Trabajo de Fin de Grado

Sensor láser de medida de distancias

Laser distance measurement sensor

Carlos Barreda Falciano

La Laguna, 4 de marzo de 2015

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78.698.554-Y, Profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Sensor láser de medida de distancias."

ha sido realizada bajo su dirección por D. **Carlos Barreda Falciano**, con N.I.F. 78.635.428-S.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de marzo de 2015.

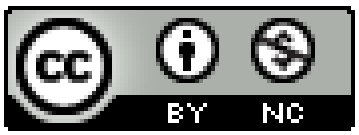
Agradecimientos

En primer lugar quisiera agradecer el desarrollo de este trabajo al tutor Jonay Tomás Toledo Carrillo. Su profesionalidad, apoyo constante y disposición han sido fundamentales para la finalización de este proyecto.

En segundo lugar, me gustaría agradecer a mi familia, en especial a mis padres y hermanas por la oportunidad que me han ofrecido, siendo participes de la culminación de este proyecto y lo que ello conlleva, sin su esfuerzo y cariño nada de esto hubiese sido posible.

Por último, me gustaría también agradecer este trabajo a mis amigos, que han sabido entenderme y expresarme su apoyo en los momentos más difíciles.

Licencia



Resumen

Todo Robot, cuya funcionalidad sea desempeñar alguna tarea autónoma, ha de estar dotado de un sistema sensorial. Gran parte de los sensores utilizados en robótica se basan en la utilización de sistemas empotrados para la realización de medidas. El objetivo de este trabajo final de grado es la construcción de un sensor láser de medida de distancia utilizando el principio de triangulación. Para la fabricación de este tipo de sensores se emplea una fuente de señal luminosa y una célula receptora de dicha señal. En este caso emplearemos un diodo láser y un captador por reflexión del tipo CCD lineal, concretamente el suministrado por la compañía SONY y cuyo modelo es el ILX511. Una vez desarrollado un prototipo funcional del dispositivo, se ha querido darle una funcionalidad concreta. Se ha diseñado el sensor para ser utilizado en la navegación de robots, entendiendo navegación como una metodología que nos permite guiar el vehículo a un destino determinado de forma segura. De este modo, se le otorga al robot la capacidad de interactuar en un entorno no controlado, siendo capaz de reaccionar ante situaciones inesperadas.

Palabras clave: Sensor Láser, Triangulación, Arduino, SONY ILX511, CCD Lineal, Navegación Robótica, ROS

Abstract

Every kind of robot, which usefulness is to carry an autonomous duty out, should have a sensorial system. The greater amount of sensors used in Robotics are based on the usage of embedded systems in order to carry measurements out. The main objective of this project is to construct a laser distance measurements sensor taking advantage of the triangulation principle. In order to produce this kind of sensors, we must need a light signal source and a receptive cell. Now, we are going to use a laser diode and a lineal CCD reflection collector, specifically, the model ILX511, which is the one that SONY provides us. Once the functional prototype device has been developed, we wanted to assign it a concrete functionality. A sensor has been designed in order to be used in the navigation of robots, understanding the navigation as a methodology that allows us to guide the vehicle to a given destination in a successful and secure way. In this way, the robot has the capacity to interact for an uncontrolled environment, being capable to react when unexpected situations appear.

***Keywords**: laser sensor, triangulation, Arduino, SONY ILX511, lineal CCD, robotic navigation, ROS*

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Antecedentes y Estados del Arte

Resulta raro imaginarnos un mundo sin el beneficio que nos aporta la tecnología. Esta condición, es la causante de que cada día le exijamos más a los dispositivos que adquirimos en el mercado, que con el paso del tiempo, están llevando a cabo tareas que hasta hace poco eran impensables.

En los inicios de la robótica las máquinas trabajaban en entornos totalmente controlados, donde cada objeto se encontraba en una posición determinada y predefinida. Con el paso de los años, se ha buscado que los robots dispongan de una capacidad de reacción que les permitan desempeñar tareas de forma autónoma. Con esta nueva filosofía aparecieron los robots basados en comportamientos, que son capaces de reaccionar en un entorno no controlado tomando decisiones únicamente en función de la entrada de sus sensores.

Una de las nuevas funcionalidades incorporada ya en muchos aparatos, es el cálculo de distancias, que ha provocado un incremento en la fabricación de distintos tipos de sensores para este fin. La tecnología empleada en sensores de proximidad suele variar, dependiendo de la precisión y el uso que se requiera, así como del rango de distancias en el que se necesite actuar. En el mundo de la robótica se suele utilizar el ultrasonido y el láser, que presentan una serie de ventajas respecto a sistemas de visión más complejos, como pueden ser: la sencillez de utilización, menor carga computacional, bajo consumo, peso reducido, menor coste, etc. Sin embargo, no todo son ventajas, pues también se ven limitados, por ejemplo, a la sensibilidad de la luz ambiente en el caso de los sensores láser o por el ángulo de la superficie de rebote en el caso de los sensores de ultrasonido.

Actualmente existen dos tipos muy extendidos de sensores láser de medida distancias, que emplean principios diferentes, por tiempo de vuelo y por triangulación.

- **Medición láser por tiempo de vuelo**

El primer método calcula la distancia utilizando el tiempo de vuelo. Esta técnica parte del concepto de que la velocidad de la luz, en un mismo medio, es constante, por lo que puede calcularse el tiempo que tarda en viajar desde una fuente a un objeto reflectante y regresar hasta donde ha sido emitida. Su funcionamiento se basa en calcular el retardo al enviar una señal modulada, midiendo el tiempo entre la emisión y la captura de la señal, se puede estimar el tiempo de vuelo de la onda. Como la velocidad de la onda es conocida, el tiempo de vuelo será proporcional a la distancia a la que se encuentre el objeto de impacto.

- **Medición láser por triangulación**

La otra tecnología, es la empleada en la fabricación de nuestro dispositivo, y se basa en aplicar el Principio de Triangulación. La mayoría de los sensores láser por triangulación presentan el mismo diseño constructivo. Para nuestro proyecto la medición se realiza en una estructura formada por un láser y un dispositivo fotoreceptor del tipo CCD. Esta técnica se llama triangulación, porque tal y como vemos en la Figura 1.1, el emisor (láser), el sensor fotoreceptor (CCD) y el punto del láser (objeto donde incide la luz) forman un triángulo. Dependiendo de la distancia a la que se encuentre el objeto más cercano, donde incide el haz de luz, el punto del láser aparecerá en un lugar (píxel) u otro en el sensor CCD.

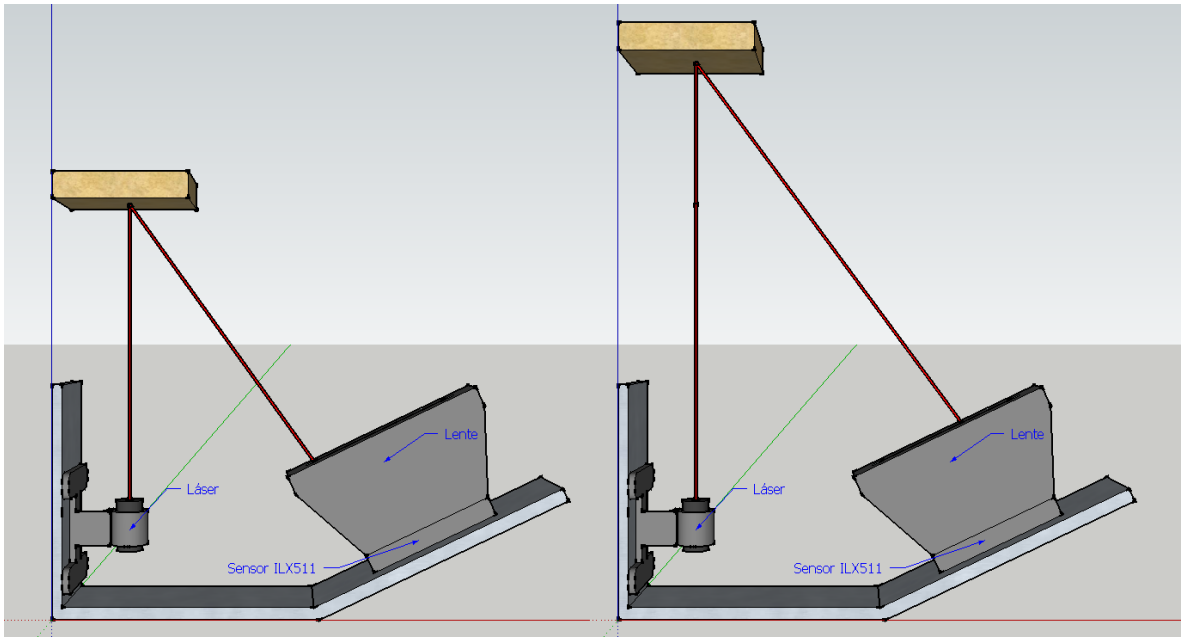


Figura1.1. Principio de Triangulación.

1.2 Motivación y Objetivo

Con la creciente aparición en el mercado de dispositivos que requieren del uso de sensores de proximidad, que les permitan desempeñar una tarea de forma autónoma, es necesario dotar a los equipos de mayor cantidad de sensores y actuadores. Es justo en este punto, donde radica la importancia de abaratar el coste de estos. El principal objetivo y motivación de este TFG no es la invención de un nuevo dispositivo que nos permita calcular distancias utilizando un diodo láser, puesto que ya existen en el mercado, sino la posibilidad de fabricar uno a un precio muy reducido.

En el ámbito personal, la incitación que supone desarrollar un proyecto relacionado con la robótica radica en enfrentarse y superar una faceta multidisciplinar, donde entra en juego diversos campos como la electrónica, la mecánica y la informática. Por lo que desde un principio, la meta particular para la finalización con éxito de este TFG ha sido la construcción de un prototipo funcional.

1.3 Fases del desarrollo

La realización de este proyecto ha seguido un progreso lineal, sin embargo, el trabajo se ha ido documentando de forma paralela a los avances logrados, al igual que para culminar las fases de desarrollo se ha tenido que ir asimilando conceptos e investigando de manera simultánea. En general, podemos diferenciar tres fases de desarrollo:

- **Fabricación del sensor que calcule la distancia a un punto**
  La primera fase de desarrollo se puede catalogar como la construcción de un prototipo funcional que nos permitirá calcular la distancia a un único punto. Al ser la fase inicial, ha estado sujeta a la investigación y estudio de la placa de desarrollo (Arduino UNO), del sensor CCD lineal (SONY ILX511), además del uso de nuevas tecnologías, como puede ser la plataforma de Arduino, y el manejo de muchas herramientas como el osciloscopio y el multímetro.

- **Modificar el sensor para calcular la distancia de un perímetro**
  Con la finalización de la primera fase de desarrollo tenemos la distancia a un punto concreto, el siguiente paso es poder calcular la distancia de un perímetro. Para realizar esta etapa fue necesario investigar el uso y manejo de motores paso a paso, así como sus alternativas y los controladores necesarios (L298N) para facilitar su implantación en un prototipo.

- **Elaboración de un barrido láser 2D**
  Por último, para generar un barrido 2D a tiempo real ha sido necesario introducirse brevemente en el mundo de software especializado en robótica, comunicando nuestro sensor, a través del puerto serie, con la plataforma ROS.

1.4 Estructura de la memoria

Capítulo 1: Es el primer capítulo de la memoria, donde se redacta una introducción al trabajo desarrollado. Se ha descrito la situación actual de los sensores de proximidad en el mundo de la robótica. También los objetivos que nos hemos marcado y las fases de desarrollo para llevarlos a cabo.

Capítulo 2: Se explica el diseño del prototipo implementado, se muestran los esquemas de conexión y los detallan los componentes para la fabricación del sensor láser de medida de distancia.

Capítulo 3: Explicamos funcionamiento lógico del sistema desarrollado a través del código implementado. Se hace una trazabilidad del programa que se carga en la placa de Arduino y se explican las funciones que han sido desarrolladas.

Capítulo 4: Se comentará como se ha calibrado el dispositivo construido, además de la función que se obtuvo que nos indique la distancia al objeto más próximo. También hablaremos de algunas variables estructurales que afectan al calibrado.

Capítulo 5: Explicamos brevemente el software especializado en robótica (ROS) y el trabajo realizado con él para mostrar un barrido láser 2D. Se presentan los resultados obtenidos.

Capítulo 6: Se estima un presupuesto total para llevar a cabo la fabricación del sensor. Básicamente se divide en costes de material y personal.

Capítulo 7: Se presentan las conclusiones, redactando un resumen extendido de los objetivos alcanzados, así como la posibilidad de implantar futuras mejoras en el diseño del prototipo.

Capítulo 8: Es el último capítulo de la memoria. Su contenido es exactamente igual que el capítulo anterior, a diferencia de que está redactado en inglés.

Capítulo 2.
Diseño y Funcionamiento

En el capítulo anterior se ha introducido la idea general de este TFG. Para entrar en materia y para que sirva como precursor a la explicación del funcionamiento del sistema empotrado desarrollado, se procede a hacer una descripción detalla de cada uno de los componentes, así como el modo en el que han sido conectados.

2.1 Conexión del sensor CCD

La conexión del sensor CCD, su lectura e interpretación de datos corresponde a la primera fase de desarrollo. Para ello hemos empleado 5 componentes, una placa Arduino UNO, un sensor CCD lineal, un comparador, un láser y una lente. La conexión de los dispositivos electrónicos es la que se aprecia en la Figura 2.1

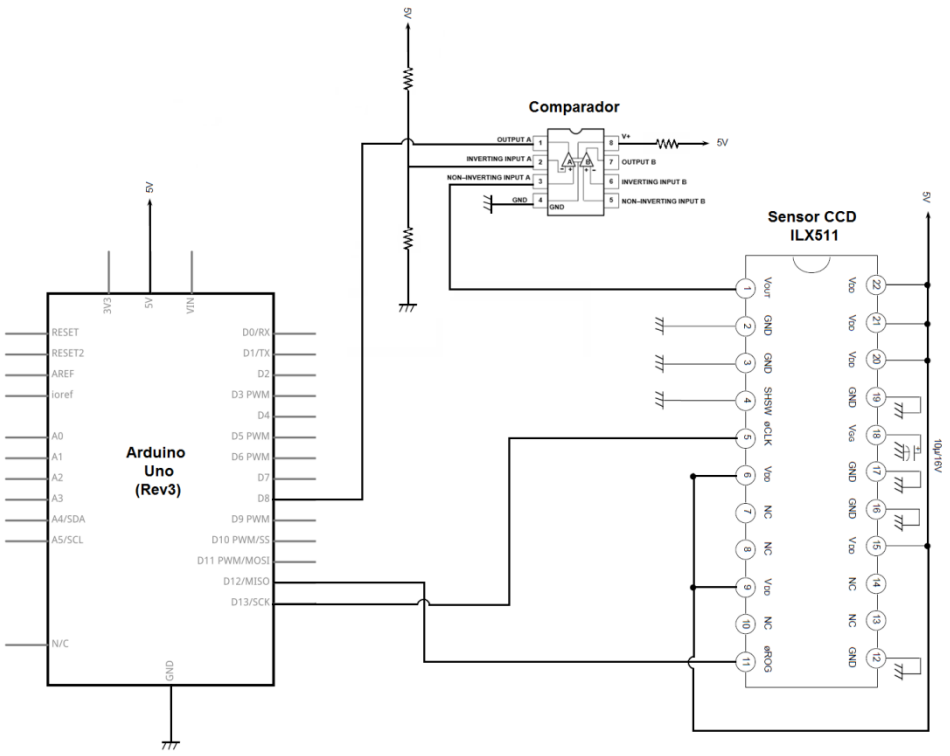


Figura2.1. Esquema de conexión CCD-Arduino.

2.1.1 Arduino UNO

El objetivo es la construcción de un prototipo funcional que combine la rapidez y la facilidad de implementación. Para ello, aunque sabemos que el precio final del dispositivo se verá afectado, recurrimos al uso de placas de desarrollo existentes. Dichas placas de desarrollo son «libres» y sus esquemas son de acceso público, por lo que pueden ser hechas a mano o adquiridas en tiendas especializadas. Este detalle nos permitirá en un futuro adaptar y unificar únicamente los componentes necesarios para nuestro dispositivo.

