Inicialmente, planteamos que el robot fuera capaz de salir de un laberinto. Sin embargo, debido a la complejidad inherente en la construcción del robot, consideramos que, con nuestros conocimientos actuales, lograr que el robot evite obstáculos ya representa una gran satisfacción y un logro significativo. Es por ello por lo que las funcionalidades del robot serán las siguientes:

- El principal sensor de nuestro proyecto será el sensor ultrasónico, ya que desempeña la función más importante: detectar la distancia a la que se encuentra cualquier objeto que se interponga en su trayectoria. Una vez detectada la distancia, esta información se enviará a la pantalla LCD para visualizarla. Es importante mencionar que existen diversas metodologías para calcular la distancia con un sensor ultrasónico. Sin embargo, tras realizar varios experimentos y analizar los errores, hemos identificado que la implementación que utilizamos es la que ha proporcionado los mejores resultados. Esta implementación se explicará detalladamente en su correspondiente apartado.

- Debido a que el sensor ultrasónico ya trabaja con señales digitales, para este proyecto también utilizaremos un sensor que proporcione información analógica, la cual será convertida a digital mediante un ADC (Convertidor Analógico a Digital). Para ello, utilizaremos un LDR (Light Dependent Resistor). Un sensor LDR es un tipo de resistencia variable cuya resistencia eléctrica cambia en función de la intensidad de la luz que incide sobre él. Aprovecharemos esta característica del LDR para controlar la intensidad de unos LEDs. Estos se colocan en la parte delantera del robot por lo que simulan los faros y proporcionaran más intensidad o menos en función de la luz que haya a su alrededor.

- Además de estos sensores, consideramos interesante que el robot pudiera moverse. Aunque nunca hemos trabajado con motores y reconocemos la complejidad que esto implica, hemos decidido implementar 4 motores (2 a cada lado). Estos motores funcionarán de manera simultánea, permitiendo que el robot gire hacia un lado cuando detecte un objeto a una distancia mínima, evitando así la colisión.

Con todo esto, creemos que este es un proyecto muy ambicioso, ya que abarca sensores novedosos con los que tenemos poco conocimiento y que no son para nada sencillos. Por lo tanto, en este proyecto no solo hemos implementado los conocimientos adquiridos en clase, sino que, a diferencia de otros proyectos, también hemos tenido que investigar y profundizar en el funcionamiento de todos los componentes antes de intentar programarlos.

2. DISEÑO ELECTRÓNICO

2.1 Placa inicial

Para la explicación de los componentes utilizados lo dividiremos en dos partes, la primera serán aquellos componentes utilizados para la creación de la placa principal que será el cerebro de nuestro robot y es aquella comprobada y utilizada para las entregas inicial e intermedia.

Para la realización de esta placa se nos proporcionó la pcb con el circuito integrado y debimos comprar los componentes, soldarlos y comprobar su funcionamiento.

Figura 2: *Placa inicial*

Tabla 1: *Componentes placa inicial.*

Cantidad	Ref. diseño	Componente
1		Cable USB, con conectores tipo A y tipo B
1	J1	Conector USB hembra tipo B circuito impreso
1	J3	Jack alimentación circuito impreso
1	SV1	Conector 2*20 pin recto macho 2,54mm
1	IC2	Regulador de tensión 7805
1	U1	Micro AT89C5131A-UM, encapsulado PLCC52
1		Zócalo PLCC52
1	Q1	Cristal de cuarzo de 12 MHz
2	S1, S2	Pulsadores normalmente abiertos pequeños
2	JP1, JP2	Jumper 2,54mm
1	D1	LED rojo
1	T1	Transistor BC547
3	R1, R5, R7	Resistencia 1,5 Kohmios
1	R2	Resistencia 1 Kohmio
2	R3, R4	Resistencia 27 ohmios
1	R6	Resistencia 100 ohmios
2	C1,C2	condensador 22 picofaradios
1	C3	Condensador 10 microfaradios
1	C4	Condensador 2,2 nanofaradios
1	C5	Condensador 10 nanofaradios
2	C6, C7	Condensador 100 nanofaradios
1	C8	Condensador 0,33 microfaradios
1	C9	Condensador 0,1 microfaradios
1	C10	Condensador 1 microfaradio

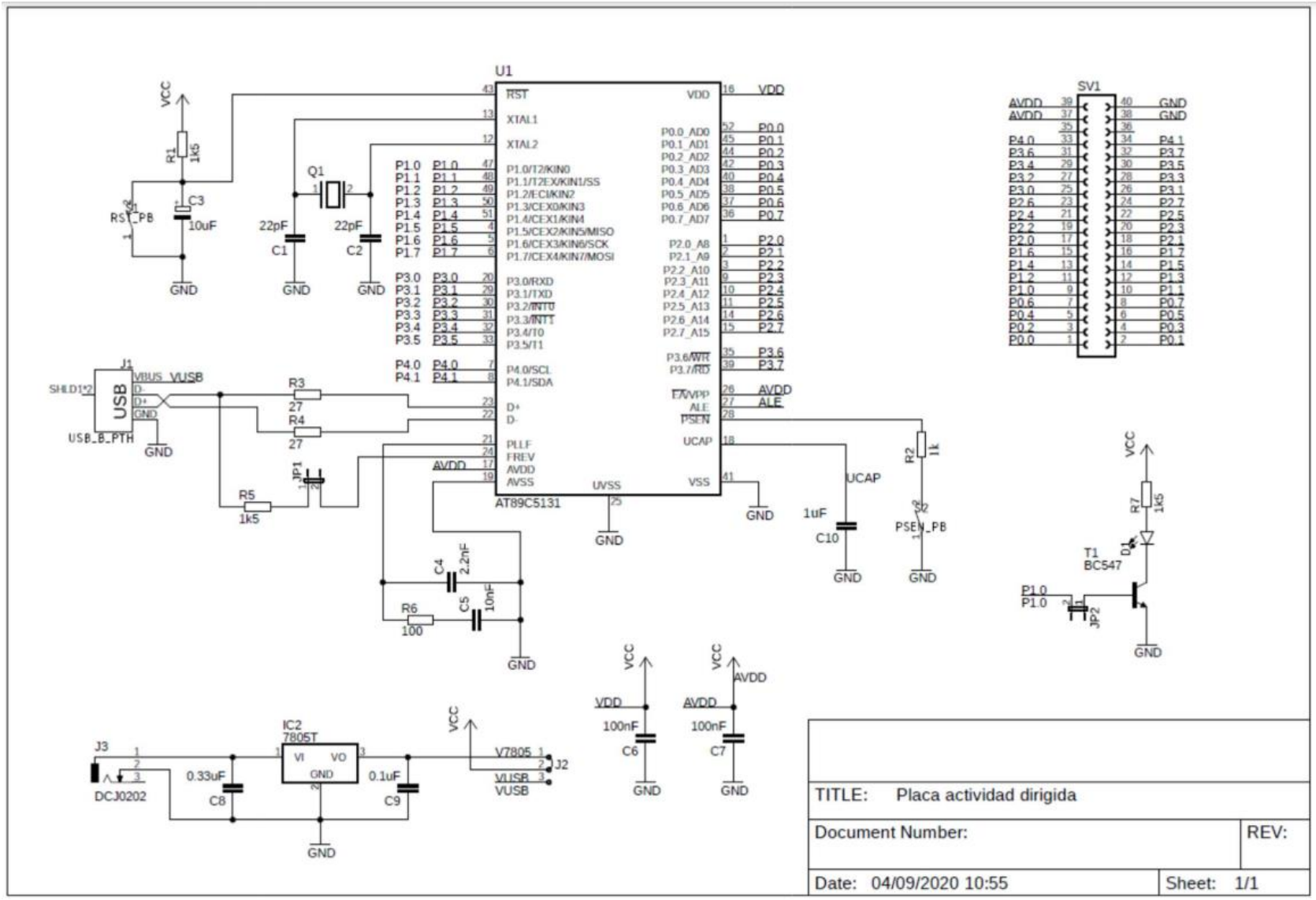


Figura 3: *Esquema electrónico de la placa inicial*

2.2 Componentes añadidos

Tabla 2: *Componentes añadidos*

Cantidad	Componente
1	Ultrasónico modelo HCSR04
6	Leds
1	Pulsador
1	ADC
1	LDR
1	Pantalla LCD
4	Motores con ruedas
1	Driver
1	Batería de pilas
1	Protoboard
1	Estructura

2.2.1.1 Ultrasónico HCSR04

Figura 4: *Ultrasónico HCSR04*

El sensor de ultrasonidos HCSR04 es el que hemos utilizado y el más comúnmente empleado en proyectos de robótica básica. Este dispositivo mide distancias con precisión mediante ondas sonoras de alta frecuencia y la detección del eco reflejado por los objetos a su alcance. Su funcionamiento se basa en la emisión de un pulso de ultrasonidos; cuando estas ondas encuentran un obstáculo, se reflejan de vuelta al emisor. Si se mide el tiempo que tarda el eco en regresar, es posible calcular la distancia al objeto.

Este fenómeno es muy interesante ya que esto permite detectar la presencia y distancia de la gran mayoría de objetos incluso a completa oscuridad. Además, es notable que este método de navegación es similar al que utilizan los murciélagos, quienes emiten ultrasonidos y escuchan los ecos para evitar colisiones y moverse en su entorno.

Además, otro factor que hemos ido descubriendo a lo largo del proyecto es que no todos los materiales reflejan bien las ondas. Hay ciertos materiales que influyen en la reflectividad de las ondas ultrasónicas debido a la superficie, densidad e incluso las propiedades del propio material. Los materiales mas comunes y que hemos descubierto que provocan un mal funcionamiento del echo son los siguientes:

- **Materiales Absorbentes**: Estos materiales tienen estructuras porosas que pueden absorber las ondas ultrasónicas, reduciendo el eco reflejado. Un ejemplo de esto podría ser desde unas fundas de gafas tal y como comprobamos en la presentación, hasta objetos de espuma y goma.

