Escuela Universitaria Politécnica
La Almunia de Doña Godina
Zaragoza

PROYECTO:

PRÁCTICA 2:
SENSOR DE ULTRASONIDOS

REALIZADO POR:

PONS VILLANUEVA, SERGIO

JUNIO-2014

INDICE

1. Objetivos

El **objetivo principal** de esta práctica es el montaje de un sensor de ultrasonidos, realizando el montaje sus componentes y diseñando el código de funcionamiento. Para ello tendremos que realizar los siguientes hitos.

El **primer objetivo e hito** es aprender a montar el hardware y verificar el prototipo guiándonos del esquema eléctrico que veremos en el marco teórico para saber que componentes necesitaremos y así poder realizar su montaje. Las condiciones de verificación también las podremos ver en el apartado teórico.

El **segundo objetivo** es la obtención de un diagrama UML que nos facilite la comprensión del funcionamiento del programa. Este objetivo se consigue al realizar el **segundo hito**, que consiste en realizar el diagrama de actividad del programa.

El **tercer objetivo** es la comprensión de cómo funciona, al menos una parte, a nivel bajo nuestro microcontrolador. Para ello cumpliremos el **tercer hito** dado, en el que tendremos que realizar un código que nos permita enviar un BEACON ultrasónico.

El **cuarto objetivo** nos obliga a entender el esquema del microcontrolador que estamos utilizando, en nuestro caso Arduino Uno, ya que es necesario diferenciar entre interrupción externa y pin digital.

Esto se consigue al realizar el **cuarto hito,** que consiste en obtener la lectura de la interrupción externa

Con este hito también cumplimos otro **objetivo (quinto),** que es la profundización en la programación, en concreto de Arduino, ya que tendremos que utilizar funciones que no habíamos utilizado nunca**.**

Funciones como “attachInterrupt”, “Timer1”, “noInterrupts”…

El **sexto objetivo** es la utilización de conocimientos de otras asignaturas, como instrumentación electrónica, ya que nos son necesarias para realizar algunos cálculos. Este objetivo engloba los siguientes hitos: **quinto hito** es la obtención del tiempo de vuelo ; el **sexto hito** realizaremos una compensación de la Temperatura y el **octavo objetivo** estableceremos un filtro digital de la temperatura.

El **séptimo objetivo** es la profundización en el conocimiento de nuestro microcontrolador al igual que de nuestro conocimiento de programación ya que en el **séptimo hito** tenemos que establecer un auto-cero y guardarlo en la EEPROM.

Para realizar el hito anteriormente dicho, tenemos que profundizar nuestro conocimiento sobre punteros.

El **octavo objetivo** implica el conocimiento para poder comunicar nuestro microcontrolador con otro dispositivo, en nuestro caso un ordenador para graficar los datos, que es lo que nos indicaba el **último hito propuesto.**

Otro **objetivo** es utilizar el ingenio para mejorar nuestro programa incluyendo **hitos** que **no** estaban **especificados**, como el de borrado del auto-cero o una función filtro que funcione a modo de subrutina tanto para la distancia como para la temperatura.

2. Listado de equipos, materiales y software utilizado.

- **Protoboard**.
- **Arduino Uno** (Microcontrolador)
    - Cable ubs-printer.
- **Escudo Arduino:**
    - Resistencias [Ω]: **1x330**, **3x1k1**, **2x1.5k, 1x33k, 1x39k, 1x68k**
    - Condensadores: **4x1µF, 1x10µF**
    - **Cables**.
    - Diodos: **2x1n4148**
    - Circuito integrados: **LM311P, MC3303P, LM35 y MAX232N**.
    - **Transductores: S, R.**
    - Estaño.
    - Soldador.
    - "Chupón". (Para quitar el estaño).
    - Botón.
- Fuente de alimentación: **Generador de tensión**.
- **Generador de señales**.
- **Polímetro.**
- **Osciloscopio.**
    - **Sondas.**
- Software:
    - **Meguno-Link.**
    - **Arduino 1.0.5**.

3. Fundamento teórico

NOTA: A excepción de que se indique lo contrario, toda la información has sido obtenida de la referencia 2 (ver apartado de referencias).

Introducción

Un ultrasonido hace referencia a las frecuencias por encima de los 20kHz, que es el límite del sonido audible.

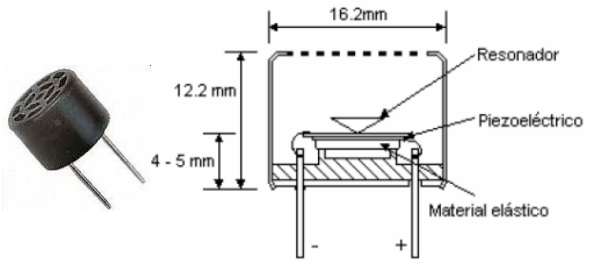
La generación y lectura de un ultrasonido se hace a través de unidades piezoeléctricas.

El ultrasonido es aplicado comúnmente en detectores de movimiento, medidores de distancia, diagnostico medico, limpieza, pruebas no destructivas (para detectar imperfecciones en materiales), soldadura entre otras más.

Transductor

Sus características son:

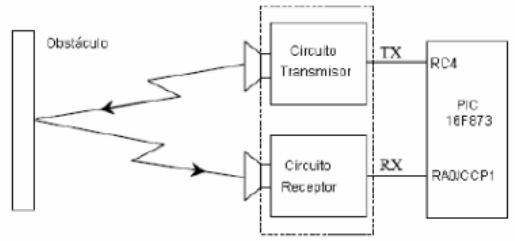
- Frecuencia de resonancia: 40kHz.
- Nivel de Presión Sonora: 115dB<
- Sensitividad: -64dB<
- Máxima entrada de voltaje: 20Vrms
- Directividad típica: 55º



Medida de distancia por ultrasonidos

Para calcular la distancia tendremos que saber el tiempo que tarda la onda en ir hasta el obstáculo y volver. Este tiempo será proporcional a la distancia recorrida por dicha onda, por lo que seremos capaces de medir la distancia.

Por otra parte, también tenemos que tener en cuenta la variable de la velocidad del sonido, ya que aunque esta es de 343m/s a 20ºC, se verá modificada en función de la temperatura que tengamos.

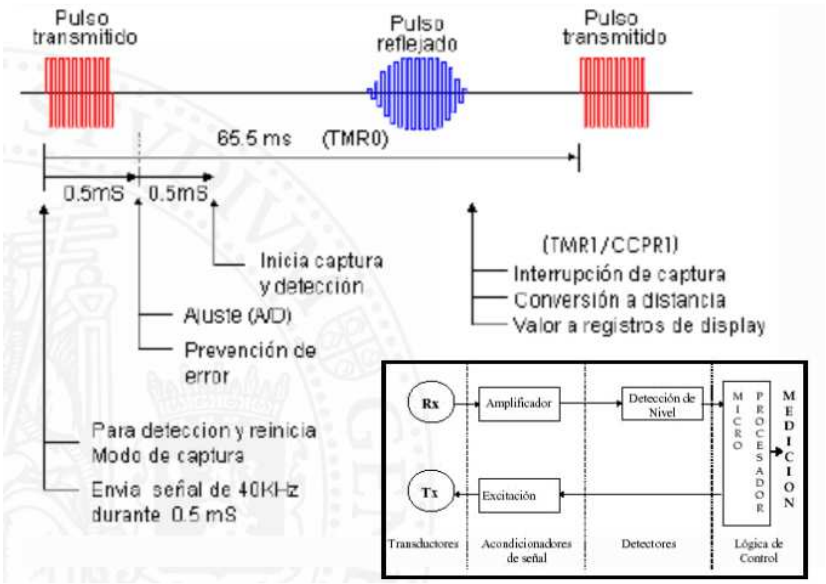


Limitaciones para la medida

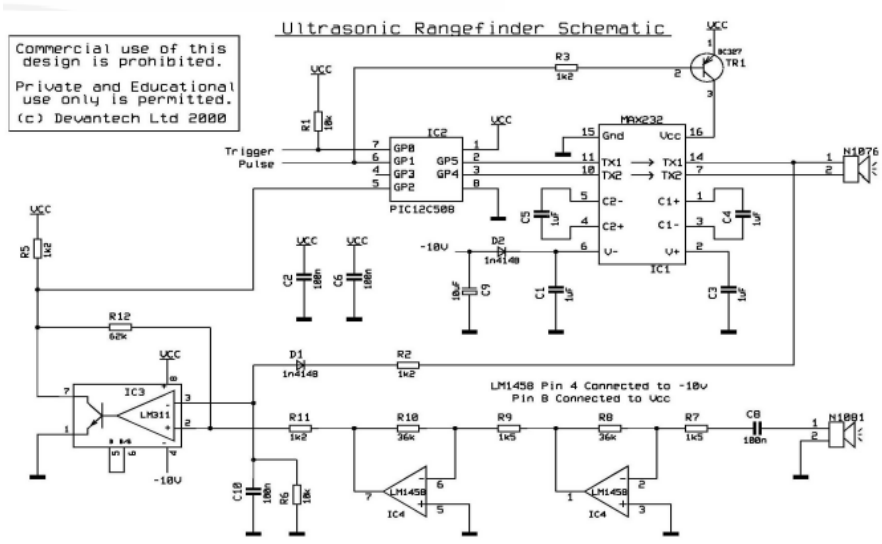
Como la mayoría de los inventos, este diseño tiene sus limitaciones, la cuales tenemos que tener muy en cuenta:

- El objeto debe estar perpendicular al medidor.
- La superficie del objeto debe ser plana
- No debe haber objetos alrededor que puedan hacer una reflexión.
- El objeto no debe ser muy absorbente, como por ejemplo tela o una pared corrugada.

