

Capítulo 7

Sistema de alimentación y carga

7.1. Introducción

En este capítulo se desarrollan las mejoras que se han realizado en el sistema de alimentación del robot, sustituyendo la batería de plomo por un sistema de baterías más eficiente. También se realiza un estudio de los distintos tipos de conectores que se utilizan en sistemas de recarga comerciales para robots similares. Finalmente se decide el sistema de conectores que se va a implementar y se realiza el diseño apropiado para la estación de carga, teniendo en cuenta tanto los conectores como la electrónica que alberga en su interior.

7.2. Sistema de alimentación

Como se ha descrito en el capítulo 3 de este documento, el robot original se alimenta con una batería de plomo de 12 voltios. Esta batería tiene numerosos inconvenientes en comparación con otros sistemas de alimentación más modernos. Por lo tanto, para hacer funcionar al robot existen diferentes opciones de sistemas de alimentación con mayores ventajas. A continuación se describen los principales tipos de baterías que se usan en aplicaciones de robótica, destacando sus ventajas e inconvenientes según el tipo de aplicación al que se destinan:

- Plomo-ácido: Es el tipo de batería que el robot original incorporaba y son las utilizadas en los vehículos comunes. Están formadas por un depósito

de ácido sulfúrico y unas placas de plomo y ofrecen voltajes de salida de 6 o 12 voltios. Son fiables y económicas, pero muy pesadas y contaminantes.

- Ni-Cd: Las baterías de Níquel Cadmio tienen rendimientos superiores a las de plomo, ofreciendo mayor autonomía y menor peso. Entre sus inconvenientes destacan su precio mayor, el efecto memoria y su alto poder contaminante. Por ello, están en desuso.
- Ni-MH: Compuestas por hidruros metálicos en lugar de cadmio, tienen mayor capacidad y una vida útil más larga que las anteriores. El efecto memoria es menor, pero son más caras y más sensibles al calor.
- Li-Ion: Las baterías de Ion Litio tienen una alta densidad de energía y apenas sufren el efecto memoria. Entre sus desventajas destacan su precio y su mal comportamiento a los cambios de temperaturas o cuando se descargan completamente. Aún así, son muy utilizadas por su reducido tamaño y peso en todo tipo de dispositivos móviles.
- LiPo: Las nuevas baterías de litio polímero son aún más ligeras que las anteriores y son muy utilizadas en aplicaciones de radiocontrol. Existen distintos tipos con capacidades, densidades de energía y costes distintos según los materiales usados en su construcción. Uno de sus inconvenientes es la necesidad de equilibrar las tensiones de cada elemento para que funcionen correctamente.

Así pues, para elegir el tipo de batería hay que tener en consideración numerosas características técnicas, así como también otros parámetros como su precio o su disponibilidad. Entre las ventajas de la batería de plomo-ácido, destacan su fiabilidad y su bajo precio. Sin embargo, su volumen y peso provocan que no sea el tipo de batería más adecuado para robótica móvil. En este caso, tal y como se puede observar en la figura 7.1 donde se muestra la batería sobre una báscula, tiene un peso de 1928 gramos.

Comparando las características de esta batería de plomo con el resto de tipos de baterías, se decide que es necesario sustituir este tipo de batería por una de mejores prestaciones y menor tamaño y peso, con el objetivo de mejorar el comportamiento general del robot y aumentar su autonomía. Además permite la posibilidad de aprovechar el espacio más eficientemente y diseñar un robot más compacto.

Entre las distintas opciones, se opta por baterías de tipo Ion-Litio. Son mucho más ligeras y pequeñas en comparación con la batería de plomo y proporcionan una gran densidad de energía. Su precio es más elevado y no ofrece buen comportamiento a los cambios de temperatura, pero para el desarrollo de este proyecto, este inconveniente no provoca una gran limitación. Otro de sus grandes inconvenientes es el mal funcionamiento cuando su carga desciende de



Figura 7.1: Batería original de plomo-ácido.

un determinado nivel, pero al disponer de una estación de carga disponible en cualquier momento, para esta aplicación tampoco supone una gran desventaja.

Para el uso de este tipo de batería se hace necesaria la utilización de circuitos electrónicos para controlar y regular tanto la carga como la descarga de sus diferentes células. Se ha diseñado un módulo para albergar 6 baterías individuales con las características mostradas en la tabla 7.1.

Marca	UltraFire
Modelo	TR 18650 Li-Ion
Capacidad	5200 mAh
Tensión	3.7 V
Peso	35 g

Tabla 7.1: Características de las baterías de Ion-Litio

Se instalarán 3 células en serie con 2 baterías de 3.7 V en paralelo cada una. Esto proporciona una tensión nominal de $3 \cdot 3.7 = 11.1$ Voltios, suficiente para que la electrónica de control funcione correctamente y proporcione la tensión de salida de 12 voltios necesaria.

El circuito de protección de la batería elegido es el RS-03 V1.0, mostrado en la figura 7.2 y se conecta según el diagrama mostrado en la figura 7.3. El módulo contiene un chip Seiko S-8254A con 6 MOS que soportan grandes intensidades y está diseñado para funcionar con 3 baterías en serie de tensiones de 3,6 V, 3,7



Figura 7.2: Circuito de protección RS-03 V1.0.

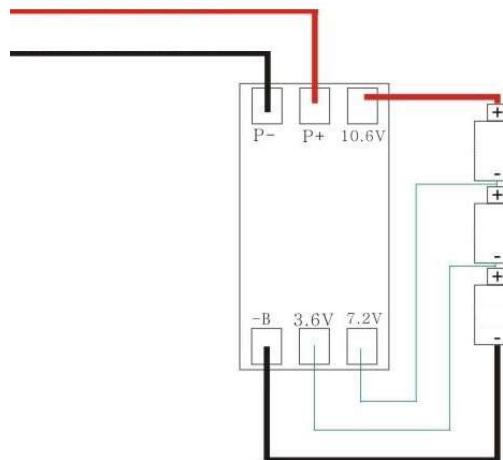


Figura 7.3: Esquema del montaje del circuito de protección RS-03 V1.0. *Fuente:* [20]

V, 4 V o 4,2 V dando lugar a tensiones de 10,8 V, 11,1V, 12 V o 12,6 V. Es importante que las baterías utilizadas tengan las mismas tensiones nominales y los mismos comportamientos de carga y descarga. Los valores de protección individual para cada célula son 2,80 V para su valor mínimo y 4,25 V para su carga máxima. El circuito de protección ofrece protección de sobrecarga y protección de carga mínima, desconectando la alimentación cuando la tensión de las baterías baja de su valor mínimo de carga. También ofrece protección para limitar la corriente máxima en 28 A en su valor pico y 8 A en su funcionamiento normal. Igualmente protege las baterías en el caso de que se produzca un cortocircuito en los terminales de salida del módulo.

Una desventaja de este sistema es la necesidad de usar el circuito de protección y tener que conectar y montar el soporte manualmente en lugar de usar una batería a la que sólo hay que conectar los bornes de salida. Sin embargo, como ya se ha explicado, se obtiene una reducción de volumen y peso muy considerable, pasando casi de 2 Kg de la batería de plomo a apenas 365 g del módulo completo, con el soporte, la electrónica de control y las baterías, tal y como se puede observar en la figura 7.4. El módulo completo fijado en la parte inferior de la plataforma se muestra en la figura 7.5. En la figura 7.6 se muestra el resultado de todas las conexiones eléctricas debajo de la plataforma del robot. Además de la instalación de la nueva batería, se observa la fijación mediante tornillos tirafondos a la base de madera de la nueva tarjeta Arduino y una pequeña placa de prototipado que conecta a todos los receptores con el Arduino. También se han instalado dos conectores externos en el lateral de la plataforma para conectar los cables USB del ordenador sin tener que acceder a la zona inferior de la plataforma. En su interior, los cables se han instalado usando fijadores y bridas de plástico.



Figura 7.4: Módulo de las baterías con el circuito de protección.

Además, aunque la tensión estará controlada por el software diseñado, se ha instalado un voltímetro que incorpora una pequeña pantalla que se ha instalado en un lateral del robot, tal y como se puede observar en la figura 7.7. Esta pantalla sirve para comprobar de un modo muy sencillo la tensión de la batería, así como la tensión del sistema de alimentación cuando se encuentra durante el proceso de recarga.



Figura 7.5: Módulo de las baterías instalado en la plataforma.

7.3. Estación de carga

Para el diseño de la estación de carga hay que tener en cuenta numerosos aspectos. Por un lado, el sistema debe albergar la electrónica que controla la emisión de las señales, así como los propios emisores. Y por otro lado, hay que diseñar un sistema de conectores que sea capaz de alimentar a las baterías del robot de forma fiable. Además, incorpora una conexión y un interruptor de encendido. Otras de las características importantes es que la estación sea lo suficientemente robusta para soportar la fuerza que el robot pueda ejercer sobre esta. Y finalmente, hay que tener en cuenta si las piezas de dicha estación se van a realizar también con la impresora 3D o usando algún otro tipo de material.

En principio se estudió utilizar una caja de plástico de tipo estándar de las que se usan para proteger circuitos electrónicos. Este tipo de caja es muy resistente y robusta, se puede mecanizar con facilidad y las hay disponibles de muchos tamaños. Sin embargo, una vez estudiados los distintos tipos de

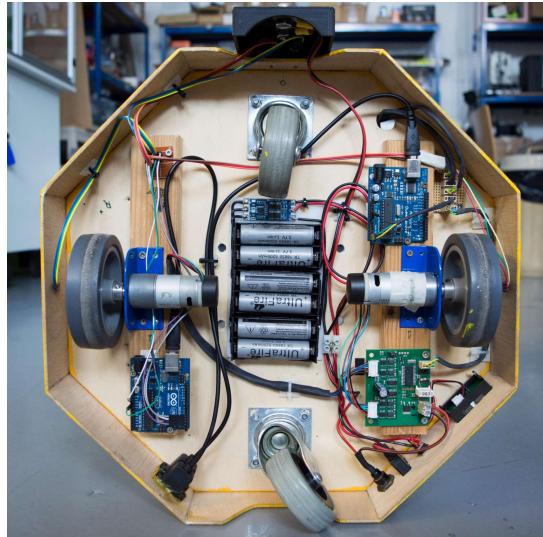


Figura 7.6: Vista inferior de la plataforma.



Figura 7.7: Voltímetro instalado en un lateral de la plataforma.

conexiones que se podían implementar, se descartó esta idea, ya que limita mucho el diseño de la estación.

Entre los diferentes tipos de conexiones posibles, se estudiaron los siguientes:

- **Conexiones frontales** Se trata de un sistema en el cuál la base dispone de dos bandas metálicas horizontales conectadas a la corriente eléctrica. En el robot móvil se instalan dos pequeños conectores a la altura de las dos bandas metálicas que cuando se acerca frontalmente a la base realizan el contacto para cargar las baterías. En la figura 7.8 se muestra un ejemplo

de este tipo de conexiones.



Figura 7.8: Robot aspirador con conexiones frontales. *Fuente:* [4]

- **Conexiones bajo la plataforma** Este sistema también dispone de dos conectores en la base y otros dos en el robot, pero la conexión no se realiza de forma frontal. En este caso los contactos del robot se sitúan en la parte baja de la plataforma y encajan justo encima de los conectores de la base, que están situados casi a la altura del suelo. Este sistema, aunque es muy similar al anterior, tiene varias ventajas. Los contactos en el robot apenas son visibles, por lo que es más difícil que se provoque un contacto indeseado. Además la estación de carga dispone de una base mayor que le proporciona más estabilidad y también sirve de guía ajustando el movimiento del robot mientras se produce el contacto. Por otro lado, su principal inconveniente es que la base es menos compacta y más difícil de fabricar. El ejemplo más conocido es la estación de carga del robot Roomba de iRobot mostrada en la figura 7.9.



Figura 7.9: Estación de carga del Roomba con conexiones bajo la plataforma. *Fuente:* [19]

- Conexión mediante conector. Este sistema se basa en diseñar un conector en forma de clavija similar al que se puede enchufar de forma manual, ya sean de corriente alterna o corriente continua y que incorpora en su interior 2 o más contactos eléctricos. Por ello, las conexiones del propio conector quedan más protegidas de contactos no deseados, pero la precisión de la conexión, así como la fuerza necesaria para realizar la conexión son sus principales inconvenientes. Se muestra un ejemplo de este tipo de conector en la figura 7.10.



