Facultad de Ciencias

Diseño una librería que controle robots siguelíneas basados en Arduino, memoria

Design of an Arduino-based line-following robot driver, scholarship's memoir

Memoria de prácticas
realizadas por un alumno del

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Álvar Tapia Ruiz

Director: Domingo Gómez Pérez

Inicio: Febrero 2019
Fin: Mayo 2019

Agradecimientos

A Domingo, por la oportunidad de hacer esta beca,
a mis padres, que han aguantado mis charlas sobre robots,
y a mis colegas, con los que me he reído tanto en el proceso.

Diseño una librería que controle robots siguelíneas basados en Arduino, memoria

Resumen

palabras clave: Arduino, robot, coche, siguelineas

Debido a la adquisición de robots de marcas diferentes, la asignatura de Algorítmica y Complejidad en 2019 necesitaba de una librería que permitiera controlar robots de marcas diferentes con el mismo conjunto de instrucciones.
De esta manera el usuario podría despreocuparse en gran medida del robot que estaba utilizando de forma que no necesitara trabajar con un robot concreto, sino que su código podría funcionar en cualquier robot al cambiar unos pocos parámetros.

Los detalles del ensamblado de los robots recién adquiridos, del diseño de la librería, del software utilizado para implementarla, de las dificultades encontradas, soluciones implementadas y opciones propuestas se encuentran en esta memoria.

Se consiguió que los robots de marcas diferentes funcionaran con el mismo conjunto de instrucciones, pero no se ha conseguido que el robot recabase información del entorno de la manera que requerían las prácticas de Algorítmica y Complejidad.

Se proponen maneras de continuar con el proyecto, al igual que alternativas al planteamiento de las prácticas, para que se pueda seguir trabajando aprovechando el esfuerzo ya realizado y conseguir cumplir con los objetivos propuestos.

Design of an Arduino-based line-following robot driver, scholarship's memoir

Abstract

keywords: Arduino, robot, car, linetracker

Due to the acquisition of robots of different branding, Algoritmica y Complejidad subject in 2019 needed an interface library that allowed working with different brand robots using the same source code.
With this library, the user would not have to worry (almost at all) about the robot he was given, because his code will work in any robot available if a few key parameters are modified.

The details of the assembly of the recently acquired robots, of the library design, of the implementation software, of the found difficulties, implemented solutions and proposed options are found in this memoir.

It was achieved that different brand robots worked with the same source code, but showing the robot how to read the environment as needed in Algorítmica y Complejidad assignments was not possible in the time given.

There are recommendations on how to continue the project, as well as alternative assignments to ease the development of the library, in hopes of seeing this project meet its initial objectives.

Índice

1 Introducción

Esta es la memoria sobre el proceso que se ha seguido para construir y programar robots siguelíneas que recorren grafos de manera autónoma.

El código comentado y fotografías se encuentran en GitHub, así como un histórico del trabajo realizado en el proyecto a lo largo del desarrollo de la beca.

1.1. Esquema del desarrollo de la beca

1.1.1. Montaje de los robots

Se dedicaron 20 horas al montaje y estudio de los robots, incluyendo el descubrimiento de las funcionalidades de los componentes, planificación de las conexiones, construcción y ensamblado, además de las reparaciones que se hayan necesitado a lo largo del desarrollo de la beca.

1.1.2. Pruebas de los robots

Se dedicaron 5 horas a comprobar que todos los elementos de los robots Clónicos estaban bien conectados, que respondían como se esperaba, y que se integraban de manera correcta.
Después de cada reparación también se realizaban pruebas, para comprobar que las funcionalidades se mantenían intactas.

1.1.3. Estudio de código legado

Se han necesitado 10 horas para entender y documentar el código legado por la asignatura de Algorítmica y Complejidad. La falta de documentación y comentarios de las bibliotecas hizo necesaria una inversión de tiempo mayor a la esperada.

1.1.4. Estudio de requisitos

Se invirtieron 5 horas en entrevistas para conocer los requisitos específicos del proyecto, de manera que el código se adaptara a las prácticas lo máximo posible.

1.1.5. Diseño de las bibliotecas

Se invirtieron 15 horas en el diseño y refactorización de las bibliotecas Bluetooth, Siguelíneas, Infrarrojos, Morse, Robot y Cnosos.

1.1.6. Documentación de las bibliotecas

Se dedicaron 10 horas a documentar las interfaces y a comentar todo código que se consideraba confuso.
Esta labor hará que proyectos futuros sobre estas bibliotecas sean más fáciles de abordar.

1.1.7. Implementación de las bibliotecas

Hubo una inversión de 20 horas en conseguir que todas las funcionalidades de las bibliotecas se comportaran como se esperaba.

1.1.8. Pruebas de las bibliotecas

Se dedicaron 5 horas en diseñar las pruebas de las bibliotecas y los criterios que debía cumplir el robot para considerar esas pruebas como superadas.

1.1.9. Escritura de la memoria

El tiempo que ha llevado la escritura de la memoria es difícil de medir, debido a que se ha ido trabajando en ella a lo largo de todo el proyecto.
Álvar estima que se han dedicado unas 40 horas a la realización de esta memoria. Recabar la información y presentarla, mientras se aprendía a usar el software de edición de texto LaTeX llevó más tiempo del esperado.

1.1.10. En resumen...

Se ha trabajado en este proyecto un total de 130 horas.
Las partes que Álvar ha encontrado más duras han sido el estudio del código legado y la implementación de las bibliotecas según los requisitos, mientras que las partes que Álvar ha considerado más laboriosas han sido el montaje de los robots y el desarrollo de la memoria.

1.2. Uso de la biblioteca

La biblioteca se encuentra en esta carpeta de GitHub.
Estas bibliotecas están diseñadas para aprovechar características de la aplicación Arduino IDE. Las bibliotecas funciona con cualquier aplicación que permita trabajar con código Arduino, pero se recomienda este programa por funcionalidades como poder obtener ejemplos de uso de las bibliotecas.

1.2.1. Instalación de la biblioteca en Arduino IDE

Tal y como está estructurado el GitHub, en primer lugar hay que descargarse el proyecto entero o clonarlo en el repositorio local Git de la máquina.

Una vez el proyecto se encuentre en nuestra máquina, en la carpeta *Codigo* se debe copiar la carpeta *RobotAlgoritmica*.
La carpeta ha de pegarse en la carpeta *Documentos* > *Arduino*, o en la carpeta *libraries* en la configuración de la aplicación Arduino.

Realizadas todas las instrucciones, la próxima vez que se abra la aplicación Arduino IDE, tanto en *Programa* > *Incluirlibrera* como en *Archivo* > *Ejemplos* debería aparecer en la lista la biblioteca RobotAlgoritmica.
A partir de ahora la biblioteca está lista para usar.

2 Construyendo el robot

2.1. Recibiendo los componentes

El proyecto empieza con dos robots montados y varios componentes sin clasificar.

Lo primero que hizo Álvar fue investigar cómo estaban ensamblados los robots, y qué partes pueden servir como referencia para el montaje del robot restante.
Había un robot con los mismos componentes que los proporcionados, y otro con componentes distintos.

La primera labor sería reconocer los componentes que se tenían y ordenarlos de manera que estuvieran a mano cuando se necesitaran.

2.2. Un momento, esto no es Elegoo

La empresa que vendió estos robots a la Universidad de Cantabria, Inven, trabaja con unos planos y componentes que son parecidos, pero no los mismos, a los de Elegoo. Los planos e instrucciones que se proporcionaron eran de Elegoo, no de Inven.

Por lo tanto Álvar se puso en contacto con Inven, y la empresa facilitó instrucciones para el montaje de los componentes.

2.2.1. Elegoo y Clónicos

Desde entonces, Álvar empezó a diferenciar los robots según características comunes.
A partir de ahora se hablará de robots Elegoo cuando se mencionen robots vendidos por Elegoo, y de robots Clónicos cuando se esté trabajando con robots vendidos por Inven. Ambos términos se encuentran en el anexo A para mayor acceso.

2.3. Preparando los motores

Álvar ha tenido que soldar los cables de los motores de los robots Clónicos. Se menciona porque los robots Elegoo tienen los cables de los motores ya soldados y preparados para instalar en la Arduino.
Como se han tenido que soldar los cables de conexión de los motores a mano, Álvar ha decidido que dichos cables van a seguir un patrón particular:
si el cable rojo del motor se conecta al positivo de una pila, y el cable negro al negativo de esa misma pila, el motor se moverá de manera que el robot avance. Si, por el contrario, se invierten las polaridades, el giro permitirá que el robot retroceda.

El chasis nos obliga a poner motores de dirección inversa (los contactos «se miran» el uno al otro). Por ello, este patrón obliga a conectar un cable negro y un cable rojo en cada zócalo del driver de motores.
Se han estañado estas conexiones para hacer más fácil la instalación de los cables.

2.4. Driver de motores

Es la placa que más problemas ha dado en el montaje del robot. Al principio Álvar consideró conectar cada cable en un zócalo, pero el inconveniente era que se tenían 8 cables de motores para 4 zócalos del driver.

Trabajando en el robot Álvar se dio cuenta de que los motores pueden ir en parejas: para girar al lado derecho, por ejemplo, se necesita que los motores situados en la izquierda del robot giren hacia delante, y que los motores de la derecha giren hacia atrás.
Con este conocimiento, se pudieron conectar todos los cables al driver, dado que en cada zócalo hay 2 cables. El driver no está preparado para funcionar de esta manera (ahora los dos motores tienen que compartir la potencia que se supone que iría para uno solo), pero es la única manera de conectar 4 motores a 4 zócalos.

2.4.1. Pruebas del driver

Con 4 pines de la Arduino conectados a los 4 pines de control del driver, Álvar diseñó un pequeño programa para probar los motores.

Resulta que la tensión necesaria para mover cuatro motores a la vez tiene que ser considerable, y cualquier perturbación en las conexiones hacía que alguno de los motores dejase de girar.
También hay que tener en cuenta que se usó un portapilas de 4 pilas (unos $6V$) con pilas usadas, factor que reduce el voltaje producido. El voltaje recomendado para trabajar con motores es de $12V$, pero no había forma de generar esa diferencia de potencial en el transcurso de las pruebas. Esta situación provocaba que los motores girasen de muy lentamente, o que simplemente se parasen.
Por lo tanto, se optó por realizar las pruebas utilizando la salida de $5V$ de la Arduino y conectándola a la alimentación del driver, con la Arduino siendo alimentada por el ordenador con un cable USB.

