Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de un robot para registro de variables ambientales en invernadero

Design of a robot to record environmental variables in greenhouse

Autor

Mikel Brun Martínez

Director/es

Jose Luis Santolaya Sáenz

Enrique Tardío Monreal

Grado en Ingeniería de Tecnologías Industriales

Mayo de 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Mikel Brun Martínez,

con nº de DNI 73120822-G en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Diseño de un robot para registro de variables ambientales en invernadero,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19 de abril de 2017

Fdo: Mikel Brun Martínez

RESUMEN

Diseño de un robot para registro de variables ambientales en invernadero

El objetivo de este proyecto es el diseño de un robot para medir las variables ambientales y permitir el control constante de las condiciones de los cultivos dentro de un invernadero. El robot se encarga de recoger datos de temperatura, humedad, luminosidad y concentración de CO_2 haciendo un recorrido programado por la instalación y enviando los datos a una estación central.

Se han diseñado los sistemas electromecánicos que permiten el desplazamiento del robot y el registro de datos a lo largo del invernadero. Para ello, se han definido una serie de condiciones de operación, monitoreo y control, se han seleccionado los sensores de medida, se han propuesto soluciones de las diferentes partes del robot de acuerdo con las especificaciones iniciales y se ha desarrollado la programación que permite una realización autónoma de tareas.

Para que el robot pueda recorrer el invernadero, se ha propuesto un diseño de vehículo que se mueve de manera controlada y tiene capacidad para realizar desplazamientos en línea recta y cambios de dirección. Además se ha incorporado un sistema de detección de obstáculos que envía una señal de alarma en el caso de que el vehículo quede detenido y no pueda realizar el recorrido previsto. El robot lleva incorporadas unas baterías recargables con autonomía mínima de una hora. Los datos tomados son enviados a una estación central desde la cual se establece comunicación mediante un dispositivo inalámbrico. Un ordenador portátil es utilizado como estación central.

El microcontrolador elegido ha sido Arduino y el control de los dispositivos del robot se realiza con Arduino Ide. El lenguaje de programación soportado por Arduino Ide es C++. Todos los componentes electrónicos se han integrado dentro de un dispositivo móvil compacto, que dispone de un motor paso a paso para controlar desplazamientos y de un motor servo para realizar cambios de dirección. Los sensores para medida de variables son compatibles con el entorno de programación.

Finalmente, el diseño propuesto ha sido construido y montado. A partir de este prototipo se han realizado las primeras pruebas de funcionamiento en un entorno físico de dimensiones reducidas.

ÍNDICE

1. INTRODUCCIÓN

En un invernadero, la producción es capaz de verse aumentada considerablemente si se controlan adecuadamente las variables medioambientales del entorno, ya que se favorece el ciclo vegetativo de las plantas (fotosíntesis). Por lo tanto, es de vital importancia el poder diseñar sistemas capaces de poder monitorizar diferentes variables como la temperatura o el porcentaje de humedad.

En el control de parámetros ambientales se pueden utilizar sistemas de sensores distribuidos convenientemente a lo largo de la instalación y conectados a un ordenador central que almacena los datos. Sin embargo con el desarrollo de la robótica es posible monitorizar variables de una manera más económica y práctica, utilizando para ello un equipo móvil que disponga de una serie de sensores de medida.

1.1. Tipos de robots móviles

Los robots móviles que se pueden encontrar en el mercado utilizan ruedas, orugas o patas para desplazarse por el entorno. Dentro de las posibles opciones se descartan las orugas y las patas por los siguientes motivos:

- Los robots con patas requieren diseños más complejos, son en general más caros y no aportan ninguna ventaja respecto a otros sistemas de locomoción para el entorno en el que se va a mover el robot. Un robot con patas puede ser más inestable en terrenos irregulares si no se le dota del suficiente número de patas, lo que significa un incremento de motores, consumo eléctrico y complejidad del sistema.
- Los robots de oruga son perfectos para moverse por terrenos irregulares pero consumen bastante energía, tienen problemas para realizar giros, y se mueven con lentitud.

Por ello, se opta por utilizar ruedas para el desplazamiento. A partir de aquí hay varias opciones a considerar que se detallan a continuación:

Sistema de dos ruedas diferencial

Este robot lleva dos ruedas y cada rueda tiene su propio motor. Para conseguir la estabilidad del robot se añade como mínimo un tercer elemento de apoyo que es pasivo. Este elemento puede ser un simple apoyo, una pequeña bola o una o dos ruedas de giro libre. Suelen ser robots de diseño triangular, romboidal o rectangular. En la figura 1 puede verse un ejemplo de este tipo de sistema.

Se llaman robots diferenciales porque el giro se consigue haciendo girar las ruedas a diferente velocidad. Incluso se puede hacer girar las ruedas en sentido opuesto para giros bruscos. Programar las ruedas con diferentes velocidades es una tarea sencilla y por eso este robot es el que tiene menor complejidad tanto mecánica como de control.

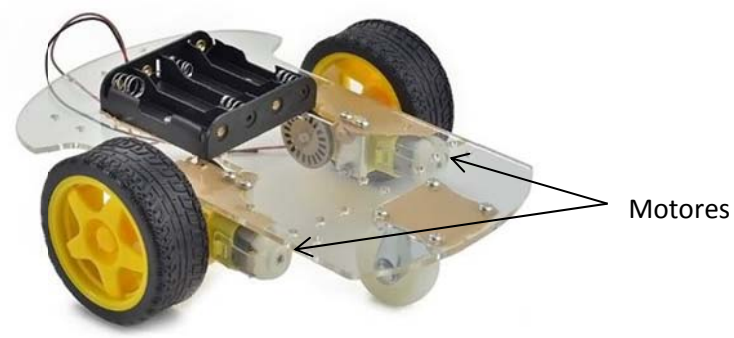


Figura 1. Sistema de dos ruedas diferencial

Sin embargo, conseguir que este robot siga una trayectoria recta es problemático. Cualquier mínima diferencia de velocidad, bien por una tolerancia en el montaje, bien por irregularidades del terreno, hace que el robot gire. Basta con que las ruedas pisen diferentes superficies para que sufran una fricción diferente y giren a diferentes velocidades (y por tanto se produzca un giro). Este tipo de robots necesitan un sistema de supervisión que indique si se deben variar las velocidades de los motores para mantener el avance en línea recta.

Sistema de tres ruedas sincronizadas

En este diseño (Figura 2) hay generalmente tres ruedas y las tres son motrices pero movidas por un único motor. Las tres ruedas están alineadas. El giro se consigue con un segundo motor y unos dispositivos mecánicos que giran las tres ruedas un ángulo determinado, garantizando que el robot sigue la nueva dirección marcada.

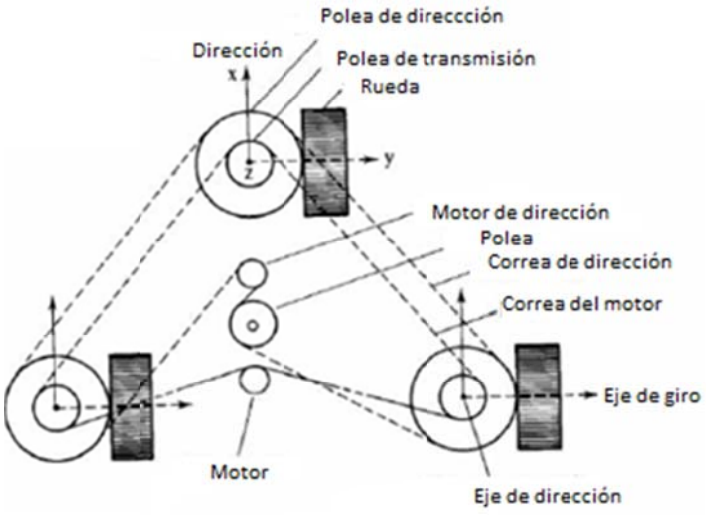


Figura 2. Esquema de sistema de tres ruedas sincronizadas

El motor que controla el movimiento transmite el par usando unas correas y en cada rueda hay un dispositivo que transforma ese movimiento en el movimiento de giro del eje de la rueda. El motor de dirección también hace girar el conjunto de las tres ruedas por medio de otras correas. La ventaja de este sistema es su precisión y el gran inconveniente es su complejidad mecánica.

Sistema de tipo triciclo

Este robot también es de tres ruedas pero hay una división entre ruedas motrices y ruedas de dirección. Normalmente las ruedas traseras son motrices y la rueda delantera es de dirección, como se puede ver en la Figura 3.

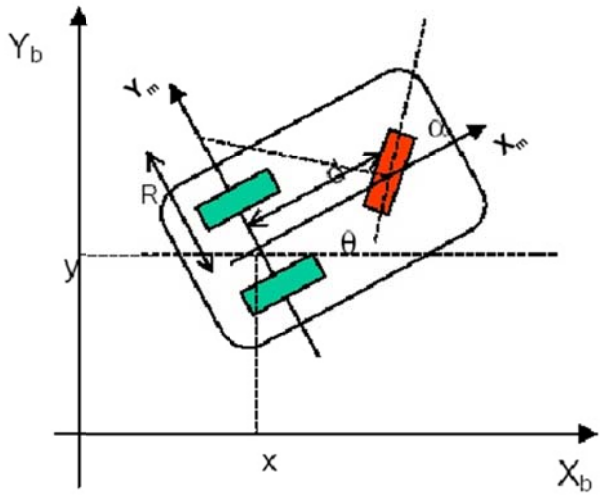


Figura 3. Esquema de sistema de tipo triciclo

Los diseños de tipo triciclo tienen dos motores. Un motor normal que acciona las ruedas de tracción (y al haber solo un motor se garantiza que la velocidad de giro de las ruedas es la misma) y un motor tipo servo que acciona la dirección. La rueda de dirección suele ser una rueda orientable centrada.

Sistema de cuatro ruedas

Los robots de cuatro ruedas son los mejores para terrenos con una cierta irregularidad ya que tienen una gran estabilidad. Para su diseño hay dos posibilidades:

- Diseño con ruedas fijas y giro diferencial.
- Diseño con ruedas de dirección y algún mecanismo que permita su giro.

El diseño diferencial tiene como ventaja su simplicidad mecánica y como inconvenientes la necesidad de tener cuatro motores (Figura 4), uno por rueda, y una mayor complejidad en la programación.

Como cada rueda lleva su propio motor cada una de ellas puede girar en un sentido y con una velocidad diferente por lo que se pueden hacer giros realmente complejos pero es difícil programarlos con precisión y se requieren muchas pruebas para conseguir resultados satisfactorios.

Figura 4. Robot de cuatro ruedas diferencial

Un diseño alternativo consistiría en utilizar solo dos motores de manera que haya o bien dos ruedas motrices o bien que cada uno de los dos motores accione las ruedas de uno de los lados. Estos diseños presentan las siguientes características:

- El diseño diferencial con solo dos ruedas motrices presenta problemas en los giros. Si en los robots de dos ruedas ya es difícil tener precisión, la inercia de tener otras dos ruedas complica el hacer giros precisos.
- Para que un motor accione las dos ruedas de un lado se necesitaría algún mecanismo, bien por correas o por engranajes, que transmita el par motor a las ruedas. Este diseño puede resultar apropiado para un robot con orugas, ya que la misma oruga da el método de transmisión del par, pero en un robot con cuatro ruedas supone una complejidad adicional en el diseño y además se necesitarán motores más potentes por las pérdidas inherentes a la transmisión. Es un diseño muy poco común.

Si se necesitan giros bruscos e incluso pivotar en el sitio el único diseño posible consiste en usar ruedas omnidireccionales de manera que cada una esté controlada por un servo. Este diseño requiere un mínimo de cuatro motores para accionar las ruedas y un motor servo por cada una de las ruedas que pueda girar, lo que nos da un diseño mucho más complejo.

Si no se necesitan giros bruscos se puede ir a un diseño tipo coche. El diseño más común consiste en que las ruedas traseras sean de tracción y las ruedas delanteras de dirección. Las ruedas traseras se accionan por uno o dos motores y en las delanteras se instala un dispositivo para controlar el giro de las ruedas.

Actualmente se están desarrollando diferentes tipos de robots adaptados a invernaderos, en algunos casos con un importante nivel de sofisticación para adaptarse a las características de este tipo de instalaciones lograr un control preciso del mayor número de variables.

Así por ejemplo se pueden encontrar robots autónomos orientándose con una cámara (Figura 5), o incluso robots mucho más complejos que cuentan con una plataforma para que se apoye un dron y tome medidas en diferentes alturas y localizaciones específicas (Figura 6).

Figura 5. Robot con cámara

Figura 6. Robot con dron

1.2. Parámetros de control

Las variables más importantes a controlar dentro de un invernadero son la temperatura, la humedad relativa del aire, el nivel de luminosidad (radiación), y la concentración de dióxido de carbono en el aire, ya que son los factores climáticos que más influyen en el cultivo en sus diferentes fases de crecimiento.

Temperatura

En todo cultivo se requiere conocer la temperatura mínima letal, que es aquella por la que debajo de ésta la planta no se desarrolla correctamente, y las temperaturas máxima y mínima biológicas que dan el rango óptimo de cultivo de la planta.

Humedad relativa

Indica la cantidad de agua que está presente en el aire, en relación con la máxima que sería capaz de contener a la misma temperatura. Saber su valor es importante ya que afecta tanto al rendimiento como al crecimiento del cultivo. Por debajo de un valor mínimo las plantas no se desarrollan y si supera un valor máximo (normalmente entre un 60-70 %), puede hacer que aparezcan enfermedades.

Luminosidad

La luminosidad trata de medir la cantidad de luz que incide (ilumina) en un objeto. Desde el punto de vista físico se relaciona con la cantidad de fotones con una frecuencia determinada en el espectro visible que inciden o son emitidos por un objeto en un determinado período de tiempo. La unidad estándar para medir la luminosidad es el lumen mientras que el lux se puede considerar como una densidad derivada del lumen, ya que se define como lumen/m^2.

La luz influye directamente en la fotosíntesis de las plantas. Como consecuencia, garantizar unas correctas condiciones de luminosidad va a beneficiar a su correcto desarrollo. Por otra parte, el nivel de luminosidad óptimo depende de la temperatura.

$Concentración\ de\ CO_2$

El dióxido de carbono influye en la fotosíntesis de las plantas, y se necesita una concentración adecuada de CO_2 para cada combinación de luminosidad y temperatura. En algunos cultivos se incrementa de forma premeditada la cantidad de CO_2 en el invernadero para favorecerla. Esto puede aumentar la velocidad de cultivo en aproximadamente en un 20%, generar un aumento del rendimiento en un 25-30% y obtener una mejor calidad de la cosecha. Los rangos óptimos de asimilación de CO_2 están entre 18-25 ^{o}C.

1.3. Objetivo del proyecto

El objetivo de este proyecto es el diseño y desarrollo de un prototipo de robot con capacidad para desplazarse y tomar medidas de diferentes variables ambientales en invernaderos. El movimiento y adquisición de datos se realizará de forma programada.

Se considera necesario que sea un vehículo dotado de un sistema que permita controlar su recorrido, que pueda adaptarse a un terreno con irregularidades y en el que se pueden encontrar obstáculos y que además pueda registrar y transmitir las medidas realizadas a una estación de control.

Para ello, en este trabajo se pretende seleccionar los componentes electromecánicos que permitan realizar un diseño de robot de acuerdo con las especificaciones previstas. A continuación se llevará a cabo su montaje y programación en un prototipo de pruebas y finalmente se ensayará su funcionamiento.

En los próximos apartados se desarrolla el trabajo realizado. En el Capítulo 2, se hace el estudio de la parte mecánica y eléctrica del robot, explicando los métodos empleados en la selección de componentes. En el Capítulo 3 se describe la estructura y desarrollo del programa de control para que el robot realice su función dentro del invernadero. Además, toda la información relacionada con las características de los componentes electromecánicos utilizados, el software empleado para la programación y control del robot y el programa completo desarrollado, se exponen en los Anexos.

2. SISTEMA ELECTROMECÁNICO

Para que el robot pueda recorrer el invernadero, ha de disponer de un sistema que le permita moverse de una manera controlada realizando desplazamientos en línea recta y cambios de dirección. Además, dispondrá de los elementos necesarios para el control del robot y la adquisición de datos relativos a las variables ambientales. Se prevé la utilización de un sistema de detección de obstáculos que envíe una señal de alarma en el caso de que el robot quede detenido y no pueda realizar el recorrido previsto.

2.1. Sistema Mecánico

De todas las opciones consideradas previamente, se ha optado por un diseño de cuatro ruedas, con tracción en el eje trasero y control de la dirección a través de las ruedas delanteras. Este sistema es estable, robusto y relativamente sencillo mecánicamente. A continuación se describe el bastidor y los sistemas de tracción y dirección planteados.

2.1.1. Bastidor

El sistema de tracción en el eje trasero y el sistema de dirección actuando en las ruedas delanteras del vehículo se montan sobre un bastidor, tal y como se muestra en la Figura 7. El bastidor es el soporte de los elementos mecánicos y a su vez sirve de apoyo para los soportes de los ejes de cada una de las ruedas.

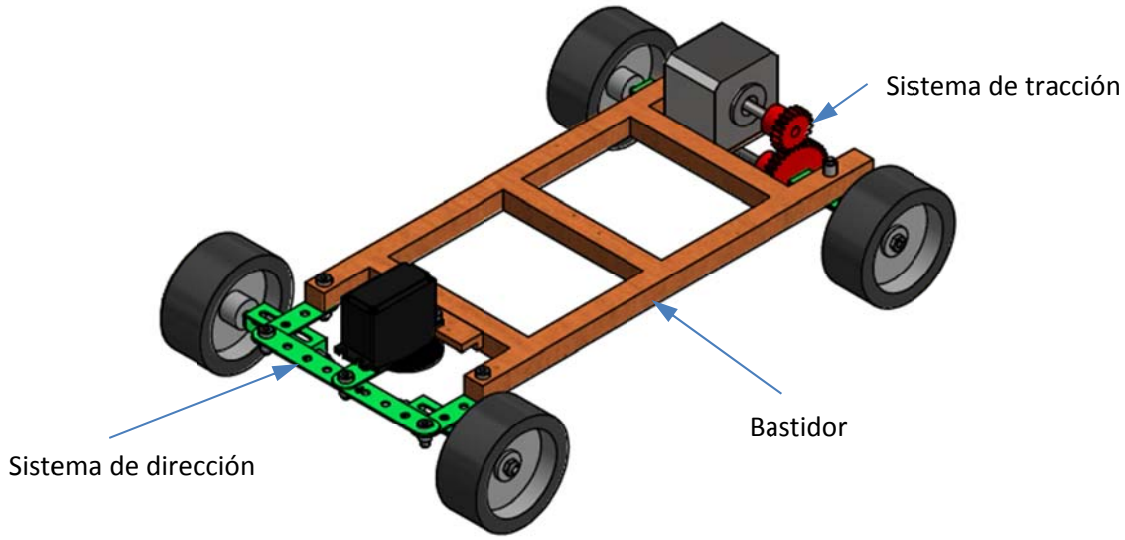


Figura 7. Bastidor con tracción y dirección

Se ha optado por una solución sencilla de bastidor, formado por dos largueros y tres travesaños, en el que se sitúan los soportes para los ejes de cada rueda y que mantienen una separación entre el eje delantero y el eje trasero de 250 mm y una separación entre el plano medio de las ruedas de 177 mm (como puede consultarse en el Anexo II). La altura del bastidor es de 47 mm. Se ha optado por ruedas de 60 mm de diámetro que permiten tener un chasis elevado y neumáticos con taco para mejorar el agarre en terrenos irregulares.

2.1.1. Sistema de Tracción

La tracción se llevará a cabo a través de un único motor. Se pueden utilizar dos tipos principales de motores: de corriente continua, y paso a paso.

Los motores de corriente continua son pequeños motores eléctricos que giran a diferentes velocidades, mientras que los motores paso a paso, tienen la gran ventaja de que giran de manera discreta por "pasos" que pueden ser medidos y eso permite calcular con una gran precisión la distancia recorrida por el robot.

Como en este caso no se dispone de ningún sistema de control de la trayectoria, como podría ser una cámara, se ha seleccionado un motor paso a paso para tener un control de la posición y la trayectoria del robot.

El motor ha de contar con una potencia y un par suficientes para poder mover el robot. Para calcular el par necesario que ha de suministrar el motor paso a paso es preciso conocer la fuerza de rozamiento a superar de las ruedas traseras. Para ello, se plantea un sistema sencillo como sigue.

Se supone el peso del robot en 1,5 kg. Debido al peso del motor paso a paso, el eje trasero soportará más carga que el delantero. Por tanto, se va a suponer que el centro de gravedad del conjunto está situado en una distancia de 1/3 del eje trasero. La distancia entre ejes es de 250 mm. En la Figura 8 se ve el diagrama de fuerzas sobre cada eje.

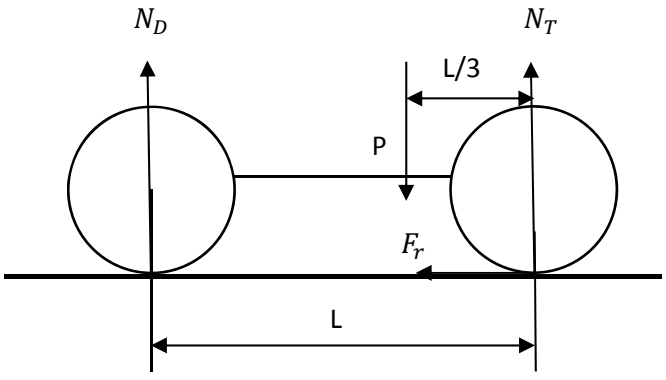


Figura 8. Diagrama de Fuerzas

Con ello, según equilibrio de fuerzas en el eje vertical y de momentos en el eje trasero, la normal trasera se calcula como sigue:

$$N_D + N_T = P$$

$$N_D L = P\frac{L}{3} \rightarrow N_D = \frac{P}{3}$$

$$\frac{P}{3} + N_T = P \rightarrow N_T = \frac{2}{3}P = \frac{2}{3} \cdot 1{,}5\ kg = 1\ kg$$

La fuerza de rozamiento que actúa en el eje trasero es de:

$$F_R = \mu \cdot N_T = 0{,}8 \cdot 1\ kg = 0{,}8\ kg$$

Se va a suponer que la fuerza rozamiento está repartido por igual entre las dos ruedas traseras.

El par mínimo necesario para mover el eje trasero será aquel que venza el rozamiento. El par de la fuerza de rozamiento se calcula como:

$$M_{F_R} = F_R R = 0{,}8\ kg \cdot 30\ mm = 2{,}4\ kg \cdot cm$$

Al haber un reductor con una relación 1:2, el par mínimo que ha de suministrar el motor será de:

$$M_{motor} = \frac{1}{2}M_{F_R} = \frac{1}{2} \cdot 2{,}4 = \ 1{,}2\ kg \cdot cm$$

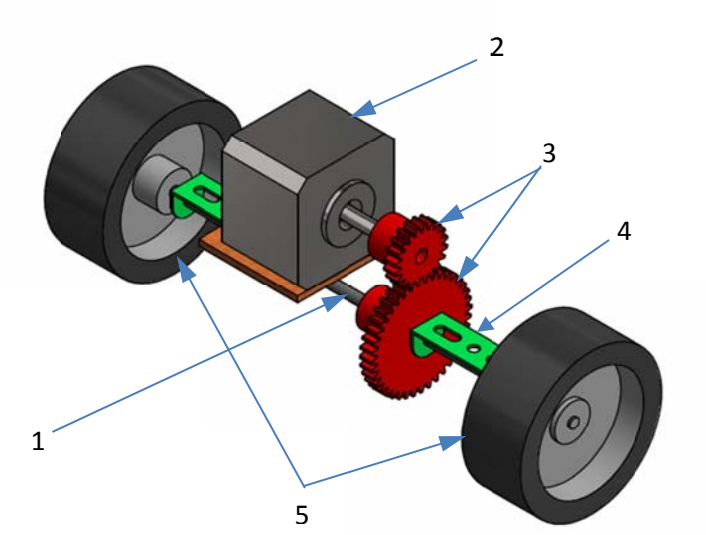
Se ha seleccionado un motor capaz de proporcionar un par de 1.6 kg.cm. El motor de tracción elegido es el motor paso a paso Adafruit Nema 17, de tipo bipolar con 4 cables de conexión.

La gran ventaja de los motores paso a paso es que giran de manera discreta por “pasos” que pueden ser medidos y eso permite calcular con una gran precisión la distancia recorrida por el robot. Su consumo es de 350mA a 12V. Como el controlador de motores se alimentará a 5V la intensidad equivalente a 5V se estima igualando potencias.

$$I = \frac{12 \times 0.350}{5} = 0{,}84A$$

El sistema de tracción se ha modelado como se muestra en la Figura 9. Consta de dos engranajes rectos, uno conectado a la salida del motor, y otro al eje de las ruedas. De esta manera, el movimiento es capaz de ser transmitido de un eje a otro.

Se ha considerado oportuno establecer una relación 1:2. La rueda dentada que va montada en el motor tiene 20 dientes y la rueda dentada que va montada en el eje trasero tiene 40 dientes.

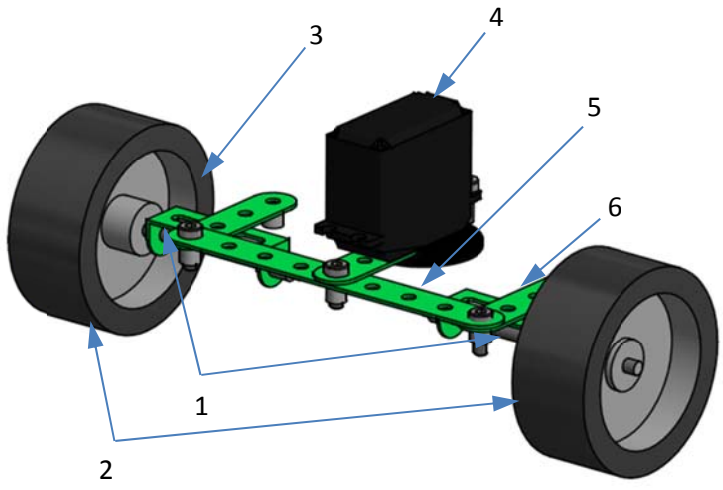


1. Eje trasero
2. Motor paso a paso
3. Transmisión de engranajes
4. Soporte eje
5. Ruedas traseras

Figura 9. Esquema Sistema de Tracción

2.1.2. Sistema de Dirección

Se propone por sencillez un sistema que permita girar las dos ruedas a la vez, en vez de tener un sistema independiente de giro para cada rueda. Para ello se va a utilizar un motor servo que permite fijar la posición circular de un determinado elemento. El ángulo suele variar entre 0° y 180°. Para transmitir el giro a las ruedas delanteras se ha dispuesto un sistema de barras que unido al bastidor actúa como mecanismo articulado. En la Figura 10 se puede ver la disposición de los elementos que componen la dirección.



