UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Control de navegación con Arduino de un modelo de barco

Proyecto Final de Carrera

Ingeniería Técnica Informática: Especialización en Sistemas

Autor: Joan Francesc de León Botella

Director: Jose Vicente Busquets Mataix

Fecha: 12 de Septiembre del 2013

Resumen

En esta memoria se detalla el proceso de desarrollo de un control de navegación para una pequeña maqueta de barco radiocontrol y dotarla así de cierta autonomía. El proyecto se ha realizado utilizando como pilar básico la plataforma de hardware libre Arduino, junto con una serie de dispositivos necesarios para el funcionamiento del sistema.

El proyecto trata aspectos vistos a lo largo de toda la intensificación de informática industrial como por ejemplo qué es y cómo funciona un microcontrolador, la comunicación entre PC y Arduino o el uso de sensores y actuadores.

Palabras clave: Arduino, hardware, navegación autónoma, gps, imu, compás digital.

Tabla de contenidos

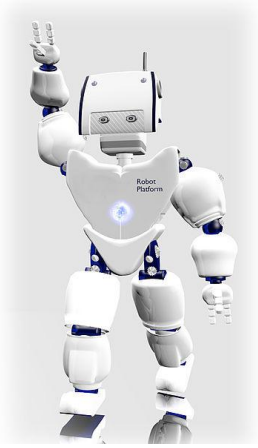
1. Introducción

1.1 Contextualización

A partir de las primeras revoluciones industriales, las máquinas han seguido una evolución que podríamos caracterizar casi como exponencial si la comparamos con el ritmo evolutivo del hombre a lo largo de la historia. En apenas dos siglos, las máquinas han pasado a estar presentes en la inmensa mayoría de los aspectos de nuestra vida: trabajo, hogar, lugares públicos, ocio…

Es tanto así, que podemos afirmar que no es simplemente que las máquinas estén presentes en nuestra vida cotidiana, sino que son parte vital de ella.

Una de las ramas de la tecnología que ha sufrido mayor evolución y que concentra todas las disciplinas de la industria juntas es la *robótica*. En ésta, se combinan distintas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física.



Desde los primeros robots que utilizaban tarjetas perforadas (1801, Joseph Jacquard inventa una máquina textil programable mediante dichas tarjetas), hasta los robots humanoides inteligentes de hoy en día, vemos que se ha producido un salto de gigante. Tanto es así, que según su cronología y avance podemos detectar cinco generaciones distintas de robots:

- ***1.- Robots Play-back:*** los cuales realizan una secuencia de instrucciones grabadas, como un robot utilizado en recubrimiento por spray o soldadura por arco. Estos robots comúnmente tienen un control de lazo abierto.

- ***2.- Robots controlados por sensores:*** estos tienen un control en lazo cerrado de movimientos manipulados, y toman decisiones basados en datos obtenidos por sensores.
- ***3.- Robots controlados por visión:*** donde los robots pueden manipular un objeto al utilizar información desde un sistema de visión.
- ***4.- Robots controlados adaptablemente:*** donde los robots pueden automáticamente reprogramar sus acciones sobre la base de los datos obtenidos por los sensores.
- ***5.- Robots con inteligencia artificial:*** donde los robots utilizan las técnicas de inteligencia artificial para hacer sus propias decisiones y resolver problemas.

Así pues, hoy en día existen muchísimos tipos de robots y muchos criterios a la hora de clasificarlos: por medio, por tamaño y peso, por función, grado de autonomía, etc. En este caso, nos centraremos en desarrollar un poco más los robots acuáticos.

Por un lado, si pensamos en robot acuático como robot submarino, hubo dos hechos que marcaron el interés por el desarrollo de esta clase de robots. El primero fue el accidente de un avión en el océano Atlántico, cerca de la costa de Irlanda, donde gracias a un robot submarino guiado remotamente, pudieron recuperar la caja negra del avión. El segundo, fue el descubrimiento del Titanic (hundido en 1912 a causa del choque con un iceberg) en un cañón a 4km por debajo de la superficie del agua; descubrimiento para el cual emplearon un robot submarino para explorarlo y filmarlo.

Hoy en día, su misión se centra sobre todo en exploración del mundo marino, y mantenimiento e inspección de tuberías que transportan petróleo en las grandes plantas petrolíferas.

Y por otro lado, tratando los robots que trabajan al nivel de la superficie del agua, su misión principal suele ser el de limpieza y recogida de basura de las costas, misiones de salvamento, pesca, etc.

Sin embargo, hay un proyecto que podríamos calificar como el desarrollo de éste elevado al máximo nivel: el Roboat.

El proyecto surge tras los desastres que tuvieron lugar como el tsunami que arrasó la costa de Japón, el vertido de petróleo en el Golfo de México o la actividad pirata en las costas de Adén (ciudad portuaria de Yemen). Estos y muchos más problemas acentuaron la idea de que sería conveniente tener una serie de "vigilantes del mar". Barcos autónomos capaces de reaccionar rápidamente y colaborar en cualquier tarea de ayuda marina. Sus aplicaciones son:

- Boyas inteligentes: dado que el barco puede conseguir energía por sí mismo, puede estar recopilando información continuamente de lagos, mares y océanos. Esto haría que fuese mucho más rentable la recopilación de información tanto del estado del lago, mar u océano, como del estado de los bancos de pesca, etc.
- Pruebas de seguridad: cualquier medida de seguridad implementada y probada en el Roboat puede ser puesta también en cualquier otro tipo de barco de vela. Además, puede interferir en situaciones peligrosas y ayudar al capitán a recuperar rápidamente el control del barco.
- Transporte ecológico de bienes materiales: dado que hoy en día el combustible es un recurso que debemos dosificar, no sólo por la cantidad de la que disponemos y su precio, sino también por su efecto en la atmósfera. Así pues, los veleros son barcos ecológicos, pero requieren de mucho personal humano. Y ahí es donde entra Roboat y su carácter autónomo.
- Reconocimiento y vigilancia: el Roboat puede ser enviado a zonas peligrosas. Gracias a su sigilo, que no posee tripulación y su fuente continua de energía puede ser usado en el seguimiento y vigilancia de barcos conflictivos.
- Fuente de datos: la grabación de un sin número de valores de medición por segundo hace que sea posible analizar por retrospectiva todas las maniobras del velero.
- Piloto secundario: en situaciones puntuales, se puede ceder el control de navegación al Roboat. Esta medida viene dada para gente que posea alguna minusvalía, si el capitán está herido o enfermo o para barcos con el personal reducido.
- Fuente de suministros: zonas donde no hay medios económicos, el Roboat puede ser una fuente importante a la hora de llevar tanto alimentos como medicamentos y otros productos de primera necesidad.

1.2 Motivación

A nivel personal, como estudiante de la intensificación de Informática Industrial, una de las cosas que me animó a elegir dicha rama era la posibilidad de aprender a trabajar con sistemas robotizados; ya que más allá de los propios robots, el hecho de poder ver reflejado mi trabajo en un medio físico (véase el comportamiento y/o movimiento de un sistema móvil o un brazo robot) es algo que como estudiante me producía una gran satisfacción personal.

Este proyecto es un buen ejemplo de ello.

¿Por qué este proyecto y no otro? Porque analizando las otras propuestas similares a ésta; realizarlo con un coche sería como la continuación del desarrollo de un pequeño robot móvil ya hecho en una asignatura anterior (es decir, sería en parte algo repetitivo); y otras opciones como el avión o un submarino pueden conllevar una enorme cantidad de riesgos y dificultades en la fase de pruebas, que fácilmente podrían complicar todo el proyecto.

Destacar también otro punto muy importante frente a las propuestas del avión y el submarino: y es que un barco no necesita un sistema de estabilización (puesto que tiene menos grados de libertad al moverse únicamente por la superficie del agua), lo que elimina una gran complicación del proyecto.

Aparte, vistas las distintas utilidades que tiene el Roboat, realizar algo a una pequeñísima escala de lo que es ese gran proyecto, supone también una motivación personal extra al saber que estoy realizando algo que realmente puede ser útil.

1.3 Objetivos

La meta principal del proyecto viene ya definida de por si en el título: diseñar y desarrollar un control de navegación para un modelo de barco radiocontrol.

A partir de ahí, y sabiendo los temas que se tratarán durante la realización de éste, podemos extraer objetivos secundarios:

- Investigar y comprender el funcionamiento de distintos dispositivos con los que puede trabajar Arduino.
- El estudio de la comunicación entre PC y el sistema.
- La dotación de autonomía al sistema.
- La consecución del mayor nivel de perfección posible a la hora de que el barco recorra la ruta. Esto hace referencia sobre todo a que sea capaz de recorrerla lo más rectilíneamente posible y con el menor número de desviaciones posible.

1.4 Método de desarrollo utilizado

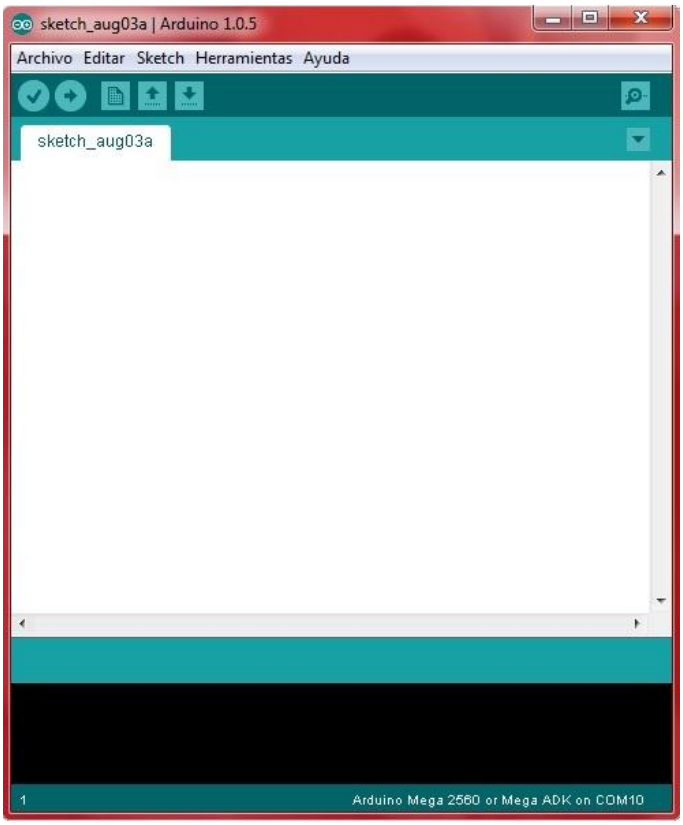
Para el desarrollo y evolución del proyecto se ha utilizado el método de *desarrollo iterativo e incremental*, ya que desde el principio, se ha ido realizando todo el proyecto en partes. Principalmente, cada parte representa uno de los dispositivos con los que Arduino debía trabajar o el tratamiento de ciertos comportamientos como el cálculo de distancias, rumbo, etc.

Así, conforme se iba completando cada una de las etapas, el resultado se iba acoplando al código principal; que a su vez iba cambiando y creciendo en consecuencia.

1.5 Entorno de desarrollo y software empleado

Para el desarrollo del software, se ha empleado el lenguaje de Arduino, basado en Wiring e implementado en C/C++, y cuyo entorno de programación se basa en Processing.

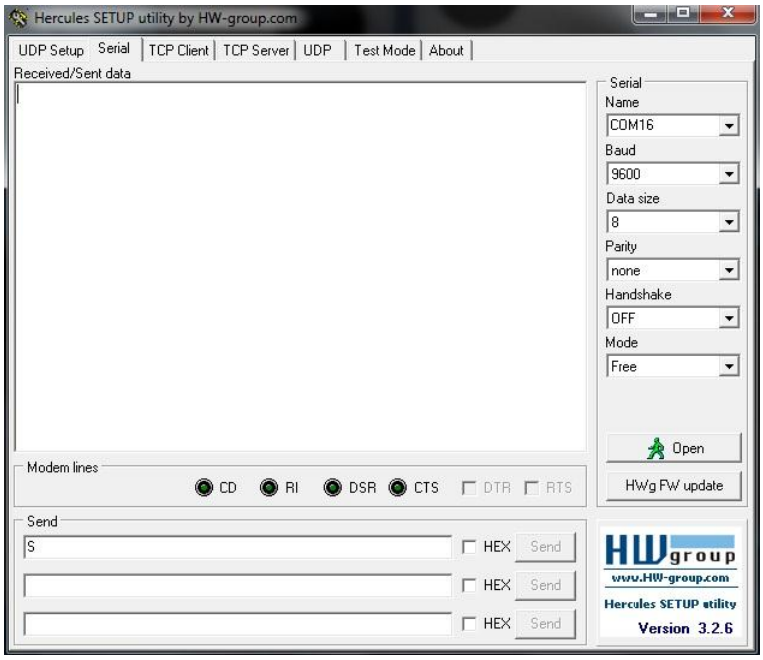
Sin embargo, Arduino soporta otros lenguajes de programación que también soportan la transmisión de datos mediante la comunicación serie (Java, Processing, Python, Visual Basic,...).



Dicho entorno se puede descargar gratuitamente de la web de Arduino. Como vemos, consta de un editor de texto, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes, y una serie de menús.

Hay que tener en cuenta, a la hora de trabajar y subir los sketches a la placa, que primero se debe seleccionar en la pestaña de "Herramientas" la placa y el puerto con el que se está trabajando. El caso del puerto es el más importante, pues si le damos a cargar en el puerto erróneo estaremos esperando hasta que salte el error correspondiente de *timeout()*.

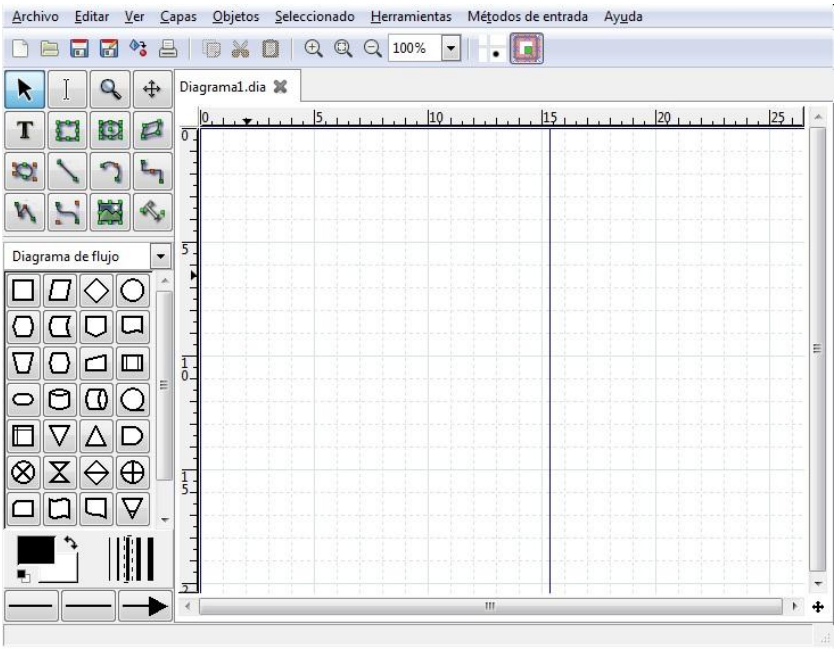
Para las pruebas finales, se empezó utilizando el terminal serie Putty, pero acabó siendo sustituido por el *Hercules*. Otro terminal más completo, sencillo, gratuito y lleno de utilidades para trabajar tanto con puerto serie, clientes y servidores TCP/IP y terminales UDP.



