FUNDACIÓN PARA LA EDUCACIÓN SUPERIOR SAN MATEO

MATERIA: CIRCUITOS

Docente: Ing. Andres Felipe Velasquez Monroy

PROYECTO ROBOT SEGUIDOR DE PARED:

“WallSRaider”

- Erick Salamanca Ramírez.

BOGOTA D.C. 14 DE MAYO DE 2017

TABLA DE CONTENIDO

1. INTRODUCCIÓN…..……..1

2. OBJETIVO Y REQUERIMIENTOS..……..…….…...2

3. SOLUCIÓN……..…...…...…….....3

3.1. ABORDAJE DEL PROBLEMA……...…….....3

3.2. COMPONENTES Y MATERIALES A USAR...…….....3

4. EJECUCIÓN....…….....…….……...9

4.1. A NIVEL HARDWARE....…….....……..……..9

4.1.2. ESQUEMAS Y MEDICIONES DEL CIRCUITO.....10

4.2. A NIVEL SOFTWARE....…...……..…..……11

5. BIBLIOGRAFÍA ……..……...……17

INTRODUCCIÓN

Hoy día el hardware y software libre nos ha permitido enfrentarnos a un cambio de paradigma y la apertura a una nueva gama de posibilidades productivas y de innovación, que día a día cobran más fuerza en todos los ámbitos de la sociedad.

Nos llena de gratitud y admiración aquellos ingenieros que comparten sus creaciones a favor de la mejora continua, el diseño y el crecimiento de más ingenieros compartiendo sus creaciones, inventos y códigos de programación y modelos de circuitos electrónicos, más resumidamente, aquellas personas que trabajan en pro de los proyectos de hardware/software.

Esta vez nos referimos a Arduino, una compañía de hardware libre y una comunidad tecnológica que diseña y manufactura motherboards[1] de desarrollo de hardware y software con entorno de desarrollo interactivo[2], en donde se programa cada placa.

Toda la plataforma, tanto para sus componentes de hardware como de software, tiene licencia de código abierto[3] que permite libertad de acceso a ellos.

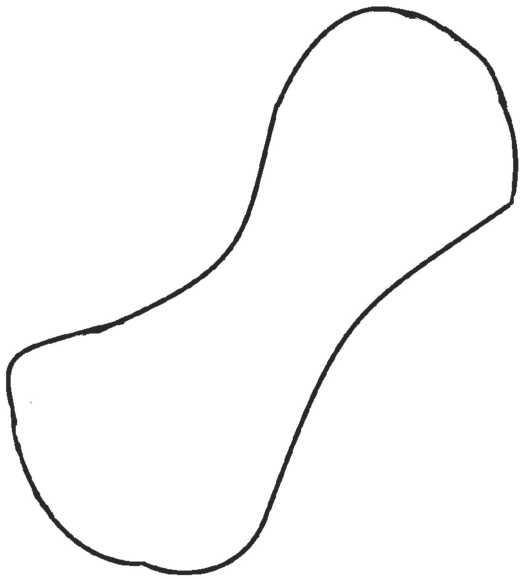
El software consiste en un entorno de desarrollo[4] basado en el entorno de Procesing[5] (basado en java) y lenguaje de programación basado en Wiring[6] (basado en java y C) y el bootloader es ejecutado desde la placa[7].

"EL HARDWARE ABIERTO SIGNIFICA TENER LA POSIBILIDAD DE MIRAR LO QUE HAY DENTRO DE LAS COSAS, QUE ESO SEA ÉTICAMENTE CORRECTO, Y QUE PERMITA MEJORAR LA EDUCACIÓN. EDUCAR EN CÓMO FUNCIONAN LAS COSAS…"

- David Cuartielles, cofundador de Arduino.

OBJETIVO Y REQUERIMIENTOS

Se debe crear un robot, que tenga la capacidad de seguir la pared de una pista hecha en forma del contorno de número “ocho”, el robot debe mantener siempre una trayectoria paralela a la pared.



