Universidad de La Laguna

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

Grado en Ingeniería Electrónica Industrial y Automática

Localización activa de robots móviles

CRISTIAN GARCÍA SAAVEDRA

TRABAJO DE FIN DE GRADO

JULIO, 2017

Dedico este proyecto a mis padres y amigos por su gran apoyo y paciencia.

Gracias, porque sin ellos nada de esto habría sido posible.

También a mis profesores y compañeros.

RESUMEN

La estimación de la localización de robots móviles es un requerimiento esencial hoy en día como ayuda en la navegación. El presente proyecto estudia la posibilidad de dotar de dispositivos de localización de bajo coste a un robot móvil, carente de dispositivo GPS. Para dicho cometido utilizaremos sensores de posición, concretamente sensores ultrasónicos, microcontroladores Arduino, un módulo de radiofrecuencias para la comunicación entre placas, y un actuador para su movimiento.

ABSTRACT

Estimating the location of mobile robots is an essential requirement today as an aid in navigation. The present project studies the possibility of providing low-cost localization devices to a mobile robot, lacking a GPS device. For this purpose we will use position sensors, namely ultrasonic sensors, Arduino microcontrollers, a radiofrequency module for communication between plates, and an actuator for its movement.

Índice de figuras

Índice de ecuaciones

Índice de tablas

Índice de gráficas

CAPÍTULO 1.
INTRODUCCIÓN

1.1 Introducción

La estimación de la localización de robots móviles es un requerimiento esencial hoy en día como ayuda en la navegación. Implementando estos sistemas, se facilita la utilización de robots móviles en entornos complejos, reduciendo con esto, la incertidumbre relativa a su posición respecto a un sistema de referencia dado.

El presente proyecto estudia la posibilidad de dotar de dispositivos de localización de bajo coste a un robot móvil, carente de dispositivo GPS.

Inicialmente, para dicho cometido utilizaremos sensores de posición, concretamente sensores ultrasónicos, microcontroladores Arduino, un módulo de radiofrecuencias para la comunicación entre placas, y un actuador para su movimiento.

Nuestro entorno de trabajo constará de la disposición de varios sensores en posiciones fijas, que actuarán emitiendo pulsos hacia el objetivo. Éste estará situado en medio de estos, en donde habrá incorporado un sensor a modo de receptor de dichos pulsos.

El método de localización que utilizaremos será la trilateración. A partir de los pulsos emitidos por los nodos emisores, se medirán las distancias que hay entre el receptor y ellos. Normalmente estos nodos emisores se sitúan formando un triángulo.

Para conseguir hallar la posición de este, se emplea una serie de ecuaciones trigonométricas, las cuales tendremos que implementar en nuestro programa. Como software de control general a la hora de realizar nuestro trabajo, utilizaremos Arduino.

Arduino es una plataforma de prototipos de electrónica de código abierto basada en hardware y software flexibles y fáciles de usar.

1.2. Contenidos de la memoria

Este documento se ha dividido en cinco capítulos, además de sus anexos. A continuación, se resume el contenido más relevante de cada uno de ellos:

CAPÍTULO 1. INTRODUCCIÓN

Descripción de los aspectos principales del proyecto, motivación del mismo y objetivos planteados inicialmente para su realización, así como una breve descripción del contenido de la memoria.

CAPÍTULO 2. MARCO TEÓRICO

Breve análisis sobre los métodos de localización existentes para determinar la posición y cuál es el más óptimo para nuestro caso. Cuáles serán los sensores empleados. Introducción del programa empleado en el que tendrán lugar todos los ensayos, el medio de comunicación establecido para llevarlo a cabo y el actuador que utilizaremos.

CAPÍTULO 3. PARTE EXPERIMENTAL

Explicación de los pasos seguidos hasta la obtención de nuestro objetivo, dando lugar a todas las pruebas, junto con las herramientas empleadas para la realización de los montajes, comunicación, etc. Además, se comentan algunas observaciones generales sobre los resultados.

CAPÍTULO 4. CONCLUSIONES Y LÍNEAS ABIERTAS

Reflexión sobre los objetivos propuestos, así como el proceso de localización realizado, comprobando su exactitud y precisión.

Además, algunas posibles líneas sobre trabajos posteriores.

CAPÍTULO 5. PRESUPUESTO

Presupuesto acorde a los materiales empleados en nuestro proyecto, en el que se ve reflejado el bajo coste que este supone.

ANEXO I. CÓDIGOS ARDUINO

Se presentan los códigos fuente de los programas realizados en Arduino, comentando su funcionamiento para un mejor entendimiento de los mismos.

ANEXO II. DOCUMENTACIÓN

En este anexo encontramos los datasheet de los componentes utilizados en la construcción del dispositivo de localización.

CAPÍTULO 2.
MARCO TEÓRICO

2.1. Localización

En [1] encontramos una descripción de los sistemas de localización que resumimos a continuación.

2.1.1. ¿Qué es la localización?

Es el proceso por el cual se estima la posición de un móvil dentro de un determinado entorno.

Esto lo consigue a partir de la información recogida del medio ya sea obtenida por el propio móvil, o a través de diferentes sistemas sensoriales. La principal aplicación de los sistemas de localización entre otras, es la ayuda en la navegación, además de la de conocer la posición de un móvil.

2.1.2. Tipos de localización

Podemos hacer una clasificación de los sistemas de localización según:

• Conociendo o no la posición del móvil, tenemos:

- Localización local: aquella en la que se realiza un seguimiento de la posición del robot utilizando odometría, sabiendo la posición inicial con el fin de poder estimar la siguiente posición.

- Localización global: es la que intenta conocer la ubicación del móvil sin información a priori.

-Secuestro: se realiza un cambio de posición del robot sin que este sea consciente. A partir de ese cambio, el robot deberá darse cuenta y relocalizarse en su nueva posición.

• La representación de la posición:

- Sistemas determinísticos: conociendo la posición actual del objeto, la información de los sensores y el comportamiento de este ante los cambios que se producen en el entorno se puede predecir, la posición siguiente.

Ejemplo de un método determinístico: la correlación cruzada.

- Sistemas probabilísticos: son aquellos sistemas en los que a partir de variables aleatorias como es el ruido, las diferentes variables que intervienen en la localización se ven afectadas. Es decir, la posición del móvil es en base a una probabilidad.

Ejemplo de método probabilístico: la localización bayesiana.

2.1.3. ¿Quién realiza la localización?

- La localización que se puede realizar en el propio dispositivo es aquella a la que llamamos localización activa. El robot obtiene su ubicación a partir de los diferentes puntos de referencia que hay en su entorno además de las diferentes medidas que realiza sobre él mismo, y disminuir con esto su error de localización.

- La localización del dispositivo a partir del entorno (entornos inteligentes), es la localización pasiva. Ésta se ejecuta en un segundo plano, mientras el robot está ocupado con su tarea. Es el entorno, a través de un procesador central, el que obtiene la ubicación de los dispositivos presentes dentro del propio entorno de trabajo.

En [3] encontramos las técnicas de estimación de la posición.

2.1.4. Técnicas de estimación de la posición

Para poder implementar un sistema de navegación, es importante que el sistema conozca la posición en la que se encuentra y de este modo poder conocer la información necesaria para poder alcanzar su destino de manera eficaz.

Los sistemas de localización actuales utilizan de manera individual o combinándolas, una serie de técnicas de estimación de la posición, algoritmos, basados en principios geométricos. La mayoría de los métodos empleados para redes inalámbricas de sensores, basan su funcionamiento en uno de estos dos principios: trilateración y triangulación, entre otras como multilateración, análisis de la escena y proximidad.

2.1.4.1. Triangulación

La triangulación se basa en el empleo de fórmulas trigonométricas para determinar la posición de un objeto. Para ello, es necesario conocer los ángulos que se forman desde los emisores que emiten las señales de localización, de los cuales tendremos que conocer su posición, hasta nuestro objetivo.

En general, se requieren dos ángulos y la distancia entre dos puntos de referencia en un entorno bidimensional. Para tres dimensiones son necesarios dos ángulos, la distancia entre dos nodos de referencia y normalmente para especificar una posición precisa, se utiliza un vector de referencia constante con valor 0˚ (por ejemplo, el norte magnético), como se muestra en la Fig. 2.1.

Los principales inconvenientes de este método son la forma de calcular el ángulo con el que el emisor está enviando su señal, y el coste computacional que se lleva a cabo para operar con fórmulas trigonométricas.

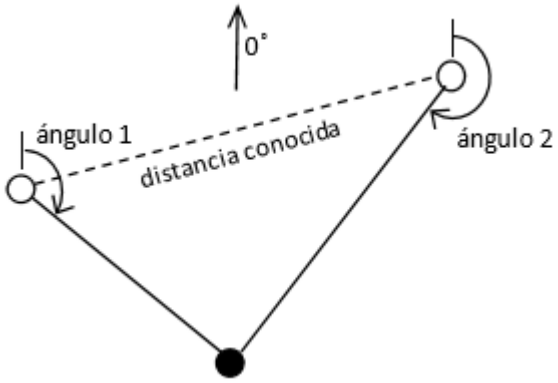


Fig. 2.1. Triangulación. Dos ángulos, la distancia a dos emisores (nodos blancos) y vector de referencia constante con valor 0˚ (apuntando al norte magnético), permiten determinar la posición (nodo negro).

2.1.4.2. Trilateración

La técnica de trilateración se asemeja bastante a la de triangulación. En este caso, calcula la posición midiendo las distancias desde él mismo objeto hasta varias posiciones conocidas. Para calcular la posición aproximada de nuestro objeto, es necesario conocer al menos las distancias desde tres transmisores que estén dispuestos como se muestra en la Fig. 2.2. en un mismo plano bidimensional. En tres dimensiones, son necesarios cuatro transmisores en planos distintos.

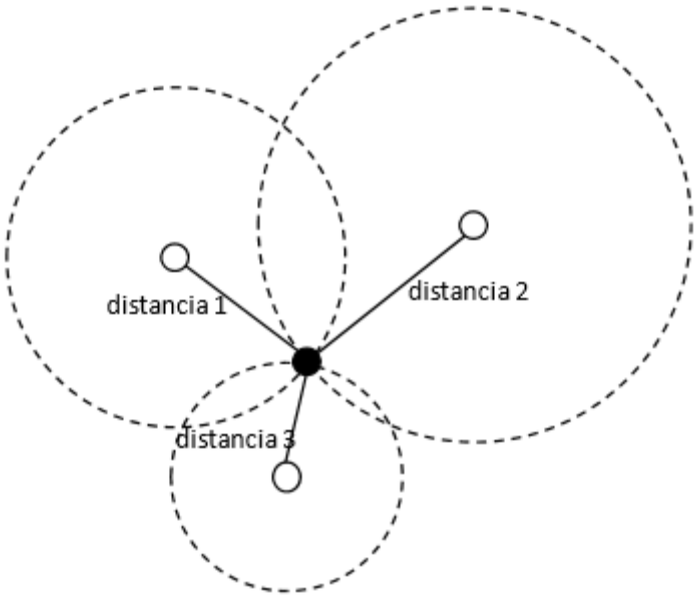


Fig. 2.2. Trilateración. La distancia a tres transmisores (nodos blancos) permite a nuestro objeto (nodo negro) determinar su posición.

Una vez que el receptor conoce la distancia, o la diferencia de distancias, a los diferentes emisores, es capaz de conocer su posición intentando encontrar aquel punto del espacio en el cual las medidas obtenidas de los diferentes emisores son coherentes. La búsqueda del mismo se realiza en función del método utilizado, siendo mediante intersección de esferas si se conocen las distancias absolutas (TOA), o mediante intersección de hiperboloides en el caso de diferencias relativas (DTOA). Algunos

ejemplos prácticos de tecnologías que se basan en estos principios son:

- Basados en diferencia de distancias: DeccaNavigatorSystem, OMEGA, NavigationSystem, GEE, LORAN-C, A-FLT…

- Basados en distancias absolutas: GPS, Galileo,…

La trilateración puede tener diversas aplicaciones, como la detección del lugar de caída de un rayo. Pero su empleo más habitual es el proceso de determinación de la posición empleado por el sistema GPS.

2.1.5. Sistema escogido

De entre los métodos explicados anteriormente, en este proyecto se optó por el método de trilateración debido a que buscamos un sistema de localización básico, que tenga un bajo coste tanto operacional como económico. Además, no ha surgido la necesidad del cálculo de su orientación, para el cual se tendría que recurrir a sistemas auxiliares.

En [5] encontramos las ecuaciones para el método de trilateración.

2.1.6. Cálculo de la posición

Como hemos mencionado anteriormente, emplearemos el método de trilateración, el cual se basa en una serie de expresiones que deberemos llevar a cabo para el cálculo de la posición del objeto.

Utilizando la ecuación de distancia, siendo r_i las distancias que hay desde nuestros emisores, cuyas coordenadas (x_i, y_i), hasta nuestro receptor, obtenemos nuestra posición (x, y). Partiendo entonces de las siguientes tres ecuaciones:

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$

$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$

$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

Ecuación 2.1. Ecuaciones de distancia.

Podemos ampliar los cuadrados de cada uno:

$$x^2 - 2x_1x + x_1^2 + y^2 - 2y_1y + y_1^2 = r_1^2$$

$$x^2 - 2x_2x + x_2^2 + y^2 - 2y_2y + y_2^2 = r_2^2$$

$$x^2 - 2x_3x + x_3^2 + y^2 - 2y_3y + y_3^2 = r_3^2$$

Si restamos la segunda ecuación de la primera, obtenemos

$$(-2x_1 + 2x_2)x + (-2y_1 + 2y_2)y = r_1^2 - r_2^2 - x_1^2 - x_2^2 - y_1^2 - y_2^2$$

del mismo modo, restando la tercera ecuación de la segunda,

$$(-2x_2 + 2x_3)x + (-2y_2 + 2y_3)y = r_2^2 - r_3^2 - x_2^2 - x_3^2 - y_2^2 - y_3^2$$

Obtenemos un sistema de dos ecuaciones con dos incógnitas:

$$Ax + By = C$$
$$Dx + Ey = F$$

Cuya solución:

$$x = \frac{CE - FB}{EA - BD}$$

$$y = \frac{CD - AF}{BD - AE}$$

Ecuación 2.2. Solución del sistema. Método trilateración.

En [6] encontramos una pequeña introducción a los sensores de posición.

2.2. Sensores de posición

En robótica móvil se usan una gran variedad de sensores que miden distintas magnitudes. De todos ellos aquellos que más se utilizan son los orientados a resolver uno de los problemas más comunes de todo sistema autónomo: la determinación de la posición.

2.2.1. Tipos de sensores de posición

Este problema se resuelve mediante medidas relativas de la posición del objeto de estudio a partir de una o varias posiciones de referencia. Dependiendo de la técnica utilizada, se pueden clasificar de la siguiente manera:

1. Sensores de resistencia variable o potenciómetros.
2. Sensores capacitivos.
3. Transformador diferencial de variación lineal (LVDT).
4. Sensores magnetorresistivos.
5. Sensores de ultrasonidos.
6. Sensores ópticos.
7. Sensores inerciales (IMU).

2.2.2. Sensor escogido

De todos ellos hay unos que destacan por ser empleados en gran parte de las plataformas móviles debido a su eficacia y bajo coste, los sensores ultrasónicos, los cuales emplearemos en nuestro trabajo. Junto con estos, los sensores basados en luz son otros de los sensores utilizados para determinación de distancias a objetos y por tanto, también para la construcción de mapas de entorno.

Estos dispositivos juegan un papel muy importante en diversos campos como en los sistemas de navegación aérea, vehículos terrestres, máquinas de inyección, equipos quirúrgicos, etc.

Tanto en [7], [8] y [9], encontramos toda la información relevante sobre los sensores ultrasónicos.

2.3. Sensor ultrasónico

2.3.1. Funcionamiento del sensor

Un sensor de ultrasonidos se basa en el envío de un pulso de alta frecuencia, con el fin de medir distancias. Esto lo consigue cuando el pulso, transmitido por el aire, rebota en los objetos cercanos y es reflejado hacia el sensor.

La obtención de la distancia hacia los objetos viene dada por una simple relación:

$$d = v * t$$

Ecuación 2.3. Cálculo de la distancia en relación a velocidad-tiempo.

Donde d es la distancia que queremos conocer, v la velocidad del sonido, y t es el tiempo transcurrido desde el envío del pulso.

Suponiendo que la velocidad del sonido es constante, 340 m/s, la distancia la obtendremos entonces, multiplicando dicha velocidad por el tiempo en que tardemos en recibir el eco.

Sabiendo que la distancia obtenida en ese caro sería resultado del tiempo de ida y de vuelta de la onda, la expresión correcta será:

$$d = \frac{340m/s * t}{2}$$

Como el tiempo que mediremos estará en microsegundos, la expresión quedaría:

$$d = 0.017 * \frac{t}{2}$$

Estos sensores son conocidos por su bajo coste y sencillez. Tienen un rango de medición teórico de 2cm a 400 cm, con una resolución de 0.3cm, en el caso del sensor SR04. En la práctica, el rango de medición real es mucho más limitado, pero en nuestro caso, nuestros sensores llegarán a un rango mayor, ya que dispondremos nuestro sistema de manera que unos sensores actuarán, emitiendo en el caso de los que actúan como emisores, o recibiendo en el caso de los receptores, es decir, se emitirá y recibirá en un solo sentido, sin rebote.

Dependiendo de la orientación de la superficie a medir, puede provocar que la onda se refleja, de ahí a que se diga que son sensores de baja precisión. Además, no resultan adecuados en entornos con gran número de objetos, dado que la onda rebota en las superficies generando ecos y falsas mediciones.

En aplicaciones en que se requiera una precisión superior en la medición de la distancia, suelen acompañarse de medidores de distancia por infrarrojos y sensores ópticos.

2.3.1.1. Pulso ultrasónico

Se inicia el ciclo partiendo de la recepción de un pulso positivo en la línea TRIG de al menos 10us. Tras recibir el pulso, se envía un tren de ondas ultrasónicas y el módulo cambia el estado de la línea ECHO de 0 a 1 y la línea permanece en 1 hasta que se recibe el eco o en su defecto hasta que ha pasado el tiempo máximo de medición:

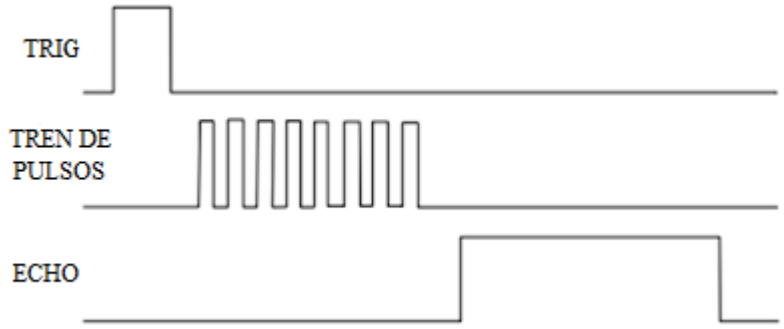


Fig. 2.3. Pulso ultrasónico.

2.3.2. Componentes básicos

Estos sensores se componen de cuatro componentes fundamentales:

Fig. 2.4. Componentes de los sensores ultrasónicos.

- **Transductor/ receptor:** emite pulsos en forma de ondas sonoras desde la cara del sensor. Además de emitir, también recibe ecos de esas ondas cuando son reflejadas en un objeto
- **Comparador:** una vez recibido el eco, se calcula la distancia comparando tiempos de vuelo con la velocidad del sonido.
- **Circuito detector:** partiendo de un elemento piezoeléctrico que tras ser comprimido por una cierta presión, generada por los pulsos del eco, deriva un nivel de tensión proporcional, el cual es procesado, calculando el tiempo transcurrido entre el ir y volver de los pulsos
- **Salida de estado sólido:** se genera una señal eléctrica interpretada por un controlador lógico programable. La señal procedente de los sensores digitales indica la presencia o ausencia de un objeto en el campo de detección, mientras que la señal que proviene de los sensores analógicos, indica la distancia de un objeto en ese mismo campo.

2.3.3. Consideraciones relativas

A la hora de conocer los sensores ultrasónicos tenemos que tener en cuenta algunas consideraciones.

2.3.3.1. Zona muerta

Existe una pequeña área a la que llamamos zona muerta o zona ciega en la que el sensor no puede detectar de manera precisa el objeto. Esta es el área existente entre la cara de sensado y la distancia de detección mínima. Si el objeto está demasiado cercano, la señal ultrasónica puede chocar contra el objeto pero como todavía se encuentra en modo transmisión, el eco es ignorado. También se puede dar el caso de que el objeto se encuentre dentro de esta zona, por lo que sucede que el eco será reflejado en la cara del sensor y reflectando nuevamente hacia el objetivo causando múltiples ecos.

Fig. 2.5. Representación de la zona muerta.

2.3.3.2. Margen de detección y haz efectivo

Este margen es el área comprendida entre los límites del rango del sensor, es decir, desde la distancia mínima a la máxima (fig.2.5.). Depende mayormente de la frecuencia de onda, además del medio de propagación entre otros.

2.3.3.3. Distancia de detección máxima

Dependerá fundamentalmente del material que se quiera detectar, debido a sus características (textura, dureza...), puede reflejar con menor o mayor intensidad. El rango de medición viene dado por el fabricante en la hoja de datos.

2.3.3.4. Ángulo del cono de disparo

Otro aspecto al que debemos prestar atención es el ángulo del cono de disparo. Fuera de este cono la señal ultrasónica existe pero es bastante débil, aunque los objetivos podrían ser detectados. El lóbulo principal tiene un ángulo de 15º aproximado.

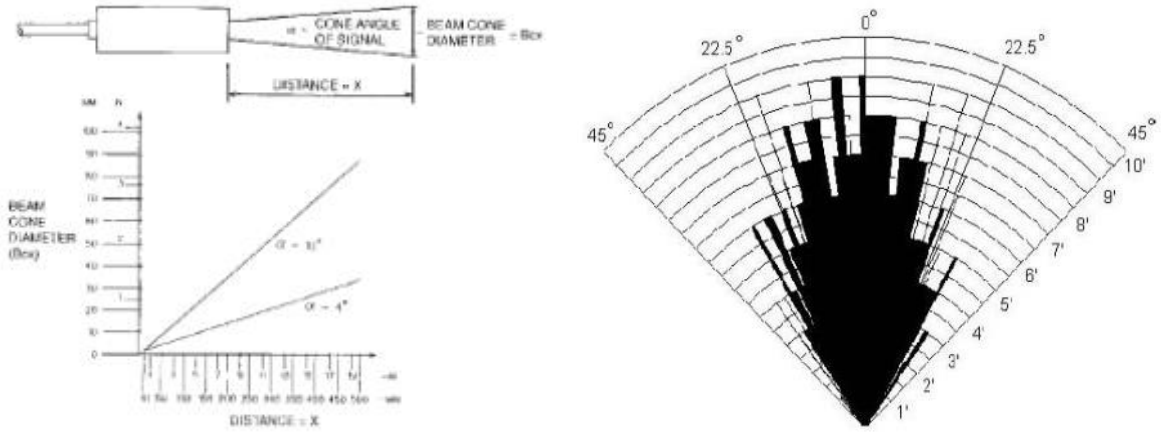


Fig. 2.6. Representación del ángulo del cono.

2.3.3.5. Inclinación del haz de ultrasonidos

A partir de un potenciómetro, se puede ajustar el cono formado por la onda emitida por el sensor para hacerlo más o menos ancho. Si se ensancha el cono la distancia de detección disminuye. Sin embargo, para objetos pequeños situados cerca del sensor, es decir, un ancho menor, aumenta su alcance.

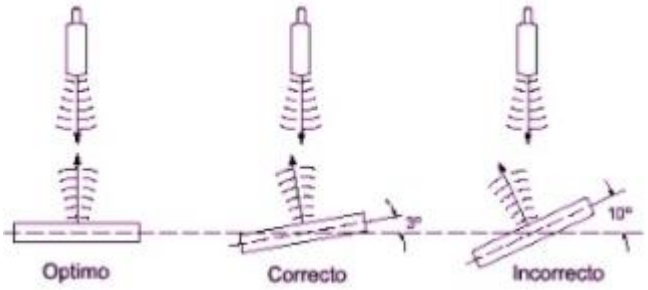


Fig. 2.7. Inclinación del haz ultrasónico.

2.3.3.6. Distancia entre sensores

El espacio entre sensores es determinado por el ángulo de onda. Los sensores deben ser ubicados de tal forma que no interfieran el uno con el otro.

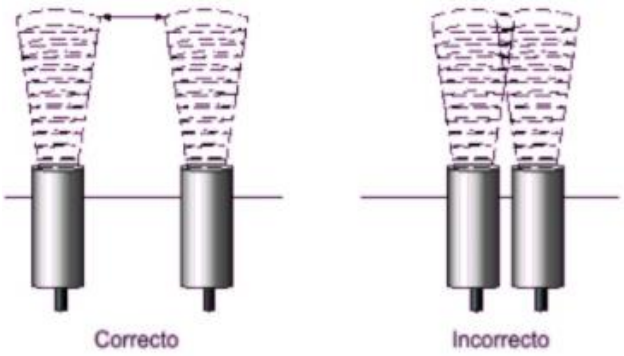


Fig. 2.8. Distancia entre sensores.

2.3.3.6.1. Efecto cross-talking

Cuando tenemos varios sensores conectados juntos, pueden verse afectados por lo llamado efecto cross-talking.

Debido a la cercanía entre sensores, un sensor puede recibir la onda de otro o un mismo sensor puede recibir su propia onda de un disparo previo si los tiempos de espera entre disparo y disparo no son adecuados.

Por tanto, si disponemos de varios sensores ultrasónicos y queremos reducir este efecto, la forma más común es secuenciando los disparos de cada sensor.

2.3.3.7. Frecuencia de disparo

La máxima frecuencia de disparo de un sensor de ultrasonidos, viene acorde a varias variables, las más significativas son:

- El tamaño del objeto: la máxima frecuencia para un objeto pequeño será menor que para un objeto grande.
- El material del que está hecho: dependiendo de su capacidad de absorber o capturar el sonido. No es lo mismo detectar una esponja que detectar el acero.
- La distancia a la que se encuentra: de esta depende en mayor medida para determinar la máxima frecuencia de disparo.

2.3.4. Ventajas y desventajas del sensor

2.3.4.1. Ventajas

- Los sensores ultrasónicos, pueden detectar objetos grandes a una distancia de hasta 8m o más.
- La detección no depende del color, del tipo o reflectividad óptica del material.
- Si tienen salida digital (ON/OFF), tienen excelente precisión y repetibilidad.
- Incluso en distancias de medición largas, es posible ignorar objetos de fondo.
- Tiene bajo coste computacional.
- La respuesta de los sensores analógicos es lineal con respecto a la distancia.
- No es necesario que haya contacto entre el objeto a detectar y el sensor.
- Son resistentes frente a perturbaciones externas tales como vibraciones, radiaciones infrarrojas, ruido ambiente y radiación EMI.