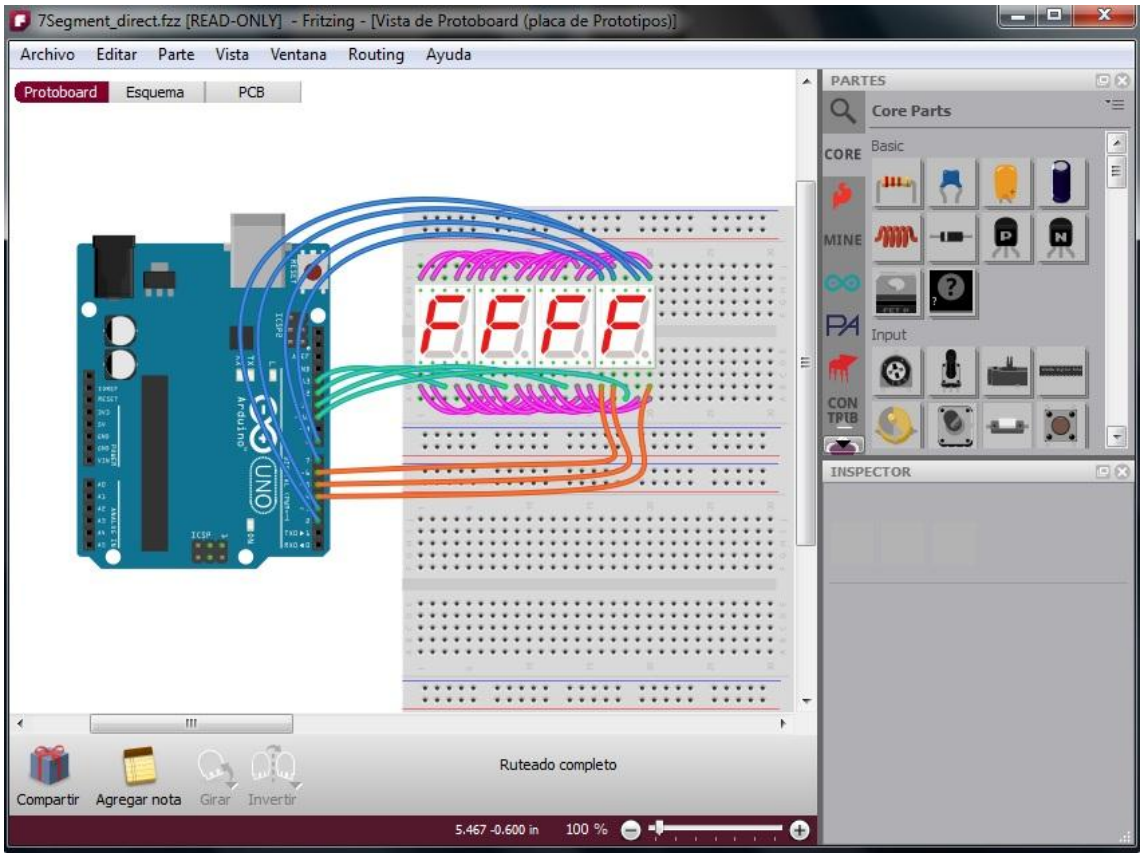
Finalmente, para la realización de los diagramas de flujo de la memoria, y el esquema de conexiones del apartado de montaje, se han empleado dos entornos de trabajo muy útiles y que podrían ser empleados en cualquier otro proyecto similar a este.

Para el diagrama de flujo se ha empleado el entorno Dia, un entorno gratuito y sencillo basado en el Visio de Microsoft y que se presenta como una alternativa para usos más casuales. En este se pueden realizar distintos tipos de diagramas: diagramas de flujo, diagramas de bases de datos, circuitos eléctricos, UMLs, etc.

Los diagramas realizados pueden ser cargados y guardados en formato XML (comprimido por gzip) y pueden ser exportados en muchos otros formatos como JPEG, pdf, PNG, WMF...



Para el esquema del apartado de montaje, se ha empleado el entorno Fritzing. Se trata de un entorno libre para la creación y documentación de circuitos electrónicos. Basado y creado principalmente para los trabajos realizados con Processing y Arduino; la idea es ayudar a trabajadores y artistas a documentar sus proyectos y crear los esquemas del circuito impreso para su posterior fabricación.



1.6 Hardware empleado

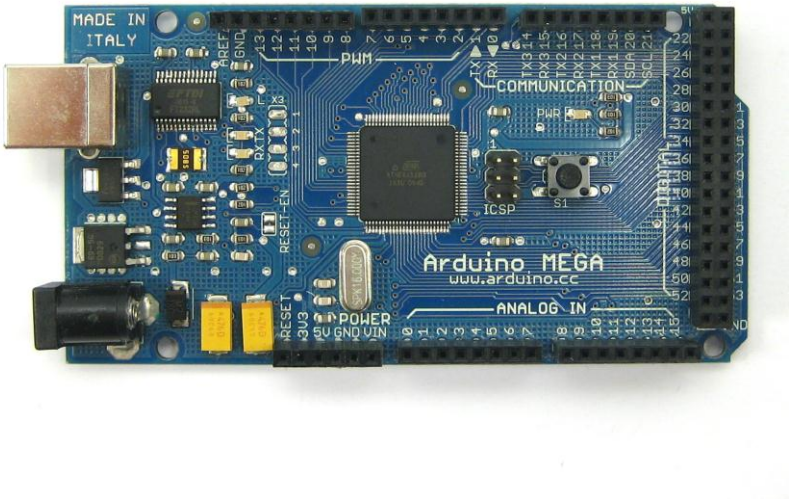
Aparte del ya mencionado Arduino, para conseguir desarrollar el control de navegación hacían falta otra serie de dispositivos que pudiesen proporcionar toda la información necesaria para poder controlar el sistema en todo momento. A continuación se detallan además de los empleados, otros que se consideraron y que podrían haber enriquecido su funcionamiento.

1.6.1 Arduino

Arduino es una plataforma electrónica libre, basada en software y hardware flexibles y fáciles de usar. Se basa simplemente en una placa con un microcontrolador y una serie de pines de entrada/salida, junto con un entorno de desarrollo.

Su éxito se debe no solo a su sencillez, sino también a su carácter open-hardware, que permite no solo crear nosotros mismos nuestra propia versión de Arduino (existen las placas "originales" creadas por la propia comunidad de Arduino, y las "no oficiales" creadas por otra gente pero que guardan muchas similitudes con las originales y son igualmente válidas), sino que además es posible realizar cualquier tipo de proyecto sin tener que obtener ningún tipo de licencia.

La versión más básica es el llamado Arduino Uno, pero para la realización del proyecto se ha utilizado el Arduino Mega, ya que posee más entradas digitales, más conexiones serie, más RAM, etc.



El Arduino Mega estaba basado en un ATMega1280; sin embargo, el empleado en el proyecto posee un microcontrolador ATMega2560. Aparte de esto, esta placa posee 54 entradas/salidas digitales (de las cuales 14 proporcionan salida PWM), 16 entradas analógicas, 4 UARTS (puertos serie por hardware), un cristal oscilador de 16MHz, conexión USB, entrada de corriente, conector ICSP y botón de reset.

Para hacerlo funcionar basta con conectarlo a una fuente de alimentación como pueda ser un PC vía USB, o una batería. En la primera conexión con el PC, si al conectarlo el ordenador no encuentra los drivers automáticamente, los podemos seleccionar manualmente seleccionando simplemente la carpeta drivers dentro del directorio donde tengamos el entorno de programación de Arduino.

Alimentación

Como ya he dicho, la placa puede alimentarse tanto por USB como con otro tipo de fuente de alimentación. Las fuentes de alimentación externas (no-USB) pueden ser

tanto un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1 mm con centro positivo en el conector hembra de la placa. Los cables de la batería se pueden conectar a los pines Gnd y Vin en los conectores de alimentación. Luego, el propio Arduino escoge automáticamente el origen de la alimentación.

La placa puede trabajar un con un voltaje entre 6 y 20v. Sin embargo, conlleva una serie de precauciones: la primera, es que si la placa trabaja con menos de 7v, los pines de 5v podrían no suministrar esos 5v completos, y la segunda es que con más de 12v los reguladores de corriente podrían sobrecalentarse, dañando la placa. Por ello, lo ideal es suministrar una alimentación comprendida entre esos dos valores: 7-12v.

Respecto a los pines de alimentación tenemos:

- VIN: entrada de alimentación cuando no se desea utilizar la USB como fuente.
- 5V: La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa.
- 3V3: Una fuente de voltaje a 3.3 voltios generada en el chip FTDI integrado en la placa. La corriente máxima soportada es 50mA.
- GND: Pines de toma de tierra.

Entradas/salidas digitales

Cada uno de los 54 pines digitales puede ser declarado como entrada o salida gracias a la función pinMode(). Luego, según como se haya declarado dicho pin podremos trabajar con él utilizando las funciones digitalWrite() o digitalRead(). Los pines trabajan con 5 voltios; y cada pin puede proporcionar o recibir una intensidad máxima de 40mA. Además poseen una resistencia de pull-up (desconectada por defecto) de 20-50kOhms. Aparte, hay pines que tienen una serie de funciones especiales:

- Serie: 0 (RX) y 1 (TX), Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX): Usados para recibir (RX) y transmitir (TX) datos a través de puerto serie TTL.

  Nota: tener en cuenta que generalmente, a la hora de conectarnos vía serie con otros dispositivos, deberemos conectar los pines RX y TX cruzados, véase:

  RX1<>TX2 + RX2<>TX1

- Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2): Estos pines se pueden configurar para lanzar una interrupción según el comportamiento que tenga el pin (valor LOW, valor HIGH, cambio de valor o cambio a flanco de subida/bajada).

- PWM: de 0 a 13. Proporciona una salida PWM de 8 bits de valores de 0 a 255, a través de la función analogWrite() (notar que esta función por tanto no tiene nada que ver con los pines analógicos).
- SPI: 50 (SS), 51 (MOSI), 52 (MISO), 53 (SCK). Estos pines proporcionan comunicación SPI (*Serial Peripheral Interface,* estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj).
- Pin 13: pin con un led integrado que se enciende cuando el pin toma el valor HIGH y se apaga al tomar el valor LOW.
- I2C: 20 (SDA) y 21 (SCL). Soporte del protocolo de comunicaciones I2C usando la librería Wire.
- Reset: Suministra un valor LOW para reiniciar el microcontrolador.

Comunicaciones

El Arduino Mega facilita como vemos la conexión serie con otros dispositivos, ya que además de su puerto USB y los pines que actúan también como puertos serie, tenemos una opción más, que es la librería SoftwareSerial, que será explicada más adelante.

Resumen

Microcontrolador	ATmega2560
Voltaje de funcionamiento	5v
Voltaje de entrada	7-12v
Voltaje de entrada (límite)	6-20v
Pines E/S digitales	54
Pines de entrada analógica	16
Intensidad por pin	40mA
Intensidad en pin 3.3v	50mA
Memoria Flash	256KB
SRAM	8KB
EEPROM	4KB
Velocidad de reloj	16MHz

1.6.2 SKM53

El SKM53 es el módulo GPS empleado para el desarrollo del proyecto. Posee una antena integrada la cual aumenta su rango de alcance y señal respecto a otros modelos anteriores. Características principales:

1. Funcionamiento mediante el protocolo NMEA.
2. Batería interna de apoyo.
3. Puerto Serie.
4. Rango de temperatura de funcionamiento: -40 hasta 85 ºC.
5. Tamaño: 30mm x20mm x 8.5mm.
6. Funcionamiento a 5v.
7. Tiempo de adquisición de datos en frío 36 segs.

Protocolo NMEA

El protocolo NMEA es un protocolo creado para la comunicación entre dispositivos GPS. Los datos recibidos se basan en una serie de tramas donde el primer carácter especifica qué información recibimos en el resto de la trama. Tramas que soporta el módulo:

- GGA: datos básicos de posición.
- GLL: datos de latitud y longitud.
- GSA: GNSS (*Global Navigation Satellite System)* y número de satélites activos.
- GSV: GNSS y número de satélites al alcance.
- RMC: información mínima recomendada de posición.
- VTG: contiene el rumbo y velocidad del módulo GPS.
- ZDA: datos de fecha y hora.
- DTM: datos ID de la fuente de la cual obtenemos datos.

```
$GPGGA,114814.000,3928.9605,N,00020.7784,W,1,5,1.53,-11.4,M,51.9,M,,*55
$GPGSA,A,3,15,17,25,24,12,,,,,,,,1.77,1.53,0.88*04
$GPGSV,2,1,06,12,77,292,42,24,70,054,43,25,40,250,40,15,38,169,46*74
$GPGSV,2,2,06,39,36,143,41,17,15,040,43*72
$GPRMC,114814.000,A,3928.9605,N,00020.7784,W,2.32,108.07,250613,,,A*7F
```

Como vemos en el ejemplo, esas son las tramas que recibíamos por defecto con el GPS, a continuación figura con más detalle la información que nos da cada una:

$GPGGA	
Hora hhmmss	114814
Latitud	3928.9605
Norte/Sur	N
Longitud	0020.7784
Este/Oeste	W
Indicador de posición fija	1
Satélites usados	5
HDOP	1.53
Altitud	-11.4
Unidad de altitud	M
Separación geoidal	51.9
Unidad de separación geoidal	M
NULL	
Checksum	*55

$GPGSA	
Modo1	A
Modo2	3
ID del satélite usado1	15
ID del satélite usado2	17
[…]	[…]
PDOP	1.77
HDOP	1.53
VDOP	0.88
Checksum	*04

$GPGSA	
Número de mensajes	2
Número de mensaje	1
Satélites a la vista	06
ID satélite1	12
Elevación	77º
Azimut	292º
SNR	42
[…]	
Checksum	*74

$GPRMC	
Hora	114814
Status	A
Latitud	3928.9605
Norte/Sur	N
Longitud	0020.7784
Este/Oeste	W
Velocidad respecto al suelo	2.32
Rumbo respecto al suelo	108.07
Fecha ddmmaa	250613
Variación Magnética	NULL
Dirección de la variación magnética	NULL
Modo de posición	A
Checksum	*7F

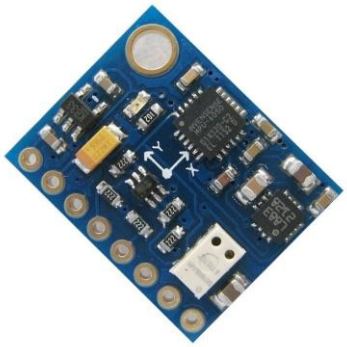
**Nota: cuando vemos que en una tabla aparece "[...]" es porque se repiten ciertos apartados iterativamente, es decir, ciertos campos para el satélite1, los mismos para el 2, y así sucesivamente hasta el último.*

1.6.3 HMC5883L

Se trata de un chip que empleamos como compás digital (brújula). Éste devolverá el número de grados de desviamiento que tenemos con respecto al norte magnético, siendo 0 el norte y 180 el sur. Para la transmisión de datos, utiliza la transmisión por bus I2C.

El I2C (Inter-Integrated Circuit) es un bus de transmisión serie que se caracteriza por tener dos líneas de información: una para la transmisión de datos y otra para el reloj.

SDA => línea de datos || SCL=> línea de reloj



Para el intercambio de datos, sigue un protocolo de maestro/esclavo, en la cual cada uno de los "participantes" puede acceder al bus cuando las líneas SDA y SCL se encuentran en estado lógico alto, y se comunican entre ellos mediante pulsos de reloj y el envío y recepción de ACKs.

