UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE TECNOLOGÍA

CARRERA DE MECÁNICA AUTOMOTRIZ

TRABAJO DE APLICACION:

NIVEL LICENCIATURA

“SISTEMA DE ADVERTENCIA PARA LA SEGURIDAD ACTIVA EN PUNTOS CIEGOS DE CAMIONES MEDIANTE UN RADAR DE PROXIMIDAD”

Presentado por Univ. JAVIER AGUILA GUTIERREZ

La Paz – Bolivia

2019

DEDICATORIA

Dedico a mis señores padres, que han sido y son la inspiración y el aliento a seguir adelante y dándome fortaleza para lograr mí objetivo.

AGRADECIMIENTO

Agradecer:

A Dios por hacerme una mejor persona, por permitirme ser parte de LA FACULTAD DE TECNOLOGÍA y por acompañarme en cada uno de mis caminos.

A los tutores por el apoyo, guía y acompañamiento durante el presente trabajo.

A la UMSA y el plantel docente, por la institución que me ha formado.

Índice

Índice De Figuras

1. INTRODUCCIÓN

El sector automotriz tiene un rol importante dentro de la economía del país, su desarrollo genera ingresos fiscales para el estado vía aranceles e impuestos y crea fuentes de empleo durante los procesos de producción y las actividades relacionadas al comercio del mismo. Resulta por lo tanto oportuno hacer un diagnóstico del sector, especialmente dentro del contexto de las repercusiones en la seguridad vial.

Teniendo en cuenta el incesante aumento del tráfico por la carretera, con el consiguiente incremento de los problemas viales, ambientales y considerando que la potencia de motores como la de los camiones. Les permite, así como subir las pendientes con facilidad, alcanzar en llano velocidades excesivas que no son compatibles con otros componentes como frenos y neumáticos. Esta situación ha creado la necesidad de implementar un sistema de advertencia para la seguridad activa en camiones, mediante un radar de proximidad para puntos ciegos en carretera, para bridar a la sociedad un dispositivo con el cual reduzca el índice de accidentes mejorando la seguridad vial.

El proyecto se desarrolló, pensando en la seguridad activa para camiones de todo el país.

El proyecto de control de distancia brinda al usuario y al conductor, la ventaja de poder prevenir accidentes de tránsito por objetos en los puntos ciegos del camión.

Con el perfeccionamiento de este sistema se espera contribuir de manera significativa a la investigación de nuevas tecnologías para la seguridad del conductor, ocupantes y peatones.

2. PLANTEAMIENTO DEL PROBLEMA

La Organización Mundial de la Salud ha establecido que Bolivia tiene la mayor cantidad de muertes por accidente per cápita de toda la región, junto a Brasil. En el país mueren al año 23,2 personas por cada 100 mil habitantes, cuando el promedio latinoamericano, dice la OMS, es de 15,9 y el de todo el mundo, de 17,4.

Como comparación, en Alemania mueren solo 4,3 personas por cada 100 mil habitantes. En la región, Argentina, Chile y Uruguay ocupan los primeros lugares, con 16 muertes o menos.

En Bolivia se tiene la siguiente cantidad de vehículos.

CLASE DE VEHÍCULO	2015		2016		VARIACIÓN PORCENTUAL
	Número de Vehículos	Participación porcentual	Número de Vehículos	Participación porcentual	
TOTAL	**1.574.552**	**100,0**	**1.711.005**	**100,0**	**8,7**
Automóvil	283.690	18,0	303.733	17,8	7,1
Camión	116.130	7,4	123.929	7,2	6,7
Camioneta	158.567	10,1	173.211	10,1	9,2
Furgón	9.686	0,6	11.360	0,7	17,3
Jeep	58.421	3,7	60.871	3,6	4,2
Microbus	19.330	1,2	19.584	1,1	1,3
Minibús	85.124	5,4	94.358	5,5	10,8
Motocicleta	336.221	21,4	391.219	22,9	16,4
Ómnibus	9.874	0,6	10.863	0,6	10,0
Quadra Track	3.716	0,2	4.239	0,2	14,1
Torpedo	99	0,0	98	0,0	(1,0)
Tracto-Camión	21.252	1,3	22.756	1,3	7,1
Trimóvil-Camión	18	0,0	18	0,0	0,0
Vagoneta	472.424	30,0	494.766	28,9	4,7

Cuadro 1. Parque automotor, según clase de vehículo, 2015 – 2016. Fuente: Registro único para la administración tributaria municipal (RUAT) - (INE)

En nuestra región gran parte de los accidentes son producidos por los conductores de camiones.

Figura 1. Accidente de tránsito de camión sector Chuquiuta - Potosí. La Razón (2019)

Figura 2. Accidente de tránsito ruta Challapata – Potosí. La Razón (2019)

El Instituto Nacional De Estadística Refleja Que:

DESCRIPCION	2014	2015
BOLIVIA	31,782	30,556
Atropellos	5,035	4,760
Colisiones	16,356	15,213
Choque a objeto fijo y vehículo detenido	7,429	7,901
Vuelcos	1,044	770
Embarrancamiento, deslizamiento y encunetamiento	1,138	1,376
Caída de personas - pasajeros	690	523
Incendio de vehículos	90	13
Chuquisaca	1,828	1,385
Atropellos	233	200
Colisiones	873	631
Choque a objeto fijo y vehículo detenido	464	393
Vuelcos	74	30
Embarrancamiento, deslizamiento y encunetamiento	129	108
Caída de personas - pasajeros	35	22
Incendio de vehículos	20	1
La Paz	10,834	10,355
Atropellos	1,941	1,932
Colisiones	5,330	4,737
Choque a objeto fijo y vehículo detenido	2,807	2,847
Vuelcos	296	270
Embarrancamiento, deslizamiento y encunetamiento	345	466
Caída de personas - pasajeros	99	101
Incendio de vehículos	16	2
Cochabamba	2,538	3,041
Atropellos	624	673
Colisiones	1,226	1,442
Choque a objeto fijo y vehículo detenido	406	624
Vuelcos	70	75
Embarrancamiento, deslizamiento y encunetamiento	150	177
Caída de personas - pasajeros	52	50
Incendio de vehículos	10	
Oruro	1,682	1,779
Atropellos	289	300
Colisiones	711	788
Choque a objeto fijo y vehículo detenido	358	401
Vuelcos	204	189

Embarrancamiento, deslizamiento y encunetamiento	79	86
Caída de personas - pasajeros	18	14
Incendio de vehículos	23	1
Potosí	1,382	667
Atropellos	195	123
Colisiones	576	259
Choque a objeto fijo y vehículo detenido	375	178
Vuelcos	118	35
Embarrancamiento, deslizamiento y encunetamiento	108	62
Caída de personas - pasajeros	9	8
Incendio de vehículos	1	2
Tarija	2,170	1,315
Atropellos	253	177
Colisiones	1,176	646
Choque a objeto fijo y vehículo detenido	480	276
Vuelcos	78	61
Embarrancamiento, deslizamiento y encunetamiento	103	123
Caída de personas - pasajeros	79	28
Incendio de vehículos	1	4
Santa Cruz	9,985	10,987
Atropellos	1,329	1,234
Colisiones	5,756	6,168
Choque a objeto fijo y vehículo detenido	2,299	3,033
Vuelcos	172	73
Embarrancamiento, deslizamiento y encunetamiento	182	213
Caída de personas - pasajeros	233	263
Incendio de vehículos	14	3
Beni	1,072	756
Atropellos	146	85
Colisiones	549	403
Choque a objeto fijo y vehículo detenido	168	92
Vuelcos	25	33
Embarrancamiento, deslizamiento y encunetamiento	38	113
Caída de personas - pasajeros	143	30
Incendio de vehículos	3	
Pando	291	271
Atropellos	25	36
Colisiones	159	139
Choque a objeto fijo y vehículo detenido	72	57
Vuelcos	7	4
Embarrancamiento, deslizamiento y encunetamiento	4	28

Caída de personas - pasajeros	22	7
Incendio de vehículos	2	

TABLA 1. Accidentes de tránsito registrados, según departamento y clase de accidentes. Fuente: POLICÍA NACIONAL INSTITUTO NACIONAL DE ESTADÍSTICA (2014-2015)

Muchos de estos accidentes no son registrados ya que solo se tiene daños materiales, en el peor de los casos las consecuencias son trágicas, ya que conlleva el fallecimiento de personas.

En la experiencia como chofer y mecánico se observó que la mayor parte de los accidentes con daños materiales son producidos debido a los puntos ciegos que presenta el vehículo debido a sus dimensiones.

Por lo tanto, el problema lo expresamos de la siguiente manera:

¿Es posible reducir los accidentes de tránsito que están relacionados con la falta de visibilidad del conductor?

3. OBJETIVOS

3.1. OBJETIVO GENERAL

Diseñar un sistema de advertencia para la seguridad activa, utilizando un sensor de proximidad para eliminar la falta de visibilidad en los puntos ciegos de un camión volvo.

3.2. OBJETIVOS ESPECÍFICOS

- Encontrar los puntos ciegos de un camión volvo.
- Diseñar un sistema de radar de proximidad, en base a un Arduino y el sensor HC-SR04, para los puntos ciegos del camión volvo.

- Implementar el sistema de radar de proximidad en los puntos ciegos del camión volvo.
- Verificación mediante una luz testigo en tablero.

4. JUSTIFICACIÓN

4.1. JUSTIFICACIÓN ECONÓMICA

El trabajo desarrollado se presupuestó y su instalación tiene un bajo costo, y este puede en el tiempo resultar muy pero muy económico debido a que el costo de un accidente es mayor. Y se podría evitar instalando este dispositivo.

4.2. JUSTIFICACIÓN TECNOLÓGICA

En este trabajo se refleja conocimientos adquiridos para implementarlo en una necesidad humana. Y se realiza utilizando un Arduino Uno, que es una de las tecnologías que más está creciendo en el campo de la electrónica debido a su versatilidad y fácil manejo. Este trabajo es un aporte tecnológico y puede mejorarse en base al estudio realizado.