Pulso ultrasónico



Esquema eléctrico

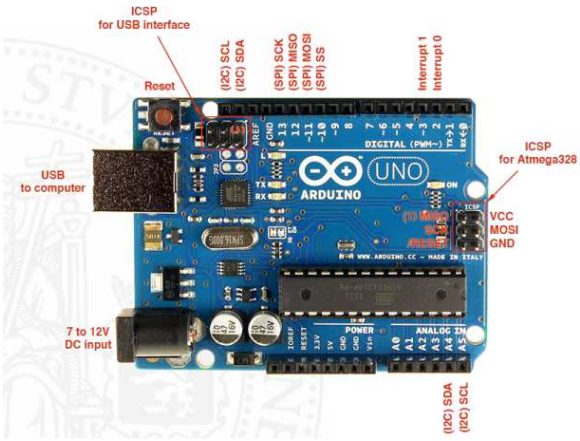


NOTA: También lo podemos ver en el apartado de anexos.

Arduino

Es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. El lenguaje de programación está basado en "*Wiring*" y su entorno de desarrollo en "*Processing*".

Hardware



Podemos ver la siguiente información en el apartado de anexos:

- Esquema general.
- Hardware USB.
- Fuente de alimentación.
- Microcontrolador.
- MCU para el escudo de Arduino.

NOTA: Podemos crear nuestro propio Arduino y descargar su software desde su página oficial http://www.arduino.cc/es/ , además de buscar información relacionada con él.

Cálculo de ganancias (1)

Para calcular la ganancia de nuestros amplificadores tenemos que aplicar la siguiente formula, donde R2 es la resistencia de mayor valor y R1 la de menor valor:

$$G = \frac{R2}{R1}$$

Como tenemos dos amplificadores iguales en serie la ganancia total sería:

$$G_T = G_1 \cdot G_2$$

Filtro exponencial

Para introducir el filtro, lo haremos en el código, hemos utilizado la siguiente formula (3), que nos dará una gráfica exponencial:

$$Y_k = Y_{k-1} + a * (X_k - Y_{k-1})$$

Donde:

Y_k - salida del filtro en el instante k.
Y_{k-1} - salida del filtro en el instante k-1.
X_k - entrada al filtro en el instante k.
a - constante del filtro (0 < a <1).

Si a = 0 , máximo filtraje, $Y_k = Y_{k-1}$
Si a = 1, no hay filtraje $Y_k = X_{k-1}$

4 .Montaje práctico

Introducción

En este apartado describiremos el proceso para poder realizar el primer objetivo. Vamos a necesitar los componentes del escudo que podemos ver en el apartado 2 (materiales), dichos componentes los uniremos a la placa mediante estaño y un soldador.

Es recomendable practicar antes de soldar directamente en la placa que vamos a utilizar, aunque en caso de que nos equivoquemos tenemos un "chupón" para quitar el componente soldado.

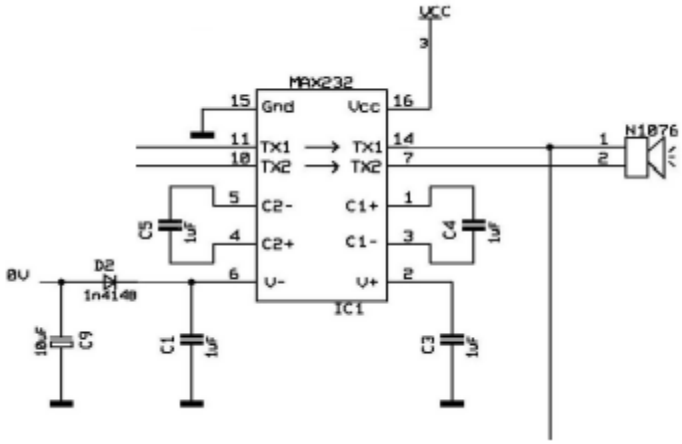
El esquema de montaje es el que podemos ver tanto en el apartado teórico, como en el de anexos (Figura x.x).Dividiremos el esquema en tres partes e iremos verificando el montaje parte a parte para así, en caso de haber cometido un error, poder detectar a tiempo el fallo y corregirlo.

Lo primero que tenemos que hacer será la distribución de los elementos del escudo en la placa, sin soldar, para ver que nos caben y tener una referencia de cómo tiene que quedar la distribución al final del montaje.

Aquí podemos ver una foto de la colocación de los integrados:

Driver de potencia

Una vez ya tengamos claro su distribución, empezaremos a soldar la primera parte. Esta será la correspondiente al integrado MAX232N**.**



Como podemos ver en la imagen anterior, y en el epatado de materiales, vamos a necesitar 4 condensadores de 1µF y uno de 10µF, además de un diodo 1n4148 y el "Sender". Los soldaremos a la placa siguiendo las uniones indicadas en la imagen. Todas las masas están llevadas a un mismo punto de la placa, ya indicado por la misma, para facilitar su unión y montaje.

Hay que tener en cuenta de que la patilla 6 del integrado sacamos tensión negativa, por lo que será la pata positiva de los condensadores los que irán a masa en esa rama.

Una de las cosas que hemos modificado con respecto a al esquema es la patilla 6, que ahora va directamente a tensión sin pasar por el transistor.

Las patillas 7 y 14 están unidas por cableado al "Sender" (transductor que envía las señales). En nuestro caso, hemos decidido colocar el transductor mirando hacia arriba, por lo que para poder medir distancias en horizontal tendremos que colocar el Arduino, junto al escudo incorporado, en vertical.

Las patillas 11 y 10 estarán conectadas al Arduino en los pines 5 y 4 respectivamente. En estos puntos tendremos que captar pulsos de ondas cuadradas.

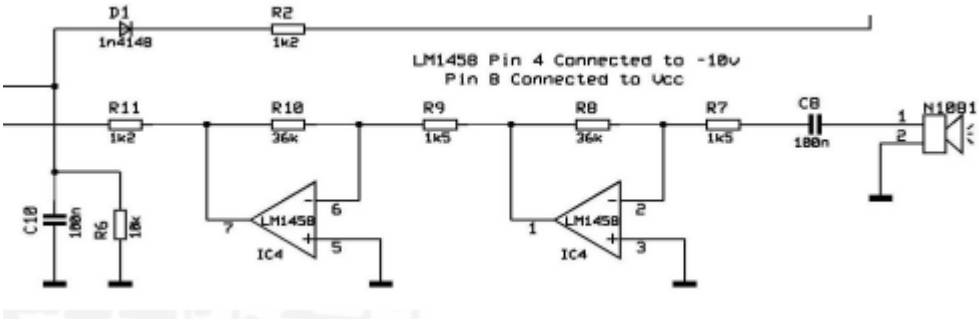
Una vez tenemos esta parte montada, pondremos dos cables auxiliares, uno en masa y otro en tensión, para poder verificar el montaje. Las condiciones para afirmar que el montaje está bien hecho son las siguientes:

- Patilla 14: +8.5V
- Patilla 7: -8.5V
- Patilla 2: +10V
- Patilla 6: -10V

NOTA: Puede que los valores no sean exactos.

Amplificación de la señal recibida

La segunda parte del esquema que montaremos una vez verificadas las condiciones anteriores, es la perteneciente al integrado MC3303P.



Una de las ramas del esquema que vemos, la inferior, se corresponde con el integrado MC3303P y sus amplificadores funcionan como el LM324. (Sin contar el Reciver)

Para esta parte del montaje vamos a necesitar coger las resistencias comerciales más próximas a los valores del esquema, de forma que modifiquemos lo mínimo posible el valor de la ganancia. Por otra parte, también necesitaremos dos condensadores de 100nF y un diodo 1n4148.

A partir de los datos de las resistencias teóricas obtendríamos la siguiente ganancia:

$$G = \frac{R2}{R1} = \frac{36k}{1{,}5k} = 24$$

Como tenemos dos amplificadores iguales la ganancia total sería:

$$G_T = G_1 \cdot G_2 = 24 \cdot 24 = 576$$

Como hemos dicho, tenemos que buscar unas resistencias comerciales de forma que obtengamos la misma ganancia total o lo más parecido posible a ella. Aunque existen resistencias de 1.5k no hay de 36k, por lo que nos decidimos a poner en un amplificador una resistencia de 39k y en el otro una resistencia de 33k. Por lo tanto obtenemos las siguientes ganancias:

$$G_1 = \frac{R2}{R1} = \frac{39k}{1{,}5k} = 26$$

$$G_2 = \frac{R2}{R1} = \frac{33k}{1{,}5k} = 22$$

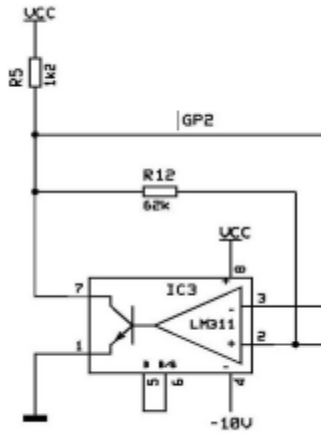
$$\boldsymbol{G_T} = G_1 \cdot G_2 = 26 \cdot 22 = \mathbf{572}$$

Por otra parte, la otra rama que incluye unas resistencias de 10k y 1k1 (ajustada al valor comercial más cercano), hará de filtro de la señal. Más adelante veremos que tendremos que modificar algunos componentes de esta rama para mejorar la señal filtrada. Esta rama estará conectada a la patilla 14 del MAX232N.

Verificamos que no han sido modificados los valores del MAX232N.

Detector de nivel o comparador

Por último, la tercera parte del esquema estará formada a partir del integrado LM311.



En esta parte necesitaremos dos resistencias, tras haberles ajustado al valor comercial más cercano, una de 1k1 y otra de 68k.

Las patillas 3 y 2 tienen que estar conectadas respectivamente a las ramas superior e inferior de la parte del amplificador que acabamos de ver.

A su vez, la patilla 2 tiene que estar conectada, pasando por la resistencia de 68k a la patilla 7. De este punto de unión iremos al pin 2 del Arduino (GP2 en la imagen) y a tensión pasando antes por la resistencia de 1k1.

Los puntos de verificación serán los siguientes:

- Patilla 2: Punto de test.
- Patilla 3: Punto de test.
- Patilla 4: -10V
- Patilla 7: Detección de eco.
- Resistencia 1k1: +5V
- Patilla 8: +5V
- Patilla 1: 0V

NOTA: Puede que los valores no sean exactos.

Una vez tenemos todo montado, tendríamos que tener el circuito completo.