Con esta configuración se obtenían mejores resultados.
Al finalizar este test se pudo asegurar que los motores funcionaban, y que se comportaban de la manera esperada.

2.5. Sensores siguelíneas

Los sensores no dieron mayores problemas. Tienen un pin de corriente, uno de tierra, y otro que transmite una señal digital que indica si la zona está «oscura» (tensión alta) o «iluminada» (tensión baja).

2.5.1. Pruebas de los sensores

Para conocer qué situaciones se identificaban con tensión alta o baja en el sensor, se conectaron dos motores y un sensor a la Arduino a modo de prueba.

Se consideró que los motores respondían un poco lento a los cambios en el sensor, por lo que se intentaron mejorar esos tiempos con otro código.

2.5.2. Pruebas escribiendo en registros de Arduino Uno

Hasta el momento se han utilizado las funciones $digitalRead(PIN)$ y $digitalWrite(PIN, HIGH|LOW)$ para detectar y escribir la tensión del pin, respectivamente, pero estas funciones son macros. Están pensadas para que el código sea más flexible, y se pueda utilizar en otras Arduino cambiando las menos instrucciones posibles.

En nuestro caso, como siempre se está trabajando con Arduino Uno, podemos estudiar cómo funciona su sistema de registros reg y utilizar operaciones AND y OR en los mismos registros para agilizar la escritura u obtención de datos.
Los programas de prueba son muy pequeños todavía, pero se nota una pequeña mejora en el tiempo de reacción utilizando operaciones lógicas directamente en los registros.

No se accederá directamente a los registros

Se ha considerado que la mejora en tiempo que proporciona acceder a los registros de forma directa no es importante en este proyecto.
En proyectos con tiempos más ajustados, es mejor centrase en una placa y optimizarla al máximo, pero como éste es un proyecto en el que el tiempo de reacción puede ser de milisegundos (cuando el procesador de Arduino puede trabajar en el orden de los microsegundos), y como al trabajar directamente en los registros se pierde mucha legibilidad de código, Álvar cree que es mejor abandonar la idea de acceder directamente a los registros de Arduino.

2.6. Nueva shield

Inven, después de conocer más detalles sobre nuestro proyecto, nos han recomendado trabajar con una «Motor Shield» shi para la Arduino, de manera que se minimice el número de cables que conectan todos los componentes del robot.

2.6.1. Nuevo driver de motores

El driver de motores es idéntico en términos del número de pines para los motores y alimentación externa. La diferencia es que la shield tiene pines digitales dedicados a controlar los motores, lo que ahorra cables, confusiones, y problemas de conexión.

Los pines 10 y 11 indican la velocidad del motor, siendo $HIGH$ la «velocidad máxima», y LOW, «parada».
Los pines 12 y 13 representan el sentido de giro de los motores. Cambiando los valores digitales del pin, se cambia el sentido de los motores.

2.6.2. Más sensores

Inven nos ha proporcionado sensores infrarrojos adicionales para los siguelíneas. Los grafos que se quieren realizar son demasiado complejos para poder utilizar únicamente 2 sensores; vamos a utilizar 3 por robot.

2.7. Desmontando un robot

Victor de Inven nos ha recomendado desmontar el robot Clónico que había de ejemplo, y hacerle consistente con el que se está construyendo. Supone trabajo extra, pero de esta forma ambos robots serán prácticamente iguales, lo que facilitará el diseño de las bibliotecas.

2.7.1. Entendiendo la maraña de cables

Álvar ha desconectado todas las conexiones que había en el robot, y ha encontrado malas praxis como pegotes de estaño que conectaban 3 cables diferentes, cables conectados a la Arduino pero que no iban a ninguna parte...

Una vez reestructuradas las conexiones, los robots Clónicos se parecen mucho más entre ellos.

2.7.2. Motores

Soldar

A un motor le faltaba un cable, el negro concretamente. Ésto obligaba a revisar las conexiones de los demás motores, por si podía haber más problemas.
Los motores, como era de esperar, no cumplían el patrón que se explicó en 2.3. Para que los motores fueran consistentes entre ambos robots, se han desestañando, cambiado los cables de posición, y se han vuelto a estañar.

Tornillos de los motores

Después de inspeccionar los motores más detenidamente, Álvar se fijó que los tornillos estaban mal instalados; además de ser imposibles de retirar cuando estaba instalado un motor en línea recta a los cabezales, la rosca de los tornillos estaban erosionando las ruedas del robot (que son de plástico).
Después de soldar los motores, se han instalado los tornillos debidamente.

El cabezal del tornillo tiene que estar mirando a las ruedas por dos motivos principales: en primer lugar porque son más accesibles de esta manera en el caso de que se necesiten desmontar, y en segundo lugar, el cabezal es la parte que más se ajusta al motor, la que menos sobresale, por lo que es muy complicado que los tornillos dañen las ruedas cuando están así instalados.

2.8. Problemas con la nueva shield

Álvar, haciendo pruebas sobre la shield con driver de motores, ha identificado un suceso extraño.

Cuando se alimenta por USB la Arduino con la shield montada, los motores se mueven muy lentamente. Al proporcionarles corriente con el portapilas de 9V (sin desconectar el cable USB), los motores van a una velocidad más razonable y todo funciona con normalidad. Pero si se desconecta el USB, o se alimenta la Arduino únicamente con el portapilas, el microcontrolador se reinicia, y, de forma esporádica, los motores giran hacia delante hasta que se reinicia de nuevo (por el programa de prueba usado).

Este suceso es un problema, porque uno de los objetivos es que el robot se mueva de forma independiente, sin estar conectado al ordenador.

2.8.1. Cómo alimentar la Arduino y los motores

La Motor Shield, cuando cambia la dirección de giro de los motores, internamente, cambia la dirección de la corriente entre los zócalos de los motores. Por lo tanto, si la Arduino recibe electricidad por el pin V_{in}, la Arduino percibe ese cambio. Ésto la Arduino lo considera un cortocircuito o una sobrecarga, y se reinicia.

Álvar, después de investigar en Internet las diferentes formas de alimentar placas Arduino, descubrió que el pin $5V$ puede servir de entrada de voltaje, aunque no es recomendable porque es habitual su uso como salida de voltaje, y puede dar lugar a confusión y otros problemas. Álvar sospecha que si se supera un poco el voltaje (se reciben $7V$ por la entrada de $5V$) no surjan mayores problemas en la alimentación de la Arduino, al contrario; Álvar cree que así hay un margen de error para soportar variaciones en el voltaje.

Por lo tanto, se ha decidido alimentar a la Arduino por el pin de $5V$ en vez de por el pin V_{in}, y los motores recibirán alimentación directamente del portapilas.

Pruebas

Se han realizado pruebas con la configuración siguiente:

- V_{in} no está conectado (el jumper de la Motor Shield ha sido retirado).
- V_{MS} está conectado al positivo del portapilas.
- $5V$ está conectado al positivo del portapilas.
- *Gnd* de la Arduino está conectado a tierra del portapilas.

El robot avanza a gran velocidad, y sólo se para cuando se retira la alimentación.
La prueba ha sido un éxito.

2.9. Integración de robots Clónicos

Sin más problemas que documentar, Álvar logró que los robots Clónicos fueran robots físicamente muy parecidos y funcionalmente idénticos. La documentación de las conexiones se encuentra en el anexo B.

2.10. Módulos Bluetooth

Las shield de ambos robots (tanto los Clónicos como los Elegoo) están diseñadas para insertar un módulo Bluetooth que funcione como entrada/salida de datos. Este módulo se conecta a unos «sockets» que son compatibles con los pines del módulo, haciendo la instalación muy sencilla.

2.10.1. Materiales

Se dispone de 2 módulos, un HC-06 y un HC-08.
El módulo HC-06 utiliza Bluetooth 2.0 y necesita $5V$ para activarse. En cambio el módulo HC-08 implementa Bluetooth 4.0 (Bluetooth Low Energy) y requiere unos $4V$ para funcionar a pleno rendimiento.

Hubo un mal uso del HC-08

Álvar, antes de conocer las características de los módulos, los trató como iguales y les suministró la misma corriente a ambos. Parece ser que parte del módulo HC-08 se ha quemado, y ahora sólo es capaz de funcionar a $5V$ y sólo puede enviar información a otro dispositivo, no puede recibir.
Puede seguir siendo útil, pero debe tenerse en cuenta este inconveniente.

2.10.2. Funcionamiento

El módulo Bluetooth se integra de tal manera que es invisible a la aplicación.
Se aprovecha que la Arduino posee una comunicación serial por los pines 0 y 1 para conectar los pines «recepción» y «envío» del módulo, respectivamente. De esta manera se puede simular que la Arduino está enviando o recibiendo información por cable, que es un proceso ya integrado en las bibliotecas de Arduino, cuando en realidad el módulo Bluetooth está intermediando entre los dispositivos.
No se añade complejidad al código gracias a que el módulo actúa de forma invisible.

2.10.3. Problemas

Los módulos Bluetooth son capaces de ralentizar considerablemente a los robots Clónicos. No causan tanto impacto con los robots Elegoo porque distribuyen la potencia de las pilas de manera distinta entre los módulos y los motores, una distribución más adecuada a este proyecto opina Álvar.

Además, tal y como se tiene que conectar la alimentación de la Arduino y de los motores en la Motor Shield, los robots Clónicos no puede leer mensajes de Bluetooth que manden otros dispositivos. Por alguna razón la Arduino debe ser alimentada por el pin V_{in} para que la placa pueda leer la información que llega del módulo Bluetooth.
Si se hicieran esas conexiones, la única opción sería alimentar los motores con la salida de $5V$ de la Arduino, con lo que los motores sólo podrían moverse si se les aplica la máxima potencia (255 de forma constante).
Ésto no debe hacerse, porque la intención del proyecto es poder trabajar con robots diferentes utilizando el mismo código. Si se necesita modificar la velocidad cada vez que se cambia de robot, no estaríamos utilizando una interfaz transparente, y la premisa con la que se inició este proyecto no se cumpliría por intentar integrar un módulo opcional.

Conclusión

Se recomienda no utilizar los módulos Bluetooth en los robots con Motor Shield, a menos que se requiera que la Arduino mande información a otro dispositivo, pero que la Arduino no necesite respuesta. Por ejemplo, se puede utilizar el módulo Bluetooth para estudiar el comportamiento de la placa mientras ejecuta código.