2.3.4.2. Desventajas

- Es necesario que el objeto a medir sea perpendicular al sensor, para poder obtener una medición exacta.
- La intervención de ruidos muy altos pueden ocasionar detecciones falsas.

- Tienen un mayor tiempo de respuesta, ya que necesitan tiempo para enviar y medir el eco recibido.
- Necesitan una distancia de detección mínima.
- Si se producen cambios en el ambiente, como temperatura, presión, humedad, etc. afectan a la respuesta del sensor.
- Existen materiales de baja densidad como son la espuma o la tela, que pueden ser difíciles de detectar a grandes distancias debido a su tendencia por absorber energía sonora.
- Hay dificultades de medición debido a que el ángulo es más crítico en una superficie lisa que en una irregular.

2.3.5. Schematic del sensor SR04

Este sería nuestro schematic del sensor ultrasónico SR04:

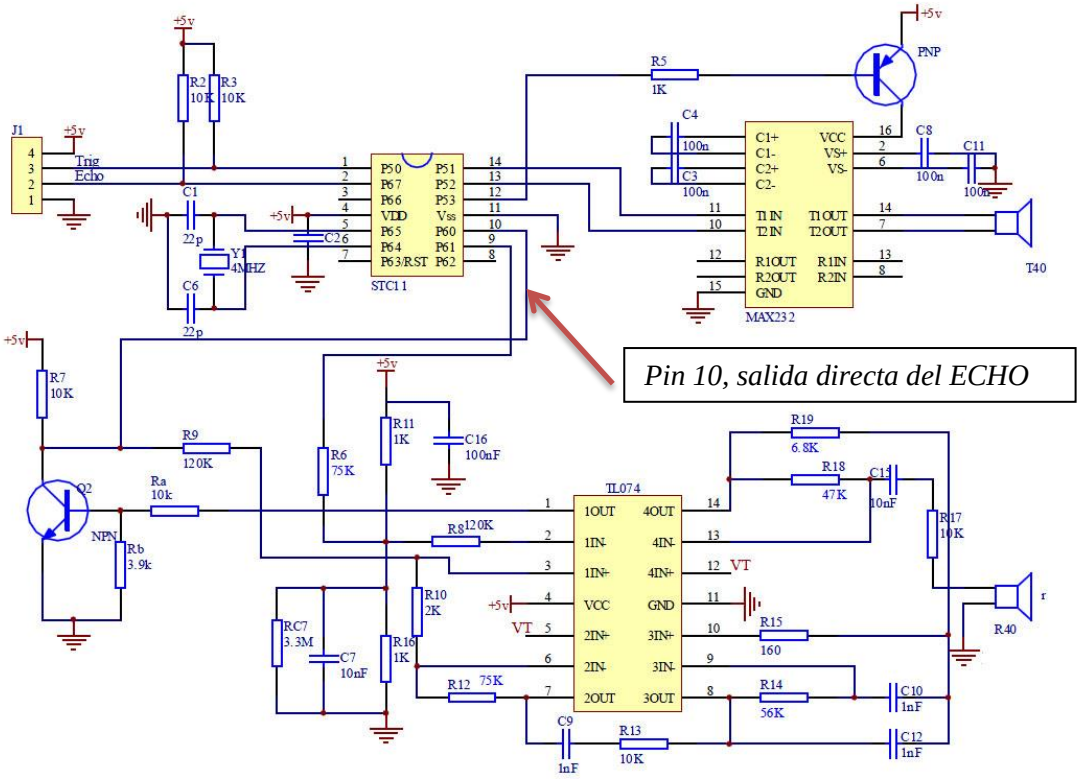


Fig. 2.9. Schematic del sensor ultrasónico SR04.

Para nuestro caso, para que un sensor ultrasónico actúe como receptor, necesitaremos hacer un cambio en el schematic, es decir, en el hardware del sensor. Si nos fijamos en el microcontrolador interno del sensor, STC11 en el schematic, el pin correspondiente al que sería al ECHO, sería el pin 10 (señalado en la fig.2.9.). En este pin soldaremos un cable para sacar directamente la salida del ECHO. De esta forma obtendremos nuestro receptor.

2.3.6. Esquema eléctrico

El esquema eléctrico que necesitamos es el siguiente:

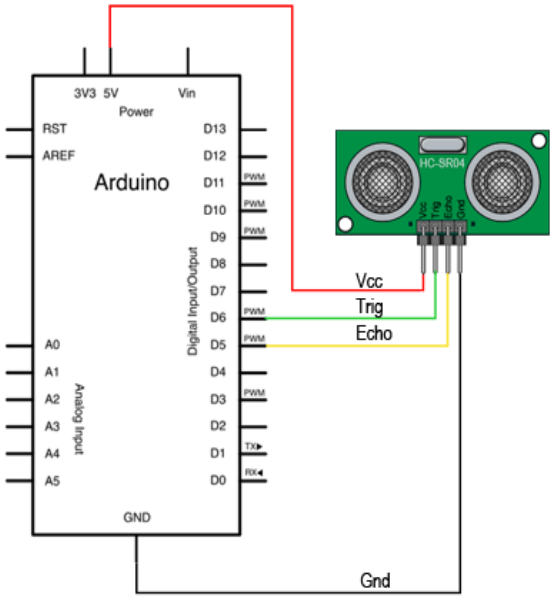


Fig. 2.10. Esquema eléctrico.

Representamos nuestro sensor conectado al microprocesador Arduino, del que hablaremos a continuación.

Para ver más características del sensor SR04, ver en el anexo ii.

2.4. Implementación en Arduino

En [10] encontramos definición y características de Arduino.

2.4.1. ¿Qué es Arduino?

Es una plataforma de prototipos de electrónica de código abierto basada en hardware y software flexibles y fáciles de usar. Básicamente, es una placa programable con entradas y salidas digitales y analógicas cuyo bajo coste la hace ideal para iniciarse en automatización o realizar diversos proyectos en electrónica, lo cual lo hace ideal para nuestro montaje.

2.4.2. Características y funcionamiento

Hay muchos otros microcontroladores y plataformas de desarrollo, pero Arduino además de simplificar el trabajo de programación, ofrece sobre otros sistemas:

- Bajo coste: las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladoras.
- Software multiplataforma: se puede ejecutar en sistemas operativos Windows, Mac y Linux, mientras que la mayoría de los sistemas microcontroladores están limitados a Windows.
- Entorno de programación simple y directo: su entorno de programación es fácil de usar tanto para principiantes como para usuarios avanzados.
- Código abierto y software extensible: el software Arduino está publicado como herramientas de código abierto, disponible para extensión por los usuarios que quieran.
- Código abierto y hardware extensible: está basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que puedes reunir los componentes y crear tu propia versión de la placa del módulo.

En [11] encontramos una descripción sobre el funcionamiento de la placa.

El funcionamiento de la placa, se compone de:

- Conexiones de entrada: a través de sensores conectados en estos pins, Arduino recibe datos del exterior.
- Microcontrolador: interpreta los datos recibidos del entorno a través de las conexiones de entrada, usando el lenguaje de programación. De esta forma, sabe que parámetros buscar y comparar, y que acciones tomar a modo de respuesta.
- Conexiones de salida: en función de las órdenes programadas en el microcontrolador, Arduino provoca la respuesta que necesitamos.
- Puertos/buses de comunicación: serie, I2C, SPI en la placa y Ethernet, wifi, modbus, can bus, RS232, etc, establecidos mediante shields.

En [12] encontramos un manual donde se describe la estructura básica de un programa cualquiera en Arduino.

2.4.3. Estructura de un programa

Según el manual de programación de Arduino, la estructura básica del lenguaje de programación es bastante simple y se compone de al menos dos funciones necesarias para que el programa trabaje. Estas encierran bloques que contienen declaraciones, estamentos o instrucciones.

```
void setup()
{
estamentos;
}
void loop()
{
estamentos;
}
```

En donde setup() es la parte encargada de recoger la configuración, y loop() es la que contiene el programa que se ejecutará cíclicamente.

La función de configuración debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras.

La función bucle (loop) siguiente contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

En [13] encontramos la comunicación entre microprocesador y pc.

2.4.4. Conexión

La conexión del Arduino con un ordenador, se realiza mediante puerto serie, normalmente. Únicamente es necesario conectar nuestra placa Arduino empleando el mismo puerto que empleamos para programarlo. Para acceder a él, podemos clicar directamente en la pestaña que se muestra en fig.2.11. o siguiendo la fig.2.12.

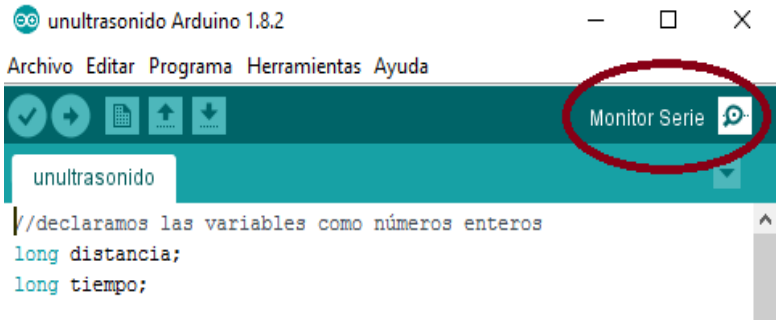


Fig. 2.11. Acceder al monitor serie directo.

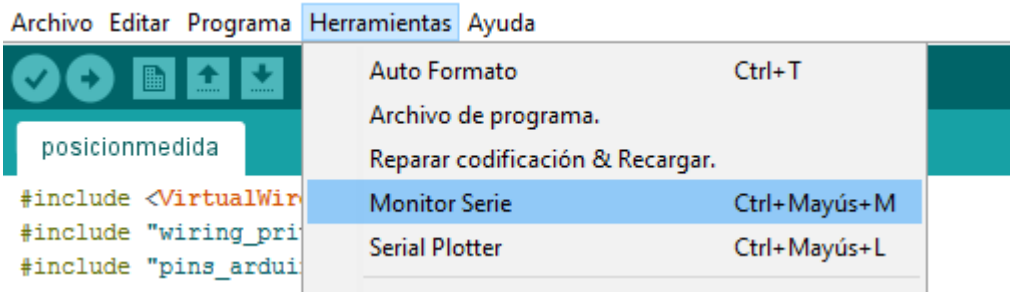


Fig. 2.12. Acceder al monitor serie.

El monitor de puerto serie nos permite enviar y recibir información a través del puerto serie. Su uso es muy sencillo, y dispone de dos zonas, una que muestra los datos recibidos, y otra para enviarlos:

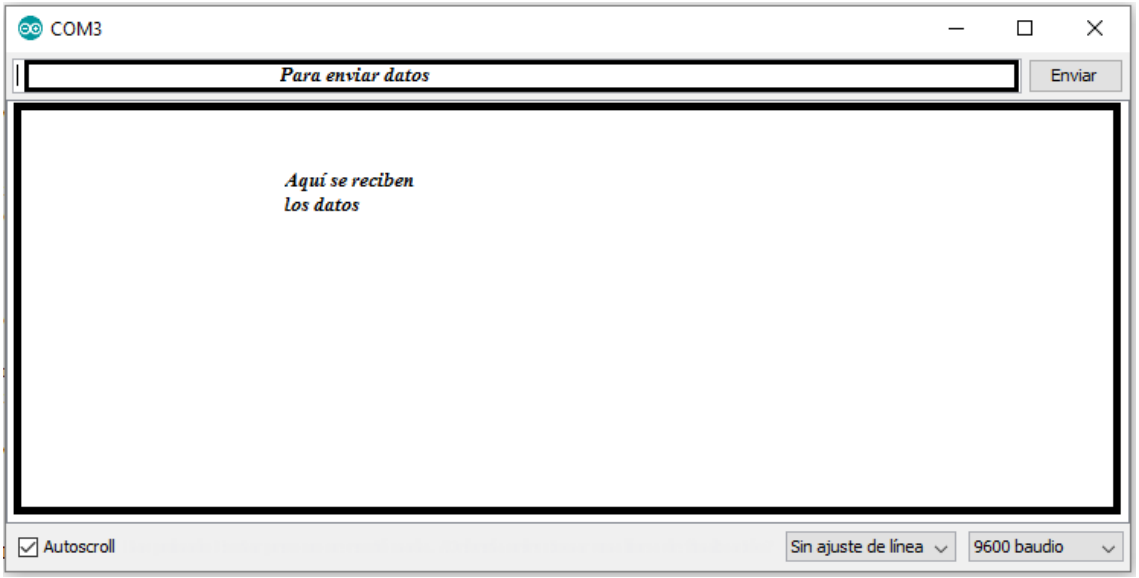


Fig. 2.13. Monitor serie.

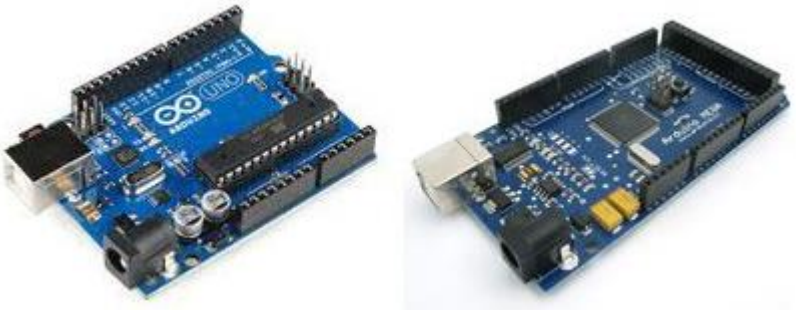
Es una interfaz sencilla, pero muy útil para realizar pruebas.

En [14] encontramos una pequeña comparación entre modelos comunes de Arduino.

2.4.5. Elección de la placa

Existen multitud de modelos de Arduino disponibles. Para saber cuál es el adecuado, normalmente nos fijamos en la cantidad de entradas y salidas que tenga, sobre todo las analógicas dado que son las que habitualmente restringen nuestro proyecto.

Para nuestro proyecto, emplearemos el modelo Arduino UNO (a), ya que es sencillo y basta para la ejecución de nuestro programa. Si tuviéramos un programa más complejo, que requiriese manejos de servos… o necesitara mayor número de salidas, utilizaríamos Arduino MEGA (b).



(a) **(b)**

Fig. 2.14. Modelos UNO (a), y MEGA (b).

En el desarrollo del proyecto, nos daremos cuenta de que necesitaremos ambos modelos para llevarlo a cabo, debido a la implementación de un servo en nuestro objeto.

2.5. Comunicación a partir de módulos de radiofrecuencias

En [15] encontramos una breve descripción de los módulos de RF 433Mhz.

2.5.1. Características

Los módulos de RF 433 MHz son transmisores/receptores inalámbricos empleados normalmente como forma de comunicación entre procesadores como Arduino.

Este tipo de módulos emisor FS1000A (a), y el receptor XY-MK-5V (b) se caracterizan por su bajo coste. La conexión es realmente sencilla. En primer lugar, alimentamos los módulos conectando Vcc y Gnd, respectivamente a 5V y GND en Arduino.

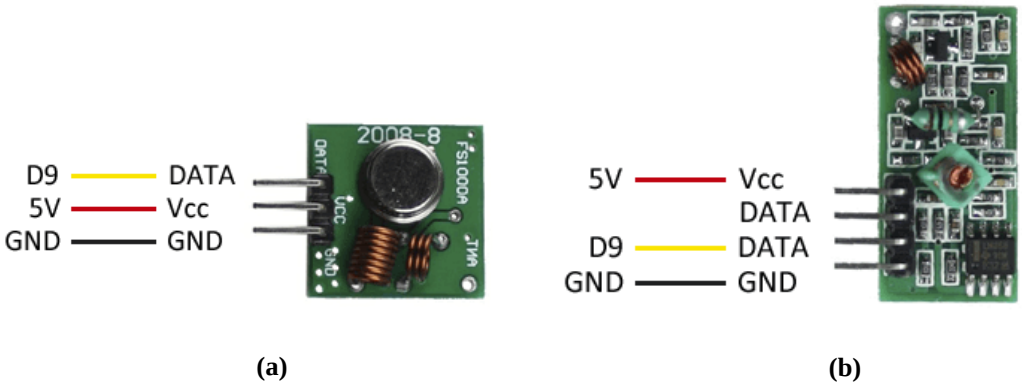


Fig. 2.15. Pines del módulo de RF donde (a) Transmisor, (b) Receptor.

Existen módulos a 315MHz, pero la frecuencia de operación será 433MHz. Ambas frecuencias pertenecen a bandas libres, por lo que su uso es gratuito.

Su alcance depende del voltaje con el que alimentemos el módulo y la antena que usemos. Con un voltaje de 5V y con la antena del módulo, el alcance no superará los 2 metros, más que suficiente para nuestro caso. En cambio, alimentando a 12V y con una antena de cobre de 16.5cm el rango puede alcanzar 300 metros.

La comunicación es simplex (canal único y unidireccional) y tienen baja velocidad de transmisión (típicamente 2400bps). Se realiza, básicamente, por modulación ASK. No disponen de filtro ni ID por hardware, por lo que si queremos una comunicación robusta tendremos que implementarlo por software.

Debido a su bajo precio y medio-largo alcance, emplearemos este tipo de comunicación RF 433MHz, cuyo datasheet estará en el anexo ii.

En [16] encontramos las funciones de la librería que debemos importar.

2.5.2. Comunicación inalámbrica entre Arduinos

Para realizar la comunicación, necesitaremos importar la librería ‘Virtual Wire’ desarrollada por Mike McCauley, en nuestro Arduino.

Para ello, nos descargamos y ubicamos la librería que estará en formato .ZIP. Abrimos el IDE de Arduino y seguimos los pasos dados en fig.2.16.

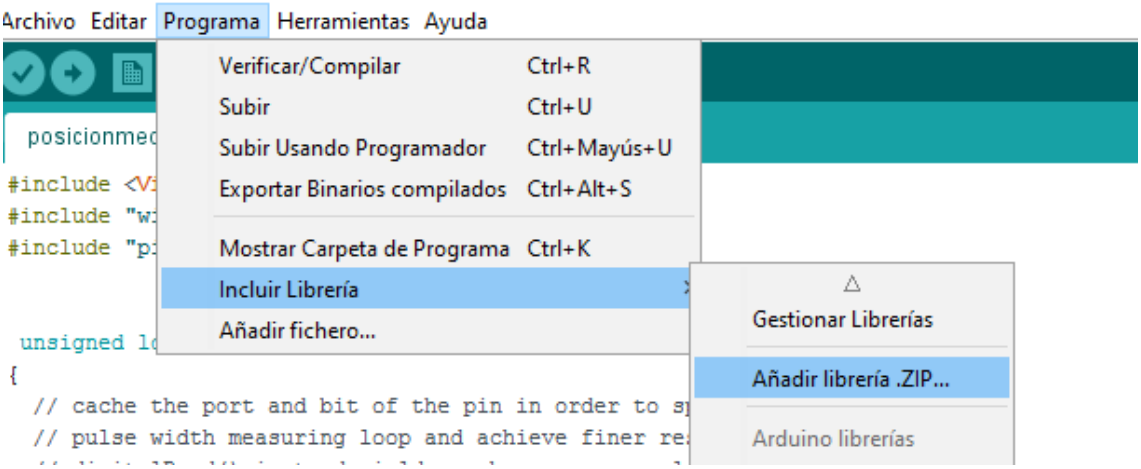


Fig. 2.16. Pasos para añadir una librería a Arduino.

Se mostrará la pantalla donde debemos indicar la ruta del archivo .ZIP que contiene la librería de arduino, lo seleccionamos y **ya deberíamos tener nuestra librería incluida.**

A continuación explicamos las funciones principales de la librería:

void vw_setup(uint16_t ***speed***)

Inicializa el software VirtualWire, como parámetro hay que indicarle la velocidad de operación, que representa los bits por segundo para la transmisión RF.

void vw_set_tx_pin(uint8_t pin)

Establece el pin por donde se va a transmitir los datos

void vw_set_rx_pin(uint8_t ***pin***)

Establece el pin por donde se va a recibir datos.

void vw_rx_start()

Empieza a escuchar los datos provenientes por el pin_rx, es necesario llamar a esta función para poder recibir los datos.

uint8_t vw_send(uint8_t * ***buf***,uint8_t ***len***)

Envia un mensaje con la longitud dada. La función termina rápido pero el mensaje será enviado en el momento establecido por las interrupciones. Dónde:

buf, es el puntero al vector para trasmitir.

len es el número de bytes a transmitir.

void vw_wait_tx()

Hace una pausa hasta que se trasmitan todos los datos.

uint8_t vw_get_message(uint8_t * buf, uint8_t * len)

Si un mensaje está disponible, lo almacena **buf** , devuelve true si la comprobación es correcta. **buf** es puntero a la ubicación para guardar los datos de lectura y **len** es un puntero a la cantidad de bytes disponibles de **buf.**

La librería proporciona ejemplos de código, que resulta aconsejable revisar, (anexo i).

Para ver más características del módulo 433Mhz, ver en el anexo ii.

2.6. Servo

En [17] encontramos las características y funcionamiento de los servos.

2.6.1. Características de los servos

Un servo es un tipo de actuador ampliamente empleado en electrónica. Esto significa que a diferencia de otros tipos de motores en los que controlamos la velocidad de giro, en un servo nos posiciona directamente en los ángulos deseados.

Normalmente, tienen un rango de movimiento limitado entre 0 y 180º, es decir, no son capaces de dar la vuelta por completo (de hecho disponen de topes internos que limitan el rango de movimiento).

Constan de un mecanismo reductor por lo que proporcionan un alto par y un alto grado de precisión, incluso décimas de grado. Pero por otro lado, las velocidades de giro frente a motores de corriente continua, son pequeños.

Respecto a la alimentación, los servos trabajan correctamente entre 4.8V a 7.2V, siendo 6V el valor más adecuado. Cuando las tensiones son inferiores el motor tiene menos fuerza y velocidad, y con tensiones superiores a 6,5V los servos empiezan a oscilar demasiado, lo cual los hace poco útiles.

La conexión en este caso también es sencilla. Se dispone de tres cables, dos de alimentación (GND y Vcc) y uno de señal (SIG).

El color de estos cables suele tener dos combinaciones:

– Marrón (GND), Rojo (Vcc) y Naranja (Sig)

– Negro (GND), Rojo (Vcc) y Blanco (Sig)

Fig. 2.17. Pines del servo.

2.6.2. Funcionamiento

Como hemos mencionado anteriormente, un servo está constituido por un motor de corriente continua, acoplado a un reductor para reducir la velocidad de giro, junto con la electrónica necesaria para controlar su posición.

Dispone de un potenciómetro unido al eje del servo, que permite al servo conocer la posición del eje. Un controlador integrado se encarga de ajustar actuar sobre el motor para alcanzar la posición deseada, a partir de la información anterior.

Mediante la transmisión de una señal pulsada con periodo de 20ms, podemos posicionar al servo en la posición que queramos. El ancho del pulso determina la posición del servo.

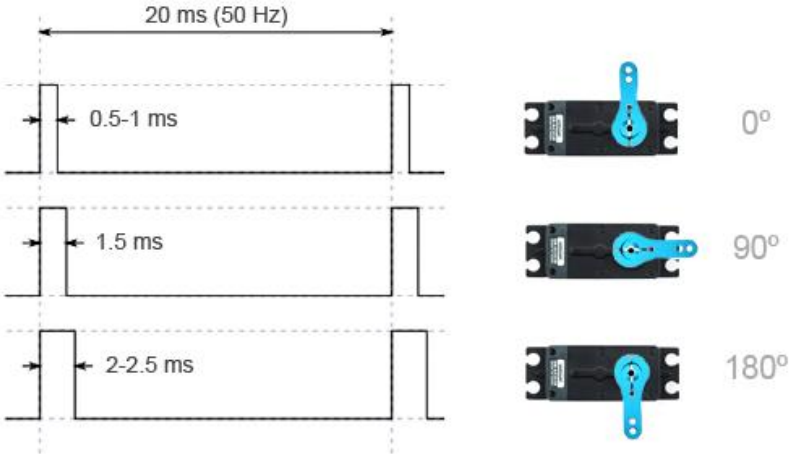


Fig. 2.18. Posición del servo según ancho del pulso.

La relación entre el ancho del pulso y el ángulo depende del modelo del motor. Pero por lo general, en todos los modelos:

- Un pulso entre 500-1000 ms corresponde con 0º.
- Un pulso de 1500 ms corresponde con 90º (punto neutro)
- Un pulso entre 2000-2500 ms corresponde con 180º

Por tanto, variando la señal en microsegundos podemos disponer de una precisión teórica de 0.18-0.36º, siempre que la mecánica del servo acompañe.

Emplearemos un servo SG-5010, cuyas características pueden verse en el anexo ii.

CAPÍTULO 3.
PARTE EXPERIMENTAL

3.1. Desarrollo del proyecto

El objetivo principal de nuestro proyecto, es dotar de dispositivos de localización a un robot móvil, carente de dispositivo GPS.

Tras la presentación de los elementos que utilizaremos para ello, dessarrollaremos a continuación los diferentes montajes que fueron necesarios para determinar el dispositivo de localización:

3.1.1. Montaje inicial (un ultrasonido)

El primer montaje, se centrará en estudiar el correcto funcionamiento del dispositivo SR04 como sistema estandard de medida de distancias, en la placa arduino UNO.

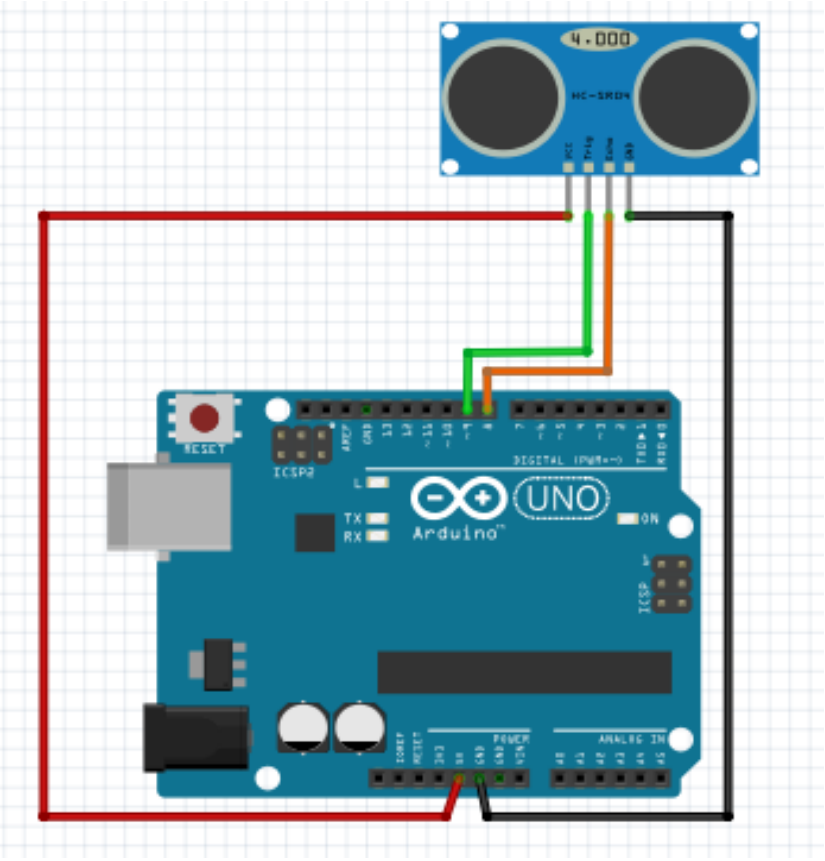


Fig. 3.4. Montaje inicial (un ultrasonido).