Modificaciones

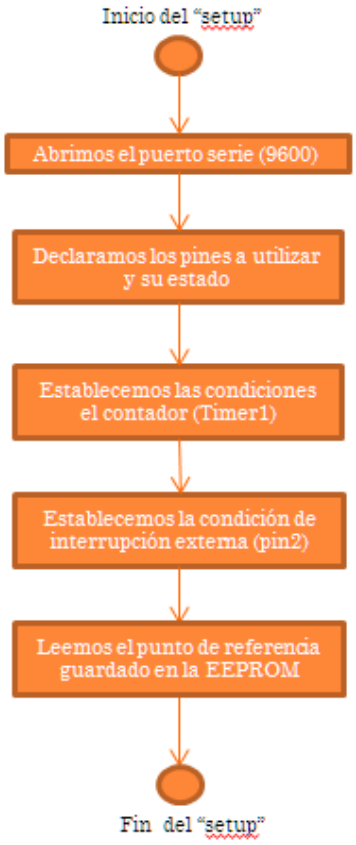
Cuando verificamos mediante el osciloscopio nos damos cuenta que la rama superior de la parte del amplificador, la cual se encarga del eco, es necesario modificarla. Para ello cambiaremos la resistencia de 1k1 por una de un valor menor, 330, y la de 10k por una de 22k.

Para el objetivo seis, vamos a necesitar el integrado LM35, por lo que será necesario añadirlo al escudo soldándolo mediante pines hembras. La posición en el escudo no tiene importancia pero se ha de tener especial cuidado a la hora de colocar el integrado en los pines, es decir colocar la patilla de tensión en el pin hembra que hemos soldado a tensión y lo mismo con masa. La de salida de datos no lleva a error porque es la del medio. (Para más información ver el "datasheet" del integrado en el apartado de anexos).

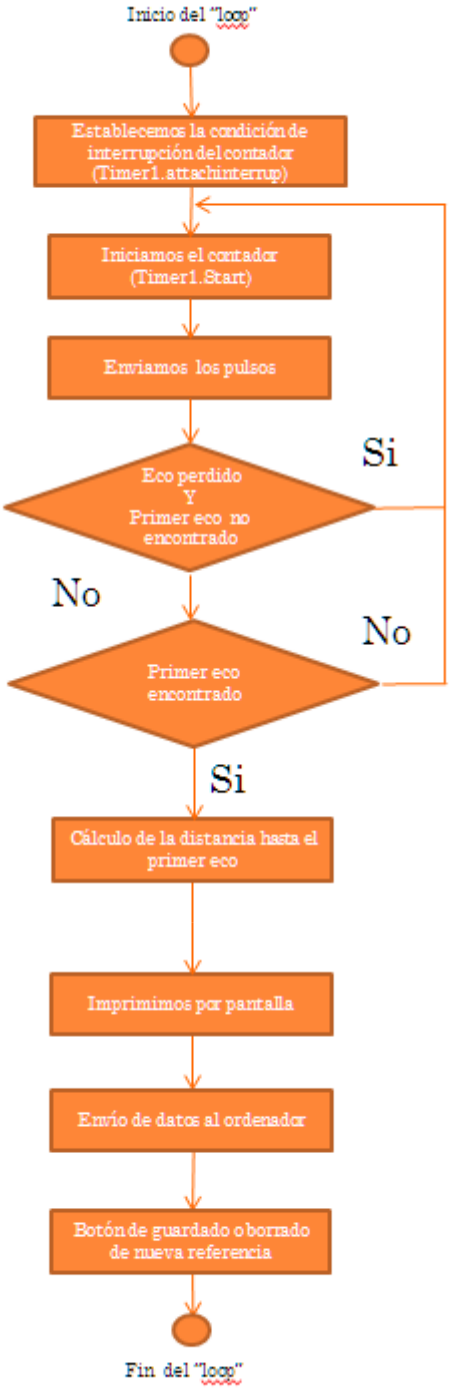
Para establecer el nuevo punto de referencia hemos añadido un botón al escudo, el cual nos facilita el uso del dispositivo. Para saber cómo se ha de colocar hemos practicado antes con el ejemplo de "Digital->Button" que podemos abrir desde los ejemplos del programa de Arduino.

5. Diagrama UML

SETUP

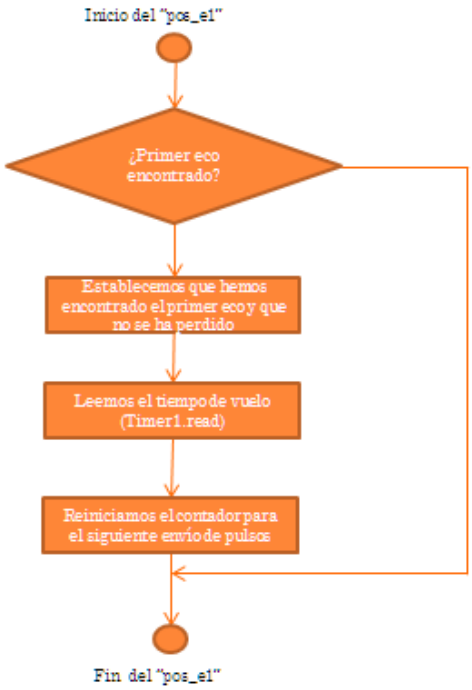


LOOP

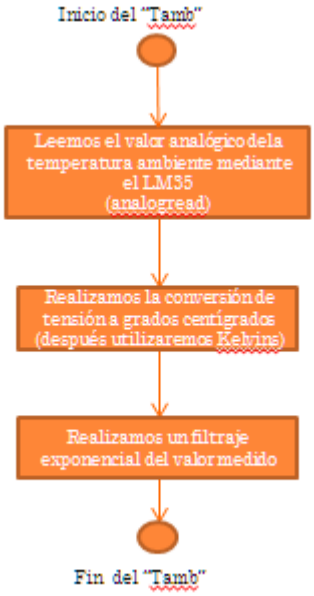

Inicio del "loop"
Establecemos la condición de interrupción del contador (Timer1.attachinterrup)
Iniciamos el contador (Timer1.Start)
Enviamos los pulsos
Eco perdido Y Primer eco no encontrado
Si
No
Primer eco encontrado
No
Si
Cálculo de la distancia hasta el primer eco
Imprimimos por pantalla
Envío de datos al ordenador
Botón de guardado o borrado de nueva referencia
Fin del "loop"

Envío de pulsos

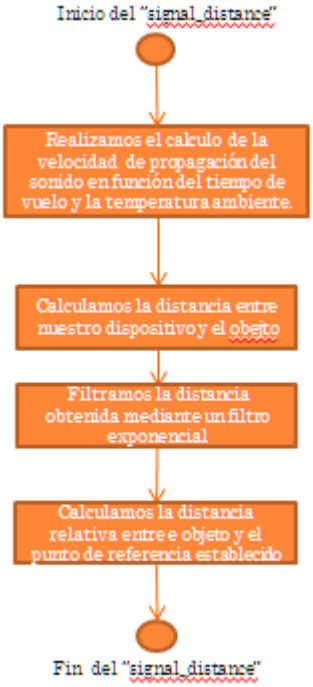
Tiempo de vuelo del primer eco



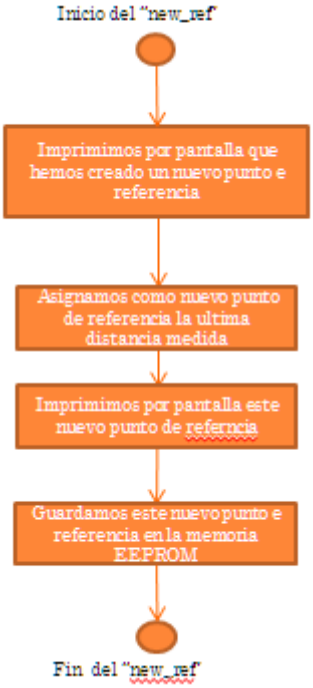
Compensación mediante temperatura ambiente



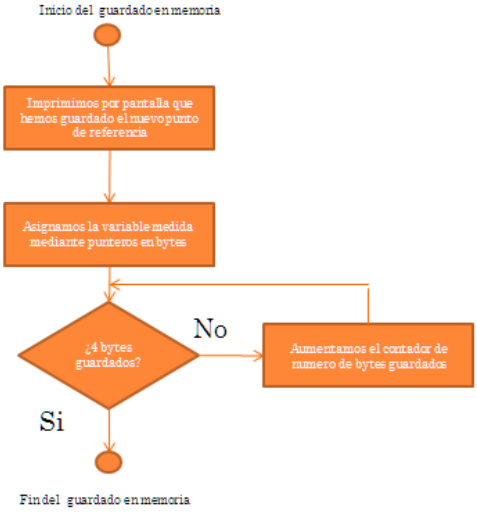
Caculo de la distancia relativa en función del tiempo de vuelo y la temperatura ambiente.