2.11. Receptor de infrarrojos

No se ha trabajado con el receptor de infrarrojos, dado que se ha considerado que los módulos Bluetooth son más versátiles, intuitivos, y permiten que la Arduino se comunique con el exterior, que es una función que el infrarrojos no tiene y que se considera de mayor importancia que las funciones que aporta el receptor de infrarrojos.

3 Programar en Arduino

3.1. Lenguaje de programación

Arduino utiliza C++ como lenguaje de programación, un lenguaje de bajo nivel, como C, pero que posee abstracciones como la orientación a objetos, lo que facilita enormemente la gestión de módulos y bibliotecas.
C++ nos permite tanto modificar los registros de Arduino (si hiciera falta), como gestionar una biblioteca para el control de un robot que recorre grafos, como es el caso. Permite trabajar a todos los niveles de abstracción que nos interesan, lo que nos va a resultar muy útil.

3.2. El «main» de Arduino

Arduino tiene una función *main*(), pero se utiliza para inicializar métodos y variables que la biblioteca usará constantemente.
En vez de un método *main*(), Arduino nos proporciona dos métodos para poder ejecutar nuestros programas: el método *setup*() y el método *loop*().

- El método *setup*() sólo se ejecuta una vez, antes que el método *loop*().
- El método *loop*() se ejecuta de nuevo cuando termina su última línea de código, nunca para.

El método *setup*() se suele utilizar para dar valores a las variables que se van a utilizar en el *loop*() (si los pines deben recibir o enviar información, si queremos que un LED esté siempre encendido...), y el *loop*() suele controlar el «cometido» de ese código Arduino (acelera los motores mientras veas una línea, envía señales en código morse...).

3.3. Entorno de programación de Arduino

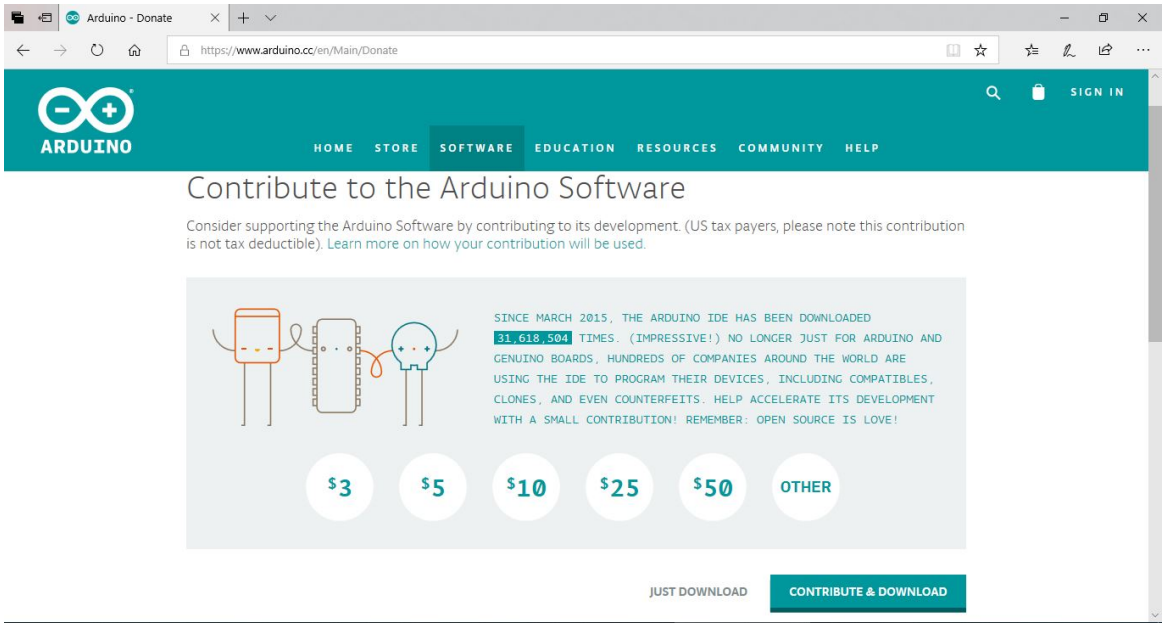
Se utilizará el Arduino IDE para programar la Arduino. Se pueden utilizar paquetes para editores como Sublime o Atom que realizan todas las funciones de este entorno, pero este software está especialmente diseñado para trabajar con Arduino, y nos permite instalar, junto con el programa, todos los drivers necesarios para que el ordenador reconozca las placas Arduino cuando se conecten.

3.4. Descargar

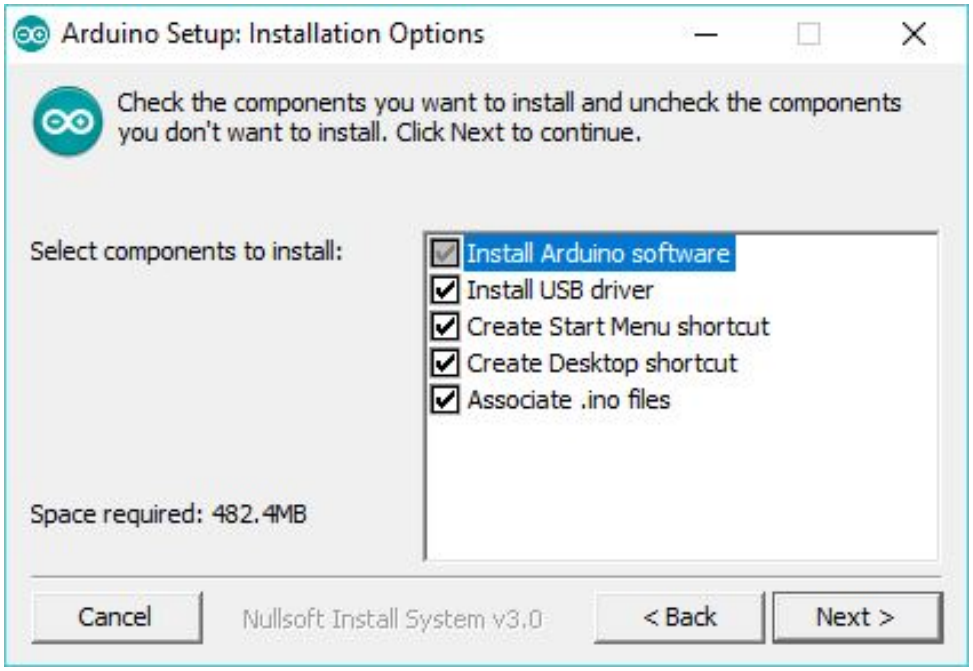
Se debe acceder a la página web Arduino Software para descargar el instalador.

Dentro de las opciones del sistema operativo, se debe seleccionar la opción «Instalador», dado que la versión portable no nos proporciona los drivers necesarios para conectar el ordenador a los Arduino (en sistemas Linux el software está empaquetado junto con los drivers).

Una vez seleccionado el sistema operativo, sólo necesitamos darle al botón «JUST DOWNLOAD» para empezar la descarga del instalador.

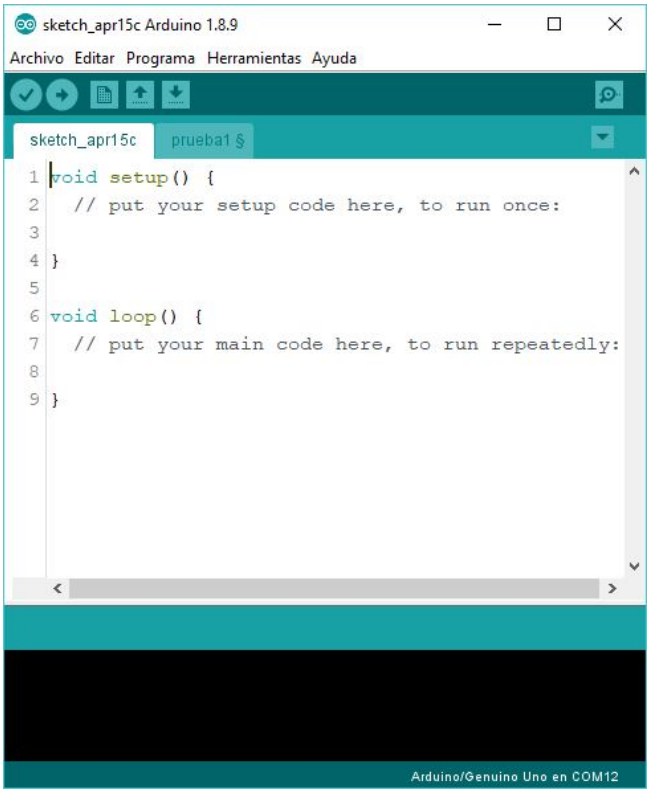


Cuando se ejecute el instalador se recuerda que la opción «Install USB driver» es crítica para el funcionamiento de las prácticas. Las demás opciones son opcionales, pero se recomienda dejarlas marcadas. Dándole al botón «Next» el instalador continuará con la instalación, y cuando termine se podrá abrir el Arduino IDE.



3.5. La interfaz gráfica

Este programa posee una interfaz gráfica muy característica:



3.5.1. Opciones

En la parte de arriba se puede observar el panel de opciones del programa, y, justo debajo, símbolos que representan las opciones más usadas: «Compilar», «Enviar a la Arduino», «Nuevo», «Abrir» y «Guardar», en ese orden.

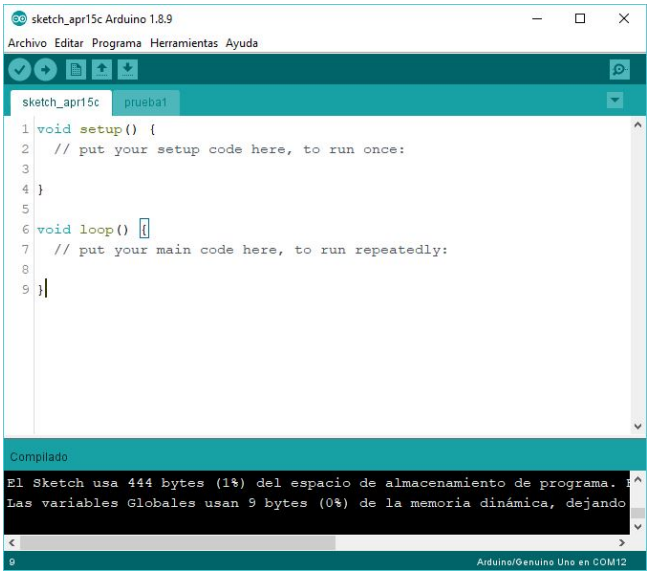
El símbolo de la derecha es el Monitor Serie, que nos permitirá enviar y recibir caracteres de la Arduino si el microcontrolador se mantiene conectado por cable al ordenador.