4.3. JUSTIFICACIÓN SOCIAL

La necesidad de evitar o reducir las pérdidas humana y materiales, hacen que el trabajo se realce y adquiera la importancia social que para nosotros representa.

5. MARCO TEORICO

5.1. SEGURIDAD

El concepto de seguridad implica garantía, libre, exento de peligro, daño o riesgo; donde peligro signifique riesgo de que suceda algún mal.

Es la acción o mecanismo que garantiza el buen funcionamiento de algo previendo que falle, se frustre o violente.

5.2. SEGURIDAD VIAL

Es la disciplina que estudia y aplica las acciones y mecanismos tendientes a garantizar el buen funcionamiento de la circulación en la vía pública previendo los siniestros viales.

Busca crear las condiciones para que los siniestros viales no ocurran, o en caso que igualmente se produzcan provoquen los menores perjuicios posibles.

5.3. SEGURIDAD PASIVA

En los casos en que es prácticamente imposible evitar un siniestro vial se proyectan los medios para aminorar los efectos perjudiciales de este, tanto para los ocupantes del vehículo como para terceras personas, a conjunto de medios se los conoce como seguridad pasiva. Estos se pueden clasificar en dos tipos:

Seguridad pasiva exterior: es la que actúa como protección en caso de colisión con peatones o ciclistas. La tendencia actual es eliminar de las partes frontales y laterales de la carrocería saliente que pueda ocasionar lesiones o traumatismos en casos de impacto.

Seguridad pasiva interior: incluye todos los elementos y dispositivos diseñados para aminorar los efectos que puedan producir un siniestro vial.

5.4. SEGURIDAD ACTIVA

La seguridad activa es el conjunto de todos los elementos que sirven para dar una mayor estabilidad al vehículo en marcha y reducir al mínimo el

riesgo de accidente, es decir, se trata de elementos eficaces para evitar accidentes. La diferencia con la seguridad pasiva es que los elementos que aminoran los daños de un supuesto accidente, actúan cuando se produce el accidente mientras que los elementos de la seguridad activa tratan de evitar que se produzca. (De la Fuente, 1995)

5.5. SISTEMAS DE SEGURIDAD ACTIVA MÁS IMPORTANTES

5.5.1. NEUMÁTICOS

Ya hablamos de los tipos de neumáticos disponibles en el mercado y ya mencionamos que el dibujo y el material de fabricación de los mismos son fundamentales para garantizar una tracción eficaz en cualquier tipo de calzada. Además, controlar su estado y mantenimiento es muy importante para conseguir la máxima adherencia al suelo y evitar accidentes.

5.5.2. SUSPENSIÓN

Las barras estabilizadoras que unen las ruedas del vehículo son uno de los sistemas de seguridad activa más importantes. Sirven para estabilizar nuestro coche en las curvas y evitar irregularidades del terreno.

5.5.3. FRENOS

El sistema de frenado es fundamental para la seguridad de conductores y acompañantes. Se trata de un sistema de seguridad activa capaz de frenar en caso de que algo falle y uno de los mejores es el famoso sistema antibloqueo ABS.

Gracias a este sistema podremos reducir la distancia de frenado y tendremos la posibilidad de modificar la dirección del vehículo en el

caso de que tengamos que evitar un obstáculo a una velocidad elevada.

5.5.4. CONTROL DE ESTABILIDAD (ESP)

El sistema de control de estabilidad de un vehículo se denomina comúnmente como sistema antivuelco. Y es que este tipo de seguridad activa se ocupa del vehículo en el caso de que el conductor pierda el control absoluto del mismo.

Existen sensores predispuestos en llantas, volante y acelerador que, gracias a un procesador, son capaces de accionar los elementos necesarios en caso de riesgo. Por ejemplo, frenar una o más ruedas.

5.5.5. DIRECCIÓN

El sistema de dirección nos da la capacidad de decidir qué maniobras realizar y cuándo realizarlas. Sin embargo, debemos tener en cuenta que a velocidades altas este sistema se endurece para evitar accidentes y eso es precisamente lo que lo convierte en un sistema de seguridad activa eficaz.

5.5.6. ILUMINACIÓN

Resulta fundamental llevar un control exhaustivo de las luces del vehículo para garantizar que todas funcionen correctamente. Es importante que el resto de conductores perciban nuestra presencia y no hay otra forma de hacerlo que, mediante las luces, un elemento capaz de evitar accidentes nocturnos y que también consideramos integrado en los sistemas de seguridad activa más importantes.

Como verás, la informática combinada eficazmente con elementos del vehículo que forman parte de la seguridad activa del mismo puede reducir el riesgo de accidente de cualquier conductor.

5.6. CAMIONES VOLVO

Volvo es una empresa fabricante de vehículos industriales, incluyendo camiones, autobuses y equipamiento de construcción. Fue fundada en 1927 con sede en Gotemburgo, Suecia, por el ingeniero Gustav Larson y el economista Assar Gabrielsson.

Fabricó autos hasta 1999, cuando vendió su filial Volvo Car Corporation a Ford Motor Company. Esta empresa, a su vez, la vendió en 2010 a Geely Automobile, de China.

En latín, Volvo significa "yo ruedo". El emblema de la marca, el círculo y la flecha, es el símbolo del acero de los antiguos alquimistas y no representa el símbolo de lo masculino como muchos creen erróneamente.

En los camiones destacan los actuales Volvo FH, FM, FMX, FL, FE, FH16 dejando relegados a los antiguos FH12, FM12, FL6, FL7, FL10, FM, TD6, TD10. En autocares y autobuses Volvo B12B, B9R, y B7R dejando relegados los B12, y B10. En la actualidad el motor del Volvo FH es de 13 litros, con potencias de 420 a 540 CV.

El camión Iron Knight de Volvo ha batido el récord de velocidad en las categorías de 500 y 1.000 metros con salida de parado. Este vehículo que bate récords tiene un diseño hecho totalmente a medida, a excepción del motor y la caja de cambios I-Shift de doble embrague.

5.7. PUNTOS CIEGOS DEL CAMIÓN

La conducción de vehículos grandes y pesados es distinta a la conducción de vehículos pequeños, particulares o familiares. En los vehículos pesados, ya sean de transporte de mercancía o personas, cambian la maniobrabilidad y la visibilidad.

Aunque los transportistas estamos capacitados en la conducción de grandes y pesados vehículos de carga y transporte, muchos desconocemos los puntos ciegos del bus y el camión, y este es uno de los puntos clave para evitar accidentes. La seguridad en las vías es responsabilidad de todos, por eso debemos prestar mayor atención en los puntos ciegos. También llamados ángulos muertos, los puntos ciegos que existen en todos los vehículos son los que, desde la posición del conductor, impiden o limitan el campo visual a tal punto que crean zonas de alto riesgo de accidentalidad. En los vehículos grandes y pesados los puntos ciegos son más extensos.

Conozcamos los puntos ciegos del camión:

Figura 3. Puntos ciegos camión de carga fuente: https://juanruedaconinternational.com

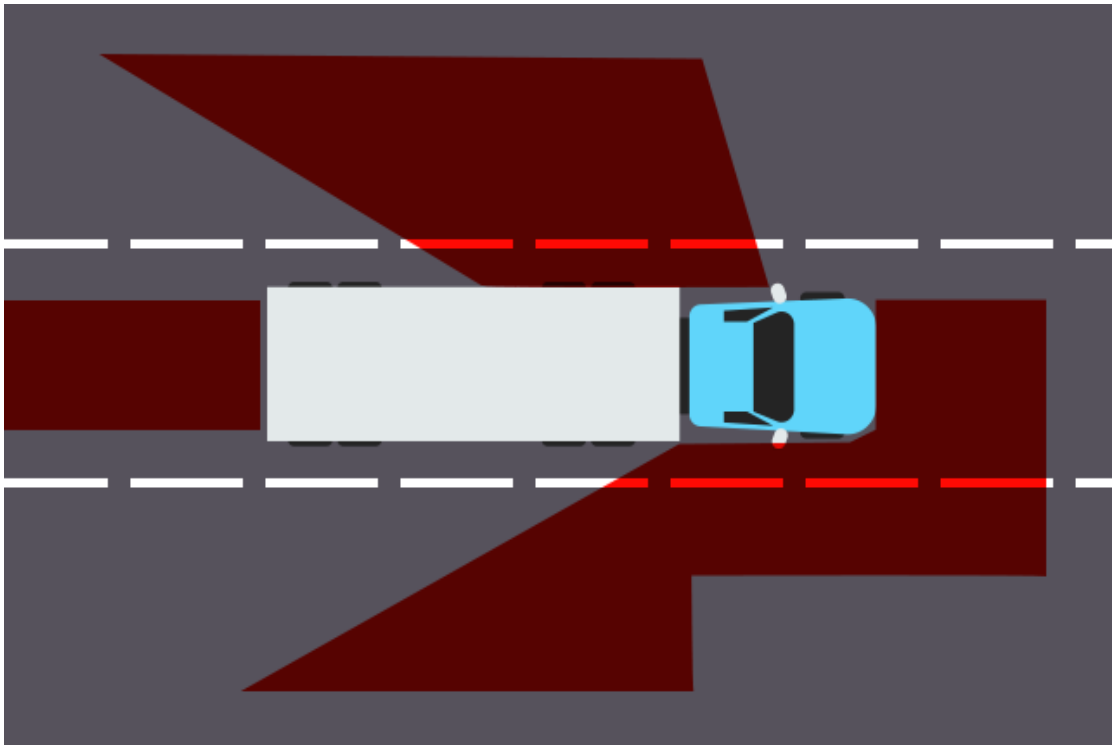


Figura 4. Puntos ciegos de camiones comerciales de carga fuente: https://juanruedaconinternational.com

En el frente: cuanto más largo sea el cofre, mayor es el punto ciego. Un auto pequeño podría caber en este lugar sin ser detectado.