Se ha elegido a Arduino por ser una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Al combinar cualquiera de sus placas con la conexión a múltiples sensores y actuadores nos permite detectar información del entorno y tomar decisiones en función de lo que ocurre. Dentro de la amplia gama de productos que nos ofrece Arduino, que dependen principalmente del tamaño, número de pines y del microcontrolador que traiga incorporado, hemos seleccionado Arduino UNO como placa de desarrollo para trabajar como controlador de nuestro sistema empotrado.

Figura2.2. Placa Arduino UNO.

Arduino UNO es la placa de desarrollo de uso profesional más genérica en su categoría. Dispone de un microcontrolador ATmega328. Cuenta con 14 pines digitales de entrada/salida (6 de los cuales pueden utilizarse para generar salidas PWM) y 6 entradas analógicas. Este modelo trabaja a 16 MHz, con un voltaje de entrada de 5V y cuenta con una memoria Flash de 32 KB (de los cuales 0,5 KB están reservados para el gestor de arranque), una memoria SRAM de 2KB y 1KB de EEPROM. La placa puede ser alimentada a través de un cable USB, mediante un adaptador de corriente o por una batería.

Resumen de las características técnicas Arduino UNO

Características (Arduino UNO)	
Microcontrolador	ATmega328
Voltaje operacional	5V
Voltaje de entrada (recomendado)	7-12V
Límites de voltaje de entrada	6-20V
Pines de E/S Digital	14 (6 pueden generar salidas PWM)
Pines de entrada Analógicos	6
Corriente continua por Pin E/S	40 mA
Corriente continua para el pin 3.3V	50 mA
Memoria Flash	32 KB (0.5 KB se usan para el gestor de arranque)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de Reloj	16 MHz
Tamaño	68.6 mm x 53.4 mm
Peso	25 g

Tabla 2.1. Características Arduino UNO

Arduino, además de un hardware flexible, nos proporciona de forma totalmente gratuita un entorno de desarrollo multiplataforma muy simple de usar. Como podemos apreciar en la Figura 2.3, y tal y como nos indica la web de dicha plataforma, el entorno de desarrollo de Arduino contiene un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para acceder a las funciones más comunes, y una serie de menús.

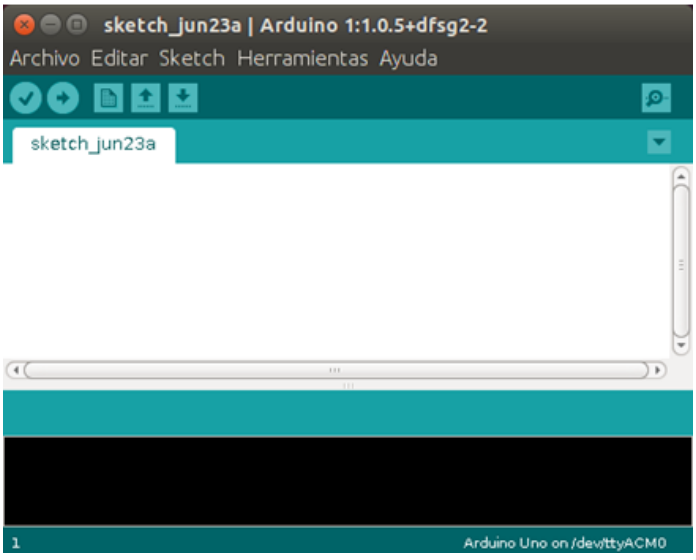


Figura 2.3. Entorno de desarrollo de Arduino.

Para que exista comunicación entre la placa y el entorno se debe seleccionar previamente el modelo de tarjeta que usamos, así como el puerto serie que queremos que el ordenador le asigne. Para ello accedemos al menú de Herramientas/Tarjetas, en el que encontramos un listado de todos los modelos de Arduino que existen en el mercado [Figura 2.4]. De igual modo, en el menú Herramientas/PuertoSerial se elige el puerto serie.

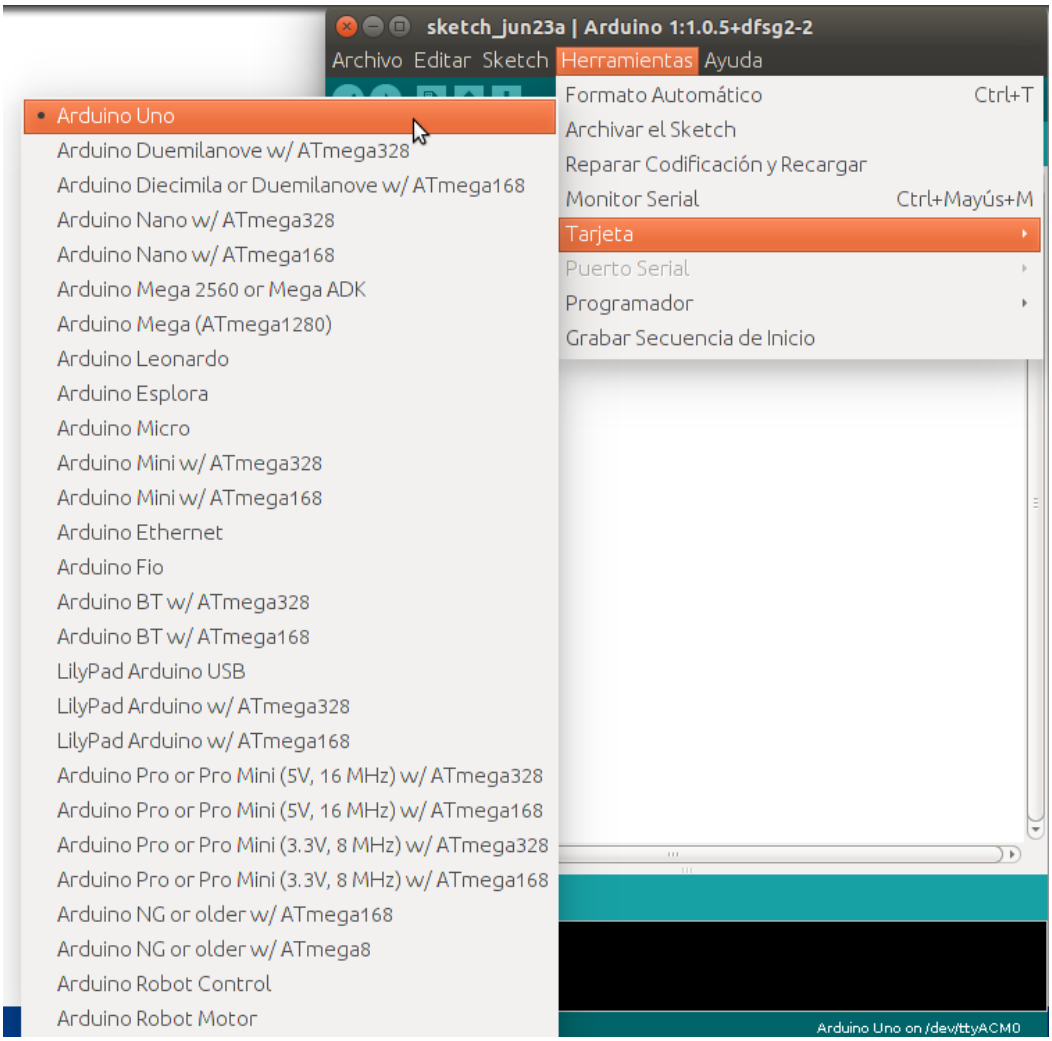


Figura 2.4. Elección de placa y puerto serie.

Los programas implementados en Arduino, conocidos dentro de la propia plataforma como "Sketch" y cuya extensión es .ino, suelen escribirse utilizando como lenguaje de programación C/C++. La estructura de cualquiera de sus programas se compone al menos de dos partes (funciones), ambas necesarias para el correcto funcionamiento del mismo.

La primera función, **setup()**, se invoca una única vez cuando empieza el programa y es la primera en ser ejecutada. Se encarga de realizar la

configuración, por lo que incluye las declaraciones de las variables así como la inicialización del modo de trabajo de los pines E/S.

La segunda función, **loop()**, realiza la mayor cantidad de trabajo, pues contiene el programa que se ejecutará cíclicamente, permitiéndonos responder de forma continuada ante los eventos que se producen en la placa (lectura de entradas, activación de salidas, etc.).

Además de las dos funciones obligatorias descritas anteriormente que han de estar siempre presentes, podemos crear todas aquellas funciones que deseemos, que deberán ser llamadas desde loop(). Para la implementación de este proyecto hemos creado algunas que serán comentadas en profundidad en el siguiente capítulo.

Como buen entorno de desarrollo, Arduino también nos ofrece la posibilidad de añadir un conjunto de librerías que nos brindan una mayor funcionalidad, o en otros casos, una simplificación del código. Algunas de estas librerías, las más habituales, se incluyen junto al software de Arduino. Sin embargo, si no fuese así, y queremos hacer uso de una librería que no venga incorporada, existe la posibilidad de descargarla e instalarla. El procedimiento para instalar una librería viene documentado dentro de la propia plataforma.

2.1.2 Sensor CCD lineal SONY ILX511

La pieza clave en nuestro diseño, que nos permite hacer uso del principio de triangulación, es un sensor CCD lineal. Los primeros dispositivos CCD fueron inventados por Willard Boyle y George E. Smith en el año 1969, en Los Angeles, Estados Unidos. Originalmente se concibió como un nuevo tipo de memoria de ordenador pero pronto se observó que tenía muchas más aplicaciones potenciales, tales como el proceso de señales y sobre todo la captación de imagen, esto último debido a la sensibilidad a la luz que presenta el silicio, material con el que se construyen. Este invento supuso para sus creadores ser galardonados en el año 2009 con el Premio Nobel de Física.

Los dispositivos de carga acoplada (en inglés charge-coupled device, conocido por sus siglas CCD) se basan en el efecto fotoeléctrico, es decir, en efectuar la conversión espontánea de luz recibida en corriente eléctrica gracias

a las propiedades de algunos materiales. Un CCD es un circuito integrado que al estar grabado, por lo general, en una superficie de silicio, forma elementos sensibles de luz llamados píxeles. Cada píxel recoge la intensidad de la luz en una fracción de tiempo del entorno y genera una carga eléctrica. El número de electrones producidos es proporcional a la cantidad de luz recibida y el ruido electrónico aumenta fuertemente con la temperatura.

Dependiendo de la distribución de los píxeles y del sistema que utilicen para recibir y capturar la luz se pueden diferenciar varios tipos de sensores CCD:

- **Arrays lineales**
  Los conjuntos lineales usan una fila única de píxeles. Los de un sólo CCD hacen tres exposiciones por separado: rojo/verde/azul (RGB) y se empezaron a usar en los primeros escáneres. Por otro lado, los Sensores Trileneales se fundamentan en el uso de tres CCD lineales unidos que se usan para capturar cada uno de los canales RGB en un sólo barrido.
- **Arrays de superficie**
  Son los más empleados actualmente en cámaras digitales, consisten en una superficie donde existen miles de píxeles sensibles a la luz organizados en filas y columnas, una matriz.

Para el diseño de nuestro sensor de medida de distancias, y teniendo en cuenta que nuestro objetivo es el cálculo de distancias basándonos en el principio de triangulación, se ha elegido un sensor CCD lineal, concretamente en suministrado por la compañía SONY, y cuyo modelo es el ILX511.

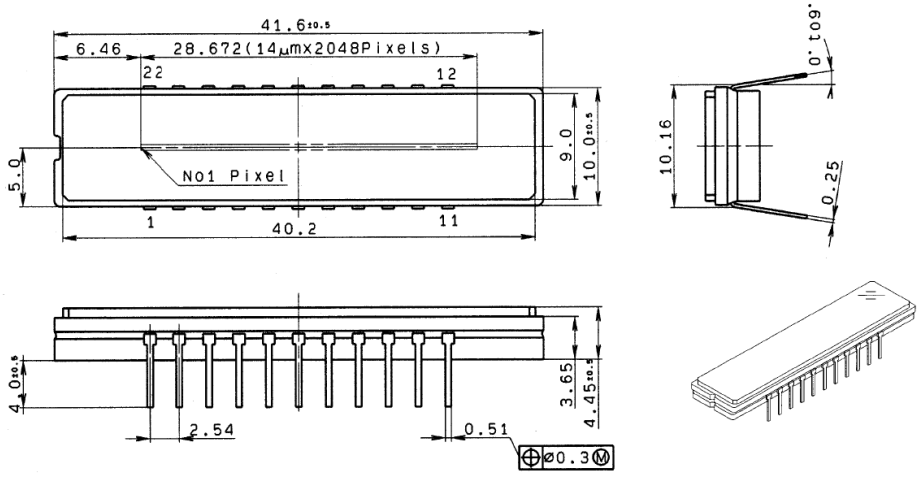


Figura 2.5. Representación CCD SONY ILX511.

Este modelo suele utilizarse para lectores manuales de código de barra o en equipos cuyo uso es la medición óptica. Dispone de 2048 píxeles dispuestos de manera lineal. El tamaño de cada uno de los píxeles es de 14µm x 200µm, dando como resultado que el sensor CCD sea un componente compacto y de unas dimensiones muy reducidas. Opera con un voltaje de entrada de 5V y una frecuencia de reloj máxima de 2MHz. Dispone de 22 pines. En la Figura 2.6 que nos proporciona la hoja de datos de fabricante se observa cómo vienen dispuestos.

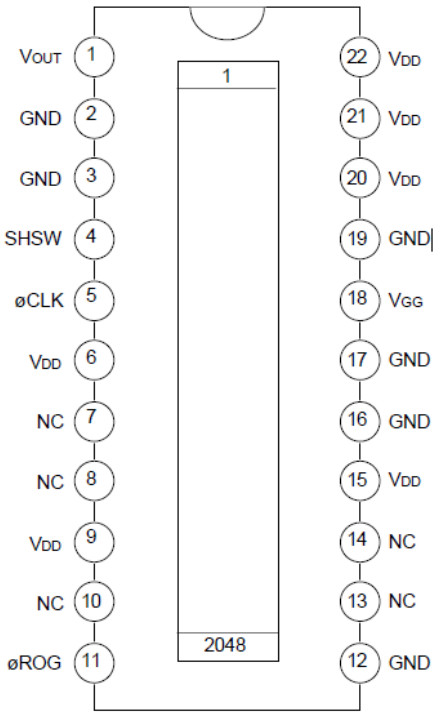


Figura 2.6. Pines CCD SONY ILX511.

Para comprender la conexión del sensor en nuestros sistema es aconsejable recurrir al esquema de conexión [Figura 2.1] del inicio del Capítulo 2, donde se aprecia que 6 pines estarán conectados a tierra (GND), 6 al voltaje de entrada (5V) y 5 pines permanecen sin conexión.

El pin 1 (VOUT) nos proporcionará la salida del voltaje del sensor. La salida del sensor (VOUT) únicamente podrá estar conectada al pin 8 de la placa, aunque deberá pasar previamente por el comparador electrónico que explicaremos más adelante.

Tal y como nos indica el fabricante existen dos posibles maneras de conectar el pin 4(SHSW), obteniendo una salida de voltaje del sensor (VOUT) diferente en cada caso.

Modo de uso PIN 4 (SHSW)	
Conectado a GND	Salida continua
Conectado a VDD	Salida con pulsos de sincronismo

Tabla 2.2. Modo de uso pin 4 (SHSW) del sensor CCD ILX511

Como vemos en la Figura 2.7, si se conecta el pin 4 del sensor a voltaje se genera una salida donde cada píxel está asociado con pulsos de sincronismo. En la hoja de datos del sensor se encuentra un esquema con esta configuración, así como la salida generada.