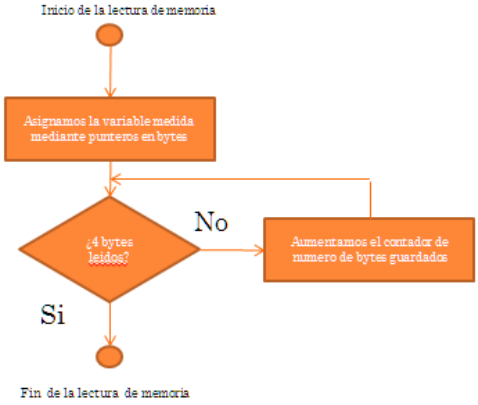


Establecimiento del nuevo punto de referencia

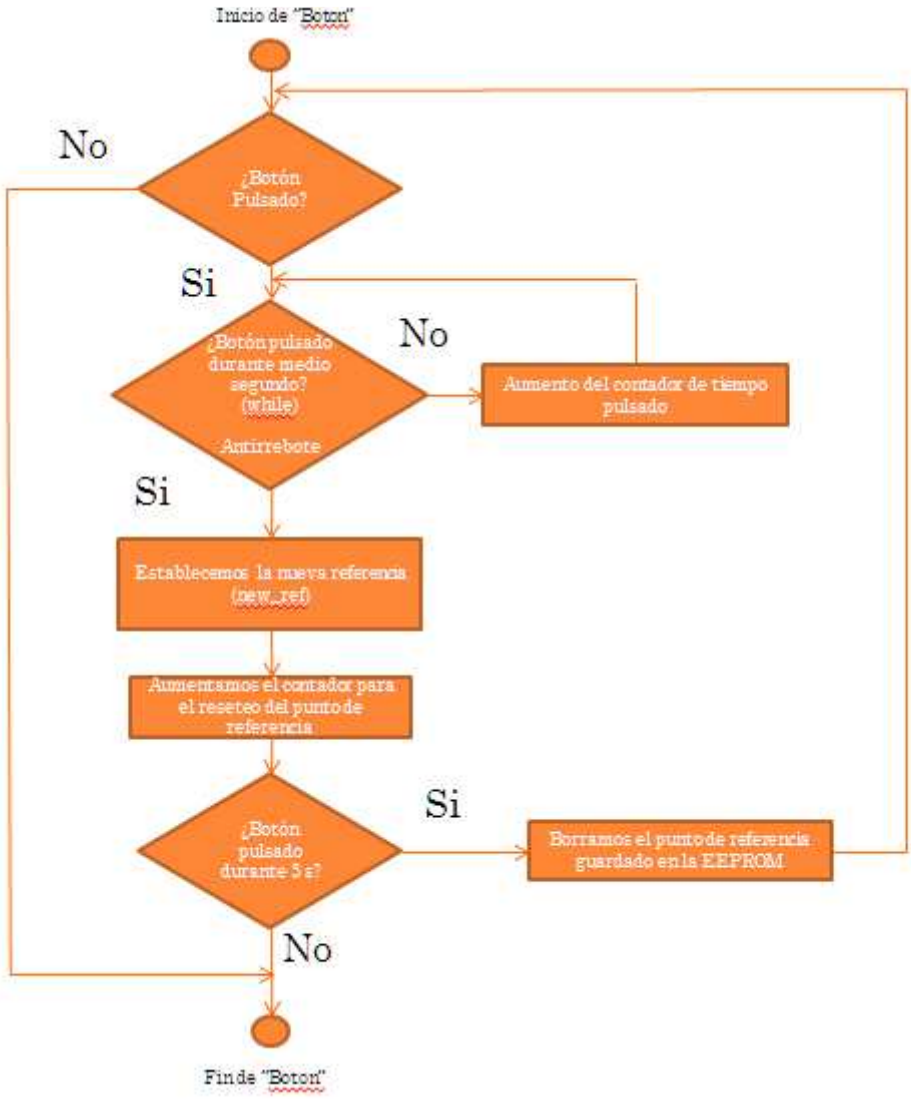


Lectura y escritura en la memoria EEPROM

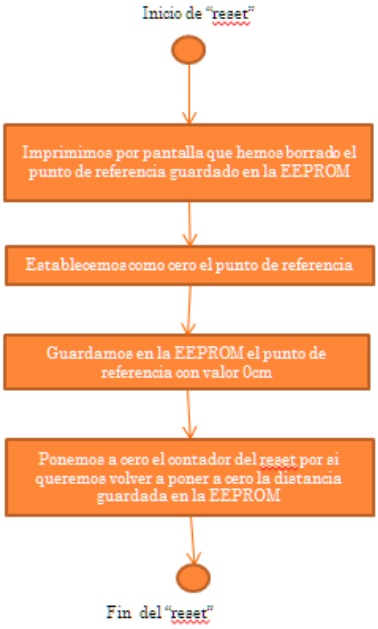




Botón de guardado o borrado del nuevo punto de referencia



Borrado del punto de referencia guardad o en la EEPROM



6. Realización del código

Tercer hito: Envío BEACON

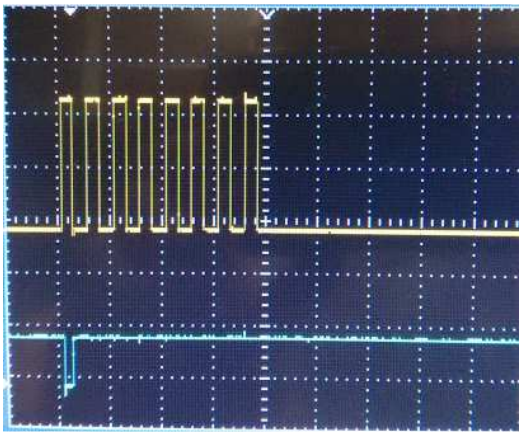
En este apartado nos centraremos en cumplir los objetivos de software propuestos, para ello empezaremos con el objetivo tres, el cual no insta a realizar un envío de señales ultrasónicas (BEACON).

A esta subrutina, que hemos llamado "**Sended_pulses**", será una señal que constará de 8 ciclos, es decir, 16 pulsos. Utilizaremos los pines 4 y 5, los cuales tendremos que declarar como salidas en el "setup" del programa, y los definiremos mediante mascaras AND y OR.

Como el tiempo de estado (nivel alto o nivel bajo) es de microsegundos no nos vale la función "delay", tenemos que hacer uso de la función "**delayMicroseconds**". Además, el tiempo que queremos es de 25 µs y esta función no acepta números con coma flotante, por lo que optamos por poner 12 y 13 µs de espera ya que su media hará los 12.5 necesarios.

```
void Sended_pulses()
 {
    int i=0;

   // Definimos los estados de los pines 4 y 5 mediante mascaras con puertas AND y OR.
   // Tienen que ser 16  pulsos (8 ciclos).(El cero tambien cuenta).
   for(i=0;i<8;i++)
   {
     PORTD=(PORTD&0xEF)|0x20;
     delayMicroseconds(12);
     PORTD=(PORTD&0xDF)|0x10;
     delayMicroseconds(13);     //Si quitamos esto tenemos pulsos , si lo ponemos ,ondas cuadradas.
   }
   delay(30);                   //Tiene que ser mas pequeño que el "Timer1.initialice" para que no corte la señal.
 }
```

Mediante una captura con el osciloscopio podemos observar una imagen de los pulsos enviados:



Una vez comprobamos que obtenemos los pulsos, podemos continuar con el programa.

NOTA: Como se explicará más adelante, este apartado se verá modificado por motivos varios. (También se explicarán los motivos).

Cuarto hito: Interrupción externa

Para cumplir este objetivo tendremos que hacer que el contador de tiempo ("Timer1") se pare cuando se produzca una bajada de nivel en el pin2, el cual hemos declarado como entrada en el "setup".

Para esta interrupción utilizaremos la función "**atachInterrupt (interrupción, función, modo**)". En interrupción establecemos el pin en el que estamos buscando la variación, pero para ello será necesario mirar la información técnica del Arduino ya que se declara como interrupción externa , que en el caso del pin2 es la numero 0; en función establecemos el nombre de la subrutina que se ejecutara cuando cumpla la condición, que en nuestro caso es la posición del primer eco ("pos_e1"); y en modo establecemos la condición para la cual si se cumple ejecuta la interrupción, que en nuestro caso es "FALLING", es decir, una bajada de nivel.

Según la web oficial de Arduino:

"Especifica la función a la que invocar cuando se produce una interrupción externa. Reemplaza cualquier función previa que estuviera enlazada a la interrupción. La mayoría de las placas Arduino tienen dos interrupciones externas: Las número 0 (en el pin digital 2) "(4)
.

Aquí podemos ver una imagen del código:

```
//Interrupcion externa del pin2.Buscamos el primer eco.
attachInterrupt(0, pos_el, FALLING);
```

Otra característica a tener en cuenta de esta función es que no podemos imprimir por pantalla en la subrutina que ejecutamos cuando se produce la interrupción, ya que conlleva mucho tiempo y deja de hacer el bucle.

Quinto hito: tiempo de vuelo

Este objetivo es una continuación del cuarto, ya que lo conseguimos a partir de la subrutina que ejecuta la interrupción externa.

Como hemos dicho anteriormente, esta subrutina con la que obtendremos el tiempo de vuelo la hemos denominado "**pos_e1**", es decir posición del primer eco.

Esta subrutina solo se ejecutara si se ha producido la condición de interrupción del pin externo explicada anteriormente y si no hemos encontrado el primer eco. Esta última condición es muy importante ya que de esta forma nos aseguramos de que estamos obteniendo el eco correcto.

Si cumplimos ambas condiciones obtenemos el tiempo del contador (timer1) y hacemos un "restart" (volver a comenzar) del mismo. Además establecemos que hemos encontrado el primer eco y que no se ha perdido ya que son las condiciones para calcular la distancia.

Aquí podemos ver el código empleado:

```
//Localizacion del primer eco (en tiempo) solo si no lo hemos buscado.
void pos_el()
 {
   if(el==0)
   {
     el=1;                          //Hemos encontrado el primer eco.
     lost=0;                        //No se ha perdido.
     time_el=Timer1.read();         //Leemos el tiempo que ha tardado
     Timer1.restart();              //Reiniciamos el timer para el siguiente envio de pulsos.
   }
 }
```

Sexto hito: Compensación mediante temperatura

Para la compensación mediante temperatura necesitaremos un LM35. A partir de él obtendremos una señal analógica que deberemos leer, para ello utilizaremos la función "analogread" donde tendremos que declarar la entrada que estamos utilizando, en nuestro caso es la "0".

Además para esta subrutina a la que llamaremos "**Tamb**" utilizaremos los conocimientos adquiridos en la asignatura de Instrumentación Electrónica para obtener la fórmula de conversión analógica de V a ºC.

Lo primero es hacer una conversión de los valores analógicos a valores de tensión, estas lecturas tendrán unos valores de 0 a 1023, donde 0 serán 0 V y 1023 serán 5V.