En los laterales: la visibilidad a los lados está limitada por puntos ciegos, el conductor solo puede confiar en lo que observa a través de los espejos retrovisores.

En la parte trasera: el punto ciego trasero es muy largo. Es importante prestar atención cuando un camión está retrocediendo.

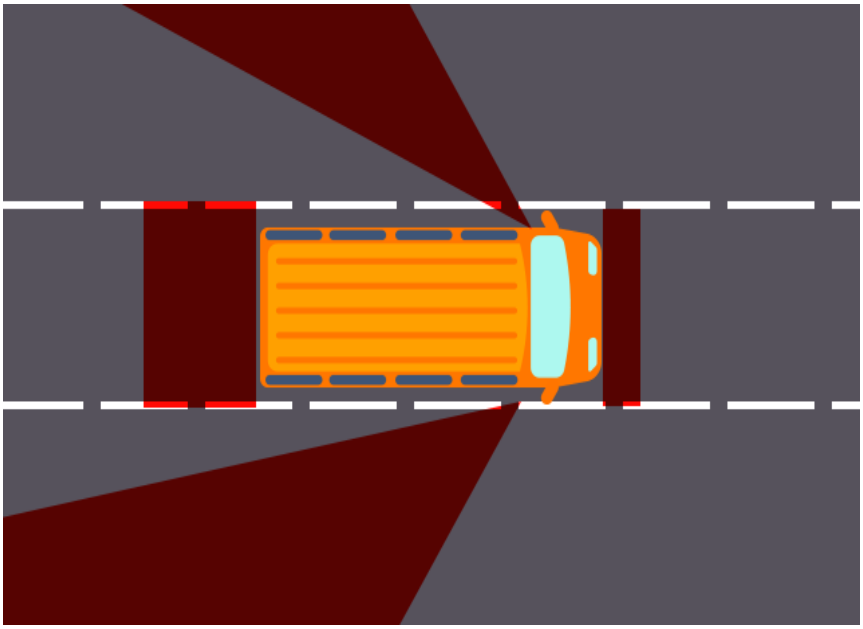


Figura 5. Puntos ciegos de camiones de transporte de pasajeros en la ciudad fuente: https://juanruedaconinternational.com

En el frente: el punto ciego delantero es pequeño porque los autobuses urbanos tienen frentes planos. Sin embargo, bastidores de bicicletas pueden limitar la visibilidad.

En los laterales: debido al gran tamaño del autobús, sus puntos ciegos laterales cubren un área importante incluso con los retrovisores correctamente ajustados. Los puntos ciegos laterales son aún más grandes para los autobuses articulados.

En la parte trasera: aunque la ventana trasera acorta el punto ciego trasero, es importante prestar atención cuando el autobús está retrocediendo.

Delante, en el lado izquierdo: cuanto más ancho es el pilar, más grande es el punto ciego.

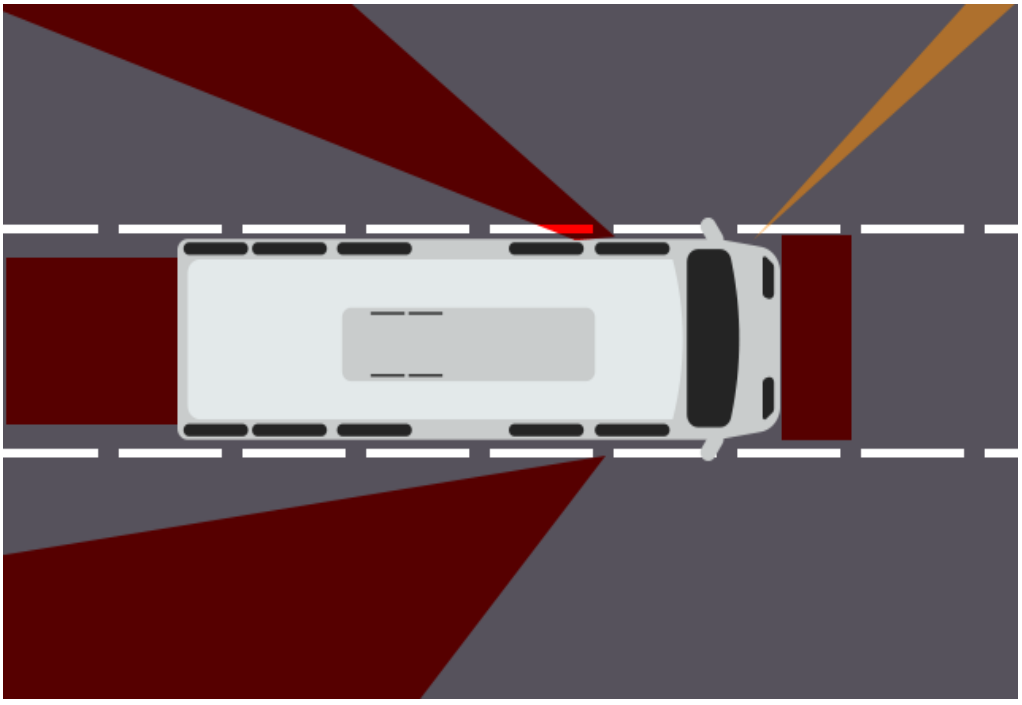


Figura 6. Puntos ciegos de camiones de transporte de pasajeros foráneos fuente: https://juanruedaconinternational.com

En el frente: el punto ciego delantero es similar al de un autobús de la ciudad debido a su frente plano. Sin embargo, el compartimiento del conductor es más alto, lo que hace que sus puntos ciegos sean más grandes.

En los laterales: debido al tamaño y altura significativos del autocar, sus puntos ciegos laterales cubren un área grande.

En la parte trasera: el punto ciego trasero es muy largo porque los autocares no tienen ventanas traseras. Es importante prestar mucha atención cuando un autocar está retrocediendo.

5.8. QUE ES UN ARDUINO

El proyecto Arduino, así como las principales características que lo definen. Se trata de uno de los tipos de las placas más populares del

mundo maker, pero que a diferencia de la Raspberry Pi, no cuenta con un único modelo, sino que ofrece unas bases de hardware abierto para que otros fabricantes puedan crear sus propias placas.

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso.

Para poder entender este concepto, primero vas a tener que entender los conceptos de hardware libre y el software libre. El hardware libre son los dispositivos cuyas especificaciones y diagramas son de acceso público, de manera que cualquiera puede replicarlos. Esto quiere decir que Arduino ofrece las bases para que cualquier otra persona o empresa pueda crear sus propias placas, pudiendo ser diferentes entre ellas, pero igualmente funcionales al partir de la misma base.

El software libre son los programas informáticos cuyo código es accesible por cualquiera para que quien quiera pueda utilizarlo y modificarlo. Arduino ofrece la plataforma Arduino IDE (Entorno de Desarrollo Integrado), que es un entorno de programación con el que cualquiera puede crear aplicaciones para las placas Arduino, de manera que se les puede dar todo tipo de utilidades.

El proyecto nació en 2003, cuando varios estudiantes del Instituto de Diseño Interactivo de Ivrea, Italia, con el fin de facilitar el acceso y uso de la electrónica y programación. Lo hicieron para que los estudiantes de electrónica tuviesen una alternativa más económica a las populares BASIC Stamp, unas placas que por aquel entonces valían más de cien dólares, y que no todos podían acceder a ellas.

El resultado fue Arduino, una placa con todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador, y que puede ser programada tanto en Windows como macOS y GNU/Linux. Un proyecto que promueve la filosofía 'learning by doing', que viene a querer decir que la mejor manera de aprender es practicando.

Arduino es una placa hardware libre que incorpora un microcontrolador reprogramable y una serie de pines-hembra (los cuales están unidos internamente a las patillas de E/S del microcontrolador) que permiten conectar allí de forma muy sencilla y cómoda diferentes sensores y actuadores.

Cuando hablamos de "placa hardware" nos estamos refiriendo en concreto a una PCB (del inglés "printed circuit board", o sea, placa de circuito impreso). Las PCBs son superficies fabricadas de un material no conductor (normalmente resinas de fibra de vidrio reforzada, cerámica o plástico) sobre las cuales aparecen laminadas ("pegadas") pistas de material conductor (normalmente cobre). Las PCBs se utilizan para conectar eléctricamente, a través de los caminos conductores, diferentes componentes electrónicos soldados a ella. Una PCB es la forma más compacta y estable de construir un circuito electrónico (en contraposición a una breadboard, perfboard o similar) pero, al contrario que estas, una vez fabricada, su diseño es bastante difícil de modificar. Así pues, la placa Arduino no es más que una PCB que implementa un determinado diseño de circuitería interna.

No obstante, cuando hablamos de "placa Arduino", deberíamos especificar el modelo concreto, ya que existen varias placas Arduino oficiales, cada una con diferentes características (como el tamaño físico, el número de pines-hembra ofrecidos, el modelo de microcontrolador incorporado –y como consecuencia, entre otras cosas, la cantidad de

memoria utilizable–, etc.). Conviene conocer estas características para identificar qué placa Arduino es la que nos convendrá más en cada proyecto.

De todas formas, aunque puedan ser modelos específicos diferentes (tal como acabamos de comentar), los microcontroladores incorporados en las diferentes placas Arduino pertenecen todos a la misma "familia tecnológica", por lo que su funcionamiento en realidad es bastante parecido entre sí. En concreto, todos los microcontroladores son de tipo AVR, una arquitectura de microcontroladores desarrollada y fabricada por la marca Atmel (http://www.atmel.com). Es por eso que, en este libro seguiremos nombrando "placa Arduino" a cualquiera de ellas mientras no sea imprescindible hacer algún tipo de distinción.

El diseño hardware de la placa Arduino está inspirado originalmente en el de otra placa de hardware libre preexistente, la placa Wiring (http://www.wiring.co).