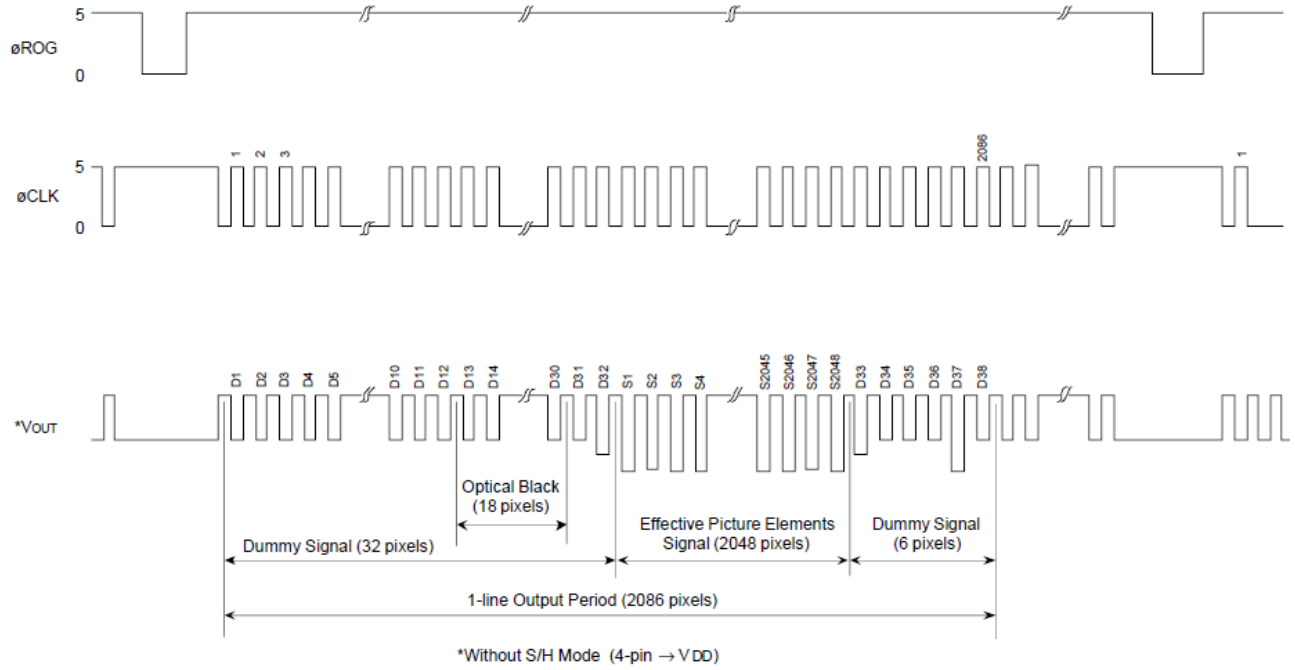


Figura 2.7. Señal de salida del sensor CCD ILX511 con el pin 4 (SHSW) conectado a VDD [Hoja de datos del fabricante].

La otra alternativa es la que hemos empleado para el desarrollo de nuestro proyecto, puesto que simplifica le lectura del sensor, tal y como se puede

apreciar en la Figura 2.8 si usamos el pin 4 (SHSW) conectado a GND, la salida del sensor da un voltaje continuo y variable relacionado con la lectura.

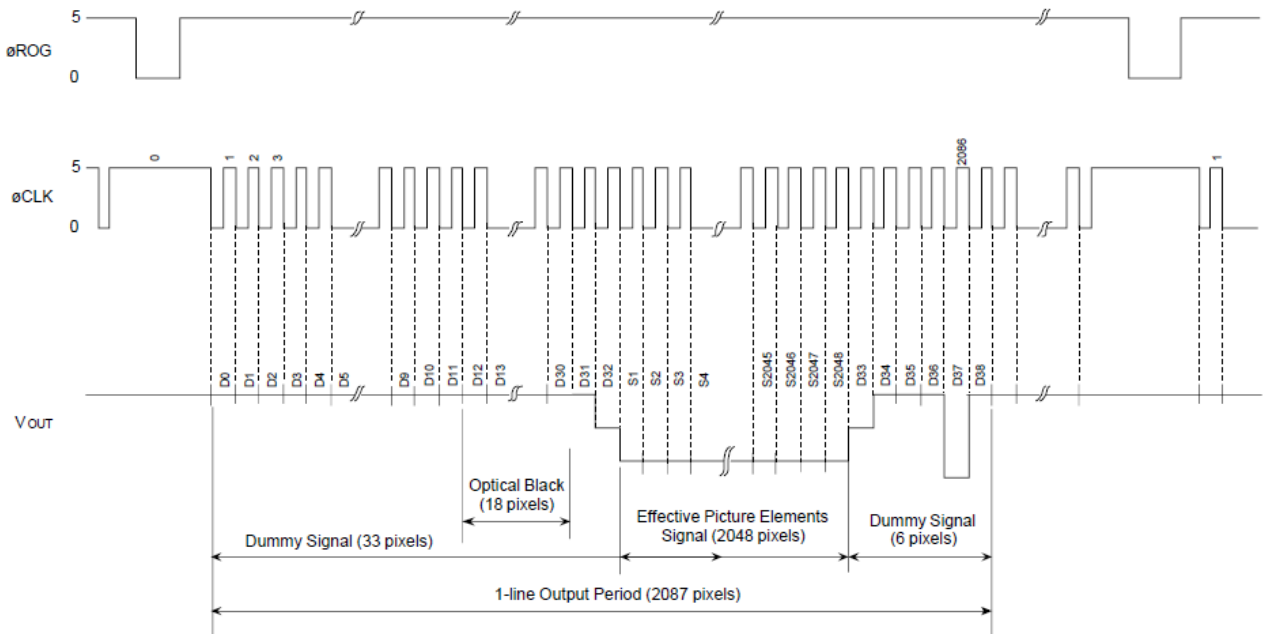


Figura 2.8. Señal de salida del sensor CCD ILX511 con el pin 4 (SHSW) conectado a GND [Hoja de datos del fabricante].

El CLK y ROG, serán las encargadas de asignar las señales de control. A través del pin 5 (CLK) podemos suministrarle al integrado una señal de reloj, mediante el pin 11 (ROG) somos capaces de controlar el tiempo de integración del sensor, es decir, el tiempo durante el cual los píxeles del sensor están acumulando luz. La cantidad de tiempo que el sensor permanece midiendo luz está sujeta a la frecuencia que se le den a los pulsos. Si esta es demasiado rápida, mide muy poco tiempo y no recibimos señal, si es demasiado lenta mide demasiado tiempo y se satura el sensor. Para seguir el esquema de tiempos y la generación de pulsos de la Figura 2.8 se ha utilizado Arduino UNO, conectando la señal ROG y CLK a Arduino en los pines digitales, 12 y 13 respectivamente [consultar Figura 2.1].

Por último el ping 18 (VGG), deberá estar conectado a un condensador, ya que el sensor necesita trabajar internamente con valores proporcionados por una fuente de voltaje estable.

2.1.3 Comparador

El modelo elegido de comparador electrónico es el LM193-N. Dicho componente consta de 2 comparadores de voltaje independientes, tal y como podemos apreciar en la Figura 2.9. Nuestro funcionamiento sólo requiere de un comparador, así que emplearemos el comparador A. Conectamos la alimentación y tierra en los pines 8 y 4 respectivamente. La entrada inversora (pin 2) estará conectada al nivel de referencia, la entrada no inversora (pin 3) se conecta a la salida del sensor CCD. Por último, conectaremos la salida de voltaje del comparador A (pin 1) al pin 8 de la placa Arduino. En la Figura 2.1 podemos ver el esquema lógico de la conexión del comparador dentro de nuestro circuito.

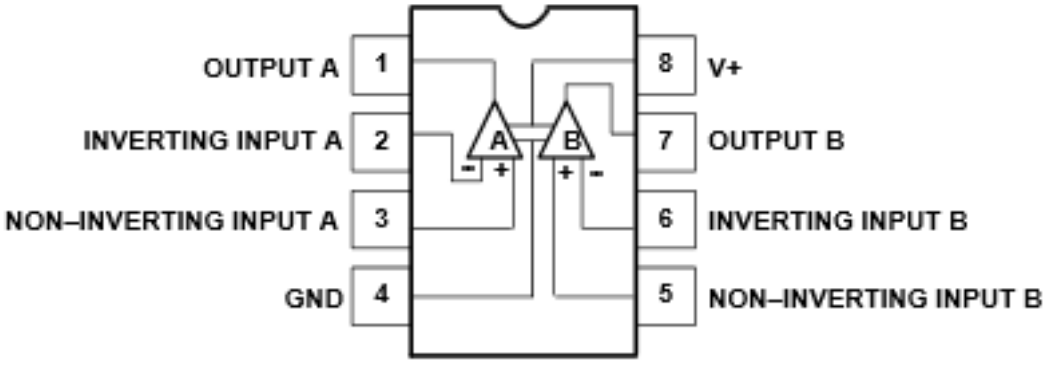


Figura 2.9. Comparador LM193N.

Para obtener la celda en la que incide mayor voltaje, lo que resultaría más fácil a priori es utilizar un conversor analógico-digital, puesto que la propia placa Arduino dispone de varios. Sin embargo, el tiempo que se tarda en efectuar la conversión en cada flanco de bajada de la Señal de Reloj, provoca un aumento en el periodo de integración del sensor haciendo que este se sature.

Analizando el problema, no necesitamos conocer completamente el valor de la señal convertida por el sensor, puesto que únicamente buscamos detectar el obstáculo más cercano. En un sistema de triangulación este obstáculo cercano coincide con el primer objeto que provoque una reflexión significativa. Esta reflexión se corresponderá con un voltaje de salida mayor a un determinado umbral, con lo que se puede utilizar un comparador para detectar cuándo se alcanza este umbral.

En la Figura 2.10 que se muestra a continuación, se puede ver la relación que existe entre la lectura de la señal de salida del sensor, y la distancia a la que se encuentra el objeto. En el osciloscopio, el canal 1, línea amarilla de la imagen, nos indica el periodo de integración del sensor. El canal 2, línea azul, muestra la salida de lectura del sensor, y tal y como hemos explicado, el flanco de bajada dependerá de la distancia a la que se encuentre el objeto más cercano.

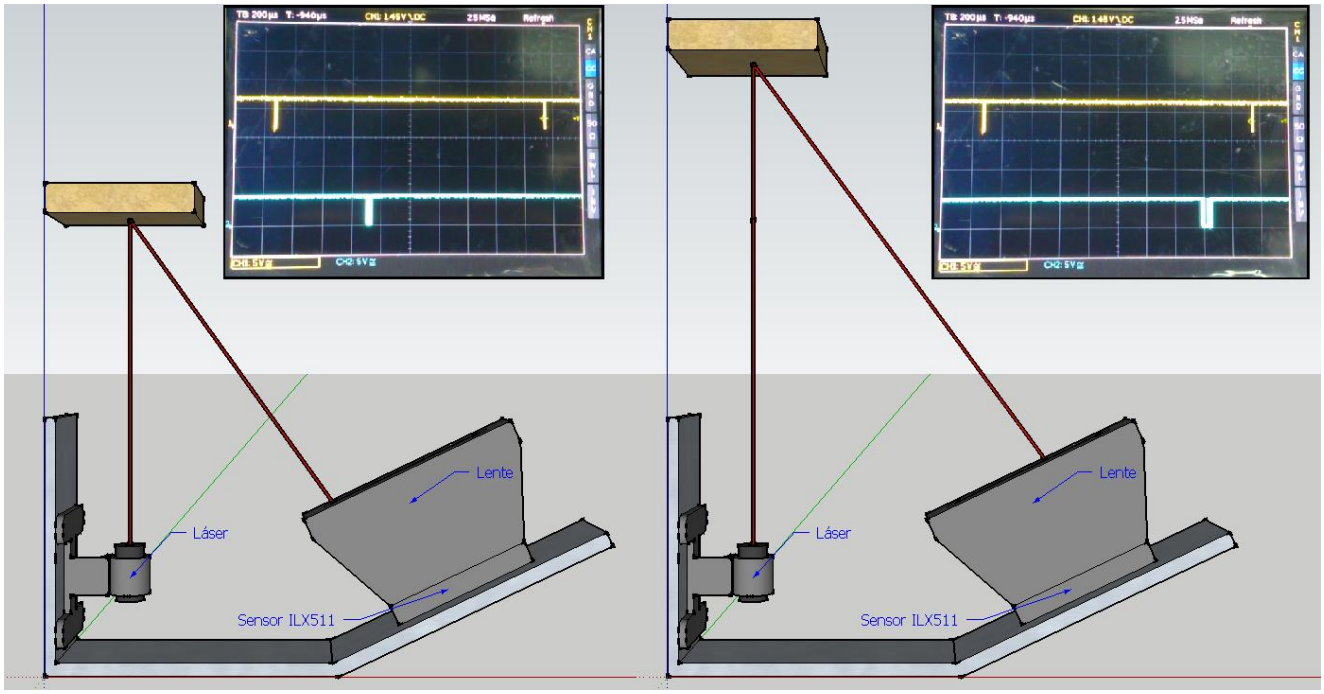


Figura 2.10. Relación entre la distancia del objeto y la salida del sensor CCD.

Para saber cuándo se produce la detección de la celda con mayor voltaje, simplemente se calcula el tiempo que ha pasado desde que se empieza el ciclo de lectura hasta que se detecta. Este tiempo es proporcional al píxel en el que se ha encontrado, es decir, cada celda del sensor CCD va a estar asociada a un determinado tiempo [Figura 2.11], de tal manera, que si iniciamos un contador de tiempo al comienzo de cada lectura podemos encontrar una relación entre el tiempo que se tarda en encontrar el píxel con mayor voltaje (flanco de bajada) y la distancia a la que se encuentra el objeto.

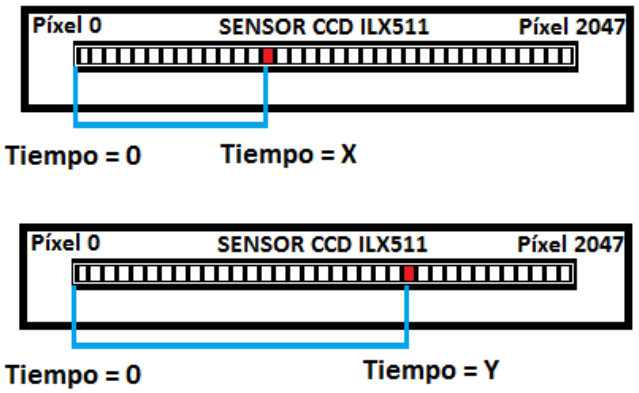


Figura 2.11. Relación entre el tiempo y la distancia al píxel donde incide mayor cantidad de luz.

Para calcular el tiempo que tardamos en encontrar la celda con mayor voltaje, se ha instalado una librería de Arduino, de nombre **TimerOne.h** que nos proporciona un temporizador (timer).

Dentro de Arduino definimos un temporizador como un registro de 8 o 16 bits que funciona independiente a la CPU y cuya propiedad fundamental es que su valor aumenta o disminuye de forma automática a una velocidad establecida por el programador (prescaler). Arduino UNO dispone únicamente de tres temporizadores, el timer0 y el timer2 son de 8 bits, por lo que nos permite trabajar con un rango de valores entre 0-255. En cambio, el timer1 es un registro de 16 bits que nos ofrece hasta 65536 posibles valores. En nuestro dispositivo utilizaremos el timer1. En la hoja de datos del Fabricante de ATmega328 se encuentra una explicación detallada de los timers de Arduino, así como su funcionamiento y los distintos modos de configuración, ya que son un recurso bastante complejo que nos proporciona la placa.

Dentro de las posibles configuraciones del timer1, se incorpora una unidad de "Input Capture" que permite capturar eventos externos asociados a cambios en el pin ICP1 (en Arduino Uno el pin 8). Para establecer este modo de configuración, el timer1 utiliza varios registros (TCCR1, TIMSK y TIFR). La asignación de bits, de una manera determinada especificada en la hoja de datos de ATmega, dentro de estos registros nos permite iniciar el contador y configurar el ICP1 para capturar flancos de bajada. Esta configuración del timer1 hace que se ejecute una interrupción cuando se detecta el evento (flanco de bajada) como se ha visto anteriormente, en las Figuras 2.10 y 2.11.

Para que sirva como resumen aclaratorio de este apartado, al inicio de cada lectura del sensor CCD, se inicia el incremento del Timer1, el tiempo que se tarde capturar el evento de flanco de bajada (píxel con mayor voltaje), será almacenado en un registro temporal y estará asociado proporcionalmente a la distancia a la que se encuentra el objeto.

2.1.4 Lente

Con el objetivo de evitar ruido, y para que los rayos de luz incidan de manera correcta en el sensor CCD, ha sido necesario emplear una lente. En el mercado disponemos de una gama amplia, pero como la idea es acoplar la lente al sensor ILX511, cuya superficie es rectangular, se ha considerado oportuno reciclar la lente de un escáner, puesto que su base se ajusta correctamente al sensor.

2.1.5 Láser

Un láser es una fuente de luz. A diferencia de otras fuentes de luz más comunes, como las bombillas, cuyo funcionamiento se basa en la emisión espontánea, el láser emite utilizando el mecanismo físico denominado emisión estimulada, de ahí su acrónimo (Luz amplificada por emisión estimulada de radiación). Para entender lo que es la emisión espontánea y la emisión estimulada hay que conocer la física de la interacción de átomos con fotones. Sin embargo, lo que pretendemos es explicar el por qué de la necesidad de usar este dispositivo en nuestro sensor. Un láser nos va a proporcionar las propiedades que el mecanismo de emisión estimulada confiere a la luz, como son la alta potencia (y su capacidad para ser amplificada), la direccionalidad (emisión en forma de "rayos"), la frecuencia de emisión bien definida (color de la luz), así como la capacidad de emitirse en pulsos de muy corta duración.