Por otro lado, el chip del HMC, viene en un módulo en el que también tenemos otro tipo de sensores. El módulo comprado es: 9DOF ITG3205 ADXL345 HMC5883L IMU MWC; donde:

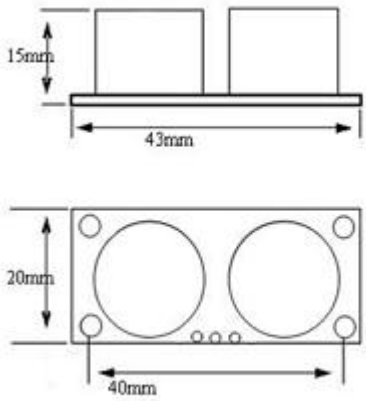
- Chip ITG3205: giroscopio
- Chip ADXL345: acelerómetro
- Chip HMC5883L: compás digital

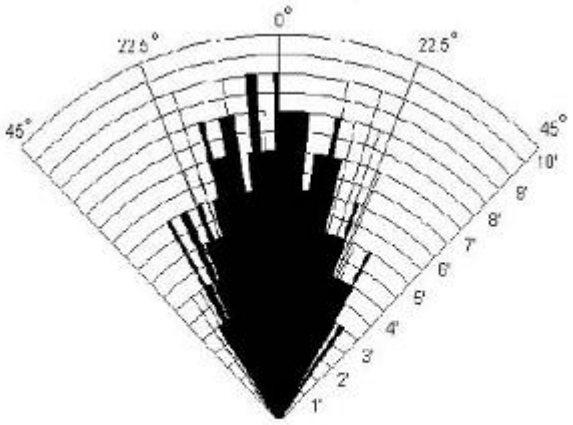
1.6.4 HC-SR04

El HC-SR04 es un sensor de proximidad que usa un sonar para la detección de obstáculos. Por uno de sus sonares envía la onda sonora en un cono de unos 30º hacia delante, y por la otra recibe de nuevo la onda sonora, que vuelve tras haber chocado con el obstáculo. Sus pines Trig y Echo son las conexiones a cada uno de éstos.

Tiene un alcance que va desde los 2 cm hasta los 4 m. Su funcionamiento no se ve afectado por la luz ni por ninguna superficie negra (algunos sensores de proximidad se

ven afectados por esto al parecer), pero puede tener algún problema con algunos materiales como la tela que son difíciles de detectar.





Practical test of performance,
Best in 30 degree angle

Pines:

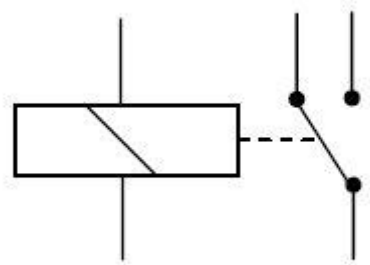
- VCC = +5VDC
- Trig = Trigger input of Sensor
- Echo = Echo output of Sensor
- GND = GND

Características:

- Alimentación :+5V DC
- Corriente en reposo : <2mA
- Corriente de trabajo: 15mA
- Ángulo eficaz: <15°
- Rango de distancia : 2cm – 400cm
- Error : 0.3 cm
- Ángulo de medición: 30°
- Dimensión: 43mm x 20mm x 15mm

1.6.5 Módulo relé

Inventado por Joseph Henry en 1835, un relé es un dispositivo electromecánico que funciona como un selector. Está formado por una serie de contactos, una bobina y un electroimán. Su funcionamiento es muy sencillo: los contactos, están inicialmente en cierta posición, pero cuando excitamos la bobina, se crea un campo magnético que hace que los contactos cambien (comportamiento de selector). Con esto conseguimos seleccionar la activación de dos circuitos eléctricos distintos.



Contactos

Los contactos del relé pueden ser N.A (normalmente abierto), N.C (normalmente cerrado) o de conmutación.

- Los contactos Normalmente Abiertos conectan el circuito cuando el relé es activado; el circuito se desconecta cuando el relé está inactivo. Este tipo de contactos son ideales para aplicaciones en las que se requiere conmutar fuentes de poder de alta intensidad para dispositivos remotos.
- Los contactos Normalmente Cerrados desconectan el circuito cuando el relé es activado; el circuito se conecta cuando el relé está inactivo. Estos contactos se utilizan para aplicaciones en las que se requiere que el circuito permanezca cerrado hasta que el relé sea activado.
- Los contactos de conmutación controlan dos circuitos: un contacto N.A y uno N.C con una terminal común.

Pines:

- VCC : 5v
- GND
- Chanel1 => relé1
- Chanel2 =>relé2

1.6.6 Comunicación

El caso de la comunicación es especial, aquí inicialmente se pretendía utilizar unos módulos de radiofrecuencia de largo alcance, pero hubo muchos problemas a la hora de probarlos. Un módulo que ejercía de emisor y otro de receptor, pero no se conectaban entre ellos y se quedaban colgados buscándose el uno al otro.

A raíz de este problema se pasó al plan B, y la comunicación se realizó mediante un módulo bluetooth.

1.6.6.1 Módulo rf1100

Éstos son los módulos de radiofrecuencia con los que inicialmente se deseaba realizar el proyecto.

Fue una pena no conseguir que funcionasen correctamente pues tenían un alcance muy grande (250-300m en espacios abiertos, como iba a ser el caso del lugar de pruebas).

Sin embargo, planteaban un posible problema futuro, y es que lo más probable fuese que para aprovechar al máximo ese alcance, la antena debería ir asomada por fuera de la carcasa del barco, lo que implicaría tener que agujerearla, lo que con un poco de agua, si no estuviese bien sellado originaría un grave problema para la seguridad de la electrónica.

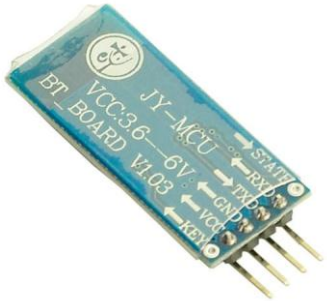
1.6.6.2 Módulo JY-MCU

Un bluetooth es un dispositivo que permite el envío de datos mediante radiofrecuencia por la banda ISM (las ISM son bandas internacionalmente reservadas para el uso no comercial de radiofrecuencia) de los 2,4GHz. Los principales objetivos del bluetooth son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

El JY-MCU es un módulo muy sencillo de utilizar ya que es totalmente transparente. Para comenzar a utilizarlo simplemente hay que alimentarlo y esperar a que se conecte a un bluetooth maestro. Si utilizamos como maestro por ejemplo el bluetooth de un ordenador portátil, en la zona de dispositivos bluetooth del panel de control lo

encontraremos como “linvor” por defecto. El dispositivo también posee una contraseña predeterminada de serie, que es “1234”. Cuando lo alimentemos, mientras el led rojo esté parpadeando, significará que está buscando un dispositivo al que conectarse, y cuando pare, será que ya lo habrá logrado.



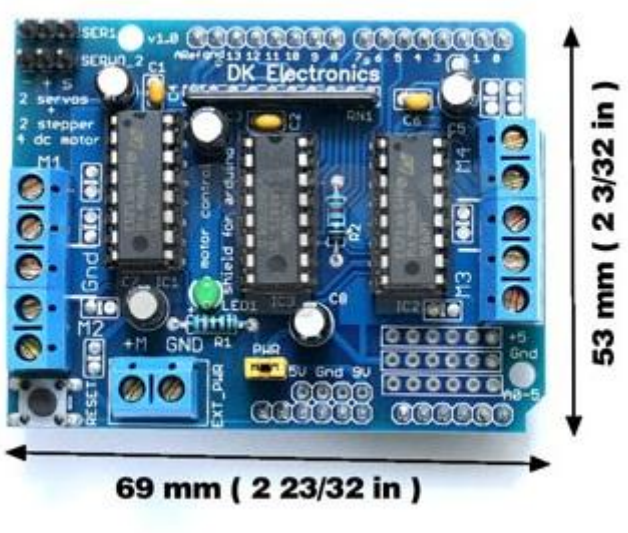
Su funcionamiento es como si de un cable se tratase. Mediante una transmisión serie, para iniciar el intercambio de datos simplemente debemos seleccionar el puerto que ocupa el módulo y podremos comenzar.

**Muy importante recordar el conectar los RX y TX cruzados.*

1.6.7 Control de los motores

Para el control de los motores se empleó un shield para Arduino (*Motor Drive Shield Expansion Board L293D*) que permitía regular tanto la velocidad de giro como el sentido de los motores. Para ello, el shield posee dos chips L293D, chips que llevan integrado un circuito de puente en H.

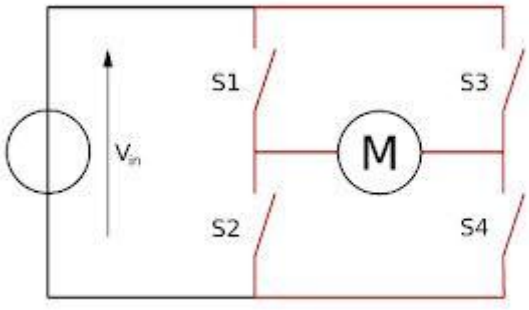
El shield posee características como la posibilidad de hacer funcionar 4 motores DC, o 2 servos, o 2 motores paso-a-paso; la posibilidad de hacer funcionar motores desde 4.5 hasta 36v (teniendo cuidado con el calor) y dos terminales para poder alimentarlo. Puede alimentarse por tanto simplemente conectándolo sobre los pines de Arduino, o por la entrada de alimentación externa para obtener un voltaje superior.



En las pruebas iniciales, el shield conectado únicamente a Arduino no poseía casi fuerza para mover un único motor de pruebas; para que funcionase correctamente había que conectar una pila a la entrada de alimentación externa (EXT_PWR).

Puente en H

Un puente en H es un circuito electrónico que sirve para que un motor DC pueda girar en ambos sentidos y poder tener la posibilidad tanto de avanzar como de retroceder. Su nombre de H viene de la disposición de los distintos interruptores por los que circula la corriente.

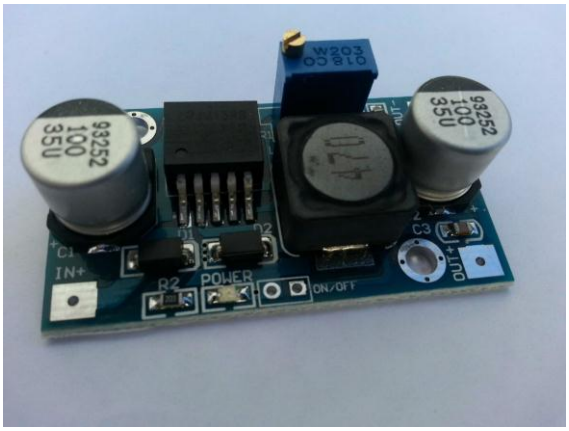


Además de avance y retroceso, el puente H también puede usarse para frenar un motor. Esto se logra al hacer un cortocircuito entre los bornes del motor, o incluso puede usarse para permitir que el motor frene bajo su propia inercia, cuando desconectamos el motor de la fuente que lo alimenta.

1.6.8 Convertidor DC-DC

Aunque no llegó a utilizarse porque no fue necesario, sí que es un dispositivo necesario para los que usen una fuente de alimentación más potente.

El DC-DC es un dispositivo que transforma corriente continua de una tensión en otra. Se usa sobre todo para regular la corriente. Su uso conlleva la ventaja de reducir o simplificar la alimentación del sistema, porque permiten generar las tensiones donde se necesitan, reduciendo la cantidad de líneas de potencia necesarias. Además permiten un mejor manejo de la potencia, control de tensiones de entrada y un aumento en la seguridad. Sin embargo, también hay que destacar que generan algo de ruido.

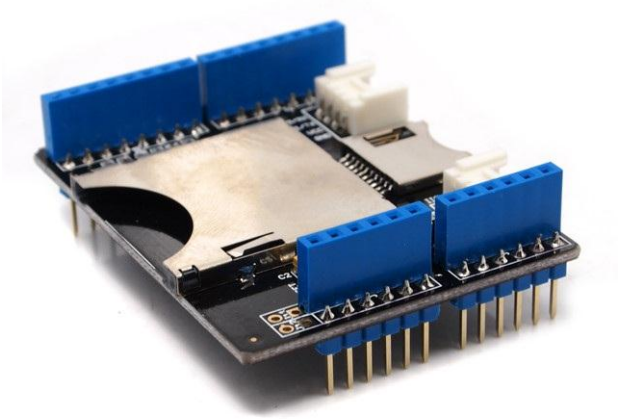


1.6.9 Shield SD

Otra opción interesante que se barajó, lamentablemente ya bastante tarde (por el tema de adquirirlo más que nada y la etapa del proyecto en la se encontraba), fue el hecho de utilizar otro shield para Arduino, en este caso para que contuviese una tarjeta Micro SD.

La idea por una parte era que dado que no se iba a poder tener comunicación continua con el barco debido a que el alcance del bluetooth era muy limitado (unos 10m), se podría haber utilizado la SD para guardar los datos recopilados durante la ruta y más tarde, conectar la SD al PC, crear un archivo .kml y observar la ruta recorrida.

Los ficheros .kml son utilizados por Google Earth para guardar los distintos puntos de una ruta. Dichos ficheros pueden ser luego cargados y ser observados sobre el mapa, lo que supondría una comprobación gráfica y fiable de los resultados.



1.7 Modelo de barco

Para la elección del barco que contuviese todo el sistema, se visitó una tienda de modelismo, y se estudió la elección entre los siguientes criterios: espacio, precio y resistencia.

De los modelos vistos, muchos eran muy pequeños y la parte accesible no tenía prácticamente espacio ni para contener todo el sistema ni para maniobrar e intentar meterlo de algún modo. Otro barco que si cumplía todos los requisitos era uno con forma de barco pesquero. El problema era que efectivamente, su forma estaba hecha para ello, y se trataba de un barco que se utiliza para pescar realmente, por lo que su precio era desorbitado al tratarse de un modelo que nada tenía de juguete. Finalmente, estudiando las lanchas, se encontró la elegida. Ésta presentaba una serie de pros y

contras: por un lado, poseía mucho espacio en el tren delantero para albergar todos los dispositivos del sistema y económicamente era de las más viables que había; sin embargo, lo que se ganaba por un sitio se perdía por otro, y es que el material del chasis era de un material poco resistente, un plástico muy sencillo por el que probablemente un golpe fuerte acarrearía graves consecuencias, además, la batería era muy mejorable en cuanto a voltaje y duración, ya que de serie, su tiempo de funcionamiento era de entre 10 y 15 minutos.

La resistencia, a pesar de ser un inconveniente importante, se podía compensar con mucho cuidado a la hora de probar. El caso es, que la ventaja económica pesaba mucho, y es que el precio de esta lancha eran unos 75€, mientras que para la siguiente, también con mucho espacio pero de un material mucho más resistente, su precio ascendía ya a 250€. Así se eligió la lancha PX-16.

1.8 Riesgos

Hablamos de riesgos cuando nos referimos a todos aquellos factores o situaciones que pueden alterar el funcionamiento o desarrollo del sistema de una forma crítica. En nuestro caso, la mayoría se centran en la fase de pruebas, ya que como vamos a trabajar con un medio físico, cualquier contratiempo va a afectarle drásticamente:

- A nivel de la estructura física del sistema, correspondiente a la maqueta, el riesgo principal es que debido al frágil material del que está hecha su carcasa; es

muy fácil romperla o causarle una brecha; por lo que en un choque se podría provocar su hundimiento y con ella todo el sistema electrónico.