Esta placa surgió en 2003 como proyecto personal de Hernando Barragán, estudiante por aquel entonces del Instituto de Diseño de Ivrea (lugar donde surgió en 2005 precisamente la placa Arduino).

Un software (más en concreto, un "entorno de desarrollo") gratis, libre y multiplataforma (ya que funciona en Linux, MacOS y Windows) que debemos instalar en nuestro ordenador y que nos permite escribir, verificar y guardar ("cargar") en la memoria del microcontrolador de la placa Arduino el conjunto de instrucciones que deseamos que este empiece a ejecutar. Es decir: nos permite programarlo. La manera estándar de conectar nuestro computador con la placa Arduino para poder enviarle y grabarle dichas instrucciones es mediante un simple cable USB, gracias a que la mayoría de placas Arduino incorporan un conector de este tipo.

Los proyectos Arduino pueden ser autónomos o no. En el primer caso, una vez programado su microcontrolador, la placa no necesita estar conectada a ningún computador y puede funcionar autónomamente si dispone de alguna fuente de alimentación. En el segundo caso, la placa debe estar conectada de alguna forma permanente (por cable USB, por cable de red Ethernet, etc.) a un computador ejecutando algún software específico que permita la comunicación entre este y la placa y el intercambio de datos entre ambos dispositivos. Este software específico lo deberemos programar generalmente nosotros mismos mediante algún lenguaje de programación estándar como Python, C, Java, Php, etc., y será independiente completamente del entorno de desarrollo Arduino, el cual no se necesitará más, una vez que la placa ya haya sido programada y esté en funcionamiento.

Un lenguaje de programación libre. Por "lenguaje de programación" se entiende cualquier idioma artificial diseñado para expresar instrucciones (siguiendo unas determinadas reglas sintácticas) que pueden ser llevadas a cabo por máquinas. Concretamente dentro del lenguaje Arduino, encontramos elementos parecidos a muchos otros lenguajes de programación existentes (como los bloques condicionales, los bloques repetitivos, las variables, etc.), así como también diferentes comandos – asimismo llamados "órdenes" o "funciones" – que nos permiten especificar de una forma coherente y sin errores las instrucciones exactas que queremos programar en el microcontrolador de la placa. Estos comandos los escribimos mediante el entorno de desarrollo Arduino.

Tanto el entorno de desarrollo como el lenguaje de programación Arduino están inspirado en otro entorno y lenguaje libre preexistente: Processing (http://www.processing.org), desarrollado inicialmente por Ben Fry y Casey Reas. Que el software Arduino se parezca tanto a Processing no es

casualidad, ya que este está especializado en facilitar la generación de imágenes en tiempo real, de animaciones y de interacciones visuales, por lo que muchos profesores del Instituto de Diseño de Ivrea lo utilizaban en sus clases. Como fue en ese centro donde precisamente se inventó Arduino es natural que ambos entornos y lenguajes guarden bastante similitud. No obstante, hay que aclarar que el lenguaje Processing está construido internamente con código escrito en lenguaje Java, mientras que el lenguaje Arduino se basa internamente en código C/C++.

Con Arduino se pueden realizar multitud de proyectos de rango muy variado: desde robótica hasta domótica, pasando por monitorización de sensores ambientales, sistemas de navegación, telemática, etc. Realmente, las posibilidades de esta plataforma para el desarrollo de productos electrónicos son prácticamente infinitas y tan solo están limitadas por nuestra imaginación. (Torrente 2013).

5.9. CÓMO FUNCIONA ARDUINO

El Arduino es una placa basada en un microcontrolador ATMEL. Los microcontroladores son circuitos integrados en los que se pueden grabar instrucciones, las cuales las escribes con el lenguaje de programación que puedes utilizar en el entorno Arduino IDE. Estas instrucciones permiten crear programas que interactúan con los circuitos de la placa.

El microcontrolador de Arduino posee lo que se llama una interfaz de entrada, que es una conexión en la que podemos conectar en la placa diferentes tipos de periféricos. La información de estos periféricos que conectes se trasladará al microcontrolador, el cual se encargará de procesar los datos que le lleguen a través de ellos.

El tipo de periféricos que puedas utilizar para enviar datos al microcontrolador depende en gran medida de qué uso le estés pensando

dar. Pueden ser cámaras para obtener imágenes, teclados para introducir datos, o diferentes tipos de sensores.

También cuenta con una interfaz de salida, que es la que se encarga de llevar la información que se ha procesado en el Arduino a otros periféricos. Estos periféricos pueden ser pantallas o altavoces en los que reproducir los datos procesados, pero también pueden ser otras placas o controladores.

5.10. FORMAS ARDUINO

Arduino es un proyecto y no un modelo concreto de placa, lo que quiere decir que compartiendo su diseño básico te puedes encontrar con diferentes tipos de placas. Las hay de varias formas, tamaños y colores para a las necesidades del proyecto en el que estés trabajando, las hay sencillas o con características mejoradas, Arduinos orientados al Internet de las Cosas o la impresión 3D y, por supuesto, dependiendo de estas características te encontrarás con todo tipo de precios.

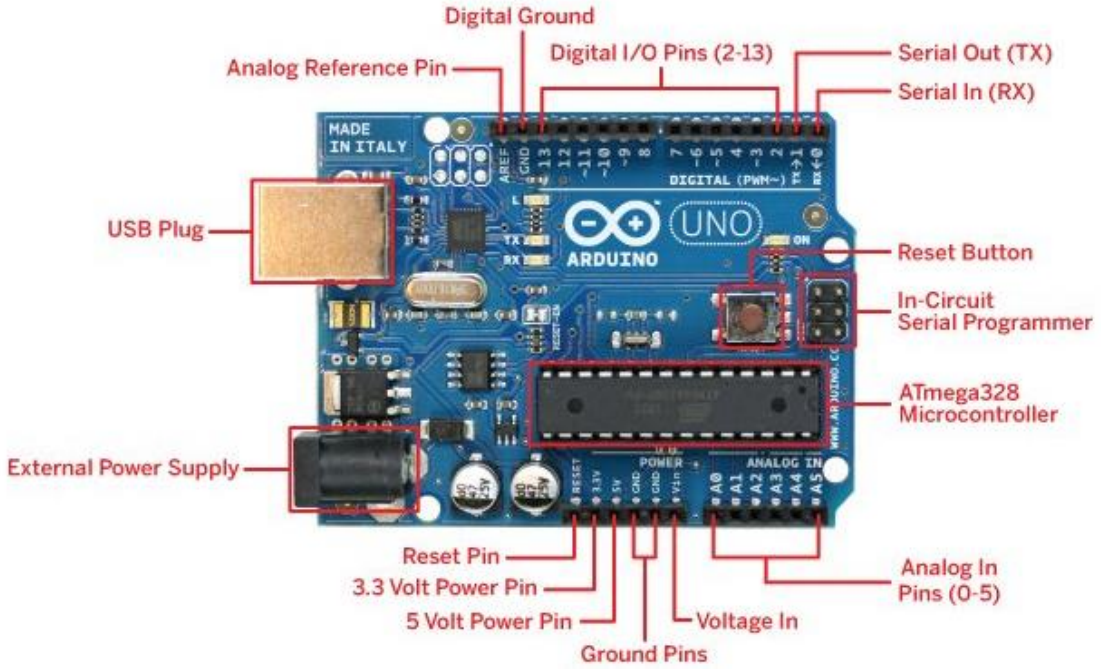


Figura 7. Arduino UNO fuente: https://aprendiendoarduino.wordpress.com

5.11. SENSOR DE PROXIMIDAD

Hemos visto, en los documentales, que los murciélagos son capaces de volar en completa oscuridad y, sin embargo, evadir obstáculos o atrapar insectos en vuelo. Sabemos que lo hacen, pero rara vez pensamos como.

Tenemos una vaga idea de que se llama ecolocalización y que más o menos tiene que ver con unos sonidos agudos que emiten y que después recogen con esas enormes orejas que Dios les ha dado, pero rara vez nos planteamos cómo es esto posible.

Delfines y ballenas utilizan un sistema similar para atrapar a sus presas, y hasta hemos visto que, en cualquier película de submarinos, en el momento álgido el capitán ordena emitir un pulso único de sonar para localizar al enemigo.

El concepto básico, es siempre el mismo, sabiendo a qué velocidad viaja el sonido, si emitimos un pulso sónico corto y escuchamos cuanto tiempo tarda en regresar el eco podemos calcular la distancia a la que se encuentra el objeto en el que ha rebotado la señal.

El radar funciona de modo similar, aunque usando ondas de radio frecuencia muy cortas y con una problemática propia descomunal. Un pulso de radiofrecuencia se emite desde la antena y se recoge el eco que vuelve a la velocidad de la luz.

Lo que haremos en este trabajo, es utilizar un sensor de distancia sencillo HC-SR04 (y muy parecido a los sensores de aparcamiento de los coches modernos), que nos permite enviar estos pulsos ultrasónicos y escuchar el eco de retorno. Midiendo este tiempo, podemos calcular la distancia hasta el obstáculo.

El oído humano no percibe sonidos por encima de 20kHz. Por eso, a las ondas de mayor frecuencia las llamamos ultrasonidos (más allá del sonido). Los sensores de ultrasonidos funcionan sobre los 40 kHz.

No son perfectos, les influye la temperatura ambiente, la humedad y los materiales en los que reflejan, lo que genera una cierta incertidumbre. Pero a cambio son baratos y efectivos hasta un poco más de 3 metros en condiciones normales si la precisión no es un problema determinante.

Figura 8. Sensor de proximidad fuente: https://www.prometec.net

6. MARCO PRÁCTICO

6.1. DETERMINACIÓN DE LOS PUNTOS CIEGOS

Primeramente, determinamos los puntos ciegos del vehículo, en este caso Volvo FH 13, modelo 2010.