Figura 7.10: Estación de carga de robot Freight de Fetch Robotics con conector frontal. *Fuente:* [10]

A pesar de que el segundo sistema estudiado tiene algunas ventajas interesantes, se decide realizar la estación de carga con conexiones frontales horizontales. Por un lado, el diseño y la fabricación de la base con una impresora 3D es más sencilla a priori. Por otro lado, este sistema es más flexible y permite realizar modificaciones para adaptarlo a otros robots, por ejemplo, con distintas alturas. En contra, al realizar el contacto de forma frontal, es más difícil realizarlo de forma estable que en el modelo en el que un contacto se sitúa encima del otro. Esto se produce porque cuando el robot se acerca y sus contactos toquen con las bandas metálicas, se puede producir un pequeño rebote que haga que el contacto no sea lo suficientemente fuerte. Para contrarrestar este inconveniente, se requiere que alguno de los contactos no sea totalmente fijo. Por ello, se decide que las bandas metálicas de la base dispongan de un cierto movimiento horizontal, en forma de pulsador, que cuando los contactos del robot realicen la conexión con ellas, un sistema de muelles empuje las bandas metálicas hacia el robot.

Con estos requerimientos, se decide diseñar una estación de carga basada en un cubo hueco con tres ranuras en su frontal. La ranura más alta será la ventana tras la que se sitúan los emisores. Las otras dos serán los huecos en los

que los conectores se deslizarán hacia dentro cuando los contactos del robot los presione. Estos conectores estarán provistos de una banda metálica para realizar el contacto eléctrico, que será una lámina de cobre.

Antes de decidir el sistema de muelles a utilizar para los contactos móviles, se estudió realizar el contacto con láminas de cobre curvadas. Estas láminas, al ser muy delgadas, son flexibles pero resistentes a la vez, por lo que pueden soportar su flexión y recuperar su forma posteriormente. Sin embargo, el sistema de muelles bien diseñado es más robusto y compacto, sin dejar las piezas curvadas y sus filos tan expuestos. Otra solución estudiada fue imprimir esos contactos en plástico con una parte más delgada que le proporcionara flexibilidad al contacto con respecto a la caja dónde se instalan. Aunque es un sistema que se podría estudiar más en profundidad, el espacio dentro de la caja es muy reducido y el ejecutar un diseño de esta forma requiere más espacio en el interior.

Así pues, el sistema se basa en un conector impreso en plástico al que se le incorpora una banda de cobre de 12 x 90 mm. Igualmente se le instalan dos tornillos de métrica 3 de longitud mayor a la de los muelles, cuyas dimensiones son 7 x 12.5 mm. Estos tornillos fijos a esta pieza serán las guías para el movimiento de los conectores hacia adentro. A la caja se le fija una pieza que sirve de soporte para los conectores y tendrá unos taladros por los que se desplazan los tornillos con los muelles anteriores. Estos tornillos se introducen en tubos de plástico para evitar el rozamiento entre el tornillo y la pieza fija. En las figuras 7.11 y 7.12 se muestran las vistas trasera y frontal de la instalación de los conectores al soporte mediante los tornillos y los muelles. Igualmente se observan las láminas de cobre con sus cables de conexión.

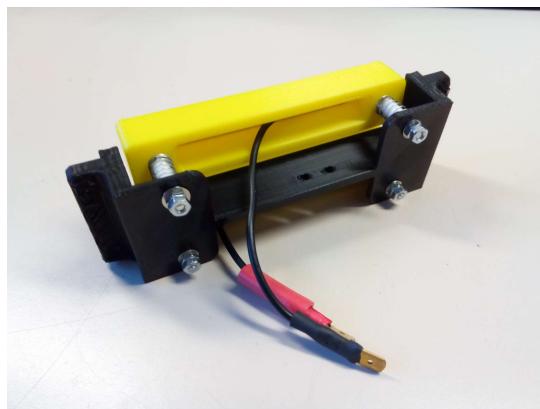


Figura 7.11: Vista trasera del soporte y los conectores.

La caja inicial, además de las 3 ranuras principales en el frontal, dispone de algunas aberturas más. Por un lado, se deja un hueco para el interruptor y para el conector del cable con el que se alimenta la estación, que será una

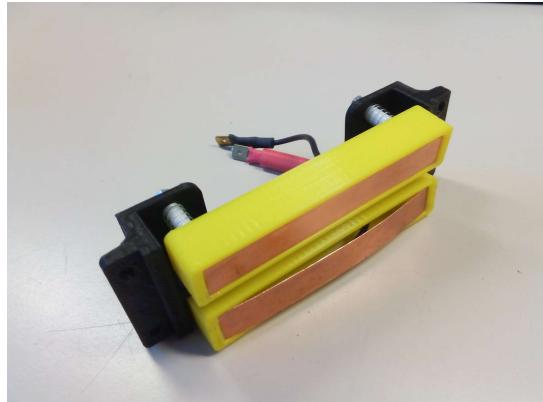


Figura 7.12: Vista frontal del soporte y los conectores.

fuente de alimentación o un cargador que suministre 14 voltios. También se realizan taladros en la parte baja para atornillar la tarjeta arduino a la caja y otros para atornillar los soportes de los conectores. Esta pieza aunque es fija, se ha diseñado independientemente a la base para facilitar la impresión de ambas piezas y además facilita el montaje de los componentes interiores. En el interior también se añade un soporte para atornillar la placa donde se instalan los emisores. Finalmente se añaden los taladros para atornillar una tapa trasera a la caja con cuatro tornillos. Esta tapa se ha obtenido del corte de una lámina de plástico de 3 mm de grosor y 11.8 x 13.5 cm, facilitando así la accesibilidad al interior de la caja. Todos los tornillos usados para la fijación de todas las piezas son de métrica 3. La vista frontal y el montaje interior de todos los componentes se muestran en las figuras 7.13 y 7.14. En el capítulo de planos se muestran las vistas isonométricas y proyecciones frontales de todas las piezas diseñadas e impresas en 3D. Los archivos con los diseños de las piezas también se pueden encontrar en la web del laboratorio [35] o en el repositorio GitHub [11].

La pieza que instalada en el robot y que incorpora los contactos eléctricos con sus cables de conexión se muestra en la figura 7.15. Ha sido diseñada para instalar en ella los dos receptores infrarrojos e incorpora dos tornillos con cabeza redonda de métrica 8 que se conectan a las entradas del circuito protector de las baterías. Dispone de los huecos necesarios para pasar los cables de las conexiones tanto de los contactos como de los receptores y se fija a la plataforma mediante dos tornillos. Además el frontal de la pieza se ha tenido que separar de la plataforma la distancia mínima para evitar el bloqueo de la rueda de apoyo delantera, ya que las ruedas de apoyo sobresalen de la plataforma durante los giros. En la figura 7.16 se muestra una fotografía del robot junto a la base con todos los componentes instalados.



Figura 7.13: Vista frontal de la estación de carga.

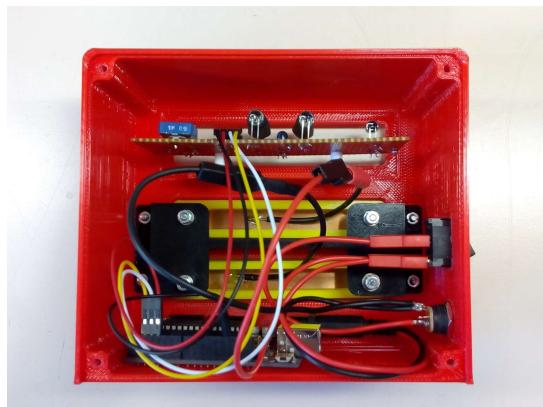


Figura 7.14: Vista del montaje de los componentes en el interior de la estación de carga.

7.4. Conclusiones

En este capítulo se ha descrito el estudio de los distintos sistemas de alimentación más comunes, se han comparado con la batería que el robot llevaba incorporada y se han analizado sus ventajas e inconvenientes. Tras el análisis se decide reemplazar el sistema de alimentación por uno de baterías de Ión-Litio y se ha detallado la circuitería que es necesaria incorporar para el uso de dichas

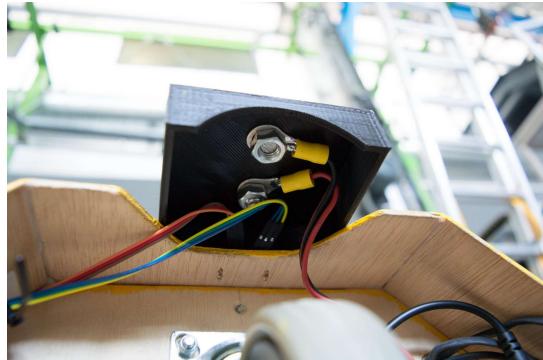


Figura 7.15: Vista de las conexiones interiores de los contactos frontales.



Figura 7.16: Vista del robot junto a la estación de carga.

baterías. Finalmente se han analizado los diferentes tipos de conectores para sistemas de carga automática y se ha completado el diseño de la estación de carga, describiendo todas las partes de las que se compone. De esta forma se finaliza todo el diseño electrónico, eléctrico y mecánico del sistema de carga. En los siguientes capítulos se describen los algoritmos de bajo y alto nivel que controlan los diferentes dispositivos electrónicos y permiten que el sistema funcione correctamente.

Capítulo 8

Programación de bajo nivel

8.1. Introducción

En este capítulo se desarrollan los algoritmos que se programan para las tres tarjetas Arduino que se requieren para el robot y el sistema de recarga. Por un lado, se desarrolla el algoritmo que se encarga de controlar la emisión de las señales desde la estación de carga, siguiendo el patrón de señales diseñado en el capítulo 5. Igualmente, se estudian distintos algoritmos para el control de los receptores instalados en el robot móvil, el posterior análisis de las señales recibidas por ellos y el envío de mensajes desde el Arduino al PC.

Por otro lado, también se desarrolla otro algoritmo que será el que se comunique con la tarjeta controladora de los motores y los sensores ultrasónicos mediante el protocolo I2C. Dicha programación también tiene que ejecutar el intercambio de mensajes del Arduino con el ordenador.

8.2. Estación de carga - Emisores

La programación para la tarjeta Arduino que controla las emisiones de las tres señales anteriormente diseñadas se puede realizar de distintas formas, pero hay que tener en cuenta varios aspectos muy importantes. Por un lado, hay que diseñar un algoritmo que realice la modulación a 38 kHz cuando se quiera enviar un pulso. Para ello, es necesario diseñar un bucle que envíe pulsos y pausas a 38 kHz, que como ya se ha comentado anteriormente, equivale a pulsos y pausas de unos 13 microsegundos.

Y debido a este requisito, aparecen los primeros problemas para realizar tal demodulación. Una de las ventajas de usar una tarjeta Arduino es la fácil programación que permite, usando sentencias sencillas que permiten realizar código más complejo. Pero la velocidad de trabajo de estas placas es bastante limitada en uno de los modelos más básicos, como es el modelo que se está usando, el Arduino Uno. En la programación de estas placas, dos de las funciones más usadas e importantes son “digitalwrite” y “digitalread”, comandos que ejecutan la escritura o lectura de algunos de los puertos digitales. Primeramente se declaran los pines de la placa que funcionan como entradas y salidas, ya sean analógicas o digitales. Posteriormente, con dichas sentencias, se obtienen las lecturas de dichos pines, pudiéndose guardar sus valores en variables del programa, o se escriben los valores deseados en los pines correspondientes declarados como salidas. El problema es que estas funciones requieren mucho tiempo de ejecución, tomando el caso del “digitalwrite” más de 50 ciclos de reloj, equivalente a una frecuencia de 142 kHz o más de 7.000 nanosegundos.

Como ya se ha comentado, la demodulación necesita realizar operaciones de escritura a 38 kHz, es decir, durante el envío de un pulso, se tiene que modificar la salida de 0 a 1, o viceversa, cada 13 microsegundos. Por lo que el uso de la función “digitalwrite” no es posible para obtener tal frecuencia. Por otro lado, hay que recordar que el arduino es una placa compuesta por un microchip AVR, por lo que también existe la posibilidad de programar este microcontrolador utilizando código nativo. Aunque este código sea más complejo y más difícil de depurar, la operación de escritura directa en los puertos necesita sólo de un par de ciclos de reloj, permitiendo realizar la modulación a la velocidad que deseamos. Así pues, la solución a este problema pasa por usar sentencias que controlen directamente los puertos del microchip.