3.5.2. Código

Debajo de las opciones se encuentran todos los archivos que formarían parte del proyecto con el que se está trabajando en este momento.
Podemos ver los nombres de dichos archivos, el documento que se está editando actualmente, y se pueden identificar los cambios que no han sido guardados si al nombre de algún archivo le acompaña el símbolo §.
Cuando se pulse el botón «Compilar» o «Enviar», todos los archivos del proyecto se guardarán automáticamente y formarán parte de la compilación.

3.5.3. Consola

Cuando se compile el proyecto, la parte inferior de la aplicación informará del proceso.
Si no ha habido ningún problema, la consola devolverá los recursos que requiere la Arduino para ejecutar ese código.



Si ha habido errores de compilación, la consola indicará las líneas en las que se ha encontrado el problema.

Se recomienda leer la consola antes que dejarse guiar por la posición del cursor en el texto. Cuando el código es un poco complejo (varias llamadas a métodos, uso de bibliotecas...), el cursor suele posicionarse en lugares que no son útiles para corregir el código. La consola proporciona datos más claros sobre el error, y la línea y columna donde se ha desencadenado.

3.5.4. Línea y modelo de Arduino

Debajo de la consola se nos informa, a la izquierda, del número de línea en el que está el cursor en el editor de texto, y a la derecha, del modelo de Arduino que se está utilizando y por qué puerto se conecta.
Para estas prácticas siempre se usará Arduino/Genuino Uno, y cada robot interactúa con el ordenador por un puerto distinto.

3.6. Problemas frecuentes

3.6.1. El software no conecta con la placa

Aunque todos los robots de esta práctica utilizan el modelo Arduino Uno, cada uno se conecta al ordenador por un puerto diferente.
En *Herramientas > Puerto* debería poder seleccionarse el puerto de la placa Arduino que esté conectada en este momento.
Si no es así, debe comprobarse la conexión USB.
En última instancia, se debería reiniciar el ordenador, y si sigue sin funcionar, reinstalar los drivers de Arduino.

3.6.2. El código no se sube a la placa

Cuando se quiere enviar código a la placa antes de terminar un envío realizado previamente, el programa puede que no sea capaz de conectar con la placa correctamente.
Reiniciar la aplicación debería solucionar el problema.

3.6.3. El *setup*()/*loop*() ya está definido

Sólo puede haber un método *setup*() y un método *loop*() en todo el proyecto. Si hay más de uno, el código no compila.

Se deben eliminar o comentar los *setup*() y *loop*() que no se vayan a utilizar en esa ejecución.
Si se necesita que el código de un método a comentar se ejecute dentro de otro, se pueden renombrar los métodos, o copiar el contenido de las funciones en el *setup*() y *loop*() que se vaya a utilizar finalmente.

3.6.4. No recibo información de la Arduino por el monitor

Asegúrate de que la Arduino está conectada al ordenador, y de que los baudios son los mismos en ambos lados de la comunicación. De otra manera el ordenador y la Arduino estarán desincronizados.
Si todo lo anterior está bien, comprueba el código que le has mandado a la Arduino.

4 La biblioteca

4.1. Descripción de la biblioteca

Las interfaces se encuentran en el apéndice C.

4.1.1. Clase Robot

Clase que, al saber los pines de la Arduino a los que están conectados los motores, permite controlar dichos motores de manera que el robot se mueva en la dirección establecida.

Constructor de un objeto Robot

Álvar ha considerado que lo único que debe ser obligatorio al inicializar el robot es declarar los pines de los motores. Todo lo demás se implementará como módulos, extensiones independientes a la implementación del robot que aumentan sus capacidades.

4.1.2. Módulos de Robot

Cada uno de los módulos será su propia clase, y cada marca de robot tendrá su propia instancia de la clase Robot. De esta manera, además de hacer el código más legible, se puede cambiar de configuración de los robots únicamente cambiando de objeto, sin tener que cambiar parámetros de la biblioteca.

Siguelíneas

Módulo que guarda los pines que se conectarán a los sensores siguelíneas, y las correspondientes operaciones de lectura de dichos sensores.

Morse

Clase que utiliza código morse para comunicarse con el usuario.
Puede enviar mensajes por medio de luz (LED) o de sonido (zumbador).

Bluetooth

Este módulo utiliza tecnología Bluetooth para que el robot se pueda comunicar con otro dispositivo mientras está activo (con un dispositivo Android, por ejemplo).

Infrarrojos

Con un mando que manda señales infrarrojas, este módulo puede recibir información de forma inalámbrica y que la Arduino la reciba mientras está ejecutando código.

IMPORTANTE La biblioteca *RobotIRremote* causa muchos conflictos.
RobotIRremoteTools.h y *IRremoteTools.cpp* en concreto son problemáticos. Como no se van a usar en esta práctica, Álvar ha retirado estos archivos. La biblioteca vuelve a compilar, y todo funciona de

manera correcta.
También es posible que *RobotIRremote* genere conflictos con el método *tone*(), usado en la biblioteca Morse. La solución que ha encontrado Álvar es cambiar el timer de la biblioteca *RobotIRremoteInt.h*, en la línea 67. Después de esta corrección, el código es capaz de compilar.

Se ha decidido no utilizar esta biblioteca. Causa demasiados problemas, y se prefiere que la Arduino mande mensajes antes de que pueda recibirlos (lo que se conseguiría con el módulo Bluetooth). El input en tiempo de ejecución se puede sustituir por valores constantes en el código.

4.2. Abstraer los robots

Como se va a trabajar con robots de distintas empresas, Álvar ha diseñado una clase para abstraer la implementación concreta de cada robot y poder trabajar siempre con una misma interfaz.
Los robots Clónicos se abstraen fácilmente con la documentación en B, y los robots Elegoo son Smart Robot Car v2.0, y en su página web se puede encontrar documentación sobre su funcionamiento.

4.2.1. Clases en C++

Álvar ha adquirido el conocimiento de programar clases en C++ gracias al libro Liberty [1999], dado que no sabía programar en C++.

4.2.2. Control de motores

La mayor diferencia entre ambos modelos de robot es cómo gestionan los motores.

En el caso de los robots Smart Robot Car v2.0, cada lado del robot tiene 2 pines que indica la dirección en la que giran los motores. Por lo que el giro de los motores se controla con 4 pines.
Esto no es así con los otros robots, que sólo necesitan 1 pin para controlar la dirección de giro de cada lado.

Contando con ello, Álvar ha diseñado un constructor que pide 2 pines de control de motores para cada lado, pero en el caso de los robots Clónicos, se dará un pin inaccesible para la Arduino cuando se pida el segundo pin de control. Ensucia el código y puede empeorar un poco el rendimiento, pero de esta manera las interfaces de ambos robots son exactamente iguales.

Código de prueba

Utilizando diferentes constructores, Álvar ha cargado el mismo programa a robots de diferentes marcas y ha probado a hacerles funcionar a la vez.
Excepto por una pequeña diferencia en distancia recorrida (seguramente porque un robot tenga las pilas más gastadas que el otro), los dos robots funcionan igual, cumpliendo los mismos tiempos y realizando las mismas acciones.

La interfaz funciona correctamente.

4.2.3. Sensores siguelíneas

Dificultades

Los robots Clónicos venían con sensores de diferentes marcas. En principio no sería un inconveniente, pero resulta que cada marca envía señales diferentes.
Uno de los sensores responde a la pregunta *¿hay blanco en el suelo?* y el otro responde a la pregunta *¿hay negro en el suelo?*. Son preguntas inversas, y es un problema al intentar hacer una interfaz,

porque desde la biblioteca no podemos saber qué sensor está en qué puerto, sólo se puede conocer la marca del sensor analizando la implementación.

Álvar ha pedido a Inven que todos los sensores sean de la misma marca, porque de otra manera sería muy complicado hacer una interfaz que sea consistente con todos los robots de los que se dispone.

Álvar ha desmontado los sensores siguelíneas que causaban problemas y los ha intercambiado por sensores más fiables.
Los sensores se integran perfectamente con la biblioteca.

4.2.4. Bluetooth

Para poder «debuggear» la biblioteca (o las prácticas más adelante), sería muy útil conocer la información que posee el robot en un momento determinado.
Las placas Arduino trabajan con memoria volátil. Una vez se dejan de alimentar, los registros no pueden leerse. Por eso se necesita que mande la información que nos interesa mientras está encendido, mientras se está generando esa información.
Como el objetivo es que el robot se mueva de forma autónoma, no se contempla tener un cable conectado al robot de forma continua. Por esa razón se va a investigar la posibilidad de utilizar tecnología Bluetooth para que el robot nos informe del estado de la placa cada cierto tiempo, y si hay algún problema, leer el «log» generado para intentar solventarlo.

Elegoo BLE Tool

Elegoo tiene una aplicación para controlar sus robots de manera remota con tecnología Bluetooth. Esta aplicación se llama Elegoo BLE Tool, e instalada en un dispositivo Android permite mandar órdenes sencillas a un módulo Bluetooth que se puede añadir al robot.
Por el momento Álvar no ha encontrado ninguna opción para que el robot pueda mandar mensajes al dispositivo, que es lo que se necesitaría para poder diagnosticar errores en la programación del robot. Por ello se van a investigar alternativas.

bluetoothctl

Álvar ha estado investigando Bluetooth commands para poder leer inputs. Álvar no ha encontrado la manera de utilizar comandos para emparejar un dispositivo con tecnología Bluetooth Low Energy con un ordenador.

Elegoo BLE Tool 1.0

La versión 1.0 de la aplicación de Android, que se encuentra en la documentación del Smart Robot Car v2.0, se adapta a las necesidades del proyecto. Lee input, es capaz de mandar datos con una interfaz parecida a una consola, y se empareja al módulo de Bluetooth de forma sencilla.
Se han buscado alternativas que sean más sencillas de instalar, que reciban y manden información al módulo Bluetooth, y que ofrezcan la apariencia de consola de esta aplicación.

Serial Bluetooth Terminal

La aplicación preferida de Álvar. Es una aplicación ligera (5MB), tiene una interfaz de consola muy limpia y práctica que permite enviar y recibir mensajes de forma sencilla, y puede emparejarse tanto con Bluetooth clásico como con Bluetooth BLE.
Para todas las pruebas del módulo Bluetooth, Álvar usará esta aplicación.