1. Ejes delanteros
2. Ruedas delanteras
3. Soporte eje
4. Motor servo
5. Barra de dirección
6. Barra apoyo chasis

Figura 10. Esquema Sistema de Dirección

Para controlar la dirección se ha elegido un motor servo de Adafruit modelo Towerpro SG-5010. Tiene un par de 5.5 Kg·cm que se considera suficiente para los requerimientos.

Según la dinámica angular el momento para que gire una rueda es igual a $M = I\alpha$ donde I es el momento de inercia de la rueda y α es la aceleración angular. La rueda se puede considerar como un disco que gira por un eje que pasa por su centro y el momento de inercia vale $I = \frac{m_R R^2}{4}$ donde m_R es la masa de la rueda y R su radio. La aceleración angular se puede estimar considerando que en un tiempo t la rueda gira un ángulo θ partiendo del reposo y por tanto se cumple $\theta = \frac{1}{2}\alpha t^2 \Longrightarrow \alpha = \frac{2\theta}{t^2}$. Se puede estimar α con el dato del fabricante de que en 0,2 segundos la rueda gira 60º. Expresado en radianes $\theta = \frac{60 \cdot 2\pi}{360} = 1{,}047 rad$. La aceleración angular es $\alpha = \frac{1{,}047 \cdot 2}{0{,}2} = 10{,}472\, rad/s^2$.

Considerando una rueda de $30gr = 0{,}03Kg$ con un radio de $3cm = 0{,}03m$ el momento de inercia vale $I = \frac{0{,}03 \cdot 0{,}03^3}{4} = 0{,}00000675Kg \cdot m^2$ y por tanto el par para mover una rueda es de $\tau \simeq 0{,}00003N \cdot m$. Para mover las dos ruedas se necesita el doble del par, luego el par total para mover las ruedas es el siguiente:

$$\tau_1 = 0{,}00007N \cdot m$$

Si el peso sobre la rueda fuera grande se produciría una deformación de la rueda y el rozamiento ejercería un momento que se opondría al giro de la rueda pero al ser el peso pequeño no se produce esa deformación y la superficie de contacto con el suelo es muy pequeña y por tanto se puede despreciar el momento que ejerce la fuerza de rozamiento.

Para ejercer ese momento hay dos varillas junto a las ruedas que giran a una aceleración similar. El momento de inercia de una varilla que gira por un de sus extremos es $I = \frac{m_v L^2}{3}$.

El giro se transmite a las dos varillas por otra varilla que está conectada al motor servo luego se tienen tres varillas que giran con una determinada aceleración y por tanto el momento total para girar el sistema de dirección es el siguiente:

$$M_D = 3\frac{m_v L^2}{3}\alpha = m_v L^2 \alpha$$

Si se considera las varillas de 3cm que pesen unos 10gr cada una el par necesario para hacerlas girar es el siguiente:

$$\tau_2 = 0{,}01 \times 0{,}03^2 \times 10{,}472 \simeq 0{,}00009N \cdot m$$

La suma total de los momentos es $\tau_1 + \tau_2 \simeq 0{,}0001N \cdot m$. El par del motor de $0{,}54N \cdot m$ es más que suficiente para el objetivo.

Este motor trabaja a 5V y el fabricante no da el consumo eléctrico pero se puede estimar de la siguiente manera:

- La velocidad media, según el catálogo del fabricante, es de 0.2 segundos para girar 60°, como 60° son $60 \times \pi/180 = 1{,}047 rad$ la velocidad angular es de $1.047/0.2 = 5{,}235\, rad/seg$.
- La potencia es momento por velocidad angular luego la potencia desarrollada es de $0.54 \times 5.235 \simeq 2.83w$

- Para tener en cuenta las pérdidas se aconseja multiplicar esta potencia por un factor 1.5 luego la potencia de cálculo será $4.245w$
- Por otra parte la potencia eléctrica es voltaje por intensidad luego la intensidad mínima para obtener esta potencia es la siguiente (notar que esta potencia está medida a 4.8V):

$$\frac{4.245}{4.8} = 0.884 \simeq 0.9A = 900mA$$

2.2. Electrónica de Control

Se requieren elementos que permitan controlar y dirigir las acciones de los motores que forman parte de la tracción y del sistema de dirección del robot. A continuación se describen los componentes seleccionados y sus principales características para lograr el funcionamiento correcto del robot. Para más especificaciones en concreto de cada componente seleccionado se puede acudir al Anexo I.

2.2.1. Microcontrolador

Se puede definir microcontrolador como un ordenador muy sencillo en el que un procesador es capaz de ejecutar una serie de órdenes almacenadas en memoria. El conjunto de órdenes almacenadas en memoria que ejecuta el microcontrolador es el programa de control.

Un microcontrolador consta de tres bloques fundamentales, que son los mismos que se pueden considerar en cualquier en cualquier tipo de ordenador: procesador o unidad central de procesamiento (en inglés CPU, Central Process Unit), memoria y periféricos de entrada/salida. El procesador en capaz de entender un conjunto de órdenes definido, que se denomina código máquina. Con este conjunto de instrucciones se pueden elaborar programas que permiten definir las tareas que tiene que hacer el microcontrolador. Tanto el programa como los datos necesarios para su ejecución tienen que estar almacenados en la memoria. El microcontrolador se comunica con el entorno utilizando periféricos.

Existen muchos microcontroladores en el mercado y de muy diversos precios. Para el desarrollo del proyecto se han elegido microcontroladores Arduino. Se han considerado dos alternativas: Arduino Uno es la paca aconsejada para proyectos sencillos. Incorpora un procesador ATmega328P. Tiene 14 pines digitales de los cuales 6 permiten un control PWM (control por ancho de pulsos) que pueden ser usados para controlar motores, 6 pines de entrada analógica, admite programas de hasta 32KB de tamaño (Flash Memory en las especificaciones de la placa) y su memoria de datos es de 2KB (SRAM en las especificaciones de la placa).

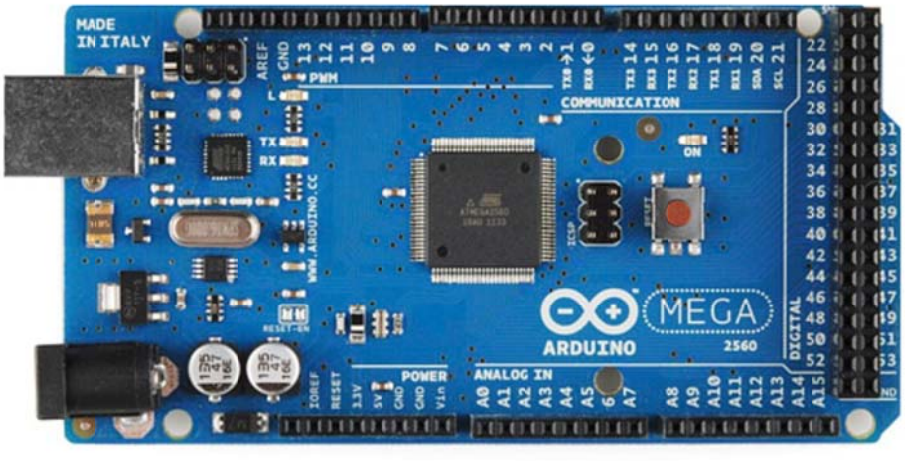


Figura 11. Arduino Mega 2560 R3

Arduino Mega (Figura 11) es una placa más potente que Arduino Uno. Incorpora un procesador ATmega2560. Tiene 54 pines digitales, de los cuales 15 permiten control PWM, 16 pines de entrada analógica, admite programas de hasta 128KB de tamaño y su memoria de datos es de 8KB.

Para la elección del microcontrolador se ha considerado fundamentalmente la capacidad de almacenamiento de datos de ambos microcontroladores. Aunque el robot diseñado es un prototipo y su programa de control va a ser bastante sencillo, el diseño puede evolucionar hasta conseguir un robot lo suficientemente inteligente como para que sea capaz de reconocer el entorno en el que se mueve y tomar decisiones sobre la trayectoria a seguir. Este es un campo en el que se está investigando intensamente en estos momentos en el mundo de la robótica y la gran mayoría de soluciones pasan por elaborar un mapa del entorno en el que se mueve el robot.

El mapa del entorno podría estar constituido por celdas de manera que en cada celda se indicara si el robot puede pasar, debe evitarla, se ha detectado un obstáculo, etc. Si se considera un invernadero de 7m x 30m se obtiene una superficie de 210m^2 y si se consideran celdas de 30cm x 30cm (0,09m^2) el número de celdas a almacenar es de 2333 y considerando un byte de información por celda el mapa del invernadero agotaría la memoria del Arduino Uno (2KB) por lo que se ha optado por el microcontrolador Arduino Mega 2560 R3 de Adafruit.

Se ha elegido la compañía Adafruit como suministrador preferente de componentes debido a sus precios muy competitivos, su amplia gama de productos y la gran documentación que ofrece incluyendo esquemas electrónicos, guías de conectividad, especificaciones técnicas, librerías para Arduino y programas ejemplo para comprobar el funcionamiento de sus productos. La elección de una marca preferente tiene la clara ventaja de que la interconexión y el funcionamiento de todos los componentes necesarios está garantizado.

El consumo de energía del Arduino Mega depende de diversos factores como el número de sensores u otros elementos que tenga que alimentar. Para evaluarlo se utilizará un estudio de consumos para Arduino Mega dependiendo del voltaje de alimentación de la placa y de la frecuencia de reloj del Arduino, que se muestra en la Tabla 1.

	500 KHz	1 MHz	2 MHz	4 MHz	8 MHz	16 MHz
5 V	18,1	18,8	19,6	21,8	25,4	35,2
6 V	26,2	26,9	28	30,7	35,3	44,2
7 V	49	49,8	51,2	54,3	58,3	67,4
8 V	49,7	50,3	51,4	54,8	59,7	71,3
9 V	49,7	50,3	51,4	54,8	59,7	71,3
10 V	49,7	50,3	51,4	54,8	59,7	71,3
11 V	49,7	50,3	51,4	54,8	59,7	71,3
12 V	49,7	50,3	51,4	54,8	59,7	71,3

Tabla 1. Consumo Arduino Mega en diferentes condiciones (Fuente: UPC)

Como se va a alimentar la placa a 5V (un voltaje mayor simplemente provoca una mayor disipación de calor) y como se va a trabajar 16 MHz (ver características en Anexo I), se podría tomar como consumo 35.2mA. Teniendo en cuenta los consumos de los sensores que son de unos 10 mA en total, tal y como se ve en los Apartados 2.3 y 2.4, y para tener un cierto margen se puede estimar un consumo de unos 70mA.

2.2.2. Placa Controladora de Motores

El microcontrolador Arduino permite alimentar eléctricamente otros dispositivos siempre y cuando su consumo no sea muy elevado y además podría controlar motores que se conectaran directamente. Sin embargo los consumos de intensidad eléctrica de los motores y las complejidades de su control hacen que en el diseño del robot se tenga que emplear un componente específico que se dedique al control de los motores.

Se ha elegido la placa Adafruit Motor Shield V2 (Figura 12). Al ser un shield se monta directamente encima de la placa Arduino y la comunicación se hace por el bus I2C, que es uno de los dos buses que tiene Arduino para la comunicación entre diferentes placas. El bus I2C es el más sencillo de ellos y consta de cuatro líneas: dos de alimentación eléctrica (Vcc y GND), una de reloj denominada SCL y una de datos denominada SDA. Al tener una única línea de datos la transmisión es bit a bit, es decir, es un bus serie, y además es síncrono porque todas las transferencias de información se realizan sincronizadas con el flanco de bajada de la señal de reloj. No es objeto de este trabajo entrar en detalles del protocolo de funcionamiento del bus I2C, simplemente enunciar que se utiliza en la comunicación entre Arduino Mega y la placa controladora de motores.

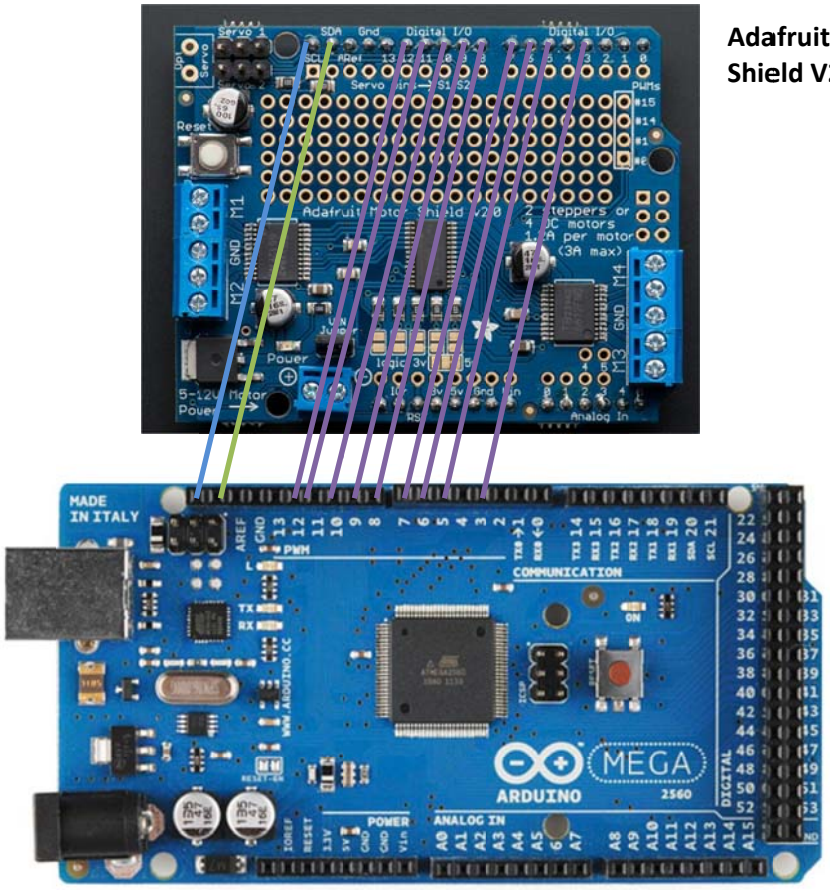


Figura 12. Esquema Conexión Placa Motores-Arduino

Los motores de corriente continua y los motores paso a paso giran en un sentido u otro a una determinada velocidad dependiendo de la intensidad que llega al motor.

El consumo eléctrico depende fundamentalmente de los motores, Tiene cuatro conexiones para motores y cada conexión puede suministrar hasta 1.2 A. Los motores servo funcionan a 5V y el rango de tensiones que puede suministrar para los motores normales va de 4.5V a 13.5V. Se puede ver que las conexiones son directas y el conector del servo encaja directamente en la placa, tal como se ve en la Figura 13.

El control de los motores de corriente continua y/o paso a paso se hace a través de puentes H. Un puente H es un circuito electrónico que permite invertir el sentido de la corriente en un circuito, lo que permite invertir la polaridad de los motores y hacer que giren en un sentido u otro. Funciona a través de interruptores que permiten el paso de corriente en una dirección o en otra. Para robótica, los puentes en H se implementan en circuitos integrados debido a las bajas potencias que se suelen manejar y los interruptores están implementados por medio de transistores.

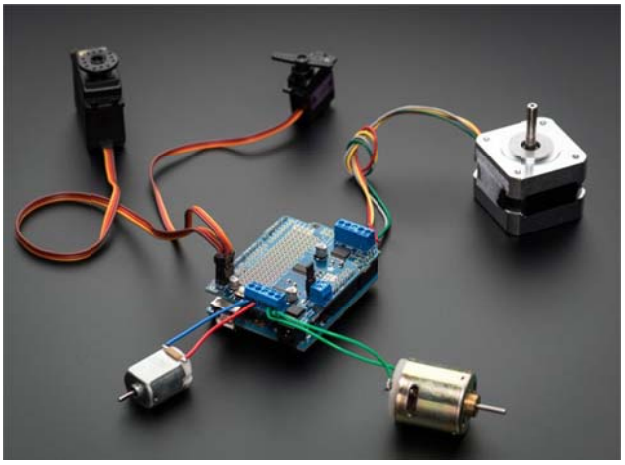


Figura 13. Placa motores conectada a diversos motores

Este controlador de motores incorpora dos chips TB6612 y cada uno de los chips incorpora dos puentes H. Un motor paso a paso requiere dos puentes H para su funcionamiento. Cada chip es capaz de proporcionar hasta 1,2A de intensidad y soportar picos de hasta 3A de intensidad. Este chip también se encarga de la adaptación de la tensión de alimentación de los motores ya que se alimenta a 5V mientras que los motores funcionan normalmente a una tensión nominal entre 6V y 12V. También incluye un circuito protector para el caso de un calentamiento excesivo.

La posición de del motor servo se controla por PWM lo que quiere decir que dada una frecuencia de funcionamiento la anchura del pulso a 5V determina la posición. Por ejemplo si en un pulso de una onda el 5% del tiempo el valor es 5V y el 95% es 0V la posición del servo estará cercana a los 0° pero si el 95% del tiempo el valor es 5V y el 5% del tiempo el valor es 0V la posición del servo estará cercana a los 180°.

La placa también presenta una alimentación separada para los chips TB6612 como se puede ver en la Figura 14.

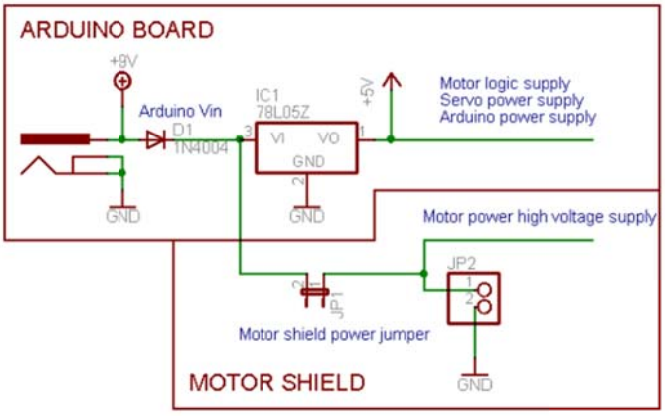


Figura 14. Esquema Eléctrico (Fuente: Adafruit)

Hay una doble alimentación eléctrica de la placa: a través de los pines de alimentación del Arduino para los motores servo y a través de los conectores de alimentación para los chips TB6612. Hay un jumper que permitiría una alimentación única de la placa, lo que podría ser útil si se elige un motor servo de bastante potencia. Lo que no se debe hacer nunca es colocar el jumper y alimentar toda la placa desde el Arduino sin alimentación externa ya que eso significaría alimentar los motores directamente desde el Arduino y eso podría dañar el microcontrolador.

2.2.3. Sistema Comunicación Inalámbrica

El robot ha de ser capaz de comunicarse con estación central para que éste le mande las órdenes que ha de realizar, y para que el robot sea capaz de mandar los registros de variables tomadas a la estación central.

Para la comunicación se han elegido dispositivos Xbee (Figura 15), que funcionan como si se tratara de una red Wifi. Se ha buscado un shield que el fabricante garantice que es compatible con Arduino Mega. Este es solamente el shield para su conexión; para que la comunicación se pueda realizar se necesitan dos dispositivos Xbee (uno para el robot y otro para un dispositivo receptor de los datos) y un adaptador Xbee-USB para la conexión del receptor. Se ha elegido una conexión USB para el receptor por ser el tipo de conexión estándar más empleado en todo tipo de dispositivos como ordenadores, tablets e incluso teléfonos móviles.

El consumo eléctrico máximo de un dispositivo Xbee es de 33 mA aunque un estudio de la UPC eleva la cifra a 45 mA. Se ha tomado esta cifra por seguridad.

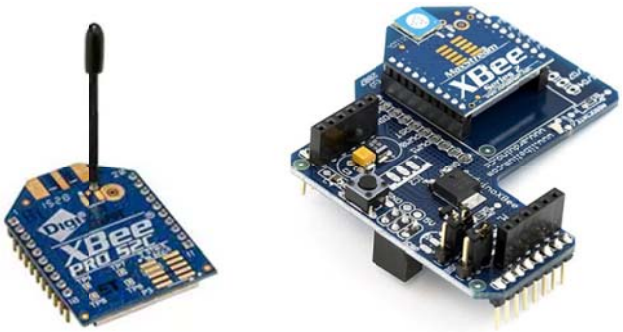


Figura 15. Dispositivo xBee S2 (izda.) y Shield para Xbee (dcha.)

2.3. Sensores de medición

Como se ha empleado Arduino como entorno de programación, los sensores seleccionados para medir las variables deseadas (temperatura, porcentaje de humedad, concentración de dióxido de carbono y luminosidad) han de ser compatibles con dicho entorno y deberían ser programados de una forma sencilla.

2.3.1. Medida de Temperatura y %Humedad

Se ha seleccionado el sensor DHT22 del fabricante Adafruit, que es capaz de medir temperatura y porcentaje de humedad con el mismo dispositivo. Se ha optado por un modelo encapsulado en plástico, funciona a 5V con un pequeño consumo de 2.5mA y tiene los siguientes rangos de medición:

- Temperatura: -40°C a 80°C.
- Humedad: 0% a 100%.

A la hora de realizar la conexión hay que colocar una resistencia pull up tal como indica el fabricante para que la lectura de los valores sea más precisa. Esto mismo se puede hacer por software y se puede evitar la resistencia. El pin de salida de datos se conecta a un pin digital del Arduino, y los pines de 5V y tierra a los terminales de alimentación y tierra correspondientes del resto del circuito (Figura 16).

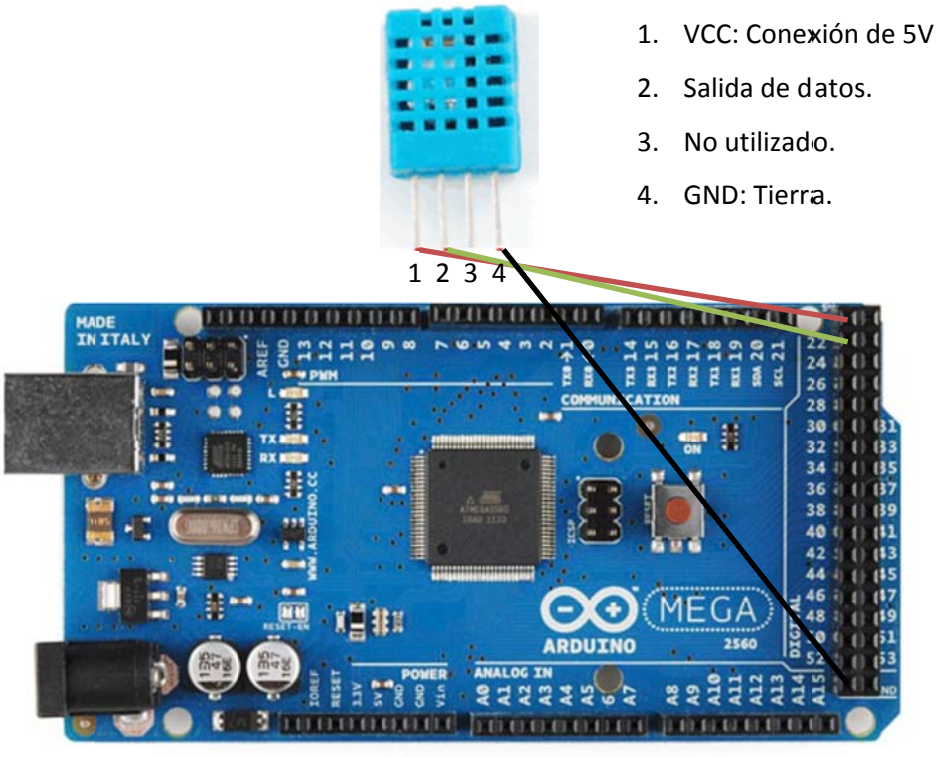


Figura 16. Conexión DHT22-Arduino

El funcionamiento del sensor necesita un programa y una librería específica que se puede encontrar en la web del fabricante.

2.3.2. Medida de Luminosidad

Se selecciona el sensor TSL2561, también compatible con Arduino, del fabricante Adafruit. Utiliza el bus I2C para su comunicación con el microcontrolador Arduino. Este dispositivo tiene un rango de medición de 0.1 a 40000 luxes, y su consumo eléctrico es de 0.5mA.

Para conectar este sensor, se toman los pines de 5V y tierra a los terminales correspondientes en el Arduino, y los dos pines de SDA y SCL con sus correspondientes en Arduino Mega, que son los pines 20 y 21 (Figura 17).

1. Vin: Alimentación eléctrica de 5V.
2. GND: Tierr.
3. Los conectores 3Vo, Addr y Int no se utilizan. Sirven para diferenciar sensores de luminosidad si se conectan varios.
4. SDA (Data): pin de datos para I2C.
5. SCL (Clock): pin para el reloj en I2C.

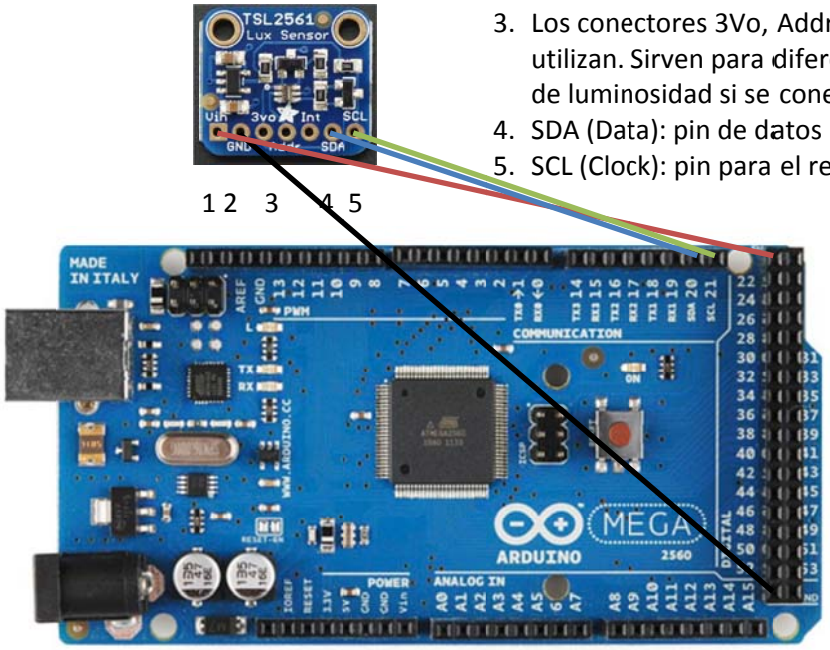
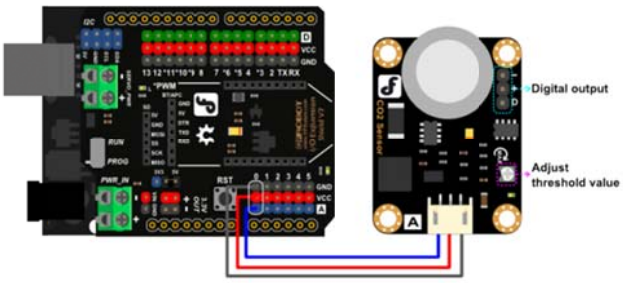


Figura 17. Conexión TSL2561-Arduino

Para utilizar el sensor se necesita la librería de sensores de Adafruit y la librería propia del sensor que se puede encontrar también en la web del fabricante. Por medio de un programa el sensor mide la luminosidad en luxes.