Para el funcionamiento de nuestro sensor de medidas de distancia, el láser va a emitir un haz de luz de forma permanente. Además, dentro de la amplia gama que se puede encontrar en el mercado, se ha decidido trabajar en el espectro del rojo, habiendo descartado el infrarrojo, puesto que al no poder visualizar donde incide el puntero láser, se dificultan tareas simples como la de calibrado.

En la siguiente imagen, Figura 2.12, se puede ver el prototipo tras haber terminado la primera fase de desarrollo. Como se aprecia, hasta este momento el diseño no ha sido motivo de preocupación, ya que el objetivo era saber si podíamos calcular la distancia a un punto.

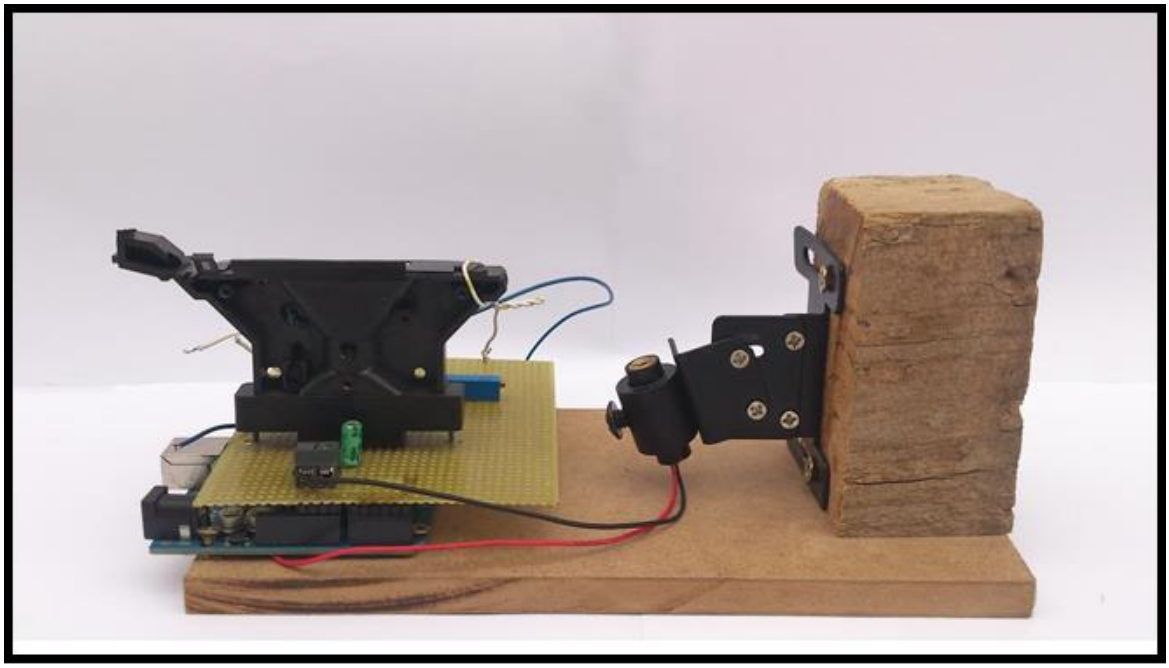


Figura 2.12. Primer prototipo funcional.

2.2 Conexión del Motor Paso a Paso

Nos hemos propuesto como objetivo calcular la distancia a la que se encuentran los objetos en un plano determinado, por lo que necesitamos de algún mecanismo que nos permita mover toda la estructura (sensor CCD, láser y lente) de forma controlada, es decir, sabiendo cuánto y hacia dónde se ha desplazado. Esta es donde empieza la segunda fase de desarrollo y para la cual hemos usado un Motor Paso a Paso, un controlador de motores (L298N) y un interruptor óptico. En la siguiente imagen, Figura 2.13, se presenta el esquema de conexión de todos los componentes aquí implicados.

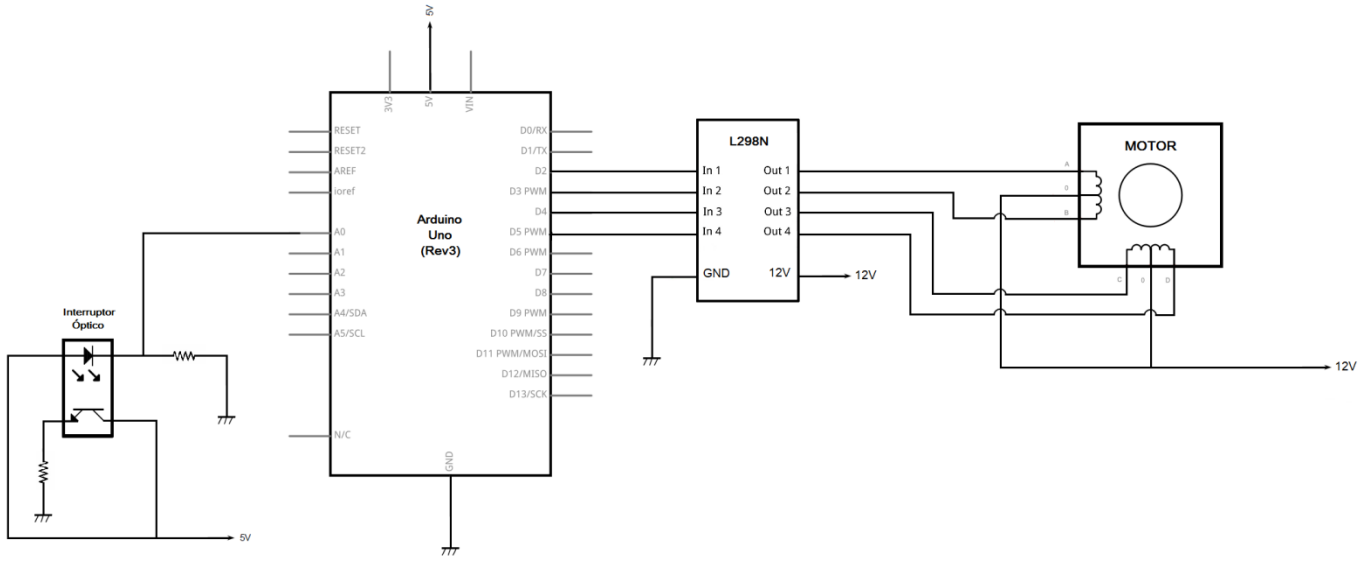


Figura 2.13. Esquema de conexión Motor PaP - Arduino.

2.2.1 Motor Paso a Paso (Stepper)

Un motor paso a paso, conocido también en español como motores PaP y en inglés como stepper, se define como un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares, siendo un componente imprescindible para la construcción de dispositivos donde se requiere de movimientos muy precisos. El resultado de este movimiento, fijo y repetible, es un posicionamiento exacto y fiable. Un motor PaP puede girar en ambos sentidos, con un incremento mínimo determinado por el diseño. El funcionamiento de este tipo de motores se basa en la secuencia de encendido y apagado del par de bobinas que lo forman. Dependiendo del número de cables podemos distinguir entre:

- **Motores Bipolares.**
  Tiene dos bobinados, correspondiendo cada uno de ellos a una fase. Dispone de cuatro hilos, dos para cada bobinado. El control se realiza forzosamente de forma bipolar, normalmente mediante un puente.
- **Motores Unipolares.**
  El bobinado por cada fase es doble, unido en el interior y puesto en serie nos entrega 6 hilos, agrupados de tres en tres para cada fase (uno de estos es el punto común, por lo que dependiendo del esquema interno del motor, encontramos motores unipolares de 5 cables). El control es

unipolar, aunque se puede realizar un control bipolar dejando el hilo central al aire.

Además del número de cables, dos de las principales características que debemos tener en cuenta, a la hora de adquirir un motor paso a paso, son su resolución y el voltaje con el que debe ser alimentado. La resolución de un motor paso a paso nos permite saber cuántos pasos necesita para dar un giro completo, por tanto, si tenemos un motor PaP cuya resolución es de 3,6 grados por paso, significa que necesitará de un total de 100 pasos para completar un giro (360 grados). Además, como todo componente electromecánico, los motores PaP precisan de una tensión eléctrica de trabajo, por lo general impreso en su carcasa o especificado en la hoja de datos. Sin embargo, a veces puede ser necesario aplicar un voltaje superior para lograr que un determinado motor cumpla con el torque deseado, lo que provoca un aumento de la temperatura, viéndose afectada a largo plazo la vida útil del motor.

Durante la segunda fase de desarrollo, con la intención de aprender el manejo de motores PaP utilizando Arduino, se decidió reciclar este componente de una impresora, puesto que nuestro diseño no requiere de unas características específicas, sino más bien, algo usual que nos puede ofrecer casi cualquier motor PaP que encontramos en el mercado. El motor PaP empleado en el proyecto es de tipo Unipolar, con 5 cables. Los valores de alimentación oportunos están entre 6V-12V. Nos ofrece una resolución de 1,8 grados por paso, es decir, para completar un giro de 360 grados necesitaremos dar 200 pasos.

Figura 2.14. Motor paso a paso Unipolar.

Para controlar el motor paso a paso se ha usado otra librería, de nombre "Stepper.h". En este caso no necesita ser instalada, viene junto al propio software de Arduino y nos va a permitir controlar el motor paso a paso de forma sencilla, pues se complementa simplemente de 3 funciones. Es importante destacar que trabajando con esta librería el circuito de conexión cambiará en función del uso de un motor Unipolar o Bipolar. Además nos permite controlar cualquier motor PaP usando dos o cuatro pines. Como no se tiene previsto incluir más componentes en la placa, decidimos escoger el esquema de conexión más simple y hacemos uso del control del motor usando 4 pines, siendo un dato importante de escalabilidad para futuras versiones del sensor láser de medida de distancia.

- **Stepper (steps, pin1, pin2, pin3, pin4)**
  Crea una instancia de la clase Stepper. El número de parámetros dependerá del esquema de conexión que se haya utilizado. Al utilizar 4 cables para controlar el motor, se le deben pasar 5 parámetros. El primero corresponde con el número de pasos del motor PaP necesarios para dar una vuelta de 360 grados (steps). Los otros cuatro parámetros son los pines de control para ser conectado con Arduino. Debe ser llamada antes del setup() y de loop().
- **setSpeed(rpm)**
  Establece la velocidad del motor en revoluciones por minuto (RPM).
- **step(steps)**
  Hace que el motor gire un cierto número de pasos, steps. Si el valor de steps es positivo gira en un sentido, si es negativo gira en el sentido opuesto.

Para manejar el motor deberemos hacer uso de un controlador. Arduino dispone de varios controladores en el mercado que facilitan el manejo de motores paso a paso. A continuación pasamos a describir el empleado en nuestro diseño.

2.2.2 L298N Controlador de Motor Paso a Paso

El manejo de motores de corriente continua requiere de un circuito muy conocido en electrónica llamado "Puente H" que permite al motor girar en ambos sentidos. El circuito está compuesto por 4 interruptores mecánicos o

mediante transistores. El nombre “Puente H” proviene de la representación gráfica del circuito, tal y como podemos ver en la Figura 2.15.

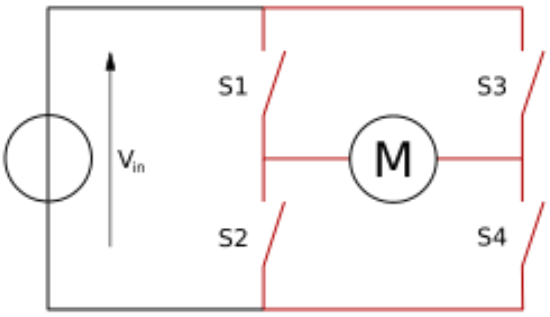


Figura 2.15. Representación gráfica de un circuito Puente H.

Como vemos en la representación anterior [Figura 2.15], cuando los interruptores S1 y S4 están cerrados, permaneciendo S2 y S3 abiertos, se aplica una tensión positiva en el motor, haciéndolo girar en un sentido. Del mismo modo, si abrimos los interruptores S1 y S4 y cerramos S2 y S3, el voltaje se invierte, obteniendo como resultado que el motor gire en sentido contrario. Además, los interruptores S1 y S2 no deberán estar cerrados al mismo tiempo, puesto que esto cortocircuitaría la fuente de alimentación. Lo mismo sucede para S3 y S4.

Los puentes H están disponibles en el mercado como circuitos integrados, pudiéndose construir si se desea utilizando componentes discretos, aunque no supone un gran diferencia de coste final. El controlador que hemos adquirido para el manejo del motor paso a paso es el L298N, Figura 2.16. Este componente no es más que la combinación de dos puentes H, lo que nos permite manejar dos motores de corriente continua o un motor paso a paso.

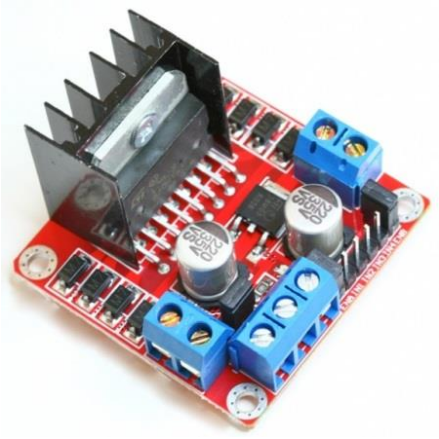


Figura 2.16. L298N. Controlador de Motores paso a paso.

La ventaja de utilizar este módulo, cuando deseamos hacer funcionar un motor paso a paso con Arduino, es que no se necesita de componentes adicionales.

El módulo dispone de un regulador LM7805 que suministra 5 V a la parte lógica del integrado L298N. Además los jumpers de selección nos permiten habilitar cada una de las salidas del módulo (A y B). La salida A está formada por OUT1 y OUT2, la salida B por OUT3 y OUT4. Por otro lado dispones de dos pines de habilitación para cada una de las salidas que deberán estar conectados (ENA y ENB). La explicación de los jumpers queda detallada en la Figura 2.17.

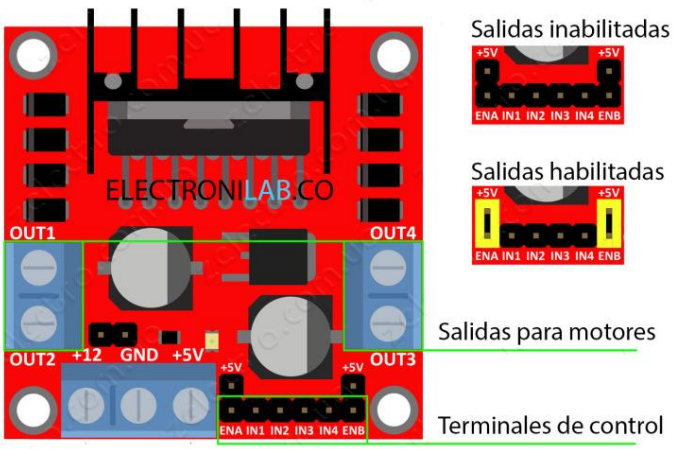


Figura 2.17. Jumpers de selección del módulo L298N.

Para ver como se ha efectuado la conexión entre el motor y el controlador L298N con Arduino acudir a la Figura 2.13.

- **Resumen de las características técnicas L298N**

Características (Arduino UNO)	
Alimentación lógica	6V - 12V
Rango de alimentación	4.8 ~ 35V
Corriente máxima	2A
Temperatura de funcionamiento	-25 a +130 ºC
Dimensiones	45×45mm
Peso	29g aprox.

Tabla 2.3. Características L298N

2.2.3 Interruptor óptico

El motor paso a paso nos proporciona un posicionamiento conocido y estable en todo momento durante su recorrido, sin embargo se desconoce el punto de partida. Al no empezar siempre en el mismo punto, no se puede saber cuál es su posición respecto al entorno. A esta posición inicial, que será siempre la misma, se le conoce comúnmente como Home Position. Para asignar un punto de partida al motor paso a paso se ha hecho uso de un interruptor óptico.