- Dependemos también de factores externos como pueda ser el viento o alguna pequeña corriente de agua, que puede provocar desde pérdidas del rumbo a una colisión del barco con un obstáculo.
- El balance de pesos, ya que como vamos a añadir "extras", aunque no sean muy pesados, sí que es cierto que si la lancha o barco no está equilibrado/a, habrá más probabilidad de que se produzcan más desviaciones en el rumbo o que incluso no flote correctamente en el agua.
- La importancia del tiempo: usando una programación modular, en la que se tiene que controlar el periodo con el que se realizan las distintas funciones, una actividad que conlleve demasiado tiempo puede conllevar que las mediciones se realicen demasiado tarde, encadenándose fallos en el rumbo, orientación, etc.
- La importancia de calibrar bien el movimiento y reacción de los motores ya que si cuando pretendamos corregir el rumbo para mantenernos en la ruta programada, ejecutamos movimientos con una gran brusquedad, nos encontraremos con que más que recuperar el rumbo, lo habremos empeorado.
- Otro riesgo, viene dado por el GPS. Si por cualquier motivo se perdiese la conexión con los satélites, las coordenadas leídas por Arduino serían 0.0000000 y 0.0000000, lo que haría que el barco cambiase drásticamente de rumbo y se alejase descontroladamente sin control hasta que se recuperase la conexión con los satélites (aunque para evitar o contrarrestar esto tendríamos el control manual del barco como as bajo la manga).

2. Especificación de requisitos

En la ingeniería de sistemas, definimos los requisitos como una serie de condiciones o términos que el sistema debe cumplir para que el objetivo principal del proyecto se cumpla. En este caso, para que el control de navegación sea lo más completo posible.

Dentro de los requisitos, diferenciaremos entre los *funcionales* y los *no funcionales*.

Los funcionales son aquellos que definen características del sistema referentes a qué debe hacer el sistema. Y los no funcionales se refieren más a factores como la accesibilidad, mantenimiento, calidad, etc.; y a cómo debería realizar el sistema ciertas acciones.

2.1 Requisitos funcionales

- ✓ El sistema debe ser capaz de leer una serie de puntos que constituirán la ruta a seguir. Una vez interpretados debe recorrerla lo más precisamente posible.
- ✓ El sistema debe ser capaz de evadir posibles obstáculos que encuentre durante su trayecto.
- ✓ Debe poseer un sistema de emergencia para el caso en que se quede colgado. Dicho de otro modo, se deberá mantener también la electrónica original del barco para en caso de ser necesario, poder controlarlo mediante el mando radiocontrol.
- ✓ Deberá avisar cuando haya finalizado la ruta.

2.2 Requisitos no funcionales

- ✓ Dada su simple funcionalidad, la interfaz debe ser lo más sencilla posible.
- ✓ El sistema debe estar perfectamente preparado para trabajar en grandes espacios acuosos (lagos, piscinas, grandes fuentes, etc.).
- ✓ Debe proteger la electrónica lo mejor posible.

3. Planificación

La planificación está enfocada desde dos puntos de vista: el organizativo de las distintas partes de desarrollo del proyecto y la parte económica, es decir el presupuesto de todos los materiales necesarios.

3.1 Descomposición en fases del desarrollo del proyecto

El proyecto ha seguido un orden en el desarrollo de cada una de las partes que lo compone. Como se dice en el apartado del método de desarrollo empleado, se ha ido desarrollando cada vez una parte correspondiente a cada uno de los componentes y acoplando así al programa principal. Por ello, se han detallado las distintas tareas que se han llevado a cabo con cada parte del proyecto.

- ***Etapa1. Fase de documentación 1.***

Declarar el objetivo o finalidad del proyecto

Requerimientos básicos y riesgos

Lista de material principal que va a utilizarse

- ***Etapa 2. Fase de documentación 2.***

Informarse acerca de los medios que se van a utilizar: entornos de desarrollo, información sobre el material, alternativas...

- ***Etapa 3. Base para el desarrollo.***

Diagrama de flujo básico. Un diagrama simple sobre lo que debe ser aparentemente el comportamiento del sistema.

Descomposición de éste en fases para poder dividir el desarrollo y la implementación en módulos.

- ***Etapa 4. Coordenadas.***

Diseñar el paso de coordenadas. Para ello, probar la comunicación PC-Arduino y el rellenado del array.

- ***Etapa 5. Compás digital.***

Probar las distintas librerías e interpretar los distintos resultados.

Extraer el método para comprobar el rumbo del sistema.

- ***Etapa 6. GPS y comienzo de la comunicación.***

Probar las distintas librerías. Extraer el método para la obtención de la posición actual.

Prueba de los primeros módulos de radiofrecuencia.

Desarrollar el método para el cálculo de la distancia.

Desarrollar el método para la comprobación de si el punto está alcanzado.

Revisión del diagrama de flujo y actualización del programa principal.

- ***Etapa 7. Sensor de proximidad.***

Prueba de la librería y sus ejemplos.

Desarrollo del método para la detección de obstáculos.

Desarrollo del método para esquivarlos.

Revisión del diagrama de flujo y actualización del programa principal.

- ***Etapa 8. Control de motores.***

Prueba de la librería del shield.

Prueba del tratamiento del movimiento con un motor de pruebas.

Desarrollo del método de movimiento del barco.

Desarrollo del método para corregir el rumbo.

Revisión del diagrama de flujo y actualización del programa principal.

- ***Etapa9. Comunicación.***

Prueba del bluetooth.

Revisión y modificación del programa principal.

- ***Etapa 10. Montaje.***

Montaje de todo el sistema en el barco

- ***Etapa 11. Fase de pruebas.***

Pruebas pertinentes de todo el sistema ya montado.

3.2 Presupuesto

Dejando a un lado lo que podrían ser los costes de personal si este proyecto perteneciese a una empresa, en este presupuesto principal figuran los precios y el total de los materiales principales utilizados:

Arduino Mega	15.95 €
Compás digital	18.81 €
GPS	24.95 €
Módulo relé	4.50 €
Sensor de ultrasonidos	2.90 €
Shield para el control de los motores	3.85 €
Módulo bluetooth	15.45 €
Materiales secundarios	10 €
Barco/lancha radiocontrol	75 €
TOTAL	**171.41 €**

Este presupuesto podría tener ciertas variaciones si en lugar del bluetooth hubiésemos empleado los módulos de radiofrecuencia iniciales **(23.32 €),** o si hubiésemos podido añadir finalmente el shield para la SD **(1.11 (shield) + 5.20 (tarjeta SD)=6.31€).**

4. Análisis

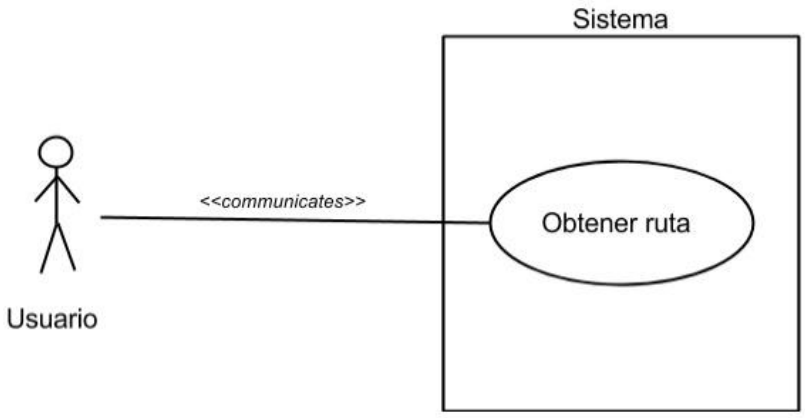
En éste apartado se analiza todo el comportamiento que tiene el sistema. Por un lado tenemos los casos de uso, y por otro el diagrama de flujo.

4.1 Casos de uso

Llamamos casos de uso al conjunto de actividades que pueden o deben realizarse para que se lleve a cabo el proceso. Dichas actividades son aquellas en las que intervienen los distintos usuarios que se encuentren utilizando la aplicación y el propio sistema.

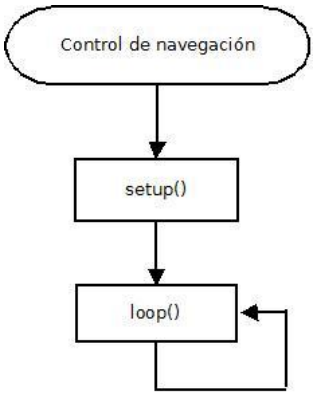
Una de sus principales aplicaciones suele ser la captación de requisitos funcionales al mostrar cómo reacciona el sistema ante ciertos eventos.

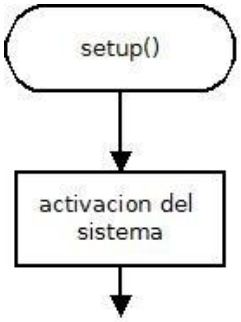
En el caso de este proyecto, el diagrama resulta muy sencillo, pues la única interactividad que hay con el usuario es el pase de coordenadas para la ruta del barco.

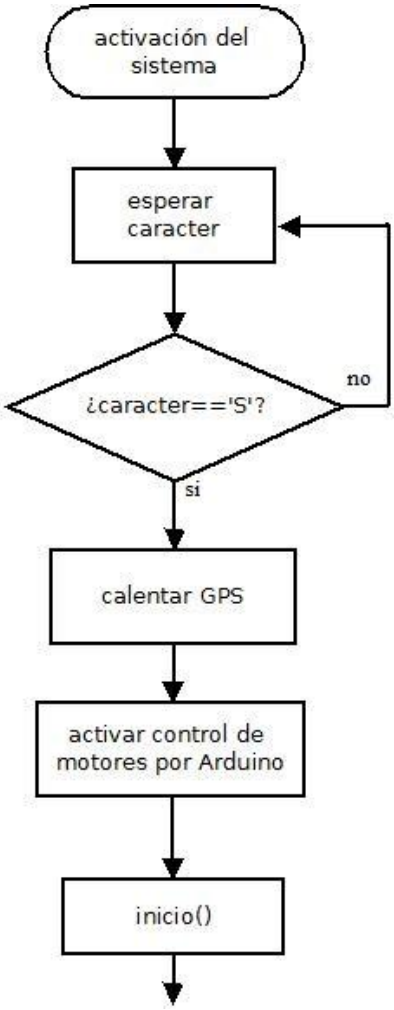


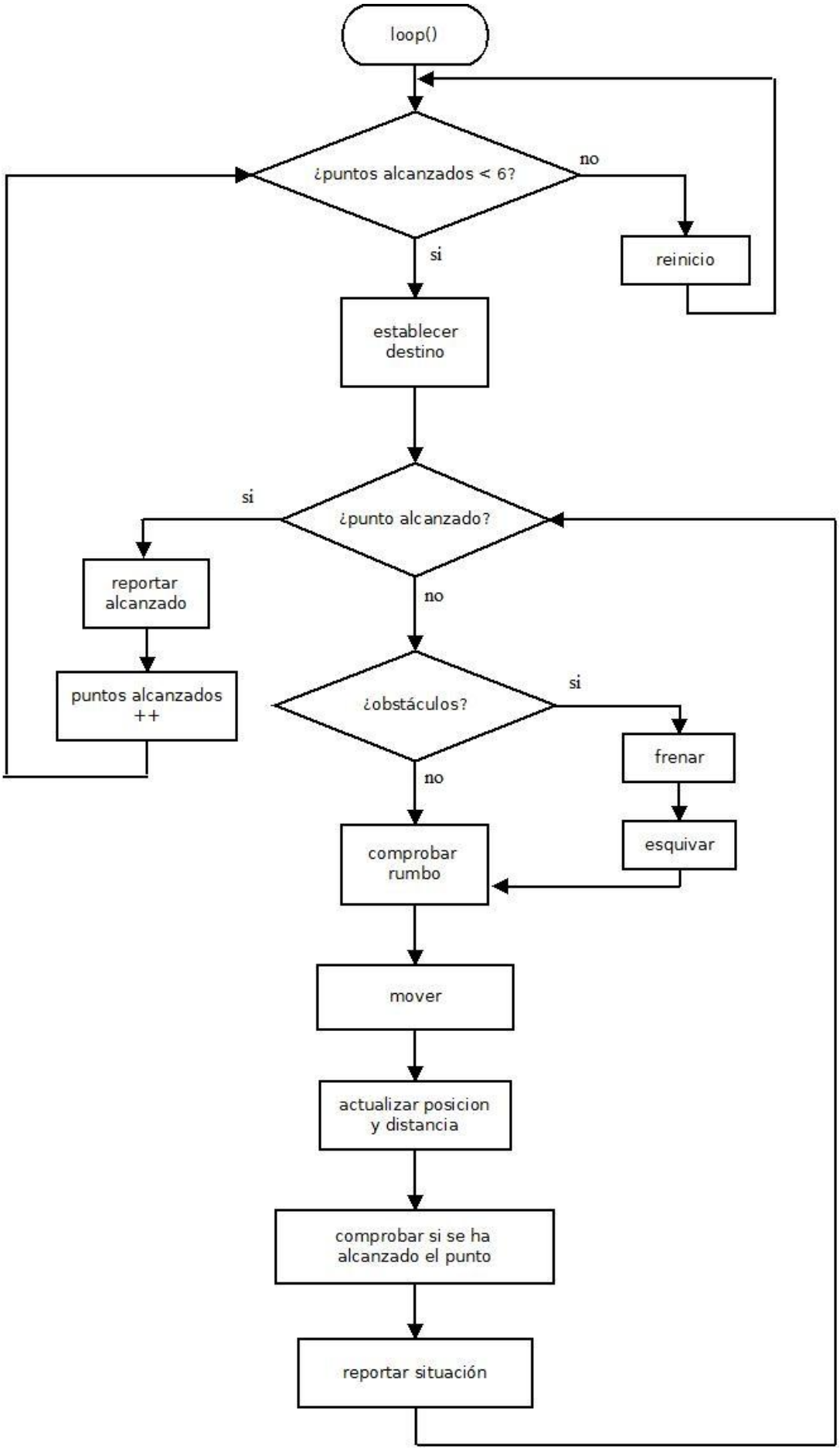
4.2 Diagrama de flujo

El diagrama de flujo es una representación gráfica del comportamiento de lo que queremos implementar. Se utilizan en la inmensa mayoría de proyectos pues permiten ver fácilmente qué se quiere hacer, posibles cambios que se puedan realizar, o problemas que a primera vista puedan pasar desapercibidos.


Control de navegación
setup()
loop()


setup()
activacion del sistema


activación del sistema
esperar caracter
¿caracter=='S'?
no
si
calentar GPS
activar control de motores por Arduino
inicio()


loop()
¿puntos alcanzados < 6?
no
si
reinicio
establecer destino
si
¿punto alcanzado?
reportar alcanzado
puntos alcanzados ++
no
¿obstáculos?
si
frenar
esquivar
no
comprobar rumbo
mover
actualizar posicion y distancia
comprobar si se ha alcanzado el punto
reportar situación

5. Diseño

5.1 Estructura

El sistema está organizado en seis archivos. Cada uno de estos contiene una serie de métodos relacionados con unas funciones comunes. El hecho de dividir todo el programa en diversos archivos favorece la modularidad de éste.