Verificamos el funcionamiento del montaje conectando el dispositivo a un osciloscopio. El canal 1 trasmitirá la señal de la placa (señal que está procesada) y el canal 2 directamente del sensor (tren de pulsos enviados), como se ve a continuación:

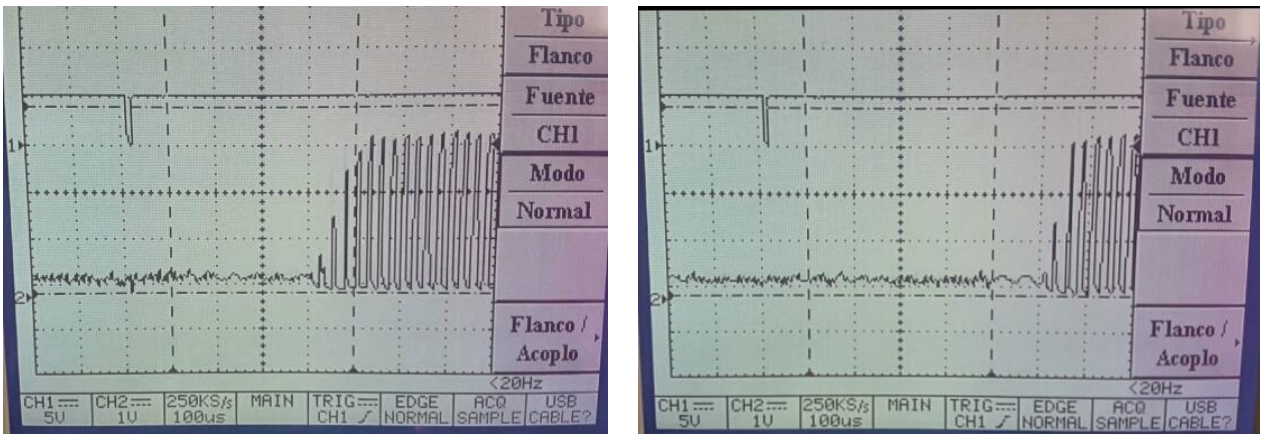


Fig. 3.2. Verificación en osciloscopio del funcionamiento del dispositivo SR04.

A medida que la distancia a medir es mayor o menor, el tren de pulsos del ECHO se aleja o se acerca.

A partir del programa realizado en Arduino, (ver anexo i), abrimos el monitor serial y vemos que el sistema de medida funciona:

Fig. 3.3. Medida del montaje del dispositivo SR04.

3.1.2. Segundo montaje (dos ultrasonidos misma placa)

Ahora conectamos dos sensores ultrasónicos a la misma placa.

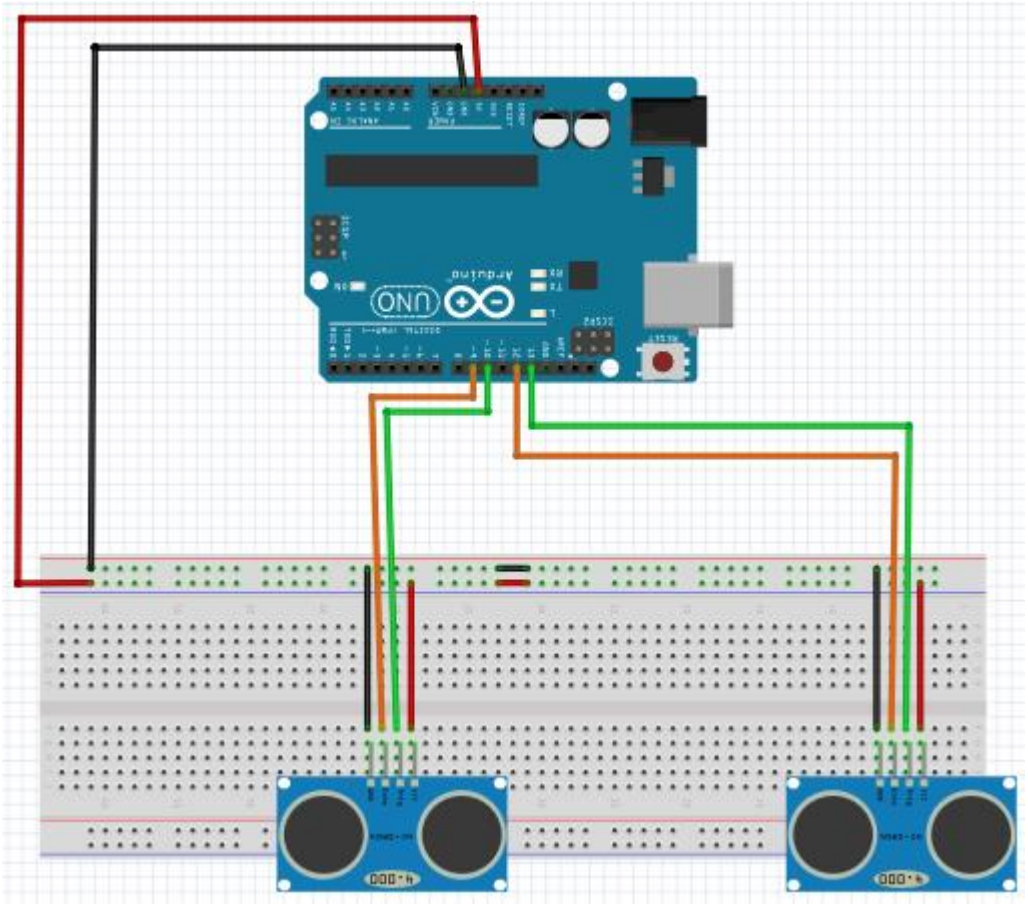


Fig. 3.4. Segundo montaje (dos ultrasonidos misma placa).

De esta manera, obtenemos el mismo resultado que en el montaje inicial pero con dos ultrasonidos. Cada uno mide de forma totalmente independiente aunque comparten el mismo código (ver anexo i).

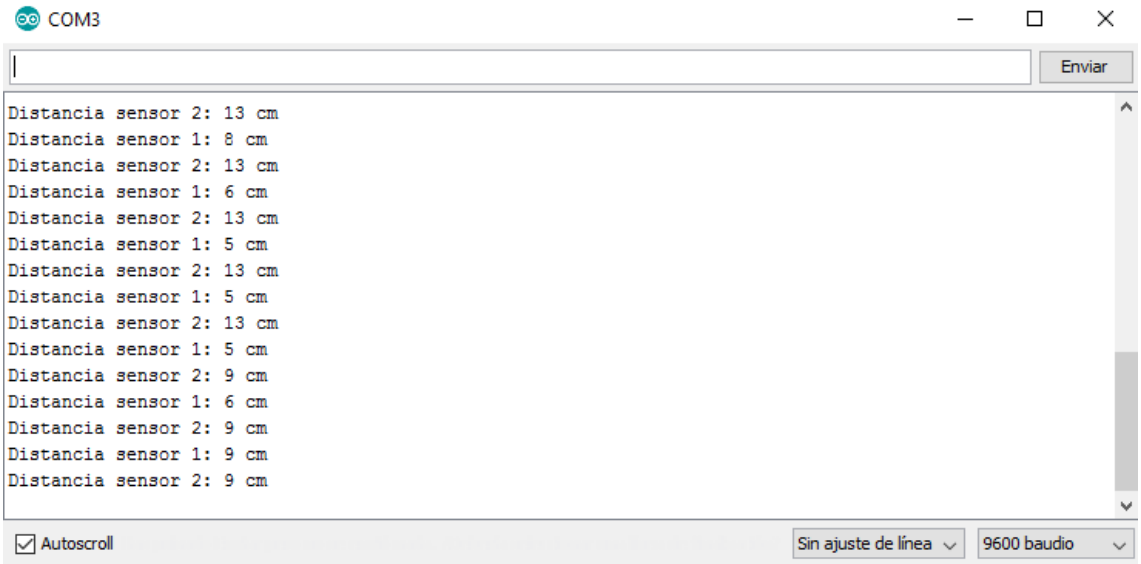


Fig. 3.5. Medida del montaje de dos dispositivos SR04.

3.1.3. Tercer montaje (emisor y receptor misma placa)

A la hora del tercer montaje, nos dimos cuenta que dado nuestro objetivo, necesitábamos realizar un cambio tanto en su software como en su hardware. Ese cambio consistirá en que uno actúe únicamente como transmisor de señal, mientras que otro, actúe como receptor.

El cambio para que funcione como emisor es directo en el propio programa, activando únicamente el pin TRIG. Por otro lado, para lograr nuestro receptor, hemos tenido que centrarnos en el hardware, en el que hizo falta soldar un cable para sacar la salida del pin ECHO directamente, (explicado anteriormente en el capítulo 2). Como nos interesa que no se supere los 5V de tensión, conectaremos la salida del receptor a un limitador de tensión, compuesto por una resistencia de 100 ohm y un diodo zener polarizado a la inversa.

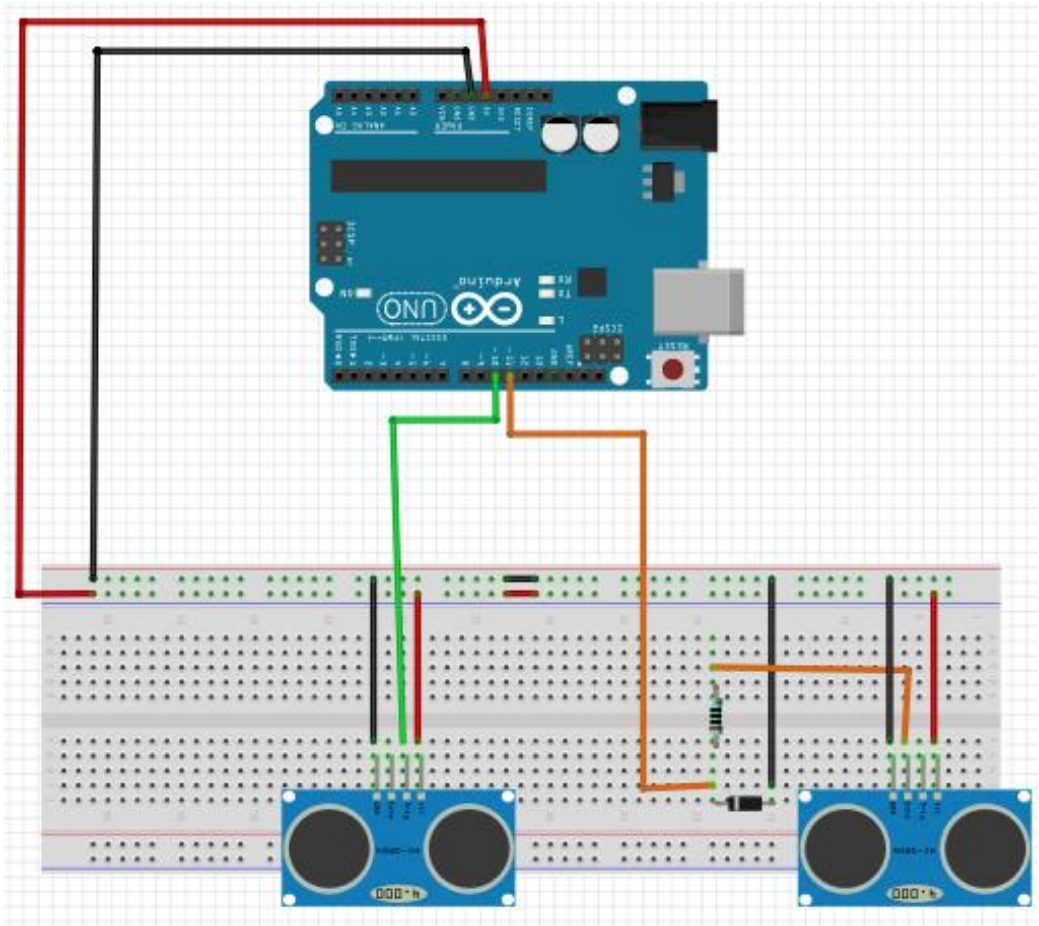


Fig. 5.6. Tercer montaje (emisor y receptor misma placa).

Una vez realizado los cambios mencionados, se produce un cambio en el timer empleado para el cálculo de la variable *tiempo*. Esto origino diversos problemas en búsqueda, entendimiento de distintos timers, e intentos de modificación. Por ello, pasamos a la comunicación entre sensores, que veremos en el cuarto montaje.

3.1.4. Cuarto montaje (emisor y receptor distinta placa)

La comunicación se llevará a cabo a partir de módulos de radiofrecuencia de 433 mHz. Como los transmisores y receptores en este caso se encuentran separados, cada uno de ellos deberá tener su propio microcontrolador.

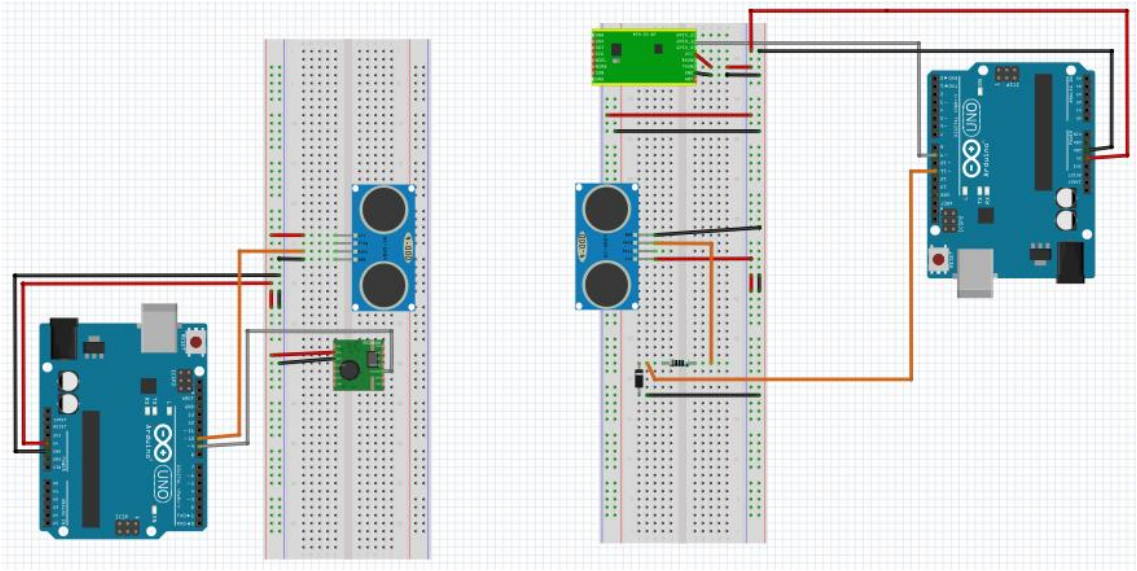


Fig. 3.7. Cuarto montaje (emisor, derecha; receptor, izquierda, distinta placa).

Llevaremos a cabo algunas pruebas, para verificar la comunicación de forma correcta y para entender el funcionamiento de los módulos entre sí. Utilizamos una herramienta auxiliadora, Realterm, para poder ver el serial del emisor, (en arduino), y receptor, (en Realterm), por separado al mismo tiempo. Presenta una interfaz, donde se muestra un menú con el que crearemos la conexión:

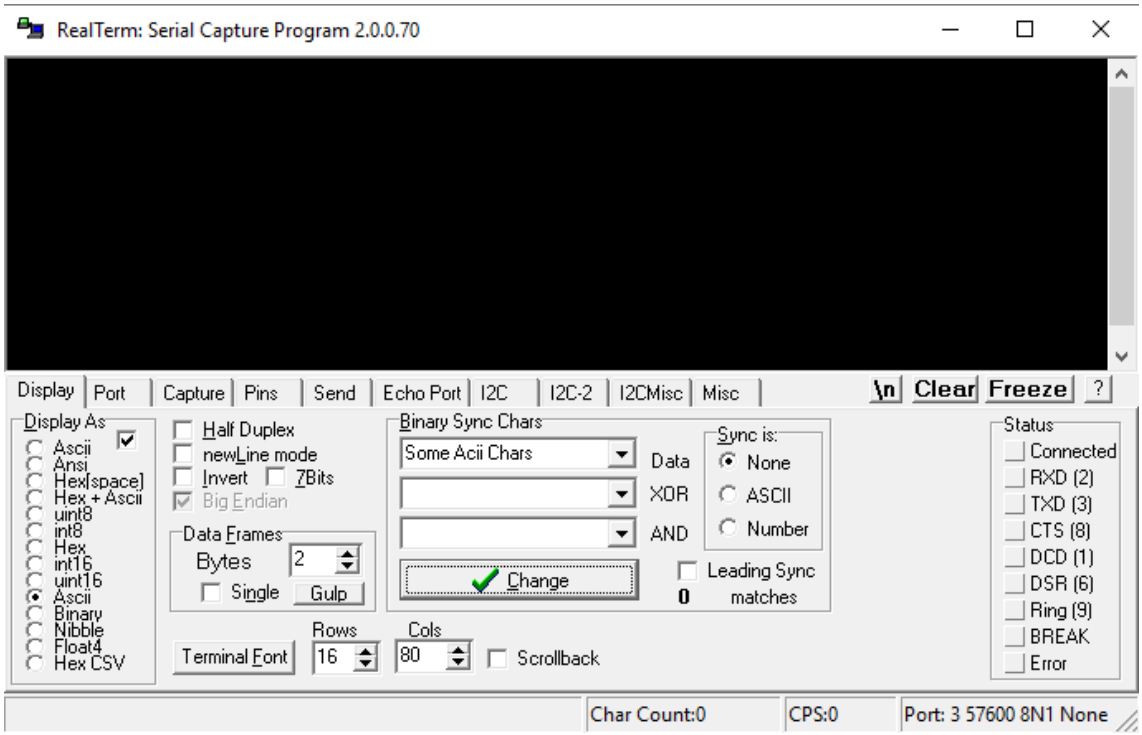


Fig. 3.8. Interfaz Realterm.

En el apartado Display, seleccionamos Ascii, y como Binary Sync Chars, Some Acii Chars. A continuación, pasamos al puerto del TRIG, llamado Port.

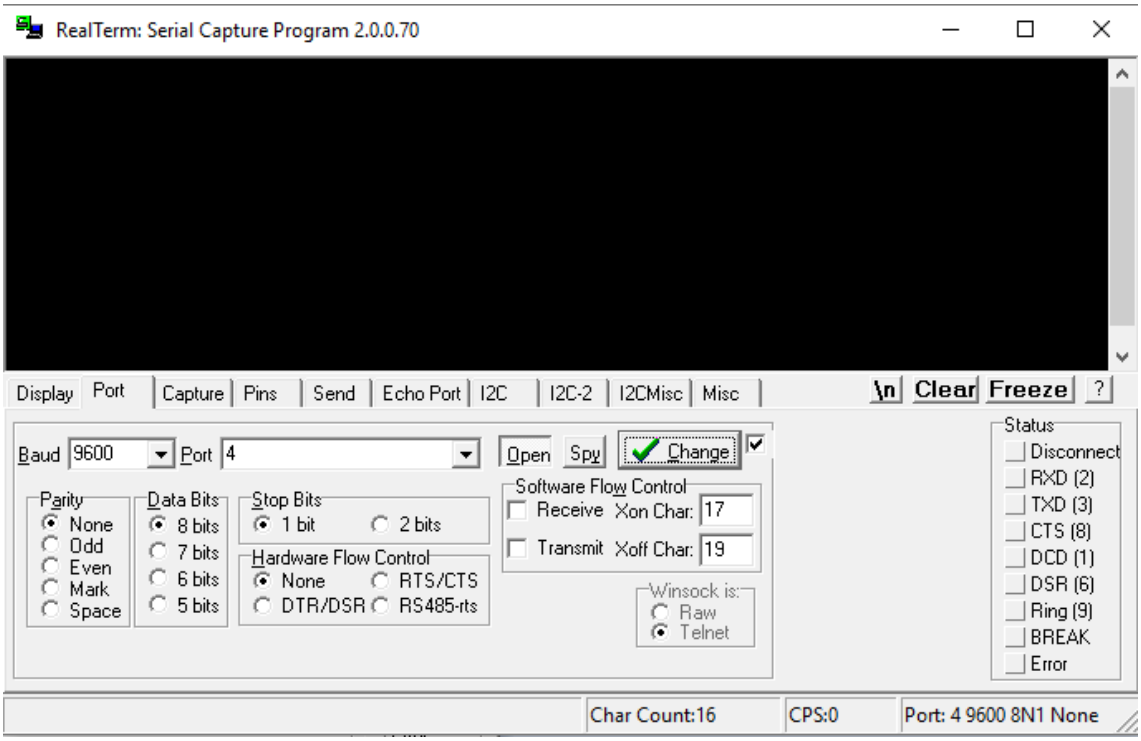


Fig. 3.9. Selección de puerto emisor.

En la pestaña Port, cambiamos los parámetros que vienen por defecto por:

- La velocidad de baudios por 9600.
- El puerto correspondiente, en este caso el 4.

Conectado con el puerto equivalente a nuestro emisor, el cual monitorizamos desde Arduino:

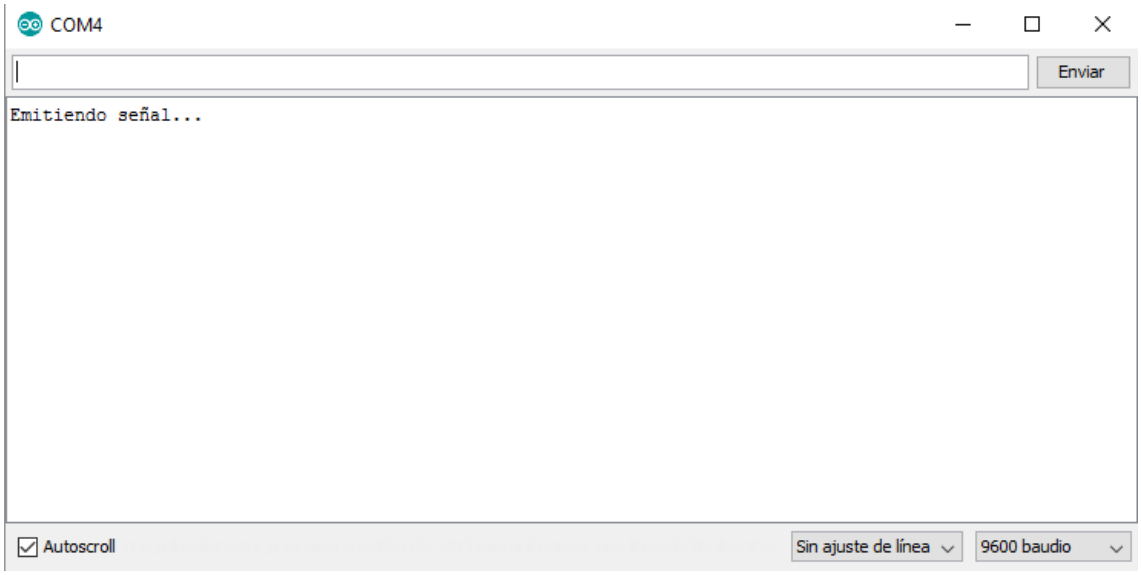


Fig. 3.10. Monitor serie de Arduino. Emisor.

Pasamos a la conexión con el receptor:

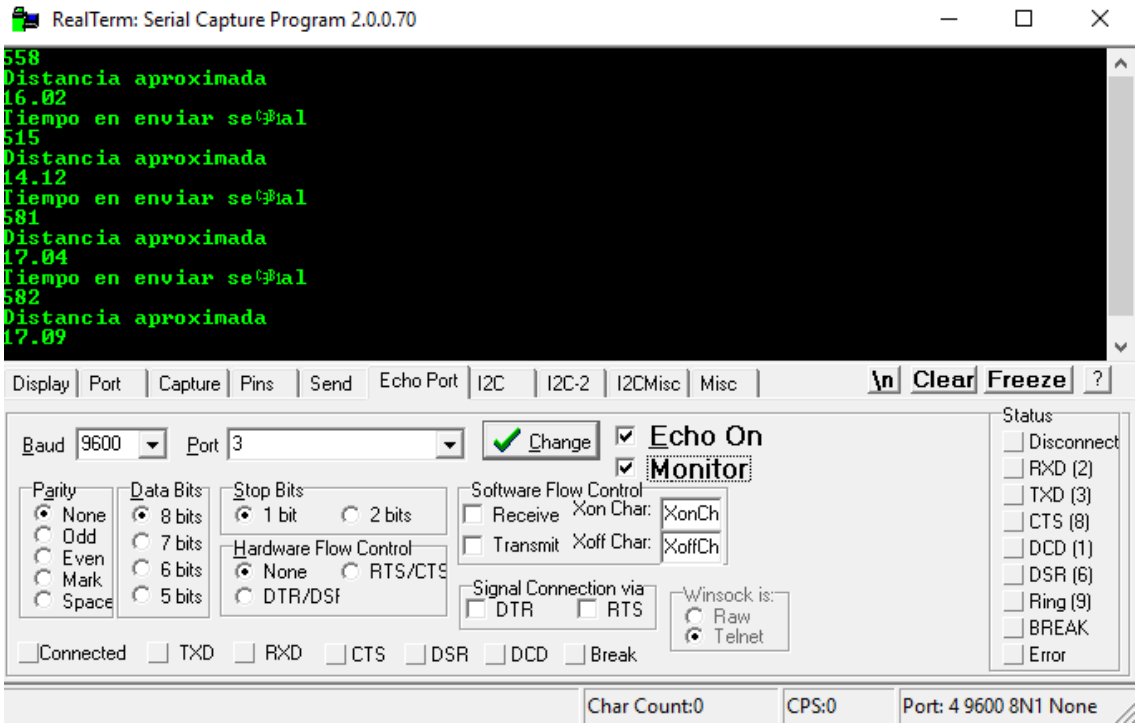


Fig. 3.11. Selección de puerto receptor.