El funcionamiento de un interruptor óptico consiste en enfrentar dos elementos electrónicos, por un lado un diodo emisor de infrarrojos y en el otro extremo un fotodiodo o fototransistor como elemento de detección. Mientras el receptor reciba el haz de luz producida por el diodo el circuito permanece cerrado, desde el momento que un objeto se interponga entre ambos, el circuito se abre.

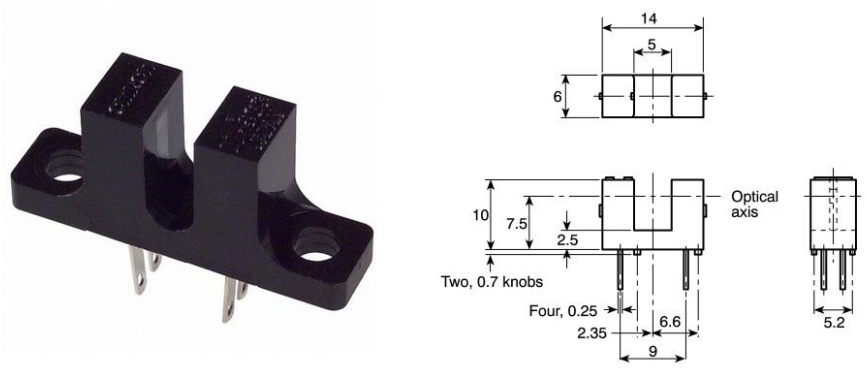


Figura 2.18. Interruptor óptico.

Para localizar la Home Position, desde la que nos interesa que el motor comience a efectuar un barrido cíclico, por ejemplo de 180 grados, le hemos colocado al eje del motor una extensión que nos permite cortar el haz de luz en una posición determinada, de tal forma que el interruptor tendrá unos valores similares para todos los pasos del motor, exceptuando aquel en el que el haz de luz se interrumpe. Para leer estos valores analógicos que nos proporciona el interruptor óptico, se ha conectado la salida de este componente a un ping de entrada analógica de Arduino, que haciendo uso del ADC nos proporciona valores entre 0-1023 en función de la luz que recibe el fotodiodo. La conexión de este componente a la Placa Arduino la vemos nuevamente en la Figura 2.13.

En la Figura 2.19, se ve el diseño del prototipo correspondiente a la finalización de la segunda fase de desarrollo, preparado para generar un barrido láser 2D.

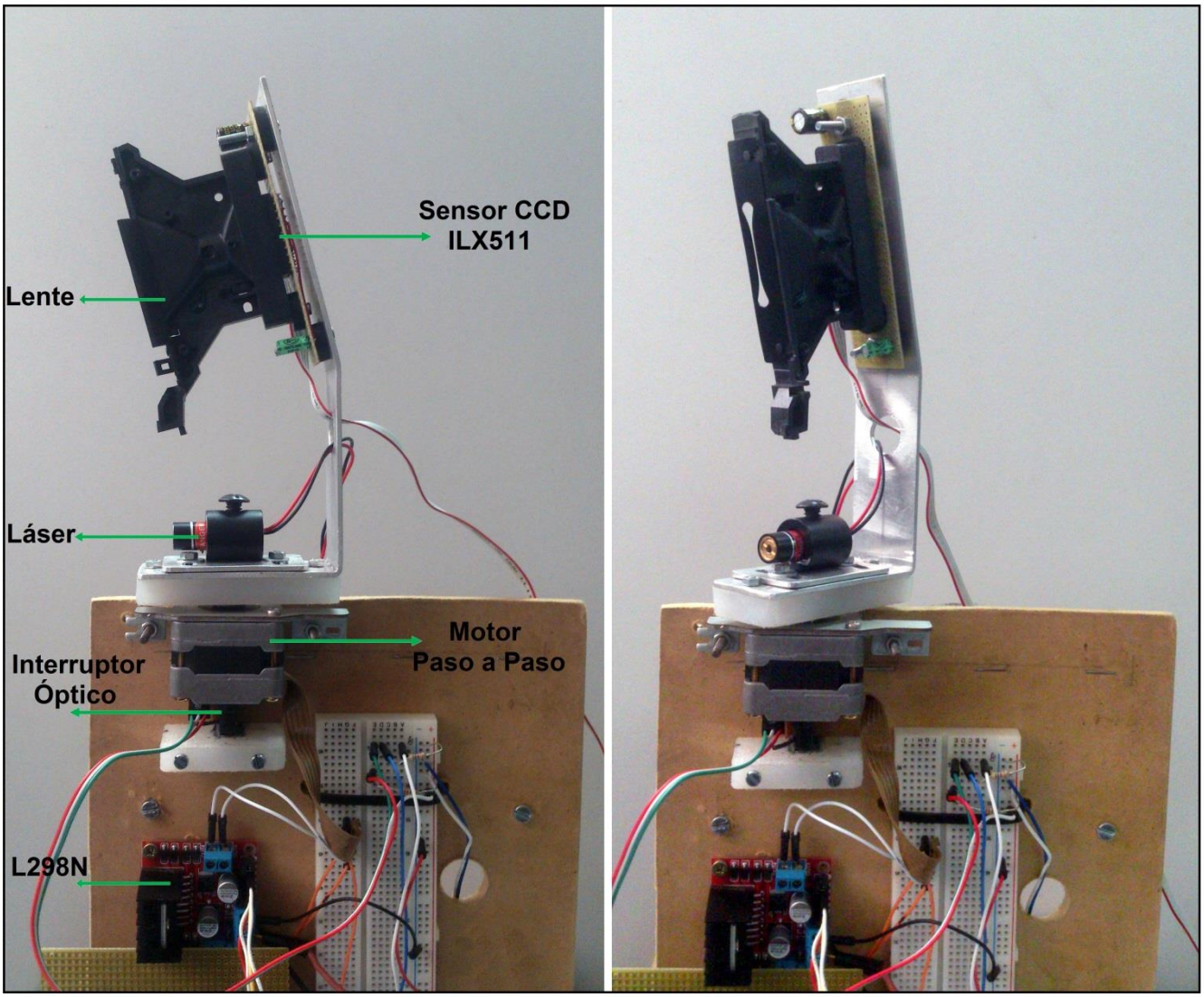


Figura 2.19. Último prototipo realizado.

Capítulo 3.
Desarrollo

El objetivo de este capítulo es entender el código que hemos cargado en la placa de Arduino para hacer funcionar nuestro sensor láser de medida de distancias. Para ello se explicaran las funciones que hemos implementado, apoyándonos en un diagrama de flujo [Figura 3.1], que servirá de ayuda para comprender la trazabilidad que sigue el programa general, que combina tanto la lectura del sensor CCD como el movimiento de la estructura llevado a cabo por el motor paso a paso.

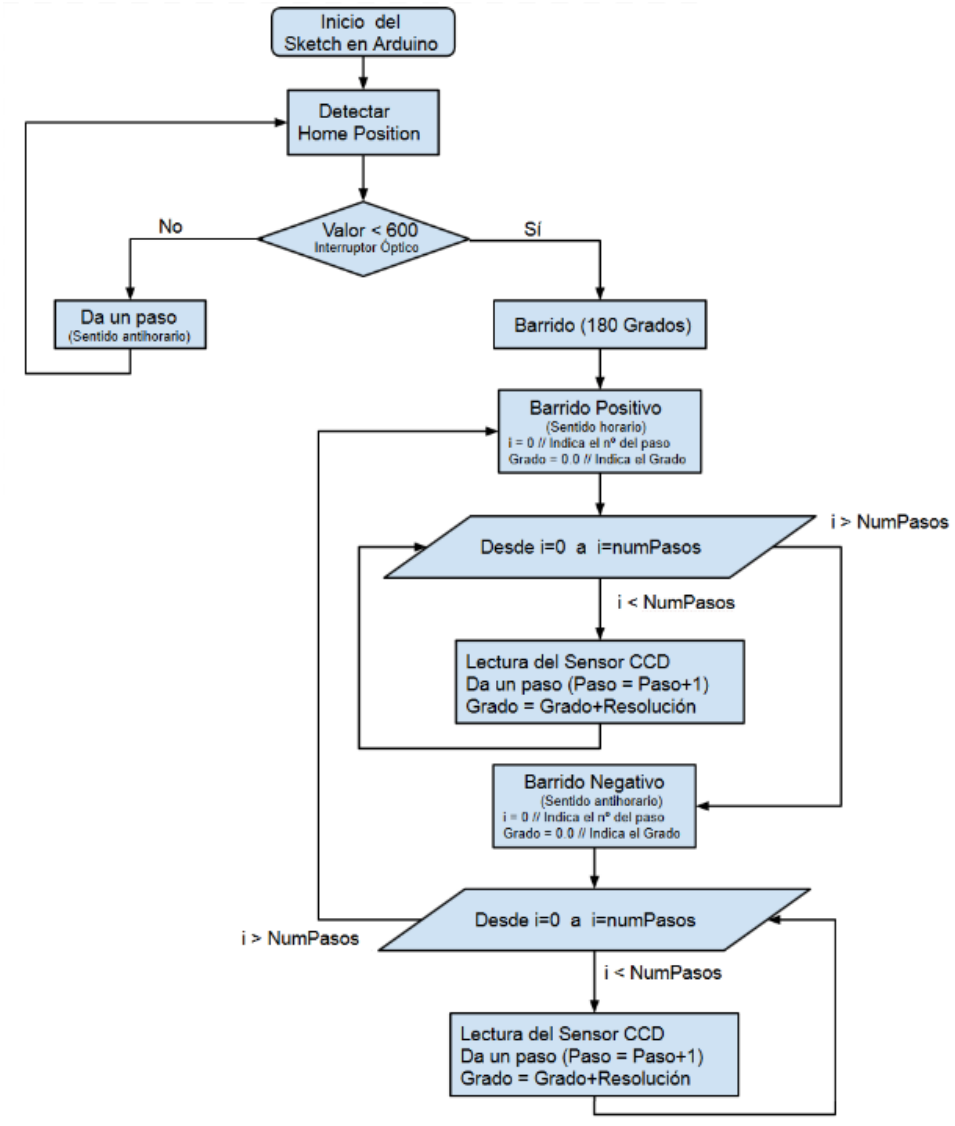


Figura 3.1. Diagrama de flujo.

3.1 Funciones implementadas

3.1.1 Detectar Home Position, homePosition()

El primer punto que tenemos que tener en cuenta es la detección de una misma posición cada vez que arranquemos el dispositivo. Para detectar la posición inicial del motor paso a paso que se explicó en el capítulo anterior hemos desarrollado una función llamada homePosition(), que lee el valor que nos proporciona el CAD al que está conectado el interruptor óptico. Cuando el haz de luz no está interrumpido, la lectura que se obtiene son valores cercanos a 1000. Lo ideal sería que la pieza que le hemos colocado al eje del motor para cortar el haz de luz nos generase un cero, o valores próximos a este cuando se interrumpe el haz. Sin embargo el valor que se obtiene, puesto que se "filtra" algo de luz, es de 600. Si los valores del interruptor óptico están por encima de 600, significa que no es la posición inicial, por lo que se da un paso en sentido antihorario y se vuelve a leer el valor del sensor hasta que encontremos un valor por debajo de 600, siendo ese el paso desde el que debemos iniciar siempre el dispositivo. Cuando encontremos la Home Position (valor menor de 600 en el interruptor óptico) la siguiente acción a ejecutar es la llamada de la función barrido(), tal y como vemos en el Diagrama de Flujo [Figura 3.1].

3.1.2 Generar un barrido, barrido()

La función barrido(), como su propio nombre indica, efectuará un barrido de 180 grados, es decir, dará de manera cíclica 100 pasos en sentido horario y otros 100 pasos en el sentido opuesto. A cada paso hace una lectura del sensor CCD, por lo que se obtiene el tiempo que se tarda en localizar la celda donde está incidiendo mayor reflexión. Este tiempo es almacenado por el timer1 en una variable temporal, que imprimimos por el puerto serial.

Además de leer el sensor CDD en cada paso que damos, se va incrementando la variable "grado". El resultado es que a cada paso, y por tanto, cada 1,8 grados, se obtiene un tiempo que se imprime en consola y será traducido a distancias por el ordenador utilizando la función que explicamos en el Capítulo 4. De esta forma queda asociada una distancia a un determinado ángulo, obteniendo un barrido 2D.

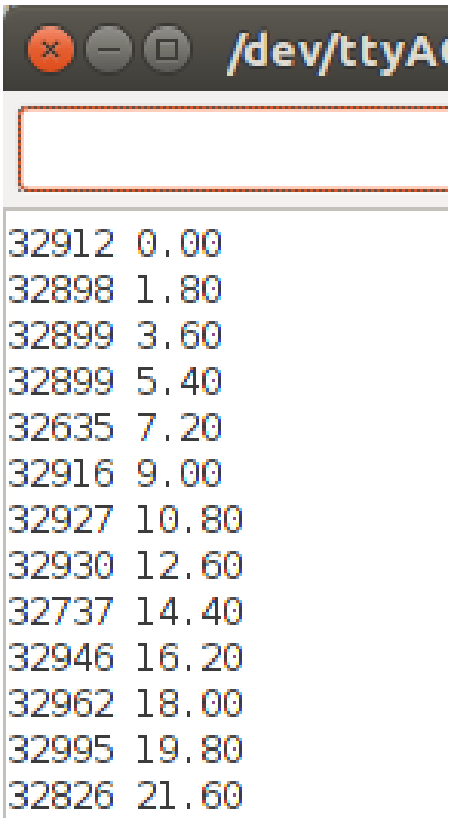


Figura 3.2. Relación de Tiempo de lectura del CCD y grado del motor PaP.

3.1.3 Lectura del sensor CCD, lecturaSensor()

Para seguir el quema de tiempos del sensor CCD [Figura 2.8] que nos indica el fabricante, hemos implementado una función, lecturaSensor(), que será llamada a cada paso del motor.

Como vemos en la tabla de tiempos, al principio de un ciclo de lectura es necesario reiniciar el periodo de integración del sensor. Se consigue poniendo a cero, durante un corto instante, la señal ROG. Cuando se vuelve a activar esta señal, comienza el ciclo de lectura del sensor, y debido al número de píxeles, se necesita generar más de 2088 pulsos de reloj, para lo cual, se efectúa un bucle donde se activa y desactiva la señal de reloj 2100 veces.

El código de implementación de estas tres funciones se encuentra en el Apéndice A.

Capítulo 4. Calibrado

Con la calibración de nuestro dispositivo relacionamos, para cada lectura del CCD, el tiempo que se tarda en encontrar el primer píxel que presente un voltaje de reflexión alto, con el objeto más cercano en el que impacte el haz de luz del láser. Para ello hemos hecho uso de un telémetro láser industrial. Alineando ambos dispositivos y haciendo incidir la luz del láser en un mismo plano, obtenemos de nuestro sensor una serie de valores de tiempo proporcionados por el timer1 y una distancia real al objeto suministrada por el telémetro láser.

- **Rango de actuación.**

Existen dos magnitudes en la estructura física del dispositivo que afectan al calibrado, una de ellas es la distancia entre el sensor CCD y el láser. La otra es el ángulo del sensor CCD respecto al láser. Si se modificase alguna de estas magnitudes, sería necesario calibrar el dispositivo nuevamente. Estás dos variables, afectan a la sensibilidad del sensor CCD y al rango de actuación del dispositivo, es decir, a las distancias mínimas y máximas que podremos medir.

Tras probar con distancias y ángulos diferentes entre los componentes de nuestro dispositivo, se confeccionó una estructura con un ángulo de 15 grados y una distancia entre el sensor y el láser de 8 centímetros. Con esta configuración geométrica se obtiene un rango de actuación cuya distancia mínima medible es de 6 centímetros, y como distancia máxima se supera escasamente 1 metro. Si decidimos modificar alguna de estas variables estructurales, se modificará el rango de actuación del sensor. Se podrán obtener distancias mayores aumentando la distancia que separa los dos componentes, o bien, si se disminuye el ángulo del sensor CCD respecto al láser. Esta acción tiene por contrapartida que la distancia mínima medible aumenta considerablemente, por lo que se pierde la esencia del proyecto, puesto que en navegación importan más las distancias cercanas al sensor. El esquema gráfico de esta explicación lo vemos en la Figura 4.1.