Aquí podemos ver el código empleado:

```
//Calculos para obtener la temperatura ambiente en °C. (Hay que pasarlos a °K)
float Tamb()
{
  float LM35;              //Señal analogica recivida del LM35.
  float Tgrad;             //Temperatura ambiente en grados.

  LM35=analogRead(0);                 //Leemos lo que nos llega a la entrada analogica;
  Tgrad=(LM35*5/1024)*100;            /*Con el parentesis hacemos la conversion analogica a V
                                      y multiplicando por 100 obtenemos grados.*/

  return Tgrad;
}
```

NOTAS: Esta subrutina se verá modificada más adelante. Tal como podemos ver en la imagen, el dato resultante esta en grados centígrados, pero para el cálculo de la distancia tendremos que pasarlo a grados kelvin.

Séptimo hito: Auto-cero y guardado en la EEPROM.

Con este hito lo que conseguimos es guardar un punto de referencia y guardarlo en memoria para que así, en caso de sufrir una interrupción por cualquier motivo, tengamos este último punto de referencia.

Un caso práctico en el que podríamos utilizar este sistema es, por ejemplo, una maquina apiladoras de cajas. Este sensor se colocaría de tal forma que midiésemos la distancia que ocupan verticalmente, y al establecer el punto de referencia veríamos cuanto le queda de espacio. En caso de sufrir un paro del sistema, al volver a encenderlo esta distancia restante que teníamos hasta ocupar el espacio no se vería alterada ya que la teníamos guardada en memoria.

Para realizar este hito, hemos tenido que indagar más en programación, además de aprender a buscar información en diversas fuentes de información, tales como foros dedicados a este tipo de prácticas.
Además de buscar información sobre el propio funcionamiento del microcontrolador, cumpliendo así varios objetivos que persiguen esta práctica.

Para empezar, hemos tenido que ver el funcionamiento de los punteros ("&" y "*"); qué implica que declaremos una variable, como "float" en este caso, a nivel de espacio en el programa; por qué necesitamos utilizar punteros auxiliares o como utilizar la memoria EEPROM de nuestro microcontrolador.

Las subrutinas que vamos a utilizar contendrán dos variables, que serán la celda de memoria de la EEPROM, en nuestro caso la 0, y el puntero al valor que queremos guardar, en nuestro caso la nueva referencia.

Un "float" ocupa 32bits, es decir, 4bytes por lo que necesitaremos 4 celdas de memoria, ya que estas solo pueden guardar un byte.

Necesitamos un puntero auxiliar, "p", porque si no lo utilizásemos, al hacer el bucle "for" haríamos saltos de 4 en 4bytes, y lo que queremos es de byte en byte. De esta forma, utilizando un postincremento de las variables, estamos guardando o extrayendo, dependiendo de si guardamos o leemos la información, la variable "float" byte a byte.

Para hacer uso de la memoria EEPROM tendremos que incorporar su librería:

```
#include <EEPROM.h>              //Libreria para utilizar la memorira EEPROM
```

Una entendemos su funcionamiento, tenemos que utilizar dos subrutinas, una de lectura de la memoria y otra de escritura. Aquí podemos ver el código utilizado:

```
//Lectura de la EEPROM.
void EEPROM_rf(int EE, float *val)
{
  unsigned char* p = (unsigned char*)val;//Valor en bytes del float.
  unsigned char i;
  for (i = 0; i < 4; i++)
  {
    *p++ = EEPROM.read(EE++);//Lee el valor de la celda y la asigna al valor.
                             //Ambas con postincremento.
  }
}

//Escritura en la memoria EEPROM
void EEPROM_wf(int EE, float *val)
{
  Serial.println("Guardado de nueva referncia");
  unsigned char *p = (unsigned char*)val; //Valor en bytes.
  unsigned char i;
  for (i = 0; i < 4; i++)        //No estaban los corchetes.
  {
    EEPROM.write(EE++, *p++);    //En cada celda del la memoria EEPROM escrive el valor en bytes.
                                 //Ambas condiciones con postincremento.
  }

}
```

Como se puede observar, hemos hecho que se imprima por pantalla la verificación de que hemos guardado la nueva referencia.

La lectura de la memoria la tendremos que hacer en el "setup" del programa.(Lo podremos ver al final cuando se muestro el código completo).

El auto-cero consiste en tomar la distancia medida como nuevo punto de referencia, a esta subrutina la hemos llamado "**new_ref**". Para ello serán necesarias varias subrutinas, tales como la medida de la distancia, "**signal_distance**" y el guardado en memoria "**EEPROM_wf**" cuyo funcionamiento acabamos de explicar. Además, incluiremos una subrutina mediante la cual ejecutaremos esta nueva referencia llamada "**Boton**".

Como hemos dicho, utilizamos la subrutina "**Boton**". Esta consiste en que si pulsamos el botón que hemos incorporado al hardware de nuestra práctica establecemos el nuevo punto de referencia.

Para ello será necesario leer el estado del botón, y lo hacemos mediante un "digitalRead" del pin al que hemos asignado, mediante soldadura, nuestro botón. Este está a nivel alto cuando no lo pulsamos, por lo que cuando lo pulsemos estará a nivel bajo. Además, necesitaremos hacer un antirrebote del botón como medida de protección para no quemar la memoria EEPROM.

Aquí podemos ver el código empleado para la subrutina "**Boton**":

```
//Pulsamos tecla para envio de datos al ordenador.
void Boton()
{
  if (digitalRead(7)==LOW)                //Boton pulsado.
  {
    int cnt=0;

    //Serial.println("Boton pulsado");
    while(cnt<500 && digitalRead(7)==LOW) //Antirrebote del boton.
    {
      delay(1);
      cnt++;
    }
    new_ref();
    rs++;
```

NOTA: En el código podemos ver funciones omitidas las cuales únicamente nos sirven como puntos de verificación en caso de que falle el programa y no sepamos donde está el problema.

En la subrutina "**new_ref**" lo que hacemos es asignar como nueva referencia el valor que en el momento de pulsar el botón estaba midiendo el sensor. Además de guardarlo en la memoria EEPROM del programa e indicarlo por pantalla.

Aquí podemos ver el código empleado para esta subrutina:

```
//Establecer nuevo punto de referencia.
void new_ref()
{
  Serial.print("Nueva referencia ");
  nr=signal_distance();  //Nuevo punto de referencia.
  Serial.println(nr);
  EEPROM_wf(0,&nr);      //Llamada a la funcion.(celda de memoria, casilla distancia)

}
```

NOTA: Siempre que le demos al botón se nos mostrará una vez la distancia que hemos tomado como nueva referencia respecto al punto de referencia original (0 cm).

Para este nuevo punto de referencia haremos uso de la subrutina del cálculo de la distancia de la señal, "**signal_distance**".

En ella calcularemos a partir del tiempo de vuelo, explicado anteriormente, y de la velocidad del sonido la distancia que hay entre nuestro sensor de ultrasonidos y el objeto que estemos midiendo.

Para saber el valor del sonido hemos tenido que buscar la fórmula para su cálculo, y como hemos podido comprobar hace uso de la temperatura ambiente, por lo que haremos uso de la subrutina de la compensación mediante temperatura explicada anteriormente.

Una vez tenemos la velocidad de propagación del sonido y el tiempo de vuelo, tendremos que vivir el resultado de su producto, para obtener distancia, entre 10^6 para obtener metros y entre dos porque el cálculo del tiempo de vuelo cuenta tanto la ida como la vuelta.

Esta distancia que calculamos será la distancia hasta el primer eco respecto a nuestro dispositivo.

Código de la subrutina:

```
//Calculos para obtener la distancia hasta la señal en cm.
float signal_distance()
{
  float d_el;             //Distancia hasta el primer eco.

  vs=sqrt(y*R*(Tamb()+273)/M)*100;      //Lo multiplicamos por 100 para pasarlo a cm.
                                        //Añadimos la temperaura ambiente medida con el LM35.
  d_el=time_el*vs/(1000000*2);          //Lo dividimos por 10^6 para tener us, y por dos por que es ida y vuelta.

  return d_el;
}
```

NOTA: este apartado se verá modificado más adelante.

Una vez tenemos la distancia hasta el primer eco, tan solo hemos de restarle el nuevo punto de referencia que tenemos guardado en la memoria EEPROM para obtener la distancia relativa.

Código empleado:

```
d_rel=d_el-nr;                          //Distancia de raltiva.

return fd_el;
```

Octavo hito: filtro digital (e hito adicional: función de filtrado)

Aunque en este hito solo no dice de utilizar el filtro para la temperatura, también lo utilizaremos para el cálculo de la distancia, por lo que se verán modificados varias partes de códigos expuestos anteriormente.

Lo primero es buscar la fórmula matemática con la que realizaremos un filtro exponencial. Una vez obtenida (véase apartado teórico), tan solo tenemos que modificar parte del código expuesto anteriormente, como en el caso del cálculo de la temperatura ambiente o de la distancia de la señal.

$$Y_k = Y_{k-1} + a * (X_k - Y_{k-1})$$

Hemos realizado dos formas de utilizar el filtro, una de ellas, la primera que realizamos y la cual nos es marcada por el octavo hito, es aplicar una fórmula de forma directa a la subrutina. La otra forma que hemos utilizado (solo utilizar una de ellas), es la utilización de una subrutina a la que hemos denominado "**Filter**" que es llamada por las otras subrutinas ("**Tamb**" y "**signal_distance**") y que funciona mediante punteros.

Para la **primera opción**, tendremos que incorporar un "static float" a nuestra subrutina (para ambos casos, temperatura ambiente y distancia de la señal), ya que sino machacaríamos el valor cada vez que entrásemos en la subrutina. Por otra parte, si esta variable la declaramos como global, no nos haría falta que fuera estática.

Una vez utilizada la formula solo tenemos que devolver el valor calculado. El valor e "a" será el mínimo, 0.01, para un filtrado máximo.