Dicha modularidad a su vez favorece la independencia entre las distintas partes del programa. La ventaja principal de dicha independencia es el hecho de que modificar uno de los módulos afectará de la menor forma posible a los demás. Además, subdiviendo el problema principal en otros más pequeños se consigue una mayor sencillez y lo hace más legible y manejable.

Así, cada módulo cumple una tarea concreta y necesaria dentro del objetivo final que es el problema o programa principal. Esta organización en subdivisiones es la base del diseño estructurado del software.

Archivos del sistema:

1. Control de navegación
2. Compas
3. GPS
4. Funcionamiento
5. Movimiento
6. Reportes

5.1.1 Control de navegación

Es el archivo principal. Podemos dividirlo en cuatro partes:

La primera contiene la lista con todas las librerías que van a utilizarse y la definición de las variables constantes necesarias.

La segunda es la declaración de variables globales y de control que vamos a necesitar; además de su inicialización.

Y ahora, las dos siguientes corresponden con la estructura básica de cualquier programa de Arduino: el *setup* y el *loop*.

La tercera es pues el *setup()*. Se utiliza para inicializar variables también, inicializar librerías, el estado de los pines, etc. Esta parte de código solo se ejecutará una vez bien por dos motivos: el primero, cuando la placa se conecte a una fuente de alimentación, o la segunda, si se pulsa el botón de reset de la placa.

Y la cuarta el loop(), que es el fragmento de código que se ejecuta continuamente.

5.1.2 Compás

En este archivo se encuentran los métodos relacionados al trabajo con el compás digital, y aquellos que tienen que ver con el rumbo del barco: cálculo y comprobación de éste.

5.1.3 GPS

Este archivo contiene todo lo relacionado a la obtención de la posición actual y el cálculo de la distancia.

Cálculo de la distancia

Como estamos trabajando con distancias relativamente grandes, para calcular con mayor exactitud la distancia entre dos puntos en lugar de usar la trigonometría simple, debemos emplear un método que relacione las coordenadas de latitud y longitud de ambos puntos con el radio de la tierra.

Dicho método es la fórmula de Haversine. Sin embargo, ésta fórmula conlleva un pequeño error, y es debido a que parte de que la Tierra en completamente redonda, cuando realmente, no lo es. Fórmula de Haversine:

$$R = radio\ de\ la\ Tierra$$

$$\Delta lat = lat2 - lat1$$

$$\Delta long = long2 - long1$$

$$a = sin^2(\Delta lat/2) + cos(lat1) \cdot cos(lat2) \cdot sin^2(\Delta long/2)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

5.1.4 Funcionamiento

Este archivo contiene los métodos para el funcionamiento básico del sistema, pero que nada tienen que ver con los dispositivos conectados a Arduino a excepción del sensor de ultrasonidos, que por la sencillez de su parte fue incluida aquí también para ahorrar espacio. Respecto a los otros métodos de funcionamiento, encontramos algunos como el de activación, el pase de coordenadas, la conversión de decimal a radianes y viceversa, etc.

5.1.5 Movimiento

Aquí se encuentran los tres métodos relacionados con el movimiento. Por una parte, el método encargado de esquivar obstáculos, para el cual, existe una pequeña parte aleatoria, que es el hecho de si intenta esquivar el obstáculo por la derecha o por la izquierda. Con esta aleatoriedad se ha querido simular lo que sería la capacidad del propio sistema de si probar a ir por un lado o por otro. A continuación, está el propio método encargado del movimiento normal del barco. Y finalmente, el método para frenar.

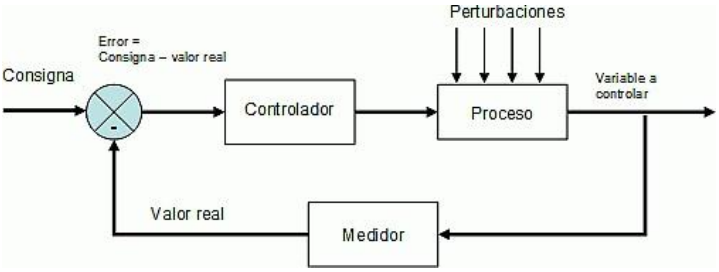
Para la gestión del movimiento se plantearon dos posibilidades:

La primera, más sencilla, consiste en: partiendo del valor de velocidad que supone girar el motor 1º, y en base al error de desviación que lleve el barco junto con un valor de velocidad máxima, corregir el valor de la velocidad del motor (valor para el pin PWM que va desde 0 hasta 255) descontando a la velocidad máxima el valor de desviación necesario para corregir el rumbo.

Y la segunda utilizar un controlador de lazo cerrado. Para este apartado, es necesaria una explicación mayor sobre dicho controlador.

Controlador de lazo cerrado

Un controlador de este tipo actúa en los sistemas en los que la acción de control está en función de la señal de salida. Los sistemas de circuito cerrado usan la retroalimentación (característica principal) desde un resultado final para ajustar la acción de control en consecuencia.



En este caso, se empleó un controlador PID.

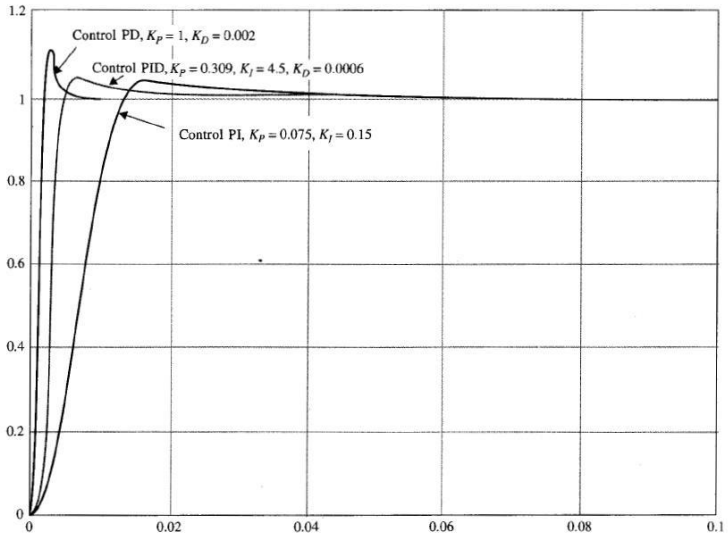
Proporcional, Integral y Derivativo (PID)

El controlador PID se basa en calcular el error entre un valor medido y un valor consigna o referencia, para luego aplicar una respuesta que haga que el sistema se acerque a dicho valor ideal. Como vemos, distinguimos tres tipos de acción:

1. La proporcional determina la reacción frente el error absoluto.
2. La integral proporciona una corrección para compensar las perturbaciones y mantener la variable controlada en el punto de consigna.
3. La derivativa anticipa el efecto de la componente proporcional para estabilizar más rápidamente la variable controlada después de cualquier perturbación.

$$Out(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}$$

La primera parte corresponde a la acción proporcional, la segunda a la integral, y la tercera a la derivativa. Además, como vemos, la fórmula posee solo tres parámetros, que son los que debemos ajustar para calibrar el controlador. También, dependiendo del tipo de función o sistema en el que deba trabajar el controlador, alguno de estos parámetros podrá valer 0, eliminando así una componente completa del controlador. Este es una opción muy usada y que otorga mayor versatilidad a las aplicaciones del PID. Según si la componente "eliminada" es la integral o la derivativa, el controlador resultante en cada caso es PD (proporcional derivativo) o PI (proporcional integral) respectivamente.



Para poder calibrarlo correctamente, primero hay que conocer bien el efecto que tendrá modificar cada una de las componentes:

K_p* => *Ganancia proporcional. Cuanto mayor es su valor, más violenta será la reacción creada por el controlador y por tanto, menos tiempo tardará en alcanzar el valor referencia. Sin embargo, una respuesta demasiado agresiva puede hacer que el sistema sea más inestable y aparezcan grandes oscilaciones.

K_i* => *Ganancia integral. Cuanto mayor sea, antes se eliminarán los errores en el estado estacionario. Un valor muy alto creará sobreoscilaciones iniciales muy altas, lo que algunos sistemas no pueden aguantar.

K_d* => *Ganancia derivativa. Valores altos contrarrestan el efecto de la sobreoscilación de la ganancia integral, pero vuelven al controlador menos violento y por consiguiente más lento. Ésta constante es muy sensible a ruidos y perturbaciones, lo que puede llevar fácilmente al sistema a la inestabilidad.

5.1.6 Reportes

Archivo con los métodos encargados de imprimir por pantalla todos los datos de la situación actual del sistema.

5.2 Interfaz

Para el seguimiento del barco, se puede visualizar en el terminal del puerto serie su posición actual (longitud y latitud), distancia hasta el punto destino, y el rumbo actual que lleva el barco; para así podernos hacer una idea o un seguimiento rápido de qué hace.

Posición actual: Longitud: x.xxxxxxx Latitud: y.yyyyyyy
Distancia hasta el destino: z.zzz
Rumbo actual: h.hh

5.3 Plan de pruebas

La realización de las distintas pruebas ha venido determinada por cada etapa del proyecto. Cada etapa en la que se trabajaba una parte nueva del sistema: partes de código, funcionamiento de un dispositivo... se finalizaba con una serie de pruebas para verificar su correcto funcionamiento, ya fuese con su función real dentro del propio sistema, o simples ejemplos para verificar que efectivamente, el tema tratado en esa etapa funcionaba correctamente.

En el apartado de pruebas se verá más en detalle lo realizado en cada una de las etapas.

6. Implementación

6.1 Librerías

A continuación figuran las distintas librerías que se han empleado para conseguir realizar el comportamiento general del sistema o el funcionamiento de los distintos dispositivos de Arduino. Para algunos, como por ejemplo el compás digital, se probaron varias opciones y finalmente se escogió la que proporcionaba un método y resultados más sencillos con los que trabajar.

6.1.1 SoftwareSerial

Librería que trae por defecto el entorno de Arduino. Sirve para crear puertos serie en los pines digitales de la placa mediante software. Sin embargo, como no es un puerto serie "puro" tiene una serie de limitaciones con respecto a los UARTS de la placa:

- Admite velocidades hasta 9600 baudios.
- El método *Serial.available()* no funciona.
- El método *Serial.read()* se queda esperando a que le lleguen todos los datos.
- Solo recibirá datos mientras se encuentre escuchando mediante el método *read()*; si llegan datos mientras se encuentra realizando otra tarea, esos datos se pierden.

En el programa principal, se ha utilizado esta librería para la comunicación con el módulo GPS.

6.1.2 Wire

Librería que viene también por defecto con el entorno de Arduino. Se utiliza para la comunicación con dispositivos que usan el bus I2C, en este caso, el compás digital.

6.1.3 HMC5883L

Librería encargada del funcionamiento del compás digital. Permite establecer diversos valores de escala y dos métodos de trabajo: continuo e individual. Para el sistema, debemos hacerlo funcionar en continuo, si no solo realizará una única medida al activar la placa. Esta librería funciona en conjunto con la Wire.

6.1.4 TinyGPS

Librería para extraer más fácilmente la información de las tramas NMEA. Sin embargo, el dispositivo GPS debe estar conectado a un puerto UART o creado con SoftwareSerial, ya que la información se obtiene a través de estos, la librería simplemente facilita la interpretación de la información recogida.

6.1.5 Ultrasonic

Librería sencilla para trabajar con el módulo de ultrasonidos.

6.1.6 PID_v1

Librería obtenida del playground de Arduino. Se trata de una librería que calcula la salida producida por un control PID, y permitiendo tener de forma sencilla más de un controlador trabajando a la vez.

PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction)

Input: la variable que queremos controlar.

Output: variable donde guardar la salida.

Setpoint: el valor de consigna o referencia.

Kp, Ki, Kd: parámetros del controlador PID.

Direction: DIRECT o INVERSE. Determina en qué dirección actuará la salida.

6.1.7 AFMotor

Librería para la gestión de los motores mediante el shield de Arduino. Basta con asignar cada motor necesario a una variable, y empezar a gestionar su velocidad y dirección de giro.

6.2 Apuntes sobre el código

En los apartados referentes a los distintos archivos en los cuales está contenido el sistema, se detallan posibles alternativas o comentarios relevantes respecto al código.

6.2.1 Variables generales y de control

- *VEL_MAX y VEL_GIRO*: constantes que reflejan la velocidad máxima a la que pueden trabajar los motores, y la velocidad para efectuar giros. La velocidad máxima es necesaria limitarla a un valor por debajo del 255, puesto que uno de los motores del barco es más lento que el otro y se pretende así compensar un poco esa diferencia.
- *Coord:* Struct que define las dos componentes de una coordenada: latitud y longitud. De aquí creamos el array *"ruta"* que contiene las coordenadas de toda la ruta, y las variables *punto_actual* y *punto_destino*.
- *Distancia:* distancia entre la ubicación actual del barco y el punto objetivo.
- *Rumbo_actual* y *rumbo_desdeado:* orientación actual del barco y orientación que debe seguir el barco desde el último punto alcanzado para llegar al siguiente; respectivamente.
- *Delta:* error entre el rumbo actual y el rumbo deseado.
- *Distancia_obstaculos:* variable donde se almacena la lectura del sensor de proximidad.
- *En_rumbo:* variable booleana para saber si el barco lleva el rumbo correcto.
- *Punto_alcanzado:* variable booleana que indica si el punto destino ha sido alcanzado.

6.2.2 Control de navegación.ino

Como ya se ha explicado en la parte de diseño, este archivo contiene el código principal del sistema y que corresponde al diagrama de flujo general:

Nota: se ha omitido el incorporar la declaración de las variables.

```
/* Lista de librerías */

/* declaración de variables*/

void setup()
{
```

```
      /* inicialización de variables: motores, compás, pinMode…
*/
      /*   inicio   de   la   comunicación   Serial1,   Wire,   y
SoftwareSerial*/

      activacion();
}
void loop()
{
      while(puntos_alcanzados<6)
        {
             establecer_destino();
             reportar_destino();
      while(!punto_alcanzado)
      {
                     if(comprobar_obstaculos())
                     {
                     frenar();
                     esquivar();
                     }
                     comprobar_rumbo();
                     myPID.Compute();
                     mover();
                     actualizar_posactual();
                     punto_alcanzado=comprobar_destino();
                     reportar_situacion();
      }
      reportar_alcanzado();
      puntos_alcanzados++;
       }
       reinicio();
}
```

6.2.3 Compas.ino

En la comprobación del rumbo hay dos partes que deben ser estudiadas:

La primera y más compleja, es sobre el tipo de movimiento que se ha escogido para la gestión del barco. Por un lado teníamos el controlador PID que se encargaba de ir produciendo la respuesta correctora para las desviaciones del rumbo, o el método sencillo, que consistía de ejercer una respuesta en base al grado de desviación actual. En caso de haber escogido el PID, en función del valor de la delta (error entre rumbo actual y deseado) la dirección de la salida del PID variará entre directa o inversa.

Y la segunda, más sencilla, es el margen de error que se tolera para el rumbo, por defecto, se ha marcado 3º como desviación permitida, valor que puede ser modificado si es necesario. Si el valor absoluto de delta es mayor que tres, entonces la variable *en_rumbo* pasa a tener el valor *false*.

Por otro lado, hay una posible variante de la implementación para el cálculo y gestión del rumbo.

El cálculo del rumbo para llegar al destino, se realiza en la función *establecer_destino()*. Entonces en dicha variante, el cálculo del rumbo necesario se haría en cada iteración del bucle while, y no en el *establecer_destino()* de fuera. Con esto lo que se conseguiría es tener un movimiento más dinámico, ya que cada vez se actuaría en función de un valor referencia de rumbo distinto, pero que a su vez, todos esos valores pretenderán llegar a un mismo destino. Así pues, se calcularía el error en función de este nuevo rumbo calculado y el barco se movería intentando corregirlo, con lo que además evitaríamos un posible problema ante una desviación muy grande del rumbo inicial.