4.3. Integración de la biblioteca Cnosos

Álvar ha recibido una biblioteca llamada «Cnosos» que funcionaba bien con los robots Elegoo, y se quiere integrar en la biblioteca que se está diseñando.

4.3.1. Descripción de la biblioteca

Cnosos es una librería que permite a la Arduino navegar un grafo dibujado en el suelo leído por sensores siguelíneas.

Entre las rutinas que ofrece se incluyen la gestión de los motores para que el robot se mantenga siempre encima del grafo, la lectura de identificadores de nodos, el reconocimiento de vértices del grafo, o funciones básicas de navegación como «gira en el vértice 2».

4.3.2. Estudio del funcionamiento de la biblioteca

Álvar encontró difícil entender el problema que solucionaba cada función de la biblioteca.
La falta de comentarios fue lo que confundió más a Álvar, debido a que se mencionan conceptos y cálculos de teoría de grafos que Álvar no dominaba del todo, por lo que tardó un tiempo en inferir la funcionalidad de cada método, además del cómo y cuándo debía ser utilizado cada uno para gestionar correctamente la navegación del grafo.

4.3.3. Integración de Cnosos en el proyecto

Se ha podido hacer una traducción casi directa del código original a las bibliotecas del proyecto.
Los métodos de control de motores eran muy parecidos en ambas bibliotecas, al igual que los métodos para leer los sensores siguelíneas.

El cambio más importante que se ha implementado es que ahora se pide un objeto Robot para poder empezar a usar los métodos (antes suponía que era un robot Elegoo), y esa funcionalidad se ha conseguido encapsulando todas las subrutinas de Cnosos dentro de una clase (dado que antes los métodos eran accesibles en todo momento, estuviese el robot preparado o no).

Evitar la función *delay*()

Una de las partes que más preocupaba a Álvar fue la gestión de los *delay*() repartidos por la biblioteca Cnosos.
Los tiempos que había en la biblioteca Cnosos eran específicos para los robots Elegoo a unas velocidades muy concretas. Es muy difícil conseguir valores tan exactos al gestionar distintas marcas de robots, que funcionan a diferentes voltajes y con diferentes conexiones.

Álvar ha intentado sustituir los *delay*() por otros métodos de espera, como instrucciones *while*() que esperen hasta que se llegue a un estado concreto de los siguelíneas.
A fecha de la entrega de esta memoria, Álvar no ha conseguido resultados estables, pero sí ha observado que es una alternativa muy prometedora. Recomienda seguir este estudio para que la biblioteca Cnosos pueda controlar robots de diferentes marcas como es capaz de controlar ahora los robots Elegoo.

5 Decisiones de diseño de la biblioteca

5.1. Arrancar o no arrancar

Los robots Elegoo no necesitan arrancar cada vez que cambian de dirección. Así que se ha añadido un atributo llamado $NECESITA\_ARRANCAR$.

- Si $NECESITA\_ARRANCAR = true$, el robot arrancará cada vez que se cambia de dirección.
- Si $NECESITA\_ARRANCAR = false$, el robot NO arrancará al cambiar de dirección.

5.1.1. Siempre arranca, pero durante tiempos distintos

Álvar ha descubierto que *delay*(0) en verdad sólo es una llamada a un método, que consume un tiempo despreciable para nuestro proyecto. Por lo tanto, para ahorrar el if que rodeaba al método *arranca*(), siempre se arranca.
Si el robot necesita estar un tiempo arrancando, ese tiempo debe haberse indicado en el constructor. Si no necesita arrancar, ese tiempo será 0.

El método *setTiempoArranque*() permite modificar el tiempo de arranque una vez construido el objeto.

5.2. Dirección de los motores como máquina de estados

Álvar ha estado estudiando el funcionamiento de la biblioteca, y se ha dado cuenta que los robots se comportan como si tuvieran distinto código al «cambiar de estado» (cambiar el sentido de los motores y arrancar) que al «mantenerse en un estado» (girar a cierta velocidad).

Como *arranca*() llama al máximo de velocidad, no queremos que se esté ejecutando cada vez que se llama al método. Pero es muy práctico que el propio método sepa cuando tiene que llamar a *arranca*(), y que el usuario no se preocupe de si hay que arrancar el robot o no.
Por lo tanto Álvar plantea la posibilidad de modelar este código como una pequeña máquina de estados, que evitaría que el robot llamase constantemente al método *arranca*().

5.2.1. Estados identificados como enumerados

Álvar está utilizando una clase enumerada para determinar el estado actual del robot. Si el robot se mantiene en el mismo estado (la variable que guarda el enumerado actual tiene el mismo valor que el enumerado que se iba a escribir), no cambia de dirección los motores ni llama a la función *arranca*().

5.3. Parámetros opcionales o distintos constructores

C++ no permite que los objetos se inicialicen a $NULL$ como se puede en Java. Sólo los punteros ($Robot*$ por ejemplo) se pueden inicializar a $NULL$ pero se ha decidido que trabajar con punteros es complicar el código, por lo que no se trabajará con ellos en esta ocasión.

Por lo tanto, se ha de declarar un constructor que permita no introducir argumentos para inicializarse, de manera que la sintaxis se parezca a la de Java (que es a lo que los estudiantes están acostumbrados).
Álvar ha sugerido dos opciones: o que los constructores tengan todos los argumentos opcionales, o construir varios constructores, uno con argumentos, y otro sin ellos.

5.3.1. Parámetros opcionales

+ Sólo aparece un constructor en la interfaz.
+ Se pueden usar los argumentos que se crean convenientes.
− No están claros los argumentos fundamentales para definir el módulo.
− No genera errores al no introducir parámetros clave para el módulo, como puede ser los pines en los que debe fijarse el robot para leer los sensores siguelíneas.

5.3.2. Distintos constructores

+ Cada constructor tiene un uso concreto.
+ Se identifican mejor las opciones necesarias para definir el módulo.
+ Errores de inicialización de los módulos se manifiestan antes incluso de conectar la Arduino.
− Hay más constructores, que puede que el usuario final nunca use. Ensucia el código.

5.3.3. Conclusión

Valorando las opciones, se van a utilizar distintos constructores para inicializar los módulos.

Esta biblioteca intenta que programar un robot en Arduino sea accesible y que cause los menores problemas posibles. Si esos problemas se manifiestan mientras el robot está moviéndose (el caso de un código compilado correctamente), son más difíciles de depurar que si el compilador avisa de un error, y estas situaciones pueden causar gran frustración.

Si la biblioteca no permite al usuario subir el código a la Arduino, se ve obligado a corregir el código, y tendrá una experiencia más satisfactoria que si el robot no lee las entradas del siguelíneas porque no se inicializó el objeto con todos los argumentos, por ejemplo.

5.4. Ejemplos de código

La Arduino IDE permite escribir código que sirva de ejemplo para el uso de las bibliotecas.
Se ha escrito un pequeño programa para demostrar el uso de cada módulo.

5.5. Gestión de las baterías

5.5.1. Modificando la velocidad de los motores

Álvar, después de investigar ejemplos de código Arduino, ha descubierto que ciertos pines digitales pueden utilizar la función $analogWrite()$ PWM, y que los pines que están controlando los motores pueden usarla.

Con esta información es posible mantener la velocidad de los motores de manera relativamente controlada y constante de forma independiente a la carga de las pilas.
La siguiente tabla muestra ejemplos de la afirmación anterior:

Voltaje pilas	40 % capacidad	33 % capacidad	25 % capacidad
$9V$	$3'6V$	$3V$	$2'25V$
$7'4V$	$2'96V$	$2'47V$	$1'85V$
$5V$	$2V$	$1'67V$	$1'25V$

De una diferencia de $4V$ entre los valores máximo y mínimo que pueden tener las pilas, podemos obtener diferencia de $1V$, lo que consigue velocidades más controladas y menos dependientes del voltaje, además del ahorro energético que se consigue al no utilizar las pilas a su máxima capacidad.

El último aspecto por tratar es el voltaje mínimo al que los motores son capaces de mover el robot.

Voltaje mínimo que aceptan los motores

Después de experimentar con voltajes que pueden sacar los pines, Álvar ha llegado a la conclusión de que cada motor necesita al menos $0'4V$ ($1'6V$ si sumamos los 4 motores) para poder mantener la inercia, y $1'2V$ ($4'8V$ contando los 4 motores) para arrancar.
Esta información nos obliga a considerar que en el caso de que la pila sólo tenga $5V$, vamos a necesitar prácticamente un 100 % de ese voltaje para arrancar, y con el 33 % del voltaje de la pila debería servir para mantener al robot en funcionamiento. Como es posible que se le añadan más módulos al robot, usaremos un 40 % de la capacidad de las pilas por precaución.

Voltaje necesario para realizar giros

Álvar, haciendo pruebas con el voltaje mínimo para que los motores puedan hacer girar al robot, se ha percatado que los motores necesitan mucho más voltaje que en cuando el robot va recto.

Álvar ha deducido que la potencia efectiva que reciben las ruedas es un porcentaje de la potencia suministrada, y cree que ese porcentaje guarda relación con el coseno de la circunferencia que traza el robot al girar.
No se ha podido caracterizar de manera matemática, pero experimentalmente se ha concluido que los giros se trazan bien al 70 % de la capacidad de las pilas.

La cinta y los motores

La cinta que se usa para marcar el grafo tiene menos resistencia que el papel que estamos utilizando. Se ha aumentado la potencia de giro al 80 % de la capacidad de las pilas para que las ruedas puedan rotar sin problemas sobre ambas superficies.

Cambio de sentido de los motores

El robot, cuando necesita girar, ha de parar los motores de un lado, o cambiar su sentido de giro. Eso quiere decir que toda la inercia acumulada se pierde, y debemos «arrancar» el robot de nuevo para que pueda darse suficiente energía a los motores para que empiecen a girar en la dirección deseada.

Álvar ha considerado utilizar la función *arranca*() cada vez que se quiere cambiar de dirección.
De esta manera el usuario no se daría cuenta de este problema, y una vez finalizado el giro el robot se moverá con la potencia indicada.

5.5.2. ¿Controlar la velocidad dependiendo del voltaje de las pilas?

Investigando, Álvar ha encontrado la forma de leer input analógico en la placa Arduino ana. La limitación que ha encontrado es que los pines indicados para la lectura analógica tienen como rango voltajes entre $0V$ y $5V$, y superar ese límite puede dañar los lectores de la placa.
Por el momento no se va a observar el voltaje de las pilas para calcular la velocidad del robot.