En la pestaña Echo Port, cambiamos de nuevo los parámetros que vienen por defecto por:

- La velocidad de baudios por 9600.
- El puerto correspondiente, en este caso el 3.
- Activamos Echo On y el Monitor para poder recibir el mensaje.

Como vemos, se establece correctamente la comunicación. En la parte emisora, transmitimos la señal de RF junto con la señal ultrasónica, que será lo que estará esperando el receptor por recibir. Gracias a la herramienta Realterm, podemos monitorizar el puerto receptor al mismo tiempo que monitorizamos el puerto emisor en Arduino.

Volviendo a la búsqueda del timer, nos fijamos en el pulseIn, del cual modificamos el código fuente para hacerlo funcionar con pulsos como los que emite el ultrasonido. Declaramos un nuevo 'timer' llamado *mipulseIn* en nuestro programa, (parte del receptor), quedando de la siguiente forma:

```
#include "wiring_private.h"

#include "pins_arduino.h"

unsigned long mipulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
```

```
{
 //Cache el puerto y el bit del pin para acelerar el bucle de medición del ancho de pulso y lograr
 //una resolución más fina. Llamar a digitalRead () en su lugar produce una resolución mucho
 //mejor.
 uint8_t bit = digitalPinToBitMask(pin);
 uint8_t port = digitalPinToPort(pin);
 uint8_t stateMask = (state ? bit : 0);
 unsigned long width = 0; // Mantener la inicialización fuera del tiempo de área crítica

 //Convertir el tiempo de espera de microsegundos a un número de veces a través del bucle
 //inicial; Toma 16 ciclos de reloj por iteración.
 unsigned long numloops = 0;
 unsigned long maxloops = microsecondsToClockCycles(timeout) / 16;

//Comentamos la parte del programa que no nos interesa

  // wait for any previous pulse to end
  //while ((*portInputRegister(port) & bit) == stateMask)
     //if (numloops++ == maxloops)
        //return 0;

  // wait for the pulse to start
  //while ((*portInputRegister(port) & bit) != stateMask)
     //if (numloops++ == maxloops)
        //return 0;

 // Esta será la parte del pulseIn que usaremos, donde cuenta el tiempo hasta que haya un pulso
 //positivo
 while ((*portInputRegister(port) & bit) == stateMask) {
  if (numloops++ == maxloops)
    return 0;
  width++;
 }

 // Convertir la lectura en microsegundos. El bucle se ha determinado que tiene 20 ciclos de
 // reloj de largo y tiene alrededor de 16 relojes entre el borde y el inicio del bucle. Habrá algún
 // error introducido por los manejadores de interrupción.
 return clockCyclesToMicroseconds(width * 21 + 16);
}
```

Con el timer actualizado, para calcular de nuevo la distancia entre dispositivos, deberemos hallar nuevos parámetros. Para ello, tuvimos que recurrir al empleo de un sondeo de medidas, obtenidas de la ecuación inicial.

$$distancia = (0.0175 * tiempo);$$

Ecuación 3.1. Cálculo de la distancia en Arduino.

Esas medidas fueron recogidas en relación a la distancia real entre nodos y el tiempo transcurrido en la obtención de la medida. Elaboramos unas tablas donde a partir de varias medidas de una misma distancia, realizamos una media de los tiempos de envío de la señal.

DISTANCIA REAL (cm)	TIEMPO (ms)
4	287
20	668.8
40	1123.2
65	1670.4
100	2479.2
130	3098.6
160	3805.4
200	4703.6
240	5600.6
280	6510.6
300	6908.8
370	8412.4
400	9085.4
420	9520.8
440	9940.2
470	10605.2
500	11274

Tabla 3.1. Relación distancia real y tiempo.

Partiendo de esta tabla, con ayuda de la herramienta Matlab, podemos hallar la constante que los relaciona, además de un parámetro que marca la diferencia entre obtener una medida parcialmente acertada, a una medida bastante aproximada a la real.

Partiremos de la expresión:

$$\mathbf{Y = mX + b}$$ **Ecuación 3.2.** Ecuación de la recta.

donde Y sería la distancia real, X el tiempo medido, m nuestra cte. y b nuestro otro parámetro. En la interfaz Matlab declaramos:

X= [287, 668.8, 1123.2, 1670.4,……..]

Y= [4, 20, 40, 65,…….]

Lo siguiente es hallar los parámetros a partir de:

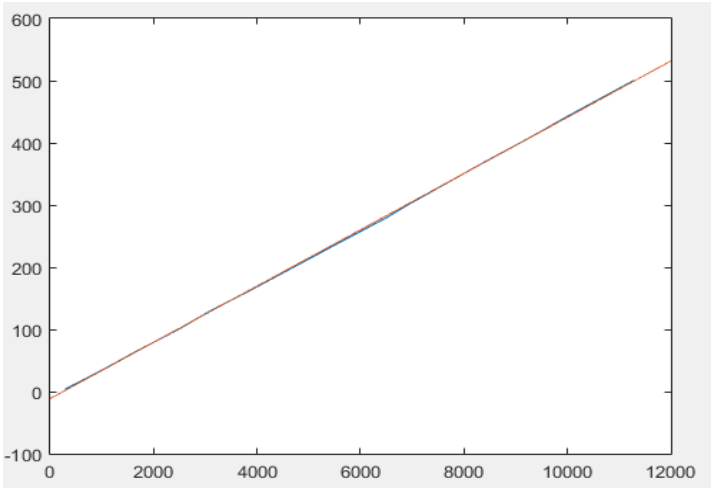
[A, B] = polyfit(X,Y,1)

Obtenemos: A = 0.0443 -8.6947

,donde 0.0443 es nuestra constante, y -8.6947 sería el otro parámetro.

Para verlo gráficamente, x = [0:1:12000], realizamos el siguiente plot:

plot (Y,X,x,x*A(1) + A(2))



Gráfica 3.1. Relación distancia real y tiempo.

La nueva ecuación vendrá dada de la siguiente forma:

$$distancia = (0.0443 * tiempo - 8.6947);$$

Ecuación 3.3. Expresión final cálculo distancia.

Verificamos a continuación que realiza la medida correctamente comparándola con la medida real. La medida experimental vendrá dada a partir de una media de varias medidas:

DISTANCIA REAL (cm)	DISTANCIA EXPERIMENTAL (cm)
4	4.28
20	19.65
40	40.02
65	64.47
100	99.70

Tabla 3.2. Comparación distancia real y distancia ecperimental.

Los valores se aproximan bastante al real. Ahora, el siguiente paso será el despliegue de otros dos dispositivos que actuarán de emisores.

3.1.5. Quinto montaje (cuatro sensores)

Para efectuar el método de localización elegido, deberemos implementar otros dos dispositivos ultrasónicos que actuarán igualmente como emisores.

La medida podría hacerse con dos transmisores, pero te plantea la situación de dos posiciones distintas del receptor. Es el tercer sensor el que te indica exactamente si estas en un lado o en otro, por eso necesitaremos tres sensores que actúen como emisores. Al igual que en el primer caso, simplemente activamos únicamente los pin TRIG de cada uno. El montaje sería el siguiente:

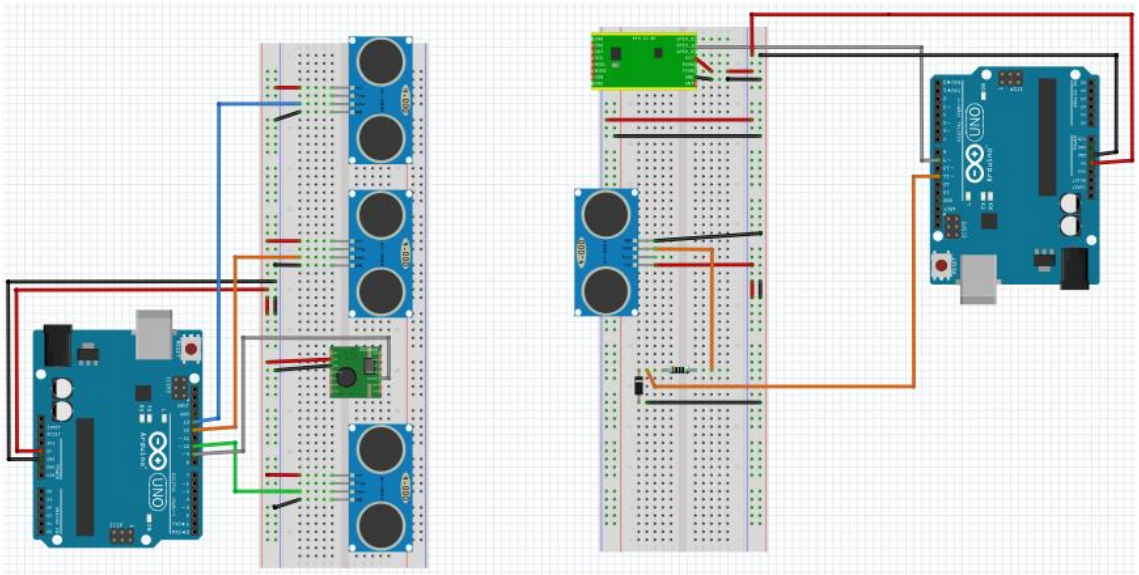


Fig. 3.12. Quinto montaje (cuatro sensores).

No hace falta verificar continuamente la comunicación con Realterm, podemos ver los resultados directamente en el monitor de Arduino:

Fig. 3.13. Medidas del quinto montaje.

3.1.6. Sexto montaje (trilateración)

Tras esto, definimos las coordenadas de cada uno de nuestros nodos emisores para emplear las ecuaciones de trilateración. Estas serán definidas en el programa receptor que se encargará de interpretar la información obtenida por los transmisores para hallar las coordenadas del nodo receptor. Conectaremos en tres protoboards diferentes, cada emisor.

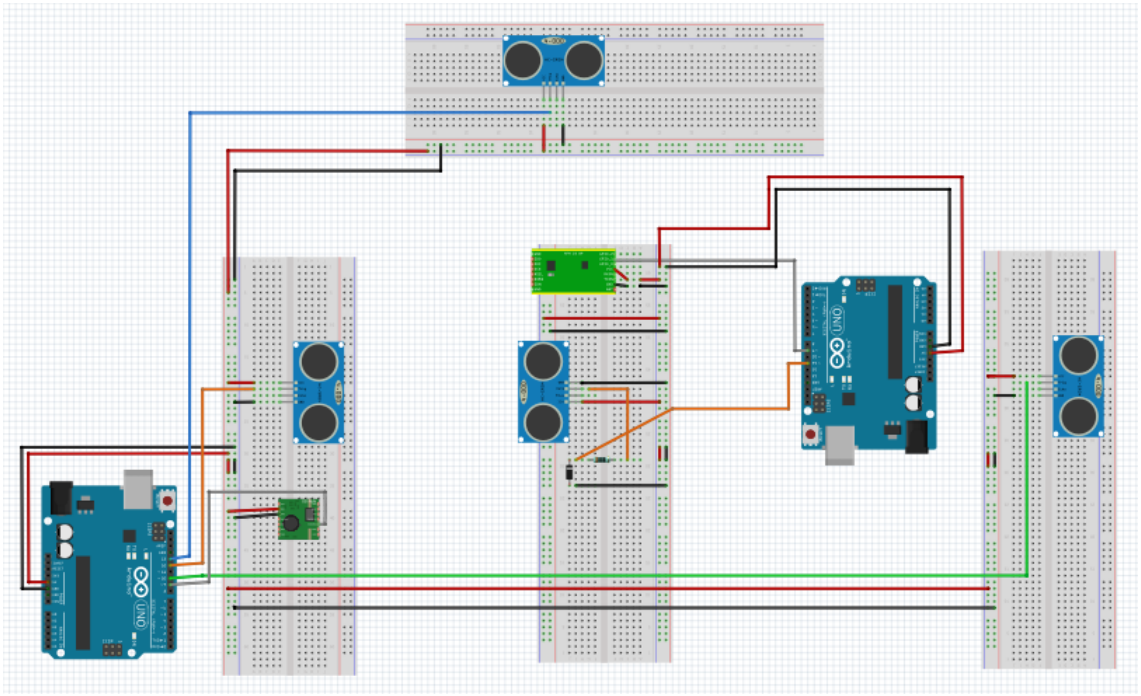


Fig. 3.14. Sexto montaje (trilateración).

Para nuestro caso, basándonos en las ecuaciones de distancia para cada nodo, siendo $\boldsymbol{d_i}$ las distancias que hay desde nuestros emisores, cuyas coordenadas $(\boldsymbol{x_i}, \boldsymbol{y_i})$, hasta nuestro receptor, obtenemos nuestra posición (x, y). Partiendo entonces de las siguientes tres ecuaciones:

$$d1^2 = (x - x1)^2 + (y - y1)^2$$

$$d2^2 = (x - x2)^2 + (y - y2)^2$$

$$d3^2 = (x - x3)^2 + (y - y3)^2$$

Estas serían nuestras soluciones para el sistema:

$$x = \frac{d1^2 - d2^2 + x2^2}{2 * x2}$$

$$y = \frac{d1^2 - d3^2 + x^2 + (x - x3)^2 + y3^2}{2 * y3}$$

Ecuación 3.4. Soluciones a nuestro sistema.

Utilizando la herramienta ‘desmos’, hicimos una representación ejemplo, donde a partir de unos parámetros dados situamos a nuestro receptor:

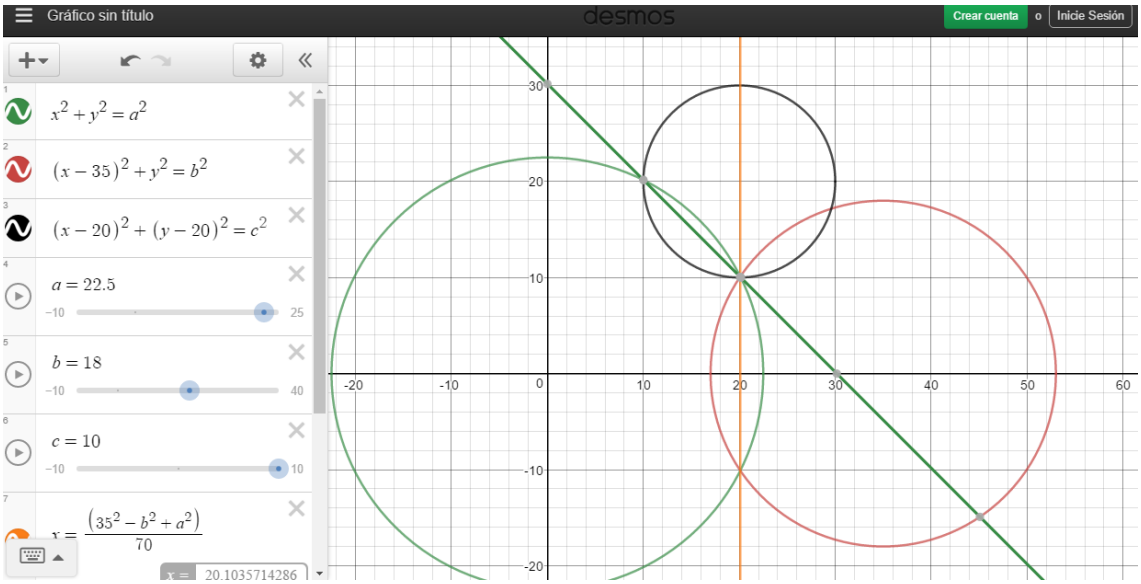


Fig. 3.15. Representación trilateración.

Conseguida la posición del objeto, implementaremos un servo, el cual se encargará de hacer girar nuestro receptor para recibir mejor las medidas de cada uno de los nodos emisores.

3.1.7. Séptimo montaje (servo un receptor)

Inicialmente, partiríamos de dos placas arduino UNO. Al implementar nuestro servo, este interfiere usando mismos recursos que nuestros módulos de RF, en concreto del timer. Como solución, sustituiremos la placa receptora que es donde situaremos al servo, por una placa arduino MEGA.

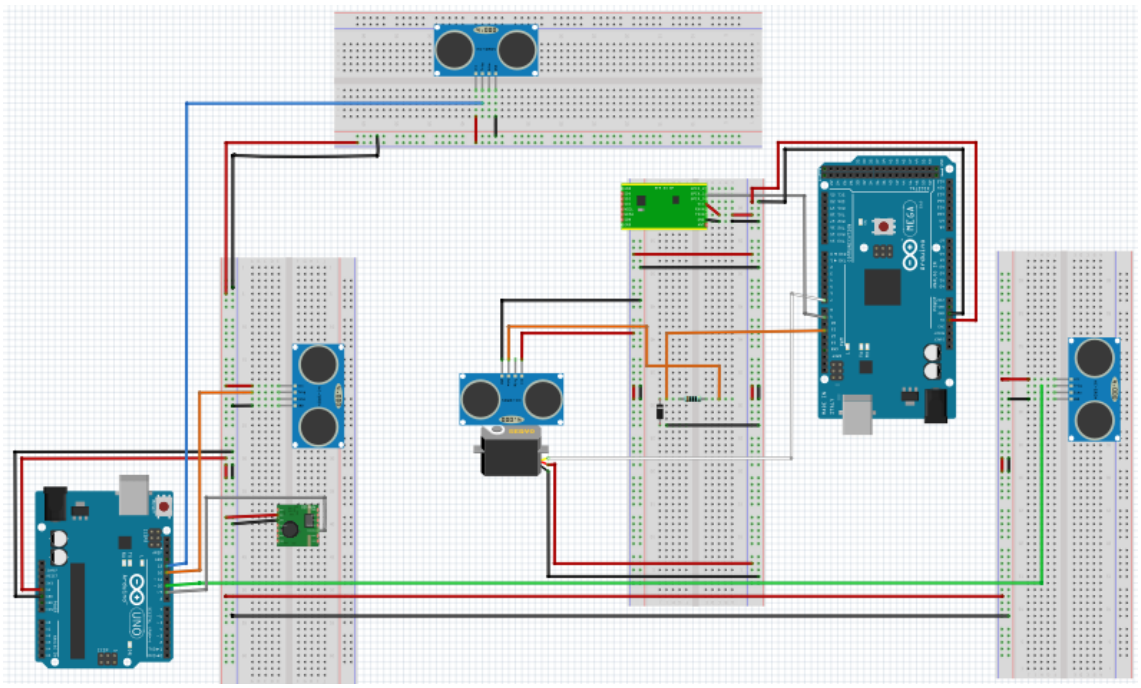


Fig. 3.16. Sétimo montaje (servo un receptor).

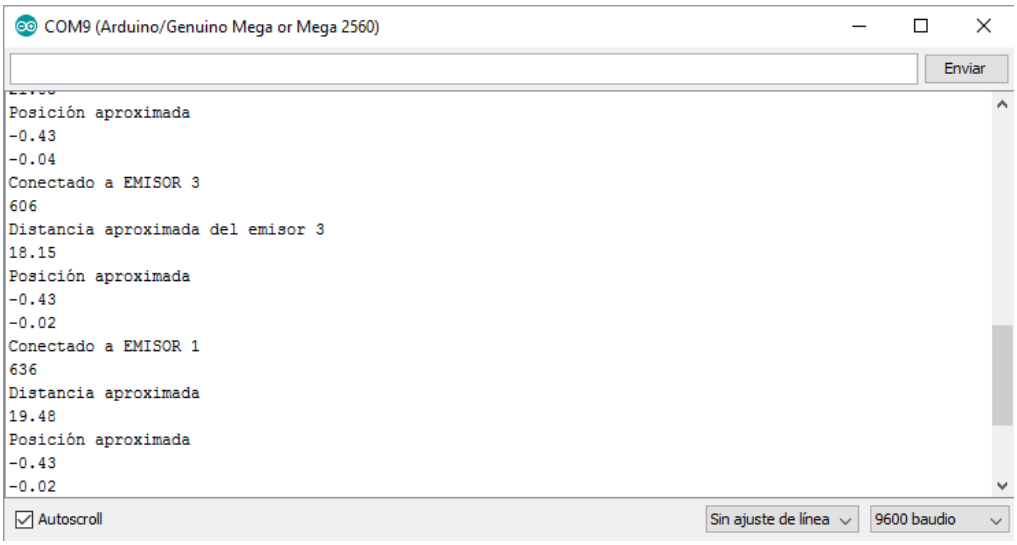


Fig. 3.17. Medidas del séptimo montaje.

Observando los resultados obtenidos, el servo al estar limitado a 180º, no abarca todos los sensores. Además, la medida de la posición aun es errónea, ya que al recibir cualquier medida de cualquier emisor, calcula directamente las coordenadas X e Y.

3.1.8. Octavo montaje (servo dos receptores)

Para poder barrer los 180º restantes, acoplamos otro dispositivo ultrasónico que actuará igualmente como receptor. En este caso, obtenemos información de medida que con un barrido de sólo 180º hubiera sido imposible.

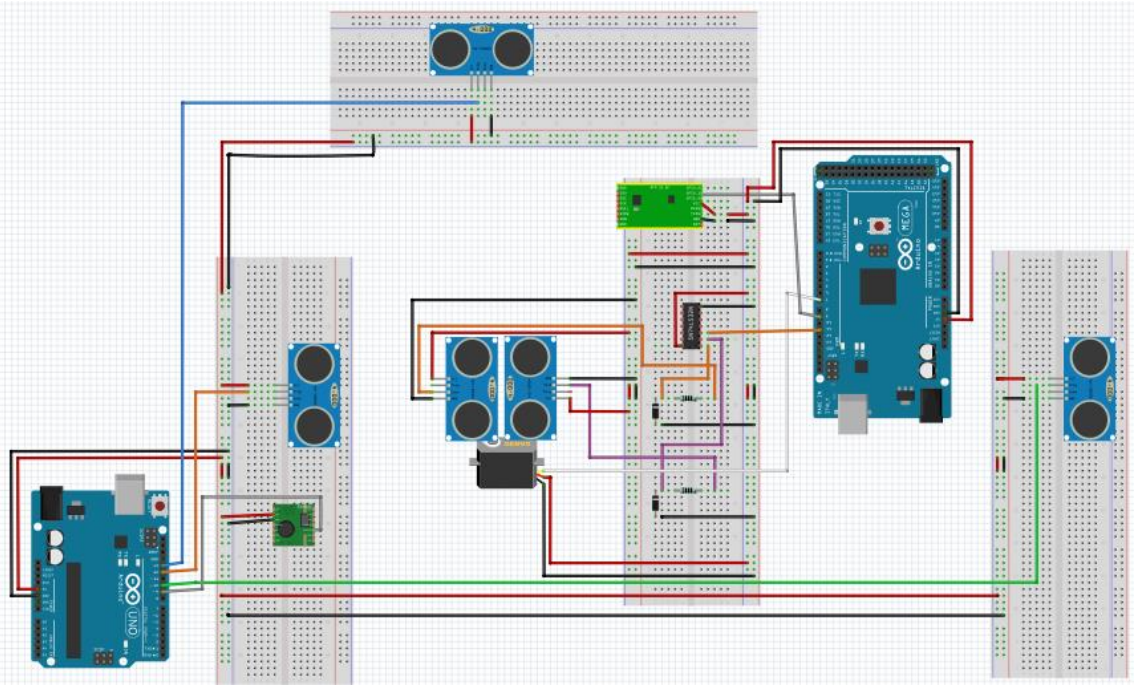


Fig. 3.18. Octavo montaje (servo dos receptores).

A la hora de implementar el nuevo dispositivo, añadimos una puerta lógica OR, que se encargará de sumar ambas salidas cada vez que ésta reciba una medida.

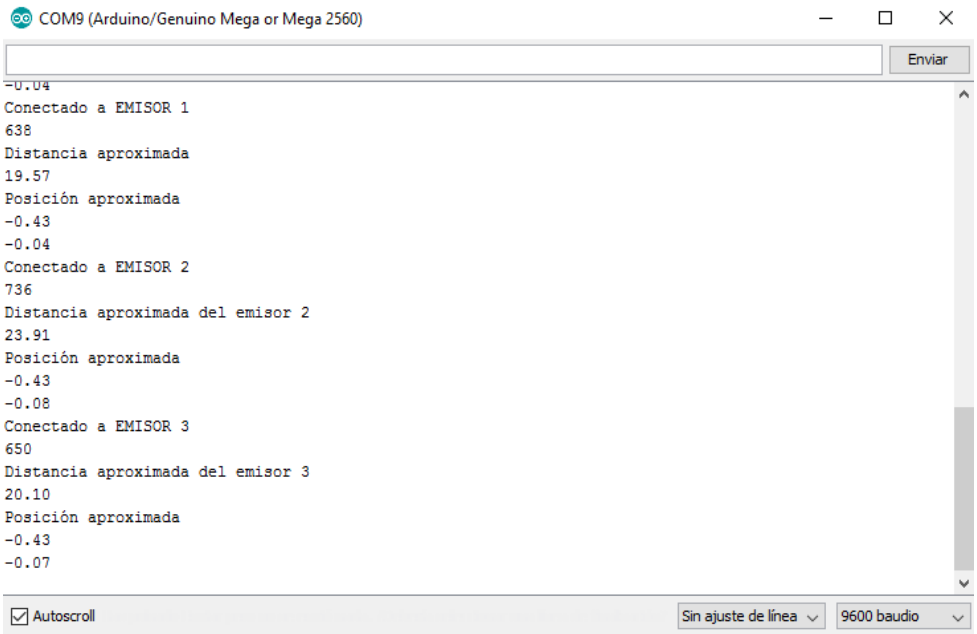


Fig. 3.19. Medidas del octavo montaje (servo dos receptores).

La medida de la posición aun es errónea, pero en el barrido consigue recibir las señales de los tres emisores.

3.1.9. Montaje final (cálculo matemático)

Para terminar, realizaremos un programa en el cual se miden las distancias de cada emisor al receptor, eligiendo un rango de posibles soluciones válidas. Después de esto, dichos valores nos darán la posición aproximada de nuestro receptor tras realizar las ecuaciones de trilateración.