2.3.3. Medida de Concentración de CO2

Se ha seleccionado el sensor SKU: SEN 0159 porque desafortunadamente no hay ningún sensor de Adafruit. El fabricante ofrece muy poca documentación sobre este sensor y la información que se ha conseguido es del fabricante Sandbox que ofrece un producto similar. Su rango de medición es de 400 a 10.000 ppm. Su voltaje de alimentación es a 5V, y la corriente de operación de 2 mA. En la Figura 18 se puede ver un esquema del sensor.



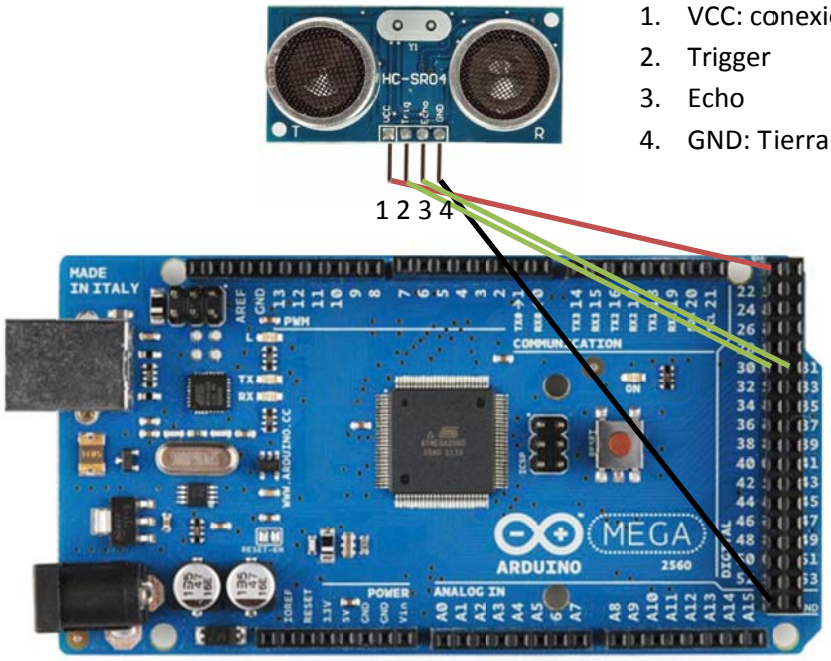
1. Negro: Tierra (GND).
2. Azul: Digital Output. A conectar a un pin digital del Arduino.
3. Rojo: Alimentación a 5V.

Figura 18. Sensor SKU:SEN 0159

Para conectar este sensor se conectan los pines de alimentación y tierra del sensor a los correspondientes terminales del Arduino, y el pin digital a uno cualquiera del Arduino. El fabricante facilita un código ejemplo de utilización del sensor sin necesidad de recurrir a librerías específicas como en los otros sensores mencionados, y su implementación es inmediata.

2.4. Detección de obstáculos

Hay diversos sensores que permiten medir la distancia a un obstáculo. Se suelen basar en dos tecnologías diferentes: láser y ultrasonidos. Los sensores láser tienen una mayor precisión y menor rango de distancias. Los sensores por ultrasonidos dan una precisión suficiente para esta aplicación por lo que se ha optado por esta tecnología.



1. VCC: conexión a 5V
2. Trigger
3. Echo
4. GND: Tierra

Figura 19. Conexión HC-SR04-Arduino

El sensor elegido es el HC-SR04, que tiene un rango de precisión que va de unos pocos centímetros hasta unos 4 metros. El sensor tiene cuatro conectores: tensión y tierra como todos los sensores y dos conectores denominados trigger y echo. Su conectividad a Arduino se ilustra en la Figura 19. Cuando se activa el conector trigger se envía un pulso de untrasonidos y al rebotar en un obstáculo el valor del pin echo indica el tiempo transcurrido. Haciendo un sencillo cálculo con la velocidad del sonido se puede saber la distancia al obstáculo. La utilización de bibliotecas específicas facilita la programación.

Este sensor va instalado en la parte delantera del vehículo acoplado al sistema de dirección, debido a que por su finalidad ha de estar en la parte delantera del robot.

2.5. Alimentación Eléctrica

Teniendo en cuenta los consumos máximos de los diferentes componentes:

- Motor paso a paso: 840mA.
- Motor servo: 900mA.
- Arduino Mega (incluido consumo de sensores): 70mA.
- Xbee: 45mA.

Se obtiene un total de 1855mA, y considerando las pérdidas que se podrían ocasionar en el resto de circuitería se estima un consumo máximo de 2A. Hay que destacar que éstos son *consumos máximos con los motores a plena potencia*.

El consumo calculado va a servir para seleccionar el tipo de alimentación a utilizar. Las pilas AA de mayor duración son las alcalinas de 3000mAH. Se necesitarían cuatro pilas de 1,5V en serie para conseguir un voltaje de 6V, adecuado para alimentar el robot. Para recargar las pilas se necesitaría extraerlas del robot y colocarlas en un cargador externo.

La tendencia en proyectos de este tipo es incorporar una batería potente que se pueda recargar conectando el robot a una toma USB. La batería seleccionada es el pack compacto de baterías de litio de 3,7V y 6600mAh, es capaz de proporcionar una intensidad continuada de descarga máxima de 3.3A. Es una batería tipo lipo (Polímero de Litio) y permitiría el funcionamiento del robot durante unas tres horas.

Para conectar la batería al robot será necesaria una placa de alimentación que suministre el voltaje requerido por los diferentes componentes electrónicos. Se ha elegido la placa Adafruit Powerboost 1000 C. La intensidad nominal soportada es de 2A con picos de hasta 2.5A. Es capaz de proporcionar una intensidad máxima de carga de 1Am.

Las dos principales ventajas de emplear un sistema de baterías son las siguientes:

- Se carga conectando un cable USB al robot, no hay necesidad de desmontar nada.
- La placa permite avisar si el nivel de la batería es bajo.

2.6. Modelado del Robot y construcción del prototipo

Una característica interesante de los microcontroladores Arduino es la posibilidad de apilar placas una encima de otra para realizar las conexiones, lo que permite simplificar enormemente el cableado y de esta manera proponer un diseño de vehículo más compacto. Se ha utilizado esta propiedad para apilar la placa controladora de motores y el adaptador para XBee encima de la placa Arduino Mega.

Con el fin de facilitar las conexiones eléctricas, se ha hecho uso de una placa de conexiones estándar para conectar la alimentación y tierra tanto de sensores como del microcontrolador, facilitando el acceso de todos los componentes a dichos terminales. El robot modelado se muestra en la Figura 20.

Electrónica de control:
- Arduino Mega,
- Motor Shield V2
- Xbee Shield

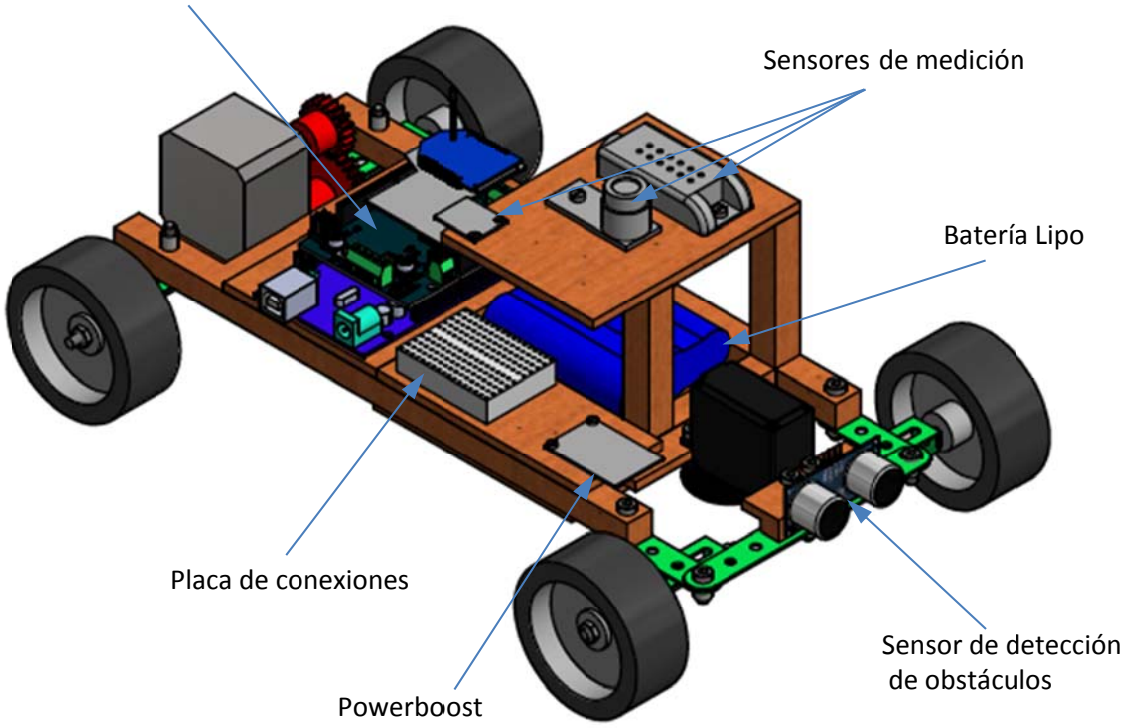
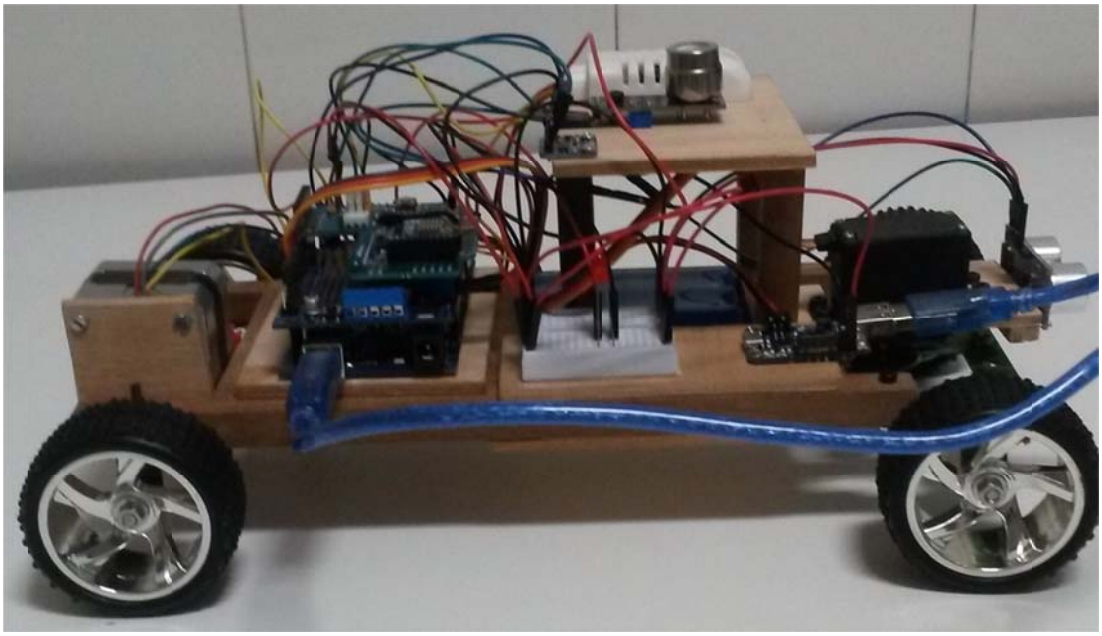
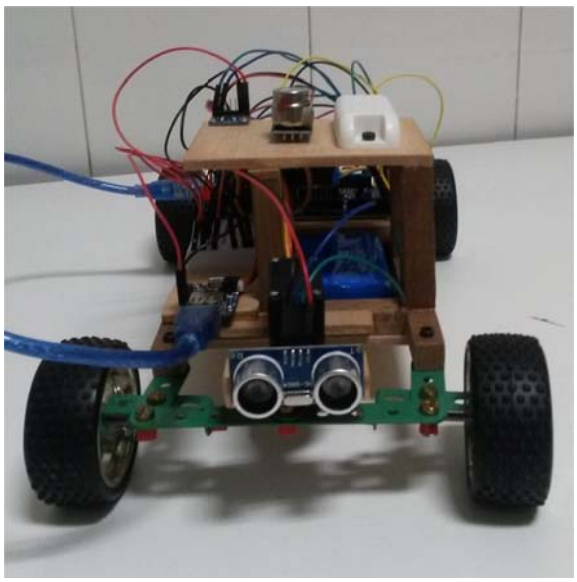


Figura 20. Robot modelado

El modelo propuesto ha sido construido y montado. El prototipo final se muestra en la Figura 21. Utilizando este prototipo y desarrollando la programación que se expone en el siguiente capítulo, se han realizado las primeras pruebas de funcionamiento.





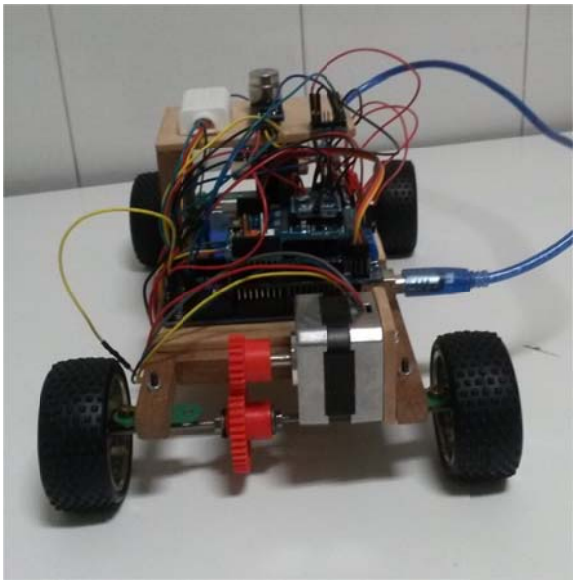


Figura 21. Prototipo robot invernadero

3. CONTROL DEL ROBOT

El robot ha de recibir como entrada unas determinadas instrucciones relacionadas con el recorrido a realizar por el invernadero y ha de enviar como salida un conjunto de datos relacionados con las medidas tomadas a lo largo del recorrido, incluyendo la opción de que en su camino encuentre un obstáculo que le impida completar el recorrido.

Las premisas anteriores implican que en realidad no se debe hablar de un único programa sino de dos programas, uno que se ejecutará en el microcontrolador Arduino y otro que se ejecutará en un dispositivo externo de control, que puede ser un ordenador convencional, una Tablet o incluso un Smartphone. A este dispositivo se le llamará *dispositivo de control*. Lo primero que se debe analizar es cómo se van a comunicar esos programas.

La comunicación entre el dispositivo de control y el robot se realiza utilizando dos dispositivos xBee (Figura 15), uno instalado en un shield en el robot y otro en un adaptador USB, por tanto hay un requisito previo que tienen que cumplir los dispositivos de control: su sistema operativo debe ser capaz de comunicarse con el dispositivo xBee conectado por USB. Detalles de cómo se realiza la comunicación entre dichos dispositivos se encuentra en el Anexo IV.

Se ha probado la conectividad de xBee con los sistemas operativos Windows más recientes. En Windows 7 se necesita instalar unos controladores especiales y en Windows 10 el dispositivo se reconoce directamente. No se ha probado ni con dispositivos Linux ni con sistemas operativos Android o IOS, propios de Tablets y Smartphones.

El Shield xBee de Arduino utiliza el puerto serie del microcontrolador y Windows asocia el adaptador USB xBee a un puerto serie, eso quiere decir que ambos dispositivos se comunican *como si hubiera un cable serie conectándolos*, es decir, utilizan el protocolo serie que transmite la información entre dos dispositivos bit a bit utilizando dos señales de datos denominadas RX y TX. Cualquier sistema de desarrollo de programas tiene incorporado este protocolo.

No obstante, todas las librerías que trabajan con el protocolo serie trabajan a nivel de byte, no de bit, de manera que la unidad de información más pequeña que se puede distinguir es un carácter. Es responsabilidad del controlador del puerto serie (que puede estar programado en el hardware de la máquina) recibir los bits, convertirlos en bytes y almacenarlos en una memoria intermedia a la espera de que el programa los trate, de manera que no se pierda información. A la hora de transmitir información, ésta se graba en esa memoria intermedia y el controlador la va transmitiendo bit a bit.

La comprensión de cómo se comunican el Arduino y el dispositivo de control es importante para definir la manera de enviar y recibir la información, que es lo único que van a tener en común el programa del microcontrolador y el programa de control. Se ha decidido implementar un sencillo lenguaje de comandos consistente en un carácter y un número entero, de manera que el robot reaccione a dichos comandos y envíe unos determinados comandos al programa de control cuando necesite transmitir información.

Esta aproximación tiene varias ventajas muy importantes:

- Permite diseñar de manera independiente el programa de control y el programa del microcontrolador. El único requisito que ambos programas tienen que cumplir es que interpreten correctamente los comandos. Incluso se podría controlar al robot enviando manualmente los comandos adecuados utilizando un programa de comunicación serie como Putty o el monitor serie del Arduino Ide.
- Permite controlar cualquier robot con cualquier programa de control ejecutándose en cualquier dispositivo siempre y cuando ambos programas "entiendan" los comandos y ambos dispositivos sean capaces de comunicarse.
- El sistema puede crecer en funcionalidad ya que basta con añadir nuevos comandos para conseguir nuevas funcionalidades siempre y cuando se implementen las nuevas funcionalidades tanto en el programa de control como en el programa del microcontrolador.

En general los comandos relacionados con los motores van a requerir algún parámetro como distancia a recorrer o velocidad mientras que el resto de los comandos no van a necesitar ese parámetro. La lista de comandos principales se indica a continuación y la lista de los admitidos por el sistema se detalla en el Anexo V.

Ordenes al robot:

- sn: Fijar consola, si n=1 modo consola, si n=0 modo programa de control:
- dn: Girar servo a la derecha, n es el número de grados a girar.
- in: Girar servo a la izquierda, n es el número de grados a girar.
- c0: Servo a posición central para avanzar en línea recta.
- xn: Fijar la distancia de seguridad, n es la distancia en centímetros.
- vn: Fija la velocidad del motor paso a paso en revoluciones por minuto (valor de n).
- ln: Distancia a recorrer en centímetros.
- g0: Orden para que el robot se mueva la distancia deseada.
- p0: Orden para que se pare el robot.

Petición de información:

- A0: Actualizar los valores de los sensores. No devuelve ningún valor.
- T0: Pedir la temperatura. El robot devuelve un entero que es la temperatura en grados centígrados.
- H0: Pedir la humedad. El robot devuelve un entero que es el porcentaje de humedad.
- L0: Pedir la luminosidad. El robot devuelve un entero que es la luminosidad en luxes.
- C0: Pedir CO_2. El robot devuelve un entero que es el porcentaje de CO_2 en partes por millón.
- D0: Pedir distancia al obstáculo. El robot devuelve un entero que es la distancia al obstáculo en centímetros.
- R0: Pedir distancia a destino. El robot devuelve un entero que es la distancia en centímetros que le separa de su objetivo y calculada a partir de los pasos restantes.

3.1. Programación del microcontrolador

El programa del microcontrolador se encarga de manejar todos los dispositivos y controlar el robot. Debe ser eficiente y no contener complicadas rutinas que ralenticen su funcionamiento.

El control de los dispositivos del robot se realiza con Arduino Ide, que es la utilidad estándar proporcionado por Arduino para el desarrollo de programas para microcontroladores Arduino. La instalación del programa se muestra en el Anexo III. El lenguaje de programación soportado por Arduino Ide es C/C++.

Para hacer el programa más modular, se ha separado el código en diferentes ficheros que se explican a continuación. Una explicación más detallada de cada fichero puede verse en el anexo V.

3.1.1) Comunicaciones.h

Contiene toda la funcionalidad que permite al Arduino recibir órdenes y transmitir información utilizando xBee. Todas las funciones se han incluido en una clase de C++ llamada TComunicaciones.

La principal función de esta clase es comprobar si ha llegado algún comando a través del xBee y si es así extraer la información (carácter y número) y almacenarlo en una variable para que la función encargada de interpretar el comando lo ejecute.

3.1.2) Sensores.h

Contiene toda la funcionalidad relacionada con la toma de datos de los sensores. Todas las funciones se han incluido en una clase llamada TSensores.

La clase contiene unas variables que almacenan los últimos datos leídos de temperatura, porcentaje de humedad, luminosidad y concentración de CO_2. Hay una función llamada *actualizardatos* que se encarga de leer la información de los sensores y almacenarla en dichas variables. Los valores almacenados en dichas variables son los que el robot reporta al programa de control, de manera que cuando se quiera hacer una medición hay que llamar a dicha función.

Destacar que el programa del microcontrolador no va a almacenar datos históricos, solo los últimos datos leídos de los sensores. El programa de control debería ser el encargado de realizar dicha función si fuese necesario.

3.1.3) Motores.h

Contiene toda la funcionalidad relacionada con el funcionamiento de los motores. Tiene funciones que permiten orientar el servo en una determinada dirección y ordenar al motor paso a paso que avance una determinada distancia. Se ha implementado en una clase llamada TMotores.

Debido a la modularidad del diseño el programa desarrollado se podría adaptar fácilmente a otro robot sin más que cambiar la funcionalidad de este módulo.

Esta clase contiene una variable que indica si el robot se está moviendo o no. Esta variable se utiliza en el programa principal para activar o no la detección de obstáculos.

3.1.4) Programarobot.ino

Contiene el programa principal del Arduino. Todo programa Arduino debe implementar dos funciones donde hay que encapsular toda la funcionalidad con la que se dota al microcontrolador.

- Setup: se ejecuta una única vez al encender el microcontrolador y ahí se deben programar todas las inicializaciones necesarias de los dispositivos (puerto serie, sensores y motores).
- Loop: es un bucle infinito y aquí se debe programar las tareas que estará ejecutando el Arduino hasta que se apague el robot.

La función “Loop” se ha implementado de manera muy sencilla y se encarga de:

- Comprobar si hay datos en el puerto serie y en caso afirmativo hace lo siguiente:
    - Ordena al módulo de comunicaciones que obtenga el comando.
    - Pasa el comando a una función que hace el papel de *intérprete de comandos* y que es la encargada de ejecutar la orden recibida.
- Si el robot se está moviendo comprueba:
    - Si hay un obstáculo a una distancia menor que la distancia de seguridad detiene el robot.
    - Comprueba si el robot ha llegado a su destino y en ese caso también detiene el robot.

Si el programa del microcontrolador detiene el robot por cualquiera de las condiciones vistas se envía un comando al programa de control indicando que el robot se ha detenido y la causa de la detención.

La funcionalidad del sensor de ultrasonidos se ha incluido en el módulo de sensores de manera que se trata como otro sensor más. Hay una función llamada *distancia* que actualiza una variable que almacena los centímetros a los que se encuentra un obstáculo.

En una versión más avanzada del programa en la que se pudiera incluir un mapa del entorno, el programa del microcontrolador se encargaría de marcar las celdas correspondientes como obstáculos e incluso podría calcular si se puede esquivar el obstáculo y la ruta a seguir para esquivarlo y llegar al destino. En esta primera versión del programa de control se ha decidido no incorporar esa funcionalidad avanzada y delegar en el programa de control la decisión de qué hacer al encontrar un obstáculo.

El intérprete de comandos es la función que coordina todas las tareas del robot, por ejemplo, si recibe la orden de actualizar los valores de los sensores llama a la función correspondiente del módulo de sensores y si se le pide un valor de un sensor lo solicita al módulo de sensores y lo transmite al programa de control usando el módulo de comunicaciones.

Se ha definido una variable que permite distinguir si la comunicación se realiza con un programa de control o con un programa de comunicaciones por puerto serie como el monitor serie del Arduino Ide. El objetivo es variar los mensajes que se envían desde el robot. Cuando el destinatario es un programa se envía solamente el dato solicitado mientras que si el destinatario es un programa de comunicaciones por puerto serie se añaden explicaciones textuales ya que se supone que alguien está enviando los comandos manualmente. A través de un comando que permite variar dinámicamente el modo de comunicación.

3.2. Programa de Control

Este programa se ejecuta en el equipo que envía las órdenes al robot. El dispositivo de control puede ser un ordenador con Windows o Linux o un dispositivo móvil como una Tablet o un teléfono móvil.

Para el desarrollo del programa de control se ha elegido un entorno de programación llamado Qt por los siguientes motivos: existen versiones para diferentes sistemas operativos como Windows, Linux y Mac, es una herramienta libre y permite programar en C/C++.

Se ha desarrollado un programa que permite enviar el robot de un punto a otro y leer los datos de los sensores, con una única ventana principal tal y como se muestra en la Figura 22.

La ventana principal se encuentra dividida en tres secciones: Comunicaciones, Movimi ento, Sensores y el botón de Salir del programa en la parte inferior.

- La sección de Comunicaciones sirve para seleccionar el puerto serie al que está conectado el XBee y establecer la comunicación con el robot.
- La sección de Movimiento permite controlar la dirección, la distancia a recorrer, la velocidad y la distancia a la que un obstáculo hará que el robot pare. Un control giratorio (como si fuera un volante) que está asociado a un control numérico, permite indicar los grados de giro en la dirección del robot. El rango de giro va de -30 a +30 de manera que valores negativos indican giro a la izquierda y valores positivos giro a la derecha. Otro control numérico permite especificar la distancia a recorrer en centímetros, cuyos valores van desde -5000 a 5000, indicando los valores negativos que debe retroceder y los

positivos que debe avanzar. Otro control numérico permite fijar la velocidad en rpm y va de 0 a 1000. Finalmente, se puede cambiar la distancia de seguridad por medio de otro control numérico y los valores permitidos van de 1 a 100 (en cm). El botón "Mover" permite iniciar el movimiento del robot y cuando se está moviendo su texto cambia a “Parar” para indicar que si se pulsa se detiene la marcha del robot.

- La sección de sensores simplemente presenta un cuadro de texto donde se presentarán las medidas realizadas de las variables ambientales.