6 Optimizaciones

6.1. Trabajando con bytes

Después de ver la memoria que todos los módulos estaban consumiendo, Álvar, después de estudiar la biblioteca, descubrió que podía sustituir gran cantidad de variables enteras (de 4 bytes de tamaño) por variables de tamaño byte sin afectar al funcionamiento del código.

6.1.1. Mejora conseguida

No sólo el programa ocupaba una pequeña fracción de memoria comparado con el anterior, sino que los programas se ejecutaban más rápido (las placas Arduino tienen procesadores de 8 bits, y están mejor preparadas para trabajar con datos de 1 byte de tamaño).
Esta optimización permitió que el arranque del coche, de necesitar $100ms$ pasara a necesitar $80ms$. Como se ha conseguido mayor velocidad en el cómputo, el coche reacciona antes, y necesita menos «tiempo de arranque».

7 Conclusiones

7.1. Objetivos conseguidos

7.1.1. Robots Clónicos, documentados y funcionales

Se han podido montar los robots Clónicos en el transcurso de la beca, además de documentarse sus conexiones y los controles de cada uno de sus módulos.

Los robots Clónicos sólo pueden usar Bluetooth para enviar

Por restricciones del proyecto, no es posible hacer girar los motores y recibir información por Bluetooth en los robots Clónicos. En cambio, sí son capaces de enviar información por Bluetooth, por lo que el módulo puede utilizarse para enviar información de depuración a otro dispositivo (móvil, ordenador, etc.).

7.1.2. Interfaz común para varios modelos de robots implementada

Incluso con arquitecturas diferentes, robots de diferentes marcas son capaces de recibir código casi igual (sólo cambian las inicializaciones), y de comportarse de la misma manera.
Es un gran avance en el planteamiento de las prácticas de Algorítmica y Complejidad, dado que se dispone de más infraestructura, y simplifica el control de los robots para gente que no conoce C++ pero sí otros lenguajes orientados a objetos.

7.1.3. Interfaz de la biblioteca Cnosos

Todos los métodos de la biblioteca Cnosos han sido documentados, y se ha creado una interfaz con los métodos que Álvar piensa que van a ser útiles en algún momento de la práctica, «escondiendo» las funciones que se piensa que el usuario no necesita conocer.

Implementación parcial de Cnosos

Se ha conseguido implementar todos los métodos de la interfaz Cnosos, pero no se han obtenido los resultados esperados.

7.2. Objetivos pendientes

7.2.1. Implementación completa de la biblioteca Cnosos

Para una próxima iteración de este proyecto, basándose en el código escrito en iteraciones anteriores, debería poderse completar la implementación de la biblioteca Cnosos con los resultados deseados para completar las prácticas de la manera más estable posible.

7.2.2. Adaptar los guiones de las prácticas a la nueva biblioteca

Ahora mismo los guiones de las prácticas están escritos con los métodos y definiciones de la biblioteca Cnosos antigua. Deberían reescribirse para que los guiones y la biblioteca sean consistentes.

7.3. Sugerencias

7.3.1. Adquirir un soldador y estaño

Las conexiones de los robots Clónicos están al aire, y aunque Álvar ha hecho todo lo posible para que las reparaciones sean sencillas e intuitivas, es probable que en el futuro sea necesaria una soldadura o cambiar los cables.
En esta situación será necesario un soldador y estaño para arreglar los robots. Se recomienda encarecidamente tener estos elementos a mano por cualquier eventualidad que pueda pasar.

7.3.2. Mantener las bibliotecas bien documentadas

Uno de los pocos elementos que ha parado el desarrollo del proyecto ha sido la falta de documentación de la biblioteca Cnosos.

Por ello se recomienda, ahora que las bibliotecas están documentadas, mantener, actualizar y ampliar dicha documentación cada vez que sea necesario.
Si el proyecto se desarrolla en el tiempo, es una inversión muy positiva informar en unas líneas del código que se ha escrito.

7.3.3. Cambiar el código a otro proyecto GitHub

El proyecto GitHub en el que reside el código al terminar esta memoria incluye mucho material que describe la evolución de la beca.
Cuando sólo se quieren usar las bibliotecas para una práctica, todo ese material lo único que hace es ralentizar la descarga, y puede dar problemas de tiempo.
Se sugiere que cuando se vaya a proporcionar esta biblioteca a los alumnos:

- Se entregue un archivo comprimido con las bibliotecas necesarias para completar la práctica.
- Se proporcione un enlace a otro proyecto GitHub en el que únicamente residan las bibliotecas.

7.3.4. Cambiar las marcas para evitar fallos de lectura

Álvar avisa que usar «huecos» en la cinta como una marca puede causar problemas indeseados. Dependiendo de la sensibilidad de los sensores, es posibles que se lea un «hueco» cuando el robot está simplemente corrigiendo su dirección, debido a que pierde la línea durante un momento.

Se propone utilizar sólo «cintas atravesadas» como marcas, una más fina y otra más ancha para representar diferentes bits (como si fuera código morse). Álvar piensa que se reduce en gran medida la posibilidad de un fallo de lectura con este sistema (dado que leer 3 sensores a negro suele ser menos fortuito que leer 3 sensores a blanco).

7.3.5. Considerar dibujar los nodos sobre papel

En este momento los grafos se están dibujando sobre el suelo. Álvar piensa que es una solución un poco limitada, debido a que si el espacio necesita usarse para otra finalidad, se ha de retirar todo el grafo, lo que genera trabajo cada vez que se quiera realizar una práctica.
Álvar sugiere utilizar «mapas», pintando los nodos en papel y juntando diferentes nodos con «vías», de papel también. Es una solución que aprovecha mucho más el trabajo realizado al pintar los nodos, y que aporta flexibilidad en el proceso de diseño e implementación de los grafos.
Un inconveniente que se ha encontrado es que los robots patinan más al circular sobre papel, lo que obliga a reconsiderar el voltaje que se aplica a los motores.
Debe estudiarse más a fondo si merece la pena esta sugerencia.

A Vocabulario

Este anexo es un pequeño diccionario que recoge las palabras más técnicas de la memoria.

Clónico: Robots vendidos por la empresa Inven. Un conjunto de componentes que se tenían que montar y soldar a mano. Se le ha instalado una Motor Shield shi para evitar el mayor número de cables posible, a expensas de potencia en los motores.

Driver de motores: Circuito integrado que es capaz de controlar la dirección de giro y la corriente que pasa por los motores a los que está conectado. Los robots Clónicos lo tienen integrado en la Motor Shield, mientras que los robots Elegoo tienen su driver separado de la Arduino.

Elegoo: Robots vendidos por la empresa Elegoo. No necesitan que el usuario realice soldaduras, y su driver de motores está separado de la Arduino, lo que proporciona mayor potencia a los motores.

Shield: Circuito integrado que se conecta a los pines de la Arduino (normalmente a todos los pines) que suele utilizarse para reducir cables o aumentar las capacidades de la Arduino. Por ejemplo, la Motor Shield shi tiene integrados un driver de motores y una sirena, elementos que la Arduino no tiene de fábrica. Existen shields para conectar la Arduino a Ethernet, a pantallas LCD, etc.

B Documentación conexión robots Clónicos

Estas indicaciones presuponen una orientación concreta del robot:
- Las pilas tienen que estar más cerca del usuario que los sensores
- El enfoque es de encima del robot → abajo del robot

A0 → sensor izquierda
A2 → sensor centro
A4 → sensor derecha

5V → alimentación de la Arduino, a la salida de voltaje de las pilas (rojo)
GND → tierra, a tierra de las pilas (negro)

OPT, pin de la izquierda → alimentación de motores, a la salida de voltaje de las pilas (rojo)

10 → Velocidad de MotorA, motores de la izquierda
(LOW = velocidad 0, parado; HIGH = velocidad máxima)
11 → Velocidad de MotorB, motores de la derecha
(LOW = velocidad 0, parado; HIGH = velocidad máxima)

12 → Dirección de MotorA, motores de la izquierda
(LOW = alante, HIGH = atrás)
13 → Dirección de MotorB, motores de la derecha
(LOW = alante, HIGH = atrás)

Motor alante izquierda:
Rojo → MotorB derecha
Negro → MotorB izquierda

Motor atrás izquierda:
Rojo → MotorB izquierda
Negro → MotorB derecha

Motor alante derecha:
Rojo → MotorA derecha
Negro → MotorA izquierda

Motor atrás derecha:
Rojo → MotorA izquierda
Negro → MotorA derecha

C Interfaces de la librería

C.1. Interfaz Bluetooth

```
/**
 * ------------------------------------------------------------------
 * bluetooth.h
 * Clase controlador del modulo Bluetooth.
 * Permite que la Arduino pueda enviar y recibir mensajes.
 *
 * Adaptado por Alvar Tapia, Abril 2019.
 * Legado por Algorítmica y Complejidad, Universidad de Cantabria.
 * ------------------------------------------------------------------
 */

#ifndef bluetooth_h
#define bluetooth_h

#include "Arduino.h"


class Bluetooth{
  #define BAUDIOS_DEFECTO 9600
  private:
    /// Señales por segundo a la que se establece la comunicacion.
    int BAUDIOS;
    /// Caracter de sincronizacion.
    char START_CHAR;

  public:
    //Constructor
    /**
     * Constructor del controlador.
     * @param Baudios
     *    Señales por segundo de la comunicacion.
     *    Por defecto, Bluetooth se comunica a 9600 baudios.
     * @param Caracter de sincronizacion
     *    Si se utiliza el metodo inicializa(), se esperara
     *    este caracter para seguir con el codigo.
     *    Por defecto, sera el caracter 'x'.
     */
    Bluetooth(int = BAUDIOS_DEFECTO, char = 'x');
    //Destructor
    /**
```

```
     * Si el objeto se ha inicializado, al eliminarlo
     * se finaliza la conexion.
     */
    ~Bluetooth();

    //Inicializador
    /**
     * Inicializa las conexiones de la Arduino para que pueda
     * enviar datos a otro dispositivo.
     */
    void inicializa();

    //Funciones
    /**
     * Desactiva la comunicacion de la Arduino
     * con otros dispositivos.
     */
    void finaliza();

    /**
     * La Arduino envia un String a otro dispositivo
     * (sin retorno de carro).
     * @param String con la informacion a enviar.
     */
    void envia(String);

    /**
     * La Arduino envia un String a otro dispositivo
     * e introduce un retorno de carro al final.
     * @param String con la informacion a enviar.
     */
    void enviaLinea(String);

    /**
     * La Arduino esperara al caracter de
     * sincronizacion (definido en el constructor)
     * antes de ejecutar mas instrucciones.
     */
    void sincroniza();
};

#endif
```