Figura 9. Punto ciego del lado del pasajero. Fuente: Propia

Figura 10. Punto ciego de la parte frontal. Fuente: Propia

Figura 11. Punto ciego de la parte trasera. Fuente: Propia

Figura 12. Punto ciego del lado del conductor. Fuente: Propia

De acuerdo a la revisión bibliográfica los vehículos presentan los puntos ciegos mostrados en las figuras.

6.2. DISEÑAR UN SISTEMA DE RADAR DE PROXIMIDAD

En base a un Arduino y el sensor HC-SR04, diseñamos un circuito para la seguridad activa, que pueda ser implementado con el objeto de reducir los accidentes debido a los puntos ciegos del camión u otro vehículo de grandes dimensiones.

El circuito que implementaremos será el siguiente:

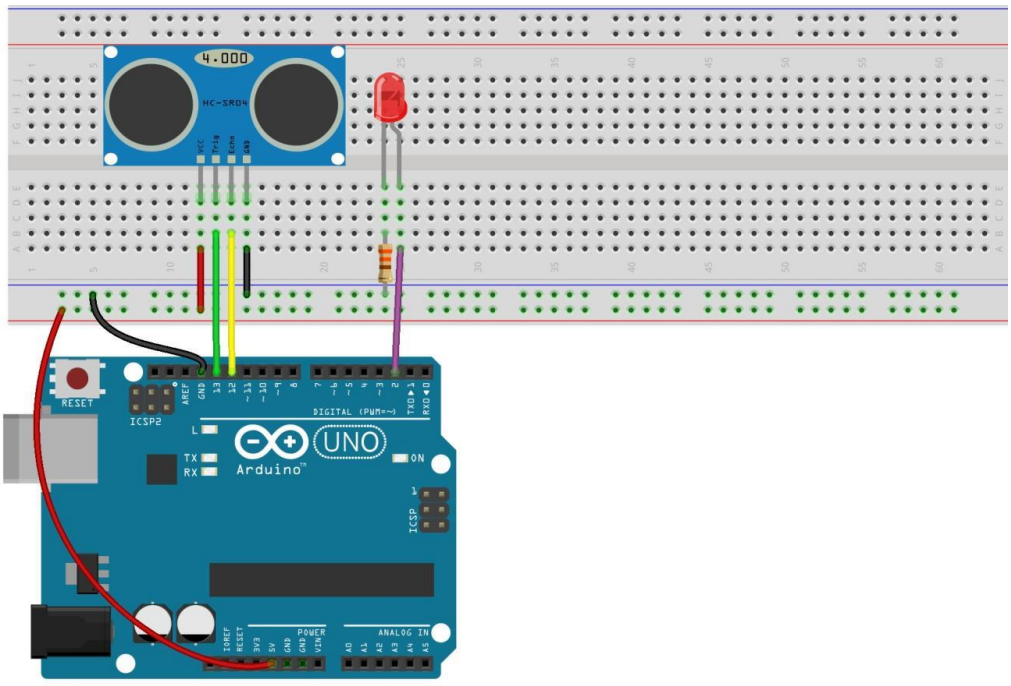


Figura 13. Circuito a implementar. Fuente: Propia

En la figura 13 se muestra el circuito implementado en un simulador y en la figura 14 se muestra el circuito en la versión real del circuito.

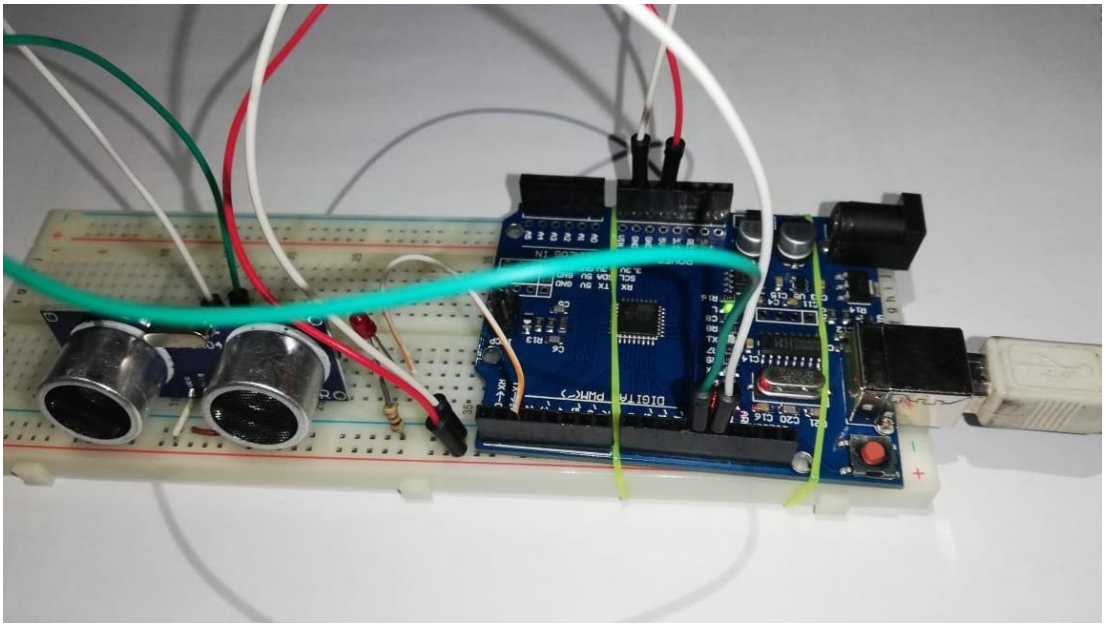


Figura 14. Circuito implementado. Fuente: Propia

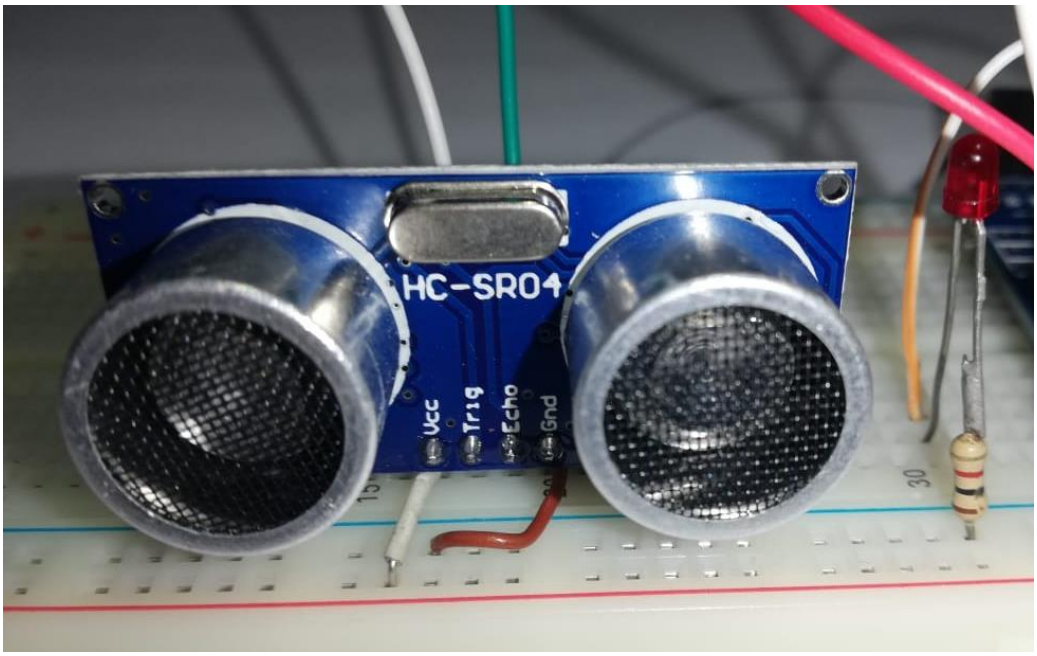


Figura 15. Sensor de proximidad HCSR04. Fuente: Propia

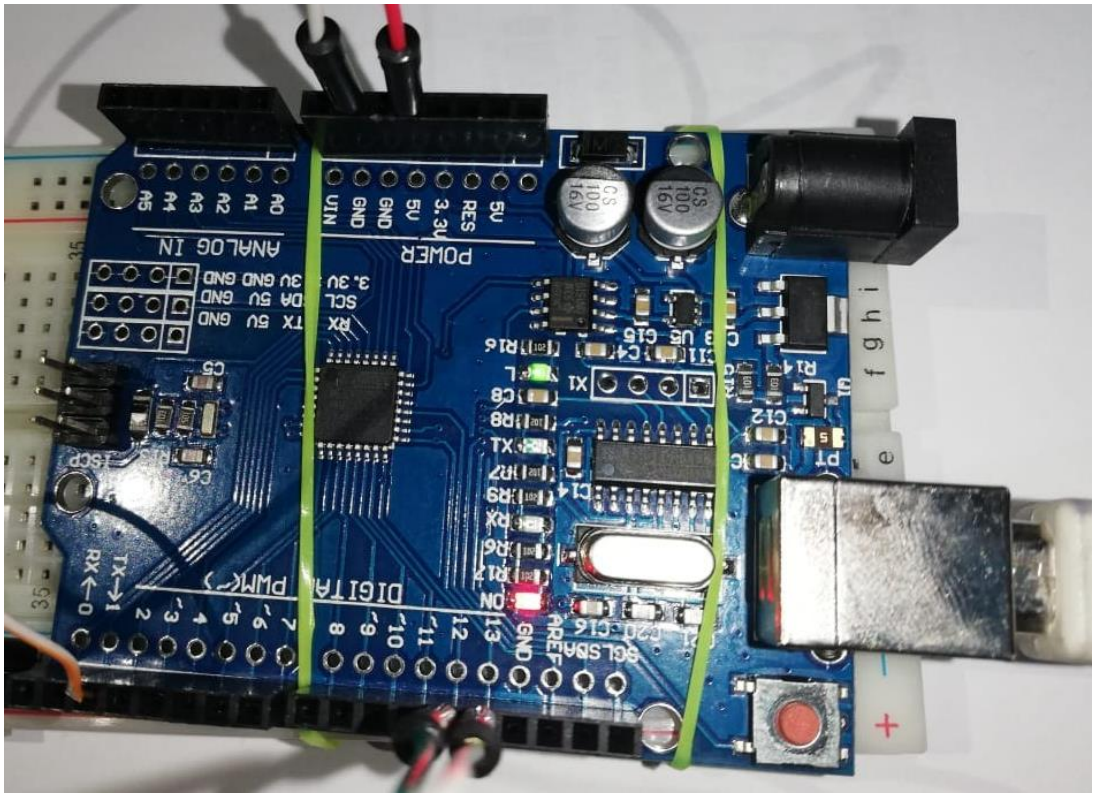


Figura 16. Placa Arduino UNO. Fuente: Propia

El circuito implementado es solo para un punto ciego, aunque el código fuente, se desarrolló para los cuatro puntos ciegos.