Figura 22. Interfaz de usuario

Si el robot detecta un obstáculo informa del evento e informa de la distancia a la que se encuentra el obstáculo. Un ejemplo final de medición de los sensores se ilustra en la Figura 23. Además, los detalles de la implementación del programa de control se muestran en el Anexo VI para más información.

El programa de control se ha probado en el prototipo utilizando como entorno un espacio físico de dimensiones reducidas. Se han probado las funciones básicas de desplazamiento y giro del vehículo, la detección de obstáculos y la toma de datos a través de los sensores. Se ha comprobado que el robot puede desarrollar un recorrido sencillo de manera autónoma y enviar datos de variables a la estación central.

El programa de control propuesto puede evolucionar hasta convertirse en una compleja aplicación. Podría alertar en el caso de que algunos valores medidos estuvieran fuera de control, podría almacenar los datos recibidos en una base de datos para estudios posteriores, podría representar gráficamente por pantalla los obstáculos detectados y proponer rutas alternativas, etc.

Figura 23. Ejemplo de toma de datos

4. CONCLUSIONES

Se ha conseguido diseñar un robot capaz de moverse de manera autónoma por un invernadero, y tomar variables ambientales del entorno para su control. Además, en caso de detectar un obstáculo, el robot se detiene y manda una señal de alarma a una estación central. La comunicación con el robot se realiza con dos dispositivos de comunicación inalámbrica Xbee, uno conectado a la estación central, y el otro instalado sobre el propio robot.

Los principales parámetros a tener en cuenta a la hora de seleccionar los componentes necesarios han sido el peso, el voltaje de funcionamiento y la corriente consumida. El peso es importante, ya que al tener un motor paso a paso que no es capaz de dar un par elevado, se ha buscado la mayor ligereza posible en todos los componentes, sobre todo al diseñar el bastidor. El voltaje de funcionamiento es vital para lograr que todos los dispositivos electrónicos funcionen de manera correcta. Por último, la intensidad que consumen los componentes electrónicos (principalmente los dos motores) ha servido para tener una idea de la batería a seleccionar para que se pudiera garantizar una mínima autonomía aceptable, y la placa Powerboost que se conecta a ella, que tiene que ser capaz de suministrar la corriente solicitada.

Una vez seleccionados los componentes, se ha procedido al modelado y montaje real del robot, buscado sencillez en la versión final del prototipo tanto en el sistema de tracción como en el de dirección.

Una de las partes principales del proyecto ha sido la implementación en lenguaje C++ el movimiento del robot y la toma de medidas, así como la creación de un programa de control que muestre en una ventana sencilla para el usuario la manera de comunicarse con el robot, y obtener información de él.

Para programar el movimiento del robot y la medida de los sensores, se ha empleado Arduino IDE. El programa realizado se ha ayudado de las librerías específicas de cada componente. Al tener un motor un motor paso a paso, se puede tener un control de la distancia recorrida. El principal problema encontrado ha sido que el par capaz de suministrar el motor paso a paso no es suficiente a velocidades altas como para mover el robot adecuadamente. Por tanto, las pruebas realizadas se han realizado a una velocidad del motor entre 75-80 rpm.

El programa de control se ha realizado con el programa Qt. La ventana del programa de control ha sido hecha con la idea de que sea lo más sencilla posible, pensando en que el usuario final del producto lo pueda manejar con facilidad. Por ello, una vez establecida la conexión con el robot, el usuario solamente introduce directamente los parámetro de giro, distancia a recorrer y velocidad del motor. El programa realizado dispone de un ejecutable que es capaz de usarse en cualquier dispositivo sin la necesidad de instalar ningún programa, siendo una ventaja para el usuario final.

BIBLIOGRAFÍA

- Invernadero robotizado: http://www.diariodeciencias.com.ar/agro-robotica-invernadero-robotizado-la-teoria-del-efecto-invernadero/
- Robot UPM: http://www.upm.es/UPM/SalaPrensa/Noticias?id=4b3e6ca20aa08510VgnVCM10000009c7648a____&fmt=detail&prefmt=articulo
- Robot Cámara: http://intainforma.inta.gov.ar/?p=11085
- Arquitectura de robots: https://www.iit.comillas.edu/~alvaro/teaching/Clases/Robots/teoria/arquitecturas%20de%20robots.pdf
- Diseño de robots móviles: http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/robotica/movil.htm
- Estudio consumo Arduino Universidad Politécnica de Cataluña: https://upcommons.upc.edu/bitstream/handle/2099.1/24745/memoria.pdf?sequence=4
- Documentación Placa controladora de motores: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino
- Montaje placa Powerboost: https://learn.adafruit.com/adafruit-powerboost-1000c-load-share-usb-charge-boost/assembly
- Librería General Sensores Adafruit: https://github.com/adafruit/Adafruit_Sensor
- Resistencia Pull-Up mediante software: https://soloarduino.blogspot.com.es/2013/07/resistencias-pull-up-pull-down.html
- Sensor DHT22 Librería: https://github.com/adafruit/DHT-sensor-library
- Sensor TSL2561 Librería: https://learn.adafruit.com/tsl2561/use
- Librería Motorshield: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software
- Librería Accel Stepper: https://github.com/adafruit/AccelStepper
- Sensor HC-SR04 Librería: https://github.com/JRodrigoTech/Ultrasonic-HC-SR04
- Código Ejemplo CO2: http://sandboxelectronics.com/?p=147
- Curva Sainsmart MG811: http://eph.ccs.miami.edu/precise/GasSensorSpecs/CO2.pdf
- Manual de Qt: http://choccac.com/?wpfb_dl=1
- Libros programación:
  - *Introducción a Arduino*. Massimo Banzi. Anaya Multimedia, 2010.
  - *C++ a su alcance, un enfoque orientado a objetos.* Luis Joyanes Aguilar. McGraw Hill, 1994
  - *Programación en C++ para ingenieros.* Fatos Xhafa. S.A. Ediciones Paraninfo, 2006

Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de un robot para registro de variables ambientales en invernadero

Autor

Mikel Brun Martínez

ANEXOS

Grado en Ingeniería de Tecnologías Industriales

Mayo de 2017

ÍNDICE

ANEXO I

CARÁCTERÍSTICAS PRINCIPALES DE COMPONENTES

Arduino Mega 2560 R3

Dimensiones	101,98x53,63x15,29 mm
Peso	34,9 gr
Microcontrolador	ATmega2560
Voltaje de operación	5 V
Voltaje de entrada recomendado	7-12 V
Límites voltaje de entrada	6-20 V
Pines I/O digitales	54 (14 con salida PWM)
Pines analógicos de entrada	16
Corriente DC por pin I/O	40 mA
Corriente DC para el pin 3.3V	50 mA
Memoria flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Velocidad del reloj	16 MHz

Motor paso a paso Nema 17

Dimensiones	Cuerpo 42x42x42 mm
Número de pasos por vuelta	200
Ángulo de paso	1,8°
Precisión del paso	±5 % (paso entero, sin carga)
Rango de temperaturas	-20-50°C
Máxima fuerza radial	28 N
Máxima fuerza axial	10 N
Tensión nominal	12 V
Corriente máxima por fase	0,35 A
Resistencia por fase	34 Ω
Inductancia por fase	33 mH
Par de retención	1,6 kg·cm
Inercia del rotor	35 g·cm^2
Peso	0,22 Kg
Par de detención	120 g·cm
Longitud del eje	34 mm

Motor servo Towerpro SG-5010

Dimensiones	40x20x38 mm
Voltaje de operación	4.8-6 V
Peso	39 gr
Par a 5 V	5,5 kg·cm
Velocidad media a 5V	0,18 s/60°
Número de dientes de salida	25

Motor shield V2.3

Dimensiones	70x55x10 mm
Voltaje de alimentación	15 V (máximo)
Voltaje de entrada	0,2-6 V
Voltaje de salida	15 V
Corriente de salida	3.2 A (pico)
Temperatura de operación	-20-85 °C
Salida de baja en resistor	0,5 Ω
Energía disipada	1,36 W

XBee S2

Dimensiones	2,761x2,438 cm
Frecuencia	2,4 GHz
Alcance	4000 ft (1220 m)
Frecuencia de datos	250 kbps
Sensibilidad	-100 dBm
Voltaje operativo de suministro	2.1-3.6 V
Corriente de suministro en transmisión	33 mA
Corriente de suministro en recepción	28 mA
Energía de salida	3.1 mW
Temperatura operativa máxima	+ 85 °C
Temperatura operativa mínima	-40 °C
Tipo de conector de la antena	Integrado

Sensor DHT 22 - T y %humedad

Dimensiones	27x58,75x13,30 mm
Voltaje de operación	3-5 V
Corriente máxima	2.5 mA
Rango de medición de Humedad	0-100 % (precisión del 2-5%)
Rango de medición de Temperatura	-40-80 °C (precisión de ± 0,5°C)
Frecuencia de medición	0,5 Hz (una vez cada 2 segundos)
Número de cables	3 (23 cm de longitud)

Sensor TSL 2561 - Luminosidad

Dimensiones	20x16,5 mm
Corriente de suministro	0.5 mA
Temperatura de operación	-30-80 °C
Rango de medición	0.1-40.000 lux
Interfaz	I2C
Voltaje de operación	3-5 V
Frecuencia de medición	0,6 Hz

Sensor SKU: SEN 0159 - [CO_2]

Dimensiones	32x42 mm
Potencia de entrada	3.3-6 V (recomendado 5 V)
Corriente de operación	2 mA
Voltaje de operación	0-5 V
Interfaz	Analógica
Energía de salida	1200 mW
Rango de medición de [CO_2]	400-10.000 ppm
Frecuencia de medición	0,7 Hz

Sensor HC-SR04

Dimensiones	45x20 mm
Voltaje de operación	5 V
Corriente estática	<2mA
Ángulo del sensor	15 °
Distancia de detección	2-450 cm
Precisión	3 mm
Peso	10 gr

Placa Powerboost 1000C

Dimensiones	36x26x2 mm
Voltaje de salida	5 V
Frecuencia de operación	700 KHz
Corriente suministrada	2 A
Corriente máxima de pico	2.5 A
Intensidad máxima de carga	1 A

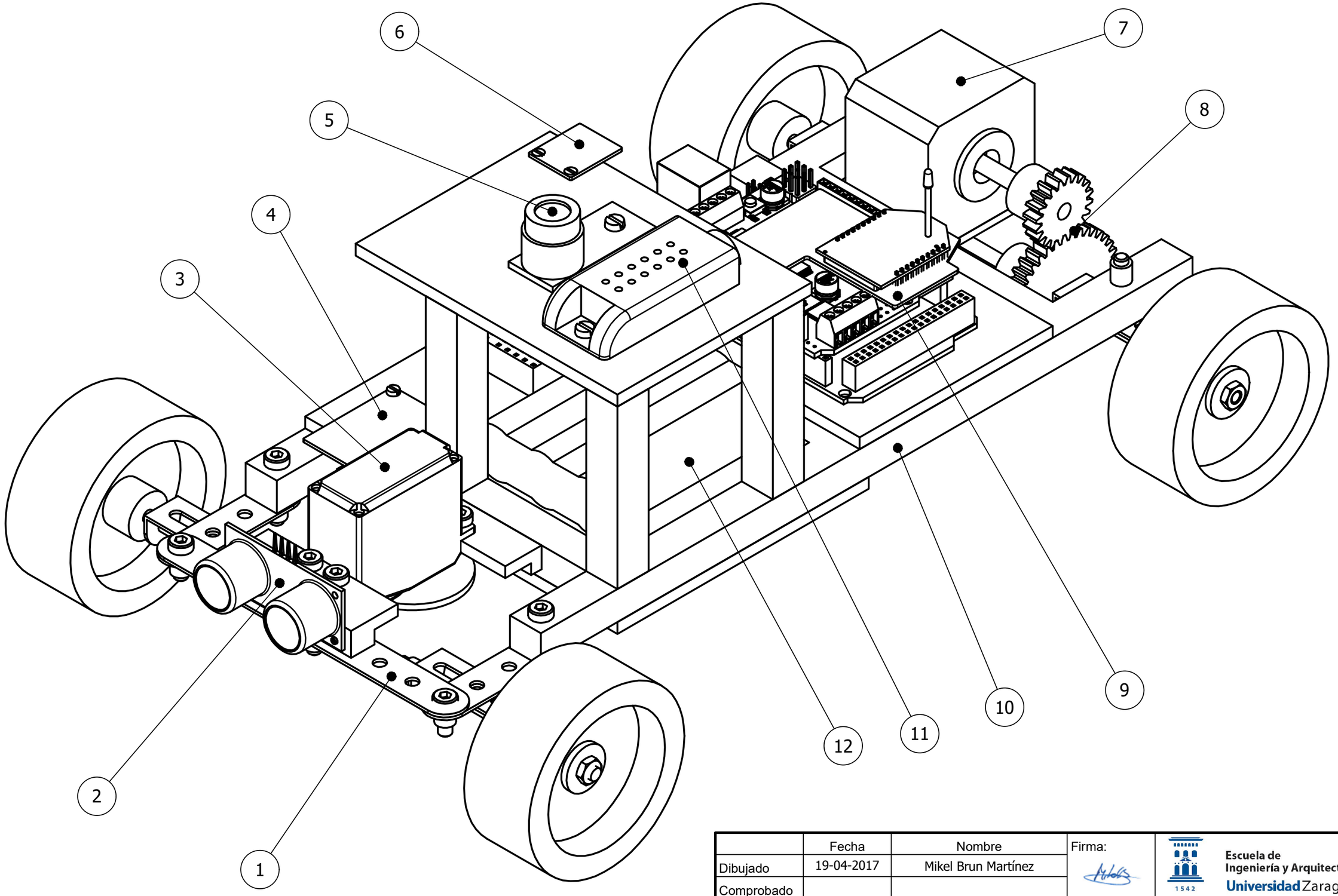
Batería de Litio

Dimensiones	69x54x18 mm
Peso	155 gr
Corriente máxima de carga	1.65 A
Corriente máxima de descarga	3.3 A
Capacidad	6600 mAh (3 celdas en paralelo)
Voltaje de opercación	3.7 V

ANEXO II

MODELADO DE COMPONENTES

1. Conjunto Robot
2. Lista de piezas
3. Vistas del Conjunto Robot



	Fecha	Nombre	Firma:	Escuela de Ingeniería y Arquitectura Universidad Zaragoza
Dibujado	19-04-2017	Mikel Brun Martínez		
Comprobado				
Escala 1:1	Título: Conjunto Robot			Nº Alumno
				Curso
				Plano Nº

A3

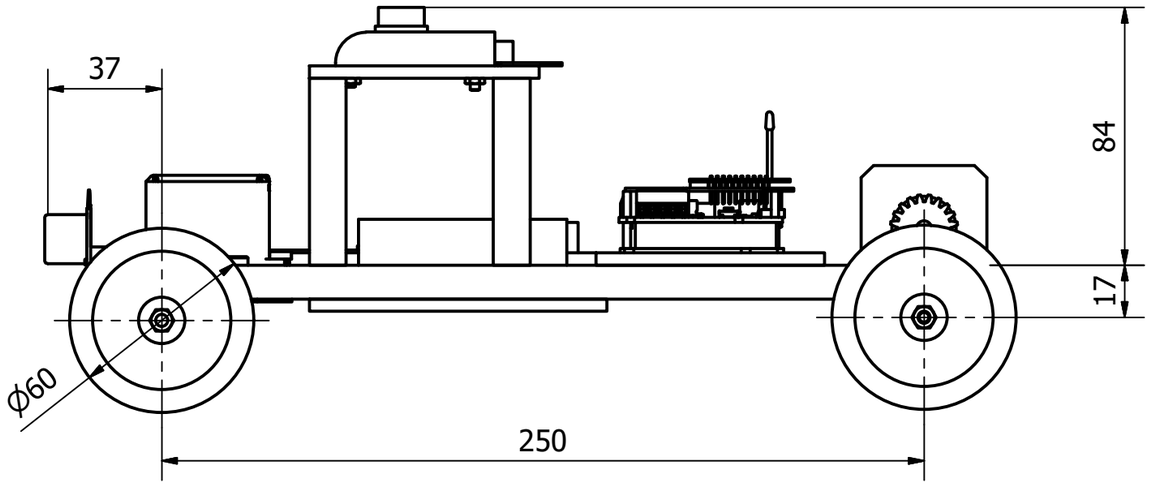
LISTA DE PIEZAS			
MARCA	CTDAD	DENOMINACIÓN	MATERIAL
1	1	Barra de dirección	Acero
2	1	Sensor Ultrasonidos	
3	1	Motor servo	
4	1	Powerboost 1000 C	
5	1	Sensor SKU: SEN 109	
6	1	Sensor TSL 2561	
7	1	Motor paso a paso	
8	1	Engranajes	Nylon
9	1	Electrónica de Control	
10	1	Bastidor	Madera
11	1	Sensor DHT22	
12	1	Batería	Polímero de Litio

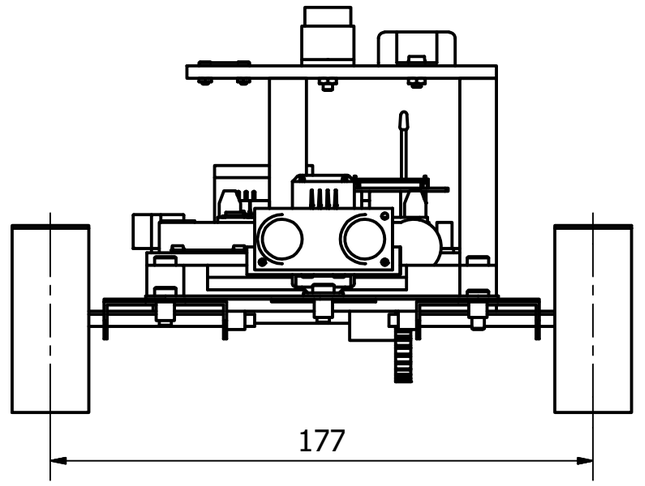
	Fecha	Nombre	Firma:
Dibujado	19-04-2017	Mikel Brun Martínez	
Comprobado			

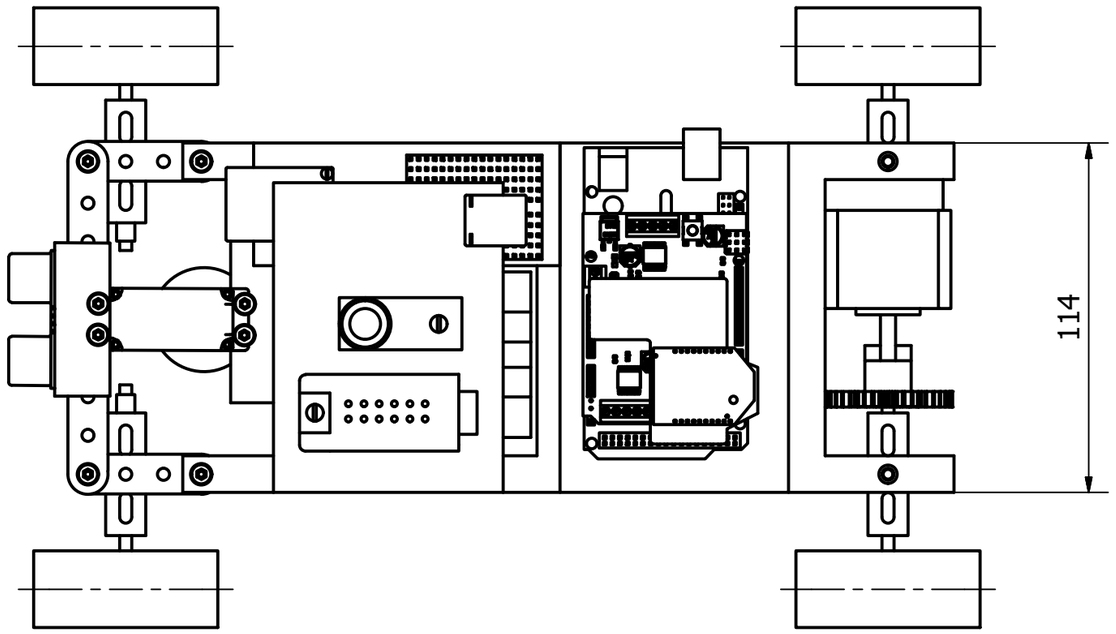
Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Escala	Título	Nº Alumno
S/E	Lista de Piezas	Curso
		Plano Nº

A4







	Fecha	Nombre	Firma:	Escuela de Ingeniería y Arquitectura Universidad Zaragoza
Dibujado	19-04-2017	Mikel Brun Martínez		
Comprobado				
Escala 1:2	Título: Vistas del Conjunto Robot			Nº Alumno
				Curso
				Plano Nº

A3

ANEXO III

SOFTWARE NECESARIO

Se han utilizado los siguientes programas:

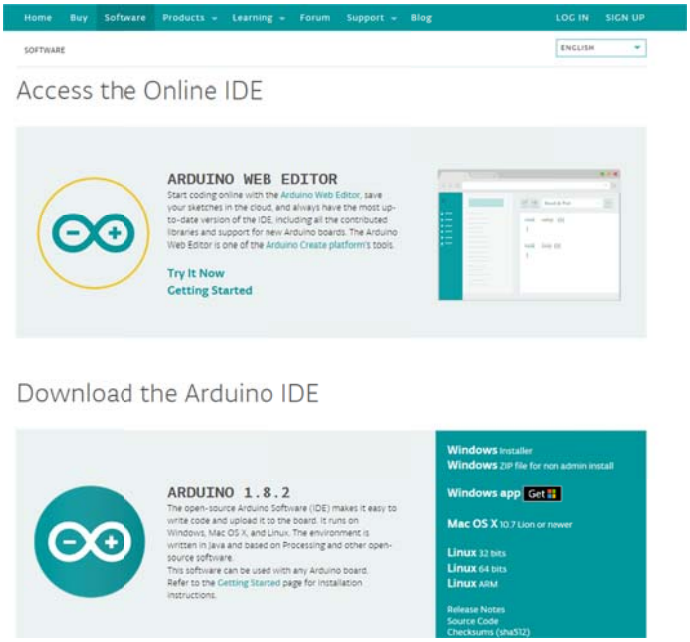
- Arduino Ide para la programación del microcontrolador.
- XCTU para la configuración de los dispositivos XBee.
- QT para la programación del programa de control que se va a ejecutar en el PC.

Todos estos programas son de uso libre y existen versiones para diferentes sistemas operativos. En este trabajo se ha utilizado el sistema operativo Microsoft Windows.

Arduino Ide

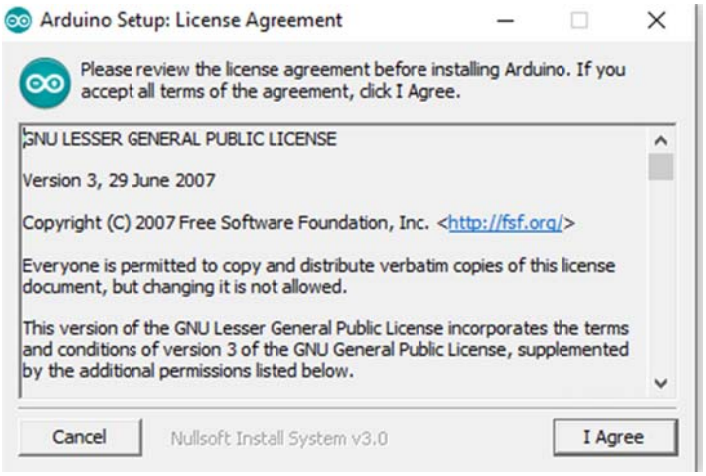
1) *Instalación del programa*

El programa Arduino Ide se puede descargar de la página web oficial de Arduino www.arduino.cc, hay que ir a la pestaña "software" y descargar la última versión. Se debe descargar el instalador para Windows (Windows installer).

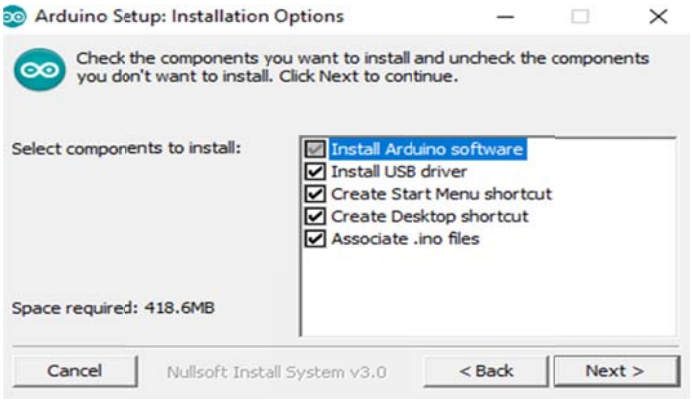


Con esta opción se descarga un archivo ejecutable que realiza todas las tareas de instalación. La versión disponible en marzo de 2017 es la 1.8.2 y el fichero ejecutable se llama arduino-1.8.2-windows.exe.

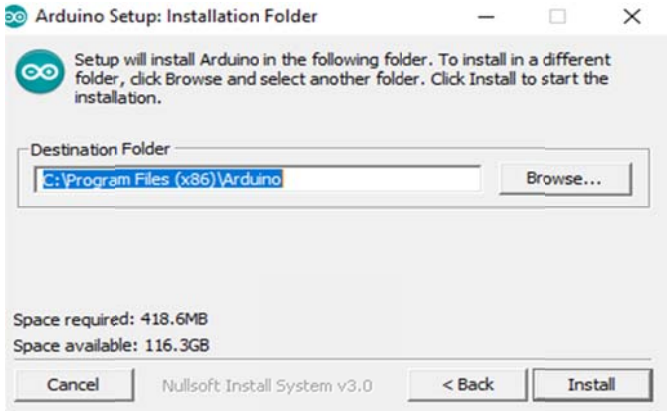
Al ejecutar el programa lo primero que aparece es una pantalla de aceptación de la licencia, que es una licencia GNU GPL (General Public License).



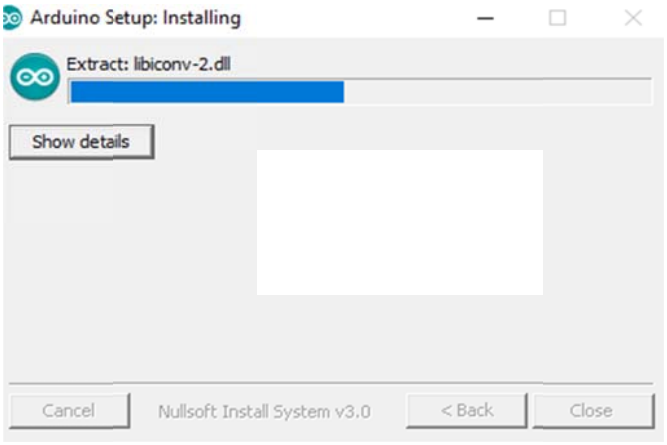
A continuación aparece una pantalla con la lista de componentes a instalar, si se desea asociar las extensiones .ino con el programa y si se desea que el instalador añada accesos directos en el menú de inicio y en el escritorio. Hay que seleccionar todas las casillas.