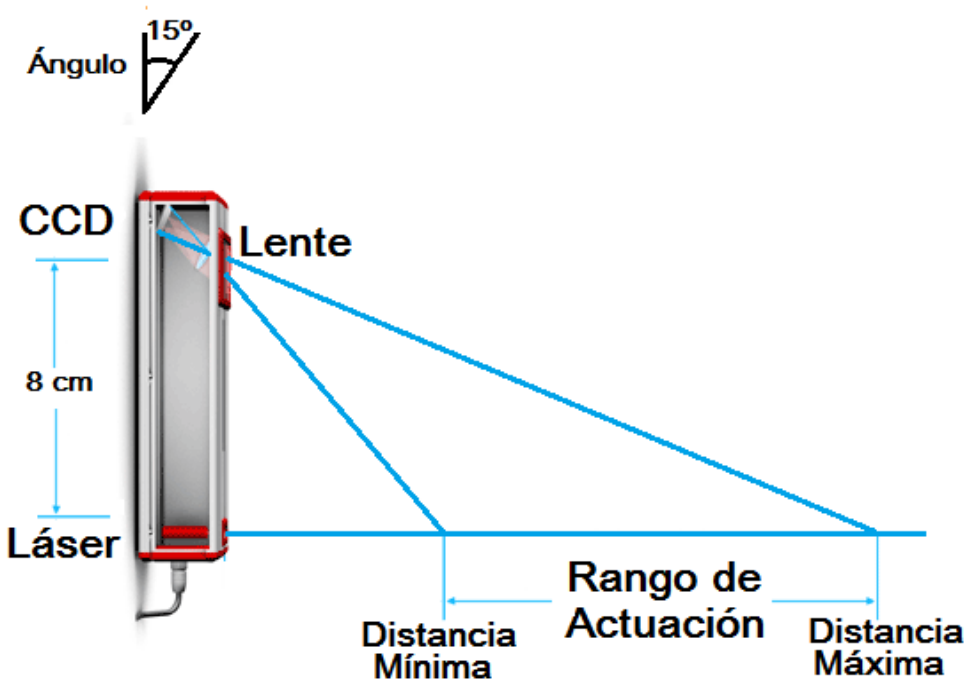


Figura 4.1. Distancia entre CCD y el láser. Rango de Actuación.

- **Eliminación del ruido**

Para efectuar un calibrado consistente, ha sido necesario eliminar el ruido y elegir una marca de tiempo que represente a cada distancia. La existencia de ruido se produce por la sensibilidad a la luz ambiente a la que se ve afectado el sensor CCD, que da lugar a que los valores obtenidos del timer1 no serán exactamente iguales para un mismo punto, aunque si presentan una correlación bastante fuerte. Para cuantificar la cantidad de ruido en cada lectura, hemos hecho uso de un contador (count), que se incremente cada vez que se ha producido un flanco de bajada. Aunque el timer1 almacena únicamente el tiempo al primer píxel, si durante la lectura se han producido varios flancos de bajada, es evidente que se ha visto afectada por una fuente de luz no controlada, descartando esa medida como válida para el calibrado. En la Figura 4.2 vemos una tabla de tiempos para un objeto que se encuentra a 25 cm. Se han descartado dos medidas de tiempo durante la lectura del sensor puesto que existen 2 flancos de bajada. Como consecuencia su valor difiere del resto, así que deducimos que esas lecturas se han visto afectadas por el ruido y no las tenemos en cuenta para calibrar el sensor.

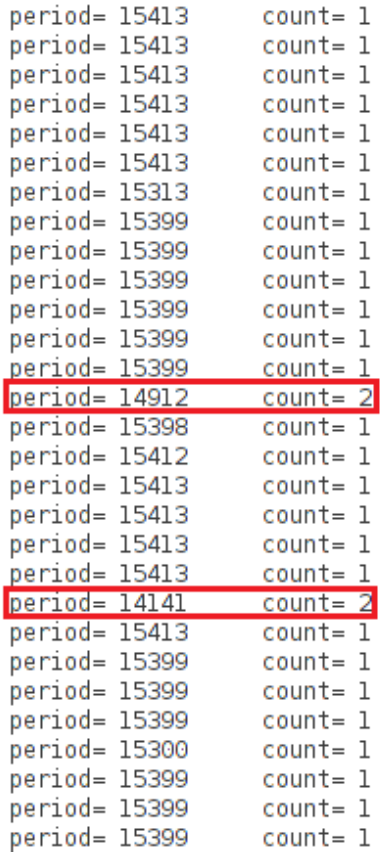


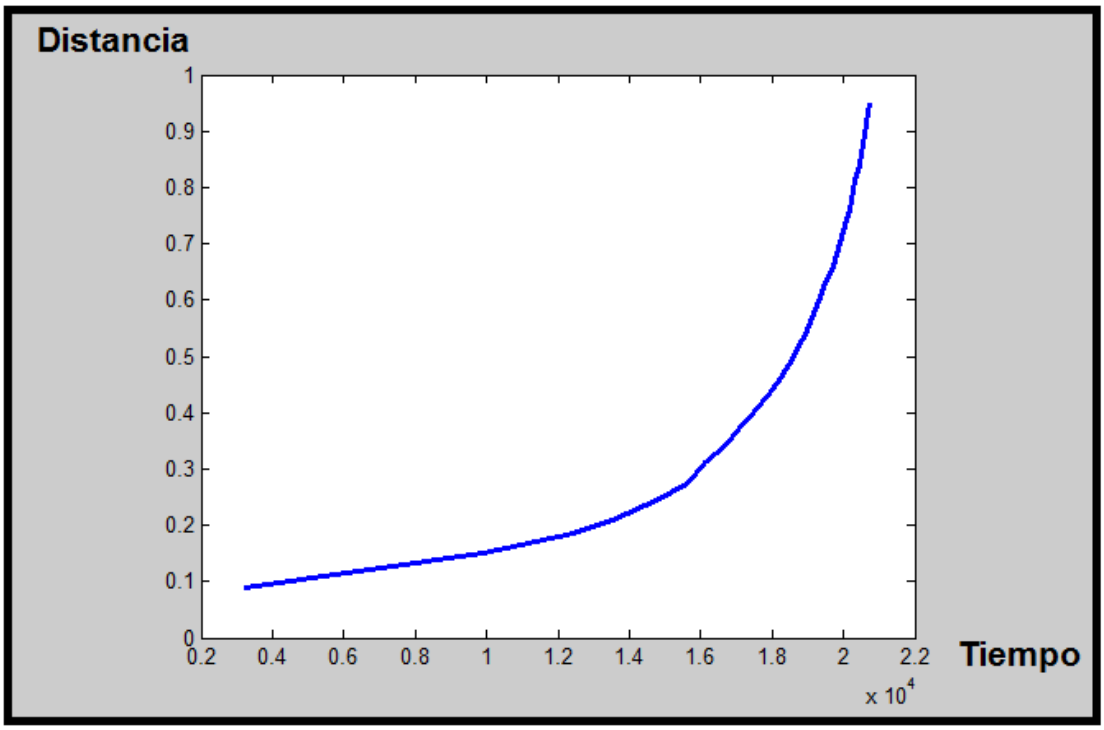
Figura 4.2. Medidas de calibrado descartadas.

Para seleccionar un valor de tiempo adecuado a una distancia determinada, se busca la moda, es decir, valor que más se repite dentro de la columna de tiempos [Figura 4.2]. A continuación se muestra una tabla [Tabla 4.1], en la que se puede visualizar una relación entre los tiempos obtenidos a diferentes distancias. Si nos fijamos detenidamente vemos que la relación de valores no sigue un incremento lineal, esto se debe al ángulo que existe entre el sensor CCD y el láser. Para obtener la función que se ajuste a la nube de puntos hemos recurrido a Matlab. Dado un conjunto de pares de datos (puntos en un plano), Matlab permite hacer un ajuste polinómico de grado n de los mismos invocando a la función polyfit. Para que exista un error pequeño, por debajo de un 2%, se ha ajustado la función de calibrado a un polinomio de grado 4, obteniendo como resultado el expuesto en la Figura 4.3.

CALIBRADO	
TIEMPO	DISTANCIA
20737	0,948
20642	0,913
20448	0,841
20335	0,812
20211	0,78
20159	0,76
20020	0,732
19840	0,691
19697	0,66
19491	0,63
19317	0,597
19140	0,569
18931	0,54
18724	0,512

CALIBRADO	
TIEMPO	DISTANCIA
18518	0,49
18210	0,458
17987	0,438
17503	0,401
17089	0,372
16709	0,343
16117	0,312
15605	0,274
14741	0,245
13621	0,211
12437	0,187
9922	0,15
3210	0,09

Tabla 4.1. Tabla de calibrado. Relación tiempo-distancia



Función calibrado:
Distancia = 7.9722e-17 x^4 - 3.5033e-12 x^3 + 5.5194e-08 x^2 - 3.3940e-4x + 0.7190

Figura 4.3. Gráfica de Calibrado.

Capítulo 5.
Barrido láser 2D

Para culminar la tercera fase de desarrollo, y por tanto este TFG, se ha hecho uso de la potencia que nos ofrece ROS, cuyas siglas en inglés significan Robot Operating System. A pesar de su nombre, no se trata de un sistema operativo propiamente dicho (funciona sobre Linux), sino más bien de una infraestructura de desarrollo, despliegue y ejecución de sistemas robóticos. Esta plataforma de trabajo proporciona a los desarrolladores abstracción de hardware, controladores de dispositivos, bibliotecas, visualizadores, paso de mensajes, gestión de paquetes, etc. La principal característica de ROS frente a otros softwares especializados en robótica es que su código es abierto y se trabaja bajo licencia BSD. Posee una amplia documentación que permite hacer frente a diferentes tareas dentro de la robótica, como puede ser la visualización, reconocimiento de objetos, o como es nuestro caso, la navegación. Para instalarlo en el ordenador hemos seguido la guía que se adjunta en la bibliografía.

A continuación, para entender la estructura interna de esta potente herramienta, describimos algunos de los términos que emplea ROS.

5.1 Terminología de ROS

A nivel computacional ROS distingue entre los siguientes conceptos:

- **Nodo**

Un nodo es un archivo ejecutable dentro de un paquete ROS. Utilizando la librería cliente incluida en ROS, un nodo será capaz de comunicarse con otros nodos, por lo que su nombre debe ser único. Además un nodo podrá publicar o suscribirse a un tema (topic) o proporcionar un servicio. Un sistema robótico complejo estará dotado de varios nodos que son conectados de forma dinámica en tiempo de ejecución.

- **Mensaje**

Los nodos se comunican entre sí mediante la publicación de mensajes a los temas (topic). Un mensaje es una estructura de datos simple que admite campos cuyos tipos sean primitivos (enteros, punto flotante, booleanos, etc.), así como vectores o matrices de estos tipos de datos.

- **Tema (Topic)**

Los mensajes se envían a través de un sistema de transporte de publicación/suscripción semántica. Cuando un nodo envíe un mensaje será publicado en un determinado tema (topic), de igual modo, si el nodo estuviese interesado en recibir algún mensaje, estará suscrito al tema correspondiente. Existe la posibilidad de que tengamos varios editores y suscriptores a un mismo tema de manera concurrente, puesto que un nodo puede publicar o suscribirse a múltiples temas.

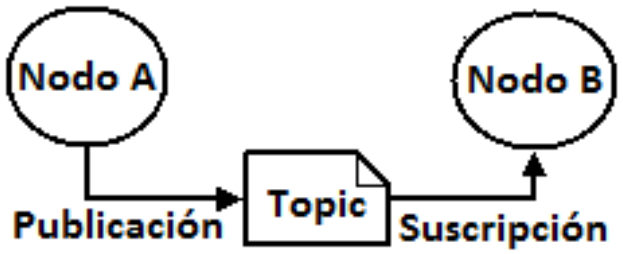


Figura 5.1. Conexión entre dos Nodos en ROS.

- **Servicio:**

Los servicios nos permiten una comunicación del tipo pregunta/respuesta, es decir, formada por dos mensajes (un mensaje pregunta y el otro responde). A diferencia de los topics, un nodo solamente podrá llamar a un servicio con un determinado nombre.

- **Master:**

Es el nodo principal de ROS, actúa de forma similar a un servidor DNS, pues su función se basa en proporcionar servicios de registro y búsqueda de nombres dentro del esquema de ejecución, es decir, realiza un seguimiento de los editores y suscriptores de los topics, así como de los servicios. Para que el resto de nodos de ROS pueda comunicarse se necesita activar el nodo Master

al inicio de la ejecución, para ello se lanza la instrucción **roscore** en una terminal, que tendrá como resultado el inicio del bucle de lectura de mensajes.

5.2 Desarrollo de un Barrido Láser 2D

Utilizando ROS, vamos a crear un barrido láser 2D, que nos permite visualizar lo que tenemos delante del plano de actuación.

Todos los nombres a los que se hace referencia a continuación han sido los utilizados en el código de implementación que se añade como Apéndice B. Teniendo en cuenta los conceptos descritos en el apartado 5.1, diremos que nuestro esquema interno de ROS sigue el representado en la Figura 5.2. En él vemos que se ha creado un nodo, de nombre serialScanner, que publica en un tópico que denominamos scan, a través de un publicador de nombre scan_pub. Los mensajes que el publicador emite, Lasermsgs, son del tipo LaserScan.

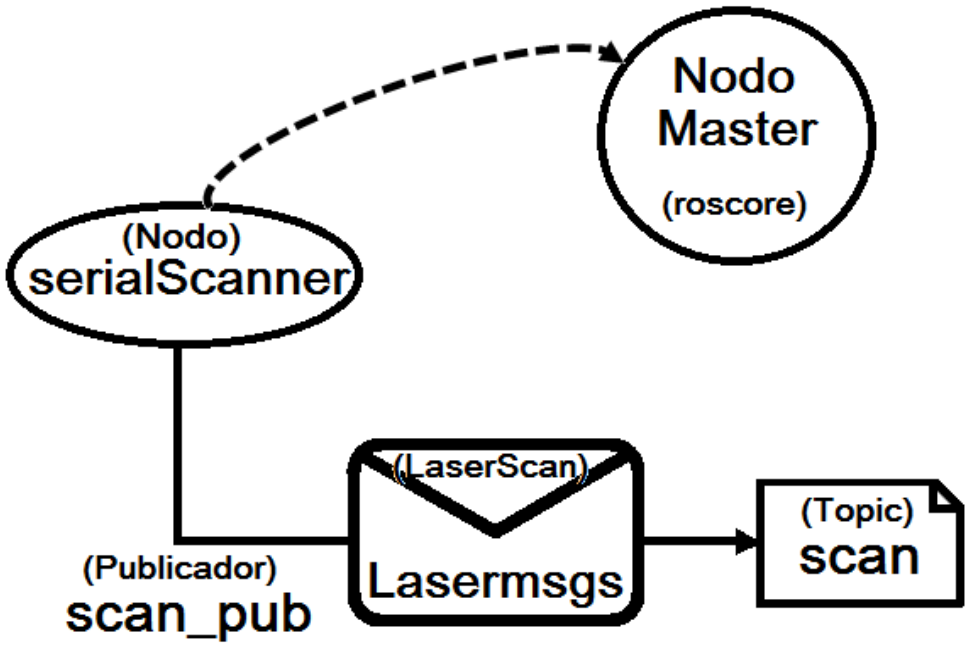


Figura 5.2. Esquema interno de ROS para un barrido láser 2D.

El tipo de mensaje emitido, LaserScan, requiere de una cabecera de configuración, en la que se indican las características (como por ejemplo, la distancia máxima y mínima, el incremento angular, el número de escaneos,

etc.) que ROS necesita conocer sobre el sensor láser de medida de distancias que se ha fabricado [Figura 5.3]

```
ros::Time scan_time;
//BARRIDO POSITIVO
//Tratamos de leer un mensaje
scan_time = ros::Time::now();
Lasermsgs.header.stamp = scan_time;
Lasermsgs.header.frame_id = "Scanner";
Lasermsgs.header.seq = count++;
//VARIABLES DE CONFIGURACION
Lasermsgs.range_min = 0.089;//9 cm
Lasermsgs.range_max = 100.0;//1 metro
Lasermsgs.angle_min = 0.0;
//Lasermsgs.angle_max = 3.1415926535;//M_PI
Lasermsgs.angle_max = M_PI;
Lasermsgs.angle_increment = M_PI/NUMSCANS;
//Lasermsgs.angle_increment = 3.1415926535/NUMSCANS;
Lasermsgs.time_increment = 1/NUMSCANS;
Lasermsgs.scan_time = 1;
Lasermsgs.ranges.resize(NUMSCANS);
```

Figura 5.3. Variables de configuración en la cabecera de un mensaje LaserScan [Apéndice B].