6.2.4 Funcionamiento.ino

En el método *activacion()*, en el diagrama de flujo vemos que hay un nodo denominado *calentar gps*. Esta parte es importante, ya que el módulo gps tarda casi medio minuto en detectar los satélites en frío. Por ello, se realizan una serie de iteraciones iniciales para que una vez arranquemos el sistema, el gps ya esté recibiendo datos útiles.

Para el pase de coordenadas de la ruta, se ha implementado para introducirlas en el formato grados, minutos, segundos. Esto se ha realizado así porque si se introducían directamente en decimal, se perdía precisión en los últimos decimales, algo que luego se traduce en graves fallos de posición.

Además, como el sistema de comunicación cambió al bluetooth, se introdujo también dos posibilidades a la hora de crear la ruta: la primera, consiste simplemente en insertar las coordenadas de los distintos puntos; y la segunda, dedica la última componente del array a la posición actual desde la que se introduce la ruta, con esto conseguimos que, como debido a que en un alcance largo no tendremos comunicación con el barco, aseguraremos el hecho de que vuelva hasta nuestra posición y no tengamos que recurrir al mando radiocontrol.

Finalmente, tanto para el método *comprobar_destino()*, como para *comprobar_obstáculos()*, al igual que con el rumbo, hay establecidos unos valores como margen de error que pueden ser modificados según el comportamiento que deseemos para el sistema. Si deseamos que el barco pase más cerca de los puntos destino, podemos reducir la cifra de la distancia referencia. Y para el sensor de obstáculos, se puede ampliar o reducir según el nivel de alcance que deseemos que tenga.

6.2.5 GPS.ino

Por lo que respecta a los métodos relacionados con el gps, hay un valor que debemos tener en cuenta:

"while (millis() - start < 500)"

En esta línea, se comprueba el tiempo entre lectura y lectura del GPS para obtener datos nuevos. El valor numérico expresa en milisegundos el periodo mínimo para obtener datos nuevos. Este valor debe ser como vemos relativamente alto, esto sucede porque los propios satélites poseen como un máximo de accesos cada cierto tiempo, por lo que si lo excedemos estaremos unos segundos sin recibir nuevos datos de posición. Y eso en el funcionamiento del sistema puede ser crítico, pues no puede estar el barco desplazándose e interpretar que se encuentra en un punto que ya ha pasado.

6.2.6 Movimiento.ino

Para el sistema de movimiento tenemos tres métodos: el encargado de evadir los obstáculos, el de movimiento "normal" y el de frenar.

Para la implementación del *esquivar()*, se ha querido simular cierto carácter autónomo del sistema, permitiéndole "elegir" hacia qué lado esquivar el obstáculo mediante un carácter aleatorio. Dicho carácter puede tomar el valor 1 o 2, y en función de este, realizará las maniobras hacia la izquierda o la derecha.

El resto del algoritmo es sencillo. Primero, el barco girará durante un segundo y medio hacia un lado, mientras comprueba si siguen habiendo obstáculos en su camino. Una vez no detecte más obstáculos, avanzará durante dos segundos y medio para alejarse un poco de estos. Una vez hecho todo esto, se recalculará la nueva posición actual y el rumbo necesario, pues el barco se habrá salido de su ruta inicial al evadir el obstáculo de su camino.

El método encargado del movimiento, lo gestionará en función de la desviación de rumbo que lleve el barco. Partiendo de una constante que representa la velocidad para que gire un grado, se multiplicará dicha velocidad por el error que lleve el barco y se descontará dicha velocidad al motor correspondiente para que el rumbo sea corregido.

7. Montaje

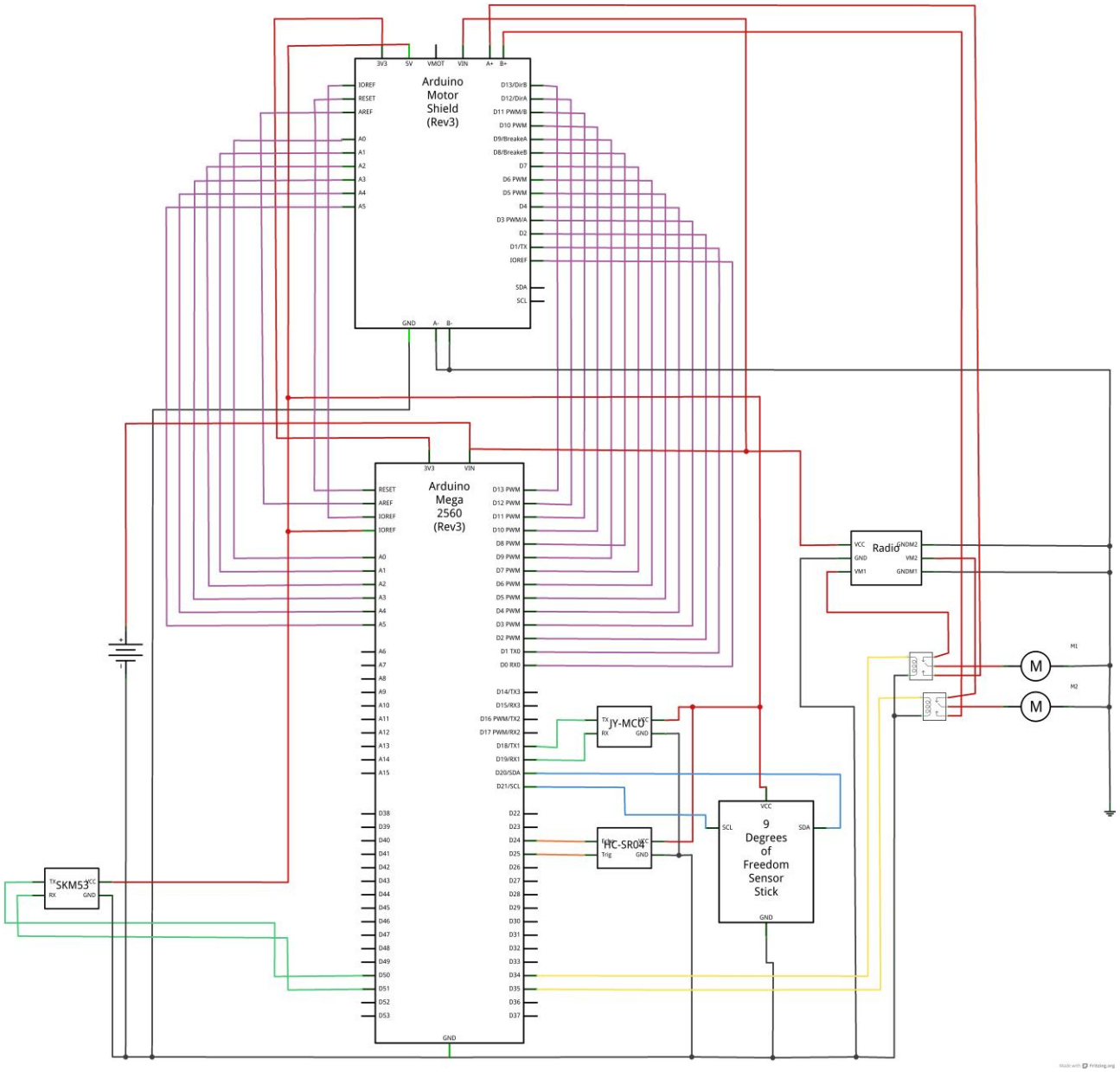


**Imagen con las distintas conexiones de los dispositivos del sistema.*

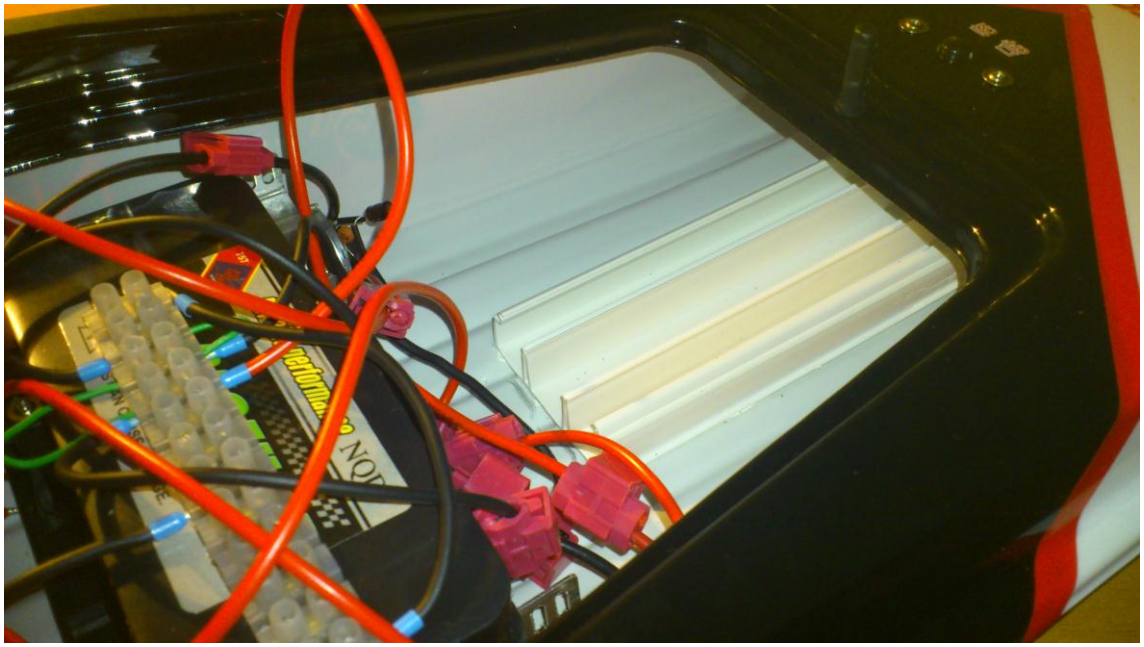
**nota1: el shield de motores reflejado en el esquema no es exactamente el mismo que se ha utilizado en el proyecto, sin embargo, como es un modelo que permite la gestión de dos motores, es válido para reflejar la misma funcionalidad que el original.*

**nota2: los relés en la imagen se han reflejado de forma separada, sin embargo, en el montaje real estos constituyen un mismo módulo.*

El montaje del sistema se ha realizado mediante un sistema bastante original, cuya justificación se verá en el apartado de pruebas. Centrándonos en el propio montaje, los distintos dispositivos se han introducido dentro del barco encima de una pieza de canaleta.

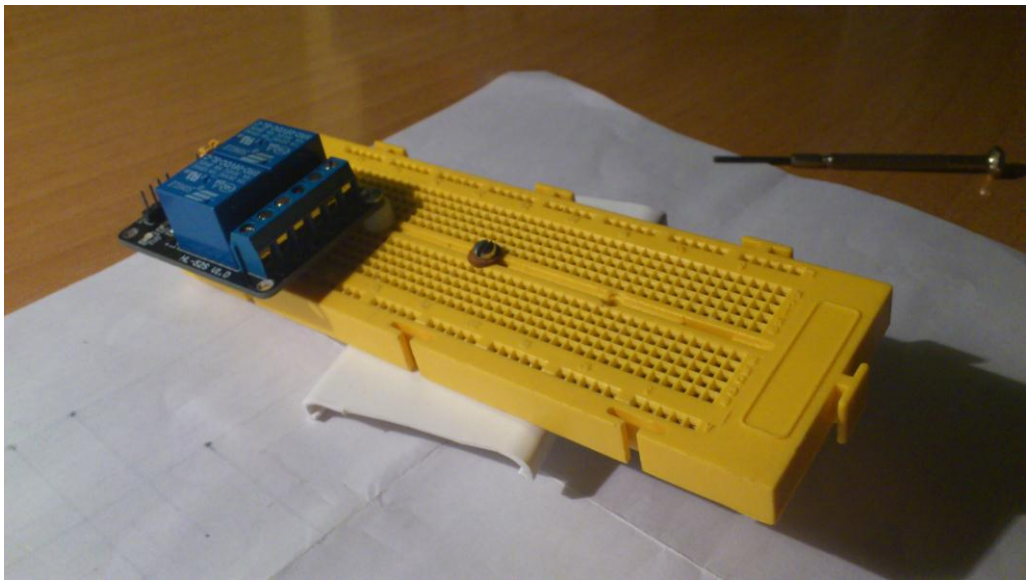
La base de la canaleta, se ha pegado a la superficie del casco de la lancha, y la parte de la "tapa" se ha dejado fuera. A continuación, se ha cortado dicha tapa en 3 fragmentos, de los cuales, cada uno lleva atornillado un componente o grupo de componentes distinto.

La pieza que va más pegada al morro corresponde a la que lleva al IMU, la siguiente, la placa board con el módulo de relés y el GPS, y finalmente, la última con el Arduino Mega y el shield para los motores justo encima. El bluetooth va conectado directamente a Arduino y queda sujeto entre los cables del sistema.



Una gran incomodidad, más que problema, que ha habido durante el montaje ha sido la falta de maniobrabilidad para poder manipular los distintos dispositivos en el barco, ya fuese para sacarlos o meterlos, reconectar algún cable, etc. Esto ha ocurrido debido

sobre todo a la caja que guarda la electrónica original, que unido a todo el cableado nuevo dificultaba mucho el trabajo.

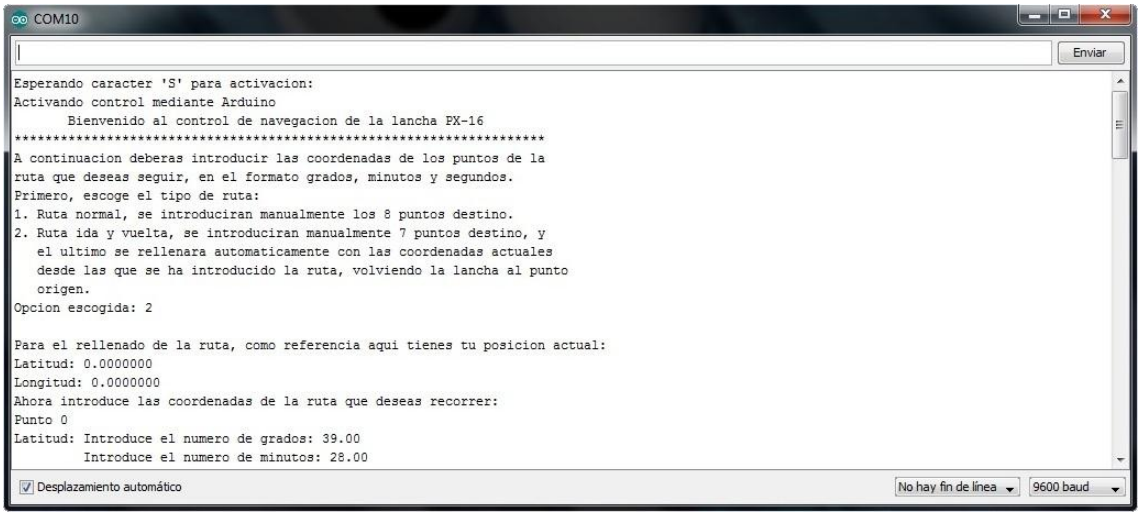


Al final, a la hora del montaje, se descartó el incorporar el módulo de ultrasonidos. Primero, por la falta de una caja hermética que lo protegiese de una forma 100%, teniendo en cuenta que debía tener cierta parte sin cubrir para los sonares. Por otro lado, reflexionando sobre los entornos en los que iba a probarse, ninguno conllevaba la presencia real de obstáculos que pudiesen comprometer la seguridad del barco, por lo cual se decidió dejar esta función como un plus.