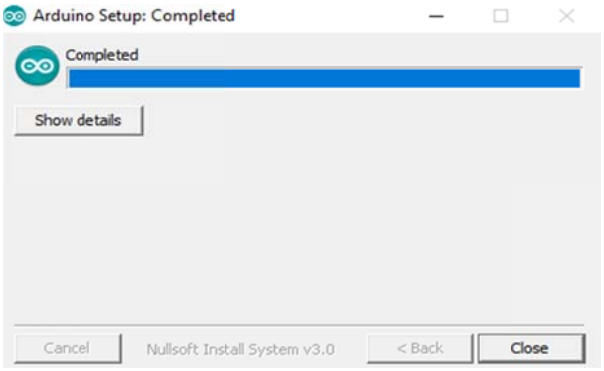
A continuación pregunta por la carpeta en la que se instalará el programa. Se puede dejar por omisión o cambiarla a otra.



A continuación el programa realiza la instalación.



Una vez finalizada la instalación aparece una pantalla informativa y el programa está listo para usarse.



2) Instalación de las bibliotecas necesarias para el proyecto

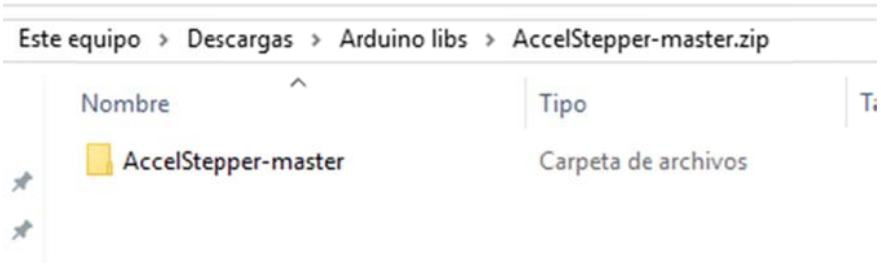
Se han utilizado una serie de bibliotecas para la programación del proyecto. Estas bibliotecas son las siguientes:

- Bibliotecas de Adafruit:
  - Adafruit sensor master: biblioteca base para la utilización de sensores Adafruit. Se puede descargar de https://github.com/adafruit/Adafruit_Sensor
  - Adafruit TSL2561 master: biblioteca para usar el sensor de luminosidad TSL2561. Se puede descargar de https://github.com/adafruit/TSL2561-Arduino-Library
  - Adafruit DHL sensor library: biblioteca para usar el sensor de temperatura y humedad. Se puede descargar de https://github.com/adafruit/DHT-sensor-library
  - Adafruit Motor Shield V2 master library: Biblioteca para el controlador de motores. Se puede descargar de https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software
- Otras bibliotecas:
  - AccelStepper master: biblioteca general para el control de motores paso a paso por pasos. Compatible con la placa Motor Shield de Adafruit. Se puede

descargar de https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software

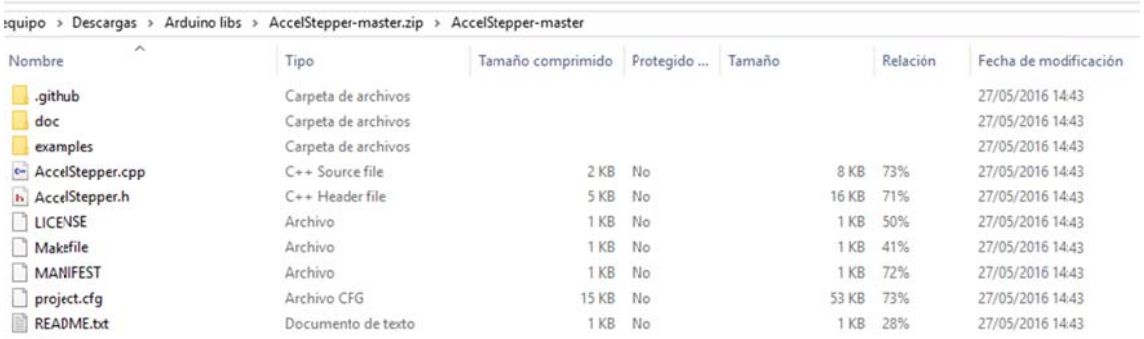
- Ultrasonic: biblioteca general para el control de sensores ultrasónicos. Se puede descargar de https://github.com/JRodrigoTech/Ultrasonic-HC-SR04

La instalación de una librería para Arduino Ide siempre se hace de la misma manera. Todas las librerías se descargan en forma de un fichero comprimido en formato zip. Ese fichero contiene una carpeta de nombre igual al nombre de la biblioteca. Un ejemplo con la biblioteca AccelStepper es el siguiente:

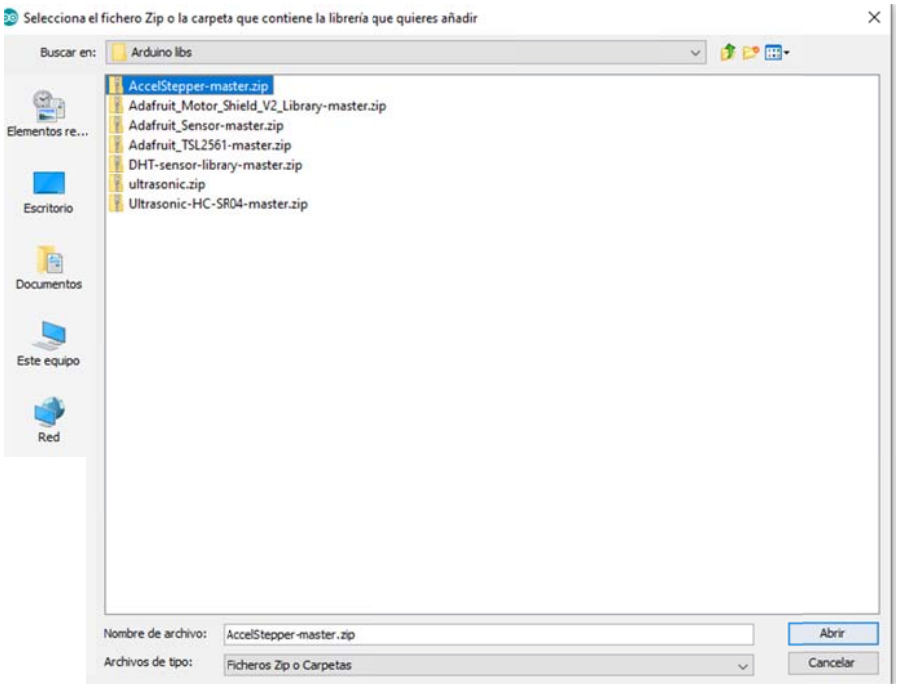


Dentro de esta carpeta siempre hay lo siguiente:

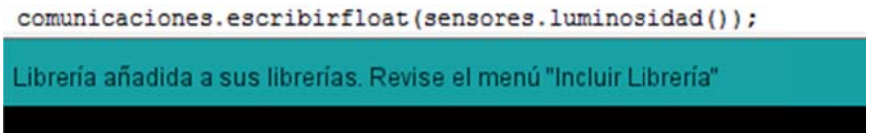
- Un fichero .h que es un fichero de texto de encabezamiento para C/C++ en el que se definen las funciones y clases de la biblioteca.
- Un fichero .cpp que es un fichero de texto que contiene el código fuente de la biblioteca para C++.
- Algunos ficheros de texto tipo readme u otros.
- Puede contener algunas carpetas con documentación y ejemplos de utilización de la librería.



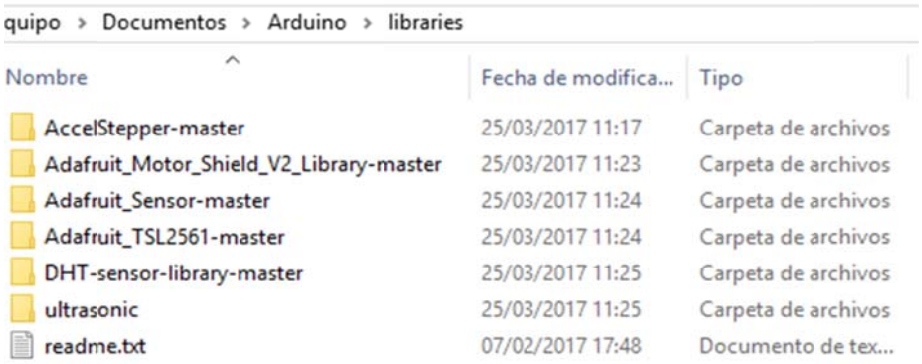
Para la instalación en Arduino Ide hay que iniciar el programa y en la barra de menús elegir la opción programa. En el submenú elegir la opción incluir librería y allí elegir la opción añador librería zip. Se abre una ventana en la cual se debe buscar la ubicación de la librería que se quiere añadir.



Pulsando el botón de abrir se añade la biblioteca al entorno. Un mensaje en la parte inferior del Arduino Ide indica que el proceso ha sido correcto.

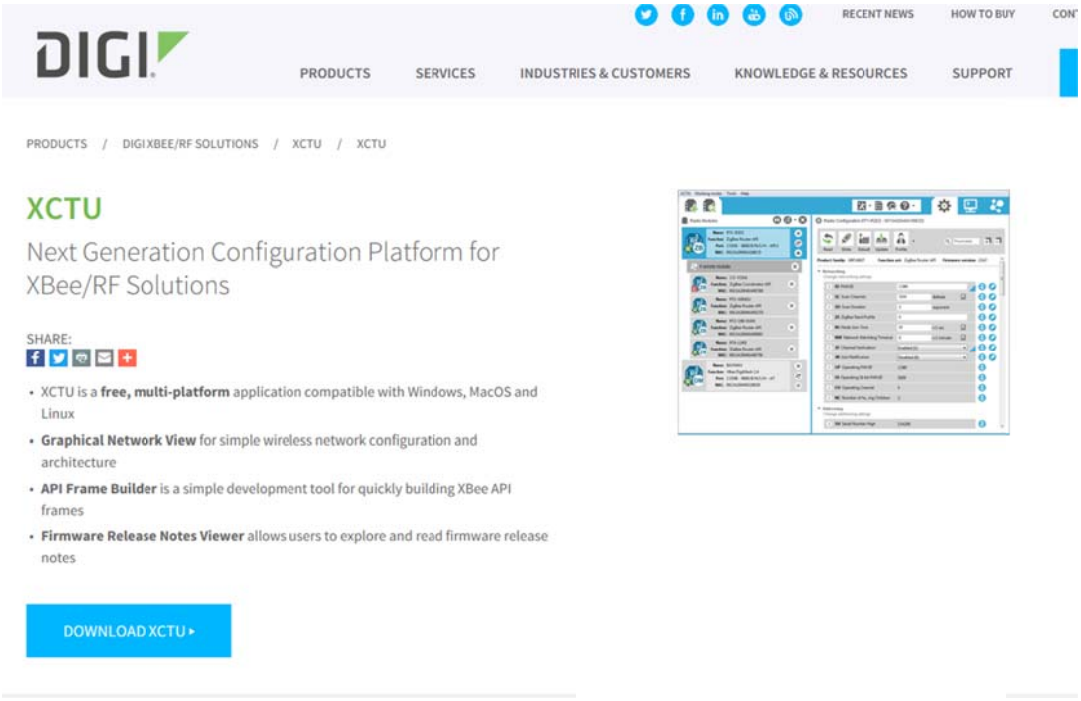


El proceso de instalar la librería es tan sencillo como copiar la carpeta en la carpeta de documentos\Arduino\libraries. Este es el contenido de la carpeta después de instalar todas las librerías.



XCTU

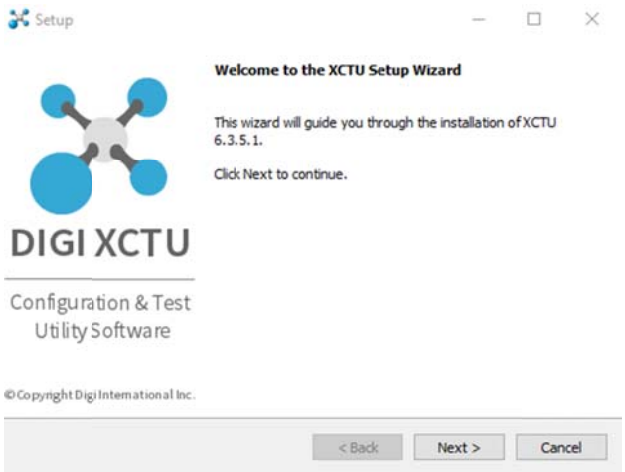
El programa XCTU se utiliza para la configuración de los dispositivos XBEE. Es un programa gratuito que se descarga de la página oficial de Digi www.digi.com accediendo a la pestaña products.



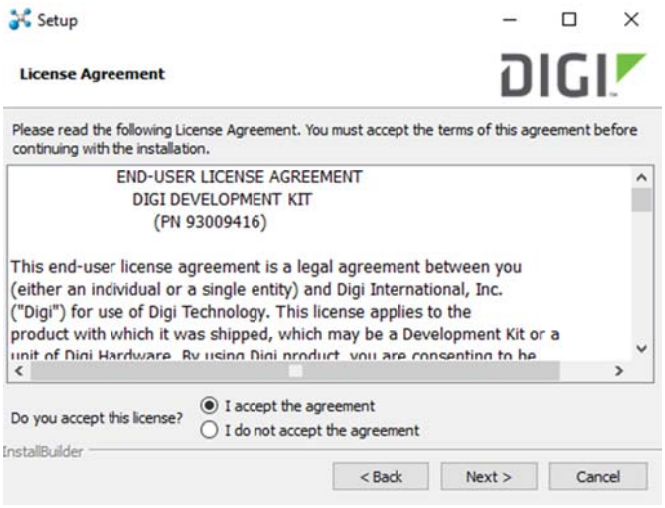
En marzo de 2017 la última versión es la 6.3.5. Hay que descargar la versión para Windows.

La descarga es un fichero ejecutable que se llama 40003026_J.exe.

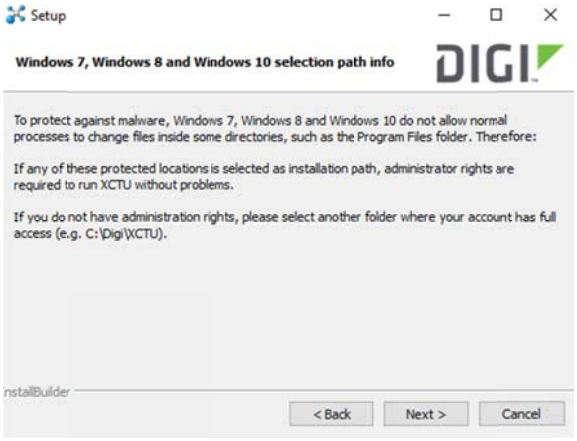
La instalación es muy sencilla. Primero aparece una pantalla de bienvenida.



Luego aparece la pantalla para aceptar las condiciones de licencia.

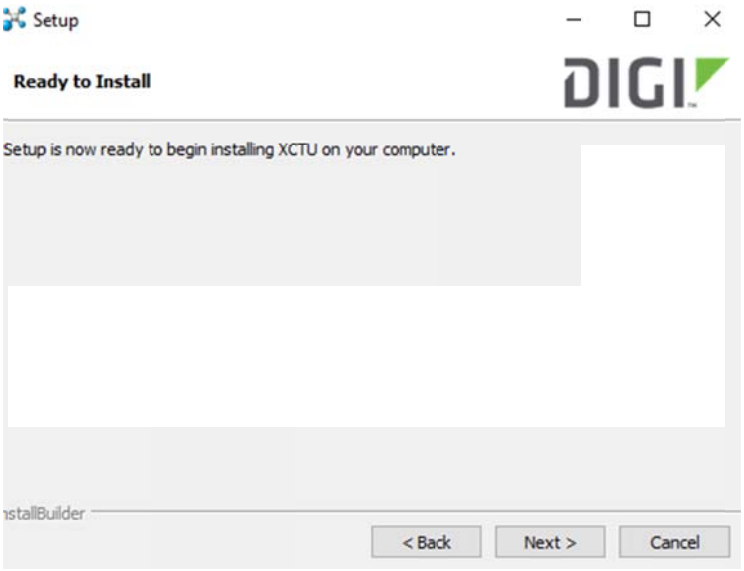


Luego aparece una pantalla para indicar que se necesitan permisos administrativos para instalar la aplicación.

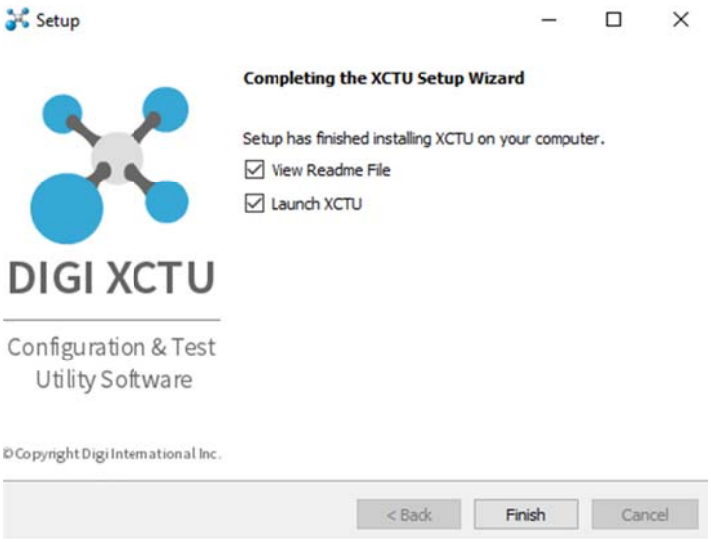


A continuación aparece la ventana para elegir la carpeta de instalación del programa.

Luego aparece una pantalla indicando que el programa de instalación está listo para comenzar.



Una vez terminada la instalación aparece una pantalla indicando que el programa ya está instalado.



La comunicación del programa con los dispositivos XBee se hace a través de un adaptador USB que lo presenta al ordenador como si fuera un dispositivo conectado a un puerto serie utilizando unos drivers FTDI. Estos drivers vienen de modo nativo en Windows 8.x y Windows 10 y se deben instalar en Windows 7. En https://learn.sparkfun.com/tutorials/how-to-install-ftdi-drivers/windows---quick-and-easy se explica cómo hacer la instalación de esos drivers en Windows 7.

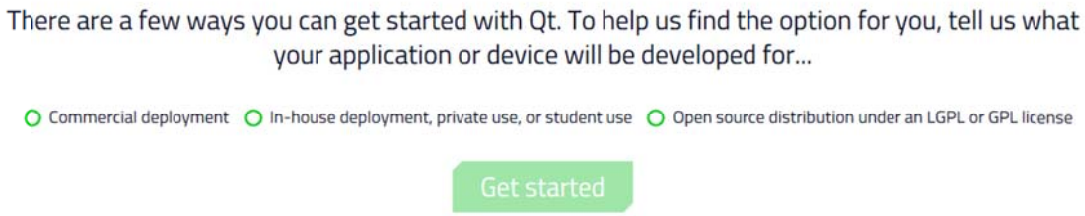
La primera vez que se instala el programa busca las últimas versiones del firmware para los controladores y por eso hay que esperar un poco antes de utilizarlo. Es importante que los dos Xbee se configuren con la misma versión del programa y usen el mismo firmware. La manera más rápida de configurar los dispositivos XBee es utilizar dos ordenadores con dos adaptadores XBee porque se puede testear que ambos dispositivos se comunican sin problemas. Esa es la configuración que se va a suponer en el anexo aunque se puede seguir si solo se dispone de un ordenador y un adaptador XBee aunque no se puedan hacer las pruebas de conexión.

QT

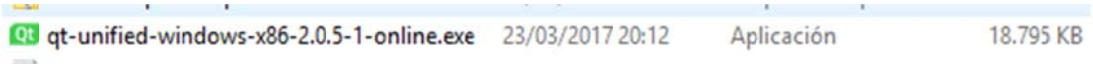
El programa de control es un programa Windows y para su desarrollo se ha empleado el entorno de desarrollo Qt que simplifica la creación de programas para Windows y además es multiplataforma, lo que permitiría crear el mismo programa para otros dispositivos como por ejemplo dispositivos Android.

El instalador del programa se puede descargar de https://www.qt.io/es/ hay un botón que permite descargarlo directamente.

Al pulsar el botón se debe elegir el tipo de producto a descargar. Se recomienda elegir la versión Open Source, que es la versión libre del producto. También existen versiones de pago para desarrolladores profesionales.

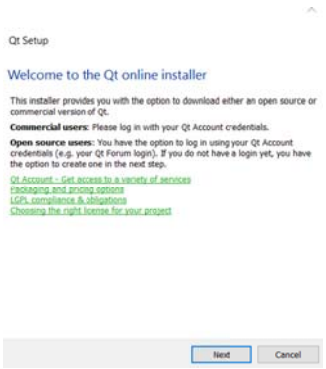


Después de pasar unas pantallas en las que se acepta las condiciones de uso del software con licencia GPL se descarga un pequeño programa que es el instalador. Este instalador necesita conexión a internet para descargarse los módulos necesarios.

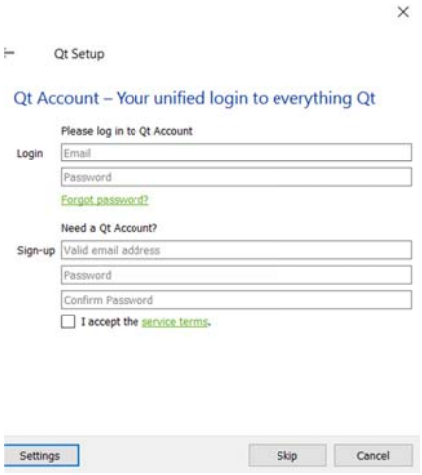


Hay que destacar que es necesario abrirse una cuenta en la web de Qt para poder descargarse el programa. Debe estar asociada a una cuenta de correo y el fabricante suele enviar correos con información de novedades del producto.

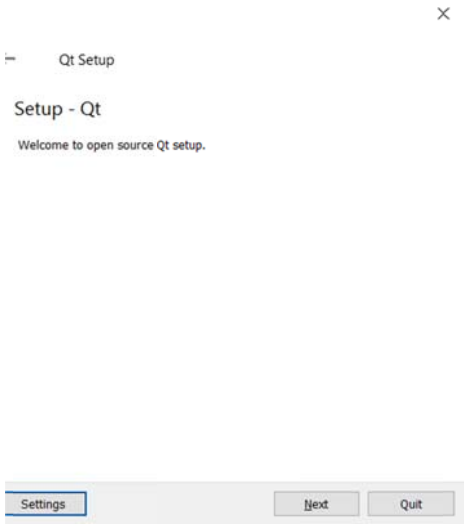
Al ejecutar el instalador lo primero que aparece es una pantalla de bienvenida.



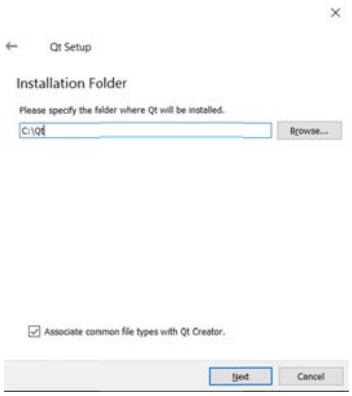
A continuación solicita una cuenta de Qt para realizar la instalación. Si no se dispone de una cuenta se puede crear en la misma pantalla. Siempre debe estar asociada a una dirección de correo.



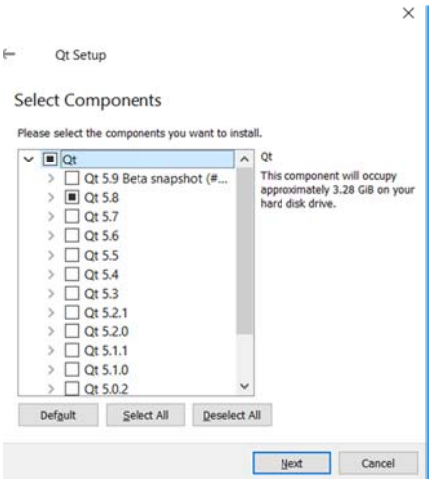
Al rellenar la información de la cuenta el botón skip pasa a ser un botón next y si todo es correcto aparece una pantalla confirmando que va a dar comienzo el proceso de instalación.



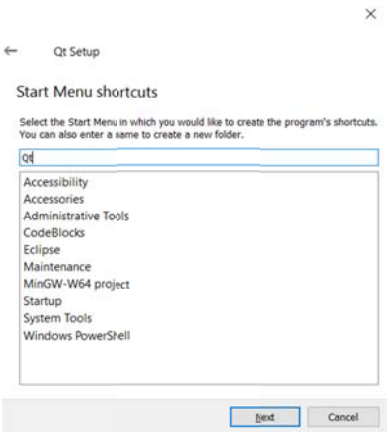
A continuación aparece una pantalla para indicar la carpeta en la que se desea instalar el producto.



La siguiente pantalla permite elegir los componentes a instalar. Si no hay programas hechos con versiones anteriores se puede instalar solamente la última versión estable para ahorrar espacio ya que es una herramienta que ocupa bastante espacio en disco duro.



A continuación se acepta el acuerdo de licencia y se elige en qué carpeta del menú de inicio se desea instalar los accesos a la aplicación.



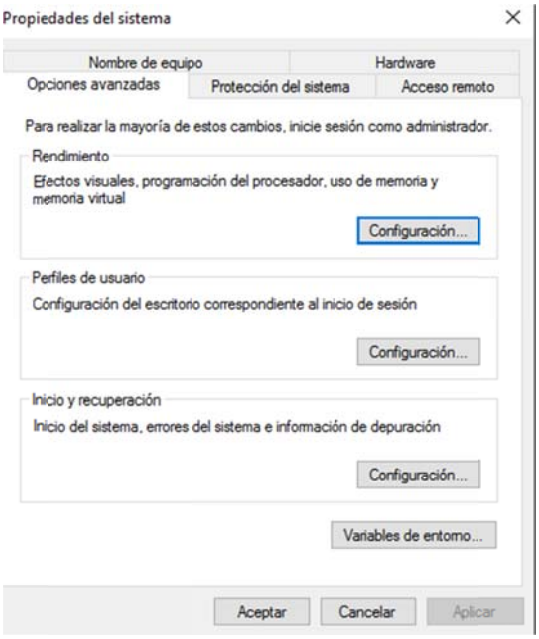
Pulsando next comienza la instalación de la aplicación. Se necesita conexión a internet para que el programa descargue los módulos seleccionados.