Código subrutina "**Tamb**" con filtro:

```
//Calculos para obtener la temperatura ambiente en °C. (Hay que pasarlos a °K)
float Tamb()
{
  float LM35;              //Señal analogica recivida del LM35.
  float Tgrad;             //Temperatura ambiente en grados.
  static float FTgrad;     //Temperatura ambiente en grados con filtro.
                           //Es necesario que sea static sino se machaca.

  LM35=analogRead(0);                 //Leemos lo que nos llega a la entrada analogica;
  Tgrad=(LM35*5/1024)*100;            //Con el parentesis hacemos la conversion analogica a V
                                      //y multiplicando por 100 obtenemos grados.

  FTgrad=FTgrad+a*(Tgrad-FTgrad);     //Filtro exponencial.

  return FTgrad;
}
```

Código subrutina "**signal_distance**" con filtro:

```
//Calculos para obtener la distancia hasta la señal en cm.
float signal_distance()
{
  float d_el;             //Distancia hasta el primer eco.
  static float fd_el;     //Distancia filtrada. Estatico para no machacarlo.

  vs=sqrt(y*R*(Tamb()+273)/M)*100;      //Lo multiplicamos por 100 para pasarlo a cm.
                                        //Añadimos la temperaura ambiente medida con el LM35.
  d_el=time_el*vs/(1000000*2);          //Lo dividimos por 10^6 para tener us, y por dos por que es ida y vuelta.
  fd_el=fd_el+a*(d_el-fd_el);           //Distancia filtrada.

  d_rel=fd_el-nr;                       //Distancia de raltiva.

  return fd_el;
}
```

Como hemos dicho, para la **segunda opción** utilizaremos la subrutina "**Filter**" (Filtro), esta funcionara con punteros y a diferencia de la primera opción, tendremos que hacer uso de una variable auxiliar para no machacar el valor calculado. (La función será la misma).

Para realizar esta parte del código, como en la EEPROM, hemos tenido que buscar información y profundizar en el conocimiento de C.

Código empleado:

```
//Funcion filtro.
float Filter(float X, float *Y_1)
{
  float fY_1;

  fY_1=*Y_1+a*(X-*Y_1);
  *Y_1=fY_1;

  return *Y_1;

}
```

Una vez tenemos la subrutina, solo hemos de llamarla en las otras subrutinas sustituyendo lo siguiente:

Código subrutina "**Tamb**" con subrutina "**Filter**":

```
//Filtro exponencial.
FTgrad=FTgrad+a*(Tgrad-FTgrad);
```

```
//Si queremos utilizar la funcion filtro.
Filter(Tgrad, &FTgrad);
```

Código subrutina "**signal_distance**" con subrutina "**Filter**":

```
//Distancia filtrada.
fd_el=fd_el+a*(d_el-fd_el);
```

```
//Si queremos utilizar la funcion filtro.
Filter(d_el, &fd_el);
```

NOTA: Esta modificación afectara también a la distancia relativa y por tanto al punto de referencia.

```
//Distancia de raltiva.
d_rel=fd_el-nr;
```

Hito adicional: borrado del punto de referencia

Como **hito adicional** hemos añadido una condición de borrado de este punto de referencia en caso de necesitarlo por cualquier motivo. Esto lo hacemos mediante la subrutina "**reset**" que como hemos dicho guarda en la memoria EEPROM la nueva referencia como valor 0 además de indicarlo por pantalla.

Código de la subrutina "**reset**":

```
//Reseteamos el punto de referencia.
void reset()
{
    Serial.println("Referencia puesta a cero");
    nr=0;
    EEPROM_wf(0,&nr);
    rs=0;
}
```

Hito adicional: contador Timer1

En este hito, nos dispondremos a explicar el contador "Timer1" y sus funciones dentro de nuestro programa. Para hacer uso de este contador tendremos que incluir su biblioteca:

```
#include <TimerOne.h>           //Libreria del timer1
```

Una de sus funciones consiste en contar el tiempo de vuelo, es decir, el tiempo que tardamos hasta localizar el primer eco, como hemos explicado anteriormente (tiempo de vuelo).

Pero también tiene otras funciones, como la interrupción del contador pasador un cierto tiempo el cual nos indica que hemos perdido el eco. Al indicaros esto, mediante una variable , determinaremos esta condición de perdido, y por lo tanto en vez de mostrarnos un valor de distancia falso, nos indicará por pantalla que hemos perdido el eco.

La **primera condición** del contador, será que pasador un cierto tiempo, se interrumpa y se reinicie. Este tiempo máximo de contaje no es arbitrario, es el resultado de cálculo de máximo alcance de nuestro dispositivo. Este tiempo tendrá que ser mayor que el resultado de la distancia máxima que puede medir, 5 m , por dos (ida y vuelta), entre la velocidad de propagación del sonido:

Formula:

$$t = \frac{d_{ida} + d_{vuelta}}{vs} = \frac{5\ m + 5\ m}{343\ m/s} \approx 30.000\ \mu s$$

Una vez pasado este tiempo, activaremos la variable con la subrutina que activa la condición de perdido:

Código empleado para el reinicio del contador:

```
//Condiciones del Timer1
Timer1.initialize(35000);
```

La **segunda condición** será establecer la condición de perdido si se interrumpe el contador por exceso de tiempo:

Código empleado:

```
Timer1.attachInterrupt(lost_eco);
```

Para establecer esta condición utilizaremos la subrutina "**lost_eco**":

```
//Funcion del eco perdido.
void lost_eco()
{
  lost=1;
}
```

Hemos utilizado otras funciones del "**Timer1**" como "timer1.start" para iniciar el contador o el "Timer1.restart" para reiniciarlo una vez obtenido el valor deseado de la distancia.

NOTA: Veremos su aplicación en el código más adelante

Rutina Setup

En esta parte del código declararemos los pines que vamos a utilizar y su estado , el putero serie por el que vamos a imprimir por pantalla, al igual que las condiciones del Timer1 mencionadas anteriormente. Tambien es donde estableceremos la condición de interrupción externa y leeremos la variable guardad en la memoria EEPROM.

Código empleado:

```
void setup()
{
  //Para imprimir por pantalla. Abre el puerto serie a 9600 bps.
  Serial.begin(9600);

  //Declaramos los pines.
  pinMode(2, INPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(A0, INPUT);       //Entrada del LM35

  //Boton:
  pinMode(7, INPUT);        //Boton de enviar a graficar.
  digitalWrite(7, HIGH);    //Estado inicial del pin.

  //Condiciones del Timer1
  Timer1.initialize(35000); // Minimo será el tiempo en recorrer 10m (5 ida y 5 vuelta).(No menos de 30000).
  Timer1.attachInterrupt(lost_eco);
  //Interrupcion externa del pin2.Buscamos el primer eco.
  //No se puede imprimir dentro de una interrupcion, lleva mucho tiempo.
  attachInterrupt(0, pos_el, FALLING);

  //Lectura de la memoria EEPROM
  EEPROM_rf(0,&nr);
}
```

Rutina loop (bucle)

En esta función lo que haremos será hacer de forma reiterativa, como un bucle, las subrutinas que pongamos dentro de ella.

En nuestro caso lo primero que hacemos es establecer la condición de interrupción el Timer1 para después iniciarlo justo antes de enviar los pulsos.

Después de verificar que no se ha interrumpido el eco y que hemos captado el primer eco calcularemos la distancia hasta este y lo mostraremos por pantalla o lo graficaremos en el ordenador.

Para graficarlo en el ordenador tendremos que hacer uso de las **bibliotecas**, al igual que con la EEPROM o el Timer1 están declaradas antes de las variables globales, las cuales explicaremos más tarde.

```
#include <GraphSeries.h>        //Libreria para poder graficar
```

Además tendremos que establecer cuáles son las variables que vamos a mostrar, al igual que sus nombres:

```
//Para graficar:
GraphSeries g_aGraphs[] = {"Distancia Relativa","Temperatura Ambiente"};

//Envio de datos al ordenador para graficarlos.
g_aGraphs[0].SendData(d_rel);          //Envio de datos de la distancia relativa.
g_aGraphs[1].SendData(Tamb());         //Envio de datos de la temperatura pasada por el filtro.
```

Por ultimo ejecutaremos la función del botón siempre que queramos establecer esta última distancia medida como nuevo punto de referencia o queramos borrar dicho punto de referencia manteniendo más tiempo el botón pulsado.

Código empleado:

```
//Bucle.
void loop()
{

  Timer1.attachInterrupt(lost_eco); //El timer se interrumpe cuando el eco se ha perdido.

  Timer1.start();     //Iniciamos el Timer1.
  Sended_pulses();    //Enviamos los pulsos.

  //Mensaje de eco perdido.
  if(lost==1 && el==0)
  {
    Serial.println("Se ha perdido el eco.");
    lost=0;
  }
```

```
  //Busqueda de la señal si hemos encontrado el primer eco.
  if(el==1)
  {
    signal_distance();

    //Ayudas para visualizar los datos:
    //Serial.println(time_el);
    //Serial.println(signal_distance());
    Serial.println(d_rel);

    //Envio de datos al ordenador para graficarlos.
    g_aGraphs[0].SendData(d_rel);          //Envio de datos de la distancia relativa.
    g_aGraphs[1].SendData(Tamb());         //Envio de datos de la temperatura pasada por el filtro.
  }

  //Para graficar.
  Boton();
}
```

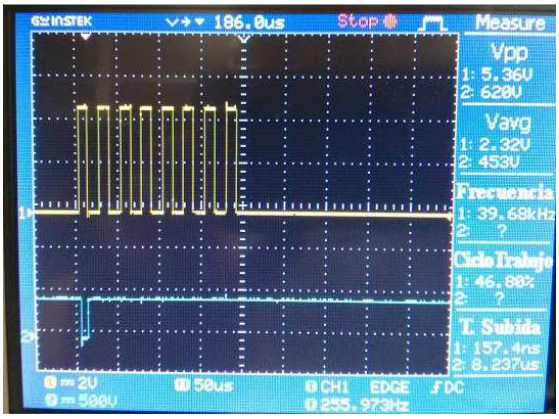
Variables globales

En este apartado veremos las variables globales utilizadas para el programa, al igual que las librerías.