- **Materiales Irregulares o Rugosos:** Si la superficie no es del todo lisa, esto puede generar confusión al echo ya que tienden a dispersar las ondas en múltiples direcciones generando una respuesta de recepción al echo muy débil.

- **Materiales Transparentes:** los vidrios e incluso plásticos extremadamente delgados, aunque pueden reflejar las ondas ultrasónicas, permiten que parte de la energía pase a través de ellos y no funcione correctamente.

- **Materiales aislantes Acústicos:** Estos materiales fueron los que más nos sorprendieron. Después de varias pruebas, inicialmente pensamos que el eco estaba defectuoso o mal programado, pero no era así. Los materiales aislantes acústicos están diseñados específicamente para absorber el sonido, lo que reduce drásticamente la reflexión de las ondas ultrasónicas.

- **Espejos y Superficies Metálicas Pulidas:** Los espejos y las superficies metálicas lisas pueden reflejar bien las ondas ultrasónicas, el ángulo de incidencia es crítico y puede reflejar en una dirección totalmente diferente y no regresar al sensor.

A pesar de esto, para nuestro proyecto es un componente bastante adecuado ya que la finalidad de nuestro trabajo es que sea capaz de detectar un objeto y girar, no se le expondrá a materiales que puedan dificultar su mal funcionamiento

La función que realizará en nuestro proyecto es a través del proceso que se comentará posteriormente, detectar a que distancia está el obstáculo que se le interpone en el camino, muestrearlo en la pantalla y accionar a los motores en función de la distancia a la que se encuentre. Esto será útil porque si la distancia que detecta este sensor es más pequeña a un valor mínimo que fijaremos, este girará.

En el modelo HCSR04 las características principales son las siguientes:

- Señal de entrada del disparador: Pulso TTL de 10 µS
- Corriente de trabajo: 15mA
- Ángulo de medición efectivo: 15°
- Rango de medición: 2cm a 400cm
- Frecuencia de operación: 40KHz

Los pines que compondrá este sensor serán los siguientes:

Figura 5: *Pines HCSR04*

- **TRIG (Disparo del sensor):** Recibe un pulso de habilitación de parte del microcontrolador indicándole al sensor que inicie la medición de distancia.
- **ECHO (Echo del sensor)**: Este pin envía una señal al microcontrolador. La duración de esta señal corresponde al tiempo que tarda el sonido en viajar desde el sensor hasta el obstáculo y regresar. La longitud de esta señal se usa para calcular la distancia al objeto.

- **VCC:** Alimentación de 5V
- **GND:** Tierra

Funcionamiento para la programación:

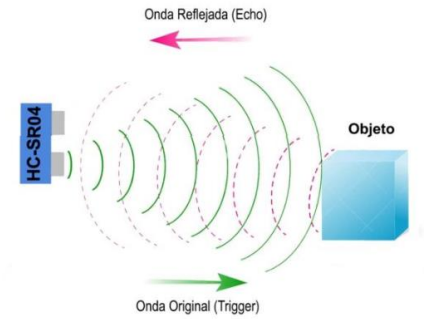


Figura 6: *Funcionamiento a seguir*

1. Enviar pulsos de disparo: El microcontrolador 8051 debe transmitir un pulso de 10 microsegundos (us) al pin Trigger del sensor HCSR04.

2. Emisión y espera: Después de recibir el pulso de disparo, el HCSR04 envía automáticamente ocho ondas sonoras de 40 kHz y espera un pulso ascendente en el pin Echo.

3. **Iniciar el temporizador en el flanco ascendiente:** Cuando el microcontrolador detecta un flanco ascendente (Echo pasa de un estado bajo, 0 lógico, a un estado alto, 1 lógico) en el pin Echo, que está conectado a una entrada del 8051, debe iniciar el temporizador.

4. **Esperar al flanco descendiente**: Cuando el pulso ultrasónico se refleja en un objeto y vuelve al sensor, el pin Echo cambia en flanco descendente (el pin Echo vuelve a cambiar de estado, esta vez de alto (1 lógico) a bajo (0 lógico)). Este cambio señala el final del tiempo de vuelo del pulso ultrasónico.

5. **Parar el temporizador y calcular la distancia:** Al detectar el flanco descendente, el microcontrolador detiene el temporizador y lee el valor contado. Este valor corresponde al tiempo total que tardó el pulso en ir y venir, y se usa para calcular la distancia.

Para calcular la distancia en centímetros aplicaremos la siguiente formula:

$$Distancia(cm) = \frac{tiempo\ (us)}{58}\ (1)$$

Esta fórmula es una simplificación de varios factores. Primero hay que tener en cuenta que la velocidad del sonido es de 0,034 cm/us. Además, el sonido hace ida y vuelta, por lo que es lo mismo que decir que el sonido hace 2 veces ida. Por lo tanto, sacamos como conclusión la fórmula 2.

$$IDA = \frac{ida\ y\ vuelta}{2}\ (2)$$

Si combinamos estos dos factores (1) y (2), sacamos la siguiente fórmula:

$$dist(cm) = \frac{tiempo(us)}{2} * 0'034\ \ (3)$$

Si simplificamos la fórmula 3, queda de la siguiente manera:

$$Distancia(cm) = \frac{tiempo\ (us)}{58}\ (4)$$

2.2.1.2 Programación ultrasónico HCSR04

A continuación, se proporciona únicamente aquello que ha sido utilizado para la programación del ultrasónico.

Primeramente, se definen los pines.

```
// Definir pines ECHO
sbit TRIG_PIN = P1^6; // Pin de disparo (TRIG)
sbit ECHO_PIN = P2^2; // Pin de eco (ECHO)
```

Para poder utilizar las funciones y ejecutarlas de manera exitosa se deben declarar las funciones previamente.

```
// Definicion funciones
void trigger_pulse();
void delay_us(unsigned int us);
unsigned int measure_distance();
void int_to_string(unsigned int num, char *str);
```

La primera función que utilizaremos será timer0_int()

```
// Función para inicializar el temporizador 0 en modo 1
void timer0_init() {
    TMOD &= 0xF0; // Limpiar bits de Timer 0
    TMOD |= 0x01; // Configurar Timer 0 en modo 1 (16 bits)
}
```

Esta función la utilizaremos para inicializar el Timer 0 del microcontrolador 8051 en modo 1, que es un temporizador de 16 bits. El timer puede operar en varios modos tal y como se nos ha explicado en clase, pero hemos utilizado el modo 1 ya que es un temporizador/counter de 16 bits que permite contar desde 0x0000 hasta 0xFFFF (0 a 65535 en decimal) y nos permite contar ciclos de reloj y medir intervalos de tiempo con alta precisión. Este modo permite contar con una resolución de 16 bits, lo cual es crucial para medir con precisión la duración del pulso de eco del sensor ultrasónico.

A continuación, crearemos los retardos para poder utilizar el ultrasónico.

```
void delay_1us() {
    TH0 = 0xFF;   // Cargar el valor alto para 1 us (ajustado)
    TL0 = 0xF4;   // Cargar el valor bajo para 1 us (ajustado)
    TR0 = 1;      // Iniciar el temporizador
    while (!TF0); // Esperar hasta que se desborde el temporizador
    TR0 = 0;      // Detener el temporizador
    TF0 = 0;      // Borrar el flag de desbordamiento del temporizador
}

// Función para generar un retardo de n microsegundos
void delay_us(unsigned int us) {
    unsigned int i;
    for (i = 0; i < us; i++) {
        delay_1us();
    }
}
```

La función delay_1us genera un retardo de aproximadamente 12 microsegundo (us) usando el Timer 0. El nombre de la función en si es erróneo, pero hace cuando trabajamos con este realmente estamos aplicando 12 microsegundos.
Para ello cabe destacar que:

- TH0 y TL0 son los registros de 8 bits que juntos forman el registro de 16 bits del Timer 0.

- Los valores que hemos añadido al valor bajo y al valor alto son FF para el alto y F4 para el bajo, si hacemos el cálculo se aprecia que de 4 a F van los 10 us deseados, pero después también hay que tener en cuenta lo que tarda en realizar la operación al activar y desactivar el temporizador con la instrucción TR0, por lo que en total son aproximadamente 12 us debido a los dos ciclos de máquina que debe de hacer para ejecutar el ciclo.

Y la función delay_us genera un retardo de (n) microsegundos llamando a delay_1us().

Una vez generado el retardo con temporizadores, generaremos la función para generar un pulso de disparo.

```
void trigger_pulse() {
    TRIG_PIN = 1; // Establecer el pin de disparo en alto
    delay_us(10); // Esperar un corto periodo de tiempo (al menos 10us)
    TRIG_PIN = 0; // Apagar el pin de disparo
}
```

Con esa función y añadiendo el valor de 10 en el delay, generaremos un disparo de 10us a través del pin Trigger.

Con la función creada solo nos queda crear una última función que medirá la distancia a la que se encuentra el objeto.

```
unsigned int measure_distance() {
    float distance_measurement, valor;
    unsigned long duration;
    unsigned int distance_cm;

    trigger_pulse();

    while (!ECHO_PIN); // Esperar hasta que el pin de eco se active (se ponga en alto)
    TR0 = 1; // Iniciar el temporizador

    while (ECHO_PIN && !TF0); // Esperar hasta que el pin de eco se desactive (se ponga en bajo)
    TR0 = 0; // Detener el temporizador

    // Calcular la distancia en centímetros
    //valor = 1.085e-6 * 34300;
    duration = (TL0 | (TH0 << 8)); // Leer el valor del temporizador
    distance_measurement = duration / 58 ; // Calcular la distancia (ida y vuelta)
    distance_cm = (unsigned int)distance_measurement;

    return distance_cm;
}
```

En esta función primeramente enviaremos el pulso con la función trigger_pulse. Posteriormente esperaremos hasta que el pin del echo se ponga en alto, indicando el inicio del temporizador, y cuando el pin del echo se ponga en nivel bajo, detiene el temporizador y calcula la distancia con la formula.
El valor que esta comentado es el que utilizamos primeramente para calcular la función con la formula completa, pero al final decidimos implementar la formula simplificada ya que era más exacta aún.