La versión gratuita de Qt viene con el compilador MinGw 5.3.0, que se encuentra en la carpeta tools\mingw530_32 dentro de la carpeta en la que se ha instalado Qt. Los programas creados con Qt ocupan muy poco y eso se debe a que no incluyen todo el código base necesario y

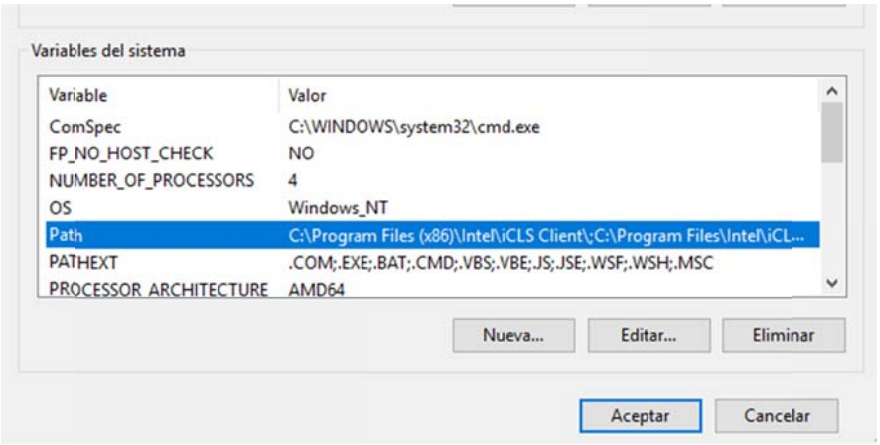
requiere de una serie de DLLs para su funcionamiento, por eso tratar de ejecutar directamente el programa puede dar errores. Aunque es posible crear programas que contengan todo el código necesario en su interior, para eso se necesitaría una instalación especial de Qt que está fuera del alcance de este proyecto.

Una manera sencilla de preparar un ordenador para ejecutar aplicaciones Qt en un ordenador cualquiera es la siguiente:

- Crear una carpeta cualquiera, por ejemplo, c:\bin, y añadirla a la variable path del ordenador, para lo cual hay que hacer lo siguiente:
    - Abrir la opción sistema en el panel de control.
    - En esa ventana elegir configuración avanzada del sistema.



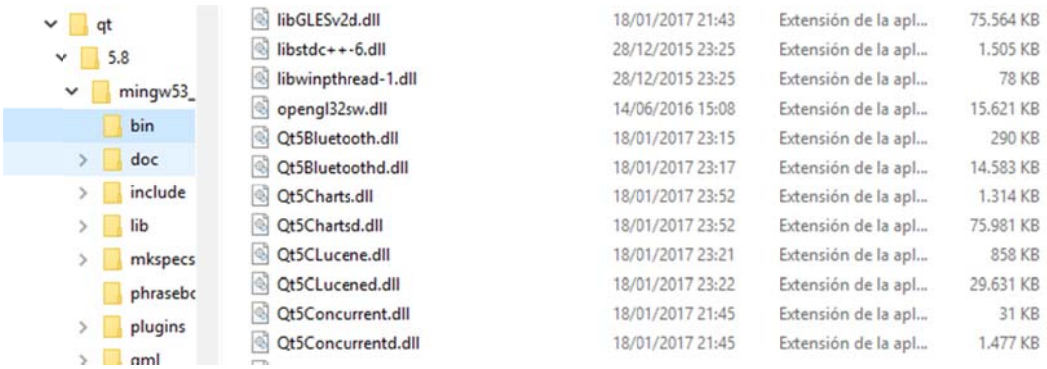
    - Elegir la opción “Variables de entorno”.
    - En la parte de las variables del sistema editar la variable path.



    - Añadir la ruta a la carpeta que se ha creado. En Windows 7 hay que ir al final de la línea, añadir un punto y coma (;) y luego añadir la ruta a la carpeta

creada. En Windows 10 hay un editor específico que facilita añadir una ruta más.

- Copiar en la carpeta creada todas las dlls necesarias que se encuentran en la carpeta 5.8\mingw53_32\bin (si el compilador usado es MinGW).

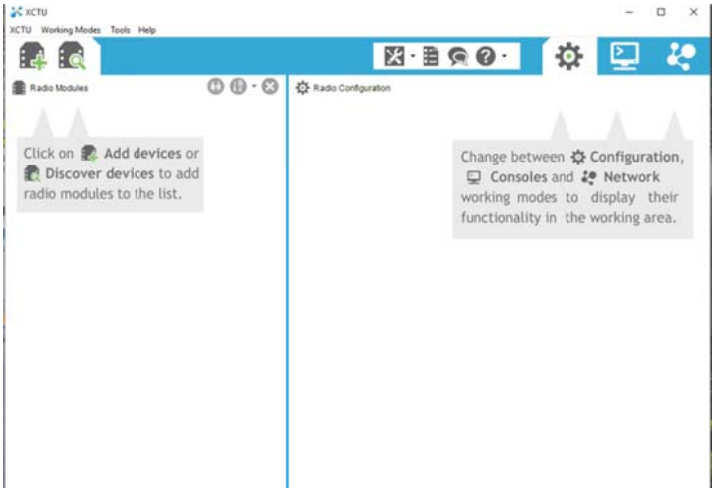


El tamaño final de la carpeta es bastante grande, unos 2,42Gb, pero con esta sencilla instalación se garantiza que todos los programas se podrán ejecutar en cualquier ordenador sin problemas. Se ha preferido añadir estas DLLs a una carpeta aparte y no a la carpeta del sistema (Windows\system32) para que la instalación sea más limpia y si se deseara eliminar esta instalación bastaría con eliminar la ruta de la variable path y borrar la carpeta.

ANEXO IV

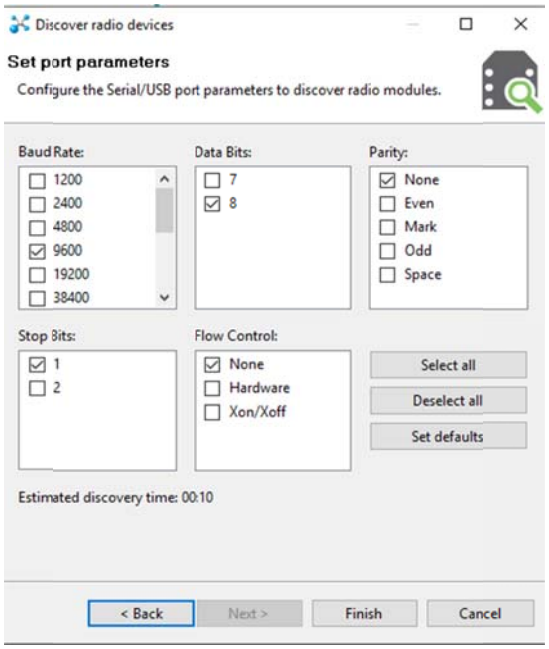
CONFIGURACIÓN DE LOS DISPOSITIVOS XBEE

Al iniciar el programa XCTU aparece la siguiente pantalla:

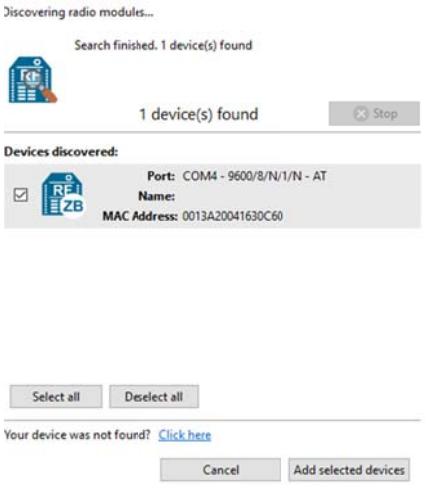


Si todo es correcto al pulsar el botón “discover radio devices” debe encontrar el Xbee conectado. Primero pide seleccionar el puerto a escanear, que es el puerto USB.

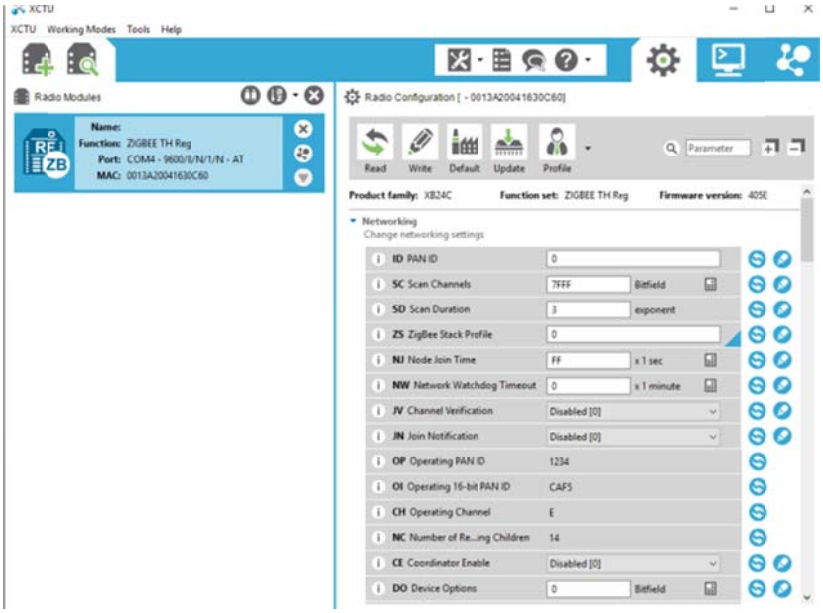
A continuación se fijan los parámetros del puerto. Se pueden dejar los parámetros por omisión. Estos parámetros deben coincidir en ambos ordenadores o la comunicación será imposible.



Si todo es correcto se verá parpadear algunas luces en el adaptador USB y después de unos instantes aparecerá el Xbee. Se pulsa Add selected devices y ya se puede comenzar con la configuración.

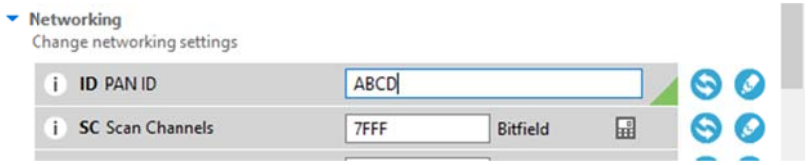


Una vez añadido el Xbee, haciendo click en el mismo se accede a la pantalla de configuración.



Existen muchos parámetros para configurar muchos tipos de comunicaciones pero para este caso se va a establecer una comunicación simple bidireccional, para lo cual se necesita cambiar unos pocos parámetros

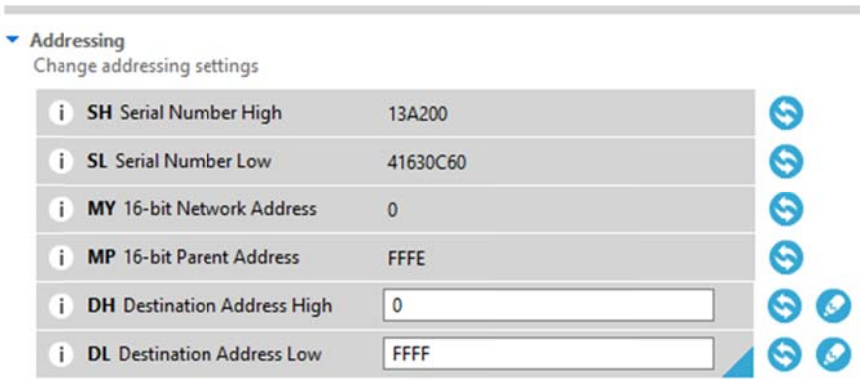
El primer parámetro es Pan ID que identifica la red. Es un número hexadecimal de cuatro dígitos. Se puede elegir uno cualquiera que sea fácil de recordar como ABCD.



Notar que al cambiarlo aparece un triángulo verde en la parte derecha. Eso significa que hay que confirmar el cambio pulsando en el botón derecho en forma de lápiz. Una vez confirmado el cambio el triángulo se vuelve azul.

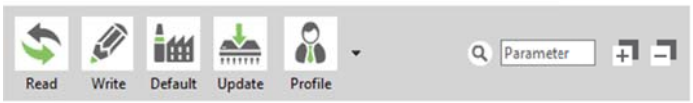
Para establecer la red, uno de los dos dispositivos tiene que ser coordinador. Se suele seleccionar el Xbee que va a estar conectado al ordenador. Se selecciona con el parámetro CE.

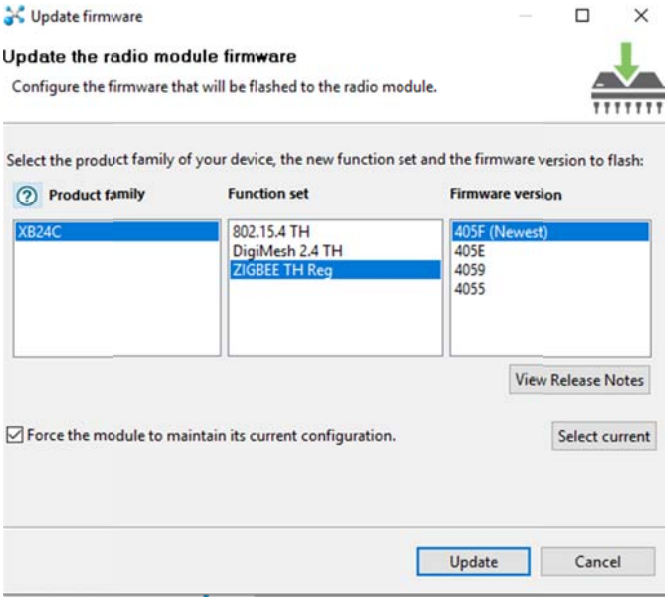
En el dispositivo coordinador hay que cambiar el parámetro dirección destino DL por FFFF para que encuentre todos los dispositivos de la red.



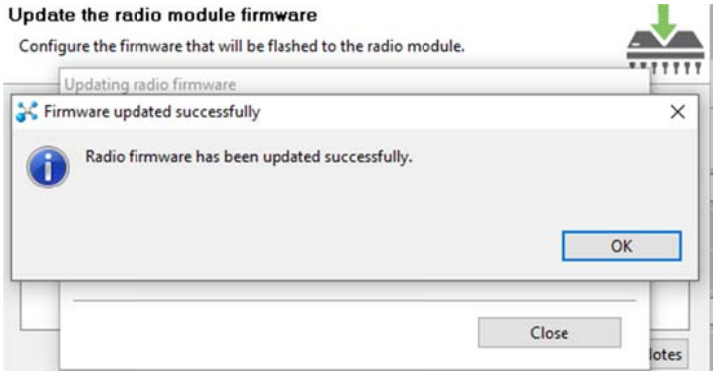
En el dispositivo que no es coordinador tan solo hay que configurar el parámetro Pan ID que identifica la red. El resto se deja por omisión.

Para que la configuración sea efectiva hay que grabarla en el dispositivo y eso se hace usando el botón update.



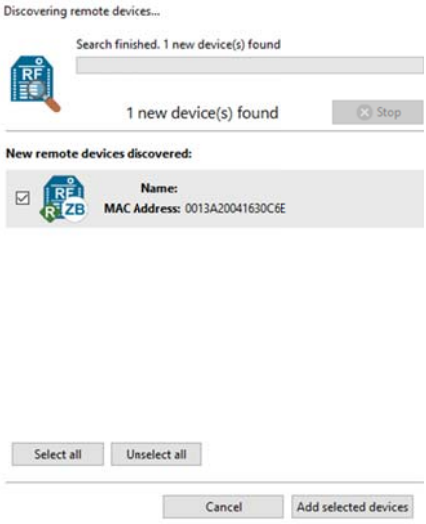


Es importante elegir la función ZIGBEE TH Reg en el segundo cuadro para poder tener toda la funcionalidad de la nueva versión S2C y elegir la última versión del firmware. Si todo va bien se recibe un mensaje de confirmación.



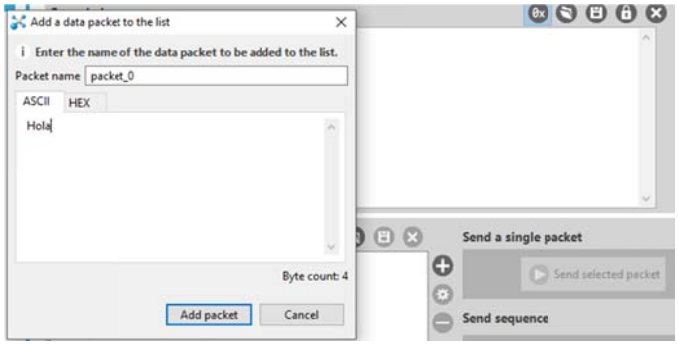
Ahora se puede probar la comunicación. Se usa la opción de buscar otros módulos en la vecindad (segundo botón a la derecha, el que tiene forma de varios nodos).

Y debe encontrar el nuevo dispositivo.

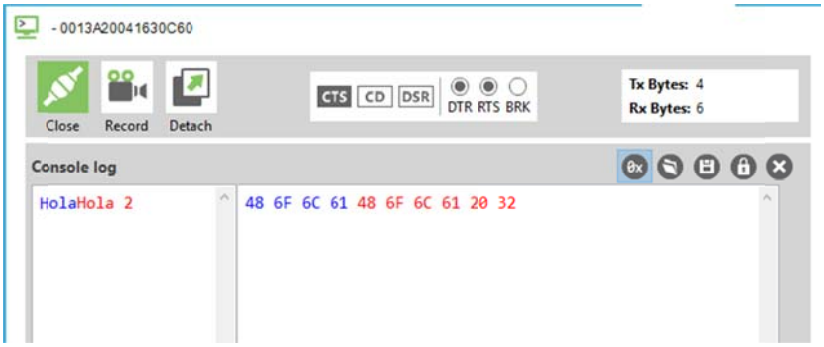


Ahora se inicia una sesión de terminal en los dos dispositivos usando el botón de terminal (el segundo).

Se pulsa open para conectar y se usa la opción send single package para enviar datos de un dispositivo a otro.



Una vez listo el texto se envía el paquete, en la consola del otro dispositivo aparece el mensaje y se comprueba que el otro dispositivo puede contestar. Si todo va bien ésta es una posible pantalla del dispositivo que ha iniciado la conversación.



En azul los datos que ha enviado este dispositivo y en rojo los datos que ha recibido. Con esto, la configuración de los dispositivos xBee es correcta.

ANEXO V

PROGRAMA ROBOT ARDUINO

1) Principios del diseño

En la elaboración del programa del microcontrolador se han seguido los siguientes criterios:

- Utilización de programación orientada a objetos.
- Modularidad: dividir el programa en módulos lo más independientes posibles.
- Optimización de recursos.
- Interfaz con el programa de control.

Se ha utilizado C++ que además de una programación orientada a objetos, permite definir clases para conseguir una mejor modularidad del diseño.

Todo programa consiste en una serie de funciones que trabajan con una serie de datos. En la programación tradicional, datos y funciones son independientes mientras que en la programación orientada a objetos los datos y las funciones que tratan con ellos se unen en una clase. Una clase se asemeja al tipo de datos conocido como estructura en programación tradicional al que se le añaden funciones para trabajar con esos datos. Al igual que en el caso de las estructuras en la programación tradicional, una clase es solo una definición, se necesita declarar al menos una variable de ese tipo de dato para poder almacenar una determinada información. Una variable de una determinada clase es lo que se conoce como un *objeto*.

Prácticamente todos los programas que se hacen hoy en día utilizan programación orientada a objetos. Todas las librerías de apoyo empleadas en la elaboración del programa utilizan orientación a objetos.

La orientación a objetos ha sido la base que ha permitido dividir el programa en módulos independientes. Se ha diseñado el programa principal para que se apoye en tres módulos principales, que son los siguientes:

- Módulo de comunicaciones: se encarga de todo lo relacionado con la recepción de comandos y envío de información al programa de control.
- Módulo de sensores: Se encarga de todas las tareas relacionadas con los sensores.
- Módulo de motores: Se encarga de todo lo relacionado con los motores.

La programación de cada uno de estos módulos se ha hecho en un fichero de encabezamiento con extensión .h de manera que con una simple instrucción #include en el programa principal ya se tiene acceso a la funcionalidad de dicho módulo. Dada la simplicidad del entorno de programación Arduino Ide se ha considerado que es una buena solución. El programa se ha estructurado en los siguientes ficheros fuente:

- El programa principal del Arduino llamado “programarobot.ino”.
- Tres ficheros tipo .h llamados “comunicaciones.h”, “motores.h” y “sensores.h”.

Otro criterio de diseño ha sido la optimización de recursos del Arduino. El programa del microcontrolador es un ejemplo claro de *control en tiempo real* donde la optimización de los recursos es importante, por eso se ha preferido usar variables globales y no locales, ya que se inicializan al comienzo del programa y ya están disponibles para siempre, y se ha decidido que las funciones no reciban parámetros siempre que sea posible, de hecho las únicas funciones que reciben parámetros están en el módulo de motores.

El microcontrolador debe recibir órdenes y transmitir información al programa de control. Se ha definido un lenguaje de comandos que permiten este diálogo entre el programa de control y el programa del microcontrolador.

A continuación se van a analizar todos los módulos que componen el programa y se detallarán los comandos empleados.

2) Módulos que forman el programa

Módulo de comunicaciones

Está definido en el fichero comunicaciones.h.

Consta de una estructura y una clase.

La estructura define cómo son los comandos a intercambiar con el programa de control, que estarán formados por un carácter y un entero.

La clase se llama TComunicaciones. La comunicación se realiza a través del puerto serie, que se encuentra conectado al puerto serie del Arduino a través del shield. Tiene una función de inicialización que inicializa la velocidad de comunicación a 9600 baudios (bits por segundo). Pese a que XBee puede funcionar a velocidades mayores se ha optado por esta velocidad por ser la más empleada y es lo suficientemente elevada para la información que se va a intercambiar. En la clase se ha definido una variable llamada “orden” que almacenará el comando recibido.

La función más importante es la que lee el comando, que se llama “obtenerorden”. Primero lee un carácter, que es el comando a ejecutar. A continuación va leyendo todos los caracteres que han llegado y los almacena en un vector interno. El programa espera que esos caracteres se correspondan con la representación de un número entero (caracteres del ‘1’ al ‘9’ y el signo ‘-‘ si el número es negativo). Una vez que se han leído todos los caracteres llama a la función “atoi” que convierte una cadena de caracteres en un número entero.

Módulo de sensores

La primera parte del módulo define las constantes que se emplearán en el programa, como por ejemplo los pines a los que están conectados los diferentes sensores. Si se decide conectar los sensores a otros pines bastará con cambiar las constantes y volver a cargar el programa en el Arduino para que todo funcione.

La segunda parte define las variables de los sensores. Hay una variable para el sensor de temperatura y humedad, otra para el sensor de luminosidad y otra para el sensor de distancia. El sensor de CO_2 obtiene los resultados de aplicar una fórmula a la lectura de la tensión en un pin determinado y por eso no necesita una variable. Notar que en los otros tres casos las variables son en realidad objetos de una clase.

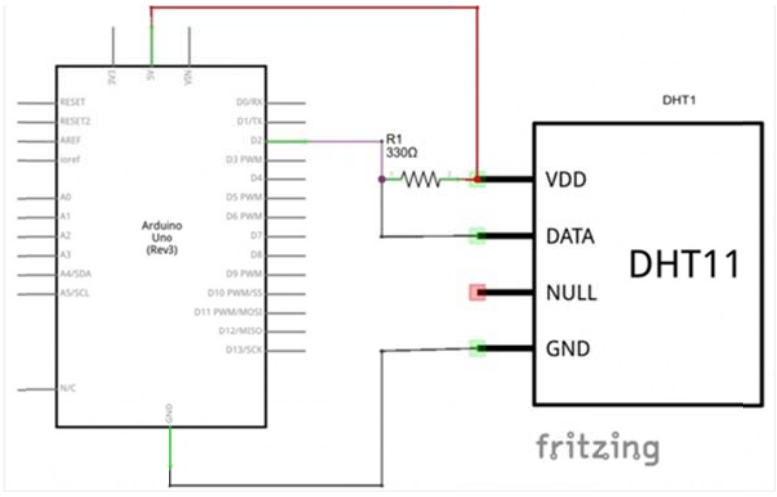
Se ha definido una clase llamada TSensores que permite obtener toda la información de los sensores. Tiene variables para almacenar los valores de luminosidad, temperatura, humedad y CO_2 de los sensores. Las funciones principales son las siguientes:

- Inicializar: Se encarga de inicializar todos los sensores tal y como se especifica en la documentación de los fabricantes. También se inicializan los pines.
- Actualizarvalores: Es la función que lee la información de todos los sensores excepto el de distancia y almacena la información en las variables definidas para que sus valores puedan ser consultados.
- Distancia: Esta función debe ser llamada con mucha frecuencia mientras el robot se está moviendo para detectar obstáculos, por eso se ha decidido que la función devuelva directamente la distancia y no la almacene en una variable intermedia como en el caso de los otros sensores.

Un detalle a destacar en la inicialización de los pines es que hay un par de pines que se definen como pines de entrada y a la vez se escribe inmediatamente un valor en ellos, por ejemplo:

```
pinMode(DHTpin,INPUT);
digitalWrite(DHTpin,HIGH);
```

Esto se hace porque para afinar la medición el fabricante recomienda que se coloque una resistencia pull-up según el siguiente esquema:



Arduino permite emular esa resistencia física elevando la tensión del pin de entrada hasta los 5V.

Módulo de motores

Este módulo es sin duda el más complejo de todos por la utilización de numerosas librerías. El tratamiento de los dos motores es diferente y por eso se debe analizar por separado. Toda la funcionalidad se encuentra en la clase TMotores.

Motor servo

El motor shield de Adafruit solo proporciona un conector para las tres señales que necesita un servo (alimentación, tierra y pin para el control por PWM). Para su control se utiliza la biblioteca "servo.h", que es estándar de Arduino. Se ha definido un objeto llamado "miservo" de la clase Servo con el que se realiza todo el control.

En la función de inicialización se utiliza la función "attach" para indicarle a la librería qué pin se utiliza para controlarlo. También se ha definido una variable llamada "pos" para almacenar la posición del servo, que se fija usando la función "write" de la clase Servo.

La posición del motor servo se fija pasándole un ángulo en grados entre 0^o y 180^o. Por diseño del robot la posición central de la dirección se corresponde con un ángulo de 90^o de manera que para ángulos entre 0^o y 90^o la dirección está girada a la izquierda y entre 90^o y 180^o está girada a la derecha. Se ha observado que el máximo ángulo que puede girar la dirección en un sentido u otro es de 50^o por lo que los valores del ángulo del servo siempre estarán entre 40^o y 140^o.