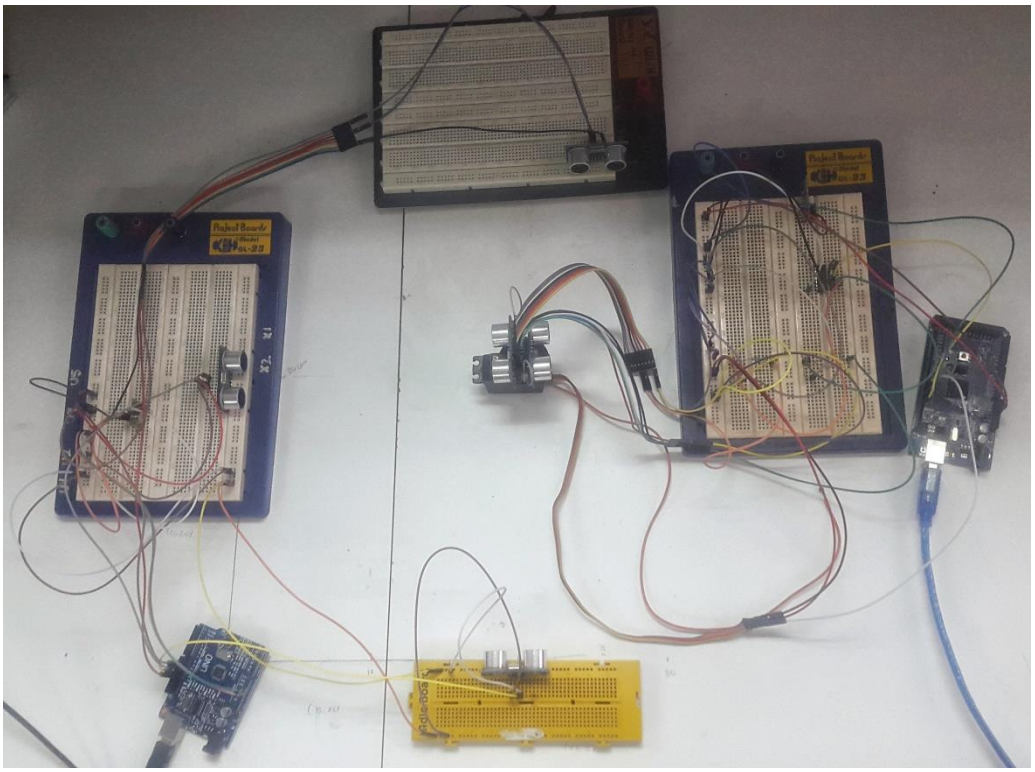


Fig. 3.20. Montaje final (cálculo matemático).

3.2. Observaciones generales

Finalizado los montajes que se han llevado a cabo en el apartado anterior, determinamos una serie de parámetros de interés a la hora de realizar una buena localización a partir del sistema diseñado. Observamos que:

Como hemos comentado, tras los cambios en el hardware y software de nuestros dispositivos, la distancia máxima será superior a 4 metros, que es la distancia máxima del sensor normal. Se ha corroborado experimentalmente que supera este límite, aunque no se ha llegado a medir la distancia máxima completa.

Una de las limitaciones del prototipo es el tiempo de medida el cual podría ser más rápido optimizando el código de procesamiento.

Por último, a la hora de calcular la posición del objeto, cabe a destacar que el error de medida es menor de lo esperado, de pocas décimas o incluso algún cm arriba o debajo, como observamos en la Tabla 3.2.

CAPÍTULO 4.
CONCLUSIONES Y LÍNEAS ABIERTAS

4.1. Conclusions

This project has been a learning process from the beginning. Ultrasonic sensors are quite common in our day to day life, but not everyone knows their operation and characteristics. Also I had never used the language of the Arduino platform which is quite simple and with which I have not had too much difficulty. The communication between plates from radiofrequency modules was also something new for me.

Overall, the results have been good. The measurements made by the sensors are quite close to the actual ones. All this, result of the changes in both the software and the hardware that were carried out, that allowed us to give a different use to the ultrasonic sensors.

4.2. Open lines

As future lines, the system could be used in more complex mobile robots and in a larger environment, as much as the radiofrequency modules allow us, since we will want to keep the initial components to continue the low cost.

In addition to locating our robot, we could add a program that is responsible for telling us its orientation, so we would have a fairly complete localization system.

4.1. Conclusiones

Este proyecto ha sido un proceso de aprendizaje desde el primer momento. Los sensores ultrasónicos son bastante comunes en nuestro día a dia, pero no todo el mundo conoce su funcionamiento y características. Además nunca había empleado el lenguaje de la plataforma Arduino el cual es bastante sencillo y con el que no he tenido demasiada dificultad. La comunicación entre placas a partir de módulos de radiofrecuencia también era algo nuevo para mí.

De forma general, los resultados han sido buenos. Las medidas hechas por los sensores son bastante aproximadas a las reales. Todo esto, resultado de los cambios tanto en el software como en el hardware que se llevaron a cabo, que nos permitieron darle un uso diferente a los sensores ultrasónicos.

4.2. Líneas abiertas

Como futuras líneas, el sistema podría emplearse en robots móviles más complejos y en un entorno mayor, tanto como los módulos de radiofrecuencia nos permitan, ya que querremos mantener los componentes iniciales para continuar con el bajo coste.

Además de localizar nuestro robot, podríamos añadir un programa que se encargase de decirnos su orientación, de esta forma tendríamos un sistema de localización bastante completo.

CAPÍTULO 5.

PRESUPUESTO

5.1. Presupuesto aproximado

Se ha elaborado un presupuesto acorde a los materiales empleados en nuestro proyecto, en el que se ve reflejado el bajo coste que este supone.

MATERIAL	CANTIDAD	PRECIO
PLACA ARDUINO UNO	2	12 €
PLACA ARDUINO MEGA	1	15 €
SENSOR ULTRASÓNICO	5	12.50 €
PROTOBOARD	4	8.40 €
RESISTENCIA 220 OHM	2	0.10 €
DIODO ZENER	2	0.20 €
CABLES CONEXIÓN MACHO A MACHO	40	2.50 €
KIT MÓDULO DE RADIO FRECUENCIA 433MHZ	1	1.50 €
SERVO SG-5010	1	3.07 €
MANO DE OBRA*	1	350 €
**en relación a la ejecución del proyecto.*	TOTAL	405.27 €

Fig. 5.1. Lista materiales-cantidad-precio.

Precios individuales, tendríamos:

PLACA ARDUINO UNO	1	6 €
SENSOR ULTRASÓNICO	1	2.50 €
PROTOBOARD	1	2.10 €
RESISTENCIA 220 OHM	1	0.05 €
DIODO ZENER	1	0.10 €

Fig. 5.2. Listado precio individual.

BIBLIOGRAFÍA

Localización

[1] Ponente D.Manuel Ocaña Miguel, "Sistemas de localización'". Universidad de Alcalá, 2006.

http://www.alcabot.com/alcabot/seminario2006/SEM06_Localizacion.pdf

[2] M.Sc. Kryscia Daviana Ramírez Benavides, "Localización". Universidad de Costa Rica.

http://www.kramirez.net/Robotica/Material/Presentaciones/Localizacion.pdf

Trilateración y triangulación

[3] Eva M. García Polo, "Técnicas de Localización en Redes Inalámbricas de Sensores". Universidad de Castilla-La Mancha, 2008.

http://www.dsi.uclm.es/personal/EvaMariaGarcia/docs/2008-Curso%20Verano.pdf

[4] Santiago Elvira Díaz, "Sistema de localización y orientación basado en recepción diferencial de ultrasonidos". Universidad Autónoma de Madrid, julio 2011.

https://repositorio.uam.es/bitstream/handle/10486/12776/61521_Elvira_Diaz_Santiago%5b2%5d.pdf?sequence=1

[5] MATHEMATICS, "Find X location using 3 known (X, Y) location using trilateration".

https://math.stackexchange.com/questions/884807/find-x-location-using-3-known-x-y-location-using-trilateration

Sensores de posición

[6] David Vara Rodríguez, "Sistemas para determinar la posición y orientación de herramientas quirúrgicas en operaciones de cirugía laparoscópica". Universidad de Valladolid, julio 2014.

https://uvadoc.uva.es/bitstream/10324/12884/1/TFG-P-161.pdf

Sensor ultrasónico

[7] Mª Esther Gilaberte Sanz, "Implementación de sensores de ultrasonidos en un sistema autónomo de tiempo real". 18 de marzo de 2003.

http://docplayer.es/19432332-Implementacion-de-sensores-de-ultrasonidos-en-un-sistema-autonomo-de-tiempo-real-m-a-esther-gilaberte-sanz.html

[8] Zarith Fabiola Niño Castillo y Victor Edwin Diaz Monroy, “Módulo didáctico para prácticas de laboratorio con detectores de proximidad para la asignatura instrumentación electrónica”. Universidad Industrial de Santander, 2010.

https://es.slideshare.net/ford81/teroria-de-sensores

[9] Linda Laura Toribio Hernández, “Diseño y construcción de un prototipo mecatrónico de navegación autónoma y simulación del control”, Capítulo 3. Universidad de las Américas Puebla, 14 de mayo de 2008.

http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/toribio_h_ll/capitulo3.pdf

ARDUINO

[10] Arduino.cl, “¿Qué es Arduino?”.

http://arduino.cl/que-es-arduino/

[11] Aprendiendo Arduino. “Cómo funciona Arduino”. 28 de marzo de 2016.

https://aprendiendoarduino.wordpress.com/2016/03/28/como-funciona-arduino/

[12] Traducido y adaptado: José Manuel Ruiz Gutiérrez, “Manual de Programación Arduino”.
https://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf

[13] Luis Llamas, “Comunicación de Arduino con puerto serie”. Ingeniería, Informática y Diseño, 16 de abril 2014.

https://www.luisllamas.es/arduino-puerto-serie/

[14] Luis Llamas, “¿Qué es Arduino? ¿Qué modelo comprar?”. Ingeniería, Informática y Diseño, 2 de octubre 2013.

https://www.luisllamas.es/que-es-arduino-que-modelo-comprar/

Módulos de RF

[15] Luis Llamas, “Comunicación Inalámbrica con módulos de RF 433Mhz”. Ingeniería, Informática y Diseño, 6 de diciembre 2016.

https://www.luisllamas.es/comunicacion-inalambrica-en-arduino-con-modulos-rf-433mhz/

[16] Naylamp Mechatronics, “Comunicación Inalámbrica con módulos de RF de 433Mhz”.

http://www.naylampmechatronics.com/blog/32_Comunicaci%C3%B3n-Inal%C3%A1mbrica-con-m%C3%B3dulos-de-RF-de.html

Servo

[17] Luis Llamas, "Controlar un servo con Arduino". Ingeniería, Informática y Diseño, 16 de junio 2016.

https://www.luisllamas.es/controlar-un-servo-con-arduino/

Realterm

[18] Created by: Chris MacLellan, FSF Edinburgh, "Setting up Windows Hyper Terminal for use with the MicroTops II".

http://fsf.nerc.ac.uk/resources/guides/pdf_guides/MicrotopsII_%20RealTerm_v1.pdf

Presupuesto

[19] K-electrónica. http://k-electronica.es/

Software gratuito

- Arduino, https://www.arduino.cc/en/main/software
- Librería VirtualWire, http://www.airspayce.com/mikem/arduino/VirtualWire/index.html
- Realterm, https://sourceforge.net/projects/realterm/?source=typ_redirect
- Matlab, https://matlab.programas-gratis.net/
- Notepad++, https://notepad-plus-plus.org/download/v7.4.2.html

Datasheets de los componentes

- Sensor ultrasónico HC-SR04
  http://www.robotshop.com/media/files/pdf2/hc-sr04-ultrasonic-range-finder-datasheet.pdf
  https://www.parallax.com/sites/default/files/downloads/28015-PING-Sensor-Product-Guide-v2.0.pdf
- Módulo de radiofrecuencias 433Mhz
  file:///C:/Users/ACER/Downloads/Hoja%20T%C3%A9cnica_M%C3%B3dulos%20RF%20433MHz.pdf
- Puerta OR, DM74LS32N
  http://www.datasheetcatalog.com/datasheets_pdf/D/M/7/4/DM74LS32N.shtml
- Servo
  http://www.datasheetspdf.com/mobile/633270/SG-5010.html

ANEXO I.
CÓDIGOS ARDUINO

Orden de programas y su utilidad

Montaje inicial. Código ‘unultrasonido’

Programa básico para un ultrasonido midiendo de forma normal.

```
//declaramos las variables como números enteros
long distancia;
long tiempo;

void setup(){

  Serial.begin(9600);
  pinMode(9, OUTPUT); //activación del pin 9 como salida: para el pulso ultrasónico
  pinMode(8, INPUT); //activación del pin 8 como entrada: tiempo del rebote del ultrasonido
}

void loop(){

  digitalWrite(9,LOW); //por cuestión de estabilización del sensor ponemos el pin a baja
  delayMicroseconds(3);
  digitalWrite(9, HIGH); //poniendo el pin a alta, enviamos el pulso ultrasónico
  delayMicroseconds(10);
  tiempo=pulseIn(8, HIGH,9000); //calculamos el tiempo

  distancia= int(0.017*tiempo);
  Serial.print("La distancia son "); //imprimimos mensaje por pantalla
  Serial.print(distancia); //nos muestra el resultado
  Serial.println(" cm");
  delay(1000); //imprime mensaje cada 1 seg
}
```

Segundo montaje. Código 'dosultra'

Programa básico empleando dos ultrasonidos, que miden de forma independiente.

```
//declaramos las variables como números enteros
long distancia1;
long tiempo1;
long distancia2;
long tiempo2;

void setup(){
  Serial.begin(9600);
  pinMode(9, OUTPUT); /*activación del pin 9 como salida: para el pulso ultrasónico*/
  pinMode(8, INPUT); /*activación del pin 8 como entrada: tiempo del rebote del ultrasonido*/
  pinMode(13, OUTPUT); /*activación del pin 13 como salida: para el pulso ultrasónico*/
  pinMode(12, INPUT); /*activación del pin 12 como entrada: tiempo del rebote del ultrasonido*/
}

void loop(){
  digitalWrite(9,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(9, HIGH); /* envío del pulso ultrasónico*/
  delayMicroseconds(10);
  tiempo1=pulseIn(8, HIGH,9000); /* Función para medir la longitud del pulso entrante. Mide el
  tiempo que transcurrido entre el envío del pulso ultrasónico y cuando el sensor recibe el
  rebote, es decir: desde que el pin 12 empieza a recibir el rebote, HIGH, hasta que
  deja de hacerlo, LOW, la longitud del pulso entrante*/

  digitalWrite(13,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(13, HIGH); /* envío del pulso ultrasónico*/
  delayMicroseconds(10);
  tiempo2=pulseIn(12, HIGH,9000);

  distancia1= int(0.017*tiempo1); /*fórmula para calcular la distancia obteniendo un valor entero*/
  /*Monitorización en centímetros por el monitor serial*/

  distancia2= int(0.017*tiempo2);

  Serial.print("Distancia sensor 1: ");//imprimimos mensaje por pantalla del sensor 1
  Serial.print(distancia1);//nos muestra el resultado del sensor 1
  Serial.println(" cm");
  Serial.print("Distancia sensor 2: ");//imprimimos mensaje por pantalla del sensor 2
  Serial.print(distancia2);//nos muestra el resultado del sensor 2
  Serial.println(" cm");
  delay(1000);//imprime mensaje cada 1 seg
}
```

Tercer montaje. Código ‘pruebita1’

En el tercer montaje, vemos el funcionamiento erróneo de ultrasonidos de forma dependiente, emisor y receptor, en una misma placa. Para hacerlos funcionar de esta forma, realizamos un cambio tanto en su software como en su hardware.

Este error fue producido debido a que los sensores, al funcionar de forma dependiente, originaron un cambio en el timer empleado para el cálculo de la variable *tiempo*. Esto origino diversos problemas en búsqueda, entendimiento de distintos timers, e intentos de modificación. Por ello, pasamos a la comunicación entre sensores, que veremos en el cuarto montaje.

```
float distancia; //declaramos como números decimal
long tiempo; //declaramos como números enteros

void setup(){
  Serial.begin(9600);
  pinMode(10, OUTPUT); /*activación del pin 10 como salida: para el pulso ultrasónico*/
  pinMode(11, INPUT); /*activación del pin 11 como entrada: tiempo del rebote del
ultrasonido*/
}

void loop(){

  digitalWrite(10,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(10, HIGH); /* envío del pulso ultrasónico*/
  delayMicroseconds(10);
  tiempo=pulseIn(11, HIGH,9000);/* Función para medir la longitud del pulso entrante. Mide
el
  tiempo que transcurrido entre el envío del pulso ultrasónico y cuando el sensor recibe el
  rebote, es decir: desde que el pin 12 empieza a recibir el rebote, HIGH, hasta que
  deja de hacerlo, LOW, la longitud del pulso entrante*/

  distancia1= (0.017*tiempo);/*fórmula para calcular la distancia obteniendo un valor entero*/
  /*Monitorización en centímetros por el monitor serial*/

  Serial.println("Distancia aproximada ");//imprimimos mensaje por pantalla
  Serial.println(distancia);//nos muestra el resultado
  Serial.println(" cm");
  delay(1000);//imprime mensaje cada 1 seg
}
```

Cuarto montaje.

Comenzamos la comunicación a partir de módulos de radiofrecuencias. En el siguiente programa vemos el funcionamiento de dos ultrasonidos dependientes, situados en distinta placa. Inicialmente, se llevará acabo un ejemplo de envío de caracter, 'Hola mundo'. Para ver la comunicación de forma directa en el mismo tiempo, utilizamos la herramienta Realterm mencionada en la memoria.

EJEMPLO 1. ENVÍO DE CARÁCTER. Código 'RFem/rec'

EMISOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

void setup()
{
  Serial.begin(9600);
  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_tx_pin(dataPin); //pin por el que recibiremos los datos
  Serial.println("Escribiendo..."); //imprimimos un mensaje para saber que funciona
}

void loop()
{
  const char *msg = "Hola mundo"; //mensaje que queremos enviar

  digitalWrite(ledPin, true);
  vw_send((uint8_t *)msg, strlen(msg));//Envia un mensaje con la longitud dada.
  //La función termina rápido pero el mensaje será enviado en el momento
  //establecido por las interrupciones
  vw_wait_tx(); //Hace una pausa hasta que se trasmitan todos los datos
  digitalWrite(ledPin, false);
  delay(200);
}
```

RECEPTOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

void setup()
{
  Serial.begin(9600);
  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_rx_pin(dataPin); //pin por el que recibiremos los datos
  vw_rx_start(); //Empieza a escuchar los datos provenientes por el pin_rx,
```

```
   //es necesario llamar a esta función para poder recibir los datos
}

void loop()
{
   //uint8_t vw_get_message(uint8_t * buf, uint8_t * len )
   //Si un mensaje está disponible, lo almacena buf , devuelve true si la comprobación
   //es correcta. buf es puntero a la ubicación para guardar los datos de lectura y
   //len es un puntero a la cantidad de bytes disponibles de buf.
   uint8_t buf[VW_MAX_MESSAGE_LEN];
   uint8_t buflen = VW_MAX_MESSAGE_LEN;

   if (vw_get_message(buf, &buflen))
   {
    digitalWrite(ledPin, true);
    Serial.print("Mensaje: "); //imprime el mensaje
    for (int i = 0; i < buflen; i++)
    {
      Serial.print((char)buf[i]);
    }
    Serial.println("");
     digitalWrite(ledPin, false);
   }
}
```

Código ‘ultRFem/rec’

Por otro lado, en el mismo montaje, solucionamos el error del timer modificando el código fuente del pulseIn para hacerlo funcionar con pulsos como los que emite el ultrasonido. Declaramos un nuevo ‘timer’ llamado *mipulseIn* en nuestro programa, (parte del receptor), quedando de la siguiente forma:

EMISOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

void setup()
{
   Serial.begin(9600);
   pinMode(10, OUTPUT); /*activación del pin 10 como salida: para el pulso ultrasónico*/
   vw_setup(2000); //fijamos velocidad de comunicación
   vw_set_tx_pin(dataPin); //pin por el que recibiremos los datos
   Serial.println("Escribiendo..."); //imprimimos un mensaje para saber que funciona
}

void loop()
{
   const char *msg = "S"; //mensaje que queremos enviar
   vw_send((uint8_t *)msg, strlen(msg));//Envia un mensaje con la longitud dada.
```

```
   //La función termina rápido pero el mensaje será enviado en el momento
   //establecido por las interrupciones
   vw_wait_tx(); //Hace una pausa hasta que se trasmitan todos los datos
   digitalWrite(10,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
   digitalWrite(10, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(1000);

}
```

RECEPTOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>
//necesarios para el pulseIn
#include "wiring_private.h"
#include "pins_arduino.h"

//Declaramos un nuevo timer, mipulseIn
unsigned long mipulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
  // Cache el puerto y el bit del pin para acelerar el
// ciclo de medición del pulso y lograr una resolución más fina. Llamando
// a digitalRead () produce una resolución mucho mejor.
  uint8_t bit = digitalPinToBitMask(pin);
  uint8_t port = digitalPinToPort(pin);
  uint8_t stateMask = (state ? bit : 0);
  unsigned long width = 0; // Mantener la inicialización fuera del tiempo  de área crítica

  // Convertir el tiempo de espera de microsegundos a un número de veces a través del
   // bucle inicial; Toma 16 ciclos de reloj por iteración.
  unsigned long numloops = 0;
  unsigned long maxloops = microsecondsToClockCycles(timeout) / 16;


  // Esta será la parte del pulseIn que usaremos, donde cuenta el tiempo hasta que haya un pulso
  //positivo

  while ((*portInputRegister(port) & bit) == stateMask) {
    if (numloops++ == maxloops)
      return 0;
    width++;
  }

  // Convertir la lectura en microsegundos. El bucle se ha determinado que tiene 20 ciclos de
  // reloj de largo y tiene alrededor de 16 relojes entre el borde y el inicio del bucle. Habrá algún
  // error introducido por los manejadores de interrupción.

  return clockCyclesToMicroseconds(width * 21 + 16);
}
```

```
const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

float distancia; /* números con decimales*/
long tiempo; /*números enteros*/

void setup() {

  Serial.begin(9600);
  pinMode(11, INPUT); /*activación del pin x como entrada: tiempo del rebote del ultrasonido*/
  /*parte RFrec*/
  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_rx_pin(dataPin); //pin por el que recibiremos los datos
  vw_rx_start(); //Empieza a escuchar los datos provenientes por el pin_rx,
   //es necesario llamar a esta función para poder recibir los datos

}

void loop() {

   //uint8_t vw_get_message(uint8_t * buf, uint8_t * len )
   //Si un mensaje está disponible, lo almacena buf , devuelve true si la comprobación
   //es correcta. buf es puntero a la ubicación para guardar los datos de lectura y
   //len es un puntero a la cantidad de bytes disponibles de buf.

   uint8_t buf[VW_MAX_MESSAGE_LEN];
   uint8_t buflen = VW_MAX_MESSAGE_LEN;

   if (vw_get_message(buf, &buflen))
   {
    if (buf[0] == 'S'){
      tiempo=mipulseIn(11, LOW,9000); //medimos el tiempo con el nuevo timer,
      //Además tuvimos que cambiar del estado HIGH a LOW para que contára el tiempo hasta
      //que haya un pulso positivo
      Serial.println("Tiempo en enviar señal ");//imprime el mensaje por pantalla
      Serial.println(tiempo); //imprime resultado

      distancia=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
     /*Monitorización en centímetros por el monitor serial*/

      Serial.println("Distancia aproximada "); //imprime el menssaje por pantalla
      Serial.println(distancia); //imprime resultado
      }
    else (Serial.println("error")); //en caso de que no se cumpla lo anterior, que imprima 'error'
   }
}
```

Quinto montaje. Código 'ultRFem2/rec2'

Para efectuar el método de localización elegido, deberemos implementar otros dos dispositivos ultrasónicos que actuarán igualmente como emisores. Emite tres señales que serán recibidas por el receptor el cuál pondrá en marcha la medida de la distancia.

EMISOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

void setup() {

  Serial.begin(9600);
  pinMode(10, OUTPUT); /*activación del pin 10 como salida: para el pulso ultrasónico*/
  pinMode(12, OUTPUT); /*activación del pin 12 como salida: para el pulso ultrasónico*/
  pinMode(13,OUTPUT); /*activación del pin 13 como salida: para el pulso ultrasónico*/

  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_tx_pin(dataPin); //pin por el que recibiremos los datos
  Serial.println("Emitiendo señal...");//imprimimos un mensaje para saber que funciona

}
void loop() {

//Emisor numero 2 asignado al pin10

  const char *msg = "S"; //mensaje que queremos enviar
  vw_send((uint8_t *)msg, strlen(msg));//Envia un mensaje con la longitud dada.
  //La función termina rápido pero el mensaje será enviado en el momento
  //establecido por las interrupciones
  vw_wait_tx(); //Hace una pausa hasta que se trasmitan todos los datos
  digitalWrite(10,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(10, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(100);

//Emisor numero 2 asignado al pin12

  const char *msg2 = "E";
  vw_send((uint8_t *)msg2, strlen(msg2));
  vw_wait_tx();
  digitalWrite(12,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(12, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(100);
```

```
//Emisor numero 3 asignado al pin13

  const char *msg3 = "T";
  vw_send((uint8_t *)msg3, strlen(msg3));
  vw_wait_tx();
  digitalWrite(13,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(13, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  Serial.println("Emitiendo...");
  delay(100);
}
```

RECEPTOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>
//necesarios para el pulseIn
#include "wiring_private.h"
#include "pins_arduino.h"

//Declaramos un nuevo timer, mipulseIn
unsigned long mipulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
  // Cache el puerto y el bit del pin para acelerar el
// ciclo de medición del pulso y lograr una resolución más fina. Llamando
// a digitalRead () produce una resolución mucho mejor.
  uint8_t bit = digitalPinToBitMask(pin);
  uint8_t port = digitalPinToPort(pin);
  uint8_t stateMask = (state ? bit : 0);
  unsigned long width = 0; // Mantener la inicialización fuera del tiempo  de área crítica

  // Convertir el tiempo de espera de microsegundos a un número de veces a través del
   // bucle inicial; Toma 16 ciclos de reloj por iteración.
  unsigned long numloops = 0;
  unsigned long maxloops = microsecondsToClockCycles(timeout) / 16;


  // Esta será la parte del pulseIn que usaremos, donde cuenta el tiempo hasta que haya un pulso
  //positivo

  while ((*portInputRegister(port) & bit) == stateMask) {
    if (numloops++ == maxloops)
      return 0;
    width++;
  }

  // Convertir la lectura en microsegundos. El bucle se ha determinado que tiene 20 ciclos de
  // reloj de largo y tiene alrededor de 16 relojes entre el borde y el inicio del bucle. Habrá algún
  // error introducido por los manejadores de interrupción.
```

```
  return clockCyclesToMicroseconds(width * 21 + 16);
}

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

float distancia; /* números con decimales*/
long tiempo; /*números enteros*/

void setup() {

  Serial.begin(9600);
  pinMode(11, INPUT); /*activación del pin x como entrada: tiempo del rebote del ultrasonido*/
  /*parte RFrec*/
  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_rx_pin(dataPin); //pin por el que recibiremos los datos
  vw_rx_start(); //Empieza a escuchar los datos provenientes por el pin_rx,
   //es necesario llamar a esta función para poder recibir los datos

}

void loop() {

   //uint8_t vw_get_message(uint8_t * buf, uint8_t * len )
   //Si un mensaje está disponible, lo almacena buf , devuelve true si la comprobación
   //es correcta. buf es puntero a la ubicación para guardar los datos de lectura y
   //len es un puntero a la cantidad de bytes disponibles de buf.

   uint8_t buf[VW_MAX_MESSAGE_LEN];
   uint8_t buflen = VW_MAX_MESSAGE_LEN;

   if (vw_get_message(buf, &buflen))
   {
    if (buf[0] == 'S'){
      tiempo=mipulseIn(11, LOW,9000); //medimos el tiempo con el nuevo timer,
      //Además tuvimos que cambiar del estado HIGH a LOW para que contára el tiempo hasta
      //que haya un pulso positivo
      Serial.println("Conectado a EMISOR 1");
      Serial.println("Tiempo en enviar señal ");//imprime el mensaje por pantalla
      Serial.println(tiempo); //imprime resultado

      distancia=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
     /*Monitorización en centímetros por el monitor serial*/

      Serial.println("Distancia aproximada "); //imprime el menssaje por pantalla
      Serial.println(distancia); //imprime resultado
      }

    if (buf[0] == 'E'){

      tiempo=mipulseIn(11, LOW,9000);
      Serial.println("Conectado a EMISOR 2");
      Serial.println("Tiempo en enviar señal ");
```

```
        Serial.println(tiempo);

        distancia=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
       /*Monitorización en centímetros por el monitor serial*/

        Serial.println("Distancia aproximada ");
        Serial.println(distancia);

        }

      if (buf[0] == 'T'){

        tiempo=mipulseIn(11, LOW,9000);
        Serial.println("Conectado a EMISOR 3");
        Serial.println("Tiempo en enviar señal ");
        Serial.println(tiempo);

        distancia=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
       /*Monitorización en centímetros por el monitor serial*/

        Serial.println("Distancia aproximada ");
        Serial.println(distancia);

      }

    }
}
```

Sexto montaje. Código 'ultRFem2 – rectrilat2'

Definimos las coordenadas de cada uno de nuestros nodos emisores para emplear las ecuaciones de trilateración. Programa receptor calcula la posición a partir de las expresiones de trilateración suponiendo A(x1,y1), en cualquier punto.

EMISOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

void setup() {

  Serial.begin(9600);
  pinMode(10, OUTPUT); /*activación del pin 10 como salida: para el pulso ultrasónico*/
  pinMode(12, OUTPUT); /*activación del pin 12 como salida: para el pulso ultrasónico*/
  pinMode(13,OUTPUT); /*activación del pin 13 como salida: para el pulso ultrasónico*/

  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_tx_pin(dataPin); //pin por el que recibiremos los datos
  Serial.println("Emitiendo señal...");//imprimimos un mensaje para saber que funciona

}
void loop() {

//Emisor numero 2 asignado al pin10

  const char *msg = "S"; //mensaje que queremos enviar
  vw_send((uint8_t *)msg, strlen(msg));//Envia un mensaje con la longitud dada.
  //La función termina rápido pero el mensaje será enviado en el momento
  //establecido por las interrupciones
  vw_wait_tx(); //Hace una pausa hasta que se trasmitan todos los datos
  digitalWrite(10,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(10, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(100);

//Emisor numero 2 asignado al pin12

  const char *msg2 = "E";
  vw_send((uint8_t *)msg2, strlen(msg2));
  vw_wait_tx();
  digitalWrite(12,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(12, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(100);
```

```
//Emisor numero 3 asignado al pin13

  const char *msg3 = "T";
  vw_send((uint8_t *)msg3, strlen(msg3));
  vw_wait_tx();
  digitalWrite(13,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(13, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  Serial.println("Emitiendo...");
  delay(100);
}
```

RECEPTOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>
//necesarios para el pulseIn
#include "wiring_private.h"
#include "pins_arduino.h"

//Declaramos un nuevo timer, mipulseIn
unsigned long mipulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
  // Cache el puerto y el bit del pin para acelerar el
// ciclo de medición del pulso y lograr una resolución más fina. Llamando
// a digitalRead () produce una resolución mucho mejor.
  uint8_t bit = digitalPinToBitMask(pin);
  uint8_t port = digitalPinToPort(pin);
  uint8_t stateMask = (state ? bit : 0);
  unsigned long width = 0; // Mantener la inicialización fuera del tiempo  de área crítica

  // Convertir el tiempo de espera de microsegundos a un número de veces a través del
   // bucle inicial; Toma 16 ciclos de reloj por iteración.
  unsigned long numloops = 0;
  unsigned long maxloops = microsecondsToClockCycles(timeout) / 16;


  // Esta será la parte del pulseIn que usaremos, donde cuenta el tiempo hasta que haya un pulso
  //positivo

  while ((*portInputRegister(port) & bit) == stateMask) {
    if (numloops++ == maxloops)
      return 0;
    width++;
  }

  // Convertir la lectura en microsegundos. El bucle se ha determinado que tiene 20 ciclos de
  // reloj de largo y tiene alrededor de 16 relojes entre el borde y el inicio del bucle. Habrá algún
  // error introducido por los manejadores de interrupción.

  return clockCyclesToMicroseconds(width * 21 + 16);
```

```
}

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

float d1=-1, d2=-1, d3=-1; /* números con decimales*/
long tiempo; /*números enteros*/

//insertar coordenadas de D1, D2, D3
float A;
float B;
float C;
float D;
float E;
float F;

float x1;
float y1;
float x2;
float y2;
float x3;
float y3;

float X;
float Y;
float posicion1;
float posicion2;

void setup() {

  Serial.begin(9600);
  pinMode(11, INPUT); /*activación del pin x como entrada: tiempo del rebote del ultrasonido*/
  /*parte RFrec*/
  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_rx_pin(dataPin); //pin por el que recibiremos los datos
  vw_rx_start(); //Empieza a escuchar los datos provenientes por el pin_rx,
 //es necesario llamar a esta función para poder recibir los datos

}

void loop() {

   uint8_t buf[VW_MAX_MESSAGE_LEN];
   uint8_t buflen = VW_MAX_MESSAGE_LEN;

   if (vw_get_message(buf, &buflen))
   {
    if (buf[0] == 'S'){

      tiempo=mipulseIn(11, LOW,9000);
      Serial.println("Conectado a EMISOR 1");
      Serial.println("Tiempo en enviar señal ");
      Serial.println(tiempo);
```

```
      d1=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
     /*Monitorización en centímetros por el monitor serial*/

      Serial.println("Distancia aproximada del emisor 1 ");
      Serial.println(d1);

      }

    if (buf[0] == 'E'){

      tiempo=mipulseIn(11, LOW,9000);
      Serial.println("Conectado a EMISOR 2");
      Serial.println("Tiempo en enviar señal ");
      Serial.println(tiempo);

      d2=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
     /*Monitorización en centímetros por el monitor serial*/

      Serial.println("Distancia aproximada del emisor 2 ");
      Serial.println(d2);

      }


   if (buf[0] == 'T'){

      tiempo=mipulseIn(11, LOW,9000);
      Serial.println("Conectado a EMISOR 3");
      Serial.println("Tiempo en enviar señal ");
      Serial.println(tiempo);

      d3=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
     /*Monitorización en centímetros por el monitor serial*/

      Serial.println("Distancia aproximada del emisor 3 ");
      Serial.println(d3);

      }

      /*Expresiones de trilateración
      (d1*d1)= ((X-x1)*(X-x1))+((Y-y1)*(Y-y1)))
      (d2*d2)= ((X-x2)*(X-x2))+((Y-y2)*(Y-y2)))
      (d3*d3)= ((X-x3)*(X-x3))+((Y-y3)*(Y-y3)))
      Damos valores fijos de las posiciones de los emisores*/
      x1=0;
      y1=0;
      x2=35;
      y2=0;
      x3=20;
      y3=20;

      A=(-2*x1)+(2*x2);
```

```
        B=(-2*y1)+(2*y2);
        C=(-2*x2)+(2*x3);
        D=(-2*y2)+(2*y3);
        E=(d1*d1)-(d2*d2)-(x1*x1)+(x2*x2)-(y1*y1)+(y2*y2);
        E=(d2*d2)-(d3*d3)-(x2*x2)+(x3*x3)-(y2*y2)+(y3*y3);

        /*
        (A*X)+ (B*Y)=C
        (D*X)+ (E*Y)=F   */

        X=((C*E)-(F*B))/((E*A)-(B*D));

        Y=((C*D)-(A*F))/((B*D)-(A*E));

        posicion1=X;
        posicion2=Y;

        Serial.println("Posición aproximada ");
        Serial.println(posicion1);
        Serial.println(posicion2);

    }

}
```

Séptimo montaje. Código 'ultRFem2 – unrec_trilat_servo'

Dejando a un lado los cálculos de posición, realizaremos un nuevo programa en el que implementaremos nuestro servo.

Parece ser que el servo interfiere usando mismos recursos que nuestros módulos de RF, en concreto del timer. Como solución, sustituiremos la placa receptora que es donde situaremos al servo, por una placa arduino MEGA. Además en el mismo directorio del archivo 'rec_servo', añadimos Servo.h, Servo.cpp, y ServoTimers.h. En estos archivos, indicamos a nuestro programa el timer, timer3, que ha de usar el servo.

En este programa, la medida de las coordenadas del receptor son erróneas aun.

EMISOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

void setup() {

  Serial.begin(9600);
  pinMode(10, OUTPUT); /*activación del pin 10 como salida: para el pulso ultrasónico*/
  pinMode(12, OUTPUT); /*activación del pin 12 como salida: para el pulso ultrasónico*/
  pinMode(13,OUTPUT); /*activación del pin 13 como salida: para el pulso ultrasónico*/

  vw_setup(2000); //fijamos velocidad de comunicación
  vw_set_tx_pin(dataPin); //pin por el que recibiremos los datos
  Serial.println("Emitiendo señal...");//imprimimos un mensaje para saber que funciona

}
void loop() {

//Emisor numero 2 asignado al pin10

  const char *msg = "S"; //mensaje que queremos enviar
  vw_send((uint8_t *)msg, strlen(msg));//Envia un mensaje con la longitud dada.
  //La función termina rápido pero el mensaje será enviado en el momento
  //establecido por las interrupciones
  vw_wait_tx(); //Hace una pausa hasta que se trasmitan todos los datos
  digitalWrite(10,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(10, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(100);

//Emisor numero 2 asignado al pin12
```

```
  const char *msg2 = "E";
  vw_send((uint8_t *)msg2, strlen(msg2));
  vw_wait_tx();
  digitalWrite(12,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(12, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  delay(100);

//Emisor numero 3 asignado al pin13

  const char *msg3 = "T";
  vw_send((uint8_t *)msg3, strlen(msg3));
  vw_wait_tx();
  digitalWrite(13,LOW); /* Por cuestión de estabilización del sensor*/
  delayMicroseconds(5);
  digitalWrite(13, HIGH); /* envío del pulso ultrasónico*/

  digitalWrite(ledPin, false);
  Serial.println("Emitiendo...");
  delay(100);
}
```

RECEPTOR

```
//incluimos librería VirtualWire necesaría para los módulos RF 433
#include <VirtualWire.h>
//necesarios para el pulseIn
#include "wiring_private.h"
#include "pins_arduino.h"
/*incluimos librería Servo pero entre comillas para que tenga en cuenta los demás programas
, es decir, Servo.h, Servo.cpp y ServoTimer.h*/
#include "Servo.h"


// Declaramos la variable para controlar el servo
Servo servoMotor;


//Declaramos un nuevo timer, mipulseIn
unsigned long mipulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
  // Cache el puerto y el bit del pin para acelerar el
// ciclo de medición del pulso y lograr una resolución más fina. Llamando
// a digitalRead () produce una resolución mucho mejor.
  uint8_t bit = digitalPinToBitMask(pin);
  uint8_t port = digitalPinToPort(pin);
  uint8_t stateMask = (state ? bit : 0);
  unsigned long width = 0; // Mantener la inicialización fuera del tiempo  de área crítica

  // Convertir el tiempo de espera de microsegundos a un número de veces a través del
   // bucle inicial; Toma 16 ciclos de reloj por iteración.
  unsigned long numloops = 0;
```

```
  unsigned long maxloops = microsecondsToClockCycles(timeout) / 16;


  // Esta será la parte del pulseIn que usaremos, donde cuenta el tiempo hasta que haya un pulso
  //positivo

  while ((*portInputRegister(port) & bit) == stateMask) {
    if (numloops++ == maxloops)
      return 0;
    width++;
  }

  // Convertir la lectura en microsegundos. El bucle se ha determinado que tiene 20 ciclos de
  // reloj de largo y tiene alrededor de 16 relojes entre el borde y el inicio del bucle. Habrá algún
  // error introducido por los manejadores de interrupción.

  return clockCyclesToMicroseconds(width * 21 + 16);
}

const int dataPin = 9; //pin del módulo de RF
const int ledPin = 13; //led asociado

float d1=-1, d2=-1, d3=-1; /* números con decimales, le damos valor -1 ya que no es un valor
posible */
long tiempo; /*números enteros*/
//insertar coordenadas de D1, D2, D3
float A;
float B;
float C;
float D;
float E;
float F;

float x1;
float y1;
float x2;
float y2;
float x3;
float y3;

float X;
float Y;
float posicion1;
float posicion2;


    uint8_t buf[VW_MAX_MESSAGE_LEN];
    uint8_t buflen = VW_MAX_MESSAGE_LEN;



long medida () { //hacemos una función donde incluimos toda la parte de medida
   long tiempo1 = millis();
   vw_setup(2000);
   vw_set_rx_pin(dataPin);
```

```
  vw_rx_start();
  while (!vw_get_message(buf, &buflen))
      if (millis() - tiempo1 > 300)
        return -1; //mientras no reciba un mensaje no lee, cuando lo reciba verá cuál de los tres emisores es


   if (buf[0] == 'S'){

       tiempo=mipulseIn(11, LOW,900);
       Serial.println("Conectado a EMISOR 1");
       //Serial.println("Tiempo en enviar señal del emisor 1");
       Serial.println(tiempo);

       d1=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
      /*Monitorización en centímetros por el monitor serial*/

       Serial.println("Distancia aproximada ");
       Serial.println(d1);

       }


   if (buf[0] == 'E'){

       tiempo=mipulseIn(11, LOW,900);
       Serial.println("Conectado a EMISOR 2");
       //Serial.println("Tiempo en enviar señal ");
       Serial.println(tiempo);

       d2=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
      /*Monitorización en centímetros por el monitor serial*/

       Serial.println("Distancia aproximada del emisor 2");
       Serial.println(d2);

       }


  if (buf[0] == 'T'){

       tiempo=mipulseIn(11, LOW,900);
       Serial.println("Conectado a EMISOR 3");
       //Serial.println("Tiempo en enviar señal ");
       Serial.println(tiempo);

       d3=(0.0443*tiempo-8.6947); /*fórmula para calcular la distancia obteniendo un valor entero*/
      /*Monitorización en centímetros por el monitor serial*/

       Serial.println("Distancia aproximada del emisor 3 ");
       Serial.println(d3);
```

```
        }

        /*Expresiones de trilateracion
        (d1*d1)= ((X-x1)*(X-x1))+((Y-y1)*(Y-y1)))
        (d2*d2)= ((X-x2)*(X-x2))+((Y-y2)*(Y-y2)))
        (d3*d3)= ((X-x3)*(X-x3))+((Y-y3)*(Y-y3)))
        Damos valores fijos de las posiciones de los emisores*/
        x1=0;
        y1=0;
        x2=35;
        y2=0;
        x3=20;
        y3=20;

        A=(-2*x1)+(2*x2);
        B=(-2*y1)+(2*y2);
        C=(-2*x2)+(2*x3);
        D=(-2*y2)+(2*y3);
        E=(d1*d1)-(d2*d2)-(x1*x1)+(x2*x2)-(y1*y1)+(y2*y2);
        E=(d2*d2)-(d3*d3)-(x2*x2)+(x3*x3)-(y2*y2)+(y3*y3);

        /*
        (A*X)+ (B*Y)=C
        (D*X)+ (E*Y)=F   */

        X=((C*E)-(F*B))/((E*A)-(B*D));

        Y=((C*D)-(A*F))/((B*D)-(A*E));

        posicion1=X;
        posicion2=Y;

        Serial.println("Posición aproximada ");
        Serial.println(posicion1);
        Serial.println(posicion2);


}


void setup() {


  // Iniciamos el monitor serie para mostrar el resultado
  Serial.begin(9600);
  pinMode(11, INPUT); /*activación del pin x como entrada: tiempo del rebote del
ultrasonido*/

  // Iniciamos el servo para que empiece a trabajar con el pin 7
  servoMotor.attach(7);
}


void loop() {
```

```
  // Desplazamos a la posición -90º
  servoMotor.write(180);
  Serial.print("Moviendo primera posición");
  delay (1000);
  if (medida() == -1)
    Serial.println("Error no ha llegado mensaje RF");

  // Desplazamos a la posición 0º
  servoMotor.write(0);
   Serial.print("Moviendo segunda posición");
   delay (1000);
  if (medida() == -1)
    Serial.println("Error no ha llegado mensaje RF");

// Desplazamos a la posición 90º
   servoMotor.write(90);
   Serial.print("Moviendo tercera posición");
   delay (1000);
  if (medida() == -1)
    Serial.println("Error no ha llegado mensaje RF");

}
```

Octavo montaje. Código 'ultRFem2 – dosrec_trilat_servo'

Como vimos en el anterior montaje, el barrido solo gira 180º, que es el máximo que abarca el servo elegido. Para poder barrer los 180º restantes, acoplamos otro dispositivo ultrasónico que actuará igualmente como receptor. En este caso, obtenemos información de medida que con un barrido de sólo 180º hubiera sido imposible.

A la hora de implementar el nuevo dispositivo, el código empleado es el mismo, añadimos una puerta lógica OR, que se encargará de sumar ambas salidas cada vez que reciba una medida, ya sea de una, de otra o de ambas.

ANEXO II.
DOCUMENTACIÓN

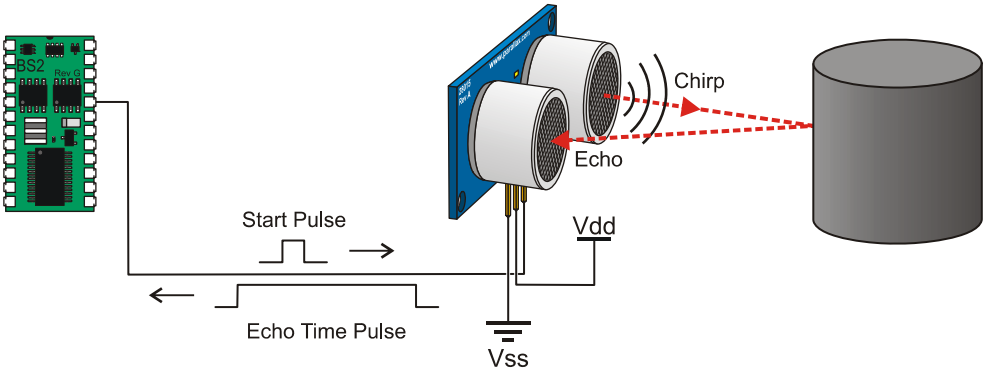
Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

PING))) Ultrasonic Distance Sensor (#28015)

The Parallax PING)))™ ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to microcontrollers such as the BASIC Stamp®, Propeller chip, or Arduino, requiring only one I/O pin.

The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width, the distance to target can easily be calculated.



Features

- Range: 2 cm to 3 m (0.8 in to 3.3 yd)
- Burst indicator LED shows sensor activity
- Bidirectional TTL pulse interface on a single I/O pin can communicate with 5 V TTL or 3.3 V CMOS microcontrollers
- Input trigger: positive TTL pulse, 2 µs min, 5 µs typ.
- Echo pulse: positive TTL pulse, 115 µs minimum to 18.5 ms maximum.
- RoHS Compliant

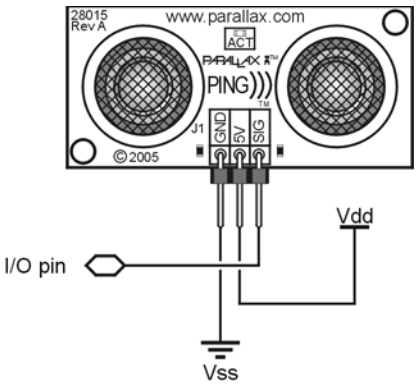
Key Specifications

- Supply voltage: +5 VDC
- Supply current: 30 mA typ; 35 mA max
- Communication: Positive TTL pulse
- Package: 3-pin SIP, 0.1" spacing (ground, power, signal)
- Operating temperature: 0 – 70° C.
- Size: 22 mm H x 46 mm W x 16 mm D (0.84 in x 1.8 in x 0.6 in)
- Weight: 9 g (0.32 oz)

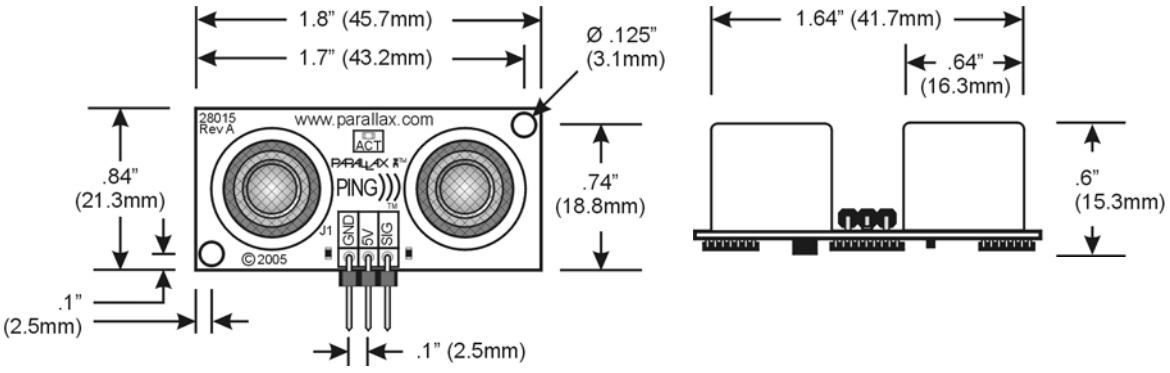
Pin Definitions

GND	Ground (Vss)
5 V	5 VDC (Vdd)
SIG	Signal (I/O pin)

The PING))) sensor has a male 3-pin header used to supply ground, power (+5 VDC) and signal. The header may be plugged into a directly into solderless breadboard, or into a standard 3-wire extension cable (Parallax part #800-00120).

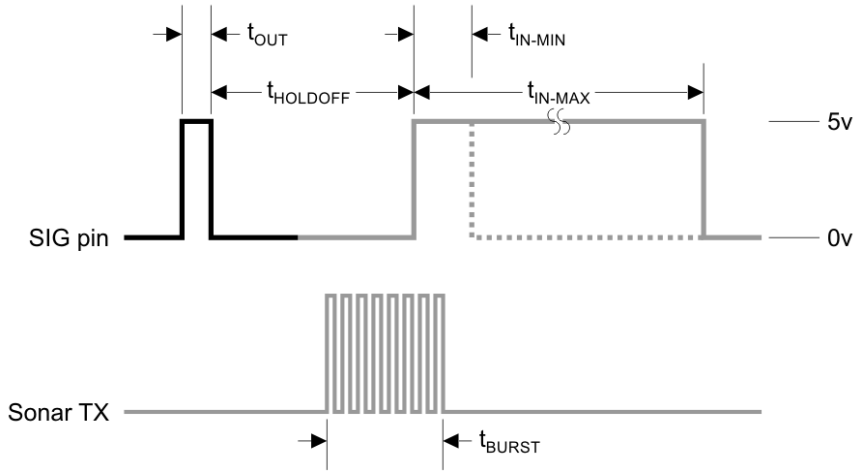


Dimensions



Communication Protocol

The PING))) sensor detects objects by emitting a short ultrasonic burst and then "listening" for the echo. Under control of a host microcontroller (trigger pulse), the sensor emits a short 40 kHz (ultrasonic) burst. This burst travels through the air, hits an object and then bounces back to the sensor. The PING))) sensor provides an output pulse to the host that will terminate when the echo is detected, hence the width of this pulse corresponds to the distance to the target.

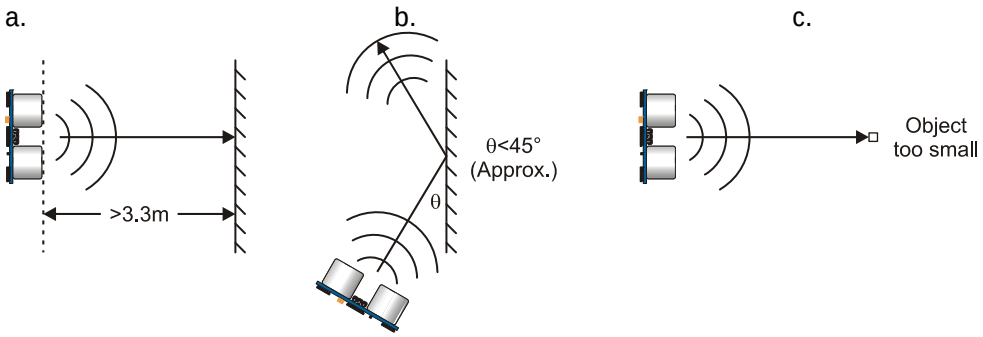


▬	Host Device	Input Trigger Pulse	t_{OUT}	2 µs (min), 5 µs typical
▬	PING))) Sensor	Echo Holdoff	$t_{HOLDOFF}$	750 µs
		Burst Frequency	t_{BURST}	200 µs @ 40 kHz
		Echo Return Pulse Minimum	t_{IN-MIN}	115 µs
		Echo Return Pulse Maximum	t_{IN-MAX}	18.5 ms
		Delay before next measurement		200 µs

Practical Considerations for Use

Object Positioning

The PING))) sensor cannot accurately measure the distance to an object that: a) is more than 3 meters away, b) that has its reflective surface at a shallow angle so that sound will not be reflected back towards the sensor, or c) is too small to reflect enough sound back to the sensor. In addition, if your PING))) sensor is mounted low on your device, you may detect sound reflecting off of the floor.



Target Object Material

In addition, objects that absorb sound or have a soft or irregular surface, such as a stuffed animal, may not reflect enough sound to be detected accurately. The PING))) sensor will detect the surface of water, however it is not rated for outdoor use or continual use in a wet environment. Condensation on its transducers may affect performance and lifespan of the device.

Air Temperature

Temperature has an effect on the speed of sound in air that is measurable by the PING))) sensor. If the temperature (°C) is known, the formula is:

$$C_{air} = 331.5 + (0.6 \times T_C)\ m/s$$

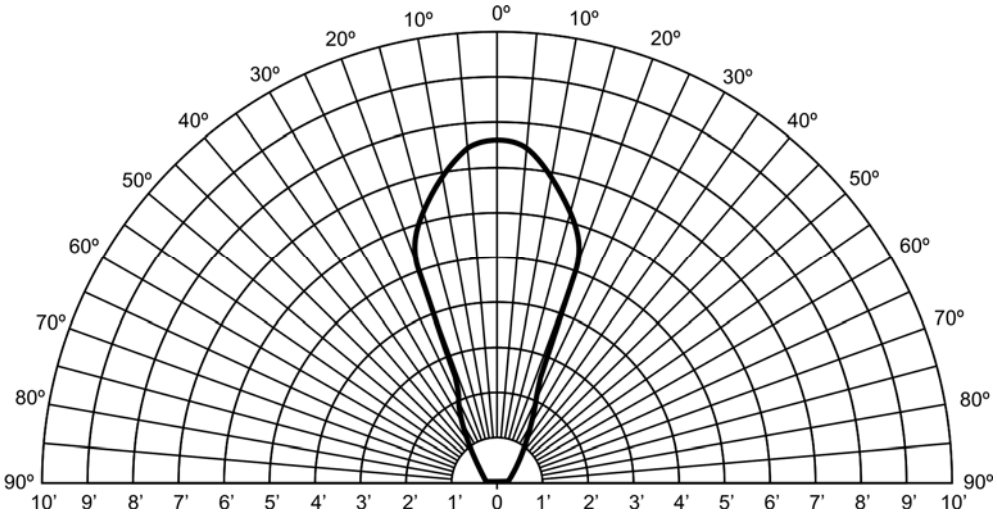
The percent error over the sensor's operating range of 0 to 70 ° C is significant, in the magnitude of 11 to 12 percent. The use of conversion constants to account for air temperature may be incorporated into your program (as is the case in the example BS2 program given in the Example Programs section below). Percent error and conversion constant calculations are introduced in Chapter 2 of *Smart Sensors and Applications*, a Stamps in Class text available for download from the 28029 product page at www.parallax.com.

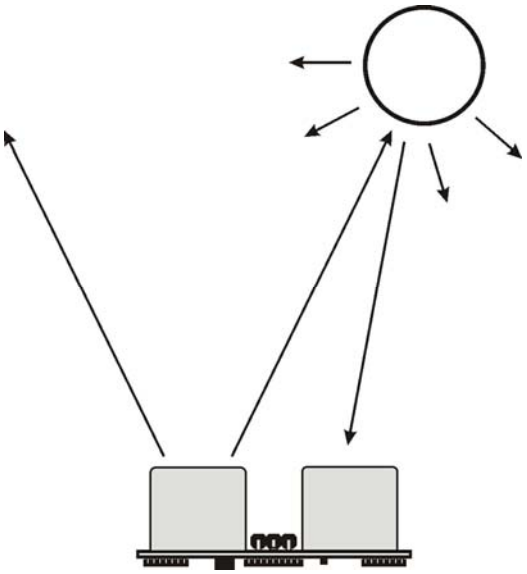
Test Data

The test data on the following pages is based on the PING))) sensor, tested in the Parallax lab, while connected to a BASIC Stamp microcontroller module. The test surface was a linoleum floor, so the sensor was elevated to minimize floor reflections in the data. All tests were conducted at room temperature, indoors, in a protected environment. The target was always centered at the same elevation as the PING))) sensor.

Test 1

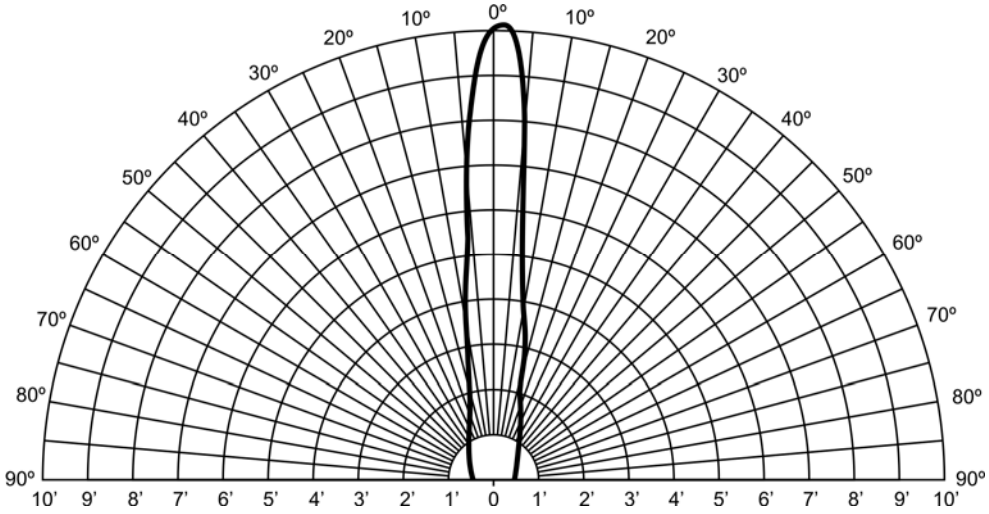
Sensor Elevation: 40 in. (101.6 cm)
Target: 3.5 in. (8.9 cm) diameter cylinder, 4 ft. (121.9 cm) tall – vertical orientation

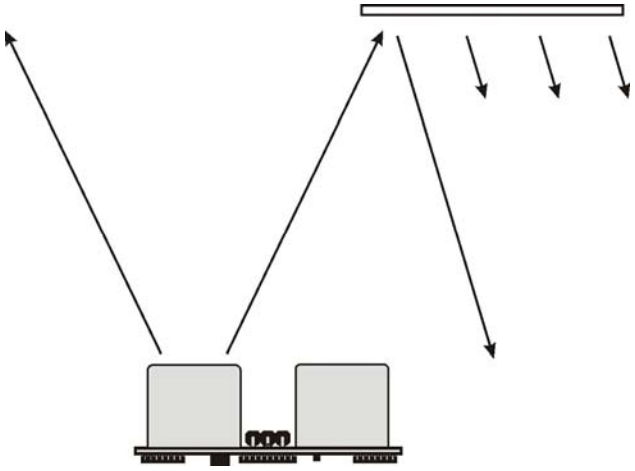




Test 2

Sensor Elevation: 40 in. (101.6 cm)
Target: 12 in. x 12 in. (30.5 cm x 30.5 cm) cardboard, mounted on 1 in. (2.5 cm) pole
Target positioned parallel to backplane of sensor

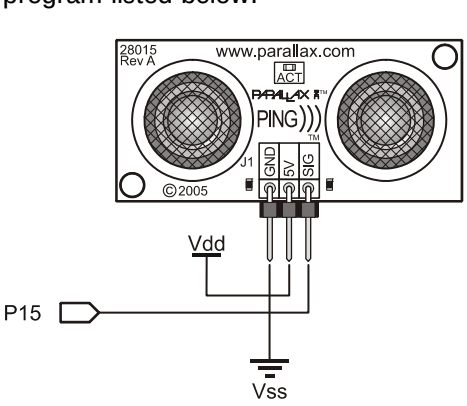


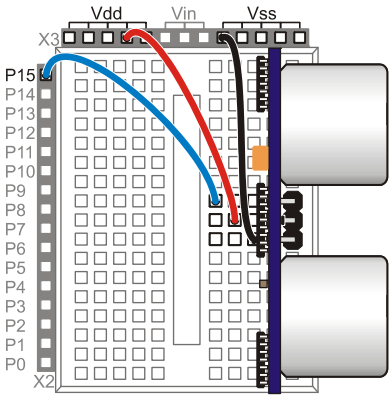


Example Programs

BASIC Stamp 2

This circuit allows you to quickly connect your PING))) sensor to a BASIC Stamp® 2 via the Board of Education® breadboard area. The PING))) module's GND pin connects to Vss, the 5 V pin connects to Vdd, and the SIG pin connects to I/O pin P15. This circuit will work with the example BASIC Stamp program listed below.

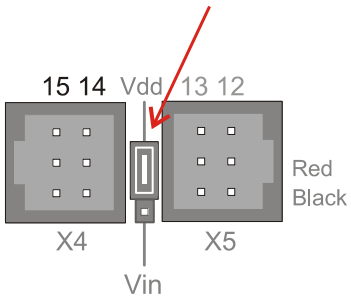




Extension Cable and Port Cautions for the Board of Education

If you are connecting your PING))) sensor to a Board of Education platform using an extension cable, follow these steps:

1. When plugging the cable onto the PING))) sensor, connect Black to GND, Red to 5 V, and White to SIG.
2. Check to see if your Board of Education servo ports have a jumper, as shown at right.
3. If your Board of Education servo ports have a jumper, set it to Vdd as shown. Then plug the cable into the port, matching the wire color to the labels next to the port.
4. If your Board of Education servo ports do not have a jumper, **do not use them with the PING))) sensor**. These ports only provide Vin, not Vdd, and this may damage your PING))) sensor. Go to the next step.
5. Connect the cable directly to the breadboard with a 3-pin header as shown above. Then, use jumper wires to connect Black to Vss, Red to Vdd, and White to I/O pin P15.



Board of Education Servo Port Jumper, Set to Vdd

Example Program: PingMeasureCmAndIn.bs2

This program for the BASIC Stamp 2 displays distance measurements in both inches and centimeters in the BASIC Stamp Debug Terminal. The example program can be downloaded from the 28015 product page at www.parallax.com. The BASIC Stamp Editor software, which includes the Debug Terminal, is a free download from www.parallax.com/basicstampsoftware.