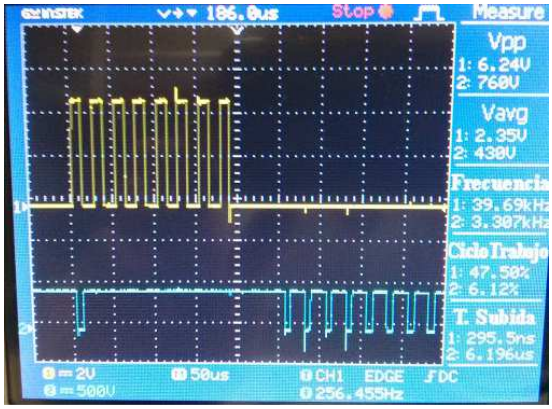
```
//Declaramos las librerias que vamos a utilizar:
#include <TimerOne.h>          //Libreria del timer1
#include <GraphSeries.h>       //Libreria para poder graficar
#include <EEPROM.h>            //Libreria para utilizar la memorira EEPROM

//Para graficar:
GraphSeries g_aGraphs[] = {"Distancia Relativa","Temperatura Ambiente"};

float vs,  a=0.1, nr, d_rel, fd_el;
unsigned long int time_el;
boolean el=0, lost=0;
int EE, i, rs;

//Declaramos las varaibles para la velocidad del sonido (vs)
   float y=1.4;       //Coeficiente de dilatacion adiabatica.
   float R=8.314;     //Constante universal de los gases.
   //float T=293.15;  //Temperatura en kelvin (20°C). Una vez la calculamos con el LM35 no hace falta declarar esta variable.
   float M=0.029;     //Masa molar del gas
```

7. Resultados

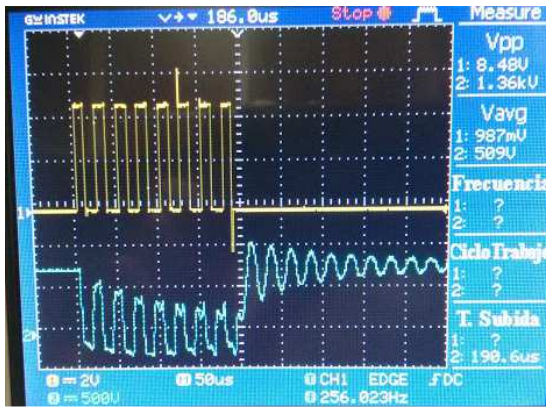
Resultado del envío de pulsos:



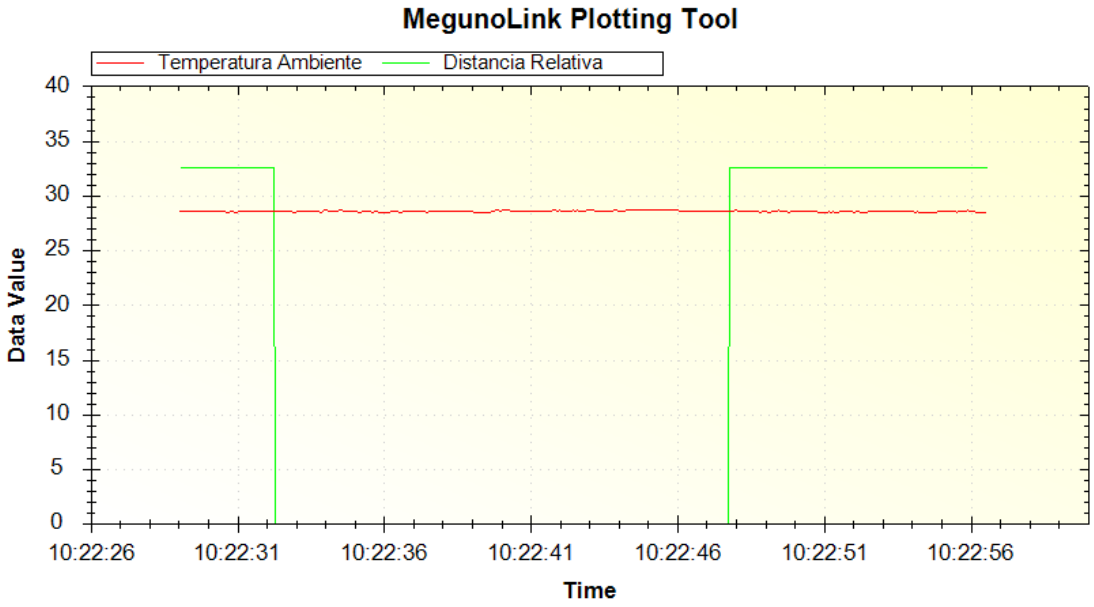
Resultado de medición de la distancia de un objeto vista desde el pin 2:



Resultado de la protección que hacemos para escuchar solo el primer eco:



Resultado de envío de datos al ordenador:



Como podemos observar en la imagen anterior, la línea roja representa la temperatura ambiente y la verde la distancia relativa entre nuestro punto de referencia y el objeto.

La línea verde sufre esta variación de 32 cm a 0 cm porque hemos pulsado del botón que guarda la distancia medida como nuevo punto de referencia, por lo que la distancia relativa pasa a ser nula.

Luego vemos como la línea verde vuelve a su valor original, esto se debe a que hemos mantenido pulsado el botón y por tanto a borrado el "nuevo punto de referencia", volviendo a medir desde nuestro dispositivo.

8. Conclusiones

Como conclusión del **primer objetivo**, el montaje practico, hemos deducido que se necesita práctica, aunque cualquiera con dedicación puede hacerlo en lo que respecta a soldar los componentes. Por otra parte, es necesario comprender el esquema y tener conocimientos eléctricos por si se requieren modificaciones del esquema principal para poder adaptarlo a nuestras necesidades. También deducimos que es necesario saber buscar información, ya sea vía internet, libros o profesores.

Es necesario hacer un planteamiento de lo que vamos a montar así como una planificación de donde vamos a montar cada componente, para que después no tengamos problemas. Por otra parte, aunque cometamos algún error casi siempre se puede solucionar.

Con el **segundo objetivo** podemos concluir que el uso de un esquema o diagrama nos facilita la comprensión de lo que hemos realizado. Ayuda tanto a terceros que no tienen por qué tener conocimientos de programación pero que si tienen que hacer uso de estos "inventos" o incluso para uno mismo si hace tiempo que se hizo y quieres recordar rápidamente como funciona el programa.

Concluimos a partir del **tercer objetivo** que es necesario mejorar continuamente nuestro conocimiento sobre programación, tanto a nivel alto como a nivel bajo ya que la mayoría de los lenguajes funcionan de manera similar. Tenemos que saber extrapolar la información aprendida para poder utilizarla en cualquier caso.

En este caso, hemos utilizado lo aprendido en el programa AVR para utilizar las máscaras AND y OR y poder programar a nivel bajo para poder realizar el envío de pulsos.

Hemos tenido que profundizar en nuestro conocimiento sobre punteros para poder realizar el salvado de memoria o los filtros.

Otra **conclusión** que obtenemos es que aunque sepamos mucho de programación, siempre tenemos que adaptarnos al microcontrolador o lo que estemos utilizando para programar. Que por mucho que sepamos de uno en concreto, tenemos que saber cómo utilizar la información que nos da el fabricante para poder utilizarla en nuestro favor.

Como en este caso que hemos tenido que indagar en cómo funciona nuestro Arduino UNO.

Otra **conclusión** que hemos podido sacar es que es necesario tener conocimientos de otras materias, que no solo "vale" con saber programación. Tal es el caso, que hemos tenido que utilizar los conocimientos adquiridos de Instrumentación electrónica o Matemáticas.

Por ejemplo, para el cálculo de la temperatura o el de los filtros.

Hemos tenido que aprender a poder comunicar dos dispositivos, en nuestro caso el microcontrolador con el ordenador. Que es necesario realizar modificaciones en nuestro programa o incluir archivos que no viene de fábrica, por lo que hemos tenido que aprender a movernos en "este mundillo".

Una **conclusión** que hemos sacado, es que no solo tenemos que ver por qué funcionan las cosas y si no, porque no funcionan. Tenemos que entender lo que estamos haciendo de tal forma de que sepamos que implica que modifiquemos algo que estaba funcionando bien.

Con esto me quiero referir a que si hacemos el código bien a la primera, y lo alteramos, ya sea en el orden en que se ejecutan las cosas, alteramos el tiempo de la emisión, emitamos más pulsos, sepamos como repercute a nuestro programa.

Por último, la **conclusión final** que obtenemos es que hemos de utilizar nuestro ingenio, ya sea para mejorar algo o crearlo desde cero. Todo se puede mejorar y con forme van mejorando nuestras “herramientas” podremos crear cosas que antes no se podían.

9. Referencias

1. **Instrumentación Electrónica**
   Miguel A. Pérez García
   Juan C. Álvarez Antón
   Juan C. Campo Rodríguez
   Fco. Javier Ferrero Martín
   Gustavo J. Grillo Ortega

2. **Instrumentación Electrónica**
   ASIAIN ANSORENA, DAVID.
   EUPLA

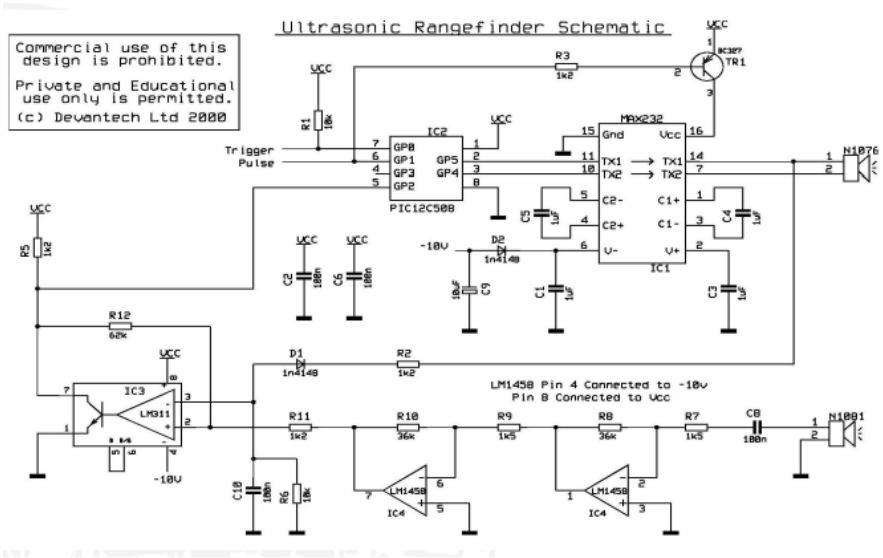
3. **Ingeniería en Automática Industrial**
   Software para Aplicaciones Industriales I
   artemisa.unicaua.edu.co/~gavasquez/res/Sw1/ProcesamientoInformacion_3.pdf