La siguiente limitación que se encuentra en la programación es que las sentencias se ejecutan de forma secuencial en las tarjetas Arduino. Es decir, se ejecuta una acción, y cuando termina esta, se ejecuta la siguiente, y por lo tanto, no se permiten realizar distintas operaciones de forma simultánea. Así pues, no se pueden realizar escrituras o lecturas de varios pines al mismo tiempo y al tener una velocidad de ejecución limitada, la sincronización de la emisión no se realiza correctamente, existiendo un pequeño retraso en cada uno de los emisores con respecto al anterior. Este inconveniente se podría solucionar usando tarjetas del tipo FPGA u otros chips que permiten realizar operaciones paralelas. Igualmente, con algún modelo de Arduino superior se pueden conseguir velocidades mayores en las que los retrasos producidos serían menores. Sin embargo, si se programa de nuevo el chip directamente con sentencias de su código nativo, se pueden realizar escrituras y lecturas de un puerto al completo. El modelo de Arduino Uno utilizado lleva incorporado el chip ATMEGA 168, que consta de 3 puertos, uno analógico y dos digitales. Por ejemplo, cada bit del registro del puerto D corresponde con las salidas digitales de los pines numerados del 0 al 7. Por lo tanto, si se realiza la escritura directamente en uno de los dos puertos digitales, se consigue realizar la escritura en distintos pines de salidas al mismo

tiempo. Así pues, de nuevo, la solución al problema de la sincronización pasa por escribir código nativo para controlar el puerto que incluya los pines de salida que corresponden a cada emisor.

Por lo tanto, usando la programación en código nativo, se desarrolla el algoritmo que realiza la operación de encendido y apagado de uno de los leds a la frecuencia de 58 kHz, correspondiente a la modulación que se necesita. El algoritmo de la función Encendido_Led_IR se muestra en la figura 8.1 para el caso del emisor derecho, pero sirve para controlar cualquiera de los emisores.

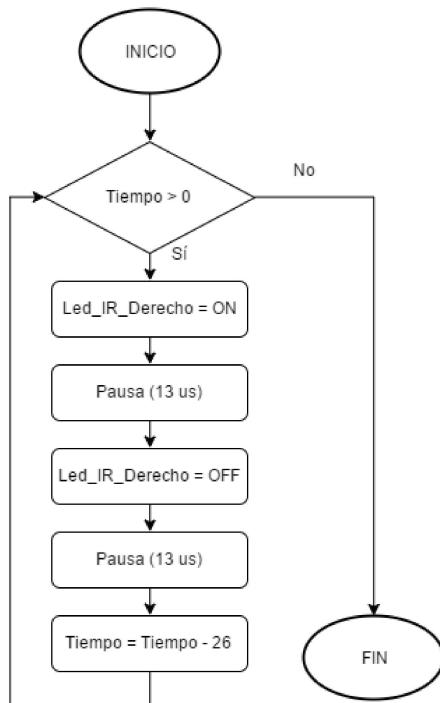


Figura 8.1: Algoritmo de la función Encendido_Led_IR.

La función consta de dos parámetros de entrada, el led que se quiere controlar y el tiempo del pulso en alto. Durante la duración de dicho pulso, el programa encenderá y apagará el led cada 13 microsegundos. Para ello, el bucle comprueba si la variable del tiempo del pulso es mayor que 0. Si es así, se procede a emitir un ciclo de encendido y apagado a la frecuencia deseada, es decir, el emisor se enciende durante 13 microsegundos y posteriormente se apaga durante 13 microsegundos. Tras esta operación, se resta al tiempo inicial la cantidad de 26 microsegundos, equivalente a la duración del ciclo. Finalmente se vuelve a comprobar si este nuevo valor de tiempo es mayor que 0. Si es así, se repite el ciclo y en el caso de que no lo sea, se sale del bucle para seguir con los comandos

del programa principal.

El programa principal se basa en un bucle que se repite continuamente. Este bucle, que se representa en la figura 8.2, realiza las operaciones para conseguir emitir la señal según se ha diseñado en los apartados anteriores. Es decir, encenderá todos los emisores a la vez, luego producirá una pausa, y a continuación encenderá un led tras otro durante un tiempo y con unas determinadas pausas entre ellos. Al final, se produce una pausa fija para todos los leds. Para ello, durante el encendido de cada emisor, se realiza una llamada a la función Encendido_Led_IR. Igualmente, esta función está diseñada para poder realizar la escritura en varios pines a la vez usando los registros del puerto adecuado. Por lo cual, la llamada a la función Encendido_Led_IR (TODOS, Tiempo_inicial) activará el envío de la señal de los tres emisores a la vez durante el tiempo deseado.

La programación de la tarjeta arduino se realiza a través de su propio IDE oficial usando un lenguaje propio basado en el lenguaje de alto nivel Processing, muy similar a C++. Al estar basado en C, Arduino soporta todas las funciones de C y algunas de C++. El código completo de dicha programación basado en los algoritmos anteriores se incluye en los anexos correspondientes.

8.3. Robot móvil - Receptores

Para el control de los receptores infrarrojos se requiere otra tarjeta Arduino que será únicamente utilizada para tal fin. Los inconvenientes de usar una tarjeta de este tipo para recibir y analizar estas señales son mayores que en el caso del control de la emisión de las señales. La operación de “digitalread” es incluso más lenta que el comando “digitalwrite”, por lo que necesita aún más ciclos de reloj. Igualmente, es necesario que la lectura de todos los sensores se realice al mismo tiempo, es decir, cuando se reciba el inicio del código en alguno de los receptores, se comienza la lectura de todos a la vez. Una vez terminada la lectura de los códigos enviados, se procede a analizar las señales recibidas y finalmente se envía un código binario a través del puerto serial con la información de las señales recibidas por cada uno de los sensores. Detectando qué señales recibe cada uno de los receptores se puede conocer en qué zona se encuentra cada uno de ellos, y por lo tanto, conocer la posición del robot respecto a la estación de carga.

Así pues, debido a las mismas razones que en el caso de los emisores, es necesario programar el chip con código nativo y haciendo uso de los comandos que controlan directamente los puertos. En este caso, el puerto utilizado es el mismo que en el caso anterior, pero los pines de la placa conectados a las salidas de los receptores están definidos, en lugar de salidas digitales, como entradas digitales.

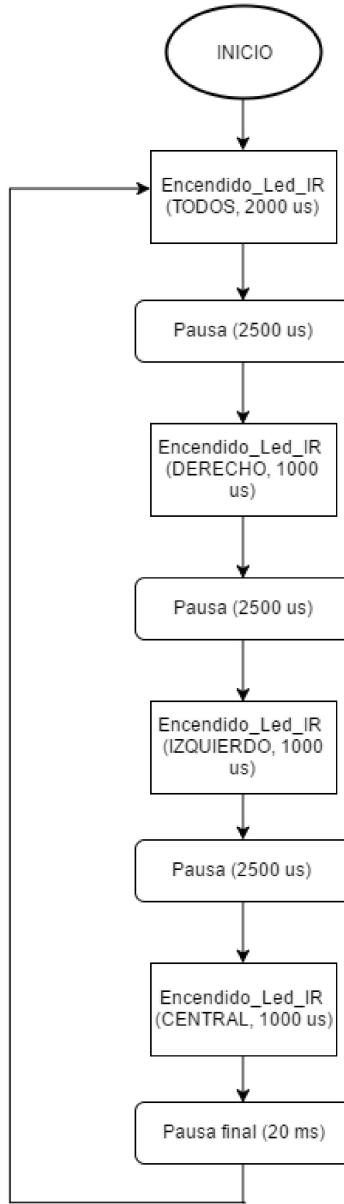


Figura 8.2: Algoritmo del programa que controla los emisores IR.

Aún así hay distintos modos de realizar el algoritmo que controle la lectura de estas señales. En todos los casos, el algoritmo de lectura tiene que ejecutar un bucle en el que el sistema se encuentra a la espera de recibir la señal que le

indica el comienzo del envío de un nuevo código. Una vez recibido este código, la lectura se puede realizar de distintos modos. Una solución es medir desde el pulso de inicio las duraciones de los pulsos en alto y las duraciones de las pausas hasta el final del código, que tiene una duración total fija. Estas duraciones se guardan en un vector y se comparan con los vectores patrón que corresponden a cada una de las señales enviadas. Para ello hay que definir primeramente un vector para cada una de las posibilidades que pueden darse y guardarlo en el programa para realizar las comparaciones con los valores de las señales recibidas. Además es necesario el uso de una variable fija que determine una tolerancia para la comparación, ya que los valores medidos en microsegundos varían de una lectura a otra.

Aunque esta primera opción fue la que se desarrolló en primer lugar para controlar la recepción de las señales, finalmente se ha programado una opción más sencilla y que no necesita ni guardar vectores patrón ni realizar comparaciones con una determinada tolerancia. Al conocer perfectamente las señales que se pueden recibir en cada uno de los sensores, se ha desarrollado un sistema en el que el programa sólo realiza tres lecturas en tiempos definidos a partir de la recepción del código de inicio. Es decir, tras la detección del código de inicio, el programa se queda a la espera hasta el momento en el que se produce el envío del pulso por el primer emisor. Como ese tiempo es conocido, el programa realiza una lectura de las salidas de todos los receptores justo durante el envío de ese primer pulso y guarda el resultado. Ese valor se guardará en un número binario en el que cada bit corresponde a la lectura de uno de los receptores. El valor de ese bit se guardará como 1 si se recibe tal pulso y 0 si no se recibe. Posteriormente, el programa realiza otra pausa hasta el tiempo en el que el segundo emisor emite su pulso. Se realiza la misma operación y se guarda su resultado en otro número binario. Y así se vuelve a repetir con el pulso correspondiente al tercer emisor. En la figura 8.3 se muestra la señal demodulada correspondiente a la superposición de las señales de los 3 emisores recibida por un receptor. En ella se marca el periodo durante el que se analiza la señal correspondiente al pulso de inicio y posteriormente se realizan las 3 lecturas. Conociendo el patrón de la señal que envían los emisores, basta con conocer los tiempos en los que realizar cada una de las tres lecturas.

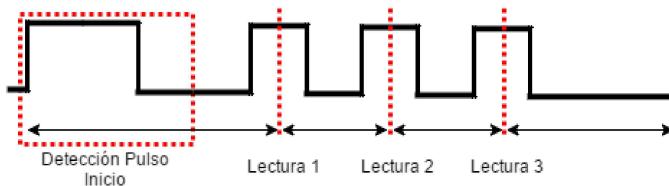


Figura 8.3: Lecturas realizadas para la recepción de los códigos.

Dichas lecturas se hacen al mismo tiempo para todos los receptores. Una vez que cualquiera de ellos reciba el pulso de inicio, se procede a la lectura del

registro del puerto deseado en los tiempos definidos. Dicho puerto incluye los diferentes pines a los que se conectan las salidas de todos los receptores, por lo que se obtienen las lecturas de todos los receptores de forma simultánea en el mismo instante de tiempo.

En cada lectura se guardan los valores recibidos por cada receptor en una variable en formato binario, y finalmente se construye un número binario de mayor dimensión encadenando los números de las 3 lecturas. Este número es el que se envía al ordenador para que sea este el que analice en qué zona se encuentra y calcule los movimientos que tiene que realizar para conectarse a la estación de carga. Si el robot no recibe ninguna señal durante un cierto periodo de tiempo, envía un mensaje al PC indicando que sigue sin detectar ningún emisor. Este mensaje sirve para detectar errores de comunicación y también para inicializar todas las comunicaciones aunque no haya lecturas de señales en ninguno de los receptores. El funcionamiento del intercambio de mensajes entre Arduino y el PC mediante ROS se explica en los capítulos siguientes, donde también se detalla el código completo y el uso de la librería ros.h.

El diagrama de flujos del algoritmo implementado se muestra en la figura 8.4.

8.4. Robot móvil - MD23 y SFR08

La última tarjeta Arduino es la que controla el movimiento del robot. Dicha tarjeta se conecta a la controladora MD23 mediante el protocolo I2C y esta se comunica con los motores para enviarles las consignas de velocidad y recibir información como pueden ser las lecturas de los encoders o la tensión en voltios. Los 3 sensores ultrasónicos también se conectan al Arduino mediante el protocolo I2C. En el modelo Arduino UNO, los pines destinados a las comunicaciones I2C son los pines analógicos A4 y A5.

El protocolo I2C es un bus de datos serial desarrollado por Philips y está diseñado para funcionar como un bus maestro-esclavo, tal y como se muestra en la figura 8.5. La conexión eléctrica se muestra en la siguiente figura. Además de los hilos de la conexión a tierra y a 5 voltios, hay dos conexiones de datos. SDA es el destinado a transmitir los datos y SCL está conectado al reloj asíncrono que indica cuando se realizan las lecturas de datos. La velocidad del reloj no es fija según el protocolo, sino que depende del reloj del dispositivo maestro.

Este bus permite conectar múltiples dispositivos esclavos con el maestro, que en este caso es la tarjeta Arduino, con sólo dos cables además de las conexiones de alimentación. Los dispositivos esclavos son los 3 sensores ultrasónicos y la tarjeta MD23, y cada uno de ellos tiene una dirección única de 7 bits.