```
' Smart Sensors and Applications - PingMeasureCmAndIn.bs2
' Measure distance with Ping))) sensor and display in both in & cm

' {$STAMP BS2}
' {$PBASIC 2.5}

' Conversion constants for room temperature measurements.
CmConstant    CON   2260
InConstant    CON   890

cmDistance  VAR   Word
inDistance  VAR   Word
time        VAR   Word

DO

  PULSOUT 15, 5
  PULSIN 15, 1, time

  cmDistance = cmConstant ** time
  inDistance = inConstant ** time

  DEBUG HOME, DEC3 cmDistance, " cm"
  DEBUG CR, DEC3 inDistance, " in"

  PAUSE 100

LOOP
```

Propeller Microcontroller

```
{{
**************************************
*       Ping))) Object V1.1          *
*      (C) 2006 Parallax, Inc.       *
* Author:  Chris Savage & Jeff Martin *
* Started: 05-08-2006                *
**************************************

Interface to Ping))) sensor and measure its ultrasonic travel time.  Measurements can be in
units of time or distance.  Each method requires one parameter, Pin, that is the I/O pin that
is connected to the Ping)))'s signal line.

  ┌───────────────────┐
  │┌───┐         ┌───┐│    Connection To Propeller
  ││ ▪ │ PING))) │ ▪ ││    Remember PING))) Requires
  │└───┘         └───┘│    +5V Power Supply
  │    GND +5V SIG    │
  └─────┬───┬───┬─────┘
        │  ┌┘   ┌ 1K
        │  └┘   └┤ Pin
        ┴

-------------------------REVISION HISTORY-------------------------
 v1.1 - Updated 03/20/2007 to change SIG resistor from 10K to 1K
}}

CON
  TO_IN = 73_746                                                    ' Inches
  TO_CM = 29_034                                                    ' Centimeters

PUB Ticks(Pin) : Microseconds | cnt1, cnt2
''Return Ping)))'s one-way ultrasonic travel time in microseconds

  outa[Pin]~                                                        ' Clear I/O Pin
  dira[Pin]~~                                                       ' Make Pin Output
  outa[Pin]~~                                                       ' Set I/O Pin
  outa[Pin]~                                                        ' Clear I/O Pin (> 2 µs pulse)
  dira[Pin]~                                                        ' Make I/O Pin Input
  waitpne(0, |< Pin, 0)                                             ' Wait For Pin To Go HIGH
  cnt1 := cnt                                                       ' Store Current Counter Value
  waitpeq(0, |< Pin, 0)                                             ' Wait For Pin To Go LOW
  cnt2 := cnt                                                       ' Store New Counter Value
  Microseconds := (||(cnt1 - cnt2) / (clkfreq / 1_000_000)) >> 1    ' Return Time in µs

PUB Inches(Pin) : Distance
''Measure object distance in inches

  Distance := Ticks(Pin) * 1_000 / TO_IN                             ' Distance In Inches

PUB Centimeters(Pin) : Distance
''Measure object distance in centimeters

  Distance := Millimeters(Pin) / 10                                  ' Distance In Centimeters

PUB Millimeters(Pin) : Distance
''Measure object distance in millimeters

  Distance := Ticks(Pin) * 10_000 / TO_CM                            ' Distance In Millimeters
```

The ping.spin object is used in an example project with the Parallax 4 x 20 Serial LCD (#27979) to display distance measurements. The complete Project Archive can be downloaded from the Propeller Object Exchange at http://obex.parallax.com. The Propeller Tool software can be downloaded from www.parallax.com/propellertool.

```
_____________________________________
Parallax Propeller Chip Project Archive
_____________________________________

 Project :  "ping_demo"