4. **Web oficial de Arduino**
   www.aruino.cc/es/

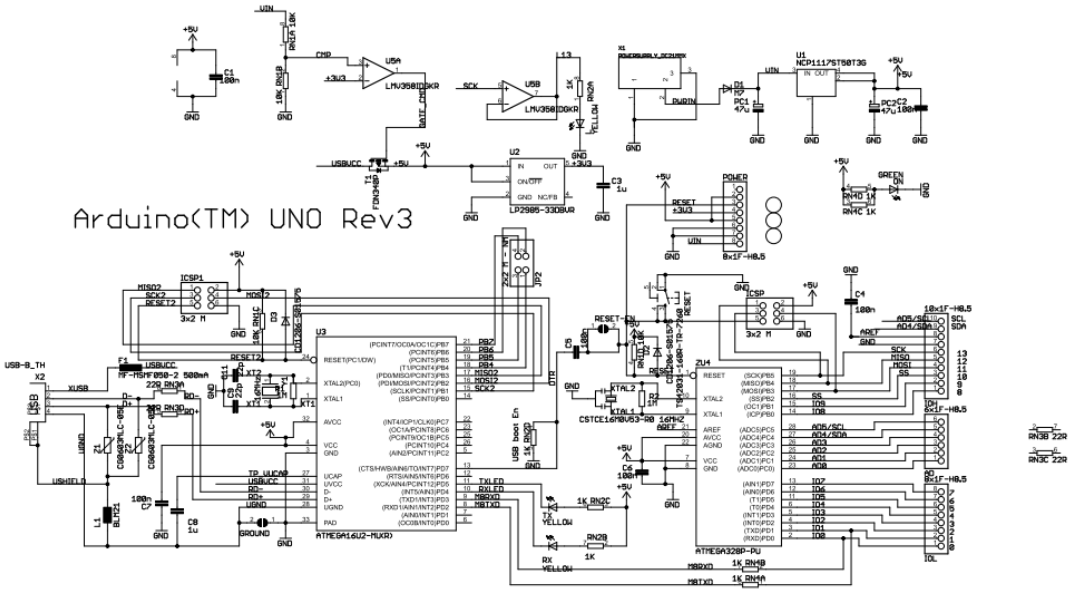
10. Anexos

Arduino

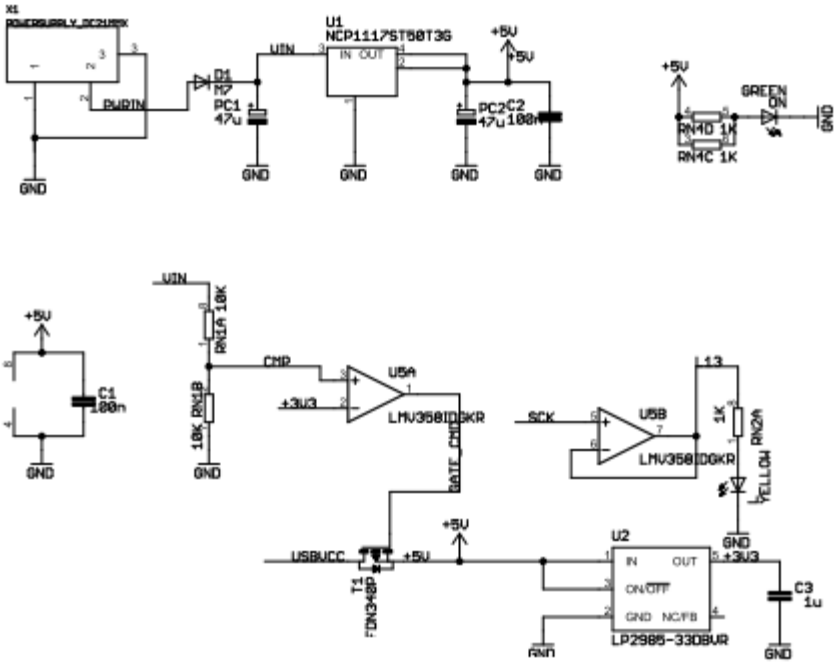
Esquema eléctrico del escudo:



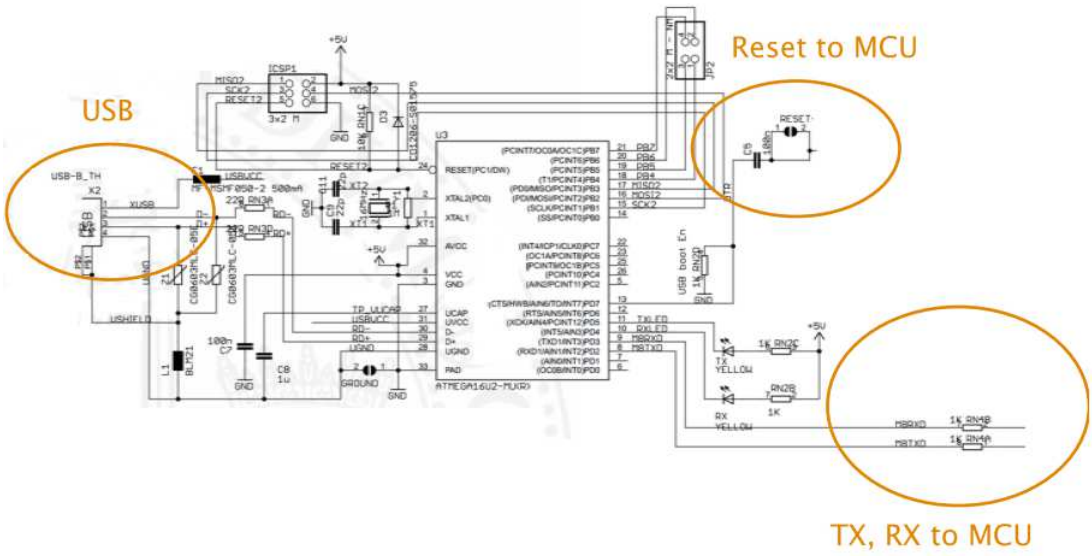
Hardware. Esquema general de Arduino.



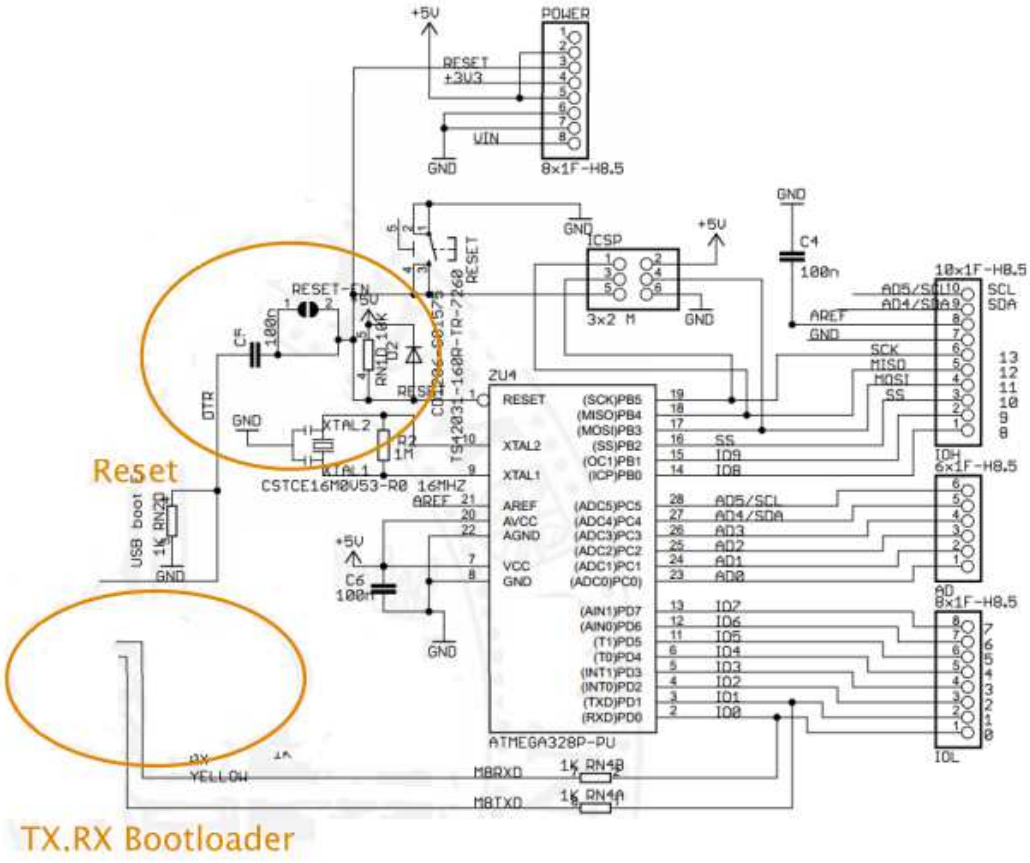
Fuente de alimentación



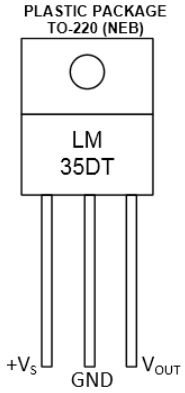
Hardware USB



Microcontrolador



LM35



FEATURES

- Calibrated Directly in ° Celsius (Centigrade)
- Linear + 10 mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at +25°C)
- Rated for Full −55°C to +150°C Range
- Suitable for Remote Applications
- Low Cost Due to Wafer-Level Trimming
- Operates from 4 to 30 V
- Less than 60-µA Current Drain
- Low Self-Heating, 0.08°C in Still Air
- Nonlinearity Only ±¼°C Typical
- Low Impedance Output, 0.1 Ω for 1 mA Load