Finalmente, para muestrear la variable en el LCD, necesitamos que fuera una string pero con la función anterior nos proporcionaba una variable int. Es por ello que decidimos implementar una función para pasar de int a string.

```
void int_to_string(unsigned int num, char *str) {
    int i = 0;
    int temp_num = num;

    // Contar el número de dígitos
    do {
        temp_num /= 10;
        i++;
    } while (temp_num != 0);

    str[i] = '\0'; // Añadir el carácter nulo al final

    // Convertir cada dígito a carácter
    while (num != 0) {
        str[--i] = (num % 10) + '0';
        num /= 10;
    }

    if (i == 1) {
        str[0] = '0';
    }
}
```

Con esta función solo queda ir al main e implementarla

```
void main() {
    char distance_str[10]; // Cadena para almacenar la distancia como texto
    char value_str[10];
    unsigned int previous_distance = 0xFFFF; // Valor inicial que nunca será igual a una distancia medida
    unsigned int previous_adc_value = 0xFFFF; // Valor inicial que nunca será igual a un valor ADC medido

    P1 = 0x00;
    P3 = 0x00;
    P2 = 0xFF; // Configurar P2 como entrada
    estado_anterior = 0;
    timer0_init(); // Inicializar el temporizador
    I2C_Init();
    LCD_init(); // Inicializar el LCD

    while (1) {
        // Medir la distancia utilizando el sensor ultrasónico
        ADS1115_WriteConfig(0x8583);
        distance = measure_distance();
        int_to_string(distance, distance_str); // Convertir la distancia a string
        adc_value = ADS1115_ReadConversion();
        int_to_string(adc_value, value_str);

        if (BOTON == 1) { // Si el interruptor está en posición encendido
          //if(distance != previous_distance || adc_value != previous_adc_value){
            LCD_cmd(0x80);
            LCD_write_string("D: "); // Mostrar "Distancia: " en el LCD
            LCD_write_string(distance_str); // Mostrar la distancia medida en el LCD
            LCD_write_string(" cm"); // Mostrar " cm" en el LCD
```

Dentro del while, añadimos int_to_string(distance, distance_str); // Convertir la distancia a string para transformar la variable a string y finalmente en el lcd muestrearla.

2.2.2.1 Leds

Figura 7: *Led*

Decidimos implementar los diodos led de una forma ms original que indicar el estado del robot. Creímos que sería interesante colocar 4 leds, 2 leds a cada lado del coche las cuales simularían los focos del coche e iluminarían más o menos en función de si hay más o menos luz en su ambiente. Pero finalmente debido al caos que se generaría en cuanto al cableado decidimos implementarlo de forma mas cómoda los 4 leds juntos en la protoboard pero explicando su principal función.

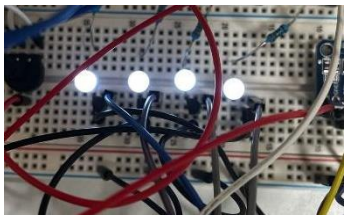


Figura 8: *Leds en la protoboard.*

2.2.2.2 Programación Leds

Primeramente, definimos los puertos de los leds.

```
// Definir pines LED
sbit LED1 = P1^1;
sbit LED2 = P1^2;
sbit LED3 = P1^3;
sbit LED4 = P1^4;
```

Para la programación de encendido de los leds creamos esta función para encender su correspondiente led en función de la luminosidad que capte a su alrededor.

```
void controlar_leds(unsigned int adc_value) {
    // Apagar todos los LEDs inicialmente
    LED1 = 0;
    LED2 = 0;
    LED3 = 0;
    LED4 = 0;

    // Encender los LEDs según el valor del ADC
    if (adc_value < 1024) {
        LED1 = 1;
        LED2 = 1;
        LED3 = 1;
        LED4 = 1;
    } else if (adc_value < 2048) {
        LED1 = 0;
        LED2 = 1;
        LED3 = 1;
        LED4 = 0;
    } else if (adc_value < 3072) {
        LED1 = 1;
        LED2 = 0;
        LED3 = 0;
        LED4 = 1;

    } else {
        LED1 = 0;
        LED2 = 0;
        LED3 = 0;
        LED4 = 0;

    }
}
```

Y simplemente llamamos la función dentro de main cuando el botón este pulsado y ya las encenderá.

Además de esta función tenemos la original que utilizamos para la inicialización y comprobación que todo funciona correctamente. La dejamos en el código porque era el primer paso que activábamos para comprobar el funcionamiento correcto de la placa y además es la secuencia que realiza cuando el botón esta desactivado.

```
void encender_leds_secuencialmente() {

        LED1 = 1;  // Encender LED1
        delay_ms(50); // Esperar 500ms
        LED1 = 0;  // Apagar LED1
        LED2 = 1;  // Encender LED2
        delay_ms(50); // Esperar 500ms
        LED2 = 0;  // Apagar LED2
        LED3 = 1;  // Encender LED3
        delay_ms(50); // Esperar 500ms
        LED3 = 0;  // Apagar LED3
        LED4 = 1;  // Encender LED4
        delay_ms(50); // Esperar 500ms
        LED4 = 0;  // Apagar LED4
        delay_ms(50); // Esperar 500ms
        LED4 = 1;  // Encender LED4
        delay_ms(50); // Esperar 500ms
        LED4 = 0;  // Apagar LED4
        LED3 = 1;  // Encender LED3
        delay_ms(50); // Esperar 500ms
        LED3 = 0;  // Apagar LED3
        LED2 = 1;  // Encender LED2
        delay_ms(50); // Esperar 500ms
        LED2 = 0;  // Apagar LED2
        LED1 = 1;  // Encender LED1
        delay_ms(50); // Esperar 500ms
        LED1 = 0;  // Apagar LED1


}
```

2.2.3.1 Interruptor palanca

En nuestro caso hemos considerado que sería más cómodo implementar un interruptor de palanca, este es un componente esencial en el diseño de nuestro robot, proporcionando un método simple y efectivo para controlar la alimentación eléctrica del sistema. La función que tendrá será simplemente abrir o cerrar el circuito mediante una acción manual proporcionando el encendido o apagado del robot.

Figura 9: Interruptor palanca

2.2.3.2 Programación interruptor palanca

Primeramente, definimos el botón.

```
// Bit boton
sbit BOTON = P2^0;
```

El botón simplemente su programación es muy sencilla, simplemente muestreamos las variables en el LCD si el botón esta activado o en caso contrario que no muestre nada

2.2.4.1 ADC

El ADS1115 es un conversor analógico digital (ADC) externo que podemos conectar a un procesador como Arduino para medir señales analógicas. El funcionamiento de este lo emplearemos para controlar el fotorresistor LDR. El LDR cambia su resistencia en función de la intensidad de la luz. Para poder utilizar esta información en un sistema digital, como un microcontrolador, es necesario convertir esta señal analógica (variación de resistencia) en una señal digital que el microcontrolador pueda procesar. El ADS1115 realiza esta conversión de manera precisa y con alta resolución (16 bits).

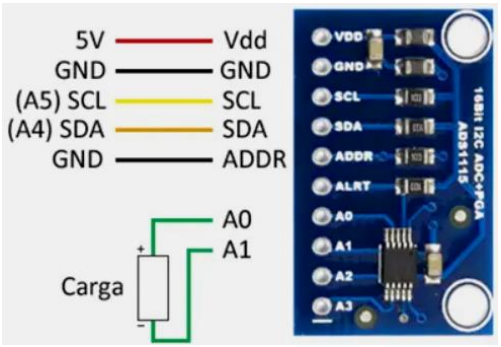


Figura 10: ADC, ADS1115

Los pines que compondrá este ADC serán los siguientes:

- **VDD**: alimentación del dispositivo
- **GND**: tierra
- **SCL (serial clock line):** Este pin es para la línea de reloj del bus I2C. El ADS1115 se comunica con el microcontrolador u otro dispositivo maestro I2C usando este pin para sincronizar la transmisión de datos.
- **SDA (serial Data line):** Este pin es para la línea de datos del bus I2C. Se utiliza para la transmisión de datos entre el ADS1115 y el dispositivo maestro I2C.
- **ADDR**: Este pin se utiliza para configurar la dirección I2C del ADS1115. Puedes conectar este pin a VDD, GND, SDA o SCL para seleccionar una de las cuatro direcciones I2C posibles (0x48, 0x49, 0x4A, 0x4B).
- **ALRT(Alert/Ready):** este pin puede servir para generar una señal de interrupción cuando una conversión está lista o cuando una medida excede un valor límite predefinido, pero nosotros no la utilizaremos.
- **A0 – A3:** Entradas analógicas del ADC. El ADS1115 tiene cuatro entradas analógicas (A0, A1, A2 y A3) que pueden configurarse como cuatro canales de entrada de un solo extremo o dos entradas diferenciales.

Como podemos ver en la figura 11, el tipo de conexionado se especifica en el código. Este conversor analógico-digital ADS1115 tiene cuatro configuraciones de dirección I2C: 0x48, 0x49, 0x4A y 0x4B. En nuestro caso, hemos utilizado la configuración de dirección 0x48. Esta configuración utiliza el siguiente tipo de conexionado: Vcc = 5V, lo cual limita su voltaje y reduce problemas de sobretensión. También hemos utilizado un divisor de tensión para su entrada analógica A0, con una resistencia de 1kΩ, asegurándonos de que los valores estén dentro del rango deseado. Esto es crucial para proteger el ADS1115 y garantizar lecturas precisas, evitando que los voltajes altos dañen el dispositivo o afecten la precisión de las mediciones.