Archived :  Tuesday, December 18, 2007 at 3:29:46 PM

    Tool :  Propeller Tool version 1.05.8


            ping_demo.spin
               |
               |--Debug_Lcd.spin
               |     |
               |     |--Serial_Lcd.spin
               |     |     |
               |     |     |--Simple_Serial.spin
               |     |
               |     |
               |     |--Simple_Numbers.spin
               |
               |
               |--ping.spin
```

Reources and Downloads

For additional example code downloads and links to videos, tutorials and robotics projects that use the Ping))) Ultrasonic Distance Sensor, visit www.parallax.com and search “28015.”

Product Change Notice

Rev A: original release
Rev B: resonator added to the SX-28 co-processor circuit. No changes to functionality
Rev C: SX-28 co-processor changed to PIC16F57. No changes to functionality.

Revision History

Version 2.0: Added revision history. Removed Javelin Stamp examples. Added URLs for programming software. Added Product Change Notice section with PCB revision information.

Ultrasonic ranging module : HC-SR04

Specifications:

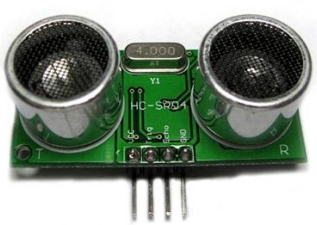
power supply :5V DC

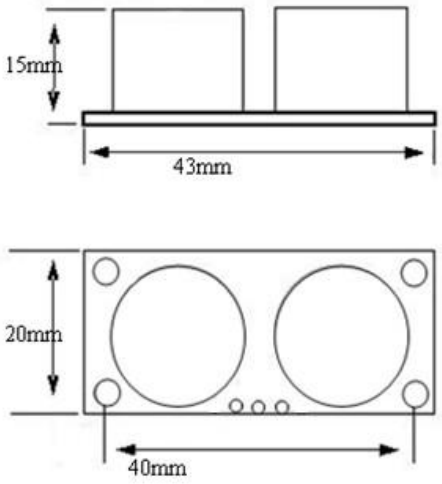
quiescent current : <2mA

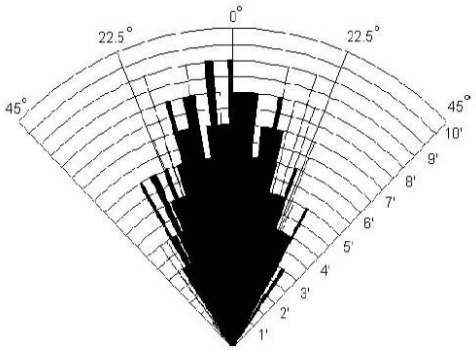
effectual angle: <15°

ranging distance : 2cm – 500 cm

resolution : 0.3 cm

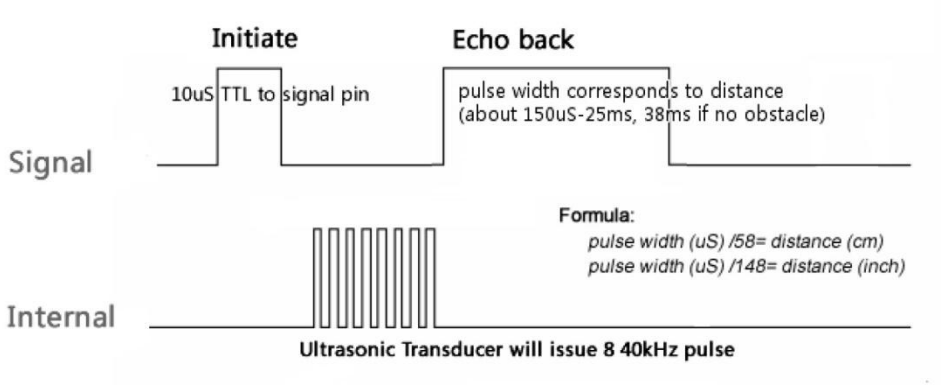






Practical test of performance,
Best in 30 degree angle

Sequence chart



A short ultrasonic pulse is transmitted at the time 0, reflected by an object. The senor receives this signal and converts it to an electric signal. The next pulse can be transmitted when the echo is faded away. This time period is called cycle period. The recommend cycle period should be no less than 50ms. If a 10µs width trigger pulse is sent to the signal pin, the Ultrasonic module will output eight 40kHz ultrasonic signal and detect the echo back. The measured distance is proportional to the echo pulse width and can be calculated by the formula above. If no obstacle is detected, the output pin will give a 38ms high level signal.

Library:
http://iteadstudio.com/store/images/produce/Robot/HCSR04/Ultrasonic.rar

Módulo transmisor de alta potencia STX882

1-Descripción

El STX882 es un transmisor de bajo costo, tamaño pequeño, ultra-alta potencia, bajo en amónicos mediante la modulación ASK. Tiene una gran estabilidad, a 3.6V alcanza una potencia de 50mW.

El módulo puede ser conectado directamente a un microcontrolador. Por lo tanto, es más conveniente para el desarrollo de productos inalámbricos.
Opcionalmente están otros dispositivos de conexión rápida como el HT12E.

2-Características

- Frecuencia de operación 433MHz
- Modulación ASK
- Tamaño reducido
- Alta potencia
- Rango de alcance 300m
- Estabiliza la frecuencia

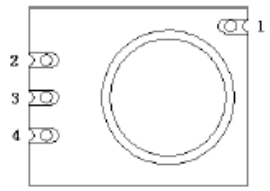
3-Aplicaciones

- Control inalámbrico de puertas
- Alarmas inalámbricas
- Control industrial inalámbrico
- Transmisión inalámbrica de datos, etc.

4-Especificaciones eléctricas

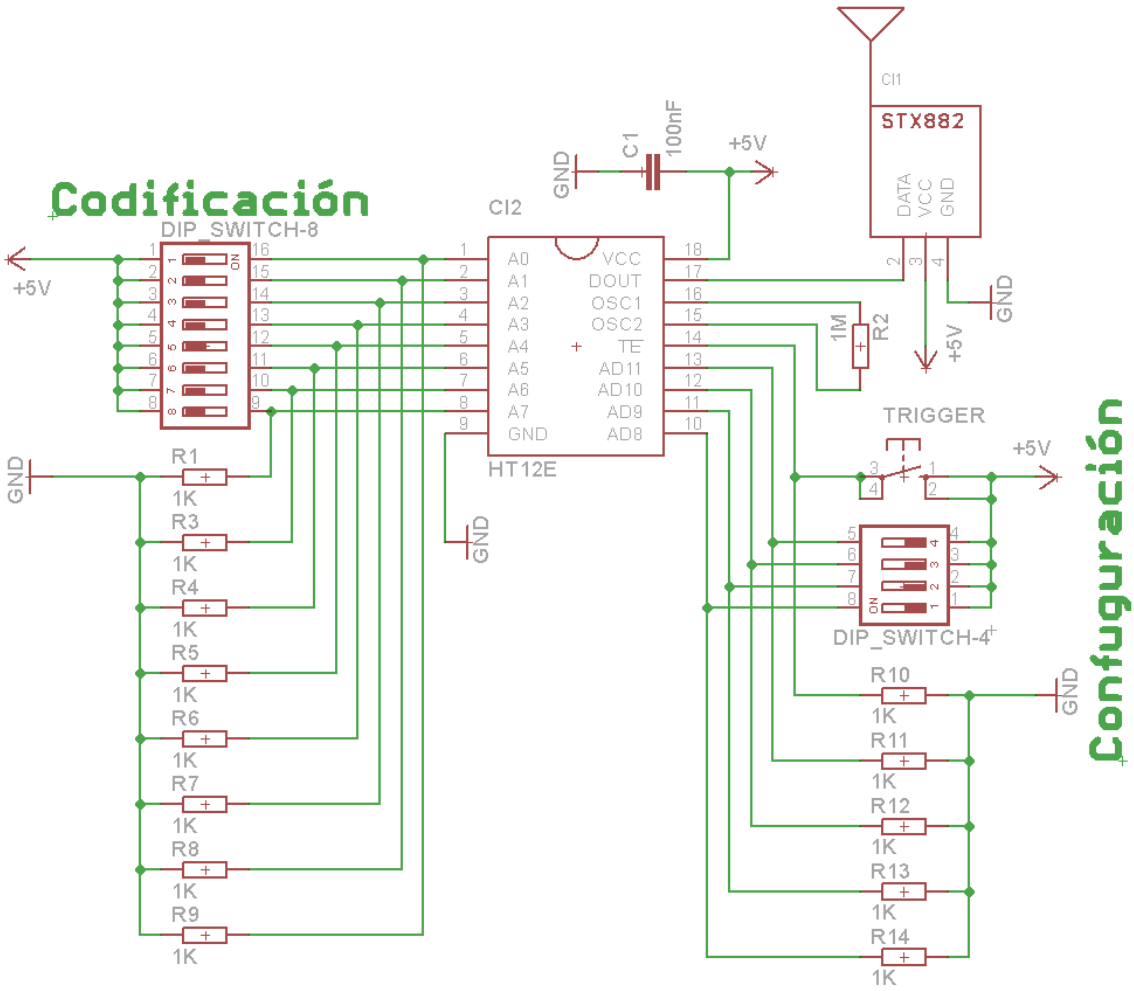
Parameter	Min.	Typ.	Max	Unit	Conditions
Operation conditions					
Supply Voltage	1.2	3.0	6	V	
Operating Temperature Range	-20	25	70	℃	
Current consumption					
TX current		20		mA	@5V,13dBm
Sleep Current		≤0.01		uA	@DATA As Low level
RF parameters					
Frequency Range	433.82	433.92	434.02	MHZ	@433MHZ
	314.9	315	315.1	MHZ	@315MHZ
RF power	12	13	13.5	dBm	@2.4V
	14	15	15.5	dBm	@3V
	19.5	20	20.5	dBm	@5V
Air rate	0.1		9.6	Kbps	

5-Configuración de pines

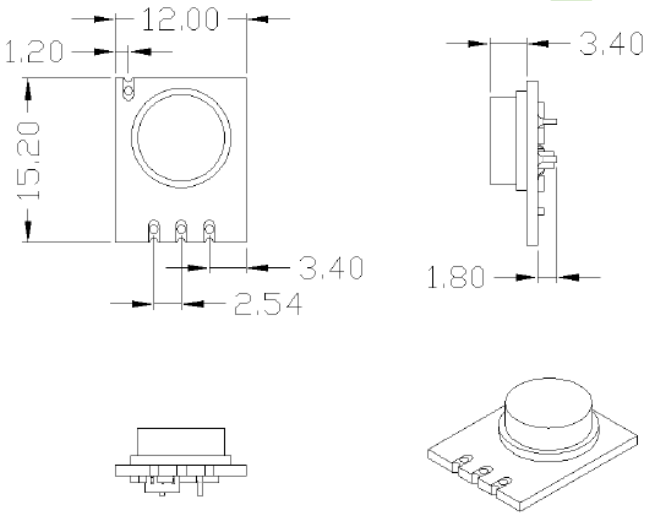


Pin Number	Pin Definitions	Description
1	ANT	Antenna input
2	DATA	Data input
3	VCC	Positive power supply
4	GND	Connected to power ground

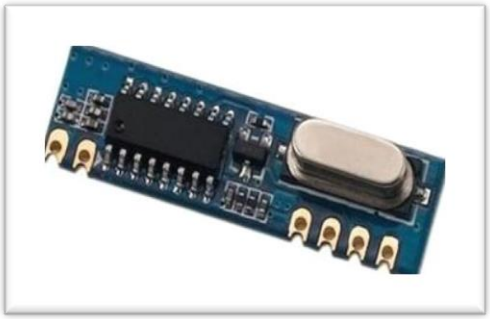
6-Diagrama de conexión



5-Dimenciones (mm)



Módulo Receptor Súper Heterodino SRX887



1-Descripción

El STX887 es un receptor con un poderoso motor súper heterodino, con gran estabilidad y anti-interferencia, tamaño pequeño, alta eficiencia a bajo costo, con certificación ROHS, FCC y CE.

El módulo puede ser conectado directamente al microcontrolador. Por lo tanto, es más conveniente para el desarrollo de productos inalámbricos.
Opcionalmente están otros dispositivos de conexión rápida como el HT12D.

2-Características

- Frecuencia de operación 433MHz
- Modulación ASK/OOK
- Tamaño reducido
- Anti-Interferencia
- Bajo consumo en modo sleep (1uA)
- Estabiliza y rehabilita la frecuencia

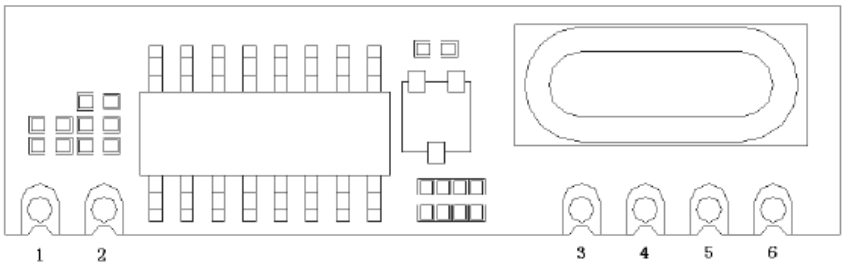
3-Aplicaciones

- Control inalámbrico de puertas
- Alarmas inalámbricas
- Control industrial inalámbrico
- Transmisión inalámbrica de datos, etc.

4-Especificaciones eléctricas

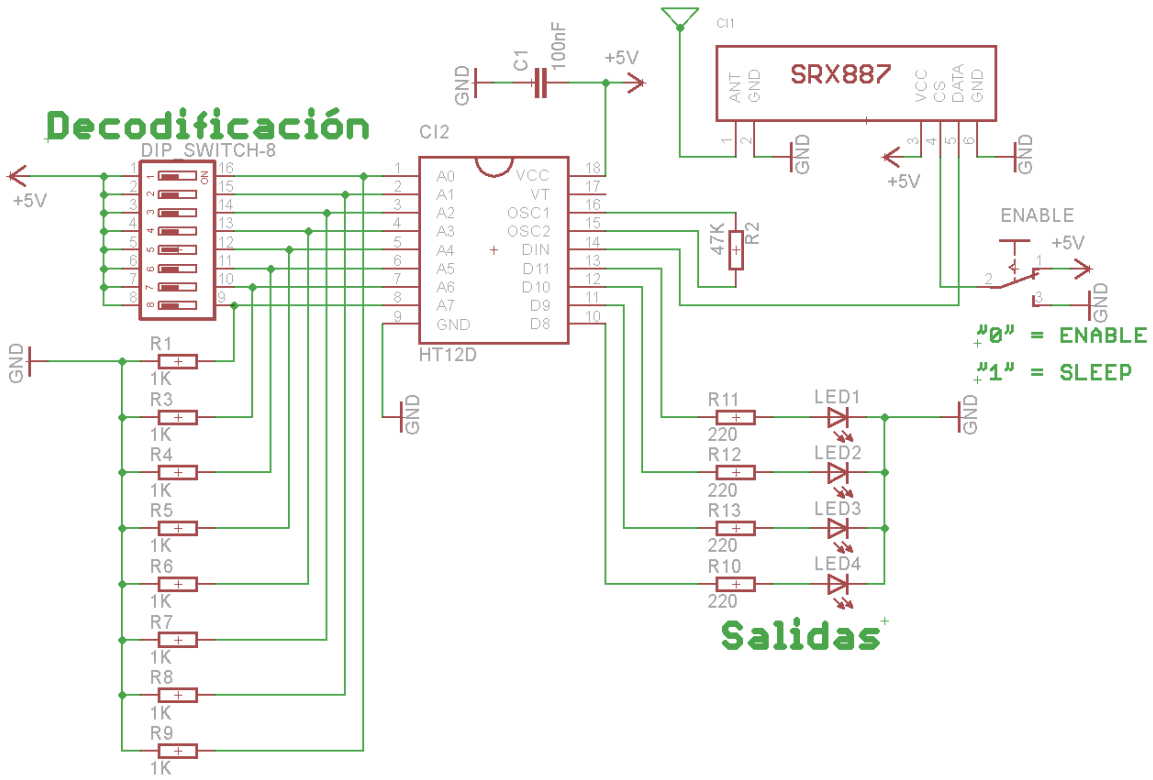
Parameter	Min.	Typ.	Max	Unit	Conditions
Operation conditions					
Supply Voltage	3.0	5	5.5	V	
Operating Temperature Range	-30		85	℃	
Latency			20	ms	@315MHZ
			9	ms	@433@HZ
Current consumption					
Consuming current		2.9		mA	@315MHZ
		4.0		mA	@433MHZ
		<1		uA	@CS=1 Or vacant
RF parameters					
Frequency Range	433.82	433.92	434.02	MHZ	@433MHZ
	314.9	315	315.1	MHZ	@315MHZ
Sensitivity		-110		dBm	@1.2Kbps
Air rate	0.1		9.6	kbps	
Receiver bandwidth		200		KHz	

5-Configuración de pines

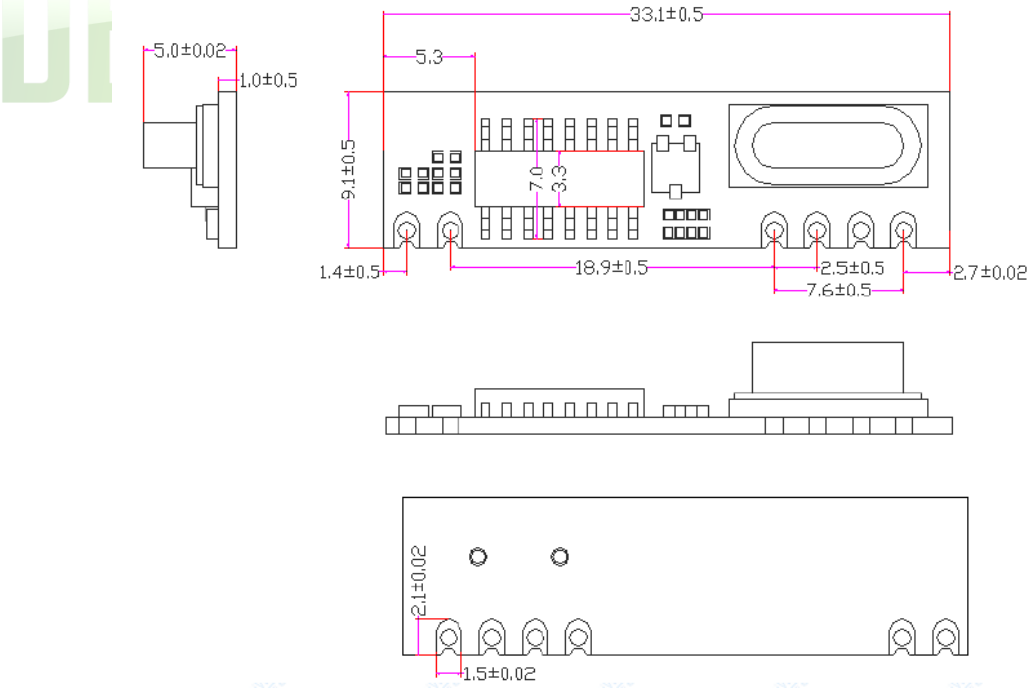


Pin Number	Pin Definitions	Description
1	ANT	Matching 50 ohm antenna
2	GND	Connected to power ground
3	VCC	Positive power supply
4	CS	Low level to enable the module, high level to enter sleep mode
5	DATA	Data output pin
6	GND	Connected to power ground

6-Diagrama de conexión



5-Dimenciones (mm)



NOTAS:

- El pin “TE” del HT12E, debe activarse al menos 100mili-segundos para activar la salida de datos, lógica negativa.

- El pin “CS” del módulo SRX887 debe estar a “0” para habilitarse y en “1” para inactividad.

TOWER PRO SERVO SPECIFICATION

Type	Dimension	Weight	Stall torque	Operating speed	Operating voltage	Temperature rang	Dead band width	Gear type	apply for
SG50	21.5x11.7x25.1mm	5	0.6kg/cm(4.8V)	0.1sec/60degree(4.8v)	4.8V	0℃_ 55℃	10μs	nylon	Helicopter,3D-flyer,F3A
SG51R	21.5x11.7x25.1mm	5	0.7kg/cm(4.8V)	0.1sec/60degree(4.8v)	4.8V	0℃_ 55℃	10μs	nylon	
SG90	23x12.2x29mm	9g	1.8kg/cm(4.8V)	0.1sec/60degree(4.8v)	4.8V	0℃_ 55℃	10μs	nylon	
SG91R	23x12.2x29mm	9g	1.8kg/cm(4.8V)	0.1sec/60degree(4.8v)	4.8V	0℃_ 55℃	10μs	nylon	
SG5010	40.2x20.2x43.2mm	38g	5.5kg/cm(4.8V) 6.5kg/cm(6V)	0.2sec/60degree(4.8v) 0.16sec/60degree(6v)	4.8-6V	0℃_ 55℃	10μs	nylon	cars&airplane
MG945	40.7*19.7*42.9mm	55g	10kg/cm(4.8V), 12kg/cm(6V)	0.23sec/60degree(4.8v) 0.2sec/60degree(6.0v)	4.8-7.2V	0℃_ 55℃	5μs	metal	1/8 buggy, cars
MG945R	40.7*19.7*42.9mm	55g	10kg/cm(4.8V), 12kg/cm(6V)	0.23sec/60degree(4.8v) 0.2sec/60degree(6.0v)	4.8-7.2V	0℃_ 55℃	5μs	metal	
MG995	40.7*19.7*42.9mm	55g	8.5kg/cm(4.8V), 10kg/cm(6V)	0.20sec/60degree(4.8v) 0.16sec/60degree(6.0v)	4.8-7.2V	0℃_ 55℃	5μs	metal	1/10 buggy, cars
MG995R	40.7*19.7*42.9mm	55g	8.5kg/cm(4.8V), 10kg/cm(6V)	0.20sec/60degree(4.8v) 0.16sec/60degree(6.0v)	4.8-7.2V	0℃_ 55℃	5μs	metal	
9805BB	66x30.2x64.4mm	160g	20kg/cm(4.8V), 25kg/cm(6V)	0.20sec/60degree(4.8v) 0.16sec/60degree(6v)	4.8-7.2V	0℃_ 55℃	5μs	nylon	Cars
MG16R	29*11.2*29mm	18.8g	2.7kg.cm(4.8v) ,2.9kg/cm(6.0)	0.10sec/60degree(4.8v) 0.08sec/60degree(6v)	4.8v-6V	0℃_ 55℃	5μs	metal	Helicopter
MG946R	40.7*19.7*42.9mm	55g	10.5kg/cm(4.8V), 13kg/cm(6V)	0.20sec/60degree(4.8v) 0.17sec/60degree(6.0v)	4.8-7.2V	0℃_ 55℃	5μs	metal	Gasoline engine plane, train-flyer
MG996R	40.7*19.7*42.9mm	55g	9.4kg/cm(4.8V), 11kg/cm(6V)	0.17sec/60degree(4.8v) 0.14sec/60degree(6v)	4.8-7.2V	0℃_ 55℃	5μs	metal	
MG955H	40.7*19.7*42.9mm	55g	7.0kg/cm(4.8V), 8.5kg/cm(6V)	0.17sec/60degree(4.8v) 0.14sec/60degree(6v)	4.8-7.2V	0℃_ 55℃	5μs	metal	Helicopter
MG956R	40.7*19.7*42.9mm	55g	7.5kg/cm(4.8V), 9kg/cm(6V)	0.15sec/60degree(4.8v) 0.12sec/60degree(6v)	4.8-7.2V	0℃_ 55℃	5μs	metal& 1 nylon gear	

Marks: "R" means RoHS, It is a material standard for European area, means non-toxic,non-leaded,no harm to enviroment

National Semiconductor

May 1989

54LS32/DM54LS32/DM74LS32 Quad 2-Input OR Gates

General Description

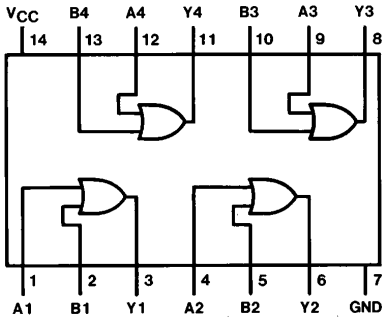
This device contains four independent gates each of which performs the logic OR function.

Features

- Alternate Military/Aerospace device (54LS32) is available. Contact a National Semiconductor Sales Office/ Distributor for specifications.

Connection Diagram

Dual-In-Line Package



TL/F/6361–1

Order Number 54LS32DMQB, 54LS32FMQB, 54LS32LMQB, DM54LS32J, DM54LS32W, DM74LS32M or DM74LS32N
See NS Package Number E20A, J14A, M14A, N14A or W14B

Function Table

$Y = A + B$

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

H = High Logic Level
L = Low Logic Level

  RRD-B30M105/Printed in U. S. A.

Absolute Maximum Ratings (Note)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	7V
Input Voltage	7V
Operating Free Air Temperature Range	
DM54LS and 54LS	−55°C to +125°C
DM74LS	0°C to +70°C
Storage Temperature Range	−65°C to +150°C

Note: *The "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. The device should not be operated at these limits. The parametric values defined in the "Electrical Characteristics" table are not guaranteed at the absolute maximum ratings. The "Recommended Operating Conditions" table will define the conditions for actual device operation.*

Recommended Operating Conditions

Symbol	Parameter	DM54LS32			DM74LS32			Units
		Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply Voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High Level Input Voltage	2			2			V
V_{IL}	Low Level Input Voltage			0.7			0.8	V
I_{OH}	High Level Output Current			−0.4			−0.4	mA
I_{OL}	Low Level Output Current			4			8	mA
T_A	Free Air Operating Temperature	−55		125	0		70	°C

Electrical Characteristics over recommended operating free air temperature range (unless otherwise noted)

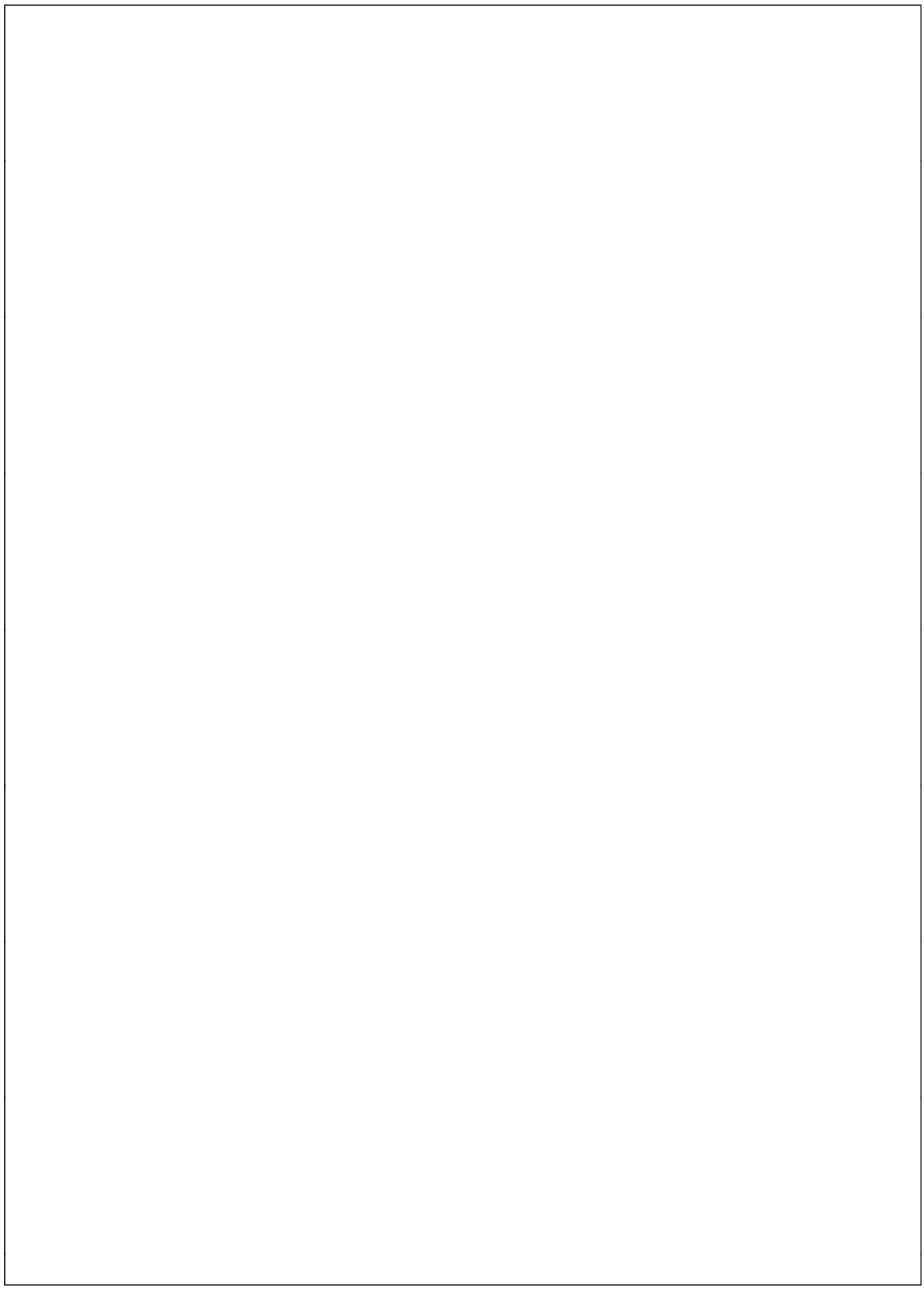
Symbol	Parameter	Conditions		Min	Typ (Note 1)	Max	Units
V_I	Input Clamp Voltage	V_{CC} = Min, I_I = −18 mA				−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, I_{OH} = Max, V_{IH} = Min	DM54	2.5	3.4		V
			DM74	2.7	3.4		
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = Max, V_{IL} = Max	DM54		0.25	0.4	V
			DM74		0.35	0.5	
		I_{OL} = 4 mA, V_{CC} = Min	DM74		0.25	0.4	
I_I	Input Current @ Max Input Voltage	V_{CC} = Max, V_I = 7V				0.1	mA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.7V				20	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V				−0.36	mA
I_{OS}	Short Circuit Output Current	V_{CC} = Max (Note 2)	DM54	−20		−100	mA
			DM74	−20		−100	
I_{CCH}	Supply Current with Outputs High	V_{CC} = Max			3.1	6.2	mA
I_{CCL}	Supply Current with Outputs Low	V_{CC} = Max			4.9	9.8	mA

Switching Characteristics at V_{CC} = 5V and T_A = 25°C (See Section 1 for Test Waveforms and Output Load)

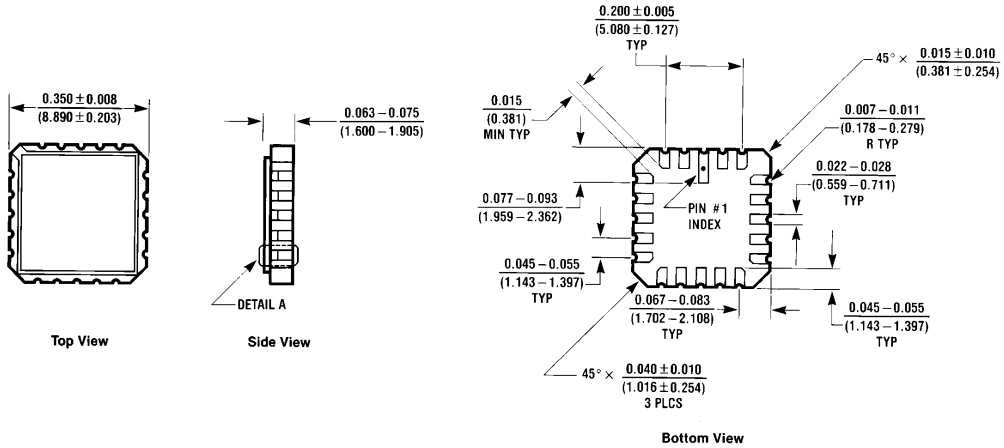
Symbol	Parameter	R_L = 2 kΩ				Units
		C_L = 15 pF		C_L = 50 pF		
		Min	Max	Min	Max	
t_{PLH}	Propagation Delay Time Low to High Level Output	3	11	4	15	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	3	11	4	15	ns

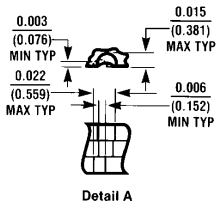
Note 1: All typicals are at V_{CC} = 5V, T_A = 25°C.

Note 2: Not more than one output should be shorted at a time, and the duration should not exceed one second.



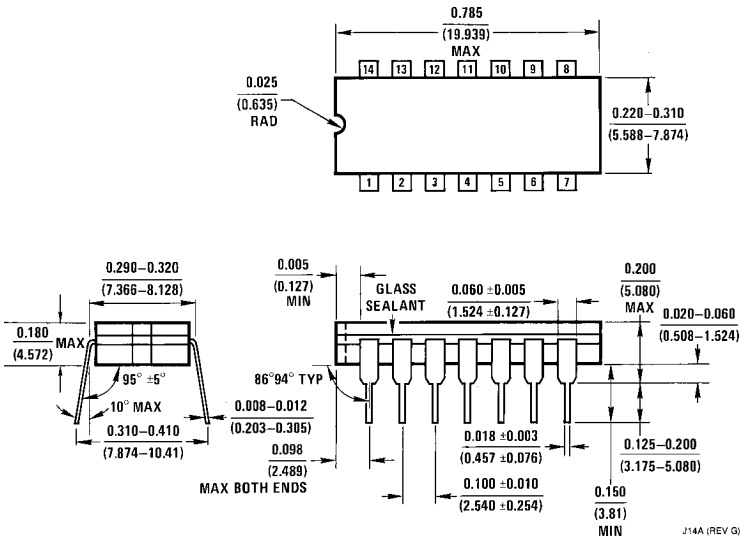
Physical Dimensions inches (millimeters)





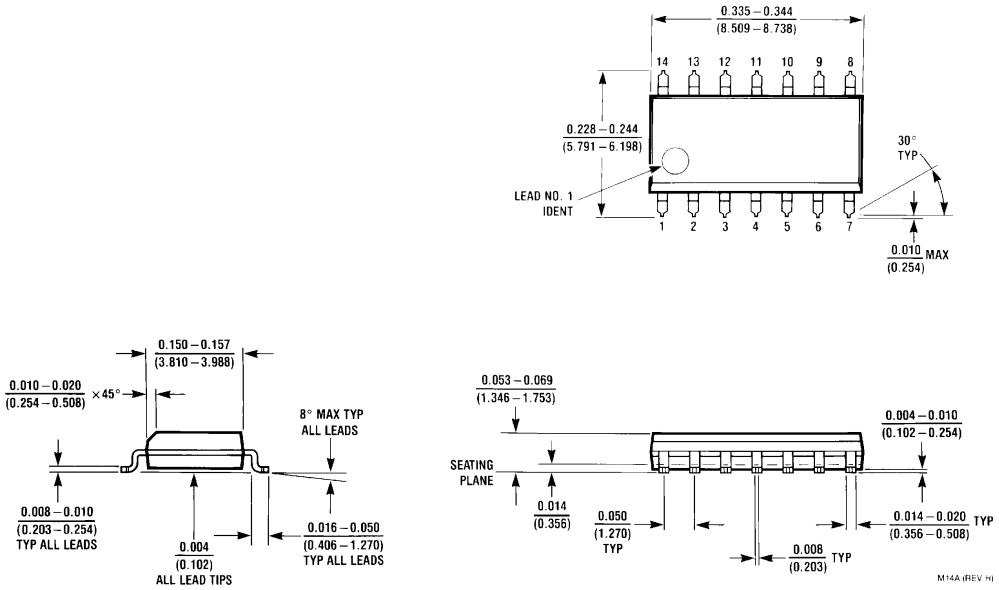
E20A (REV D)

Ceramic Leadless Chip Carrier Package (E)
Order Number 54LS32LMQB
NS Package Number E20A

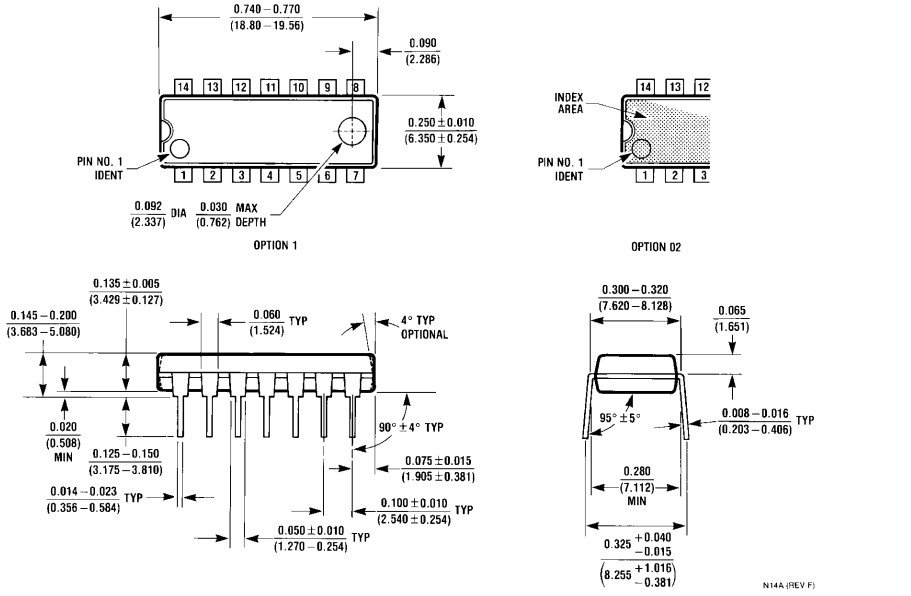


14-Lead Ceramic Dual-In-Line Package (J)
Order Number 54LS32DMQB or DM54LS32J
NS Package Number J14A

Physical Dimensions inches (millimeters)

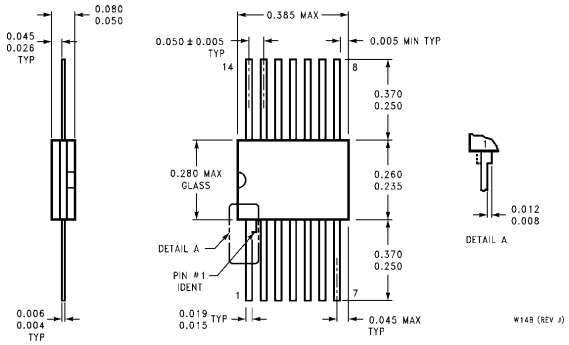


14-Lead Small Outline Molded Package (M)
Order Number DM74LS32M
NS Package Number M14A



14-Lead Molded Dual-In-Line Package (N)
Order Number DM74LS32N
NS Package Number N14A

Physical Dimensions inches (millimeters) (Continued)



14-Lead Ceramic Flat Package (W)
Order Number 54LS32FMQB or DM54LS32W
NS Package Number W14B

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

National Semiconductor was acquired by Texas Instruments.

http://www.ti.com/corp/docs/investor_relations/pr_09_23_2011_national_semiconductor.html

This file is the datasheet for the following electronic components:

54LS32FMQB - http://www.ti.com/product/54ls32fmqb?HQS=TI-null-null-dscatalog-df-pf-null-wwe

54LS32LMQB - http://www.ti.com/product/54ls32lmqb?HQS=TI-null-null-dscatalog-df-pf-null-wwe

DM74LS32N - http://www.ti.com/product/dm74ls32n?HQS=TI-null-null-dscatalog-df-pf-null-wwe

DM74LS32M - http://www.ti.com/product/dm74ls32m?HQS=TI-null-null-dscatalog-df-pf-null-wwe

DM74LS32M - http://www.ti.com/product/dm74ls32m?HQS=TI-null-null-dscatalog-df-pf-null-wwe

54LS32FMQB - http://www.ti.com/product/54ls32fmqb?HQS=TI-null-null-dscatalog-df-pf-null-wwe

54LS32DMQB - http://www.ti.com/product/54ls32dmqb?HQS=TI-null-null-dscatalog-df-pf-null-wwe

54LS32DMQB - http://www.ti.com/product/54ls32dmqb?HQS=TI-null-null-dscatalog-df-pf-null-wwe

54LS32LMQB - http://www.ti.com/product/54ls32lmqb?HQS=TI-null-null-dscatalog-df-pf-null-wwe

DM74LS32M - http://www.ti.com/product/dm74ls32m?HQS=TI-null-null-dscatalog-df-pf-null-wwe

DM74LS32N - http://www.ti.com/product/dm74ls32n?HQS=TI-null-null-dscatalog-df-pf-null-wwe

DM74LS32N - http://www.ti.com/product/dm74ls32n?HQS=TI-null-null-dscatalog-df-pf-null-wwe