>Esquema aproximado de la pista que se usará.

SOLUCIÓN

ABORDAJE DEL PROBLEMA:

Se plantea la idea de elaborar un Robot de investigación, el cual tendrá como modo de transportarse ruedas, las cuales se moverán con motores/motorreductores, tendrá un sensor que detecte la distancia de la pared, la modalidad de sensor[8] será un sensor/emisor de ping[9] de ondas ultrasónicas[10] el cual irá conectado a un centro de procesamiento (Arduino versión uno), el cual dará la instrucción a los motores de moverse según qué tan lejos esté el artefacto de la pared de la pista.

COMPONENTES Y MATERIALES A USAR

- Placa de arduino uno
- Báquelita universal de 9 cm por 15 cm
- Sensor ultrasónico de proximidad
- Cable USB tipo b
- Cables para protoboard
- Módulo relay para arduino
- Cautín, estaño, pasta para soldar
- Protoboard
- Motor y motoreductor
- Placa de arduino uno:

Placa de arduino uno:

Microcontrolador:ATmega328 - x8 @16 MHz

Voltage: 5V

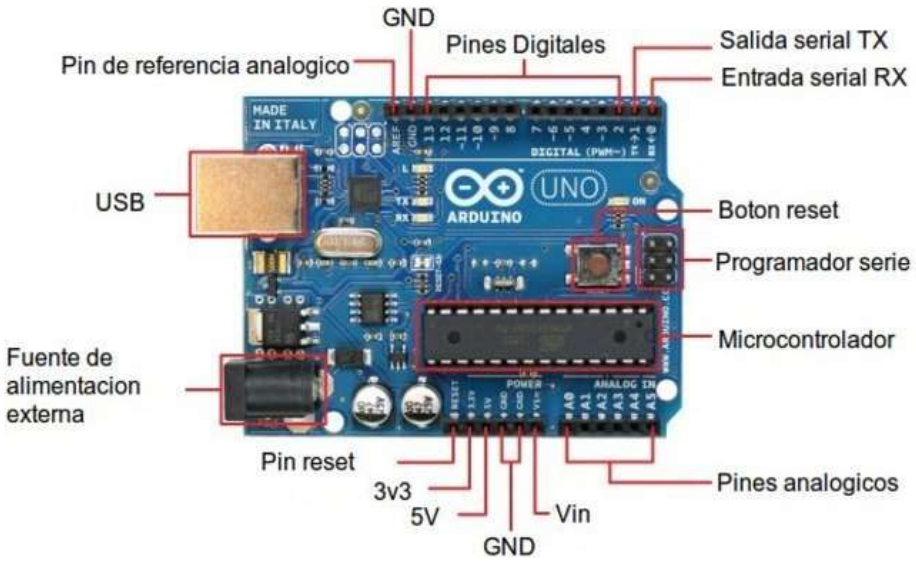
Voltaje entrada: 6-12V

Pines digitales: 14 (de los cuales 6 son salida PWM)

Memoria Flash: 32 KB de los cuales 0.5 KB son utilizados para el arranque

SRAM (RAM estática) : 2 KB

EEPROM (RAM no volátil): 1 KB



Cable usb tipo B

Sensor de proximidad ultrasónico.