C.2. Interfaz Siguelineas

```
/**
 * ------------------------------------------------------------------
 * siguelineas.h
 * Clase que permite utilizar los sensores siguelineas.
 * Se espera que se usen como mucho tres sensores por objeto.
 *
 * Adaptado por Alvar Tapia, Abril 2019.
 * Legado por Algorítmica y Complejidad, Universidad de Cantabria.
 * ------------------------------------------------------------------
 */

#ifndef siguelineas_h
#define siguelineas_h

#include "Arduino.h"

class Siguelineas {
  protected:
    /// Pines a los que estan conectados los siguelineas.
    byte PIN_SIGUELINEAS_IZDA;
    byte PIN_SIGUELINEAS_CENTRO;
    byte PIN_SIGUELINEAS_DCHA;

  public:
    //Constructores
    /**
     * Establece tres pines para los sensores siguelineas.
     * @param Pin siguelineas izquierdo
     *     Pin conectado al siguelineas izquierdo.
     * @param Pin siguelineas central
     *     Pin conectado al siguelineas central.
     * @param Pin siguelineas derecho
     *     Pin conectado al siguelineas derecho.
     */
    Siguelineas(byte, byte, byte);
    /**
     * Establece dos pines para los sensores siguelineas.
     * @param Pin siguelineas izquierdo
     *     Pin conectado al siguelineas izquierdo.
     * @param Pin siguelineas derecho
     *     Pin conectado al siguelineas derecho.
     */
    Siguelineas(byte, byte);
    /**
     * Establece un pin para el sensor siguelineas.
     * @param Pin siguelineas central
     *     Pin conectado al siguelineas central.
     */
    Siguelineas(byte);
    //Constructor por defecto
```

```
    /**
     * Constructor que no necesita argumentos,
     * y que permite inicializar variables como las presentes en robot.h.
     * Las variables se inicializan a valores no recomendables
     * como pines inaccesibles.
     * IMPORTANTE: Sustituir todas las instancias no inicializadas
     *     o inicializadas de esta manera para que el modulo
     *     funcione como se espera.
     */
    Siguelineas();
    //Destructor


    //Inicializador
    /**
     * Permite a los pines establecidos en el constructor
     * recibir señales de los modulos siguelineas.
     */
    void inicializa();

    //Funciones
    /**
     * Accede al sensor izquierdo y devuelve su lectura.
     * @return TRUE si hay una linea debajo del sensor izquierdo.
     */
    bool readIzda();

    /**
     * Accede al sensor central y devuelve su lectura.
     * @return TRUE si hay una linea debajo del sensor central.
     */
     bool readCentro();

    /**
     * Accede al sensor derecho y devuelve su lectura.
     * @return TRUE si hay una linea debajo del sensor derecho.
     */
     bool readDcha();
};

#endif
```

C.3. Interfaz Morse

```
/**
 * ------------------------------------------------------------------
 * morse.h
 * Libreria que permite que la Arduino pueda enviar señales en Morse.
 * Necesita de un dispositivo que pueda cambiar notablemente de estado,
 * como un LED o una sirena.
 *
 * Adaptado por Alvar Tapia, Abril 2019.
 * Legado por Algorítmica y Complejidad, Universidad de Cantabria.
 * ------------------------------------------------------------------
 */

#ifndef morse_h
#define morse_h

#include "Arduino.h"


class Morse{
  /// Frecuencia por defecto de la sirena, en hercios.
  #define DEFAULT_FREC_SIRENA 18000
  /**
   * Tiempo que va a estar la señal activa para representar una raya,
   * en ms.
   */
  #define TIEMPO_RAYA 300
  /**
   * Tiempo que va a estar la señal activa para representar un punto,
   * en ms.
   */
  #define TIEMPO_PUNTO 100
  /**
   * Tiempo que va a estar la señal desactivada
   * para diferenciar simbolos morse, en ms.
   */
  #define TIEMPO_ENTRE_SIMBOLOS 100

  private:
    /**
     * Pin que transmitira la señal morse.
     * Puede conectarse a un LED o a una sirena.
     */
    byte PIN_MORSE;
    /**
     * Caracter que guarda el dispositivo de salida.
     * 'l' = LED, dispositivo que funciona de manera binaria,
     *       ENCENDIDO o APAGADO.
     * 's' = sirena, dispositivo que produce sonidos.
     *       Puede variar de frecuencia ademas
     *       de poder estar ENCENDIDO o APAGADO.
```

```
    */
    char DISPOSITIVO;
    /// Frecuencia a la que funcionara la sirena.
    unsigned int FREC_SIRENA;
  public:
    //Constructor
    /**
     * Constructor de la libreria.
     * @param Pin output
     *     Guarda el pin que se va a utilizar para enviar las señales.
     *     Debe estar conectado a un dispositivo que cambie notablemente
     *     de estado.
     * @param Tipo dispositivo
     *     'l' = Dispositivo tipo LED.
     *     's' = Dispositivo tipo sirena.
     * @param Frecuencia sirena
     *     Frecuencia a la que va a trabajar la sirena, en hercios.
     *     Si se esta trabajando con LEDs, este campo se puede ignorar.
     *     Los humanos podemos oír entre 20 Hz y 20 kHz.
     *     Las frecuencias mas bajas se perciben con mas volumen
     *     y pueden llegar a ser muy molestas.
     *     Por defecto, toma el valor de DEFAULT_FREC_SIRENA.
     */
    Morse(byte, char, unsigned int = DEFAULT_FREC_SIRENA);
    //Constructor por defecto
    /**
     * Constructor que no necesita argumentos, y que permite
     * inicializar variables como las presentes en robot.h.
     * Las variables se inicializan a valores no recomendables como
     * pines inaccesibles y caracteres que no tiene significado en la clase.
     * IMPORTANTE: Sustituir todas las instancias no inicializadas
     *     o inicializadas de esta manera para que
     *     el modulo funcione como se espera.
     */
    Morse();
    //Destructor


    //Inicializador
    /**
     * Permite al pin establecido en el constructor
     * mandar señales en Morse.
     */
    void inicializa();

    //Funciones
    /**
     * Manda una señal "raya", y espera un tiempo callado.
     */
    void raya();

    /**
```

```
    * Manda una señal "punto", y espera un tiempo callado.
    */
   void punto();

   /**
    * Manda un conjunto de señales que comunican "sos".
    * Al terminar el mensaje, espera un largo tiempo callado.
    */
   void sos();
};

#endif
```