El mensaje contiene, además de la cabecera, un vector con las 100 medidas de tiempo proporcionadas por el sensor CCD, cada una de ellas asociadas a un grado al ejecutar el barrido de 180º. Cada medida de tiempo del sensor CCD deberá convertirse en distancia antes de ser almacenada en el vector, para ello se emplea la función de calibrado explicada en el Capítulo 4. Teniendo las 100 medidas de distancia y el ángulo correspondiente a cada una de ellas (2D) se publica el mensaje. Este proceso se ejecuta de forma cíclica y en ambos sentidos.

Para finalizar, haremos una representación visual de los datos obtenidos empleando rviz. Este programa es un visualizador adjunto a Ros que nos permite ver datos de cámaras, mediciones de distancias de infrarrojos, datos de sonar, etc. Ejecutando rviz podremos mostrar una representación en vivo de los valores del sensor y por tanto del barrido láser 2D. El resultado se presenta en la FIGURA 5.4.

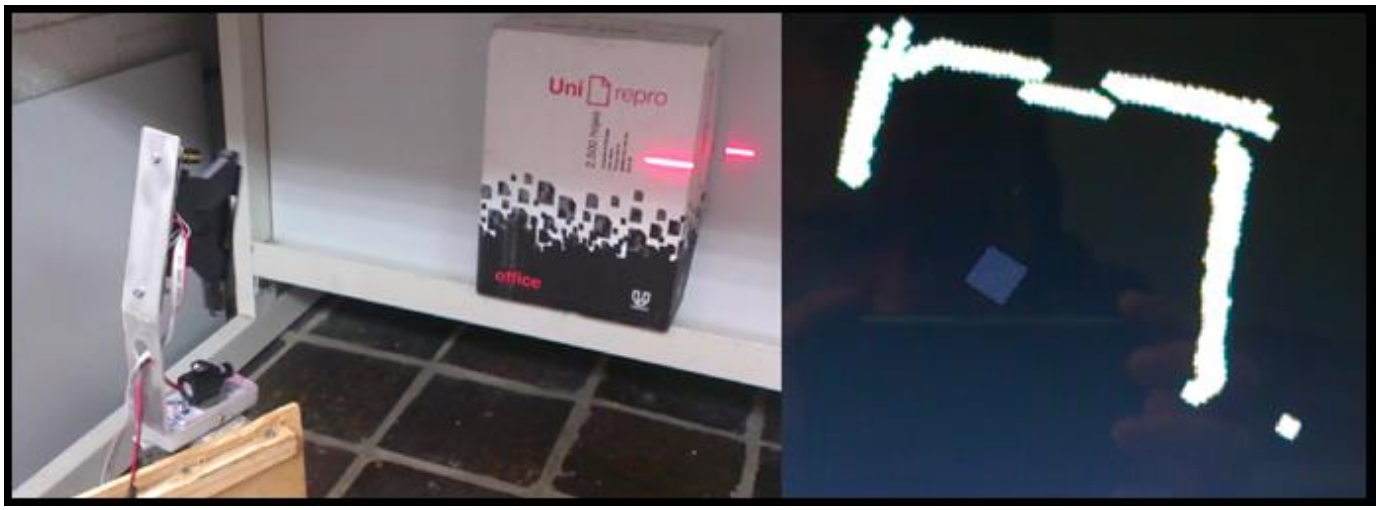


Figura 5.4. A la izquierda una foto del entorno. A la derecha el barrido láser 2D realizado en rviz.

Tras obtener el barrido láser 2D, se puede dar por finalizada la tercera y última fase de desarrollo del proyecto.

Capítulo 6. Presupuesto

En este capítulo intentaremos estimar un presupuesto que englobe el coste total para la construcción del sensor láser de medida de distancias que se ha fabricado. Dividiremos el presupuesto en coste en material y coste de personal.

6.1 Coste de materiales

Para recoger los cálculos en este apartado se descartan el uso de herramientas propias de un laboratorio especializado, como pueden ser los equipos informáticos, multímetro, osciloscopio, soldador, etc. En nuestro caso, el proyecto ha sido desarrollado en el laboratorio de Computadores y Control en la facultad de Física y Matemáticas de la Universidad de La Laguna.

Por otro lado, sin perder de vista que la finalidad es la construcción de un sensor láser de medida de distancias a bajo coste, se ha buscado realizar este TFG haciendo uso de plataformas de software libre, eligiendo Ubuntu 14.04 como Sistema Operativo donde se instalaron las dos plataformas de desarrollo (Arduino y ROS). Por lo que se contabiliza el coste en software a cero euros.

El coste en materiales generado para desarrollar el proyecto viene a cargo de los componentes empleados en su fabricación. Como se ha comentado en el documento, se han reciclado algunos de ellos. Para que el resultado del presupuesto del dispositivo sea real, y se ajuste al trabajo desempeñado, se ha descartado el valor de estos componentes (lente y motor PaP).

Componentes	Unidades	Precio (€)
Arduino Uno	1	15.25
Sensor CCD ILX511	1	4
Comparador	1	0.90
Lente	1	-
Láser	1	4.25
Motor Paso a Paso	1	-
Controlador L298N	1	4.6
Interruptor Óptico	1	1.25
Placa Perforada	1	2.75
TOTAL		**33**

Tabla 8.1. Presupuesto de materiales.

6.2 Coste en personal

Para estimar el coste de personal que requiere el desarrollo del sensor de distancias, no hemos colocado en el siguiente supuesto.

El TFG es un proyecto que debe realizarse de forma individual por el alumno. Para no incidir en temas de gestión empresarial, el perfil que más se ajusta para el alumno es el de un Ingeniero trabajando Freelance. Para este rol, se ha establecido un precio por hora de 12€. Para estimar esta cifra, se ha recurrido a un salario base español para una titulación de Ingeniero Informático, que ronda los 22.203€/anuales.

Por otro lado, hemos tenido en cuenta las horas concertadas con el tutor, que reflejaremos en el presupuesto como un servicio de consultoría profesional dentro de la rama de las Ingenierías. El salario anual para un perfil senior en una consultoría informática en España, está alrededor de 46700 €/anuales, que se traduce en 24€/hora.

Tarea	Horas	Coste (€)
Investigación de la Placa de Desarrollo (Arduino)	80	960
Investigación del Funcionamiento del sensor CCD	50	600
Investigación del Funcionamiento de Motor PaP	25	300
Investigación del funcionamiento de ROS	50	600
Desarrollo en la Plataforma Arduino	90	1080
Desarrollo en la Plataforma ROS	50	600
Diseño y construcción mecánico	35	420
Diseño electrónico	35	420
Documentación del Trabajo	90	1080
Consultoría Profesional	12	288
TOTAL	**517**	**6348**

Tabla 8.2. Presupuesto en personal.

6.3 Presupuesto total

El coste total del sistema desarrollado es de 6381€. Sin embargo podemos decir que se ha cumplido con el objetivo principal de fabricar un dispositivo de bajo coste, puesto que debemos tener en cuenta que el número de horas que se emplean para la investigación incrementan de forma notable el coste final del producto.

Categoría	Coste (€)
Coste de material	33
Coste en personal	6348
TOTAL	**6381**

Tabla 8.2. Presupuesto total.

Capítulo 7.
Conclusiones y líneas futuras

Como conclusión, se puede considerar alcanzado con éxito el objetivo propuesto. Este proyecto perseguía la fabricación de un prototipo, a bajo coste, de sensor de medidas de distancias utilizando el principio de triangulación, un sensor CCD lineal como receptor y un láser como fuente emisora. Empleando un motor paso a paso se ha conseguido calcular la distancia de varios puntos en el espacio dentro de un mismo plano, generando un barrido láser 2D.

Como todo prototipo es mejorable y en líneas futuras se trabajará para perfeccionar el sensor fabricado, tanto en el ámbito del software como del hardware empleado.

- **Posibles mejoras de software**
  Se puede optimizar tanto el software desarrollado en Arduino como en ROS. Trabajando con un sistema empotrado esta mejora es fundamental, pues podemos optimizar características como el consumo, la carga computacional, etc.

- **Posibles mejoras de Hardware y componentes**
  Es donde radica la mayor cantidad de mejoras. El dispositivo podrá perfeccionarse utilizando otros componentes, sin embargo no debemos olvidarnos que el prototipo fue construido de forma manual, y esto conlleva unas limitaciones.
  Cambiar la conexión del motor utilizando 2 pines en lugar de 4 es un dato de escalabilidad que ya se tuvo en cuenta a la hora de desarrollar el proyecto. Además se puede mejorar entre otras cualidades, la velocidad de escaneo, sustituyendo el motor paso a paso por un servo, aunque sería necesario cambiar el modelo de placa de Arduino.
  Por otro lado, si se encuentra la manera de calibrar el láser de forma automática, sería interesante modificar el ángulo y la distancia entre el

sensor CCD y el láser una vez se haya iniciado, lo que nos permite en tiempo de ejecución, actuar en un rango de operatividad distinto.

- **Probar otras funcionalidades**
  También hubiese sido interesante enfocar el prototipo a una funcionalidad distinta a la navegación, como por ejemplo un escáner 3D. El amplio alcance que ofrece un proyecto de este ámbito y a la creciente expansión que ha sufrido la robótica en los últimos años, hace posible experimentar y rescatar sensores "antiguos" para asignarles nuevas aplicaciones.

En el ámbito personal, la realización de este trabajo ha supuesto enfrentarse a diferentes ramas de conocimiento de ingeniería. Con cada paso logrado, la finalización de este TFG, más que un reto, se ha convertido en un conjunto de buenas experiencias de las que he adquirido los conocimientos necesarios para finalizar el proyecto.

Capítulo 8.
Summary and Conclusions

As a conclusion, we can state that the proposed objective has been reached successfully. This project pretended to achieve and produce the initial prototype, at reduced cost, of a laser distance measurements sensor using the triangulation methodology, a lineal CCD sensor as a receiver and a laser as an emitting source. Taking advantage of stepper we have achieved to figure different measures out within the same level, and it generates a 2D laser scanning.

Having in mind that there is room for improvement and, eventually, we will work to perfect the sensor, not only in the software field but also in the hardware we used to produce it.

- **Possible Software improvements**
  It can be optimized both the developed software in Arduino and in ROS. Working with an embedded system, this improvement is essential because we can improve diverse features such as the consumption, the computing workload, etc.

- **Possible Hardware and components improvements**
  Here we can find the greater amount of improvements. The device could be perfected using other components, although we cannot forget that this prototype was constructed by hand and this, obviously, involves limitations.
  Change the stepper connection using 2 pins instead of 4 is an important scalability piece of information that we took into account when we were developing the project. Besides this, we can improve, among others attributes, the scanning speed, replacing the stepper by a servo, although here it is necessary to change the Arduino motherboard model.
  Apart from that, if we find the way to calibrate the laser automatically, it would be very interesting to modify the angle and the distance among

the CCD sensor and the laser once this has started; this fact allows us to operate in a different effectiveness range.

- **Try another usefulness**
  It would have been interesting to focus on the prototype a different usage from navigation, for instance, a 3D scanner. This project offers a wide-ranging in this field and, also, in the increasing expansion that the robotics have suffered during the last years; all these facts make possible to experience and save "old" sensors to assign new applications.

Personally speaking, the realization of this work has meant to face myself to different areas within the engineering field of knowledge. Every little step I have achieved, with the TFG completion and so on, this work has become in more than a challenge, in a good experiences ensemble in which I do have acquired the necessary knowledge to finish my project.

Apéndice A.
Algoritmo ARDUINO

A.1. Algoritmo: Lectura de Sensor CCD y barrido de 180º en un motor PaP

```
/*********************************************************************************
  * AUTORES: Jonay Tomas Toledo Carrillo
  *          Carlos Barreda Falciano
  * FECHA: 8/07/2015
  * DESCRIPCION: Código implementado en la plataforma de desarrollo de Arduino, cuya
  * función es la lectura de un sensor CCD (SONY ILX511) y la ejecución de un barrido
  * de 180 grados utilizando un motor paso a paso.
*********************************************************************************/
```

```
1  //Librerias
2  #include "TimerOne.h"
3  #include <Stepper.h>
4
5  #define STEP_TIME 1 //Tiempo entre paso y paso
6  #define rog 12 //Integracion del sensor CCD ROG al pin 12 Arduino
7  #define clk 13 //Reloj del sensor CCD en el ping 13 de Arduino.
8  #define inputCapture 8 //VOUT del CCD en el pin 8 Arduino
9  #define STEPS 200 // Se define el numero de pasos del motor
10
11 //Librerias para asignar o limpiar bits.
12 #ifndef cbi
13 #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
14 #endif
15 #ifndef sbi
16 #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
17 #endif
18 /*Cada declaración 'nop' se ejecuta en un ciclo de
19 máquina (a 16 MHz), dando un retraso de 62,5 nanosegundos*/
20 #define DELAY()   __asm__("nop\n\t")
21
22 //Declaracion de variables del Timer 1
23 volatile word period = 0; //Cuenta el tiempo
24 volatile word lastPeriod = 0;
25 volatile word count = 0;//Cuenta Flancos de bajada en una lectura
26 int i=0; // Para recorrer el for del barrido
27 /* Crea una instancia de la clase stepper, especificando el
28 numero de pasos del motor, y los pines asignados.
29 Stepper stepper(STEPS, 16, 17, 18, 19);
30 //INTERRUPTOR OPTICO
31 int analogPin = 15; //Pin 15 para Interruptor Optico
32 int valAnalog = 0;//Valor del Interruptor Optico a cero
33 //Cuenta el tiempo utilizado en la funcion millis().
```

```
34  unsigned long currentMillis;
35  unsigned long currentMillis2;
36
37  //Subrutina que ejecuta el Timer1
38  ISR (TIMER1_CAPT_vect) //Timer1 capture vector
39  {
40     cli();// deactivate other interrupts
41     if (period == 0)
42       period = ICR1; //copy Time Stamp
43     ++count;// flancos de bajada producidos en una lectura.
44     //TCNT1 = 0x0000;  //reset Timer1 counter.
45     // sei();// reactivate other interrupts
46  }
47
48  // The setup routine runs once when you press reset:
49  void setup() {
50    //CONFIGURACION DEL TIMER
51    //Asigna los bits de configuracion delprescaler 100 para Timer1
52    cbi(TCCR1B,CS12);
53    cbi(TCCR1B,CS11);
54    sbi(TCCR1B,CS10);
55    //Configuracion Timer1 En modo Capture
56    cbi(TCCR1B,WGM13);//"waveform generation mode bit description"
57    cbi(TCCR1B,WGM12); // normal counter mode
58    cbi(TCCR1A,WGM11);
59    cbi(TCCR1A,WGM10);
60
61    sbi(TCCR1B,ICES1);//Configuración del Input Capture
62    sbi(TCCR1B,ICNC1);  //Se activa la cancelación de ruido
63    //Se configura el pin8 de arduino como entrada
64    pinMode(inputCapture, INPUT); //Salida del sensor ILX511)
65
66    Serial.begin(115200);
67
68    sbi(TIMSK1,ICIE1);  // Input Capture interrupt enable
69    cbi(PRR,PRTIM1);    //Se activa el Timer/Counter1
70    sei();              //Activate global interrupts
71
72    //Se activa el Pin 12 y Pin 13 del aruidno a 1
73    DDRB = B00110000; //pinMode(clk,OUTPUT); pinMode(rog,OUTPUT);
74    PORTB = B00110000;
75    // Se asigna la velocidad del motor a 30 RPMs
76    stepper.setSpeed(30);
77    currentMillis = millis();
78    currentMillis2 = millis();
79  }
80  // the loop routine runs over and over again forever:
81  void loop() {
82    //LLamada a la funcin que detecta la HomePosition
83    homePosition();
84    //MUESTREO DEL TIMER
85    /*Serial.print("period= ");
86    Serial.print(period);
87    Serial.print("\tTCNT1= ");
88    Serial.print(TCNT1);
89    Serial.print("\tcount= ");
90    Serial.println(count);*/
91  }
92
93  /*DETECTA LA HOME POSITION LEYENDO EL VALOR DEL INTERRUPTOR
94 OPTICO UNA VEZ DETECTADA LLAMA A LA FUNCIÓN BARRIDO*/
95  void homePosition(){
```

```
 96    //Se lee el valor del Interruptor Optico
 97    valAnalog = analogRead(analogPin);
 98    //Si detecta la Home position, comienza el barrido.
 99    if(valAnalog = analogRead(analogPin) < 600){
100      barrido();
101    }
102    /*Se comprueba el valor del sensor, si no ha detectado la HP se 103     da
otro paso y se vuelve a comprobar*/
104    if(millis()-currentMillis>10){// Tiempo entre paso y paso
105      if(valAnalog = analogRead(analogPin) > 600){
106        stepper.step(-1);//Mueve el motor un paso
107         currentMillis=millis();
108      }
109    }
110  }
111
112  //LECTURA DEL SENSOR CCD ILX511
113  void lecturaSensor(){
114    //CONFIGURACION DEL PULSO INICIAL
115    PORTB = B00110000;//HIGH clk (5) y HIGH Rog(4);
116    delayMicroseconds(1);
117    //digitalWrite(clk, LOW);
118    PORTB = B00010000;//LOW clk (5) y HIGH Rog(4);
119    delayMicroseconds(1);
120    //digitalWrite(clk, HIGH);
121    PORTB = B00110000;//HIGH clk (5) y HIGH Rog(4);
122    delayMicroseconds(3);
123    //digitalWrite(rog,LOW);
124    PORTB = B00100000;//HIGH clk (5) y LOW Rog(4);
125    delayMicroseconds(10);
126    //digitalWrite(rog,HIGH);
127    PORTB = B00110000;//HIGH clk (5) y HIGH Rog(4);
128    delayMicroseconds(3);
129    //REALIZAMOS EL CICLO DE LECTURA
130    for(int i=0;i<2100;i++){
131     // digitalWrite(clk,LOW);
132      PORTB = 0 << 5 | 1 <<4 ;
133      //DELAY();
134      __asm__("nop\n\t");
135      __asm__("nop\n\t");
136      __asm__("nop\n\t");
137      __asm__("nop\n\t");
138      // digitalWrite(clk, HIGH);
139      PORTB = 1<<5 | 1 << 4;
140      //DELAY
141      __asm__("nop\n\t");
142      __asm__("nop\n\t");
143      __asm__("nop\n\t");
144      __asm__("nop\n\t");
145    }
146  }
147
148  // EFECTUA UN BARRIDO DE 180 GRADOS
149  void barrido(){
150    //Serial.println("En Barrido");
151    float grado=0.0;
152    while(1){
153        //BARRIDO POSITIVO
154         grado=0.0;
155         currentMillis2=millis();
156         for(i=0;i<100;i++){
157             //Inicializacin del Timer1 y llamada a lecturaSensor()
```

```
158               TCNT1 = 0x0000;
159               period = 0;
160               count = 0;
161               lecturaSensor();
162               //Muestra el valor del TIMER
163               Serial.print(period);
164               Serial.print(" ");
165               //Se muestra el valor de la variable grado
166               Serial.println(grado);
167               while(millis()-currentMillis2<STEP_TIME);
168                 stepper.step(1); //Se avanza un paso
169                 grado=grado+1.8; //Se incrementa los grados
170                 currentMillis2=millis();//Tiempo entre paso y paso
171           }
172           //BARRIDO NEGATIVO
173           grado=0.0;
174           currentMillis2=millis();
175           for(i=0;i<100;i++){
176             //Inicializacin del Timer1  y llamada a lecturaSensor()
177               TCNT1 = 0x0000;
178               period = 0;
179               count = 0;
180               lecturaSensor();
181               //MUESTREO DEL TIMER
182               Serial.print(period);
183               Serial.print(" ");
184               //Se muestra el valor de la variable grado
185               Serial.println(grado);
186
187                 while(millis()-currentMillis2<STEP_TIME);
188                 stepper.step(-1); //Se avanza un paso
189                 grado= grado+1.8; //Se incrementa los grados
190                 currentMillis2=millis();//Tiempo entre paso y paso
191           }
192       }
193   }
*******************************************************************************/
```