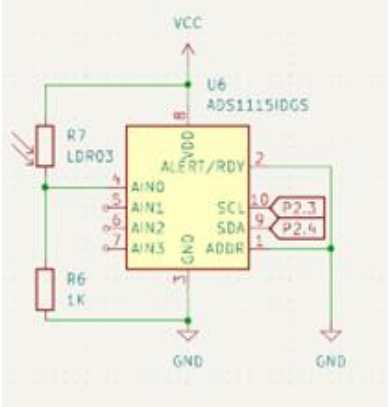


Figura 11: ADC, ADS1115

A continuación, se muestra el código de configuración del ADS1115, junto con la definición de sus pines y las funciones asociadas. Este código utiliza el protocolo I2C, que se explicará en detalle en el próximo apartado.

2.2.4.2 Definiciones y Configuración del ADS1115

El ADS1115 es un convertidor analógico a digital de 16 bits con cuatro entradas multiplexadas, capaz de proporcionar medidas precisas de voltaje a través de una interfaz I2C. A continuación, se definen las direcciones y registros necesarios para su configuración y operación.

```
// Direcciones del ADS1115
#define ADS1115_ADDRESS 0x48  // Dirección I²C del ADS1115

// Registro de configuración del ADS1115
#define CONFIG_REGISTER 0x01
#define CONVERSION_REGISTER 0x00
```

La dirección I2C del ADS1115 puede variar dependiendo de la configuración de los pines ADDR del dispositivo. En este caso, se asume que la dirección es 0x48.

2.2.4.3 Funciones de Configuración y Lectura

Función para Escribir la Configuración del ADS1115
La función ADS1115_WriteConfig configura el ADS1115 enviando los datos adecuados al registro de configuración a través de la interfaz I2C.

```
void ADS1115_WriteConfig(unsigned int config) {
    I2C_Start();
    I2C_Write((ADS1115_ADDRESS << 1) | 0);  // Dirección + bit de escritura
    I2C_Write(CONFIG_REGISTER);  // Registro de configuración
    I2C_Write((config >> 8) & 0xFF);  // Byte alto de la configuración
    I2C_Write(config & 0xFF);  // Byte bajo de la configuración
    I2C_Stop();
}
```

Esta función toma un valor de configuración de 16 bits y lo divide en dos bytes para enviarlos al registro de configuración del ADS1115. La comunicación se maneja a través de las funciones I2C_Start, I2C_Write, y I2C_Stop, que hemos definido previamente nosotros, entendiendo el protocolo y cómo funciona la tecnología.
Función para Leer la Conversión del ADS1115
La función ADS1115_ReadConversion lee el valor de conversión actual desde el ADS1115.

```
unsigned int ADS1115_ReadConversion(void) {
    unsigned int value;
    I2C_Start();
    I2C_Write((ADS1115_ADDRESS << 1) | 0);  // Dirección + bit de escritura
    I2C_Write(CONVERSION_REGISTER);  // Registro de conversión
    I2C_Start();
    I2C_Write((ADS1115_ADDRESS << 1) | 1);  // Dirección + bit de lectura
    value = ((unsigned int)I2C_Read(1) << 8);  // Leer byte alto
    value |= I2C_Read(0);  // Leer byte bajo
    I2C_Stop();
    return value;
    adc_value = value;
}
```

Esta función inicia la comunicación I2C, selecciona el registro de conversión del ADS1115 y lee los dos bytes del valor de conversión. Similar a la función de escritura, utiliza las funciones de la librería I2C para manejar la comunicación.

2.2.4.4 Protocolo de comunicación I2C.

¿Qué es el Protocolo I2C?

I2C (Inter-Integrated Circuit) es un protocolo de comunicación serie desarrollado por Philips Semiconductor en 1982. Se utiliza para comunicar dispositivos de baja velocidad en la misma placa de circuito. Además, este protocolo únicamente utiliza dos líneas de transmisión bidireccionales SCL y SDA. Sus funciones son:

- SCL (Serial Clock Line): Línea de reloj de control.
- SDA (Serial Data Line): Línea de datos de información.

¿Cómo funciona?

El protocolo de comunicación utiliza el modelo maestro-escalvo. En este contexto, el maestro inicia y controla la comunicación, mientras que el esclavo responde a todas las ordenes del maestro.

1. **Condición de inicio:** El protocolo es iniciado siempre por el maestro. Para iniciar la comunicación, partiendo de que tanto SDA como SCL se encuentran a nivel lógico 1 el maestro debe de bajar SDA a 0 y después bajar SCL a 0 también.

2. **Enviar información:**
    a. A continuación, con el SCL en bajo, el maestro pone el valor de bit en SDA.
    b. Posteriormente SCL se pone en alto para que el esclavo lea el bit del SDA.
    c. Se repite el proceso para todos los bits del byte.

3. **Señal ACK**: Cada vez que se haya transmitido un byte, el receptor envía una señal ACK (Acknowledgement), para esto:
    a. Después de enviar cada byte, el maestro libera la línea SDA.
    b. El esclavo pone SDA en bajo para enviar un ACK (nivel lógico 0).
    c. SCL se pone en alto para leer el ACK.
    d. Si la comunicación es exitosa, el esclavo envía un ACK (0 lógico). Si no puede responder, envía un NACK (1 lógico).

4. Fin de la comunicación: El maestro pone SCL en alto y luego SDA en alto.

Para escribir datos, primero se envía la dirección del esclavo junto con la orden de escritura. Luego, se especifica el registro al que se desea acceder y, finalmente, se envía el valor que se quiere escribir en dicho registro. Aunque el registro puede constar de múltiples bytes, toda la comunicación se realiza byte a byte, con una señal de reconocimiento (ACK) entre cada uno de los bytes.
Para leer datos, el protocolo se inicia de manera similar: enviando la dirección del esclavo junto con una orden de lectura, seguida del registro que se desea leer. Posteriormente, se vuelve a enviar la señal de inicio del protocolo (sin enviar una señal de paro), esta vez con la dirección del esclavo y la señal de lectura. En este momento, el esclavo toma el control de la línea de datos (SDA) para enviar los datos, mientras que el maestro sigue controlando el reloj (SCL), a menos que el esclavo lo "secuestre". Cada vez que se recibe un byte de información, el maestro debe responder con una señal de reconocimiento (ACK). Cuando se desea finalizar la lectura, después del último byte, se envía una señal de no reconocimiento (NACK, es decir, un 1 lógico) para indicar al esclavo que detenga el envío de datos.

2.2.4.5 Funciones de Inicialización y Control del Protocolo I2C

Para implementar el protocolo de comunicación I2C en un microcontrolador, es necesario definir una serie de funciones que gestionen las operaciones de inicio, parada, escritura y lectura en el bus I2C. A continuación, se describen estas funciones:

Función de Inicialización: void I2C_Init(void)

```
void I2C_Init(void) {
    SDA = 1;
    SCL = 1;
}
```

La función I2C_Init se utiliza para configurar las líneas SDA y SCL en un estado inicial. Al establecer ambas líneas en 1 (alto), el bus I2C se prepara para iniciar una comunicación. Esto garantiza que el bus esté en un estado de reposo, listo para cualquier operación subsiguiente.

```
void I2C_Start(void) {
    SDA = 1;
    SCL = 1;
    SDA = 0;
    SCL = 0;
}
```

La función I2C_Start inicia una comunicación en el bus I2C. Se asegura que tanto SDA como SCL están en alto antes de bajar la línea SDA a 0, seguido de la línea SCL a 0. Este cambio de nivel en SDA mientras SCL está en alto es la señal de inicio para el protocolo I2C, indicando a todos los dispositivos en el bus que una nueva comunicación está comenzando.

Función de Parada: void I2C_Stop(void)

```
void I2C_Stop(void) {
    SCL = 0;
    SDA = 0;
    SCL = 1;
    SDA = 1;
}
```

La función I2C_Stop finaliza la comunicación en el bus I2C. Coloca primero la línea SCL en bajo, luego baja la línea SDA. Después, pone SCL en alto y finalmente SDA en alto. Este cambio de niveles indica el final de la comunicación, liberando el bus para otras operaciones.

```
void I2C_Write(unsigned char dat) {
    unsigned char i;
    for (i = 0; i < 8; i++) {
        SDA = (dat & 0x80) >> 7;
        SCL = 1;
        dat <<= 1;
        SCL = 0;
    }
    SDA = 1;  // Liberar línea de datos para ACK
    SCL = 1;
    SCL = 0;
}
```

La función I2C_Write envía un byte de datos al bus I2C. El proceso se realiza bit a bit:

1. Para cada bit del byte, coloca el bit más significativo en la línea SDA.
2. Luego, pone la línea SCL en alto para que el esclavo pueda leer el bit.
3. Desplaza el byte hacia la izquierda y repite el proceso para los ocho bits.
4. Finalmente, libera la línea SDA para recibir la señal de ACK del esclavo, y controla la línea SCL para sincronizar la comunicación.

```
unsigned char I2C_Read(unsigned char ack) {
    unsigned char i, dat = 0;
    SDA = 1;  // Liberar línea de datos para entrada
    for (i = 0; i < 8; i++) {
        dat <<= 1;
        SCL = 1;
        dat |= SDA;
        SCL = 0;
    }
    SDA = ack ? 0 : 1;  // ACK o NACK
    SCL = 1;
    SCL = 0;
    return dat;
}
```

La función I2C_Read recibe un byte de datos del bus I2C. Funciona de la siguiente manera:

1. Libera la línea SDA para permitir la entrada de datos del esclavo.
2. Para cada uno de los ocho bits, desplaza el byte acumulado hacia la izquierda.
3. Controla la línea SCL para leer el bit del esclavo y lo añade al byte acumulado.
4. Después de leer el byte completo, coloca la línea SDA en bajo si se espera un ACK, o en alto para un NACK.
5. Controla SCL para sincronizar la señal de reconocimiento y devuelve el byte leído.

Estas funciones forman la base para la comunicación I2C en un microcontrolador, permitiendo la transmisión y recepción de datos a través del bus I2C con dispositivos esclavos.