Alimentación de 5 volts

Interfaz sencilla: Solamente 4 hilos Vcc,

Trigger, Echo, GND

Rango de medición: 2 cm a 400 cm

Corriente de alimentación: 15 mA

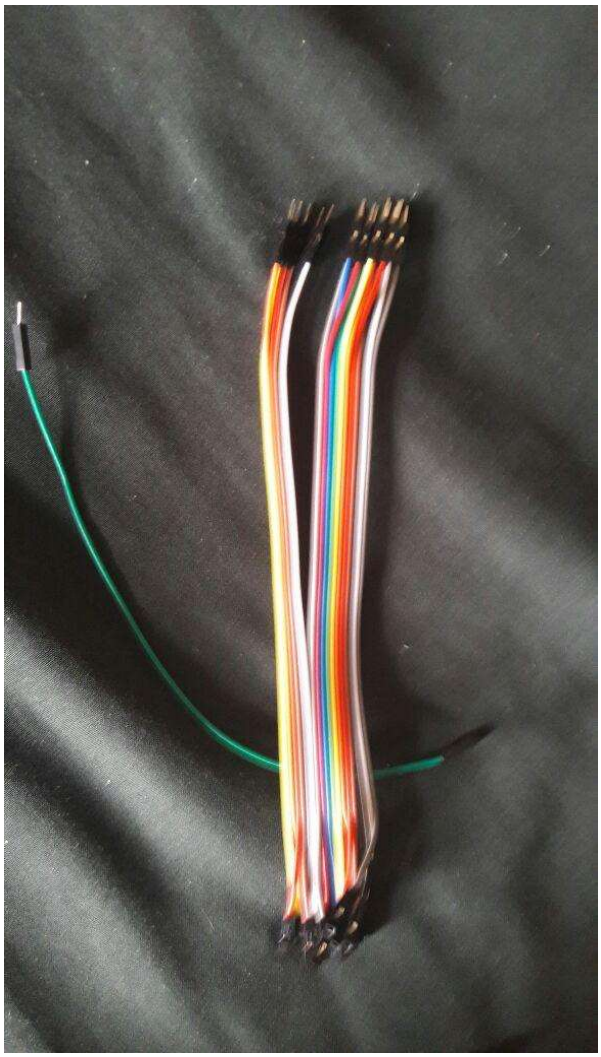
Frecuencia del pulso: 40 Khz

Apertura del pulso ultrasónico: 15º

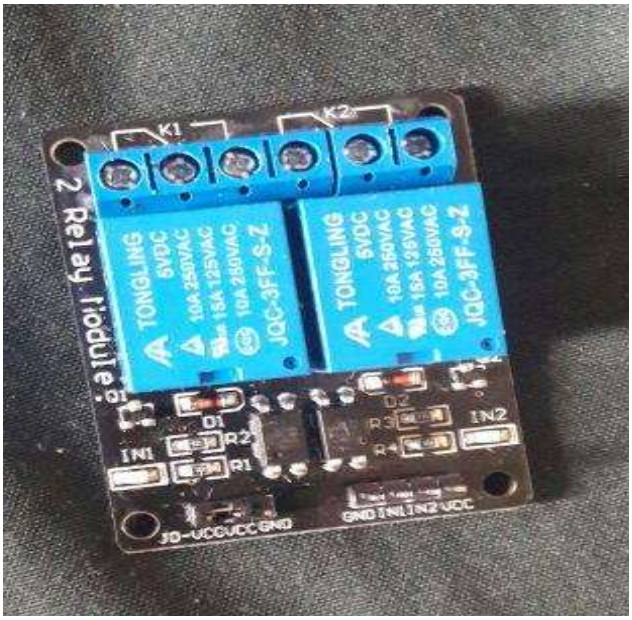
Señal de disparo: 10uS

Dimensiones del módulo: 45x20x15 mm.

Cables para protoboard



Módulo relay para arduino.



Especificaciones del módulo relay:

Tensión de alimentación: 5V

Carga: 250V 10A o 30V 10A

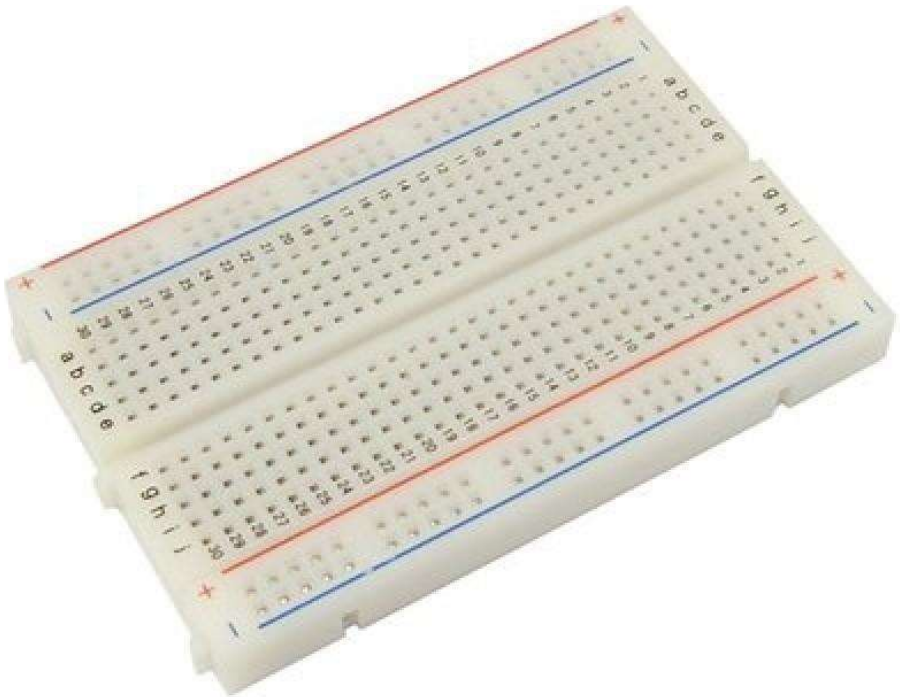
VCC: alimentación del sistema positivo,

GND: cátodo fuente de alimentación del sistema;