Las funciones que controlan el servo son las siguientes:

- Derecha: Gira el servo a la derecha un ángulo que debe estar comprendido entre 0^o y 50^o. Si X es ese ángulo, el ángulo del servo será 90^o+X.
- Izquierda: Igual que la función anterior pero girando a la izquierda, por lo que el ángulo del servo será 90^o-X.
- Centro: Posiciona el servo en el centro, es decir 90^o.
- Posición: Permite saber la posición del servo en un momento dado, que es el valor de la variable "pos".

Motor paso a paso

El control del motor paso a paso es más complejo.

Primero se tiene que definir un objeto que representa el motor shield de Adafruit indicando la dirección del bus I2C en el que está conectado y que por definición es el valor hexadecimal 60. Esto se hace con la siguiente declaración:

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x60);
```

A continuación hay que indicar que hay un motor paso a paso conectado indicando dónde está conectado y las características del motor en términos de número de pasos por vuelta. Esto se hace en la siguiente instrucción:

```
Adafruit_StepperMotor *miMotor = AFMS.getStepper(PASOS, 2);
```

La constante PASOS se ha definido como 200 porque el motor utilizado es de esas características. El número 2 indica que se ha conectado al segundo terminal de conexiones.

Para que todo funcione hay que hacer una inicialización que consiste en lo siguiente:

- Inicializar la biblioteca Wire haciendo “Wire.begin()”. Wire es un objeto de esa librería. La biblioteca Wire es estándar de Arduino y se utiliza para comunicaciones usando el bus I2C. La emplea la librería del controlador de motores.
- Inicializar el shield de motores (definido en la variable AFMS) usando la función begin (instrucción AFMS.begin()).
- Se deja el motor en reposo fijando su velocidad a cero. Notar que hay una variable que almacena la velocidad que se solicita al motor.

Para que el motor se mueva es necesario:

- Fijar la velocidad deseada en revoluciones por minuto.
- Llamar a una función determinada pasándole el número de pasos que se desea mover el motor.

El problema de las funciones de la biblioteca de Adafruit es que *interrumpen la ejecución del programa hasta que el motor se ha movido ese número de pasos* y eso impide hacer otras tareas como controlar que no hay ningún obstáculo en el camino. Lo que se necesita es una función que después de avanzar una parte del camino *devuelva el control al programa* para que se puedan hacer esas comprobaciones. Esa es la función de la biblioteca AccelStepper, que Adafruit aconseja utilizar para estas tareas.

La librería AccelStepper es independiente de cualquier controlador de motores y de cualquier motor paso a paso, a diferencia de las librerías de Adafruit, que solo sirven para sus controladores de motores. Tan solo necesita saber cómo hacer que el motor paso a paso ejecute un paso y ese es el objeto de las dos funciones definidas como “adelante” y “atras”, cuyo código está ligado al motor que se ha implementado:

```
void adelante() {
  miMotor->onestep(FORWARD,SINGLE);
}
void atras() {
  miMotor->onestep(BACKWARD,SINGLE);
}
```

Con estas dos funciones se define el objeto del tipo AccelStepper.

```
AccelStepper motor(adelante,atras);
```

Y todas las funciones para mover el motor van a hacer referencia a la variable “motor” y no al motor propiamente dicho, representado por la variable “miMotor”.

Se han utilizado las siguientes funciones de la librería AccelStepper:

- setSpeed: Fija la velocidad del motor en revoluciones por minuto. Según la documentación de la librería lo que hace es simplemente establecer un retardo después de dar un paso usando la función estándar de Arduino “delay”.
- setCurrentPosition: Establece la posición actual del motor en pasos.
- Move: Establece el número de pasos de la posición final.

- Run: Si la posición actual es diferente de la posición final se mueve un paso. Devuelve “true” si ha ejecutado el paso y “false” si ya ha llegado a su destino y no ejecuta el paso.
- distanceToGo: Devuelve el número de pasos que faltan hasta llegar al objetivo.

Si la posición actual en número de pasos es mayor que la posición final el motor paso a paso se mueve hacia atrás y si la posición final en número de pasos es superior a la posición actual el motor se mueve hacia adelante.

La documentación de la librería recomienda llamar a la función “run” en el bucle principal del programa del microcontrolador para que el movimiento sea continuo.

Estas características de la librería AccelStepper han permitido programar el movimiento del motor de la siguiente manera:

- Se ha definido una variable booleana llamada “r” para indicar si el motor debe moverse o no.
- Dada una distancia que se desea desplazar el robot lo primero que hay que hacer es calcular el número de pasos equivalente. Una vez calculado el número de pasos se fija la posición inicial a cero y la posición final a ese número de pasos.
- Cuando se desea mover el robot simplemente se cambia el valor de la variable “r”.
- En el bucle principal del Arduino se llamará a la siguiente función:

```
bool TMotores::unpaso() {
  if (r) return !motor.run();  // Solo un paso
   else return true;         // Esta parado
}
```

Esta función devolverá “true” si el motor ya ha llegado a su destino o bien si está parado y “false” si el motor tiene que seguir moviéndose. Si en un momento determinado se quiere parar el robot basta con hacer que la variable “r” valga “false”.

La función distancia a mover recibe como parámetro los centímetros que se debe mover el robot. Esos centímetros se deben traducir a pasos y se necesita un factor de conversión que se ha calculado a partir del radio de las ruedas, el número de pasos por vuelta y el factor de reducción de la transmisión.

```
float distanciaporpaso=(2*3.14*RADIO)/(PASOS*REDUCTOR);
```

Este mismo factor se emplea en la función que devuelve la distancia que falta hasta el destino. El código de ambas funciones es el siguiente:

```
void TMotores::distancia(int l) {
int p=round((float) l/distanciaporpaso);
motor.setCurrentPosition(0);
motor.move(p);
}

int TMotores::distancia() {
return round(((float)motor.distanceToGo())*distanciaporpaso);
```

```
}
```

3) Estructura del programa principal

El programa principal define los objetos “sensores”, “comunicaciones” y “motores” para trabajar con las librerías anteriores, una variable para almacenar la distancia de seguridad y otra para indicar si se quiere trabajar en modo consola (envío directo de comandos desde el monitor del puerto serie, por ejemplo) o por programa de control.

La función de inicialización es muy simple y se limita a llamar a las funciones de inicialización definidas en las diferentes librerías.

```
void setup() {
comunicaciones.inicializar();
sensores.inicializar();
motores.inicializar();
pinMode(PINLED,OUTPUT);
digitalWrite(PINLED,LOW);
}
```

El bucle principal del programa comprueba si se ha recibido algún comando y si es así lo recibe y lo interpreta y si el robot se está moviendo entonces realiza las tareas de comprobación de que todo va bien. Su código es bastante sencillo.

```
void loop() {
if (comunicaciones.hayorden()) {
  comunicaciones.obtenerorden();
  interpretarorden();
}
if (motores.semueve()) {
  tareas();
}
}
```

El intérprete de comandos se ha implementado como una simple instrucción switch de C y no requiere de grandes explicaciones. En el siguiente apartado se listan todos los comandos admitidos.

```
void interpretarorden() {
  switch(comunicaciones.miorden.clave) {
  // Fija el modo consola
    case 's':fijarconsola(comunicaciones.miorden.valor);break;
  // Mandar datos sensores
    case 'A': mandaractualizacion();break;
    case 'T': mandartemperatura();break;
    case 'H': mandarhumedad();break;
    case 'L': mandarluminosidad();break;
    case 'C': mandarCO2();break;
    case 'D': mandardistancia();break;
```

```
   case 'R': mandarlongitud();break;
  // Recibir ordenes de giro
   case 'd':motores.derecha(comunicaciones.miorden.valor);break;
   case 'i':motores.izquierda(comunicaciones.miorden.valor);break;
   case 'c':motores.centro();break;
  // Recibir ordenes para movimiento
   case 'x':fijardistancia(comunicaciones.miorden.valor);break;
   case 'v':motores.velocidad(comunicaciones.miorden.valor);break;
   case 'l':motores.distancia(comunicaciones.miorden.valor);break;
   case 'g':motores.marchar();contador=200;break;
   case 'p':motores.parar();digitalWrite(PINLED,LOW);break;
  }
  if (consola)  comunicaciones.escribircadena("Esperando\n");
}
```

Notar que los comandos para petición de información se han codificado con letras mayúsculas y los comandos para ordenar al robot que haga algo se han codificado en minúsculas. Las funciones para mandar un determinado valor al programa de control se han codificado todas de la misma manera, por ejemplo, la función “mandartemperatura” se ha programado de la siguiente manera:

```
void mandartemperatura() {
if (consola) comunicaciones.escribircadena("Temperatura: ");
comunicaciones.escribirfloat(sensores.temperatura());
comunicaciones.saltarlinea();
}
```

Notar el uso de la variable “consola” para enviar mensajes informativos con el dato o no hacerlo dependiendo del modo de comunicación elegido y que siempre se envía el salto de línea para indicar que se ha terminado de enviar la información. Esto sirve para sincronización en el programa de control.

Cuando el robot se está moviendo se ejecuta la función “tareas” cuyo código es el siguiente:

```
void tareas() {
  if (motores.unpaso())
  { // Si se esta moviendo y ha llegado al final, parar
    motores.parar();
    if (consola) comunicaciones.escribircadena("Final\n");
    else {comunicaciones.escribirchar('s');
          comunicaciones.escribirentero(0);
          comunicaciones.saltarlinea();};
  } else contador++;
  if(contador>=200) {
    contador=0;
    if (sensores.distancia()<distanciaseguridad) {
        digitalWrite(PINLED,HIGH);
        motores.parar();
        if (consola) {comunicaciones.escribircadena("Obstaculo en ");
                comunicaciones.escribirentero(sensores.distancia());
                comunicaciones.escribircadena("cm\n");}
```

```
      else {comunicaciones.escribirchar('s');
           comunicaciones.escribirentero(1);
           comunicaciones.saltarlinea();}
  } else digitalWrite(PINLED,LOW);
  }
}
```

Notar que hay dos tareas a realizar:

- Comprobar si hay un obstáculo que impida su marcha, para lo cual se compara la distancia del sensor al obstáculo con la distancia de seguridad.
- Comprobar si se ha llegado al final del trayecto solicitado.

Si no ocurre ninguna de estas circunstancias el robot debe seguir moviéndose. Si ocurre alguna de estas circunstancias el robot se para e informa al programa de control.

En las pruebas se ha visto que el sensor de distancia consume mucho tiempo y eso ralentiza la marcha del robot. Se ha decidido que la consulta se haga cada 200 pasos, lo que equivale a una vuelta completa de las ruedas. La distancia de seguridad se deberá fijar en una cantidad superior a la distancia recorrida por el robot en una vuelta o no se podrá garantizar que el robot no choque con un obstáculo.

Ahora ya se puede ver los pasos que hay que dar para mover el robot:

- Se fija la posición del servo para indicar si va a ir en línea recta o va a girar.
- Se fija la distancia que se desea recorrer y la velocidad deseada en revoluciones por minuto.
- Se le da la orden al robot de moverse.
- En cualquier momento se le puede dar la orden de parar.
- Si el robot se para indicará la causa de su detención: obstáculo o ha llegado al final.
- Siempre se puede consultar la distancia al obstáculo y la distancia que queda por recorrer.

4) Lista de comandos

La lista de comandos se obtiene directamente de la función “interpretarorden” y es la siguiente:

Ordenes al robot:

- sn: Fijar consola, si n=1 modo consola, si n=0 modo programa de control:
- dn: Girar servo a la derecha, n es el número de grados a girar.
- in: Girar servo a la izquierda, n es el número de grados a girar.
- c0: Servo a posición central para avanzar en línea recta.
- xn: Fijar la distancia de seguridad, n es la distancia en centímetros.
- vn: Fija la velocidad del motor paso a paso en revoluciones por minuto (valor de n).
- ln: Distancia a recorrer en centímetros.
- g0: Orden para que el robot se mueva la distancia deseada.

- p0: Orden para que se pare el robot.

Petición de información:

- A0: Actualizar los valores de los sensores. No devuelve ningún valor.
- T0: Pedir la temperatura. El robot devuelve un entero que es la temperatura en grados centígrados.
- H0: Pedir la humedad. El robot devuelve un entero que es el porcentaje de humedad.
- L0: Pedir la luminosidad. El robot devuelve un entero que es la luminosidad en luxes.
- C0: Pedir CO_2. El robot devuelve un entero que es el porcentaje de CO_2 en partes por millón.
- D0: Pedir distancia al obstáculo. El robot devuelve un entero que es la distancia al obstáculo en centímetros.
- R0: Pedir distancia a destino. El robot devuelve un entero que es la distancia en centímetros que le separa de su objetivo y calculada a partir de los pasos restantes.

5) Programación

Módulo de comunicaciones

```
struct orden {
   char clave;
   int valor;
};

class TComunicaciones {
public:
   orden miorden;
   void inicializar() {Serial.begin(9600);};
   void escribirfloat(float f) {Serial.print(f);};
   void escribirentero(int i) {Serial.print(i);};
   void saltarlinea() {Serial.print("\n");};
   void escribircadena(String s) {Serial.print(s);};
   void escribirchar(char c) {Serial.print(c);};
   void obtenerorden();
   bool hayorden(void) {return Serial.available()>0;};
};

void TComunicaciones::obtenerorden() {
char cadena[10];
int i;

miorden.clave=Serial.read();
memset(cadena,0,sizeof(cadena));
delay(5);
i=0;
while(hayorden()) {
  cadena[i]=Serial.read();
```

```
 i++;
 delay(5);
}
miorden.valor=atoi(cadena);
}
```

Módulo de sensores

```
#include <DHT.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>
#include <Ultrasonic.h>

#define DHTTYPE DHT22
#define MGPIN 10 // Pin de lectura analogica
#define BOOLPIN  32  // Pin digital
#define DCGAIN 8.5 // Ganancia del amplificador
#define VOLTAJECERO 0.22  //Voltaje para una concentracion de 400PPM
#define INCREMENTOVOLTAJE 0.02 // Caida del voltaje para aire con 1000ppm
#define MUESTRAS 50 // Numero de muestras a tomar
#define INTERVALO 5 // Milisegundos para las muestras

#define PINTRIG 30
#define PINECHO 31
#define MAXDISTANCIA 30000 // Equivalente a 5 metros

const int DHTpin=22;  // Pin digital para la lectura
float CurvaCO2[3]={2.602,VOLTAJECERO,INCREMENTOVOLTAJE/(2.602-3)};

// Variables globales
DHT midht(DHTpin,DHTTYPE);
Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);  //
Luminosidad
Ultrasonic sensordistancia(PINTRIG,PINECHO,MAXDISTANCIA);

// Definicion de la clase

class TSensores {
public:
void inicializar();
void actualizarvalores(void);
float temperatura() {return t;};
float humedad() {return h;};
float luminosidad() {return l;};
int co2() {return c;};
int distancia  () { return sensordistancia.Ranging(CM);};
private:
float t;
float h;
float l;
int c;
```

```
int leerMG();
};


void TSensores::inicializar(void) {
midht.begin();
pinMode(DHTpin,INPUT);
digitalWrite(DHTpin,HIGH);
pinMode(BOOLPIN,INPUT);
digitalWrite(BOOLPIN,HIGH);
tsl.begin();
tsl.enableAutoRange(true);
tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);
}

void TSensores::actualizarvalores(void) {
sensors_event_t evento;

tsl.getEvent(&evento);
t=midht.readTemperature();
h=midht.readHumidity();
l=evento.light;
c=leerMG();
}


int TSensores::leerMG() {
int i;
int valor;
float v=0;

  for(i=0;i<INTERVALO;i++) {
    v+=analogRead(MGPIN);
    delay(MUESTRAS);
  }
  v=(v/INTERVALO)*5/1024;
 if((v/DCGAIN)>VOLTAJECERO) return 0;
 else valor=pow(10,((v/DCGAIN)-CurvaCO2[1])/CurvaCO2[2]+CurvaCO2[0]);
 if(valor<0) return 0;
 else return valor;
}
```

Módulo de motores

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"
#include <AccelStepper.h>
#include <Servo.h>

// Constantes para el motor
#define PASOS 200
```

```
#define RADIO 3.0   // En cm
#define REDUCTOR  3   // Relacion de reduccion

float distanciaporpaso=(2*3.14*RADIO)/(PASOS*REDUCTOR);


Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x60);
Adafruit_StepperMotor *miMotor = AFMS.getStepper(PASOS, 2);
Servo miservo;

void adelante() {
  miMotor->onestep(FORWARD,DOUBLE);
}

void atras() {
  miMotor->onestep(BACKWARD,DOUBLE);
}

AccelStepper motor(adelante,atras);

class TMotores {
public:
   void inicializar();
// Servo
   void derecha(int grados);
   void izquierda(int grados);
   void centro();
   int posicion() {return pos;};
// Motor paso a paso
   void velocidad(int v);
   int velocidad() {return vel;};
   void marchar() {r=true;};
   void parar() {r=false;};
   bool semueve() {return r;};
   bool unpaso();
   void distancia(int l);
   int distancia();
private:
   int pos;  // Posicion del srvo
   int vel;  // Velocidad en rpm
   long pas;  // Distancia inicial en pasos al objetivo
   bool r;    // Indicador de movimiento
};

void TMotores::inicializar() {
  Wire.begin();

  AFMS.begin();
  vel=0;
  motor.setSpeed(vel);  // rpm

  pos=90;
```

```
  miservo.attach(9); // Servo en pines servo 1 que se corresponden al pin 9. Si estuviera en serv
2 el pin seria el 10
  miservo.write(pos);
  r=false;
}


void TMotores::derecha(int grados) {
if((grados>0) && (grados<=50)) {
   pos=90+grados;
   miservo.write(pos);
}
}

void TMotores::izquierda(int grados) {
if((grados>0) && (grados<=50)) {
   pos=90-grados;
   miservo.write(pos);
}
}

void TMotores::centro() {
pos=90;
miservo.write(pos);
}

void TMotores::velocidad(int v) {
if ((v>=0) && (v<=1000)) {
   vel=v;
   motor.setSpeed(vel);
}
}

bool TMotores::unpaso() {
  if (r) return !motor.run();  // Solo un paso
   else return true;         // Esta parado
}


void TMotores::distancia(int l) {
int p;

p=round((float) l/distanciaporpaso);
motor.setCurrentPosition(0);
motor.move(p);
}

int TMotores::distancia() {
return round(((float)motor.distanceToGo())*distanciaporpaso);
}
```

Programa Principal

```
#include "sensores.h"
#include "comunicaciones.h"
#include "motores.h"
#define PINLED 41

TSensores sensores;
TComunicaciones comunicaciones;
TMotores motores;

int distanciaseguridad=30;  // Distancia de seguridad. Inicialmente vale 30cm
int contador;

bool consola=false; // Por omision no manda mensajes a consola, solo datos;

void tareas() {
  if (motores.unpaso())
 { // Si se esta moviendo y ha llegado al final, parar
   motores.parar();
   if (consola) comunicaciones.escribircadena("Final\n");
   else {comunicaciones.escribirchar('s');
        comunicaciones.escribirentero(0);
        comunicaciones.saltarlinea();};
 } else contador++;
 if(contador>=200) {
   contador=0;
   if (sensores.distancia()<distanciaseguridad) {
       digitalWrite(PINLED,HIGH);
       motores.parar();
       if (consola) {comunicaciones.escribircadena("Obstaculo en ");
                comunicaciones.escribirentero(sensores.distancia());
                comunicaciones.escribircadena("cm\n");}
       else {comunicaciones.escribirchar('s');
            comunicaciones.escribirentero(1);
            comunicaciones.saltarlinea();}
 } else digitalWrite(PINLED,LOW);
 }
}

void fijarconsola(int valor) {
 if (valor>0) consola=true;
  else consola=false;
 if(!consola) comunicaciones.saltarlinea();
}

void fijardistancia(int valor) {
 if(valor>0) distanciaseguridad=valor;
}

void mandartemperatura() {
if (consola) comunicaciones.escribircadena("Temperatura: ");
```

```
comunicaciones.escribirfloat(sensores.temperatura());
comunicaciones.saltarlinea();
}

void mandarhumedad() {
if (consola) comunicaciones.escribircadena("Humedad: ");
comunicaciones.escribirfloat(sensores.humedad());
comunicaciones.saltarlinea();
}

void mandarluminosidad() {
if (consola) comunicaciones.escribircadena("Luminosidad: ");
comunicaciones.escribirfloat(sensores.luminosidad());
comunicaciones.saltarlinea();
}

void mandarCO2() {
if (consola) comunicaciones.escribircadena("CO2: ");
comunicaciones.escribirentero(sensores.co2());
comunicaciones.saltarlinea();
}

void mandardistancia() {
if (consola)  comunicaciones.escribircadena("Distancia: ");
comunicaciones.escribirentero(sensores.distancia());
comunicaciones.saltarlinea();
}

void mandarlongitud() {
if (consola)  comunicaciones.escribircadena("Distancia a destino: ");
comunicaciones.escribirentero(motores.distancia());
comunicaciones.saltarlinea();
}

void mandaractualizacion() {
sensores.actualizarvalores();
if (consola)  comunicaciones.escribircadena("Valores actualizados.");
comunicaciones.saltarlinea();
}

void interpretarorden() {
  switch(comunicaciones.miorden.clave) {
  // Fija el modo consola
    case 's':fijarconsola(comunicaciones.miorden.valor);break;
  // Mandar datos sensores
    case 'A': mandaractualizacion();break;
    case 'T': mandartemperatura();break;
    case 'H': mandarhumedad();break;
    case 'L': mandarluminosidad();break;
    case 'C': mandarCO2();break;
    case 'D': mandardistancia();break;
    case 'R': mandarlongitud();break;
```

```
  // Recibir ordenes de giro
    case 'd':motores.derecha(comunicaciones.miorden.valor);break;
    case 'i':motores.izquierda(comunicaciones.miorden.valor);break;
    case 'c':motores.centro();break;
  // Recibir ordenes para movimiento
    case 'x':fijardistancia(comunicaciones.miorden.valor);break;
    case 'v':motores.velocidad(comunicaciones.miorden.valor);break;
    case 'l':motores.distancia(comunicaciones.miorden.valor);break;
    case 'g':motores.marchar();contador=200;break;
    case 'p':motores.parar();digitalWrite(PINLED,LOW);break;
  }
  if (consola)  comunicaciones.escribircadena("Esperando\n");
}

void setup() {
comunicaciones.inicializar();
sensores.inicializar();
motores.inicializar();
pinMode(PINLED,OUTPUT);
digitalWrite(PINLED,LOW);
}

void loop() {
if (comunicaciones.hayorden()) {
  comunicaciones.obtenerorden();
  interpretarorden();
}
if (motores.semueve()) {
  tareas();
}
}
```

ANEXO VI

PROGRAMA DE CONTROL

1) Diseño general del programa de control

Se ha diseñado un programa sencillo con una única ventana principal tal y como se muestra en la siguiente figura:

La ventana principal se encuentra dividida en tres secciones: Comunicaciones, Movimi ento, Sensores y el botón de Salir del programa en la parte inferior.

- La sección de Comunicaciones sirve para seleccionar el puerto serie al que está conectado el XBee y establecer la comunicación con el robot.
- La sección de Movimiento permite controlar la dirección, la distancia a recorrer, la velocidad y la distancia a la que un obstáculo hará que el robot pare. Un control giratorio (como si fuera un volante) que está asociado a un control numérico, permite indicar los grados de giro en la dirección del robot. El rango de giro va de -30 a +30 de manera que valores negativos indican giro a la izquierda y valores positivos giro a la derecha. Otro control numérico permite especificar la distancia a recorrer en centímetros, cuyos valores van desde -5000 a 5000, indicando los valores negativos que debe retroceder y los

positivos que debe avanzar. Otro control numérico permite fijar la velocidad en rpm y va de 0 a 1000. Finalmente, se puede cambiar la distancia de seguridad por medio de otro control numérico y los valores permitidos van de 1 a 100 (en cm). El botón "Mover" permite iniciar el movimiento del robot y cuando se está moviendo su texto cambia a “Parar” para indicar que si se pulsa se detiene la marcha del robot.

- La sección de sensores simplemente presenta un cuadro de texto donde se presentarán las medidas realizadas de las variables ambientales.

Hay dos particularizaciones que se han hecho en el fichero de proyectos. Para que la aplicación tenga un icono que la identifique se ha copiado el fichero “robot.ico” en la carpeta del proyecto y se ha añadido la siguiente línea en el fichero de proyecto:

```
RC_ICONS = robot.ico
```

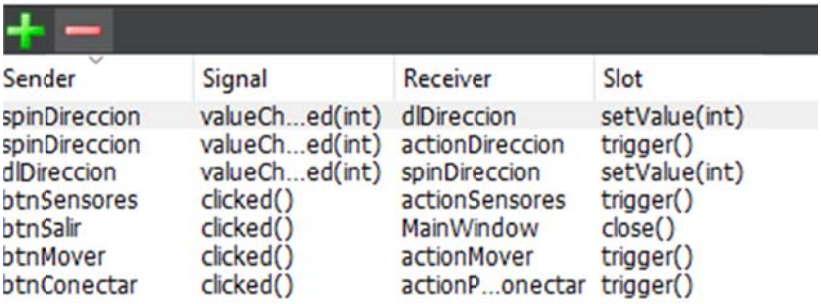
Por indicación de la documentación de Qt también se ha añadido la siguiente línea al fichero de proyectos.

```
QT       += serialport
```

Según la documentación de Qt, si no se añade esa línea el programa da error al compilar al tratar de utilizar las funciones relacionadas con el puerto serie.

2) Signals y slots del programa

Además de diseñar la ventana principal se han definido las asociaciones entre los diferentes eventos y las funciones particulares de la ventana principal. Este es el cuadro de asociaciones definido:



Sender	Signal	Receiver	Slot
spinDireccion	valueCh...ed(int)	dlDireccion	setValue(int)
spinDireccion	valueCh...ed(int)	actionDireccion	trigger()
dlDireccion	valueCh...ed(int)	spinDireccion	setValue(int)
btnSensores	clicked()	actionSensores	trigger()
btnSalir	clicked()	MainWindow	close()
btnMover	clicked()	actionMover	trigger()
btnConectar	clicked()	actionP...onectar	trigger()

Hay tres asociaciones automáticas y cuatro asociaciones a acciones. Estudiándolas por secciones se tiene lo siguiente:

- En la sección de comunicaciones hay una asociación que une el click del botón conectar (btnConectar) con una acción llamada “actionConectar”.
- En la sección de movimiento hay las siguientes asociaciones:

- Una asociación entre el control circular (dlDireccion) y el control numérico de dirección (spinDireccion) de manera que un cambio en el control circular implica un cambio en el control numérico.
- La asociación inversa a la anterior de manera que si se edita el valor del control numérico se refleja en el control circular.
- Una asociación entre el control numérico de dirección (spinDireccion) y una acción llamada "actionDireccion" de manera que también se ejecuta una función programada en la ventana principal, que será la encargada de transmitir las órdenes de dirección al robot.
- Una asociación entre el botón del movimiento (btnMover) y una acción llamada "actionMover".