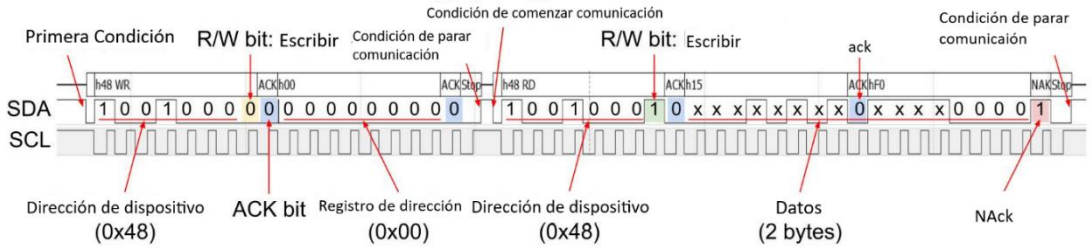


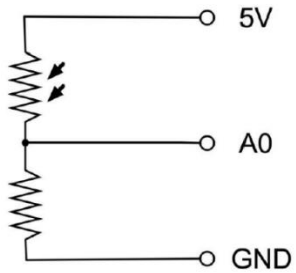
Figura 12: Secuencia comunicación

La secuencia de comunicación comienza con el maestro generando una condición de inicio (SDA alta mientras SCL está alta), seguida del envío de la dirección del esclavo con un bit de lectura/escritura. Luego, los datos se transmiten bit a bit, con el maestro controlando el reloj (SCL). Después de cada byte enviado o recibido, el receptor envía una señal de reconocimiento (ACK). La comunicación finaliza con una condición de parada (SDA alta mientras SCL está alta). Las funciones I2C_Init, I2C_Start, I2C_Stop, I2C_Write, y I2C_Read implementan estas operaciones, gestionando la inicialización del bus, la transmisión y recepción de datos, y las señales de inicio y parada necesarias para el correcto funcionamiento del protocolo.

2.2.5 LDR (Resistor dependiente de luz)

La fotocélula utilizada es un tipo de componente conocido como resistor dependiente de luz, o LDR (Light Dependent Resistor). Como su nombre indica, estos componentes actúan como resistencias la cual varía en función de la cantidad de luz que incide sobre ellos. Nuestro LDR tiene una resistencia de aproximadamente 50 kΩ en condiciones de casi oscuridad y alrededor de 500 Ω en luz brillante. La manera más sencilla de utilizarlo es combinarlo con una resistencia fija. Juntas forman un divisor de tensión que cambia su salida en función de la luz ambiente tal y como se aprecia en la siguiente figura.

Figura 13: *LDR*



Cuando la luz es muy brillante, la resistencia del LDR es muy baja en comparación con la resistencia fija, lo que provoca una caída de tensión que se puede interpretar como un valor alto (como si el potenciómetro estuviera al máximo). En condiciones de poca luz, la resistencia del LDR es mucho mayor que la de la resistencia fija (1 kΩ, por ejemplo), resultando en una caída de tensión baja (como si el potenciómetro estuviera en su mínima posición).

Figura 14: *Divisor de tensión (LDR+ resistencia)*

Tras considerar esta opción, también se evaluó la posibilidad de no necesitar una división de tensión, ya que el fotodiodo podría ser suficiente para nuestras necesidades.
El fotodiodo, al ser sensible a los cambios en la luminosidad del entorno, permite que el circuito reaccione de manera eficiente ante variaciones en la luz ambiental. En nuestro caso específico, este comportamiento se traduce en la capacidad de ajustar automáticamente la funcionalidad de los diodos LED colocados en la parte delantera del robot. De este modo, el

sistema puede aumentar o disminuir la iluminación según la cantidad de luz detectada, mejorando la adaptabilidad del robot a diferentes condiciones de luminosidad.

Sin embargo, al optar por utilizar un divisor de tensión para el LDR (resistor dependiente de la luz), es crucial considerar la configuración del conversor analógico-digital ADS1115. Este componente es fundamental para convertir las señales analógicas provenientes del LDR en datos digitales que el microcontrolador pueda procesar. La correcta implementación de un divisor de tensión asegura que las variaciones en la resistencia del LDR, debidas a cambios en la luminosidad, se traduzcan en voltajes adecuados para el ADS1115.

Según la configuración especificada en la figura 10, debemos conectar el divisor de tensión de manera precisa para garantizar un funcionamiento óptimo. Esta configuración implica conectar el LDR y una resistencia fija en serie entre el voltaje de alimentación y tierra, con el punto de unión entre ambos conectado a una de las entradas analógicas del ADS1115. De esta manera, el cambio en la resistencia del LDR, causado por variaciones en la luz, produce una variación correspondiente en el voltaje de entrada al ADS1115.

La utilización de esta configuración no solo permite una lectura precisa de los niveles de luz ambiental, sino que también mejora la estabilidad y la precisión del sistema en general. Al implementar correctamente el divisor de tensión y asegurarnos de que el ADS1115 esté configurado según las especificaciones, limitando también el voltaje de alimentación al LDR, garantizamos que nuestro robot pueda adaptarse eficazmente a diversas condiciones de iluminación, optimizando así su rendimiento y funcionalidad.

2.2.6.1 Pantalla LCD

Una pantalla LCD (Liquid Crystal Display) es un dispositivo de visualización que utiliza cristales líquidos para producir imágenes. La pantalla LCD se controla mediante un conjunto de pines de datos y control conectados al microcontrolador. Al enviar comandos y datos específicos a través de estos pines, el microcontrolador puede manipular los cristales líquidos para formar caracteres y gráficos en la pantalla. Antes de utilizar la pantalla LCD, es necesario inicializarla mediante una secuencia específica de comandos. Esta inicialización incluye configurar el número de líneas de la pantalla, el tamaño de los caracteres y otras opciones de visualización. En nuestro proyecto, se nos pedía muestrear una variable en esta, nosotros hemos decidido utilizar la distancia ya que es la más interesante y además nos ayudaba a comprobar si el ultrasónico funcionaba correctamente para evitar colisiones.

Concretamente la pantalla que hemos escogido contiene una retroiluminación de LED y puede mostrar dos filas con hasta 16 caracteres en cada fila. Puede ver los rectángulos para cada carácter en la pantalla y los píxeles que componen cada carácter. La pantalla es blanca en azul y está diseñada para mostrar texto.

Figura 15: *Pantalla LCD*

Los pines que contiene este componente son los siguientes:

- **VSS**: Pin que se conecta a tierra
- **VDD**: Pin que se conecta a un + 5V fuente de alimentación
- **VO**: Un pasador que ajusta el contraste de LCD1602
- **RS**: Un registro seleccione pin que controla donde en memoria de la pantalla LCD datos de escritura. Usted puede seleccionar el registro de datos, que es lo que pasa en la pantalla, o un registro de instrucción, que es donde busca controlador de LCD para obtener instrucciones sobre qué hacer.
- **R/W:** Pin A lectura y escritura que selecciona el modo de lectura o escritura
- **E**: Permitiendo a un puerto con energía de bajo nivel, módulo causas la LDC para ejecutar instrucciones.
- **D0-D7:** puertos de leer y escribir datos
- **A y K:** Control de la retroiluminación LED de los puertos

Además, para la configuración de este LCD podríamos haber utilizado varias librerías que nos facilitarían muchísimo el trabajo, pero creímos que una parte importante de nuestro proyecto era aprender y que mejor manera que adaptar los códigos que nos proporcionaba el profesorado para hacerlo a nuestra manera. Es probable que no sea un éxito total, pero estamos bastante orgullosos de el resultado obtenido.

Sumando a esto, otro factor que jugaba en nuestra contra es que la gran mayoría de librerías están creadas para utilizar un LCD que tenga diferentes pines para leer y escribir, en nuestro caso debido al desconocimiento de esto, compramos sin darnos cuenta una pantalla donde se utilizaba un solo pin para estos dos, lo cual nos ha dificultado su funcionamiento.

2.2.6.2 Programación pantalla LCD

Al inicio, definiremos los pines de control y envio de datos que vamos a tener conectados a nuestro LCD.

```
// Definir pines para el LCD
sbit RS = P3^0;
sbit EN = P3^1;
sbit D4 = P3^2;
sbit D5 = P3^3;
sbit D6 = P3^4;
sbit D7 = P3^5;
```

Posteriormente, definiremos la inicialización del LCD, una secuencia de envío de comandos al LCD para configurarlo correctamente para su uso.

```
// Inicialización del LCD
void LCD_init() {
    delay_ms(10); // Espera de inicialización
    LCD_cmd(0x02); // Modo de 4 bits de datos.
    LCD_cmd(0x28); // 2 líneas, 5x8 matriz de puntos
    LCD_cmd(0x0E); // Mostrar cursor
    LCD_cmd(0x06); // Incrementar el cursor
    LCD_cmd(0x01); // Limpiar pantalla
    delay_ms(10); // Espera de limpieza de pantalla
}
```

Crearemos una función de envío de datos de cada comando visto en el anterior bucle. Necesario para poner a funcionar el LCD en el modo que queremos.

```
// Enviar comando al LCD
void LCD_cmd(unsigned char cmd)
{
    RS = 0; // Modo comando
    D4 = (cmd >> 4) & 0x01;
    D5 = (cmd >> 4) & 0x02;
    D6 = (cmd >> 4) & 0x04;
    D7 = (cmd >> 4) & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_us(1);
    D4 = cmd & 0x01;
    D5 = cmd & 0x02;
    D6 = cmd & 0x04;
    D7 = cmd & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_ms(2); // Espera para comando
}
```

Además, se creará una función para enviar caracter por caracter de nuestro string que se desea mostrar por el LCD.

```
// Función para escribir una cadena en el LCD
void LCD_write_string(const char *str) {
    while (*str) {
        LCD_data(*str++);
    }
}
```

Y, por último, una función de control para que el LCD entienda la información que se quiere mostrar.

```
// Enviar datos al LCD
void LCD_data(unsigned char datos)
{
    RS = 1; // Modo datos
    D4 = (datos >> 4) & 0x01;
    D5 = (datos >> 4) & 0x02;
    D6 = (datos >> 4) & 0x04;
    D7 = (datos >> 4) & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_us(1);
    D4 = datos & 0x01;
    D5 = datos & 0x02;
    D6 = datos & 0x04;
    D7 = datos & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_ms(2); // Espera para datos
}
```