Puerto de control del relé IN1 IN2

Cautín, estaño y pasta para soldadura

Protoboard



Motor motorreductor

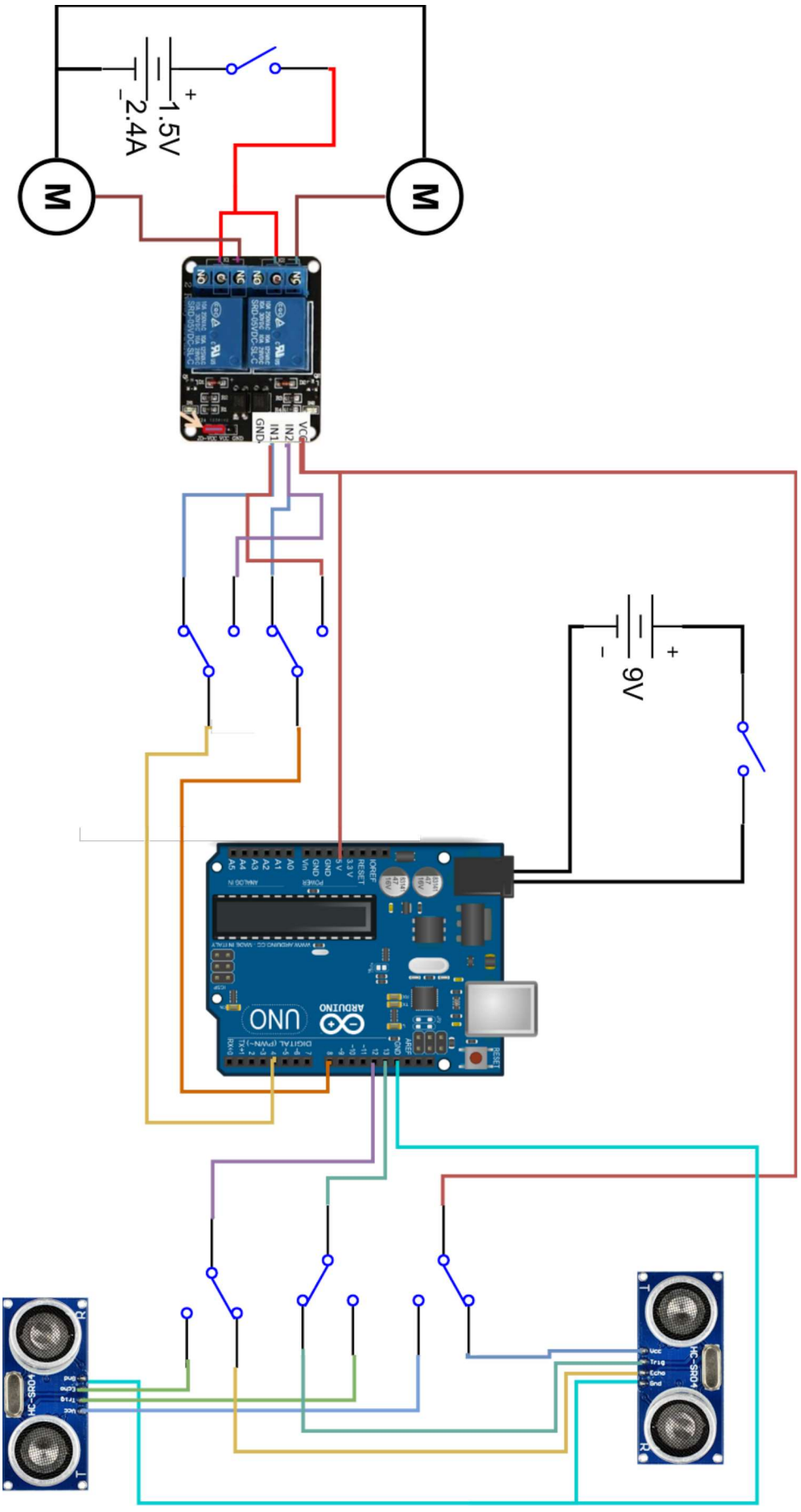
EJECUCIÓN

A NIVEL HARDWARE:

Luego de hacerse el mapa mental de que el modo de funcionamiento será; los dos motores funcionan al tiempo dando movilidad en línea recta al carro, si se corta la alimentación al motor derecho, el motor para, esto haciendo que todo el carro de giro a la derecha, por ende si se quita la alimentación al motor izquierdo el carro dará vuelta a la izquierda, para ejecutar esta función automáticamente, usaremos un relé, este relé dependerá de las instrucciones que el arduino, a su vez el Arduino sabrá a que distancia esta de la pared usando 2 módulos de ping de ultrasonido (HC-SR04) que medirá la distancia enviando ondas y detectando en cuanto vuelven (más o menos la misma funcionalidad que tiene un ping en red), se usarán switches para conmutar las conexiones entre el Arduino y los dos módulos de ultrasonido, es decir si necesita que al arduino entre las señales del módulo que se encuentra al costado derecho, los switch (3) se cambiaran de posición manualmente a la derecha y así en el caso contrario que se necesite la señal izquierda

En la práctica se detecta que, como se cambia la señal de la lugar, la señal a los motores sigue siendo la misma, por ejemplo la señal que llega desde la izquierda, cuando está muy cerca a la pared el arduino enviará instrucción al motor derecho que se se detenga, esto haciendo que el carro siga girando a la izquierda, este problema lo abordamos de igual manera con switches , conmutando la señal que llega del Arduino al relé, de esta forma, agregando el código en el Arduino, en teoría el robot queda funcional.

ESQUEMAS Y MEDICIONES DEL CIRCUITO:


2.4A
1.5V
M
M
9V
UNO
ARDUINO
HC-SR04
HC-SR04

A NIVEL SOFTWARE:

Se utilizó una librería NewPing, redactada por la comunidad de Arduino forum.arduino.cc/,

```
#ifndef NewPing_h
#define NewPing_h

#if defined (ARDUINO) && ARDUINO >= 100
 #include <Arduino.h>
#else
  #include <WProgram.h>
  #include <pins_arduino.h>
#endif

#if defined (__AVR__)
  #include <avr/io.h>
  #include <avr/interrupt.h>
#endif

// Shouldn't need to change these values unless you have a specific need to do so.
#define MAX_SENSOR_DISTANCE 500 // Maximum sensor distance can be as high as 500cm, no reason to wait for ping longer than sound takes to travel this distance and back. Default=500
#define US_ROUNDTRIP_CM 57      // Microseconds (uS) it takes sound to travel round-trip 1cm (2cm total), uses integer to save compiled code space. Default=57
#define US_ROUNDTRIP_IN 146     // Microseconds (uS) it takes sound to travel round-trip 1 inch (2 inches total), uses integer to save compiled code space. Defalult=146
#define ONE_PIN_ENABLED true    // Set to "false" to disable one pin mode which saves around 14-26 bytes of binary size. Default=true
```

```
#define ROUNDING_ENABLED false  // Set to "true" to enable distance rounding which also adds 64 bytes to binary size. Default=false
#define URM37_ENABLED false     // Set to "true" to enable support for the URM37 sensor in PWM mode. Default=false
#define TIMER_ENABLED true      // Set to "false" to disable the timer ISR (if getting "__vector_7" compile errors set this to false). Default=true

// Probably shouldn't change these values unless you really know what you're doing.
#define NO_ECHO 0               // Value returned if there's no ping echo within the specified MAX_SENSOR_DISTANCE or max_cm_distance. Default=0
#define MAX_SENSOR_DELAY 5800   // Maximum uS it takes for sensor to start the ping. Default=5800
#define ECHO_TIMER_FREQ 24      // Frequency to check for a ping echo (every 24uS is about 0.4cm accuracy). Default=24
#define PING_MEDIAN_DELAY 29000 // Microsecond delay between pings in the ping_median method. Default=29000
#define PING_OVERHEAD 5         // Ping overhead in microseconds (uS). Default=5
#define PING_TIMER_OVERHEAD 13  // Ping timer overhead in microseconds (uS). Default=13
#if URM37_ENABLED == true
  #undef  US_ROUNDTRIP_CM
  #undef  US_ROUNDTRIP_IN
  #define US_ROUNDTRIP_CM 50  // Every 50uS PWM signal is low indicates 1cm distance. Default=50
  #define US_ROUNDTRIP_IN 127 // If 50uS is 1cm, 1 inch would be 127uS (50 x 2.54 = 127). Default=127
#endif

// Conversion from uS to distance (round result to nearest cm or inch).
#define NewPingConvert(echoTime, conversionFactor) (max(((unsigned int)echoTime + conversionFactor / 2) / conversionFactor, (echoTime ? 1 : 0)))
```

```
// Detect non-AVR microcontrollers (Teensy 3.x, Arduino DUE, etc.) and don't use port registers or timer interrupts as required.
#if (defined (__arm__) && defined (TEENSYDUINO))
  #undef  PING_OVERHEAD
  #define PING_OVERHEAD 1
  #undef  PING_TIMER_OVERHEAD
  #define PING_TIMER_OVERHEAD 1
  #define DO_BITWISE true
#elif !defined (__AVR__)
  #undef  PING_OVERHEAD
  #define PING_OVERHEAD 1
  #undef  PING_TIMER_OVERHEAD
  #define PING_TIMER_OVERHEAD 1
  #undef  TIMER_ENABLED
  #define TIMER_ENABLED false
  #define DO_BITWISE false
#else
  #define DO_BITWISE true
#endif

// Disable the timer interrupts when using ATmega128 and all ATtiny microcontrollers.
#if defined (__AVR_ATmega128__) || defined (__AVR_ATtiny24__) || defined (__AVR_ATtiny44__) || defined (__AVR_ATtiny84__) || defined (__AVR_ATtiny25__) || defined (__AVR_ATtiny45__) || defined (__AVR_ATtiny85__) || defined (__AVR_ATtiny261__) || defined (__AVR_ATtiny461__) || defined (__AVR_ATtiny861__) || defined (__AVR_ATtiny43U__)
  #undef  TIMER_ENABLED
```

```
  #define TIMER_ENABLED false
#endif

// Define timers when using ATmega8, ATmega16, ATmega32 and ATmega8535 microcontrollers.
#if defined (__AVR_ATmega8__) || defined (__AVR_ATmega16__) || defined (__AVR_ATmega32__) ||
defined (__AVR_ATmega8535__)
  #define OCR2A OCR2
  #define TIMSK2 TIMSK
  #define OCIE2A OCIE2
#endif

class NewPing {
  public:
    NewPing(uint8_t trigger_pin, uint8_t echo_pin, unsigned int max_cm_distance =
MAX_SENSOR_DISTANCE);
    unsigned int ping(unsigned int max_cm_distance = 0);
    unsigned long ping_cm(unsigned int max_cm_distance = 0);
    unsigned long ping_in(unsigned int max_cm_distance = 0);
    unsigned long ping_median(uint8_t it = 5, unsigned int max_cm_distance = 0);
    static unsigned int convert_cm(unsigned int echoTime);
    static unsigned int convert_in(unsigned int echoTime);
#if TIMER_ENABLED == true
    void ping_timer(void (*userFunc)(void), unsigned int max_cm_distance = 0);
    boolean check_timer();
    unsigned long ping_result;
    static void timer_us(unsigned int frequency, void (*userFunc)(void));
    static void timer_ms(unsigned long frequency, void (*userFunc)(void));
```

```
    static void timer_stop();
#endif
  private:
    boolean ping_trigger();
    void set_max_distance(unsigned int max_cm_distance);
#if TIMER_ENABLED == true
    boolean ping_trigger_timer(unsigned int trigger_delay);
    boolean ping_wait_timer();
    static void timer_setup();
    static void timer_ms_cntdwn();
#endif
#if DO_BITWISE == true
    uint8_t _triggerBit;
    uint8_t _echoBit;
    volatile uint8_t *_triggerOutput;
    volatile uint8_t *_echoInput;
    volatile uint8_t *_triggerMode;
#else
    uint8_t _triggerPin;
    uint8_t _echoPin;
#endif
    unsigned int _maxEchoTime;
    unsigned long _max_time;
};
#endif
```