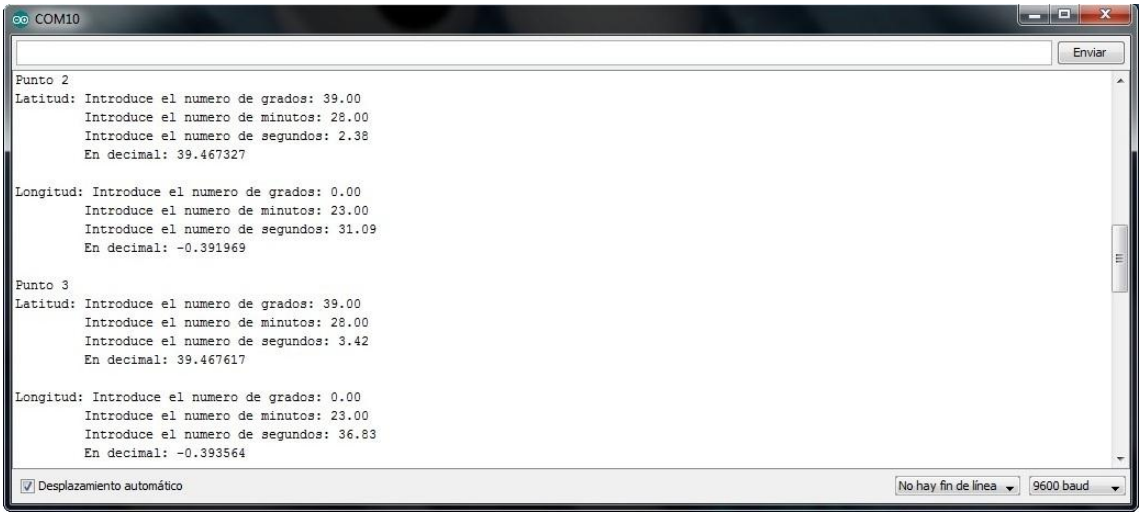
8. Fase de pruebas

8.1 Pase de coordenadas

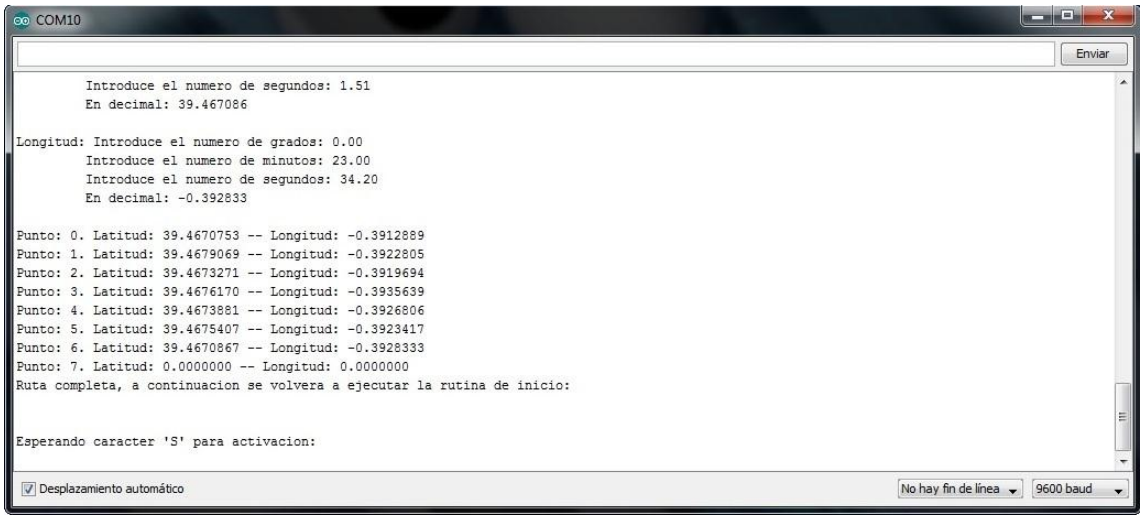
En la parte de pruebas correspondiente a esta etapa, se probó el correcto funcionamiento del menú de presentación, la selección de cualquiera de las dos opciones del menú, y la precisión de la traducción de las coordenadas de grados, minutos y segundos a sistema decimal.



Como vemos, los valores de la posición en la que nos encontrábamos son todo ceros, debido a que ahí el GPS aún no estaba conectado.



Después de la inserción de una parte de una coordenada (latitud o longitud) nos aparece su representación en decimal por si junto con Google Maps queremos corroborar que la coordenada resultante es correcta.

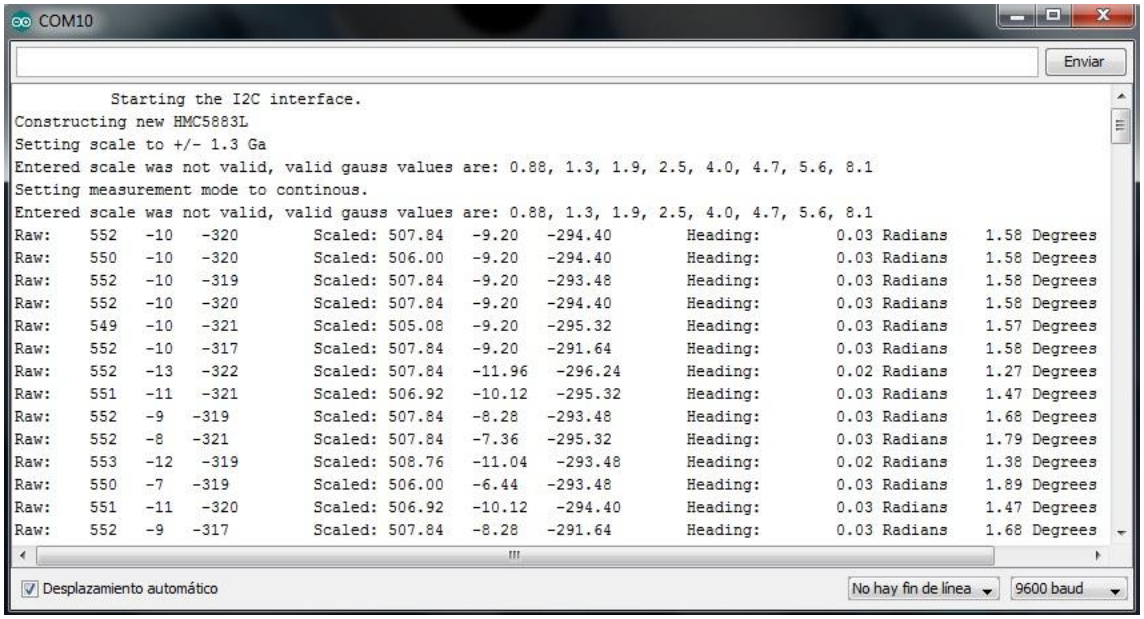


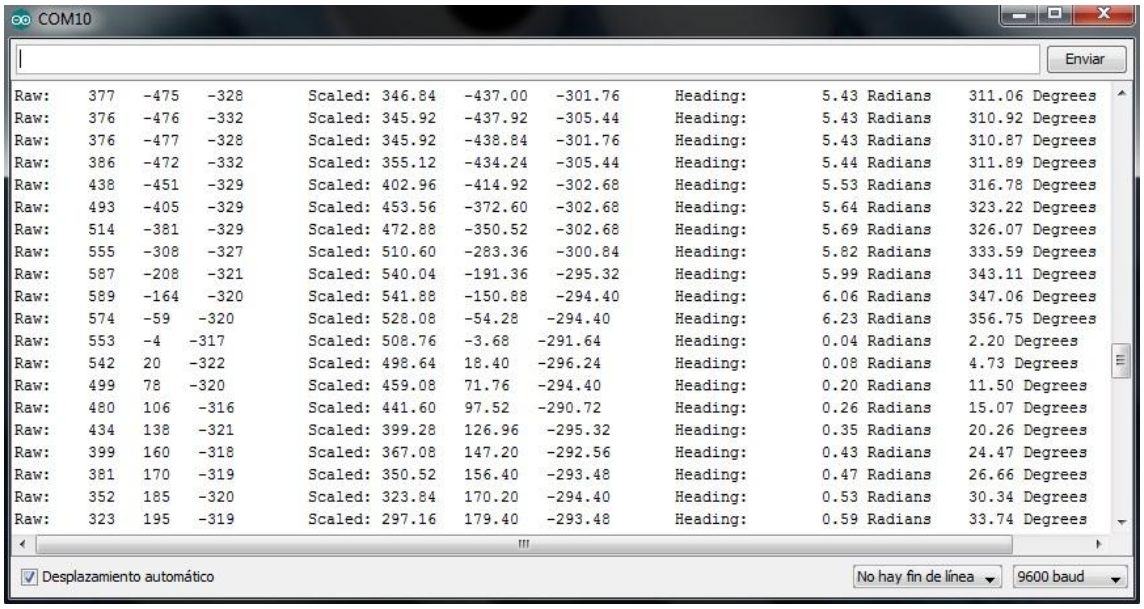
Finalmente, se muestra por pantalla toda la ruta y se prueba el método *reinicio()*. Éste es el que se encargará de una vez finalizada la ruta, se vuelva a quedar en espera el sistema para una nueva activación y ponga la variable *puntos_alcanzados* a 0.

**Puntos escogidos para la prueba del pase de coordenadas.*

8.2 Compás

Para la prueba del funcionamiento del compás se probó el ejemplo de la librería con la que se ha trabajado, y de donde se extrajo el método para la lectura de este.





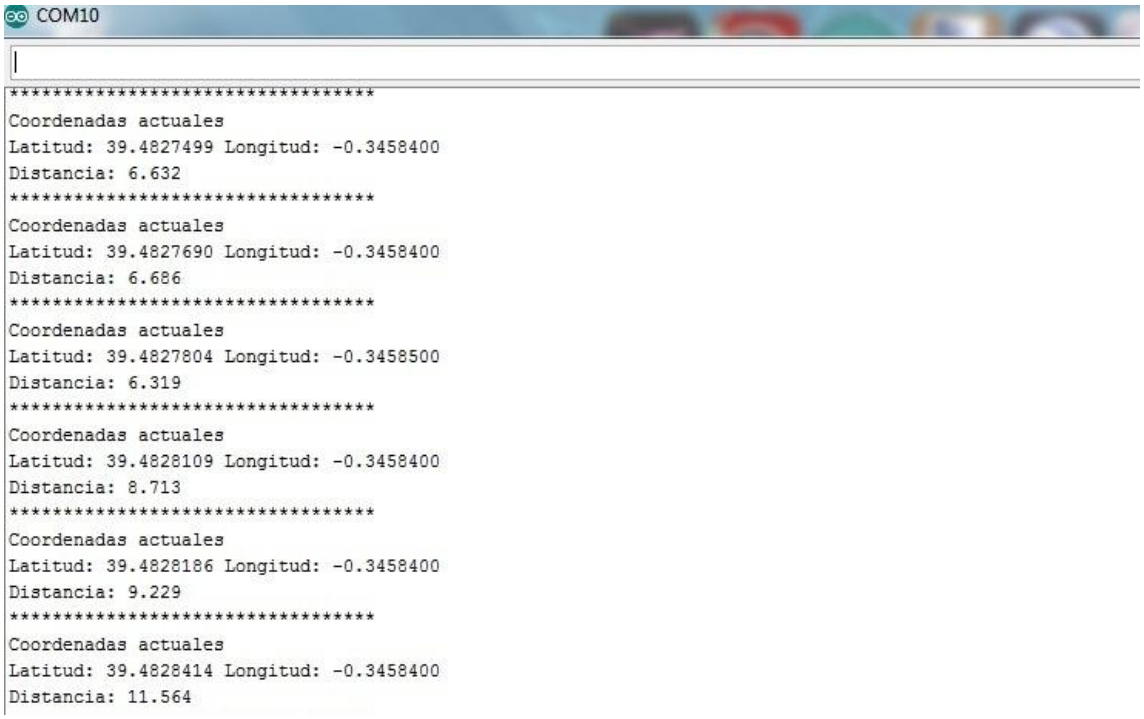
Además, para comprobar que se estaban realizando bien las mediciones del compás y no estaba siendo distorsionado por ningún otro aparato cercano, se verificó con una brújula tradicional si aproximadamente los datos eran similares.

8.3 GPS y distancia

En esta fase de pruebas se probaron tanto el funcionamiento del GPS, como el método que se encarga de calcular la distancia hasta el destino. El resultado:

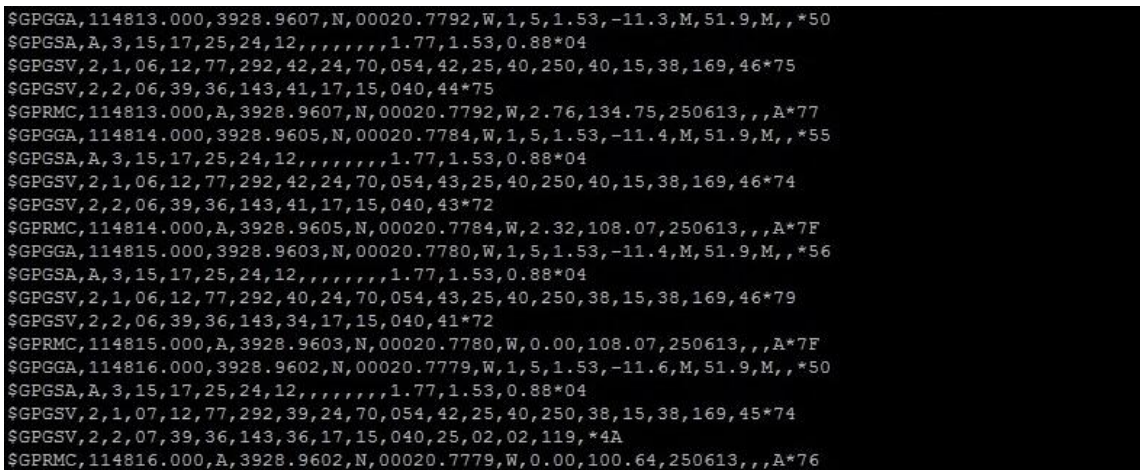
En esta imagen es donde mejor se aprecia la idea de dado un origen y un destino, intentar llegar a él. Aun sin trabajar con el compás que marcaría el rumbo, el resultado es más que favorable.

Además también se probó el hecho de que, en caso de llevar un rumbo erróneo, cómo repercutiría esto en la distancia. Así se continuó la prueba una vez "alcanzado el destino" (ya que aunque la distancia no es 0 totalmente, en el control de navegación se da por alcanzado un punto al estar a una distancia referencia de él como pueda ser uno, dos o medio metro), volviendo al punto desde el que se había partido, y así, ver como la distancia crecía también de nuevo.



Uno de los problemas que aparecieron durante la fase de pruebas fue que había ocasiones en las que los valores del GPS se quedaban estancados durante ciertos segundos, lo que luego en la práctica final podía ser una situación crítica. La causa principal que se pensó, es que debía fallar la conexión con los satélites.