Apéndice B. Algoritmo ROS

B.1. Barrido Láser 2D

```
/*********************************************************************************
   * AUTORES: Jonay Tomas Toledo Carrillo
   *          Carlos Barreda Falciano
   * FECHA: 8/07/2015
   * DESCRIPCION: El Desarrollo implementado en ROS se divide en tres ficheros.
   * La función es generar el sistema de comunicación descrito en el capítlo 5 que le
   * permita a ROS ejecutar un barrido láser 2D.
*********************************************************************************
   * FICHERO 1: Serial_conexion.h
*********************************************************************************
```

```
 1  #ifndef _ODOMETRY_CONEXION_H_
 2  #define _ODOMETRY_CONEXION_H_
 3  #include <cmath>
 4  #include <ros/ros.h>
 5  #include <realtime_tools/realtime_publisher.h>
 6  #include <sensor_msgs/Range.h>
 7  #include <tf/tf.h>
 8  #include <tf/tfMessage.h>
 9  #include <boost/asio.hpp>
10  #include <boost/asio/serial_port.hpp>
11  #include <sensor_msgs/LaserScan.h>
12
13  #define PI  3.14159265358979
14  #define RAD(x)  ((x)*PI/180)
15  #define DEG(x)  ((x)*180/PI)
16  #define TAM_MENSAJE 4
17
18
19  class sharp_conexion
20  {
21   public:
22      sharp_conexion(std::string port, unsigned int baud_rate);
23       ~sharp_conexion() {};
24  std::string readLine();
25      bool init(ros::NodeHandle& rnh);
26      /** Ejecuta el bulce de obtenciÃ³n de mensajes hasta que ros lo mande
parar */
27      void spin();
28
29   private:
30      // ================================================================
31      // Variables relativas a la comunicacion serial
32      // ================================================================
```

```
33      /** Descriptor de archivo de la serial*/
34      boost::asio::io_service m_io;
35      // ============================================================
36      // Variables de ROS
37      // ============================================================
38      sensor_msgs::LaserScan Lasermsgs; //MENSAJE (BARRIDO POSITIVO)
39      sensor_msgs::LaserScan Lasermsgs2; //MENSAJE (BARRIDO NEGATIVO)
40      ros::NodeHandle n;
41      ros::Publisher scan_pub;
42      /** Identificador del frame odom */
43      std::string odom_frame_id_;
44      /** Identificador del frame de base_link */
45      std::string base_link_frame_id_;
46      ros::Time current_time_, last_time_;
47      /** Publica mensaje de transformaciÃ³n */
48      void publishTransform();
49      boost::asio::io_service io;
50      boost::asio::serial_port serial;
51  };
52  #endif /* ifndef _ODOMETRY_CONEXION_H_ */
```

****

*** FICHERO 2:** SerialScanner.cpp

****

```
1   #include <ros/ros.h>
2   #include <serialScanner/Serial_conexion.h>
3   #include <stdlib.h>
4   #include <stdio.h>
5   //Asignación del Puerto Serie al que está conectado Arduino
6   std::string Port("/dev/ttyACM0");
7   //Misma velocidad en baudios para la transmisión de datos
8   int Baud = 115200;
9   //Nombre del Nodo Creado
10  std::string nodeName("serialScanner");
11  /* Main del programa que arranca el nodo */
12  int main(int argc, char **argv)
13  {
14      ros::init(argc, argv, nodeName.c_str());
15      ros::NodeHandle rnh;
16      ROS_INFO("Comienza  Scanner");
17
18      if (rnh.getParam((nodeName  + "/serial_port").c_str(), Port)){
19        ROS_INFO("Got param: %s", Port.c_str());
20      }
21
22      if (rnh.getParam((nodeName  + "/serial_baud").c_str(), Baud)){
23        ROS_INFO("Got param: %s", Port.c_str());
24      }
25      ROS_INFO("Setting port: %s Bauds %d", Port.c_str(), Baud);
26      sharp_conexion oc(Port, Baud);
27
28      if(oc.init(rnh))
29          oc.spin();
30
31      ROS_INFO("Termina Scanner");
32      return 0;
33  }
```

****

* **FICHERO 3:** Serial_conexion.cpp

****

```cpp
#include <serialScanner/Serial_conexion.h>
#include <stdlib.h>
#include <stdio.h>
//Se incluye math para trabajar con la funcion de calibrado
#include <math.h>
#include <boost/asio.hpp>

#define PI   3.14159265358979
#define RAD(x)   ((x)*PI/180)
#define DEG(x)   ((x)*180/PI)
/*Se define el numero de Escaneos.
//Para el barrido de 180 grados se dan 100 pasos
#define NUMSCANS   100 //Por tanto Se tomaran 100 medidas

//Se abre el puerto Serie
 sharp_conexion::sharp_conexion(std::string port, unsigned int  baud_rate):
 io(), serial(io,port){
        serial.set_option(boost::asio::serial_port_base::
        baud_rate(baud_rate));
        ROS_INFO("Puerto Serial abierto");
};
//Se lee del puerto serial linea a linea
std::string sharp_conexion::readLine(){
    //Lectura caracter a caracter hasta encontrar '\r' o '\n'
    using namespace boost;
    char c;
    std::string temp;
    std::string result;
    for(;;)
    {
        asio::read(serial,asio::buffer(&c,1));
        temp = c;
        printf("%c",c);
        switch(c)
        {
            case '\r':
                break;
            case '\n':
                return result;
            default:
                result+=c;
        }
    }
}

//
bool  sharp_conexion::init(ros::NodeHandle& node)
{
    scan_pub = n.advertise<sensor_msgs::LaserScan>("scan", 20);
    ros::Rate r(1.0);
    return true;
}

//Ejecuta el bulce de obtención de mensajes
void sharp_conexion::spin() {
    std::string result;
    //angulo: se obtiene del paso del motor
    //tiempo: del sensor CCD
```

```
58      // Se calcula con la funcion calibrado
59      double angulo, tiempo, distancia;
60      int numMedidas[NUMSCANS];
61      double medidas[NUMSCANS];
62      double vecAngulos[NUMSCANS];
63
64      int count = 1;
65      int pos;
66      int i=0;
67      ROS_INFO(" Comienza en bucle");
68      //Bucle principal
69      while(n.ok()) {
70          ros::Time scan_time;
71          //BARRIDO POSITIVO
72          //Tratamos de leer un mensaje
73          scan_time = ros::Time::now();
74          Lasermsgs.header.stamp = scan_time;
75          Lasermsgs.header.frame_id = "Scanner";
76          Lasermsgs.header.seq = count++;
77          //VARIABLES DE CONFIGURACION
78          Lasermsgs.range_min = 0.089;//9 cm
79          Lasermsgs.range_max = 100.0;//1 metro
80          Lasermsgs.angle_min = 0.0;
81          //Lasermsgs.angle_max = 3.1415926535;//M_PI
82          Lasermsgs.angle_max = M_PI;
83          Lasermsgs.angle_increment = M_PI/NUMSCANS;
84          //Lasermsgs.angle_increment = 3.1415926535/NUMSCANS;
85          Lasermsgs.time_increment = 1/NUMSCANS;
86          Lasermsgs.scan_time = 1;
87          Lasermsgs.ranges.resize(NUMSCANS);
88          // Se almacena en el vector el angulo y la distancia
89          for (int i=0; i < NUMSCANS; i++) {
90              medidas[i] = 0;
91              numMedidas[i] = 0;
92          }
93          /*Cada linea contiene el tiempo y el grado asociado,
94          ambos datos vienen separados por un espacio.*/
95          for (i=0;i<NUMSCANS;i++) {
96              result=readLine();
97              pos = result.find_first_of(' ');
98              tiempo = atof(result.substr(0,pos).c_str());
99       angulo = atof(result.substr(pos+1,result.length()).c_str());
100             //Control de ángulo negativo
101             if ((angulo <= -1) )
102                 break;
103 /*FUNCION POLINOMICA DE GRADO 4: FUNCION CALIBRADO QUE RELACIONA
104 TIEMPO CON DISTANCIA, //Donde x es el tiempo */
105         //7.9722e-17 x^4 - 3.5033e-12 x^3 +
106              5.5194e-08 x^2 - 3.3940e-04 x + 0.7190
107             distancia = ((7.9722*pow(10,-17)*(pow(tiempo,4)))-
108                 (3.5033*pow(10,-12)*(pow(tiempo,3)))+
109                     (5.5194*pow(10,-8)*(pow(tiempo,2)))-
110                         (3.3940*pow(10,-4)*tiempo)+0.7190);
111             if (tiempo==0){
112                 Lasermsgs.ranges[i] = Lasermsgs.range_max +1;
113             }
114             else{
115                 Lasermsgs.ranges[i] = distancia*100;
116             }
117         }
118         //ROS_INFO("Barrido Positivo Enviado");
119         //Para que ROS procese los mensajes
```

```
120           scan_pub.publish(Lasermsgs);
121           ROS_INFO("Barrido Positivo Enviado");
122           Lasermsgs.ranges.clear();
123 
124           //BARRIDO NEGATIVO
125           scan_time = ros::Time::now();
126           Lasermsgs2.header.stamp = scan_time;
127           Lasermsgs2.header.frame_id = "Scanner";
128           Lasermsgs2.header.seq = count++;
129 
130           Lasermsgs2.range_min = 0.089;//9 cm
131           Lasermsgs2.range_max = 100.0;//1 metro
132           Lasermsgs2.angle_min = 0.0;
133          //Lasermsgs2.angle_max = 3.1415926535;//M_PI
134           Lasermsgs2.angle_max = M_PI;
135           Lasermsgs2.angle_increment = M_PI/NUMSCANS;
136           //Lasermsgs2.angle_increment = 3.1415926535/NUMSCANS;
137           Lasermsgs2.time_increment = 1/NUMSCANS;
138           Lasermsgs2.scan_time = 1;
139           Lasermsgs2.ranges.resize(NUMSCANS);
140 
141           for (i=0; i<NUMSCANS; i++) {
142               medidas[i] = 0;
143               numMedidas[i] = 0;
144           }
145           for (i=NUMSCANS; i>0; i--) {
146               result=readLine();
147               pos = result.find_first_of(' ');
148               tiempo = atof(result.substr(0,pos).c_str());
149       angulo = atof(result.substr(pos+1,result.length()).c_str());
150               if ((angulo <= -1) )
151                   break;
152               distancia = ((7.9722*pow(10,-17)*(pow(tiempo,4)))-
153                    (3.5033*pow(10,-12)*(pow(tiempo,3)))+
154                        (5.5194*pow(10,-8)*(pow(tiempo,2)))-
155                            (3.3940*pow(10,-4)*tiempo)+0.7190);
156               if (tiempo==0){
157               //se descartan las distancias superiores a un metro
158                   Lasermsgs2.ranges[i] = Lasermsgs2.range_max +1;
159               }
160               else{
161                   Lasermsgs2.ranges[i] = distancia*100;
162               }
163           }
164           //Para que ROS procese los mensajes
165           scan_pub.publish(Lasermsgs2);
166           ROS_INFO("Barrido Negativo Enviado");
167           Lasermsgs2.ranges.clear();
168           ros::spinOnce();
169       }//Cierre Bucle principal
170   }
```

**********/

Bibliografía

[1] **Antecedentes sensores láser.**

Enlace: http://www.sensing.es/Sensores_de_distancia_laser_Cm.htm

[2] **Página oficial Arduino.**

Enlace: https://www.arduino.cc/

[3] **Datasheet ATmega328.**

Enlace:

http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf

[4] **Tutorial Timer1 de Arduino.**

Enlace: http://www.netzek.com/2013/12/atmega16-timer1.html

[5] **Funcionamiento sensor CCD.**

Enlace:
http://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0CDkQFjAE&url=http%3A%2F%2Fimartin.webs.ull.es%2Ftee%2FCCD.doc&ei=gUyEVZ-aOcHiU47stsAC&usg=AFQjCNFcIm29SSmuJ9OV4ARDJz68DkvGUw&sig2=_2rCU1BIC-_Cdc4uiBmd1g&bvm=bv.96042044,d.d24

[6] **Datasheet sensor SONY ILX511.**

Enlace:

http://www.alldatasheet.es/datasheet-pdf/pdf/47492/SONY/ILX511.html

[7] **Tutorial Motor PaP.**

Enlace:

http://www.todorobot.com.ar/tutorial-sobre-motores-paso-a-paso-stepper-motors/

[8] **Tutorial Controlador L298N.**

Enlace: http://electronilab.co/tutoriales/tutorial-de-uso-driver-dual-l298n-para-motores-dc-y-paso-a-paso-con-arduino/

[9] **Datasheet Interruptor Óptico.**

Enlace:

http://es.rs-online.com/web/p/interruptores-opticos-ranurados/2192561/

[10] **Página oficial ROS.**

Enlace: http://www.ros.org/

[11] **Instalación ROS.**

Enlace: http://wiki.ros.org/indigo/Installation/Ubuntu