Una vez tenemos todas las funciones, las introduciremos en el main de tal forma que se inicialice primeramente el LCD, y después si el interruptor esta activado, moveremos a la primera línea en caso de la distancia y a la segunda en caso del ADC. Una vez seleccionada cada línea, simplemente se pone el texto que se desea con su variable, en nuestro caso D para distancia y AD para el ADC.

```
void main() {
    char distance_str[10]; // Cadena para almacenar la distancia como texto
    char value_str[10];
    unsigned int previous_distance = 0xFFFF; // Valor inicial que nunca será igual a una distancia medida
    unsigned int previous_adc_value = 0xFFFF; // Valor inicial que nunca será igual a un valor ADC medido
    P1 = 0x00;
    P3 = 0x00;
    P2 = 0xFF; // Configurar P2 como entrada
    estado_anterior = 0;

    timer0_init(); // Inicializar el temporizador
    I2C_Init();

    // Configurar el ADS1115 (modo continuo, AIN0, PGA ±4.096V, 128SPS)
    LCD_init(); // Inicializar el LCD

    while (1) {
        // Medir la distancia utilizando el sensor ultrasónico
        ADS1115_WriteConfig(0x8583);
        distance = measure_distance();
        int_to_string(distance, distance_str); // Convertir la distancia a cadena
        adc_value = ADS1115_ReadConversion();
        int_to_string(adc_value, value_str);

        if (BOTON == 1) { // Si el interruptor está en posición encendido
          //if(distance != previous_distance || adc_value != previous_adc_value){
            LCD_cmd(0x80);
            LCD_write_string("D: "); // Mostrar "Distancia: " en el LCD
            LCD_write_string(distance_str); // Mostrar la distancia medida en el LCD
            LCD_write_string(" cm"); // Mostrar " cm" en el LCD
            delay_ms(5);
            LCD_cmd(0xC0);              // Mover a la segunda línea del LCD
            LCD_write_string("AD: ");    // Mostrar "ADC: " en el LCD
```

2.2.7.1 Motores con ruedas

Inicialmente habíamos planteado la opción de colocar dos motores y una rueda loca en la parte delantera del robot, pero debido a que ya teníamos la estructura montada y realmente no dificultaría mucho la programación decidimos utilizar 4 motores en los que los dos de la derecha funcionan igual y los dos de la izquierda también. Estos motores DC con rueda poseen una caja reductora integrada que les permite entregar un buen torque en un pequeño tamaño y bajo voltaje. La carcasa del motor es de plástico resistente, no tóxico y de color amarillo. Ideal para usar en proyectos de robótica móvil como nuestro robot autónomo.

Figura 16: Motores DC

2.2.7.2 Programación motores con ruedas

Se define al inicio el conexionado de control de nuestros motores DC.

```
//Definir puertos motor
sbit MOTOR1_FWD = P1 ^ 5;
sbit MOTOR1_BWD = P1 ^ 7;
sbit MOTOR2_FWD = P3 ^ 6;
sbit MOTOR2_BWD = P3 ^ 7;
```

Y a continuación, crearemos las funciones necesarias. La primera será la función de control de motores hacia adelante con una combinación binaria de control “10” para cada motor.

```
void motores_adelante() {
    MOTOR1_FWD = 1;
    MOTOR1_BWD = 0;
    MOTOR2_FWD = 1;
    MOTOR2_BWD = 0;
}
```

La segunda será la función de control de motores parados con una combinación binaria de control “00” para cada motor.

```
void motores_parado() {
    MOTOR1_FWD = 0;
    MOTOR1_BWD = 0;
    MOTOR2_FWD = 0;
    MOTOR2_BWD = 0;
}
```

Y la última función será de control de giro con una combinación binaria de control “01” para motor 1 y “10” para motor 2, de esta manera se rota la orientación del coche.

```
void motor1_atras() {
    MOTOR1_FWD = 0;
    MOTOR1_BWD = 1;
    MOTOR2_FWD = 1;
    MOTOR2_BWD = 0;
}
```

Una vez creadas todas las funciones solo queda implementarlo en el main de la siguiente manera:

```
if (distance > 15) {
        motores_adelante();
        } else  {
            motores_parado();
            delay_ms(50);
            motor1_atras();
            delay_ms(50);
        }
```

2.2.8 Driver

Figura 17: Driver

Debido a que utilizaremos una batería externa, hemos decidido implementar un driver específico. Este controlador/driver servirá para dos funciones principales. Primero, permitirá que los motores roten en ambas direcciones. Segundo, proporcionará protección al microcontrolador, evitando que la tensión completa destinada a los motores llegue a él y generar un sobrecalentamiento y dañar los componentes esenciales.

2.2.9 Batería de pilas

Figura 18: Porta pilas

"Debido al alto consumo energético de los motores, no sería factible alimentar todo el sistema del robot con una fuente de 5V. Por esta razón, hemos tomado la decisión de implementar un porta pilas, especialmente para alimentar los cuatro motores principales.

Este componente adicional nos permitirá generar la energía necesaria para dar vida a nuestro robot autónomo y permitirle moverse gracias al funcionamiento de los motores. Es importante destacar que esta parte del diseño es crucial, ya que, sin ella, nuestro robot sería incapaz de realizar cualquier movimiento significativo.

2.2.10 Protoboard

En un proyecto a gran escala, una protoboard, también conocida como placa de pruebas, es un dispositivo utilizado para construir y probar circuitos electrónicos de manera provisional, pero en nuestro caso debido a que es más práctico, dejaremos la placa de pruebas en la estructura. Esta, consiste en una placa con una serie de orificios conectados eléctricamente entre sí en una configuración predeterminada.

Figura 19: Protoboard

Estos orificios están dispuestos en filas y columnas, siguiendo un patrón estándar.

En nuestro proyecto nos servirá para interconectar la gran mayoría de los componentes eléctricos y su correcto funcionamiento.

2.2.11 Estabilizador

Figura 20: Estabilizador

Componente añadido con el conjunto de la batería. Se encarga de estabilizar la tensión entregada de las baterías por tal de garantizar los 5V de salida y en caso de no tener suficiente alimentación no entregar nada por tal de proteger su la vida útil.

2.2.12 Estructura

Para asegurar el funcionamiento óptimo de nuestro robot, hemos decidido incorporar un chasis similar al que se muestra en la imagen adjunta. A pesar de la falta de una fotografía de nuestro propio chasis antes de su montaje completo, esta representación visual proporciona una buena aproximación de su estructura y diseño. Además, el chasis está fabricado sobre metacrilato y lleva todas las perforaciones necesarias para la correcta colocación de todos los componentes que utilizaremos para este nuestro proyecto. Esta elección garantiza una base sólida y estable para el robot, aspecto fundamental para su rendimiento efectivo.

Figura 21: Estructura utilizada

3. DISEÑO ELECTRÓNICO DEFINITIVO

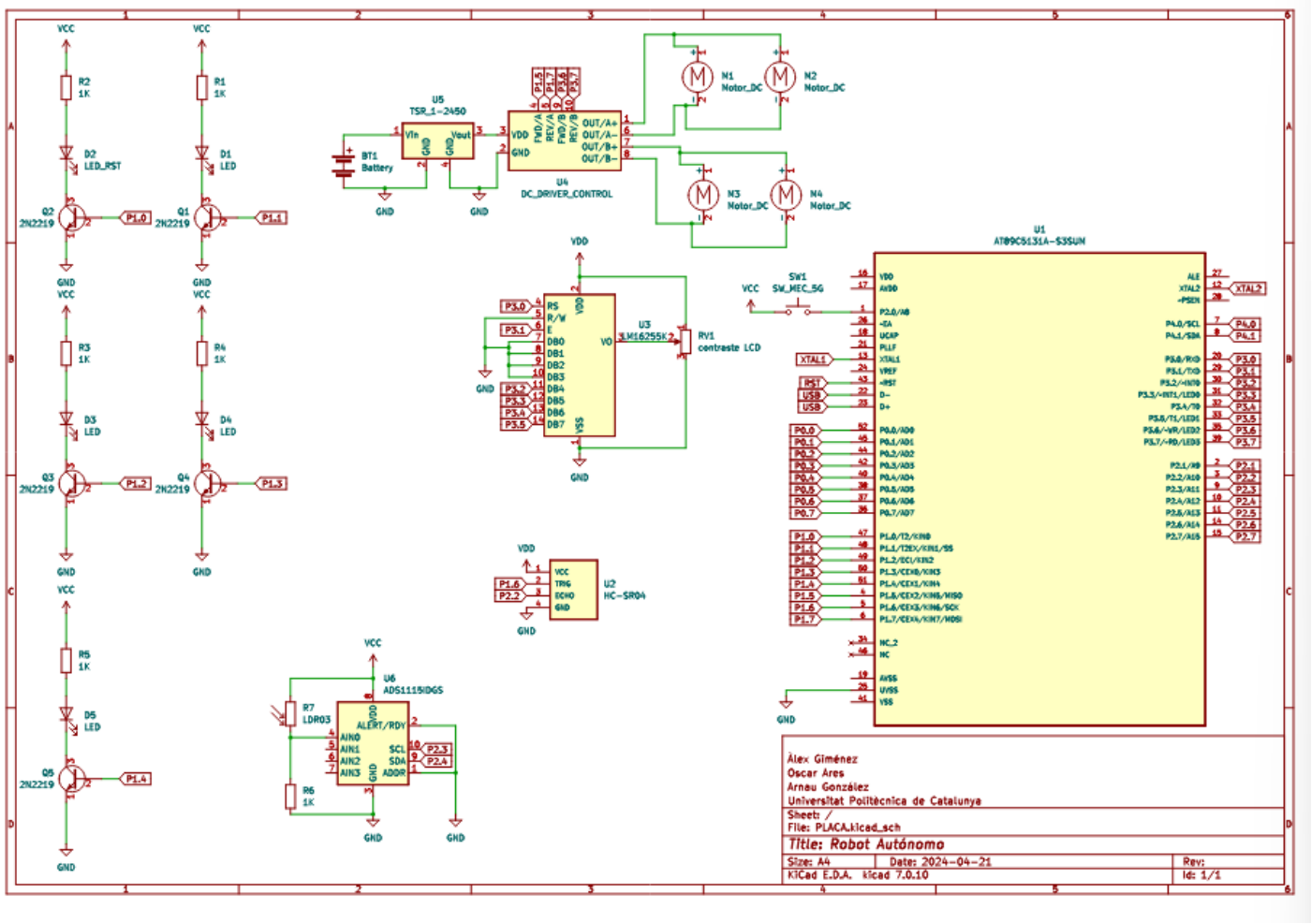


Figura 22: Diseño definitivo

4. CONCLUSIONES

Todos los objetivos de este proyecto se han cumplido con éxito. El principal objetivo era familiarizarnos con todo el proceso y la fabricación de un sistema de medida basado en un microcontrolador, trabajando en equipo y poniendo a prueba nuestros conocimientos de programación adquiridos durante la asignatura.