Para comprobar este factor, lo que se hizo fue probar a leer directamente las tramas que enviaba el GPS con el protocolo NMEA, ya que en estas aparecía información sobre los distintos satélites con los que el dispositivo establecía conexión, y además permitían seguir obteniendo datos sobre la posición.



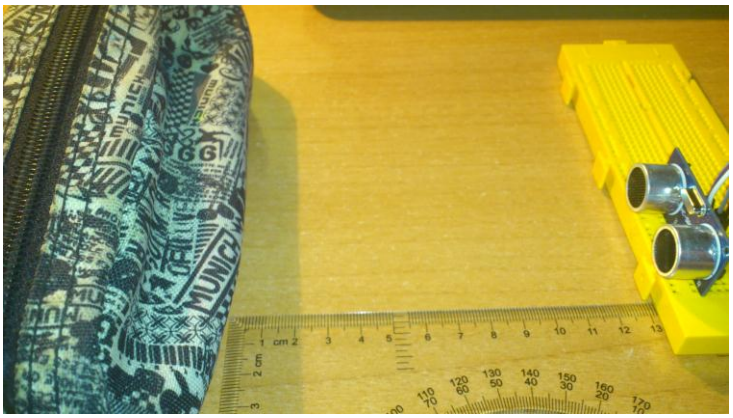
El caso es que descartando el problema de los satélites pues se recibía datos de estos. La solución llegó con la realización de otra prueba. El problema venia dado por la cantidad

de movimiento con la que nos desplazábamos. Es decir, al parecer, para que el GPS reciba cambios significativos en la posición actual, se debe llevar un movimiento con una velocidad mínima. Si al andar para comprobar el cambio de posición se llevaba un ritmo lento, se producían esos estancamientos en los valores; sin embargo, si se aligeraba el paso los valores en ningún momento dejaban de cambiar y el sistema funcionaba perfecta y continuamente.

8.3 Sensor de proximidad

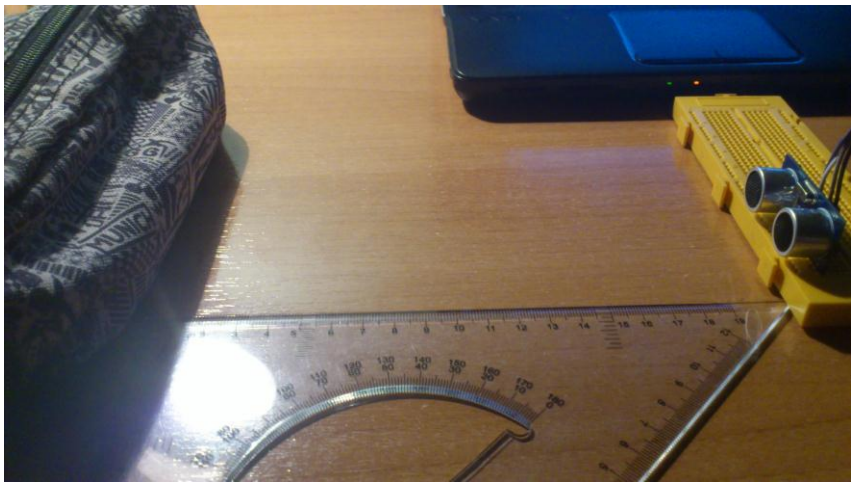
Para la detección de obstáculos se plantearon 4 casos, en los cuales un obstáculo (mi estuche) se encontraba en cada caso a una distancia distinta. Entonces, cuando se detectaba por el puerto serie la presencia de cualquier tipo de dato, el sensor actuaba y mostraba por pantalla la distancia obtenida, o por el contrario, si el obstáculo se encontraba a menos de cierta distancia, en este caso 5 cm, emitía un aviso. A continuación se encuentran fotografiados los distintos escenarios y el resultado de lo mostrado por el terminal del puerto serie.

Caso 1:

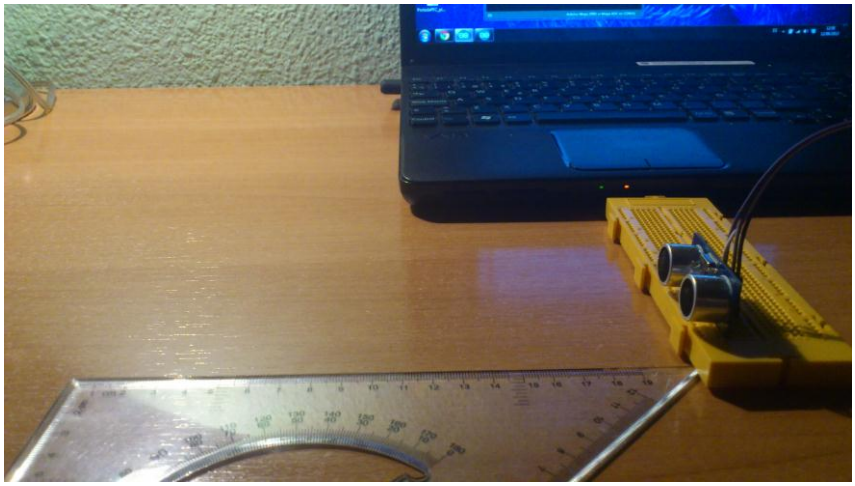


Caso 2:

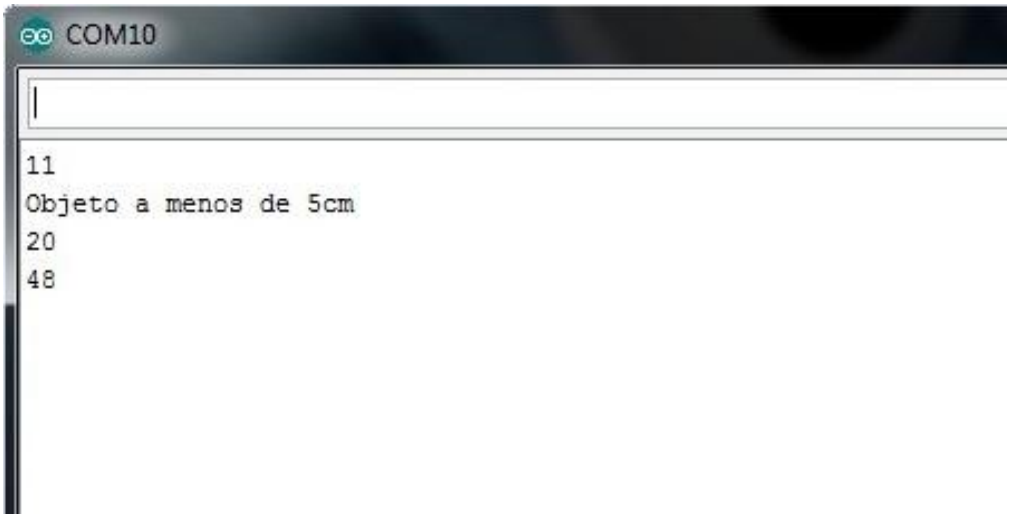
Caso 3:



Caso 4:



Finalmente, esto es lo que se mostró en pantalla:



Como vemos, en el primer caso hay un pequeño error de unos centímetros, pero en el resto, los resultados coinciden perfectamente:

1) Primer caso => 13cm – muestra 11.
2) Segundo caso=> Menos de 5cm – muestra el aviso.
3) Tercer caso=> Más de 19 cm – muestra 20, muy cerca de la medida real, el fallo a lo sumo puede ser de un centímetro, centímetro y medio.
4) Cuarto caso=> Sin obstáculo cercano- muestra 48, que medido posteriormente con un metro, detectó el teclado del ordenador de sobremesa.

8.4 Control de motores

En este caso, la fase de pruebas se centró en comprobar el funcionamiento del shield de Arduino. Para ello, simplemente se probó uno de los ejemplos que incorporaba la librería conseguida del playground de Arduino con un motor prestado por la universidad y que serviría para probarlo antes de tratar directamente con los del barco.

Dicho ejemplo simplemente hacia girar el motor en ambos sentidos posibles gracias al puente H del chip. Primero, al girar en un sentido se aumentaba progresivamente la velocidad hasta alcanzar el valor 255; y luego se disminuía progresivamente hasta 0 mientras el motor giraba en sentido contrario. La velocidad del motor era manipulada mediante el método *SetSpeed(),* que modifica la salida del pin PWM y así la velocidad del motor.

8.5 Fase de pruebas final

Esta es la parte en la que se encontraron los mayores problemas.

Primero, antes de probar todo el sistema en el barco, se creó una beta simplificada del control de navegación, que básicamente consistía en ir de un punto a otro, sin la parte de código correspondiente a los motores. La prueba consistió en, un servidor, con el portátil en una mano y la placa board con los distintos dispositivos montados, realizar el comportamiento que tendría el barco, tratando de corregir mi propia orientación según lo que marcase el compás.

El resultado fue satisfactorio, pues al acercarme a la zona del punto establecido, a una distancia de unos 3m aproximadamente, siendo la referencia de 4m (una distancia muy grande, pero como se trataba simplemente de una prueba la tomé por válida), el sistema dio el punto por alcanzado, la ruta por concluida, y reinició el sistema a la espera de una nueva activación.

***Hay un pequeño "fallo" en la distancia hasta el destino, y es que la coma está corrida dos lugares de más a la izquierda por un error sin importancia en el println. Sin embargo se ve perfectamente el comportamiento del sistema en cada iteración.*

Una vez probado el sistema de este modo, se pasó a otra prueba necesaria, el comportamiento del motor de pruebas junto con el compás. La idea era que, al girar el compás, el motor debía actuar como actuaría el sistema para recuperar el rumbo. Sin embargo, al realizar una prueba simple de lectura del compás se vieron extraños resultados. Así se descubrió el segundo problema.

El problema era que, al igual, otros muchos otros dispositivos electrónicos, o una brújula normal, el compás se veía afectado por el ruido magnético del imán del motor de pruebas. El ruido, como digo no era el ruido eléctrico que pueda producir un motor al girar, si no simplemente estando sin pase de corriente, el imán distorsionaba totalmente al compás.

De las mediciones que se tomaron, a unos 30cm las mediciones del compás eran válidas; a 26cm se iban cerca de 10 grados…y a unos 15cm se iban 150 grados.

Se estudiaron dos posibilidades para tratar este problema y darle solución.

La primera, fue recubrir los motores con una estructura de algún material ferromagnético que actuara como una jaula de Faraday. El material ferromagnético, más que bloquear el paso de las líneas de campo lo que haría sería proporcionar un camino de baja resistencia magnética a través del cual podrían fluir, haciendo que la intensidad del campo magnético fuera de la estructura fuese menor.

Sin embargo esta idea presentaba diversos problemas. El primero, el poder crear una estructura así sin comprometer o sobrecargar el peso en la parte trasera. Sobrecargarla alimentaría un problema que se ha querido solucionar con el control de los motores por software, y es el hecho de que el morro de la lancha se levanta cuando gana velocidad. Esto repercute en las medidas del compás, que para tomar mejores medidas debe estar lo más paralelo posible a la superficie.

Además otro problema sería que si creamos una estructura cerrada o prácticamente cerrada alrededor de los motores, estos podrían sobrecalentarse y quemarse.

El otro método para corregir la influencia de los imanes era mucho más sencillo, y consistía simplemente en poner el compás lo más alejado posible de los motores. Esta era la solución más viable puesto que no había ningún inconveniente que pudiese comprometer el sistema en el futuro. Así como medida para alejarlo de los motores se realizó como pone en el apartado de montaje, una guía con canaleta, que permitía colocar el compás lo más cerca posible del morro de la lancha.

Una vez solventado el segundo problema pasamos a la prueba del sistema montado. La última se realizó con el sistema ya montado dentro del barco. Antes de probarlo con el control de navegación, se cargó en la placa el sketch ejemplo de la librería del shield de los motores. El resultado de esta prueba fue, que a los pocos segundos de conectar la batería, el chip del shield empezó a calentarse demasiado, hasta el punto de quemar con el simple contacto. Visto el estado crítico del chip, se desconectó rápidamente la batería del circuito.

Como solución se intentó modificar un poco el circuito para ajustar las masas del circuito y algunos cables. El resultado continuó siendo el mismo, y es que al probar este simple sketch, el chip del shield se sobrecalentaba demasiado, por lo cual no se pudo realizar una prueba final con todo el sistema ya acabado, dado que al parecer, el shield carecía de fuerza para mover los motores de la lancha.

9. Conclusiones y trabajos futuros

A nivel personal, la realización de este proyecto me ha servido para poner en práctica muchas de las cosas estudiadas a lo largo de la intensificación, descubrir más de las incontables posibilidades que ofrece Arduino y descubrir qué es lo que realmente me atrae del mundo de la informática.

De todas las etapas en las que se ha dividido el proyecto, las etapas de análisis, diseño junto con la implementación, han sido con diferencia las que más me han motivado y agradado. Por tanto, una vez acabado este trabajo, me encaminaré al estudio de una nueva intensificación: software, donde se trabaja mucho más en estos aspectos.

De lo realizado en el proyecto, la parte que considero más sufrida o que me ha desagradado más ha sido toda la fase de pruebas. Creo que la gran dificultad que ha tenido ha sido quizá por falta de experiencia en este tipo de trabajos y que en el futuro no sería tan pesado o complejo. Sin embargo, los problemas surgidos en esta fase, que en un principio era la motivación principal del proyecto, hicieron que se convirtiera en una desmotivación, lo que hizo que me decantase y diese cuenta de que donde realmente había disfrutado del proyecto había sido en toda la parte del desarrollo del software.

Respecto a trabajos futuros, a nivel personal, como ya he dicho pretendo estudiar y especializarme en software; y con respecto a lo que ha sido este proyecto, se podría ampliar o utilizar el control de navegación para sistemas más grandes. Por ejemplo, algunas ampliaciones futuras que se podrían realizar serían: la adhesión de nuevos sensores como un sonar o una cámara; la utilización del control de navegación para la realización de un sistema más completo, capaz de realizar alguna de las tareas que puede realizar un Roboat (es decir, hacer una versión “light” de alguna de sus posibilidades), la adaptación de este control a otro tipo de robots móviles o el perfeccionamiento de este mediante un filtro de Kalman, la implementación de la parte del shield de la SD o el uso del acelerómetro para trabajar también no solo con la información de aceleraciones sino para mejorar las mediciones del compás.

10. Bibliografía

Web de Arduino :

http://arduino.cc/es/

Wikipedia :

www.wikipedia.org

Playground de Arduino :

http://playground.arduino.cc/

Web dedicada al trabajo con Arduino y otros sistemas open-hardware :

http://www.arduteka.com/

Datasheet del chip HMC5883L (compás digital) :

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf

Datasheet del chip SKM53 (GPS) :

http://www.fraskito.net/?q=system/files/u1/SkyNav_SKM53_DS.pdf

Explicación protocolo NMEA :

http://www.gpsinformation.org/dale/nmea.htm

Información del controlador PID :

http://www.dia.uned.es/~fmorilla/MaterialDidactico/El%20controlador%20PID.pdf

Influencia de un imán sobre un compás digital :

http://robotics.stackexchange.com/questions/442/how-can-the-dynamic-effects-of-motor-current-on-a-digital-compass-be-characteriz

Web del entorno Dia :

https://wiki.gnome.org/action/show/Apps/Dia?action=show&redirect=Dia

Web del entorno Fritzing :

http://fritzing.org/pages/welcome/

Bus I2C :

http://robots-argentina.com.ar/Comunicacion_busI2C.htm

Tratamiento del ruido magnético :

http://www.respuestario.com/como/como-aislar-un-iman