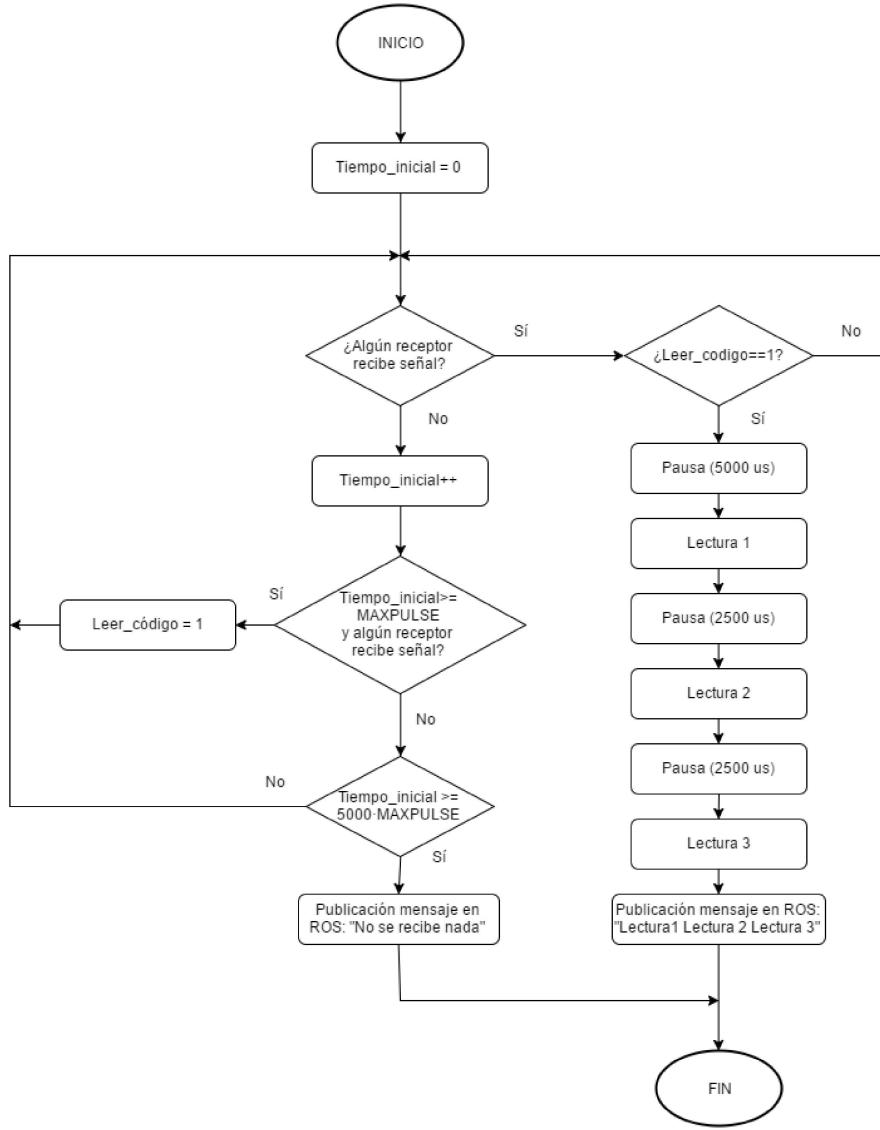


Figura 8.4: Algoritmo del programa que controla los receptores.

Las direcciones de los 4 dispositivos se pueden obtener de las hojas de datos de cada elemento y también suelen disponer de un código de luces que al conectarlos a la alimentación indican su dirección según el número o duraciones de pulsos de leds que tienen incorporados. Sin embargo, estas direcciones se pueden cambiar y en el caso de tener varios dispositivos iguales es necesario para poder comunicarse con todos ellos. Por ello, para obtener las direcciones

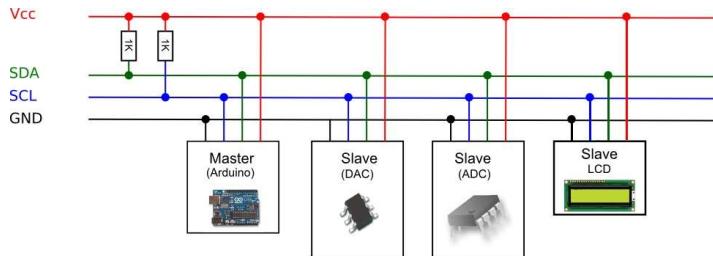


Figura 8.5: Ejemplo del bus de datos I2C. *Fuente:* [26]

se optó por programar un algoritmo que hace un barrido a todas las direcciones posibles y que permite conocer las que están siendo utilizadas por cada uno de los dispositivos. Las direcciones obtenidas son las siguientes:

- Controladora MD23: 89
- Sensor SFR08 Central: 127
- Sensor SFR08 Izquierdo: 113
- Sensor SFR08 Derecho: 114

Para facilitar la programación del Arduino, se hace uso de la librería Wire.h, que permite programar el algoritmo para realizar las comunicaciones de una forma mucho más rápida. Los comandos de esta librería, así como el código completo se muestra en los anexos. La programación se basa en el algoritmo mostrado en la figura 8.6. Su funcionamiento se basa en repetir un bucle en el que se realiza una lectura de algunos de los dispositivos y posteriormente se publica esa información en un mensaje que se envía a ROS. Dichas lecturas son las de los sensores ultrasónicos o las de algunos de los registros de la controladora MD23, que pueden ser los valores de los encoders o la tensión que alimenta a la tarjeta. Por otro lado, continuamente se están recibiendo desde ROS las consignas de velocidad para los motores, por lo que en el bucle del algoritmo también hay dos instrucciones que indican la escritura de dichos valores en los registros de la tarjeta MD23 correspondientes.

8.5. Conclusiones

Tras analizar los requerimientos necesarios en la programación de la electrónica de control para los distintos dispositivos, se describen los distintos algoritmos de bajo nivel que corresponden a las programaciones de cada una de las tarjetas Arduino. Se estudia el modo de programar las tarjetas para conseguir las

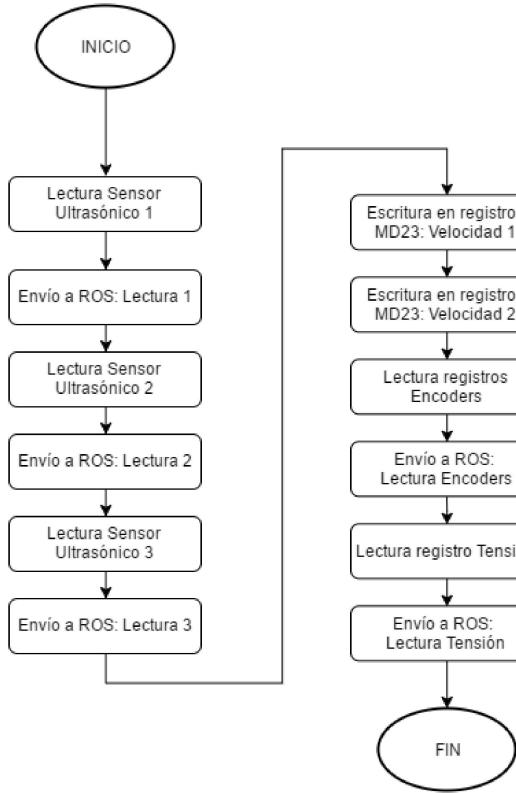


Figura 8.6: Algoritmo del programa que controla la tarjeta MD23 y los sensores ultrasónicos.

velocidades necesarias e igualmente se describe el protocolo de comunicaciones I2C usado por una de las tarjetas. Los códigos completos con sus descripciones se incluyen en el anexo correspondiente. En el siguiente capítulo se procede a desarrollar los algoritmos y códigos de alto nivel correspondientes al control de estos dispositivos desde ROS.

Capítulo 9

Programación de alto nivel

9.1. Introducción

En este capítulo se explican los conceptos principales y el funcionamiento general de ROS. Se detalla el sistema de archivos que requiere el sistema y se describen los algoritmos que se han programado para las transferencias de mensajes entre las tarjetas Arduino y el ordenador, el análisis de la información recibida y el control del movimiento del robot.

9.2. Funcionamiento de ROS

ROS apareció en 2007 para apoyar el desarrollo de un robot con inteligencia artificial en la Universidad de Stanford. A partir del año 2008, el Instituto de investigación robótica Willow Garage se ocupó de su desarrollo en colaboración con otras instituciones y desde el año 2013 es OSRF (Open Source Robotics Foundation) quien se encarga de su desarrollo y difusión. En la figura 9.1 se muestran los logos de ROS y OSRF.

ROS es un conjunto de software libre que funciona sobre Linux y provee de todas las herramientas necesarias para desarrollar software para robots. Ofrece los servicios estándar de un sistema operativo aplicado a la robótica, como pueden ser la abstracción de hardware, el control de dispositivos de bajo nivel o el intercambio de mensajes entre aplicaciones y procesos. El funcionamiento de ROS se basa en una arquitectura basada en grafos formados por nodos, que representan los procesos, y que pueden enviar o recibir mensajes. Estos mensajes pueden ser, por ejemplo, lecturas de sensores u órdenes para controlar



Figura 9.1: Logos de ROS y OSRF. *Fuente:* [23]

actuadores.

A pesar de la importancia de la baja latencia en la robótica, ROS no es un sistema operativo que trabaje con sistemas en tiempo real, aunque puede integrarse en él código con funcionamiento en tiempo real. Sin embargo, ya se encuentra en desarrollo la siguiente generación de este sistema, denominado ROS 2.0, que entre otras características nuevas, se encuentra la de estar diseñado para control en tiempo real. Por ello, en lugar de lanzar otra nueva versión de ROS con actualizaciones y mejoras, es necesario remodelar el sistema operativo completo.

Desde su aparición se han lanzado distintas versiones de ROS y estas pueden ser incompatibles entre ellas. Todas las versiones son software libre bajo licencia BSD, la cual permite licencia para uso comercial y de investigación, aunque las contribuciones que aportan los usuarios tienen licencias de todo tipo. Otra de las ventajas de ROS es la gran comunidad de usuarios que aporta y comparte códigos, paquetes completos y manuales que facilitan el uso y aprendizaje de este sistema operativo. Debido a la filosofía de compartir el desarrollo de los componentes más comunes, el usuario puede reutilizar esos recursos ya creados y compartidos permitiendo desarrollos cada vez más rápidos y específicos. Aunque oficialmente se recomienda su uso en Ubuntu, una distribución de Linux, también se está adaptando a otros sistemas operativos de forma experimental.

El ecosistema de ROS está formado por 3 bloques principales:

- La base principal del sistema operativo, que es un conjunto de herramientas que se usan para desarrollar software independientemente del lenguaje o la plataforma en la que se ejecute. Todas sus versiones se lanzan bajo licencia BSD.
- Las librerías clientes, que permite la implementación de código en distintos lenguajes, como pueden ser rosCPP para programar en C++ o rosPy para Python. Estas librerías también se distribuyen con licencia BSD.
- Los paquetes son un conjunto de códigos que implementan funcionalidades

y aplicaciones como pueden ser el control de motores, el reconocimiento de imágenes o la navegación mediante SLAM.

9.3. Conceptos

En ROS, el sistema de control está formado por distintas unidades que se distribuyen en diferentes niveles que permiten organizar el sistema en unidades más pequeñas que se interrelacionan entre ellas para dar forma al sistema completo. Los diferentes niveles del sistema de archivos son los siguientes:

- Paquetes: Son la principal unidad para crear software en ROS. Un paquete es el archivo de menor nivel que se puede construir y puede contener información sobre los procesos (nodos), librerías, archivos de configuración o cualquier otra información necesaria que se pueda organizar en un archivo.
- Metapaquetes: Son grupos de paquetes especializados que incluyen un conjunto de paquetes con funcionalidades relacionadas.
- Manifiestos: Son archivos nombrados como package.xml y proporcionan metadatos sobre un paquete, incluyendo su nombre, versión, descripción, licencia, dependencias y otras informaciones.
- Tipos de mensajes: Guardados en el archivo my_package/msg/ MyMessageType.msg, proporciona la descripción de los mensajes enviados en ROS, así como la información sobre sus estructuras.
- Tipos de servicios: Al igual que el caso anterior, la estructura de las peticiones y respuestas de los servicios en ROS se define en un archivo, en este caso en my_package/srv/MyServiceType.srv.