Decidimos abordar un proyecto bastante ambicioso, ya que nunca habíamos trabajado con este tipo de componentes. Además, era nuestro primer proyecto real, el que más se asemeja a lo que nuestra carrera aspira. La principal dificultad fue la organización. Aunque teníamos una idea general de cómo organizarnos, surgieron varios inconvenientes que prolongaron nuestro trabajo mucho más de lo previsto, desajustando nuestra planificación completamente y obligándonos a improvisar sobre la marcha.

Otro problema imprevisto al inicio fue la dificultad de encontrar los componentes. Mientras que algunos, como el sensor ultrasónico o el LDR, eran fáciles de conseguir, otros tardaron mucho en llegar, especialmente el ADC los cuales contribuyeron al retraso y complicó el progreso del proyecto.

El mayor contratiempo con el que nos enfrentamos fue con la placa. Dos días antes de la exposición, dejó de funcionar, generando mucho estrés en el grupo. Afortunadamente, el profesorado nos proporcionó otra placa, lo que nos permitió finalizar los detalles. Mientras algunos miembros del grupo avanzaban con el proyecto definitivo, otros se encargaban de reparar la placa y finalmente, logramos arreglar la placa y que todo funcionara correctamente.

Lo que más hemos aprendido de este proyecto es que siempre surgen imprevistos y es crucial adaptarse rápidamente y sobre todo debemos dejar un margen para errores en la organización. A pesar de tener una buena planificación semanal, los problemas e inconvenientes generaron mucho estrés, y con un poco más de previsión, podrían haberse solucionado previamente.

En conclusión, creemos que este proyecto ha sido todo un éxito. A pesar de ser un proyecto ambicioso y de tener poco conocimiento previo, logramos que todo funcionara según lo planeado, sin el uso de librerías externas ya que uno de nuestros objetivos no era solo que el proyecto funcionara, sino también entenderlo a fondo.