C.4. Interfaz Robot

```
/**
 * ------------------------------------------------------------------
 * robot.h
 * Clase que gestiona los motores de un robot implementado
 * con placas Arduino.
 * Pueden aplicarse modulos para mejorar la gestion
 * del movimiento del robot.
 *
 * Adaptado por Alvar Tapia, Abril 2019.
 * Legado por Algorítmica y Complejidad, Universidad de Cantabria.
 * ------------------------------------------------------------------
 */

#ifndef robot_h
#define robot_h

#include "siguelineas.h"
//#include "infrarrojos.h" //A Abril 2019, esta biblioteca no se utiliza
#include "morse.h"
#include "bluetooth.h"

#include "Arduino.h"


class Robot {
  /// Velocidades por defecto de los motores del robot.
  /// Valores de 0-255, siendo 0 = motores parados, 255 = maxima potencia.
  #define VELOCIDAD_LENTA 100
  #define VELOCIDAD_GIRO 200
  #define MAX_VELOCIDAD 255

  private:
    /**
     * Enumerados que representan la direccion a la que va el robot.
     * Necesario para saber cuando arrancar.
     */
    enum Direccion {dirNull = -1, dirAlante, dirAtras,
                    dirGiroDcha, dirGiroIzda,
                    dirRotaDcha, dirRotaIzda};
    /// Direccion actual del robot. Se encuentra parado por defecto.
    Direccion dirActual = dirNull;


  protected:
    /// Tiempo que necesita arrancar el robot, en milisegundos.
    byte TIEMPO_ARRANQUE;

    /// Pines que controlan la direccion en la que giran los motores.
    byte PIN_IZDA_ALANTE;
    byte PIN_IZDA_ATRAS;
```

```
    byte PIN_DCHA_ALANTE;
    byte PIN_DCHA_ATRAS;
    /*
     * Para que el robot se mueva, ALANTE != ATRAS.
     * p.ej. ALANTE = HIGH y ATRAS = LOW hace
     *     que el motor se mueva hacia delante.
     * Pero ALANTE == ATRAS deja de alimentar los motores,
     *     independientemente de que ese valor sea HIGH o LOW.
     */

    /// Pines que controlan la velocidad a la que giran los motores.
    byte PIN_VEL_IZDA;
    byte PIN_VEL_DCHA;

  public:
    /**
     * Variables para modulos. Se acceden de forma directa,
     * tanto para modificarlos como para usar sus funciones.
     * Para utilizar los modulos, los objetos deben establecerse e inicializarse.
     */
    /// Modulo siguelineas
    Siguelineas SIGUELINEAS;
    /// Modulo mando infrarrojo
    //Infrarrojos INFRARROJOS; //A Abril 2019, este objeto no se utiliza
    /// Modulo morse
    Morse MORSE;
    /// Modulo bluetooth
    Bluetooth BLUETOOTH;

    //Constructores
    /**
     * Inicializa un robot cuyo driver tiene 4 entradas para los motores.
     * @param Pin motor izquierdo hacia alante
     * @param Pin motor izquierdo hacia atras
     * @param Pin motor derecho hacia alante
     * @param Pin motor derecho hacia atras
     *     Pines de direccion de los motores.
     * @param Pin de velocidad del motor izquierdo
     * @param Pin de velocidad del motor derecho
     *     Pines de control de velocidad de los motores.
     * @param Tiempo de arranque
     *     Tiempo que necesita el robot para arrancar, en milisegundos.
     *     Valores entre 0-255 ms.
     */
    Robot(byte, byte, byte, byte, byte, byte, byte);
    //Constructor por defecto
    /**
     * Constructor que no necesita argumentos,
     * y que permite inicializar variables como las presentes en Cnosos.h.
     * Las variables se inicializan a valores no recomendables
     * como pines inaccesibles.
     * IMPORTANTE: Sustituir todas las instancias no inicializadas
```

```
 *     o inicializadas de esta manera para que el robot
 *     funcione como se espera.
 */
Robot();
//Destructores


    //Inicializar
/**
 * Inicializa los pines para que el robot pueda alimentar los motores.
 */
void inicializa();


//Funciones del robot
/**
 * Permite modificar el tiempo que va a estar arrancando el robot.
 * @param Tiempo de arranque
 *     Tiempo en milisegundos que estara el robot a maxima velocidad.
 *     Valores entre 0-255 ms.
 */
void setTiempoArranque(byte);


//Velocidad del robot
/**
 * Aplica mucha tension durante un periodo de tiempo definido
 * para que los motores puedan empezar a girar.
 */
void arranca();

/**
 * Establece la velocidad actual del robot.
 * @param Velocidad
 *     Velocidad a la que ira el robot.
 *     Valores entre 0-255.
 */
void setVelocidad(byte);

/**
 * Deja de alimentar los motores y para el robot.
 * Puede no ser inmediato dependiendo de la inercia que lleve.
 */
void para();

/**
 * Alimenta los motores al 40% de la capacidad de las pilas.
 */
void lento();

/**
 * Establece una velocidad moderada,
```

```
 * en la que el robot es siempre capaz de girar.
 */
void velGiro();

/**
 * Alimenta los motores al 100% de la capacidad de las pilas.
 */
void maxVelocidad();


      //Cambios de direccion
/**
 * El robot deja de alimentar los motores de la izquierda
 * para poder girar a la izquierda.
 * Permite girar mientras el robot va marcha atras.
 * @param Velocidad
 *    Se puede indicar la velocidad a la que el robot hace la maniobra.
 *    Por defecto, se utiliza VELOCIDAD_GIRO.
 */
void giraIzda(byte = VELOCIDAD_GIRO);

      /**
 * El robot deja de alimentar los motores de la derecha
 * para poder girar a la derecha.
 * Permite girar mientras el robot va marcha atras.
 * @param Velocidad
 *    Se puede indicar la velocidad a la que el robot hace la maniobra.
 *    Por defecto, se utiliza VELOCIDAD_GIRO.
 */
void giraDcha(byte = VELOCIDAD_GIRO);


//Cambios de direccion de los motores
/**
 * Los motores giran de manera que el robot vaya hacia delante.
 * @param Velocidad
 *    Se puede indicar la velocidad a la que el robot hace la maniobra.
 *    Por defecto, se utiliza VELOCIDAD_LENTA.
 */
void alante(byte = VELOCIDAD_LENTA);

/** Los motores giran de manera que el robot vaya hacia atras.
 * @param Velocidad
 *    Se puede indicar la velocidad a la que el robot hace la maniobra.
 *    Por defecto, se utiliza VELOCIDAD_LENTA.
 */
void atras(byte = VELOCIDAD_LENTA);

/**
 * El robot pivota hacia la izquierda haciendo
 * que los motores de la izquierda vayan hacia atras y
 * que los motores de la derecha vayan hacia delante.
```

```
     * @param Velocidad
     *    Se puede indicar la velocidad a la que el robot hace la maniobra.
     *    Por defecto, se utiliza VELOCIDAD_GIRO.
     */
    void rotaIzda(byte = VELOCIDAD_GIRO);

    /**
     * El robot pivota hacia la derecha haciendo
     * que los motores de la derecha vayan hacia atras y
     * que los motores de la izquierda vayan hacia delante.
     * @param Velocidad
     *    Se puede indicar la velocidad a la que el robot hace la maniobra.
     *    Por defecto, se utiliza VELOCIDAD_GIRO.
     */
    void rotaDcha(byte = VELOCIDAD_GIRO);
};


/**
 * ROBOTS MODELO
 * Para poder trabajar con ellos es necesario utilizarlos
 * desde una variable, dado que estos objetos son constantes,
 * y si se intenta realizar una funcion con ellos saltaran errores.
 */
/// Configuracion de los robots Arduino
const Robot arduino(255, 12, 255, 13, 10, 11, 80);
// El pin 255 es inaccesible, y permite escribir "digitalWrite"s
// sin afectar al funcionamiento del robot

/// Configuracion de los robots Elegoo
const Robot elegoo(6, 7, 9, 8, 5, 11, 0);

#endif
```

C.5. Interfaz Cnosos

```
/**
 * ------------------------------------------------------------------
 * Cnosos.h
 * Libreria que lee, reconoce y navega grafos con robots Arduino.
 *
 * Adaptado por Alvar Tapia, Abril 2019.
 * Legado por Algorítmica y Complejidad, Universidad de Cantabria.
 * ------------------------------------------------------------------
 */

#ifndef cnosos_h
#define cnosos_h

#include "robot.h"

class Cnosos {
  /// Redefiniciones de metodos que se usaran a lo largo de la clase.
  // Reducen y hacen mas intuitivo el codigo.
  #define SENSOR_IZDA robot.SIGUELINEAS.readIzda()
  #define SENSOR_CENTRO robot.SIGUELINEAS.readCentro()
  #define SENSOR_DCHA robot.SIGUELINEAS.readDcha()

  /**
   * Se espera este tiempo en milisegundos
   * para confirmar que se ha leido una marca.
   */
  #define TIEMPO_CONFIRMACION 0

  /** Cada vez que tiene que procesar una entrada, el robot "pensara"
   * (se quedara quieto) este numero de milisegundos.
   */
  #define TIEMPO_PENSAR 300
  private:
    /// Guarda la configuracion del robot que se va a usar en este objeto.
    Robot robot;
    /// Numero de bits que usaran los nodos para identificarse.
    byte BITS;
  public:
    //Constructores
    /**
     * Constructor de la libreria Cnosos.
     * No necesita un metodo inicializar,
     * dado que se supone que los parametros ya estan inicializados.
     * @param Robot a controlar
     *     Objeto Robot con el que se va a trabajar.
     *     Se espera que el Robot este inicializado,
     *     asi como su Siguelineas.
     *     Si se necesita debuggear o que el robot
     *     se comunique con el usuario, es necesario inicializar
     *     el modulo Morse o el modulo Bluetooth, o ambos.
```

```
 * @param Numero bits en las etiquetas
 *    Numero de bits que se usaran para identificar
 *    cada nodo del mapa.
 *    Se espera que el tamaño del parametro sea adecuado,
 *    dado que si se utiliza el valor maximo (255),
 *    se podrian identificar 2^255 nodos, que son mas de 10^76 nodos.
 *    Por defecto, se utilizaran 3 marcas
 *    para el identificador de cada nodo.
 */
Cnosos(Robot, byte = 3);
//Constructor por defecto
/**
 * Constructor que no necesita argumentos,
 * y que permite inicializar un objeto Cnosos.
 * Las variables se inicializan a valores no recomendables
 * como pines inaccesibles.
 * IMPORTANTE: Sustituir todas las instancias no inicializadas o
 *    inicializadas de esta manera para que el objeto
 *    funcione correctamente.
 */
Cnosos();
//Destructor


//Funciones
/**
 * Avanza por la linea hasta encontrar un hueco en la cinta
 * o una cinta atravesada. Sobrepasa el indicador.
 * @return  0 si ha encontrado un hueco en la cinta,
 *          1 si ha encontrado una cinta atravesada.
 */
byte siguiente();

/**
 * Lee un identificador de nodo.
 * Este metodo supone que se acaba de cruzar
 * el marcador de comienzo del identificador.
 * Por defecto, ese marcador es un hueco en la cinta.
 */
byte lee_numero();

/**
 * Recorre dos veces el perimetro del nodo
 * para obtener gran cantidad de datos.
 * Al terminar el metodo, el robot estara situado
 * despues del identificador del nodo.
 * @param[out] Identificador del nodo
 *    Devuelve el numero que identifica a este nodo.
 * @param[out] Grado del nodo
 *    Devuelve el grado (o el numero de cruces)
 *    que tiene este nodo.
 * @param[out] Cruce por el que ha entrado el robot al nodo
```

```
 *     Devuelve el numero del cruce por el que ha entrado el nodo.
 *     Si el robot empezo a recorrer el nodo justo detras
 *     del identificador, este numero sera 0,
 *     si se empezo detras del primer cruce sera 1...
 *     asi hasta 'grado', que significaria que el robot
 *     ha empezado a recorrer el nodo antes del identificador.
 */
void lee_nodo(byte&, byte&, byte&);

/**
 * En cuanto se encuentre un cruce,
 * toma ese cruce y el robot pasa al siguiente nodo.
 */
void sal_aqui();

/**
 * Toma un cruce en concreto desde la posicion
 * en la que este el robot.
 * @param Cruce a tomar
 *     Numero del cruce que se quiere tomar,
 *     siendo 1 el primer cruce de la etiqueta.
 * @param Grado del nodo
 *     Ultimo cruce que puede tomar el robot.
 *     Tambien representa el grado (o el numero de cruces)
 *     que hay en el nodo.
 * @param Cruce actual
 *     Indica cual es el ultimo cruce que el robot se ha saltado.
 *     Por defecto, se considera que el robot esta
 *     justo detras de la etiqueta, por lo que
 *     no se ha saltado ningun cruce (este valor es 0).
 */
void sal(byte, byte, byte = 0);

/**
 * Toma una salida a la izquierda.
 */
void sal_izq();

/**
 * Comunica un numero al usuario por medio de modulos.
 * Necesita tener inicializado el modulo Morse
 * o el modulo Bluetooth para comunicarse con el usuario.
 * @param Numero a comunicar
 *     Envia este numero, ya sea con el mismo numero de pitidos,
 *     o con un mensaje Bluetooth.
 */
void luce_numero(byte);

/**
 * Se ha encontrado un suceso inesperado. Se para y grita SOS.
 * Necesita tener inicializado el modulo Morse
 * o el modulo Bluetooth para comunicarse con el usuario.
```

```
     */
    void error();
};

#endif
```

Referencias

Documentación sobre el uso de analogWrite() en pines digitales (inglés).
https://www.arduino.cc/en/Tutorial/PWM

Documentación analogRead() (inglés).
https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

Documentación sobre registros de la Arduino Uno (inglés).
https://www.arduino.cc/en/Reference/PortManipulation

Documentación sobre la Motor Shield de Arduino Uno que estamos utilizando (inglés).
http://www.mantech.co.za/ProductInfo.aspx?Item=15M4321

J. Liberty. *SAMS teach yourself C++ in 24 hours*, pages 85–99. SAMS, Indianapolis, Ind., 2 edition, 1999.