La interrelación entre las distintas partes del sistema se basa en una estructura de grafos. Esta estructura es una red peer-to-peer de nodos que procesa la información en conjunto. Los principales conceptos de esta estructura son los siguientes:

- Nodos: Los nodos son los procesos que ejecutan alguna acción. ROS está diseñado para ser muy modular a una escala de grano fino, por lo que un sistema de control robótico está formado por muchos nodos. Pueden representar tanto a sensores y a actuadores o ser procesos independientes. Por ejemplo, un nodo puede controlar un láser o una cámara, otro nodo controla el movimiento de unas ruedas y otros nodos ejecutan los procesos de localización o cálculo de trayectorias.

- Máster: El Máster proporciona la información necesaria para que unos nodos se comuniquen con otros y puedan enviar o recibir mensajes y activar servicios.
- Servidor de parámetros: Se utiliza para guardar información en una localización central y forma parte del Máster.
- Mensajes: Los nodos se comunican unos con otros mediante mensajes y pueden ser enviados y recibidos por estos. Los mensajes son estructuras de información definidas, las cuales pueden estar formadas por formatos estándar básicos como números enteros, flotantes, cadenas de caracteres, etc. También pueden ser arrays compuestos de los tipos anteriores.
- Tópicos: Es el nombre utilizado para identificar el contenido de un mensaje que es enviado por un nodo. Un nodo envía los mensajes publicandolos en un determinado tópico. Si se desea que un nodo reciba esos mensajes, el nodo se suscribe al tópico para recibirlos. Por un lado, un nodo simple puede publicar y suscribirse a distintos tópicos, así como un único tópico puede ser publicado o recibido por distintos nodos al mismo tiempo.
- Servicios: Las comunicaciones en las que los nodos publican o se suscriben a tópicos son muy flexibles, pero las interacciones de peticiones y respuestas son muy convenientes en sistemas distribuidos. Los servicios están diseñados para estos casos y están compuestos de estructuras con dos mensajes definidos, uno para la petición y otro para la respuesta.
- Bags: Son formatos para guardar y leer la información de los mensajes de ROS. Son muy utilizados para realizar desarrollos y pruebas en los casos en los que obtener la información de los mensajes es difícil, como pueden ser las lecturas de un sensor láser. Por lo tanto, se pueden utilizar, por ejemplo, para simular las lecturas que un sensor láser realiza durante el movimiento de un robot.

En la figura 9.2 se muestra el caso más simple de envío de mensajes de un nodo a otro, en el que un nodo actúa como publisher y otro como subscriber, es decir, el primero publica mensajes al tópico y el segundo se suscribe a este para recibirlas.

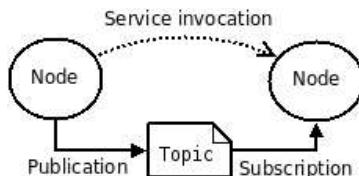


Figura 9.2: Ejemplo de dos nodos. *Fuente:* [44]

9.4. Archivos

Para la realización del presente proyecto se ha utilizado el ordenador portátil descrito en el capítulo 3 al que se le instaló el sistema operativo Ubuntu versión 14.04 disponible en la web oficial de Ubuntu [40]. Una vez actualizado el sistema, se procedió a instalar ROS siguiendo las instrucciones que se indican en su página web oficial [44]. Aunque ya se había lanzado una versión de ROS posterior, denominada Jade, la versión que se instaló fue la anterior, denominada Indigo. Sus propios desarrolladores recomiendan usar esta versión por su estabilidad, tanto por el sistema en sí como por las librerías o códigos disponibles. El control de la plataforma en el proyecto anterior fue realizado con una versión más antigua de ROS, por lo que no se ha hecho uso de los programas que se desarrollaron y se ha diseñado y programado todos los programas de nuevo.

Una vez instalado el sistema ROS sobre Ubuntu, se procede a crear el sistema de archivos necesario para hacer funcionar todos los programas. En primer lugar se crea un nuevo espacio de trabajo (Workspace) del tipo catkin y un paquete denominado Pack1. En la tabla 9.1 se observan las carpetas creadas y la jerarquía de los archivos en un espacio de trabajo genérico.

workspace_folder/	WORKSPACE
src/	SOURCE SPACE
CMakeLists.txt	Toplevel - CMake file, provided by catkin
package_1/	PACKAGE FOLDER
CMakeLists.txt	CMakeLists.txt file for package_1
package.xml	Package manifest for package_1

Tabla 9.1: Estructura genérica de los archivos en ROS

Usando los comandos adecuados, el sistema nos crea el sistema de carpetas y archivos anterior, evitando tener que crearlos uno por uno. Posteriormente se procede a modificar los archivos CMakeLists.txt y package.xml que el sistema crea por defecto. Todos los paquetes necesitan estos archivos y se modifican según los requerimientos de cada paquete.

En el caso del archivo package.xml hay que modificar y añadir las dependencias directas e indirectas de las que hace uso el paquete. Para este paquete se añaden las dependencias de las librerías std_msgs, rospy y roscpp tanto en las dependencias de construcción del paquete como de funcionamiento. En este archivo también se escribe la información acerca de la persona que ha creado o mantiene el paquete, el tipo de licencia, o cualquier otra información útil sobre el propio paquete.

El archivo CMakeLists.txt es un archivo necesario que describe cómo y dónde se instalará el paquete que se construye. En este archivo se especifican las li-

brerías y los mensajes que se utilizan en el programa. En el caso de usar un formato propio de mensajes, hay que definirlo también en este archivo.

Una vez modificados estos archivos, se procede a la construcción del paquete con la sentencia `catkin_make`, que realiza las comprobaciones para proceder a crear los archivos necesarios para poder ejecutar los programas del paquete.

Dentro de la carpeta del paquete `Pack1` se crean manualmente dos carpetas nuevas. En primer lugar se crea una denominada `scripts`, donde se guardan los programas, que funcionan como nodos. Estos programas se pueden escribir en distintos lenguajes, destacando Python y C++. En este caso se realizará la programación en Python por ser un lenguaje interpretado y no compilado, por lo que permite realizar modificaciones del código y pruebas de una forma más rápida. La otra carpeta se denomina `launch` y en ella se creará un archivo que permite el inicio de distintos nodos al mismo tiempo. De esta forma se evita tener que iniciar independientemente cada uno de los nodos. La jerarquía final de los archivos se muestra en la tabla 9.2.

<code>catkin_ws_folder/</code>	WORKSPACE
<code>src/</code>	SOURCE SPACE
<code>CMakeLists.txt</code>	Toplevel - CMake file, provided by catkin
<code>package.xml</code>	Package.xml file
<code>Pack1/</code>	PACKAGE FOLDER
<code>CMakeLists.txt</code>	CMake file
<code>package.xml</code>	Package.xml file
<code>launch/</code>	LAUNCH FOLDER
<code>control.launch</code>	Launch file
<code>scripts/</code>	SCRIPTS FOLDER
<code>listener.py</code>	Nodes files

Tabla 9.2: Estructura de los archivos en ROS

Una vez creados los programas para cada nodo y el archivo `.launch`, es necesario volver a construir el paquete completo con la sentencia `catkin_make`. Esta acción se ejecutará cada vez que se realice cualquier modificación en cualquiera de los archivos.

El algoritmo del nodo principal y su programación en Python, que se ha denominado `listener.py` se describe en el siguiente apartado. Este nodo será el encargado de comunicarse con las dos tarjetas Arduino, enviar y recibir la información necesaria y calcular los movimientos que el robot tiene que realizar.

El archivo `.launch` inicia el nodo `listener.py` y otros dos nodos, denominados `ArduinoMotor` y `ArduinoIR`. Estos dos nodos que controlan cada una de las tarjetas Arduino que incorpora la plataforma se inicializan ejecutando dos veces el mismo programa con argumentos distintos. El programa es el denominado `serial_node.py` del paquete `rosserial_python` y se ejecuta dos veces con denomi-

naciones y argumentos distintos. Por lo tanto, los argumentos para el inicio de cada uno de los nodos que corresponde a cada Arduino son los puertos USB a los que están conectados, que son dev/ttyUSB0 y dev/ttyUSB1.

9.5. Algoritmos

El programa principal en el sistema de archivos se denomina listener.py y se guarda en la carpeta scripts. Este programa tiene dos funciones principales. Por un lado, se encarga de realizar las comunicaciones con las dos tarjetas Arduino y por otro analiza todos los mensajes recibidos y calcula el movimiento que el robot tiene que realizar. En la figura 1.2 se muestra un esquema con la jerarquía de los distintos elementos que componen el sistema.

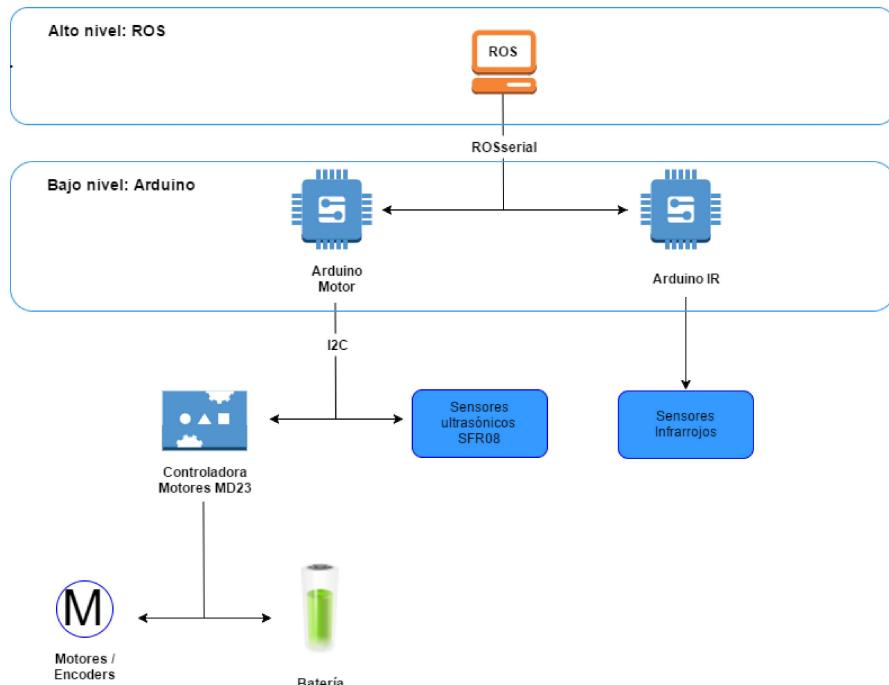


Figura 9.3: Esquema de la jerarquía de los elementos del sistema.

Las comunicaciones se realizan estableciendo enlaces entre el nodo principal listener y los dos nodos correspondientes a cada Arduino, denominados Arduino-Motor y ArduinoIR. Para ello, en el programa principal se declaran las funciones necesarias para que el nodo listener reciba los mensajes suscribiéndose a los tópicos en los que los otros dos nodos publican sus mensajes. Del nodo Arduino-

Motor se reciben los mensajes correspondientes a las lecturas de las distancias de los tres sensores ultrasónicos, los valores de los encoders de cada motor y la tensión de la batería. Del nodo ArduinoIR se recibe un mensaje de formato binario que contiene toda la información de las distintas señales recibidas por los cuatro sensores infrarrojos de la plataforma. Por tanto, para cada uno de esos mensajes se crea un tópico con un determinado nombre y un formato de mensaje establecido.

Además de la suscripción a los tópicos correspondientes a esos mensajes, también se incluyen las sentencias para realizar la publicación de mensajes desde el programa principal al nodo ArduinoMotor. Estos mensajes se publican en los tópicos de velocidad y giro, denominados speed1 y speed2, y serán recibidos por la tarjeta Arduino conectada a la controladora de los motores.

En la figura 9.4 se muestra la estructura de grafos con los nodos descritos y sus conexiones a través de los tópicos correspondientes, obtenido con el comando rqt_graph.

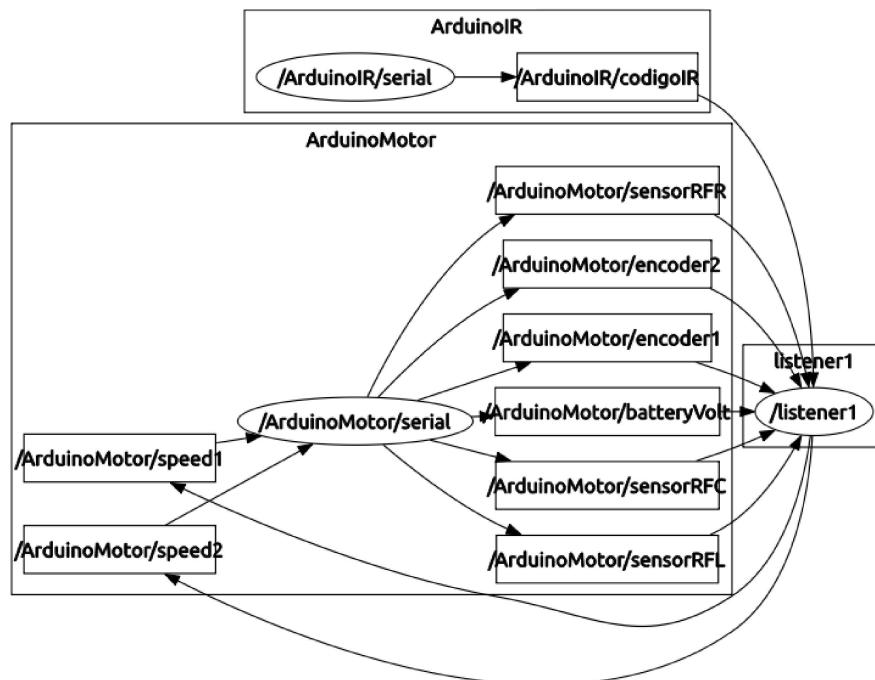


Figura 9.4: Estructura de grafos del sistema.