- En la sección de sensores hay una única asociación entre el botón leer (btnSensores) y una acción llamada "actionSensores".
- Finalmente el botón de salir de la aplicación (btnSalir) tiene una asociación automática con la función "close()" de la ventana principal.

Se puede ver que la lista de acciones coincide con las acciones vistas en las asociaciones:

Y en el fichero mainwindow.h están definidas en la sección de slots:

```
private slots:
    void on_actionPuertoConectar_triggered();
    void on_actionDireccion_triggered();
    void on_actionMover_triggered();
    void on_actionSensores_triggered();
```

Destacar la manera en la que se coordinan los dos controles de dirección. Al girar el control circular se dispara un evento automático que cambia el valor del control numérico y ese cambio hace que se llame a la función correspondiente en la ventana principal. Si el cambio se hace en el control numérico entonces se dispara el evento automático que hace que cambie el valor del control circular y a la vez se lanza el evento que llama a la función de la ventana principal.

Ahora se va a proceder a explicar el programa sección por sección.

3) Sección de comunicaciones

La sección de comunicaciones tiene dos tareas fundamentales que realizar:

- Al inicializar la aplicación tiene que encontrar la lista de puertos serie del equipo para que se pueda elegir el puerto al que está conectado el dispositivo XBee.
- Al pulsar el botón de conexión tiene que comprobar que se puede comunicar con el robot.

Se han definido las siguientes variables: m_puerto, que guarda el puerto serie por el que se realiza la comunicación y una variable booleana llamada "m_conectado" para indicar si se ha establecido conexión con el robot.

En la función de inicialización de la ventana principal se pueden ver las siguientes instrucciones relacionadas con la comunicación:

```
buscarpuertos();
m_puerto=new QSerialPort(this);
m_conectado=false;
```

La función "buscarpuertos" es la que obtiene la lista de puertos y la carga en el control correspondiente. Su codificación es la siguiente:

```
void MainWindow::buscarpuertos() {
QSerialPortInfo ListaPuertos;
QList<QSerialPortInfo> lista=ListaPuertos.availablePorts();
QStringList ListaNombres;
for(int i=0;i<lista.count();i++)
    ListaNombres<<lista[i].portName();
ui->cbPuertos->clear();
ui->cbPuertos->addItems(ListaNombres);
}
```

Según la documentación de Qt se puede obtener la lista de puertos de un objeto de tipo QSerialPortInfo llamando a la función "availablePorts". Esta función devuelve una lista con información de cada uno de los puertos encontrados en el equipo. Recorriendo esa lista en el bucle "for" se añade el nombre a una lista de cadenas de caracteres (QStringList) y esta cadena se pasa directamente al control que mantiene la lista de puertos por pantalla. Notar cómo se utiliza la variable "ui" para acceder al control "cbPuertos".

Este código significa que al arrancar el programa ya se dispone de la lista de puertos disponibles, basta con elegir el que se corresponda con el dispositivo XBee y pulsar el botón de conectar para que se establezca la conexión. El código de la acción asociada al botón es el siguiente:

```
void MainWindow::on_actionPuertoConectar_triggered()
{
    m_puerto->close(); // Por si ya estaba abierto
    m_conectado=false;
    m_puerto->setPortName(ui->cbPuertos->currentText());

    if(m_puerto->open(QIODevice::ReadWrite)) {
      m_puerto->write("s0");
      m_puerto->waitForBytesWritten();
      if(m_puerto->waitForReadyRead()) {
       ui->lblCom->setText("Comunicación establecida con éxito.");
       m_puerto->clear();
       m_conectado=true;
```

```
        }
           else ui->lblCom->setText("ERROR: El robot no responde.");
    } else
        ui->lblCom->setText("ERROR: No se puede abrir el puerto.");
}
```

Por si se ha pulsado varias veces el botón lo primero que se hace es cerrar el puerto serie, a continuación se pasa el nombre del puerto seleccionado a la variable m_puerto y se intenta abrir el puerto para lectura y escritura. Si se produce un error la función “open” devuelve false y se muestra un mensaje de error. Una vez abierto el puerto se envía el comando “s0” para que el robot pase al modo de no consola. El robot debe responder con un carácter salto de línea. Hay dos funciones que se utilizan y que conviene explicar:

- La función waitForBytesWritten espera hasta que el puerto serie ha mandado el comando o un tiempo prefijado. Es necesario llamarla para dar tiempo al puerto a que complete la transmisión.
- La función waitForReadyRead espera hasta que hay algo para leer en el puerto serie o un tiempo determinado. Si hay algo para leer devuelve true, si se ha consumido el tiempo y no ha llegado nada entonces devuelve false.

Como el robot debe responder al comando, si no se recibe contestación es que el robot está apagado o fuera de alcance. Si llega contestación quiere decir que el robot ha respondido y se ha establecido la comunicación. Como no interesa el carácter leído se descarta con la función clear(). La variable m_conectado se pone a true para indicar que hay comunicación con el robot.

4) Sección de movimiento

La sección de movimiento tiene dos acciones a analizar: control de la dirección y el botón de mover/parar. Se ha definido una variable booleana llamada m_semueve para guardar el estado de movimiento del robot.

La acción para controlar la dirección se ha codificado de la siguiente manera:

```
void MainWindow::on_actionDireccion_triggered()
{
    QByteArray comando;

    if(!m_conectado) return;
    if (ui->spinDireccion->value()>0) {
        comando.append("d");
        comando.append(QString::number(ui->spinDireccion->value()));
    }
    else if(ui->spinDireccion->value()<0) {
        comando.append("i");
        comando.append(QString::number(-ui->spinDireccion-
>value()));
    } else comando.append("c0");
    m_puerto->write(comando);
    m_puerto->waitForBytesWritten();
```

```
}
```

Algo común a todas las funciones que se van a analizar es que si la conexión no se ha producido o ha fallado se sale de la función sin hacer nada.

El código es bastante sencillo. Dependiendo del signo del valor del control se selecciona el comando de giro a derecha o a izquierda. El número se pasa a caracteres con la función QString::number y se envía el comando. Notar que si el valor del control es cero se envía el comando de colocar la dirección en el centro. Siempre que hay que mandar un comando con un dato numérico se opera de la misma manera:

- Se define una variable de tipo QByteArray, que es una tabla de bytes.
- Con la función append() se añade la información, primero el carácter y luego el número.
- El número se debe transformar en una serie de caracteres y eso se hace con la función QString::number, que recibe como parámetro un número y devuelve la representación textual de dicho número.

La función asociada al botón de Mover/Parar al robot es la siguiente:

```
void MainWindow::on_actionMover_triggered()
{
    if(!m_conectado) return;
    if(m_semueve) {
        m_puerto->write("p0");
        m_puerto->waitForBytesWritten();
        m_semueve=false;
        ui->btnMover->setText("Mover");
    } else {
        m_semueve=true;
        ui->btnMover->setText("Parar");
        moverrobot();
        QTimer::singleShot(500,this,SLOT(funciontimer()));
    }
}
```

Si el robot se está moviendo entonces la variable m_semueve vale true y entonces hay que pararlo enviando el comando “p0” y además hay que dar el valor false a m_semueve y cambiar el texto del botón a “Mover”.

Si el motor no se está moviendo entonces m_semueve vale false y se ejecuta la parte del else. Se actualiza la variable m_semueve a true, se cambia el texto del botón a “Parar” y se llama a la función moverrobot() que se encarga de todas las tareas que hay que hacer para poner en marcha el robot. El significado de la función QTimer::singleShot() se explica más adelante en la sección de control del robot mientras se mueve.

La función moverrobot() hace lo siguiente:

```
void MainWindow::moverrobot() {
QByteArray comando;

// Primero pasa la distancia de seguridad.
comando.clear();
comando.append("x");
```

```
comando.append(QString::number(ui->spinSeguridad->value()));
m_puerto->write(comando);
m_puerto->waitForBytesWritten();
// A continuacion pasa la distancia a mover.
comando.clear();
comando.append("l");
comando.append(QString::number(ui->spinDistancia->value()));
m_puerto->write(comando);
m_puerto->waitForBytesWritten();
// A continuacion pasa la velocidad
comando.clear();
comando.append("v");
comando.append(QString::number(ui->spinVelocidad->value()));
m_puerto->write(comando);
m_puerto->waitForBytesWritten();
// A continuacion ordena moverse al robot
m_puerto->write("g0");
m_puerto->waitForBytesWritten();
}
```

Para evitar el envío de demasiados comandos al robot se ha decidido que se seleccionen por pantalla todas las condiciones del movimiento (distancia, velocidad y distancia de seguridad) y se pasen al robot solamente en el momento en el que se pulsa el botón mover. Como se puede ver la función es simplemente una sucesión de comandos que se envían al robot. Primero se establecen las condiciones (distancia de seguridad, distancia a recorrer y velocidad) y luego se envía el comando “g0” para que el robot se mueva.

5) Sección de sensores

En la sección de sensores hay que analizar dos cuestiones: cómo se muestra la información por pantalla y la acción asociada al botón de actualización de la información.

El control que presenta la información es del tipo QListView, que se utiliza para visualizar cadenas de caracteres. La manera que tiene Qt de manejar este tipo de controles es siguiendo un modelo conocido como *vista/controlador* de manera que la información que aparece en la vista (el control de la ventana) es manejada por otro objeto llamado controlador.

Para el objeto QListView se debe definir un objeto de tipo QStringListModel y asociarlo. Como no es un control visual hay que hacer todo por código. La manera de hacerlo es la siguiente:

- Se declara en el fichero de encabezamiento dentro de la definición de la ventana principal.

```
QStringListModel m_modelo;
```

- En la inicialización de la ventana se asocia modelo con vista y se activa la visualización de los datos del modelo usando la función show.

```
ui->lvSensores->setModel(&m_modelo);
ui->lvSensores->show();
```

- Cuando se desee cambiar lo que se representa en pantalla hay que usar la función setStringList para pasarle una lista de cadenas de caracteres y el cambio se refleja automáticamente en el control visual de la pantalla.

Aunque en este caso tan simple este modelo complica la programación lo cierto es que en la mayoría de los casos es un modelo que ahorra mucho esfuerzo en programación.

La información de los sensores se guardará en forma de lista de cadenas de caracteres en una variable que se declara en la definición de la ventana principal.

```
QStringList m_sensores;
```

La acción de leer los sensores la ejecuta la siguiente función:

```
void MainWindow::on_actionSensores_triggered()
{

    if(!m_conectado) return;
    // Actualizar valores
    m_puerto->write("A0");
    m_puerto->waitForBytesWritten();
    m_puerto->waitForReadyRead(); // Espera que llegue la
contestacion
    m_sensores.clear();
    m_puerto->clear();
    // Temperatura
    m_sensores<<("Temperatura: "+leerunsensor("T0")+"ºC");
    // Humedad
    m_sensores<<("Humedad: "+leerunsensor("H0")+"%");
    // Luminosidad
    m_sensores<<("Luminosidad: "+leerunsensor("L0")+" luxes");
    // CO2
    m_sensores<<("CO2: "+leerunsensor("C0")+" ppm");
    // Resultados por pantalla
    m_modelo.setStringList(m_sensores);
}
```

Primero se envía el comando “A0” para que el robot lea los sensores. Como es una función que consume tiempo se utiliza la función waitForReadyRead() para esperar hasta que el robot responda, lo que indica que ha terminado de actualizar los valores de los sensores. Luego se le van pidiendo los valores. La lectura se realiza a través de la función “leerunsensor”. Por ejemplo:

```
m_sensores<<("Temperatura: "+leerunsensor("T0")+"ºC");
```

Esta instrucción añade una tira de caracteres a m_sensores, esta tira es la concatenación de la cadena “Temperatura: “ con la cadena de caracteres que devuelve la función leerunsensor y la cadena constante “ºC”. La función leerunsensor es una función que se ha codificado de la siguiente manera:

```
QString MainWindow::leerunsensor(QByteArray comando) {
    char datos[100];
    QString entrada;
    int i;
```

```
    memset(datos,0,sizeof(datos));
    m_puerto->write(comando);
    m_puerto->waitForBytesWritten();
    for(i=0,m_puerto->waitForReadyRead(100), m_puerto-
>read(&datos[i],1);
        (i<100) && (datos[i]!='\n');
        i++,m_puerto->waitForReadyRead(100),m_puerto-
>read(&datos[i],1));
    if(i<100) datos[i]='\0';
    entrada.append(datos);
    return entrada;
}
```

La variable “datos” va a recibir la entrada y se inicializan todos sus bytes a cero para que el resto de funciones no encuentren valores aleatorios. Primero se envía el comando y luego se lee la entrada carácter a carácter hasta que se encuentra el carácter salto de línea. Es necesario hacerlo así por la diferencia de velocidades entre el programa que se está ejecutando y las comunicaciones. La clave de la rutina está en el bucle for:

- En la inicialización el índice “i” se pone a cero, se espera hasta que haya algún carácter disponible y se lee almacenándolo en el variable “datos”.
- La condición siempre garantiza que no se reciben más de 100 caracteres y la condición de salida, ya que cuando se recibe el carácter fin de línea quiere decir que se ha acabado la transmisión.
- En el bucle se incrementa el índice “i”, se comprueba si hay otro carácter esperando a ser leído y se almacena en “datos”.

Una vez leídos todos los datos transmitidos, incluyendo el carácter fin de línea, se sustituye dicho carácter por un cero ya que no interesa que ese carácter se incluya en la cadena de caracteres de retorno ya que provocaría un salto de línea no deseado en la lista de valores.

Una vez obtenidos todos los datos se copia la lista de bytes leídos en una variable de tipo cadena de caracteres (variable “entrada” de tipo QStringList) y se devuelve el valor.

Notar que si el programa simplemente almacenara la información en la variable m_sensores ésta no se mostraría nunca por pantalla. Una vez recopilada toda la información hay que pasarla al controlador para que se muestre por pantalla. Ese es el propósito de la última instrucción de la función que implementa la acción de actualizar el controlador.

```
m_modelo.setStringList(m_sensores);
```

6) Control del robot mientras se mueve

La última parte del programa de control consiste en supervisar el movimiento del robot. Todas las partes del programa hasta este momento se han limitado a dar órdenes al robot y esperar alguna respuesta inmediata en caso necesario (lectura de un sensor, por ejemplo). Esta parte es radicalmente diferente porque *es el robot el que va a tener la iniciativa* y hay que detectar

cuándo el robot envía un comando indicando que se ha parado porque ha llegado al final o porque ha encontrado un obstáculo.

La mejor manera de implementar este control es lanzar una función que se ejecute periódicamente para lo cual se ha utilizado la función QTimer::singleShot(). La instrucción empleada es la siguiente:

```
QTimer::singleShot(500,this,SLOT(funciontimer()));
```

Esta función hace que después de 500 milisegundos (medio segundo) se llame a la función "funciontimer" de la ventana principal. El programa se sigue ejecutando normalmente y cuando han pasado 500 milisegundos se ejecuta esta función, por lo que el programa no se queda "congelado", de hecho se podría pulsar el botón de parar el robot antes de que se ejecutara dicha función. La función QTimer::singleShot() es de un "único disparo", es decir, si se quiere volver a ejecutar la función después de 500 milisegundos hay que volver a lanzar la función. La función "funciontimer()" se debe declarar en la sección de slots. La declaración de todas las funciones de slots en la zona de encabezamiento queda así:

```
private slots:
     void on_actionPuertoConectar_triggered();

     void on_actionDireccion_triggered();

     void on_actionMover_triggered();

     void on_actionSensores_triggered();

    void funciontimer();
```

Se puede comprobar que están declaradas todas las funciones definidas automáticamente al definir las acciones y esta nueva función que se ha declarado para la función QTimer::singleShot(). La función "funciontimer()" es la siguiente:

```
void MainWindow::funciontimer() {
    char datos[100];
    QString texto;
    int i;

    if(m_puerto->bytesAvailable()>0) { //Ha llegado un comando del
robot
        i=0;
        m_puerto->read(&datos[i],1);
        while(datos[i]!='\n') {
            m_puerto->waitForReadyRead(100);
            i++;
            m_puerto->read(&datos[i],1);
        }
        if(datos[0]=='s') {   // Ha llegado un comando de parada
            ui->btnMover->setText("Mover");
            m_semueve=false;
            if(datos[1]=='0') ui->lblDistFalta->setText("Trayecto
finalizado");
            else ui->lblDistFalta->setText("Obstaculo detectado");
         }
    };
```

```
    if(m_semueve) { // El robot se sigue moviendo y se pide la
  distancia al destino
        texto="Distancia a destino: "+leerunsensor("R0")+"cm.";
        ui->lblDistFalta->setText(texto);
        QTimer::singleShot(500,this,SLOT(funciontimer()));
        }
  }
```

Lo primero que hace es comprobar si el robot ha enviado un comando usando la función "bytesAvailable()" del puerto serie, que devuelve el número de bytes que están esperando para ser leídos. Si es así lee el comando byte a byte y lo almacena en la variable "datos". Mientras no llegue el carácter de fin de línea no se ha terminado el comando y se siguen leyendo bytes. Se incluye la función waitForReadyRead() por si el comando estaba llegando a la vez que se ejecutaba esta función. Notar que si el comando ya había llegado esta función no introduce ningún retardo.

Una vez leído el comando se comprueba si empieza por "s", lo que quiere decir que el robot se ha detenido, luego hay que cambiar el texto del botón "Parar" a "Mover" y hay que actualizar la variable m_semueve a false. A partir de ahí se examina si el comando es "s0" o "s1" para saber si se ha parado por encontrar un obstáculo o porque ha llegado al final del trayecto.

Una vez comprobado si el robot se ha parado se pasa al segundo "if" de la función. Si el robot sigue moviéndose se actualiza el mensaje que sale por pantalla para indicar la distancia que falta al destino. Por último se vuelve a utilizar la función QTimer::singleShot() para que la función se vuelva a ejecutar en 500 milisegundos.

7) Programa

Fichero del Proyecto robot.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2017-03-30T11:55:19
#
#-------------------------------------------------

QT       += core gui
QT       += serialport

RC_ICONS = robot.ico

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = robot
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which as been marked as deprecated (the exact
warnings
```

```
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated
APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain
version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

Fichero main.cppl

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Fichero mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QStringListModel>
#include <QTimer>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
     void on_actionPuertoConectar_triggered();
```

```
    void on_actionDireccion_triggered();

    void on_actionMover_triggered();

    void on_actionSensores_triggered();

    void funciontimer();

private:
    Ui::MainWindow *ui;
    QSerialPort *m_puerto;
    bool m_conectado;
    bool m_semueve;
    QStringList m_sensores;
    QStringListModel m_modelo;
    void buscarpuertos();
    QString leerunsensor(QByteArray comando);
    void moverrobot();
};

#endif // MAINWINDOW_H
```

Fichero mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QList>
#include <QtSerialPort/QSerialPortInfo>
#include <QByteArray>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    buscarpuertos();
    m_puerto=new QSerialPort(this);
    m_conectado=false;
    m_semueve=false;
    m_sensores.clear();
    m_modelo.setStringList(m_sensores);
    ui->lvSensores->setModel(&m_modelo);
    ui->lvSensores->show();
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::buscarpuertos() {
QSerialPortInfo ListaPuertos;
QList<QSerialPortInfo> lista=ListaPuertos.availablePorts();
QStringList ListaNombres;
for(int i=0;i<lista.count();i++)
    ListaNombres<<lista[i].portName();
ui->cbPuertos->clear();
ui->cbPuertos->addItems(ListaNombres);
}

void MainWindow::on_actionPuertoConectar_triggered()
```

```
{
    m_puerto->close(); // Por si ya estaba abierto
    m_conectado=false;
    m_puerto->setPortName(ui->cbPuertos->currentText());

    if(m_puerto->open(QIODevice::ReadWrite)) {
      m_puerto->write("s0");
      m_puerto->waitForBytesWritten();
      if(m_puerto->waitForReadyRead()) {
       ui->lblCom->setText("Comunicación establecida con éxito.");
       m_puerto->clear();
       m_conectado=true;
      }
         else ui->lblCom->setText("ERROR: El robot no responde.");
    } else
        ui->lblCom->setText("ERROR: No se puede abrir el puerto.");
}

void MainWindow::on_actionDireccion_triggered()
{
    QByteArray comando;

    if(!m_conectado) return;
    if (ui->spinDireccion->value()>0) {
        comando.append("d");
        comando.append(QString::number(ui->spinDireccion->value()));
    }
    else if(ui->spinDireccion->value()<0) {
        comando.append("i");
        comando.append(QString::number(-ui->spinDireccion->value()));
    } else comando.append("c0");
    m_puerto->write(comando);
    m_puerto->waitForBytesWritten();
}


void MainWindow::on_actionMover_triggered()
{
    if(!m_conectado) return;
    if(m_semueve) {
        m_puerto->write("p0");
        m_puerto->waitForBytesWritten();
        m_semueve=false;
        ui->btnMover->setText("Mover");
    } else {
        m_semueve=true;
        ui->btnMover->setText("Parar");
        moverrobot();
        QTimer::singleShot(500,this,SLOT(funciontimer()));
    }
}

void MainWindow::moverrobot() {
QByteArray comando;

// Primero pasa la distancia de seguridad.
comando.clear();
comando.append("x");
comando.append(QString::number(ui->spinSeguridad->value()));
m_puerto->write(comando);
m_puerto->waitForBytesWritten();
```

```
// A continuacion pasa la distancia a mover.
comando.clear();
comando.append("l");
comando.append(QString::number(ui->spinDistancia->value()));
m_puerto->write(comando);
m_puerto->waitForBytesWritten();
// A continuacion pasa la velocidad
comando.clear();
comando.append("v");
comando.append(QString::number(ui->spinVelocidad->value()));
m_puerto->write(comando);
m_puerto->waitForBytesWritten();
// A continuacion ordena moverse al robot
m_puerto->write("g0");
m_puerto->waitForBytesWritten();
}

void MainWindow::on_actionSensores_triggered()
{

    if(!m_conectado) return;
    // Actualizar valores
    m_puerto->write("A0");
    m_puerto->waitForBytesWritten();
    m_puerto->waitForReadyRead(); // Espera que llegue la contestacion
    m_sensores.clear();
    m_puerto->clear();
    // Temperatura
    m_sensores<<("Temperatura: "+leerunsensor("T0")+"ºC");
    // Humedad
    m_sensores<<("Humedad: "+leerunsensor("H0")+"%");
    // Luminosidad
    m_sensores<<("Luminosidad: "+leerunsensor("L0")+" luxes");
    // CO2
    m_sensores<<("CO2: "+leerunsensor("C0")+" ppm");
    // Resultados por pantalla
    m_modelo.setStringList(m_sensores);
}

QString MainWindow::leerunsensor(QByteArray comando) {
    char datos[100];
    QString entrada;
    int i;

    memset(datos,0,sizeof(datos));
    m_puerto->write(comando);
    m_puerto->waitForBytesWritten();
    for(i=0,m_puerto->waitForReadyRead(100), m_puerto-
>read(&datos[i],1);
        (i<100) && (datos[i]!='\n');
        i++,m_puerto->waitForReadyRead(100),m_puerto-
>read(&datos[i],1));
    if(i<100) datos[i]='\0';
    entrada.append(datos);
    return entrada;
}

void MainWindow::funciontimer() {
    char datos[100];
    QString texto;
    int i;
```

```
    if(m_puerto->bytesAvailable()>0) { //Ha llegado un comando del
robot
        i=0;
        m_puerto->read(&datos[i],1);
        while(datos[i]!='\n') {
            m_puerto->waitForReadyRead(100);
            i++;
            m_puerto->read(&datos[i],1);
        }
        if(datos[0]=='s') {   // Ha llegado un comando de parada
            ui->btnMover->setText("Mover");
            m_semueve=false;
            if(datos[1]=='0') ui->lblDistFalta->setText("Trayecto
finalizado");
            else ui->lblDistFalta->setText("Obstaculo detectado");
         }
    };
   if(m_semueve) { // El robot se sigue moviendo y se pide la
distancia al destino
       texto="Distancia a destino: "+leerunsensor("R0")+"cm.";
       ui->lblDistFalta->setText(texto);
       QTimer::singleShot(500,this,SLOT(funciontimer()));
   }
```

ANEXO VII

PRESUPUESTO

Los componentes seleccionados para la construcción de un prototipo del robot diseñado han sido encargados a las siguientes compañías: Adafruit Industries, PC Componentes, Farnell element 14, Amazon, Mouser, DF Robot y Hobbymodelismo. Estas compañías han sido seleccionadas de acuerdo a la disponibilidad del producto y precio ofertado.

A continuación se detallan los precios de cada uno de los componentes. En cada caso se indican el incremento del coste total por tasas e impuestos y los gastos de envío.

Adafruit Industries

Componente	Precio (€)
Motor paso a paso Nema 17	13.12
Sensor TSL2561 Luminosidad	5.58
Sensor DHT22 Temperatura y Humedad	14.06
Interruptor Powerboost	0.89
Motor servo Towerpro SG-5010	11.25
Batería Litio 3.7 V	27.65
Powerboost 1000C	18.7
Gastos de envío e impuestos	42.14
TOTAL	**133**

PC Componentes

Componente	Precio (€)
Arduino Mega 2560 R3	14.01
Sensor HC-SR04 Ultrasonido	6.57
Cable USB-USB	1.45
Gastos de envío e impuestos	9.95
TOTAL	**31.98**

Farnell element 14

Componente	Precio (€)
Xbee Shield	15.9
Gastos de envío e impuestos	14.23
TOTAL	**30.13**

Amazon

Componente	Precio (€)
Adafruit Motor Shield V2.3	20.62
Gastos de envío e impuestos	4.33
TOTAL	**24.95**

Mouser

Componente	Precio (€)
Xbee S2 (x2)	33.54
Adaptador Xbee-USB	26.4
Gastos de envío e impuesto	0
TOTAL	**59.94**

DF Robot

Componente	Precio (€)
Sensor CO2	52.49
Gastos de envío e impuestos	15
TOTAL	**67.49**

Hobbymodelismo

Componente	Precio (€)
Ruedas delanteras	9.95
Ruedas traseras	9.95
Gastos de envío e impuestos	0
TOTAL	**19.9**

Importe Total

Adafruit Industries	133 €
PC Componentes	31.98 €
Farnell element 14	30.13
Amazon	24.95
Mouser	59.94
DF Robot	67.49
Hobbymodelismo	19.9
TOTAL	**367.39 €**