Los puntos ciegos serán controlados por focos pilotos instalados en el tablero, que en realidad son leds.

Figura 17. Luz alarma y sensor a instalar en el vehículo. Fuente: Propia

6.3. DESARROLLO DEL CÓDIGO DEL PROGRAMA

```
const int EchoPin1 = 0;

const int TriggerPin1 = 1;

const int EchoPin2 = 2;

const int TriggerPin2 = 3;

const int EchoPin3 = 4;

const int TriggerPin3 = 5;


const int EchoPin4 = 6;

const int TriggerPin4 = 7;
```

```
const int LedPinVerde = 13;

const int LedPinAzul = 12;

const int LedPinAmarillo = 11;

const int LedPinRojo = 10;

float distancia1;

long tiempo1;

float distancia2;

long tiempo2;

float distancia3;

long tiempo3;

float distancia4;

long tiempo4;

 void setup() {

   Serial.begin(9600);

    pinMode(LedPinVerde, OUTPUT);

    pinMode(LedPinAzul, OUTPUT);

    pinMode(LedPinAmarillo, OUTPUT);

    pinMode(LedPinRojo, OUTPUT);

    pinMode(TriggerPin1, OUTPUT);
```

```
  pinMode(EchoPin1, INPUT);
  pinMode(TriggerPin2, OUTPUT);
  pinMode(EchoPin2, INPUT);
  pinMode(TriggerPin3, OUTPUT);
  pinMode(EchoPin3, INPUT);
  pinMode(TriggerPin4, OUTPUT);
  pinMode(EchoPin4, INPUT);
}
 void loop() {
  digitalWrite(TriggerPin1, HIGH);  //se envía un pulso para activar el sensor
  digitalWrite(TriggerPin2, HIGH);  //se envía un pulso para activar el sensor
  digitalWrite(TriggerPin3, HIGH);  //se envía un pulso para activar el sensor
  digitalWrite(TriggerPin4, HIGH);  //se envía un pulso para activar el sensor
   delayMicroseconds(10);
  digitalWrite(TriggerPin1, LOW);
  digitalWrite(TriggerPin2, LOW);
  digitalWrite(TriggerPin3, LOW);
  digitalWrite(TriggerPin4, LOW);
  tiempo1 = (pulseIn(EchoPin1, HIGH)/2);
```

```
 distancia1 = float(tiempo1 * 0.0343);
  Serial.print("Distancia1: "); // imprime la distancia en el Monitor Serie
  Serial.println(distancia1);
 tiempo2 = (pulseIn(EchoPin2, HIGH)/2);
 distancia2 = float(tiempo2 * 0.0343);
  Serial.print("Distancia2: "); // imprime la distancia en el Monitor Serie
  Serial.println(distancia2);
tiempo3 = (pulseIn(EchoPin3, HIGH)/2);
distancia3 = float(tiempo3 * 0.0343);
  Serial.print("Distancia3: "); // imprime la distancia en el Monitor Serie
  Serial.println(distancia3);
 tiempo4 = (pulseIn(EchoPin4, HIGH)/2);
 distancia4 = float(tiempo4 * 0.0343);
  Serial.print("Distancia4: "); // imprime la distancia en el Monitor Serie
  Serial.println(distancia4);

if ((distancia1 <= 200) && (distancia1 >= 20)){
      digitalWrite(LedPinVerde ,HIGH);
      }
```

```
if ((distancia2 <= 200) && (distancia2 >= 20)){
      digitalWrite(LedPinAzul ,HIGH);
      }
if ((distancia3 <= 200) && (distancia3 >= 20)){
      digitalWrite(LedPinAmarillo ,HIGH);
      }
if ((distancia4 <= 200) && (distancia4 >= 20)){
      digitalWrite(LedPinRojo ,HIGH);
      }
delay(1000);
digitalWrite(LedPinVerde ,LOW);
digitalWrite(LedPinAzul ,LOW);
digitalWrite(LedPinVerde ,LOW);
digitalWrite(LedPinVerde ,LOW);
}
```

6.4. IMPLEMENTACIÓN DEL SISTEMA EN EL CAMIÓN VOLVO

En las imágenes posteriores se puede observar la implementación del sistema en un volvo FH en los puntos donde se presentan los puntos ciegos.

SENSOR IMPLEMENTADO EN EL LADO DEL PASAJERO

Figura 18. IMPLEMENTACION DE LA LUZ DE ADVERTENCIA. Fuente: Propia

Figura 19. PRUEBA DEL SENSOR EN EL LADO DEL PASAJERO. Fuente: Propia

Figura 20. INSTALACION DEL SENSOR EN EL LADO DEL PASAJERO. Fuente: Propia

SENSOR IMPLEMENTADO EN EL LADO DEL CONDUCTOR

Figura 21. PRUEBA DEL SENSOR EN EL LADO DEL CONDUCTOR. Fuente: Propia

Figura 22. INSTALACION DEL SENSOR EN EL LADO DEL CONDUCTOR. Fuente: Propia

SENSOR IMPLEMENTADO EN LA PARTE FRONTAL

Figura 23. PRUEBA DEL SENSOR EN LA PARTE FRONTAL. Fuente: Propia

Figura 24. INSTALACION DEL SENSOR EN LA PARTE FRONTAL. Fuente: Propia

Figura 25. INSTALACION DEL SENSOR EN LA PARTE FRONTAL. Fuente: Propia

SENSOR IMPLEMENTADO EN LA PARTE DE ATRAS

Figura 26. PRUEBA DEL SENSOR EN LA PARTE TRASERA. Fuente: Propia

Figura 27. INSTALACION DEL SENSOR EN LA PARTE TRASERA. Fuente: Propia

Figura 28. INSTALACION DEL SENSOR EN LA PARTE TRASERA. Fuente: Propia

7. CONCLUSIONES Y RECOMENDACIONES

7.1. CONCLUSIONES

- Al realizar la verificación de los puntos ciegos se evidencio que existen puntos que el conductor no puede observar con los accesorios que dispone para tal efecto como son los espejos.
- Estos puntos ciegos prácticamente se encuentran debajo de los vidrios retrovisores que están a los costados del vehículo.
- El espacio denominado punto ciego es mayor cuanto más alto es el vehículo.
- En el diseño solo se consideró en camiones que no son largos, porque en vehículos largos la implementación en la parte trasera es complicada, además no se consideró en esos casos como afecta la distorsión del medio.
- Al implementar también nos dimos cuenta que para cada vehículo variara el programa ya que se debe considerar en el programa la distancia del sensor al suelo.
- La respuesta del sensor trasero es más lenta que los otros sensores esto debido a la distancia donde está instalado.

7.2. RECOMENDACIONES

- Recomendamos para mejoras del trabajo realizar un circuito auto ajustable para no modificar el programa y externamente auto ajustar.
- Para implementarlo en otro tipo de vehículo debe modificar el programa recalculando la distancia a la que se encuentra el sensor con respecto al suelo

8. BIBLIOGRAFÍA

- De la Fuente, J., (1995). *SEGURIDAD ACTIVA Y PASIVA EN EL VEHICUYLO*, Madrid España: Dossat 2000.
- Gongora E, Acosta JA, Wang DS, Brandebury K, Jordan MH, (2001). *ANALISYS OF MOTOR VEHICLE EJECTION VICTIMS ADMITTED TO LEVEL I TRAUMA CENTER*, 51:854-859
- Torrente O, (2013) *ARDUINO CURSO PRÁCTICO DE FORMACIÓN*, Mexico: Alfaomega Grupo Editor, S.A. de C.V.
- Jagadish A, (2015), *ARDUINO BY EXAMPLE*, India: Packt Publishing Ltd.
- Blum R, (2014) Sams Teach Yourself Arduino™ Programming in 24 Hours, Indianapolis, Indiana, USA: Pearson Education, Inc.
- https://www.pruebaderuta.com/seguridad-activa.php
- https://www.volvotrucks.es/es-es/trucks/volvo-fh16.html
- https://juanruedaconinternational.com/content/cu-les-son-los-puntos-ciegos-del-cami-n
- https://grupomedlegal.com/blog/puntos-ciegos-camiones-comerciales-transporte-pasajeros/
- https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno
- https://aprendiendoarduino.wordpress.com/2016/06/27/arduino-uno-a-fondo-mapa-de-pines-2/
- https://www.prometec.net/sensor-distancia/

9. ANEXOS

- ANEXO 1 CÓDIGO DEL PROGRAMA RECOMENDADO
- ANEXO 2 DATASHEET ARDUINO UNO
- ANEXO 3 DATASHEET HCSR04

ANEXO 1

CÓDIGO DEL PROGRAMA RECOMENDADO

Vamos con el programa, empezamos definiendo algunos valores:

#define trigPin 13

#define echoPin 12

#define led 2

Hasta ahora habíamos visto que podíamos definir una variable como int, por ejemplo, y también como una constante (const int pin). Aquí utilizamos otro método, el #define que es una directiva para el compilador.