Para que los nodos de las tarjetas Arduino publiquen y se suscriban a sus correspondientes tópicos, en la programación de las tarjetas se hace uso de las librerías correspondientes a la transmisión de mensajes con ROS y los

tipos de mensajes que se intercambian. El código en el Arduino que controla la transferencia de mensajes incluye la información sobre los tópicos y los tipos de formatos de los mensajes, que será distinto para cada tipo de lectura. El código completo se muestra en los anexos.

El algoritmo del nodo principal se basa en el control de 4 estados en los que se puede encontrar el robot, tal y como se muestra en la figura 9.5 que muestra una simplificación del algoritmo principal. El primer estado se denomina Búsqueda, que se representa en la figura 9.6, y es el estado que debe activarse para que el robot comience a moverse en su entorno en búsqueda de alguna señal emitida por la base de carga. En este estado, el robot se mueve hacia adelante y dependiendo de los obstáculos detectados por los sensores ultrasónicos realizará un giro u otro. De esta forma, el robot realiza trayectorias en distintas direcciones hasta que detecta alguna señal infrarroja. Si la variable “recargar” es igual a 1, indica que el robot requiere una carga de sus baterías, por lo que cuando detecte alguna señal infrarroja, pasará al estado Localizado-Lejos. En este estado, cuyo algoritmo se puede observar en la figura 9.7, el robot se encuentra en alguna de las zonas exteriores y más lejanas delimitadas por las señales infrarrojas, por lo que según las señales que detecte, realizará un movimiento u otro con el objetivo de encontrar la región central que se sitúa enfrente de la base de carga. Una vez que alcance la región central cercana a la base, el robot pasa al estado Localizado-Cerca. En este estado, el robot reduce la velocidad de sus movimientos para permitir acercarse a la base de forma más precisa y realizar el contacto con las conexiones de la base de forma correcta. Cuando el robot detecta que se produce un aumento en la tensión de los bornes de la batería, indica que el robot se ha conectado correctamente. En este momento, el robot pasa al estado de Carga y desconexión. Los diagramas de flujo de los dos últimos estados se representan en la figura 9.8. En el estado Carga y desconexión, el robot detiene sus motores y queda a la espera de la orden para finalizar la carga. En los ensayos se han realizado pruebas en las que el robot se conecta durante un determinado tiempo para posteriormente realizar la desconexión. En la desconexión, el robot se desplaza hacia atrás una distancia que asegure la desconexión y posteriormente realiza un giro para continuar con su camino, pasando de nuevo al estado inicial pero con la variable “recargar” igual a 0 para que no se vuelva a producir la búsqueda de la base hasta que se necesite de nuevo.

En la figura 9.9 se muestra un diagrama de flujo con todos los algoritmos de los diferentes estados simplificados, correspondiente a la unión de los diferentes algoritmos de cada estado. El código completo de todo el programa se incluye en el anexo correspondiente.

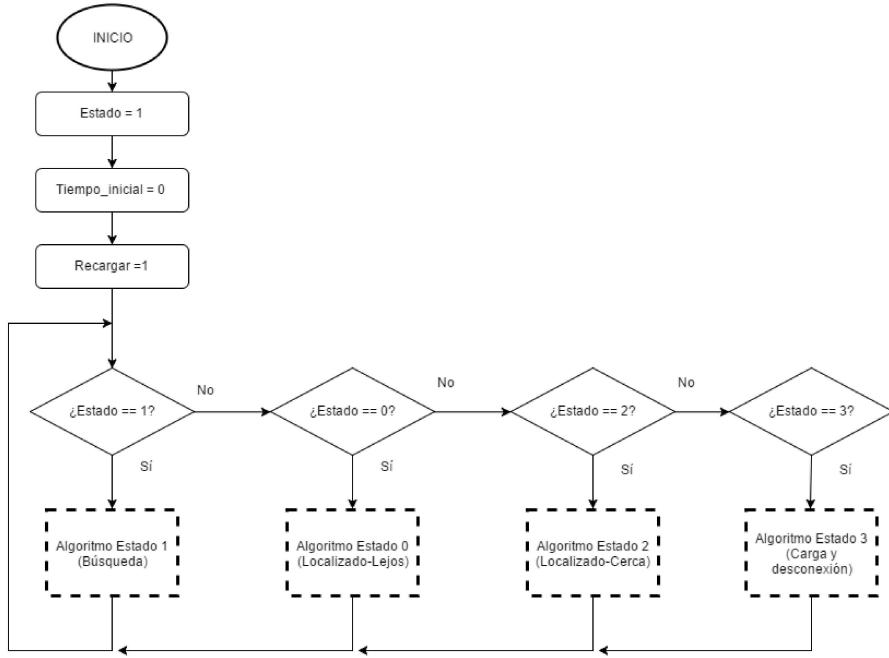


Figura 9.5: Diagrama de flujo del programa principal.

9.6. Conclusión

En este capítulo se han introducido los principales conceptos del sistema ROS y las ventajas de su utilización. Una vez estudiado el sistema de archivos necesario para la ejecución de los programas que controlan el movimiento del robot, se ha procedido a diseñar los algoritmos que definen el programa principal. El código completo de este programa y los códigos de los demás archivos de configuración y lanzamiento se adjuntan en el anexo correspondiente. En el siguiente capítulo se procede a describir el funcionamiento general del sistema, se detallan los distintos experimentos realizados y se finaliza con las líneas futuras de trabajo y las conclusiones finales.

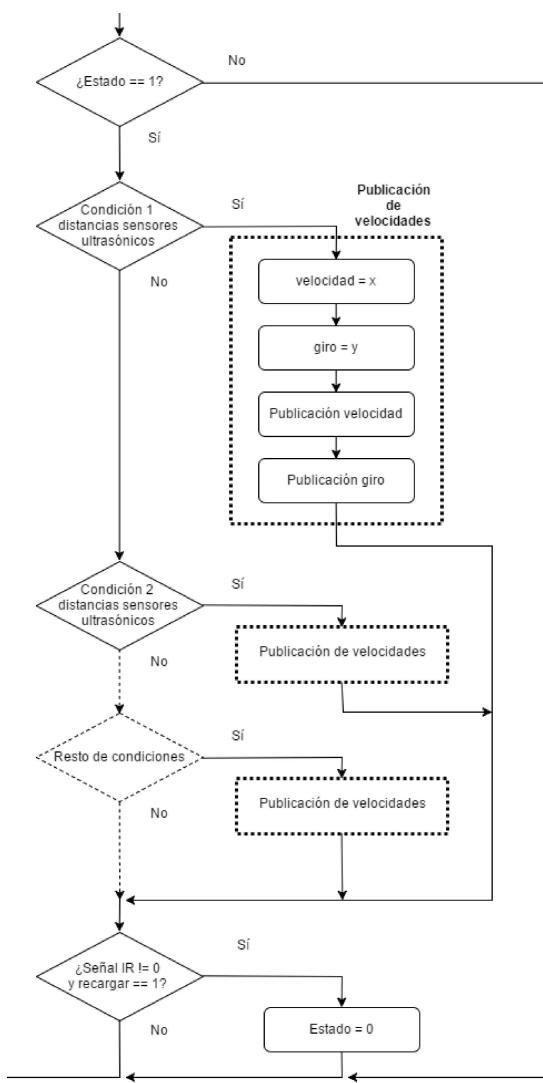


Figura 9.6: Diagrama de flujo del estado Búsqueda.

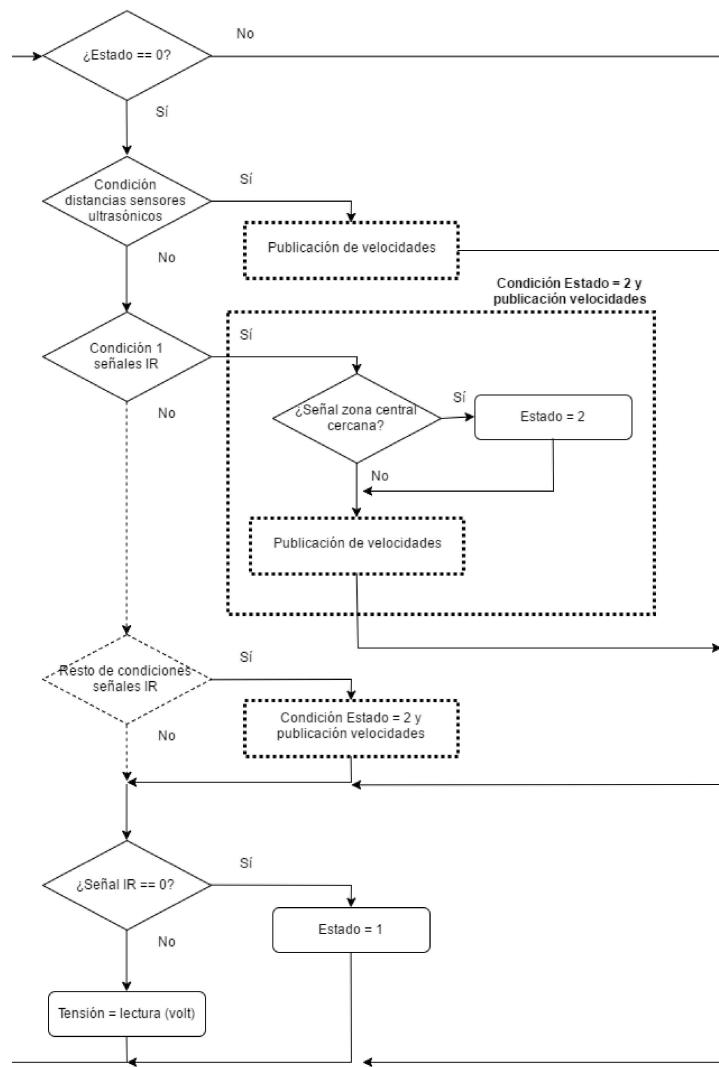


Figura 9.7: Diagrama de flujo del estado Localizado-Lejos.

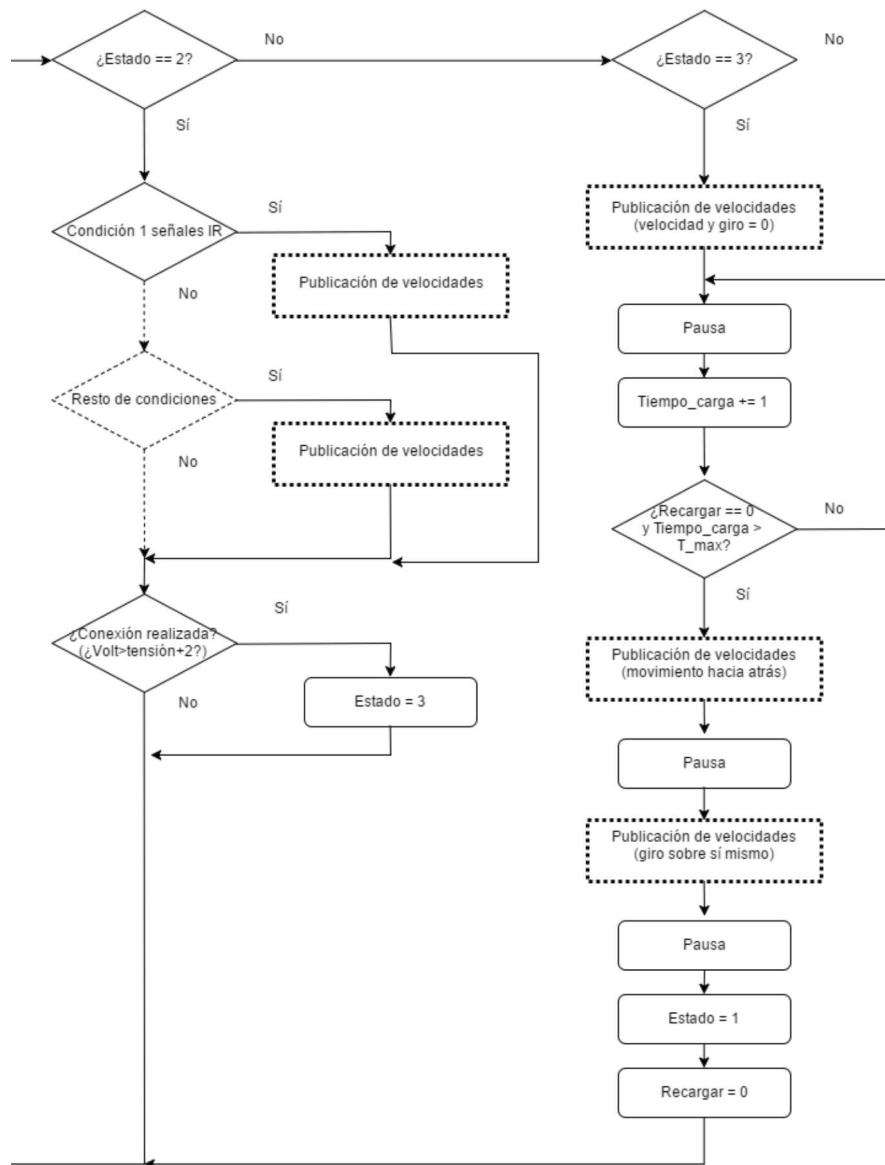


Figura 9.8: Diagrama de flujo del estado Localizado-Cerca y el estado Carga y desconexión.