5. CÓDIGO COMPLETO

```
#include <reg51.h>
#include <stdio.h>
#include <math.h>

// Definir pines para el LCD
sbit RS = P3^0;
sbit EN = P3^1;
sbit D4 = P3^2;
sbit D5 = P3^3;
sbit D6 = P3^4;
sbit D7 = P3^5;

// Bit boton
sbit BOTON = P2^0;

// Definir pines LED
sbit LED1 = P1^1;
sbit LED2 = P1^2;
sbit LED3 = P1^3;
sbit LED4 = P1^4;

// Definir pines ECHO
sbit TRIG_PIN = P1^6; // Pin de disparo (TRIG)
sbit ECHO_PIN = P2^2; // Pin de eco (ECHO)

// Definiciones para pines I²C
sbit SDA = P2^3;  // Pin de datos I²C
sbit SCL = P2^4;  // Pin de reloj I²C

//Definir puertos motor
sbit MOTOR1_FWD = P1 ^ 5;
sbit MOTOR1_BWD = P1 ^ 7;
sbit MOTOR2_FWD = P3 ^ 6;
sbit MOTOR2_BWD = P3 ^ 7;


// Direcciones del ADS1115
#define ADS1115_ADDRESS 0x48  // Dirección I²C del ADS1115

// Registro de configuración del ADS1115
#define CONFIG_REGISTER 0x01
#define CONVERSION_REGISTER 0x00

// Prototipos de funciones
void I2C_Init(void);
void I2C_Start(void);
void I2C_Stop(void);
void I2C_Write(unsigned char);
unsigned char I2C_Read(unsigned char);
void ADS1115_WriteConfig(unsigned int);
unsigned int ADS1115_ReadConversion(void);
void controlar_leds(unsigned int adc_value);

// Función de retardo
void delay(unsigned int);

// Variables flanco ascendente
unsigned int estado_anterior;

// Variable medida distancia
unsigned int distance;
unsigned int adc_value;

// Prototipos de funciones del LCD
void LCD_init();
void LCD_cmd(unsigned char cmd);
void LCD_data(unsigned char datos);
void LCD_write_string(const char *str);

// Prototipos secuencia leds
void encender_leds_secuencialmente();

// Definicion funciones
void trigger_pulse();
void delay_us(unsigned int us);
unsigned int measure_distance();
void int_to_string(unsigned int num, char *str);
```

```
78
79  // Función para inicializar el temporizador 0 en modo 1
80  void timer0_init() {
81      TMOD &= 0xF0; // Limpiar bits de Timer 0
82      TMOD |= 0x01; // Configurar Timer 0 en modo 1 (16 bits)
83  }
84
85  // Función para generar un retardo de 1 microsegundo usando Timer 0
86  void delay_1us() {
87      TH0 = 0xFF;   // Cargar el valor alto para 1 us (ajustado)
88      TL0 = 0xF4;   // Cargar el valor bajo para 1 us (ajustado)
89      TR0 = 1;      // Iniciar el temporizador
90      while (!TF0); // Esperar hasta que se desborde el temporizador
91      TR0 = 0;      // Detener el temporizador
92      TF0 = 0;      // Borrar el flag de desbordamiento del temporizador
93  }
94
95  // Función para generar un retardo de n microsegundos
96  void delay_us(unsigned int us) {
97      unsigned int i;
98      for (i = 0; i < us; i++) {
99          delay_1us();
100     }
101 }
102
103 // Función para generar un pulso de disparo al sensor ultrasónico
104 void trigger_pulse() {
105     TRIG_PIN = 1; // Establecer el pin de disparo en alto
106     delay_us(10); // Esperar un corto periodo de tiempo (al menos 10us)
107     TRIG_PIN = 0; // Apagar el pin de disparo
108 }
109
110 // Función para medir la distancia utilizando el sensor ultrasónico
111 unsigned int measure_distance() {
112     float distance_measurement, valor;
113     unsigned long duration;
114     unsigned int distance_cm;
115
116     trigger_pulse();
117
118     while (!ECHO_PIN); // Esperar hasta que el pin de eco se active (se ponga en alto)
119     TR0 = 1; // Iniciar el temporizador
120
121     while (ECHO_PIN && !TF0); // Esperar hasta que el pin de eco se desactive (se ponga en bajo)
122     TR0 = 0; // Detener el temporizador
123
124     // Calcular la distancia en centímetros
125     valor = 1.085e-6 * 34300;
126     duration = (TL0 | (TH0 << 8)); // Leer el valor del temporizador
127     distance_measurement = duration / 58 ; // Calcular la distancia (ida y vuelta)
128     distance_cm = (unsigned int)distance_measurement;
129
130     return distance_cm;
131 }
132
133 // Convertir entero a string
134 void int_to_string(unsigned int num, char *str) {
135     int i = 0;
136     int temp_num = num;
137
138     // Contar el número de dígitos
139     do {
140         temp_num /= 10;
141         i++;
142     } while (temp_num != 0);
143
144     str[i] = '\0'; // Añadir el carácter nulo al final
145
146     // Convertir cada dígito a carácter
147     while (num != 0) {
148         str[--i] = (num % 10) + '0';
149         num /= 10;
150     }
151
152     if (i == 1) {
153         str[0] = '0';
154     }
```

```
155  }
156
157  void delay_ms(unsigned int ms) {
158      unsigned int i, j;
159      for (i = 0; i < ms; i++)
160          for (j = 0; j < 500; j++);
161  }
162  void motores_adelante() {
163      MOTOR1_FWD = 1;
164      MOTOR1_BWD = 0;
165      MOTOR2_FWD = 1;
166      MOTOR2_BWD = 0;
167  }
168
169  void motores_parado() {
170      MOTOR1_FWD = 0;
171      MOTOR1_BWD = 0;
172      MOTOR2_FWD = 0;
173      MOTOR2_BWD = 0;
174  }
175
176  void motor1_atras() {
177      MOTOR1_FWD = 0;
178      MOTOR1_BWD = 1;
179      MOTOR2_FWD = 1;
180      MOTOR2_BWD = 0;
181  }
182
183  void main() {
184      char distance_str[10]; // Cadena para almacenar la distancia como texto
185      char value_str[10];
186      unsigned int previous_distance = 0xFFFF; // Valor inicial que nunca será igual a una distancia
     medida
187      unsigned int previous_adc_value = 0xFFFF; // Valor inicial que nunca será igual a un valor ADC
     medido
188
189      P1 = 0x00;
190      P3 = 0x00;
191      P2 = 0xFF; // Configurar P2 como entrada
192      estado_anterior = 0;
193
194      timer0_init(); // Inicializar el temporizador
195
196
197
198      I2C_Init();
199
200      // Configurar el ADS1115 (modo continuo, AIN0, PGA ±4.096V, 128SPS)
201      LCD_init(); // Inicializar el LCD
202      LCD_cmd(0x80);
203      LCD_write_string("Hola soy Robo ");
204      encender_leds_secuencialmente();// Encender LEDs uno tras otro con un retardo de 500ms
205      LCD_cmd(0x01); // Limpiar la pantalla antes de escribir
206      delay_ms(500);
207      lcd_init();
208
209      while (1) {
210          // Medir la distancia utilizando el sensor ultrasónico
211          ADS1115_WriteConfig(0x8583);
212          distance = measure_distance();
213          int_to_string(distance, distance_str); // Convertir la distancia a cadena
214          adc_value = ADS1115_ReadConversion();
215          int_to_string(adc_value, value_str);
216
217          if (BOTON == 1) { // Si el interruptor está en posición encendido
218
219                  // Limpiar y actualizar solo la distancia si ha cambiado
220
221                  LCD_cmd(0x80); // Ir al inicio de la primera línea
222                  LCD_write_string("D:         "); // Espacios para limpiar la línea
223                  LCD_cmd(0x80); // Ir al inicio de la primera línea
224                  LCD_write_string("D: "); // Mostrar "Distancia: " en el LCD
225                  LCD_write_string(distance_str); // Mostrar la distancia medida en el LCD
226                  LCD_write_string(" cm"); // Mostrar " cm" en el LCD
227                   // Limpiar y actualizar solo el valor ADC si ha cambiado
228                  LCD_cmd(0xC0); // Mover a la segunda línea del LCD
229                  LCD_write_string("AD:        "); // Espacios para limpiar la línea
```

```
                LCD_cmd(0xC0); // Mover a la segunda línea del LCD
                LCD_write_string("AD: "); // Mostrar "ADC: " en el LCD
                LCD_write_string(value_str);

                previous_adc_value = adc_value;
                previous_distance = distance;
                delay_ms (50);
                if (distance > 15) {
                        motores_adelante();
                        } else  {
                            motores_parado();
                            delay_ms(50);
                            motor1_atras();
                            delay_ms(50);
                        }

                controlar_leds(adc_value);
            estado_anterior = 1;
        } else {

            lcd_init();
            LCD_write_string("AD: ");    // Mostrar "ADC: " en el LCD
            LCD_write_string(value_str);
            controlar_leds(adc_value);
            delay_ms(50);
            motores_parado();

            estado_anterior = 0;
        }

        // Esperar un periodo de tiempo antes de realizar la próxima medición

        delay_ms(10);
    }
}

// Inicialización del LCD
void LCD_init() {
    delay_ms(10); // Espera de inicialización
    LCD_cmd(0x02); // Modo de 4 bits
    LCD_cmd(0x28); // 2 líneas, 5x8 matriz de puntos
    LCD_cmd(0x0E); // Mostrar cursor
    LCD_cmd(0x06); // Incrementar el cursor
    LCD_cmd(0x01); // Limpiar pantalla
    delay_ms(10); // Espera de limpieza de pantalla
}

// Enviar comando al LCD
void LCD_cmd(unsigned char cmd)
{
    RS = 0; // Modo comando
    D4 = (cmd >> 4) & 0x01;
    D5 = (cmd >> 4) & 0x02;
    D6 = (cmd >> 4) & 0x04;
    D7 = (cmd >> 4) & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_us(1);
    D4 = cmd & 0x01;
    D5 = cmd & 0x02;
    D6 = cmd & 0x04;
    D7 = cmd & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_ms(2); // Espera para comando
}

// Enviar datos al LCD
void LCD_data(unsigned char datos)
{
    RS = 1; // Modo datos
    D4 = (datos >> 4) & 0x01;
    D5 = (datos >> 4) & 0x02;
    D6 = (datos >> 4) & 0x04;
    D7 = (datos >> 4) & 0x08;
```

```
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_us(1);
    D4 = datos & 0x01;
    D5 = datos & 0x02;
    D6 = datos & 0x04;
    D7 = datos & 0x08;
    EN = 1;
    delay_us(1);
    EN = 0;
    delay_ms(2); // Espera para datos
}

// Función para escribir una cadena en el LCD
void LCD_write_string(const char *str) {
    while (*str) {
        LCD_data(*str++);
    }
}
void I2C_Init(void) {
    SDA = 1;
    SCL = 1;
}

void I2C_Start(void) {
    SDA = 1;
    SCL = 1;
    SDA = 0;
    SCL = 0;
}

void I2C_Stop(void) {
    SCL = 0;
    SDA = 0;
    SCL = 1;
    SDA = 1;
}

void I2C_Write(unsigned char dat) {
    unsigned char i;
    for (i = 0; i < 8; i++) {
        SDA = (dat & 0x80) >> 7;
        SCL = 1;
        dat <<= 1;
        SCL = 0;
    }
    SDA = 1;  // Liberar línea de datos para ACK
    SCL = 1;
    SCL = 0;
}

unsigned char I2C_Read(unsigned char ack) {
    unsigned char i, dat = 0;
    SDA = 1;  // Liberar línea de datos para entrada
    for (i = 0; i < 8; i++) {
        dat <<= 1;
        SCL = 1;
        dat |= SDA;
        SCL = 0;
    }
    SDA = ack ? 0 : 1;  // ACK o NACK
    SCL = 1;
    SCL = 0;
    return dat;
}

void ADS1115_WriteConfig(unsigned int config) {
    I2C_Start();
    I2C_Write((ADS1115_ADDRESS << 1) | 0);  // Dirección + bit de escritura
    I2C_Write(CONFIG_REGISTER);  // Registro de configuración
    I2C_Write((config >> 8) & 0xFF);  // Byte alto de la configuración
    I2C_Write(config & 0xFF);  // Byte bajo de la configuración
    I2C_Stop();
}

unsigned int ADS1115_ReadConversion(void) {
```

```
    unsigned int value;
    I2C_Start();
    I2C_Write((ADS1115_ADDRESS << 1) | 0);  // Dirección + bit de escritura
    I2C_Write(CONVERSION_REGISTER);  // Registro de conversión
    I2C_Start();
    I2C_Write((ADS1115_ADDRESS << 1) | 1);  // Dirección + bit de lectura
    value = ((unsigned int)I2C_Read(1) << 8);  // Leer byte alto
    value |= I2C_Read(0);  // Leer byte bajo
    I2C_Stop();
    return value;
    adc_value = value;
}

void delay(unsigned int count) {
    unsigned int i, j;
    for (i = 0; i < count; i++)
        for (j = 0; j < 1275; j++);
}
// Función para encender LEDs uno tras otro con un retardo de 500ms
void encender_leds_secuencialmente() {

        LED1 = 1;  // Encender LED1
        delay_ms(50); // Esperar 500ms
        LED1 = 0;  // Apagar LED1
        LED2 = 1;  // Encender LED2
        delay_ms(50); // Esperar 500ms
        LED2 = 0;  // Apagar LED2
        LED3 = 1;  // Encender LED3
        delay_ms(50); // Esperar 500ms
        LED3 = 0;  // Apagar LED3
        LED4 = 1;  // Encender LED4
        delay_ms(50); // Esperar 500ms
        LED4 = 0;  // Apagar LED4
        delay_ms(50); // Esperar 500ms
        LED4 = 1;  // Encender LED4
        delay_ms(50); // Esperar 500ms
        LED4 = 0;  // Apagar LED4
        LED3 = 1;  // Encender LED3
        delay_ms(50); // Esperar 500ms
        LED3 = 0;  // Apagar LED3
        LED2 = 1;  // Encender LED2
        delay_ms(50); // Esperar 500ms
        LED2 = 0;  // Apagar LED2
        LED1 = 1;  // Encender LED1
        delay_ms(50); // Esperar 500ms
        LED1 = 0;  // Apagar LED1


}
void controlar_leds(unsigned int adc_value) {
    // Apagar todos los LEDs inicialmente
    LED1 = 0;
    LED2 = 0;
    LED3 = 0;
    LED4 = 0;

    // Encender los LEDs según el valor del ADC
    if (adc_value < 1024) {
        LED1 = 1;
        LED2 = 1;
        LED3 = 1;
        LED4 = 1;
    } else if (adc_value < 2048) {
        LED1 = 0;
        LED2 = 1;
        LED3 = 1;
        LED4 = 0;
    } else if (adc_value < 3072) {
        LED1 = 1;
        LED2 = 0;
        LED3 = 0;
        LED4 = 1;

    } else {
        LED1 = 0;
        LED2 = 0;
        LED3 = 0;
       LED4 = 0;

    }
}
```

6. BIBLIOGRAFÍA

[1] Admin. (2022, 29 junio). 8051 I2C Interfacing Tutorial (EEPROM Interfacing). EmbeTronicX. https://embetronicx.com/tutorials/microcontrollers/8051/8051-i2c-interfacing-tutorial-eeprom/

[2] A7.8051 Interfacing ADC: LDR. (s. f.). Tutorials. https://exploreembedded.com/wiki/index.php?title=A7.8051%20Interfacing%20ADC:%20LDR%20&%20LM35

[3] eantec.ES. (2019, 9 agosto). Robot autónomo esquiva objetos | Leantec.ES. Leantec.ES. https://leantec.es/robot-autonomo-esquiva-objetos/

[4] Bala, M. (2015, 8 agosto). Ultrasonic distance meter using HC-SR04 and 8051. Gadgetronicx. https://www.gadgetronicx.com/ultrasonic-distance-meter-hcsr04-8051/

[5] Tema 4 - Programación en C51. S. Tornil. UPC EEBE. Barcelona, 2024

[6] Tema 5 - Puertos y dispositivos externos. S. Tornil. UPC EEBE. Barcelona, 2024

[7] Tema 6 - Interrupciones y temporizadores. S. Tornil. UPC EEBE. Barcelona, 2024

[8] AT89C5131 Datasheet. Atmel.

[9] I2C Communication Protocol: Understanding I2C Primer, PMBus, and SMBus | Analog Devices. (s. f.). https://www.analog.com/en/resources/analog-dialogue/articles/i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html

[10] Learning about Electronics. (n.d.). How to Set Up an LDR Circuit. https://www.learningaboutelectronics.com