Esto solo significa que el compilador (en rigor el pre procesador) cambiará todas las ocurrencias de estos #define en nuestro programa por su valor antes de compilar. Esta es la forma clásica de C de hacer esto y tiene la virtud de que no ocupa memoria definiendo una variable (y con un Arduino UNO, que va muy corto de memoria, esto puede ser crítico en ocasiones).

```
void setup()
  {
     Serial.begin (9600);
     pinMode(trigPin, OUTPUT);
     pinMode(echoPin, INPUT);
     pinMode(led, OUTPUT);
  }
```

Ya estamos más que habituados a la función delay (milisegundos), pero el reloj interno de Arduino uno mide en microsegundos y tenemos otra función parecida delayMicroseconds(µs) que simplemente congela Arduino el número especificado de microsegundos.

Para dar un pulso ultrasónico lo que hacemos es activar el pin Trigger durante unos microsegundos y para ello lo ponemos en HIGH, antes de escuchar el eco:

```
digitalWrite(trigPin, LOW);          // Nos aseguramos de que el trigger está desactivado
delayMicroseconds(2);              // Para estar seguros de que el trigger ya está LOW
digitalWrite(trigPin, HIGH);    // Activamos el pulso de salida
delayMicroseconds(10);             // Esperamos 10µs. El pulso sigue active este tiempo
digitalWrite(trigPin, LOW);     // Cortamos el pulso y a esperar el echo
```

Para escuchar el pulso vamos a usar otra función, pulseIn().

Básicamente lo que hace es escuchar el pin que le pasamos, buscando una señal que pase de LOW a HIGH (si le pasamos HIGH como parámetro) y cuenta el tiempo que tarda en volver a bajar desde que sube.

long duracion, distancia ;

duracion = pulseIn(echoPin, HIGH) ;

Ahora ya sabemos el tiempo que tarda en volver el eco en µs. Como la velocidad del sonido es de 343 metros / segundo, Necesitamos 1/343 = 0,00291 segundos para recorrer un metro.

Para usar una medida más cómoda podemos pasar esto a microsegundos por centímetro:

$$0{,}00291 * \frac{1.000.000\ \mu s}{segundo} * \frac{1\ metro}{100\ cm} = 29{,}1\ \mu s / cm$$

Como nuestro eco mide el tiempo que tarda el pulso en ir y venir la distancia recorrida será la mitad:

$$distancia = \frac{duracion}{29{,}1} * \frac{1}{2}$$

Así que el programa queda parecido a esto:

```
#define trigPin 13
```

```
#define echoPin 12
#define led 2
void setup()
  {  Serial.begin (9600);
     pinMode(trigPin, OUTPUT);
     pinMode(echoPin, INPUT);
     pinMode(led, OUTPUT);
  }
void loop()
  {  long duracion, distancia ;
     digitalWrite(trigPin, LOW);       // Nos aseguramos de que el trigger está desactivado
     delayMicroseconds(2);             // Para asegurarnos de que el trigger esta LOW
     digitalWrite(trigPin, HIGH);      // Activamos el pulso de salida
     delayMicroseconds(10);            // Esperamos 10µs. El pulso sigue active este tiempo
     digitalWrite(trigPin, LOW);       // Cortamos el pulso y a esperar el echo
     duracion = pulseIn(echoPin, HIGH) ;
     distancia = duracion / 2 / 29.1  ;
```

```
    Serial.println(String(distancia) + " cm.") ;
    int Limite = 200 ;              // Medida en vacío del sensor
    if ( distancia < Limite)
        digitalWrite ( led , HIGH) ;
    else
        digitalWrite( led , LOW) ;
    delay (500) ;                   // Para limitar el número de mediciones
}
```

Para convertir esto en un detector de movimiento hemos creado una variable un poco menor de la medida que el sensor recibe en vacío (en mi caso unos 200 cm). Si la distancia medida cae por debajo este valor es que algo se ha interpuesto y por tanto encendemos una alarma, en nuestro caso un humilde LED.

ANEXO 2

ANEXO 3

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

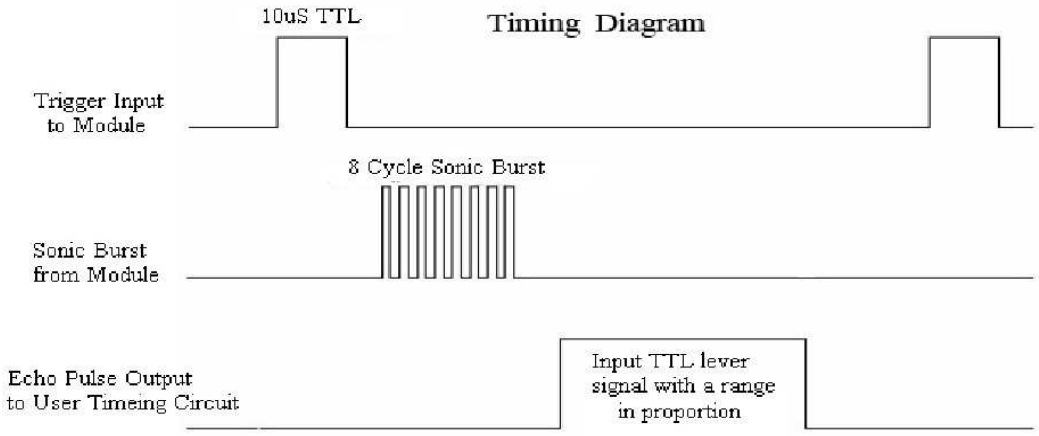
Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Vcc Trig Echo GND

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.

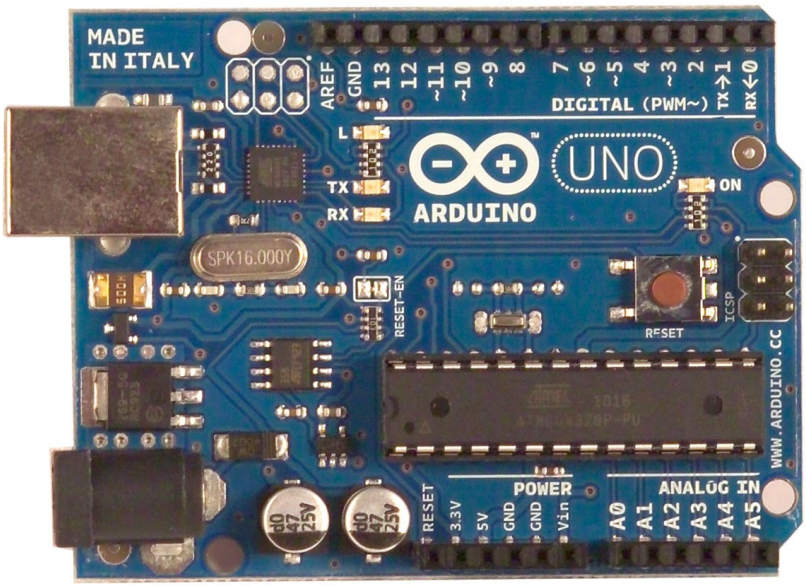


Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

www.Elecfreaks.com

Arduino UNO



Product Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduno, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Index

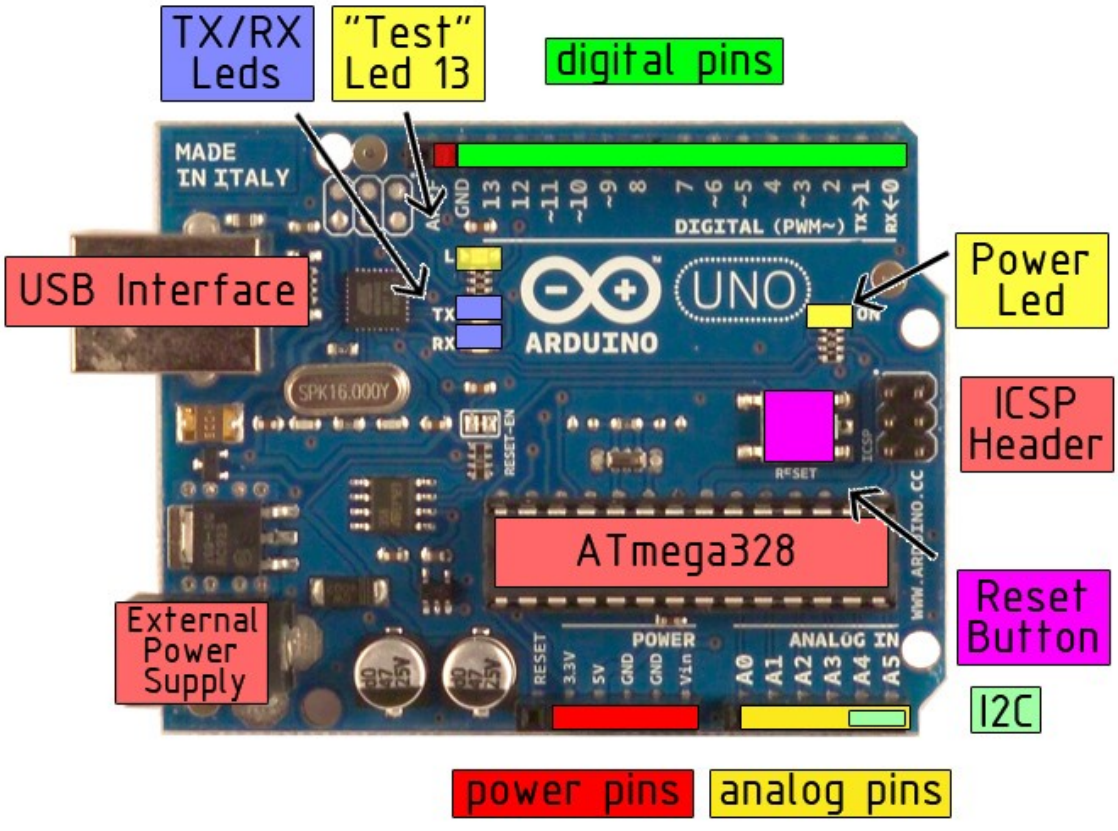
Technical Specification

EAGLE files: arduino-duemilanove-uno-design.zip Schematic: arduino-uno-schematic.pdf

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

the board



Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. TThese pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.

- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **I²C: 4 (SDA) and 5 (SCL).** Support I²C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and Atmega328 ports.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, an *.inf file is required..

The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. To use the SPI communication, please see the ATmega328 datasheet.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno w/ ATmega328" from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega8U2 firmware source code is available . The ATmega8U2 is loaded with a DFU bootloader, which can be activated by connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader).

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

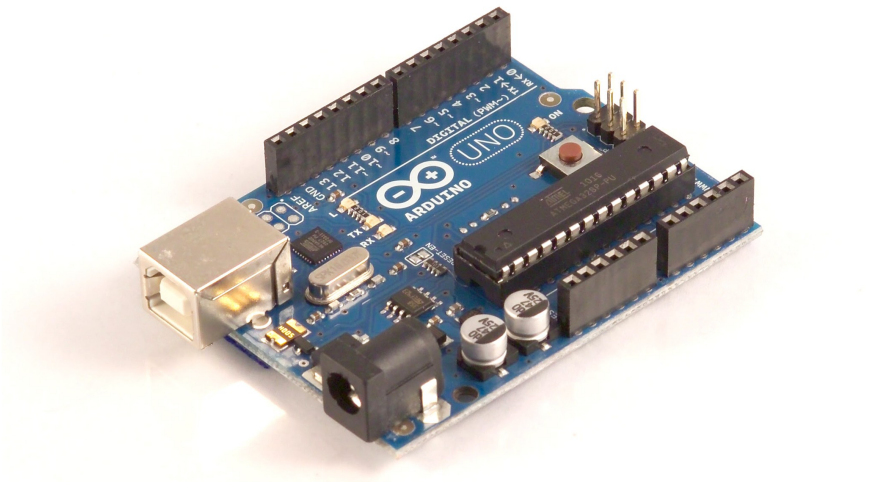
The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.



How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install Windows Install Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

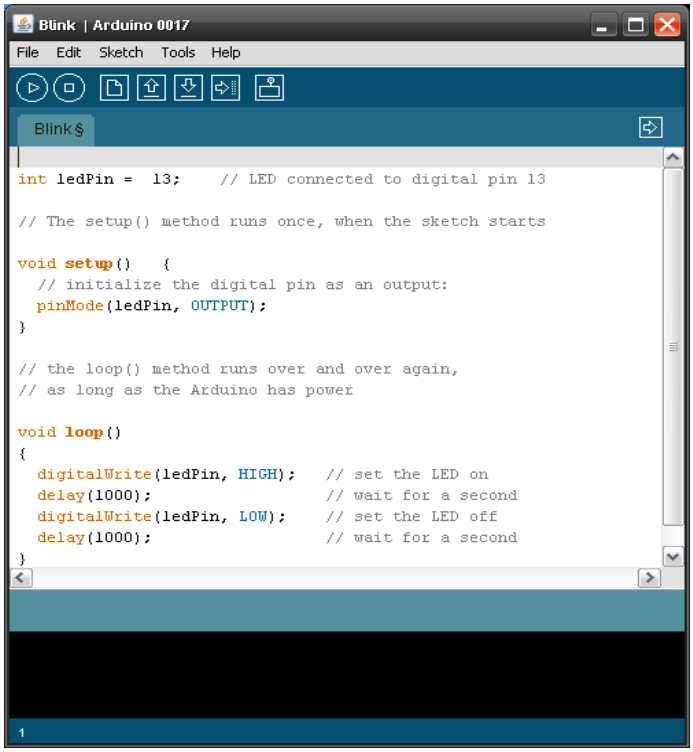
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

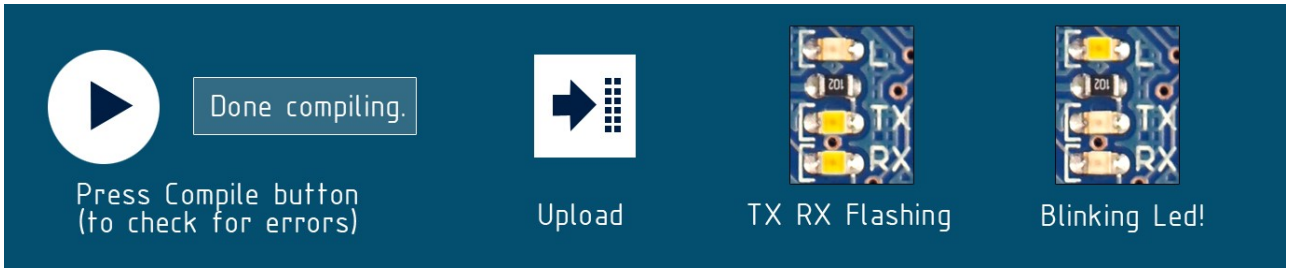
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

Once you have your skecth you'll see something very close to the screenshot on the right.

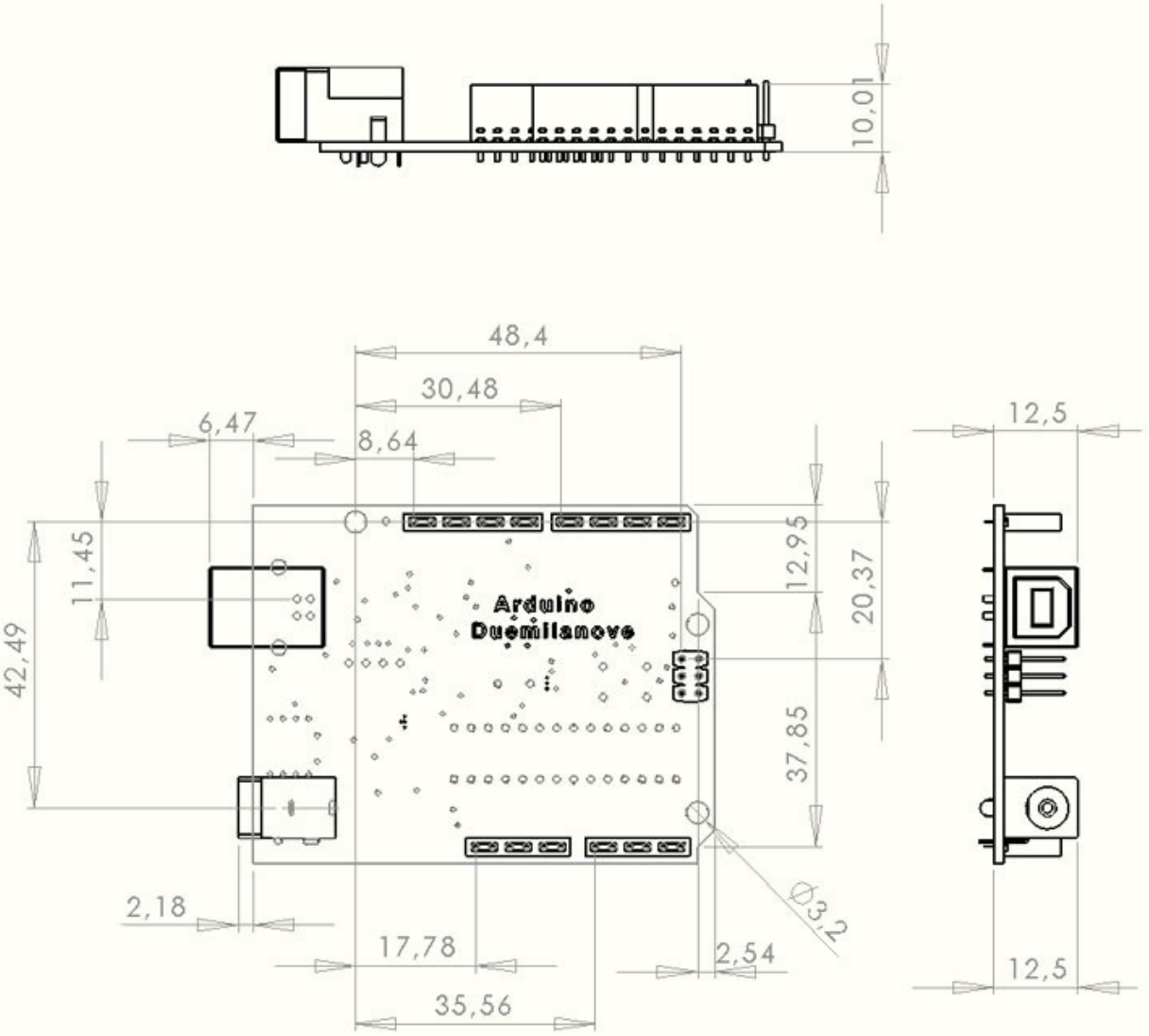
In **Tools>Board** select

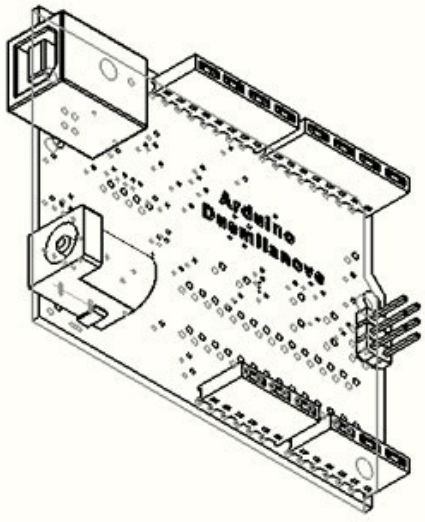
Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.





Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.