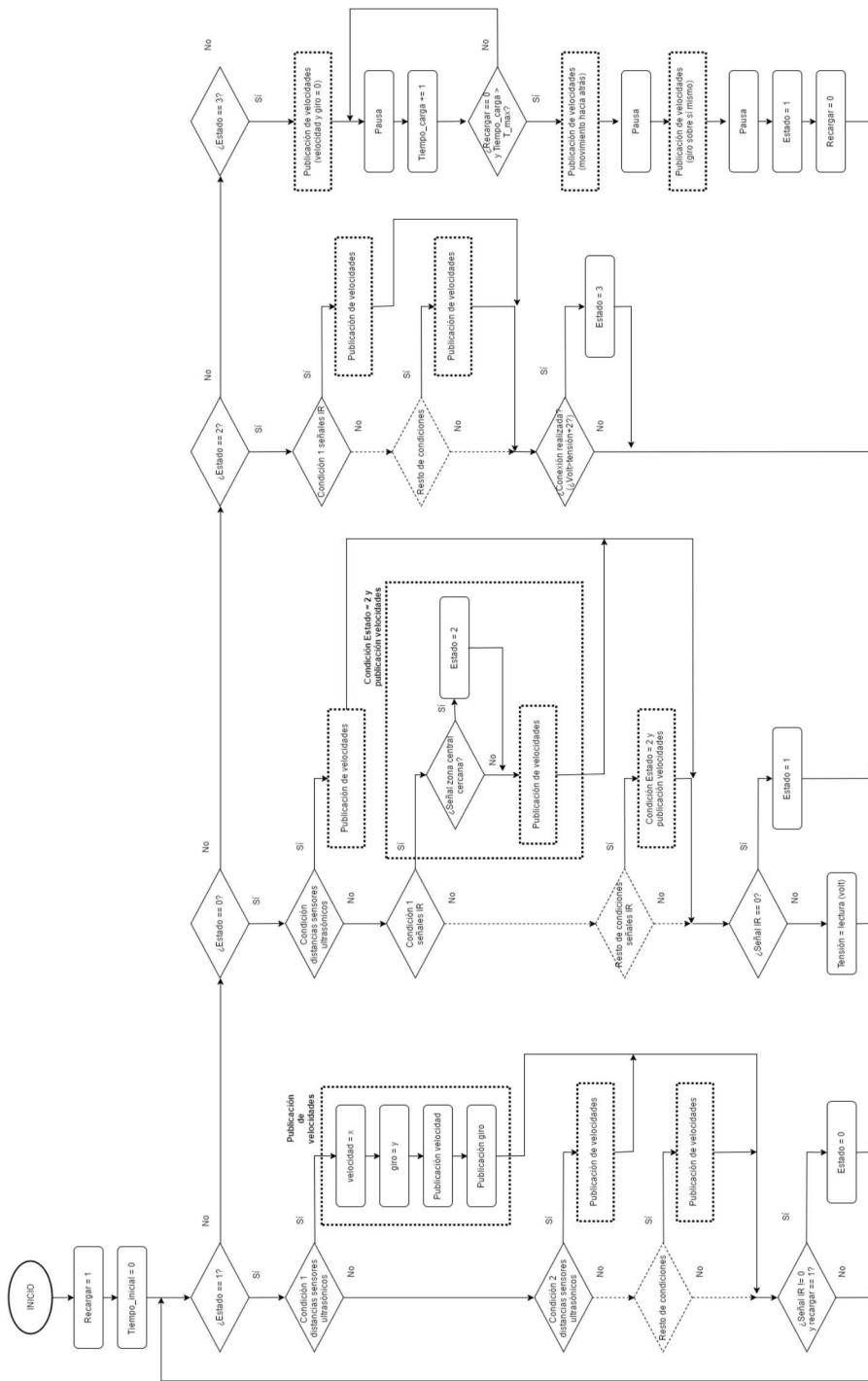


Figura 9.9: Diagrama de flujo del algoritmo principal simplificado.

Capítulo 10

Experimentos y conclusiones

10.1. Introducción

En este último capítulo se realiza una descripción general de los experimentos realizados durante todo el trabajo, abordando desde las pruebas del envío de señales, el control del movimiento del robot y la correcta aproximación y conexión a la estación de carga. También se detallan algunas de las posibles líneas futuras de trabajo, y finalmente se analizan los objetivos cumplidos y se realizan las conclusiones de todo el trabajo realizado.

10.2. Experimentos

Durante la realización ejecución de este proyecto se han realizado un gran número de experimentos para probar diferentes aspectos de todo el sistema y su funcionamiento. En primer lugar, tal y como se explica en el capítulo 6, se realizaron numerosas pruebas con el sistema de emisores infrarrojos programando las tarjetas Arduino que se encargan de la emisión y recepción de las señales. Por un lado, se comprobó el funcionamiento del envío de señales, con distintos patrones de señales y diversos modos para reconocer inequívocamente las señales recibidas. Igualmente se comprobó el correcto funcionamiento con distintos números de emisores y receptores instalados en placas de prueba, analizando las señales tanto con el osciloscopio como mostrando los códigos recibidos en la pantalla del ordenador o usando un led RGB.

Posteriormente se procedió a preparar la plataforma con el nuevo sistema de alimentación y la nueva electrónica que se incorporó. Una vez instalados los nuevos elementos, se programó el Arduino que controla los motores y los sensores ultrasónicos para comprobar el correcto funcionamiento de los diferentes dispositivos, así como de las comunicaciones I2C. En primer lugar se realizaron pruebas para comprobar el funcionamiento de los motores y la controladora MD23, enviando órdenes de movimiento directamente desde Arduino sin control externo del ordenador ni comunicación alguna con ROS. También se comprobó la correcta medición de distancias de los sensores ultrasónicos.

El siguiente paso fue instalar y configurar ROS en el ordenador, creando los archivos necesarios para el correcto funcionamiento. Para comunicar el ordenador con la plataforma fue necesario modificar los programas en las dos tarjetas Arduino para incluir el código que controla las comunicaciones Rosserial. En primer lugar se comprobó la lectura de los sensores ultrasónicos y posteriormente se configuraron los envíos del resto de mensajes. Finalmente se modificó la programación tanto del programa principal en ROS como del Arduino para enviar las consignas de velocidad desde el ordenador a la controladora de los motores. Igualmente se realizaron pruebas del movimiento del robot fijando velocidades desde el programa principal en ROS para comprobar los sentidos de los giros de las ruedas y los correspondientes valores de los encoders. Posteriormente se incluyeron en el programa principal lecturas de los sensores ultrasónicos que determinaban las velocidades y giros del robot, con el fin de realizar las comprobaciones básicas de navegación y detección de obstáculos. En la figura 10.1 se muestra una imagen del visor del metapaquete rqt, que permite ver el listado de tópicos activos en el sistema, así como sus valores.

A continuación se procedió a instalar los receptores en la plataforma y analizar las señales recibidas, que se envían a ROS para calcular el movimiento del robot. Se comenzó realizando pruebas con un único emisor y dos receptores frontales, para los que se diseñaron varios modelos de piezas impresas en plástico donde se instalaban los receptores. Los primeros experimentos se realizaron para comprobar el avance y seguimiento de la señal de uno de los emisores. Posteriormente se fueron incluyendo los otros dos emisores y se comprobaron distintas configuraciones para el algoritmo de control, así como distintos valores de velocidades y giros según las señales recibidas. También se comprobaron distintos sistemas físicos para reducir los campos de visión de los receptores o la amplitud del haz enviado por cada emisor. Finalmente, se optó por la solución de limitar únicamente el haz de los emisores y separar los dos receptores frontales con una barrera entre ellos. Para mejorar la recepción de las señales, también se optó por instalar dos receptores laterales que ayudan a localizar la base de carga desde casi cualquier ángulo. Con la configuración final de 3 emisores y 4 receptores instalados tal y como se explica en el capítulo 6, se procedió a programar y realizar numerosas pruebas en el sistema. Para ello se realizaron numerosas pruebas incluyendo modificaciones únicamente en el programa principal de ROS. Una vez establecidos los algoritmos principales, se procedía a

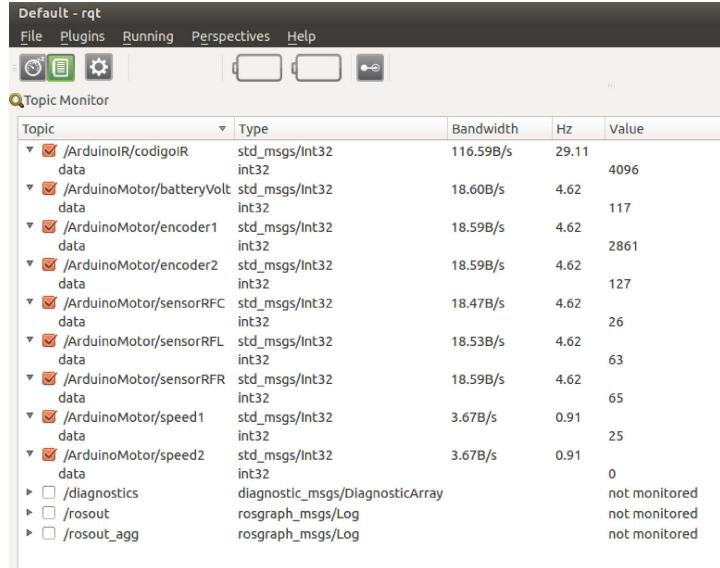


Figura 10.1: Visor rqt que muestra todos los topicos y sus valores.

realizar pruebas de la aproximación del robot a la base, modificando los valores de velocidades de refresco y publicación de mensajes, así como las velocidades de avance y giro. El paso del robot de una zona a otra requiere el ajuste de las velocidades, adaptándose según la distancia a la que se encuentre de la base de recarga. Una vez situado en la región central, el robot realiza en cada avance pequeños giros a uno y otro lado, con lo que el ajuste de los valores de dichos giros y velocidades fueron modificados numerosas veces hasta conseguir un avance suave sin giros bruscos y a su vez, lo más rápido posible para realizar la conexión en el menor tiempo posible. Respecto a las conexiones físicas entre la plataforma y la base de carga, también se estudiaron distintos tipos de contactos, pero finalmente los conectores móviles con los muelles cumplían correctamente con la funcionalidad deseada. Para realizar las pruebas de conexión y desconexión, se programó el robot para que realizara una carga de una determinada duración, y pasado ese tiempo, el robot se desconectaba para seguir cualquier otra trayectoria. Tras realizar numerosas pruebas situando el robot al inicio en una determinada posición de forma que detectara la base desde distintos ángulos, las trayectorias ejecutadas por el robot y tanto la conexión como la desconexión fueron satisfactorias, por lo que se dan por cumplidos todos los objetivos del presente proyecto. En las figuras 10.2 y 10.3 se muestran dos fotografías del robot durante el estado previo a la conexión y durante la recarga.



Figura 10.2: Vista del robot acercándose a la estación de carga.



Figura 10.3: Vista del robot conectado a la estación de carga.