Luego procedemos a redactar el código con los manuales de Arduino (bibliografía)

```
#include <NewPing.h>
#define trigPin 13
#define echoPin 12
#define MotorI 7
#define MotorD 8
    void setup()
      {   Serial.begin (9600);
          pinMode(trigPin, OUTPUT);
          pinMode(echoPin, INPUT);
          pinMode(MotorI, OUTPUT);
          pinMode(MotorD, OUTPUT);
      }
    void loop()
      {   long duracion, distancia ;
          digitalWrite(trigPin, LOW);   // Nos aseguramos de que el trigger está desactivado
          delayMicroseconds(2);   // Para asegurarnos de que el trigger esta LOW
          digitalWrite(trigPin, HIGH);  // Activamos el pulso de salida
          delayMicroseconds(10); // Esperamos 10µs. El pulso sigue active este tiempo
          digitalWrite(trigPin, LOW);       // Cortamos el pulso y a esperar el echo
          duracion = pulseIn(echoPin, HIGH) ;
          distancia = duracion / 2 / 29.1  ;
          Serial.println(String(distancia) + " cm.") ;
```

```
    int Limite = 6 ;
    int Limite2 = 7 ;
    if ( distancia < Limite)
        digitalWrite ( MotorI , LOW) ;
    else
        digitalWrite( MotorI , HIGH) ;
            delay (500) ;  // Para limitar el número de mediciones
    if ( distancia > Limite2)
        digitalWrite ( MotorD , LOW) ;
    else
        digitalWrite( MotorD , HIGH) ;
    delay (500) ; // Para limitar el número de mediciones
}
```

Bibliografía

forum.arduino.cc/

www.arduino.cc/en/Guide/ArduinoUno

http://dfists.ua.es/~jpomares/arduino/page_01.htm

https://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf

(manual principal)