10.3. Líneas futuras de trabajo

Teniendo en cuenta que el robot se basa en una plataforma móvil a la que se le pueden incorporar numerosos dispositivos, tales como sensores o actuadores, las posibles líneas de investigación y trabajo son múltiples. Los sistemas de visión basados en radares, láseres o cámaras 3D permiten multiplicar las funcionalidades del robot, pudiendo implementar tareas de navegación basadas en el SLAM o realizar otras tareas de reconocimiento del entorno. Si además se le incorpora algún tipo de actuador, como puede ser un brazo manipulador, las tareas que podría realizar serían muy distintas y abarcan muchos aspectos diferentes, desde la propia manipulación de objetos, el reconocimiento de estos o aspectos relacionados con la mecánica, la electrónica o incluso la simulación de su funcionamiento.

Respecto a la plataforma, la principal mejora que se podría realizar es la instalación de un ordenador especialmente diseñado para robótica en lugar de usar un ordenador portátil. El uso de un ordenador del tipo Raspberry Pi es

una opción, pero si se quieren implementar tareas más complejas como la navegación SLAM, su poder de procesamiento es demasiado limitado. Un ejemplo de un ordenador mucho más potente para el control de este robot es el modelo JETSON TK1 de Nvidia. Este modelo tiene un procesador Tegra y permite instalar ROS sobre la versión de Ubuntu que trae instalada por defecto. Con la instalación de este tipo de ordenador, se obtendría un gran ahorro de espacio, pudiendo instalar este dispositivo bajo la plataforma y dejando libre la superficie superior del robot para instalar otros dispositivos o utilizarlo incluso para zona de carga. Este dispositivo necesita de alimentación externa, por lo que el sistema de alimentación requeriría de modificaciones para mejorar su capacidad y autonomía. Pero a su vez, esto se convierte en una ventaja, ya que al compartir el sistema de alimentación, únicamente hay que realizar la carga de un sistema de alimentación en lugar de tener que cargar la batería del ordenador de forma independiente. De este modo, haciendo uso del sistema de carga diseñado, se tendría un robot totalmente autónomo en lo que al suministro de energía se refiere.

Finalmente, respecto al sistema de carga que se ha diseñado y desarrollado en este proyecto, las posibles mejoras pueden abarcar todos las áreas que se han estudiado. Por un lado, el sistema de luces infrarrojas y su control puede mejorarse en distintos aspectos. Se pueden estudiar otras soluciones modificando las posiciones de los emisores o receptores, así como mejorando sus algoritmos. En el caso de usar diferentes robots móviles con una única base de carga, o incluso varias bases de carga para un número mayor de robots, sería necesaria la modificación del código de las señales de cada base. Para evitar interferencias entre las distintas bases en el caso de que las señales emitidas por estas se superpongan, cada base tendría que emitir códigos que la identifiquen de forma inequívoca. A su vez, si un robot se encuentra en el proceso de conexión a una determinada base, se podría diseñar algún sistema para que otro robot no pueda detectar la misma base e intentase conectarse al mismo tiempo.

Igualmente la tarjeta Arduino que controla la emisión de las señales se podría sustituir por otra solución aún más barata, usando un microcontrolador, del tipo PIC por ejemplo, que se programe directamente y se instale sobre una placa de circuito impreso. Esto permitiría reducir aún más el tamaño de la base y si los emisores se instalan en la misma placa, el cableado interior sería más simple.

El funcionamiento mecánico de las piezas que se han impreso en plástico ha sido satisfactorio, pero igualmente se podrían realizar mejoras en el diseño y probar distintos sistemas de conectores. Por ejemplo, el sistema de muelles de los conectores podría sustituirse por piezas impresas en plástico que tengan un soporte flexible y móvil en sólo una de las direcciones, ya que si se diseña correctamente, las piezas impresas en plástico no tienen por qué ser completamente rígidas. Respecto a la parte eléctrica, también se pueden realizar algunas mejoras. Aunque las conexiones del robot están protegidas contra sobreintensidades y cortocircuitos por el circuito de protección que se instaló en las baterías, se

podrían añadir diodos en cada uno de los contactos para evitar que el circuito de protección desconecte las baterías en el caso de que el frontal del robot tocarse algún obstáculo metálico o se produzca cualquier otra situación similar. En el caso de la base de carga, se podría realizar la misma modificación para aumentar la seguridad y además se podría diseñar algún sistema que desconecte la alimentación en los contactos cuando el robot no esté en la base. Igualmente se podría instalar un amperímetro que mida la intensidad que está aportando la base al robot, por lo que se podría conocer de esa forma el momento en el que la base deja de alimentar las baterías porque ya se encuentren totalmente cargadas.

Por otro lado, los algoritmos se han diseñado desde cero hasta conseguir cumplir con los objetivos del proyecto. Partiendo de estos algoritmos, se pueden realizar mejoras que permitan realizar la conexión de forma aún más rápida y suave, modificando el sistema de detección de zonas y movimientos asociados a ellas, diseñando un nuevo sistema de estados en los que el robot pueda funcionar, o cualquier otra mejora que pueda incorporarse al algoritmo principal. Si al robot se le instala otro programa que controle el movimiento de cualquier otra forma y con otras funcionalidades extras, deben establecerse las relaciones adecuadas entre ese programa y el que aquí se ha desarrollado, de forma que el nuevo programa principal sea el que decida cuando deja de realizar su planificación del movimiento o cualquier otra tarea para pasar al algoritmo de búsqueda de la base de carga. La condición para pasar al estado de búsqueda para cargar las baterías puede ser el bajo nivel de la carga de las baterías, pero también podría configurarse para que el robot se desplace a la base de carga cuando esté inactivo durante un determinado tiempo.

Otro campo dónde es posible seguir trabajando, más relacionado con el sistema ROS, es la realización del modelado del robot utilizado, creando un modelo que pueda utilizarse para simulaciones en Gazebo o cualquier otro software similar. Este programa es un simulador dinámico 3D que recrea ambientes interiores y exteriores en los que se pueden probar el funcionamiento de robots de forma muy realista. Esto permite diseñar y probar algoritmos en numerosas situaciones distintas sin tener que hacer funcionar el robot en esas condiciones. Además el programa dispone de un gran número de sensores que pueden ser incorporados para estudiar su comportamiento antes de realizar modificaciones físicas.

Por último, aunque el sistema sólo se ha probado en la plataforma móvil descrita en este proyecto, todas las piezas y algoritmos han sido diseñados de forma que la adaptación de este sistema a cualquier otro robot similar sea lo más sencilla posible. Por ello, la última línea de trabajo que aquí se sugiere es la adaptación de este sistema a robots que utilicen un sistema similar de tracción, como puede ser el caso de PIERO. Este robot es más pequeño, más ligero y no está diseñado para trabajar con ROS, pero igualmente se podría probar la parte mecánica y la electrónica de bajo nivel y estudiar las modificaciones necesarias para que un robot de estas características pueda usar el sistema

diseñado. Posteriormente se podría estudiar la adaptación del sistema a robots con otros sistemas de tracción, como puede ser cualquier robot de 4 ruedas.

10.4. Conclusiones

Con este apartado se concluye el último capítulo del presente proyecto. A continuación se describen las principales conclusiones obtenidas tras la realización de todo el trabajo y los resultados obtenidos en relación a los objetivos iniciales del proyecto.

- Se han estudiado diferentes modos de localización en robótica móvil, se han analizado dichos sistemas en el caso de sistemas de recarga y se ha propuesto un sistema para abordar el objetivo principal del proyecto.
- Se ha realizado un estudio sobre los protocolos de la codificación de mensajes mediante señales infrarrojas y se ha diseñado un código propio para el envío de dichas señales.
- Se han implementado los circuitos electrónicos necesarios para el envío de señales infrarrojas y se han realizado experimentos con distintas configuraciones de emisores y receptores.
- Se ha mejorado el sistema de alimentación del robot, sustituyendo la batería de Plomo-ácido por un conjunto de baterías de Ion-Litio y un circuito de protección.
- Se ha hecho uso de un software de CAD y de la impresión 3D para fabricar las piezas necesarias para el desarrollo del sistema de recarga, realizando diferentes experimentos y analizando sus resultados.
- Se han estudiado distintos sistemas de conexión para recargar las baterías del robot y se ha diseñado una estación de carga con todos los elementos y mecanismos necesarios.
- Se ha desarrollado un algoritmo para el control de bajo nivel de los motores del robot móvil y los sensores ultrasónicos usando el protocolo I2C.
- Se han desarrollado algoritmos a bajo nivel para el control de la emisión y recepción de mensajes mediante señales infrarrojas.
- Se ha dotado al robot del control del movimiento mediante el sistema operativo ROS, se ha creado el sistema de archivos necesario y se han estableciendo las comunicaciones entre el ordenador y dos tarjetas Arduino.
- Se ha desarrollado un algoritmo de alto nivel en ROS para la navegación autónoma del robot evitando obstáculos.

- Se ha desarrollado un algoritmo de alto nivel en ROS para realizar la aproximación y conexión del robot móvil a la estación de carga.
- Se han realizado y analizado diferentes experimentos del funcionamiento general del sistema en los que los resultados han sido satisfactorios.

Además de los anteriores conclusiones, otro de los objetivos cumplidos ha sido el uso de herramientas de software libre para realizar todo el trabajo. A pesar de las grandes limitaciones por la utilización de software de este tipo en comparación con software profesional con un coste mucho mayor, los resultados han sido satisfactorios. No obstante, en el caso del diseño de piezas en 3D con software libre, el esfuerzo para abordar el diseño de las piezas ha sido mayor debido a las limitaciones del programa utilizado. Por otro lado, la ventaja es que tanto los diseños de las piezas como los algoritmos y diferentes códigos se puedan compartir libremente a través de internet para que cualquier persona pueda usarlos o modificarlos para adaptarlos a otros sistemas o realizar mejoras.

Las mejoras incluidas en el robot permiten que se puedan realizar numerosos desarrollos futuros, implementando nuevos sistemas de control, añadiendo nuevas funcionalidades o instalando nuevos sensores y actuadores. Además el sistema desarrollado posibilita el ser adaptado a diferentes robots con los que realizar otros estudios.

Durante la realización del proyecto se han publicado distintas entradas escritas en inglés en el blog de la web del laboratorio [35], explicando el desarrollo de algunas partes del proyecto y donde se encuentran disponibles los diseños en 3D y los códigos de todos los algoritmos en los distintos lenguajes utilizados.

En general, la realización de este proyecto me ha permitido afianzar y poner en práctica conocimientos sobre distintos ámbitos de la ingeniería industrial. Un aspecto a destacar es el gran número de disciplinas que el proyecto ha abarcado, destacando la electrónica, la automática, la programación y la mecánica. Entre algunas de las tareas más notables, se puede destacar que he desarrollado circuitos de distinto tipo y he realizado sus montajes usando distintas técnicas y herramientas, he estudiado e implementado sistemas de codificaciones de señales y he aprendido a diseñar piezas en 3D con el software libre FreeCAD y la herramienta online Onshape. Además he usado otros programas para la realización de esquemas y diagramas para ilustrar el presente documento, destacando Fritzing para los esquemas electrónicos y Draw.io para los diagramas de flujos y otros esquemas. Igualmente he mejorado mis conocimientos básicos en L^AT_EX editando la memoria del proyecto con la herramienta online Overleaf y con el editor TEXMaker. Además he aprendido a desarrollar algoritmos y desarrollar código en distintos lenguajes de bajo y alto nivel. Me he familiarizado con el sistema ROS, un software en auge y con un gran potencial en el futuro no sólo en el ámbito académico, sino también en la investigación y en la industria.

Bibliografía

- [1] Academia Testo, *Radiación infrarroja*,
<http://www.academiatesto.com.ar/cms/radiacion-infrarroja-en-el-espectro-de-ondas-2>, Consulta: octubre 2015.
- [2] Arduino, *Arduino UNO*,
<https://www.arduino.cc/en/main/arduinoBoardUno>, Consulta: octubre 2015.
- [3] Arduino, *Tutorials*, <https://www.arduino.cc/en/Tutorial/HomePage>, Consulta: octubre 2015.
- [4] Ayerbe, *Robot aspirador Deepoo*, <http://www.ayerbe.net/productos/robot-aspirador-depoo-d76/>, Consulta: octubre 2015.
- [5] A. Barrientos...[et.al.], *Fundamentos de Robótica*. 2º Ed. Madrid: Mc. Graw Hill, 2007.
- [6] Blog DRK, *Control remoto infrarrojo*,
<http://blog.drk.com.ar/2013/control-remoto-infrarrojo-con-arduino>, Consulta: octubre 2015.
- [7] R. Cassinis, F. Tampalini, P. Bartolini, R. Fedrigotti, *Docking and charging system for autonomous mobile robots*, University of Brescia, Italia.
- [8] Clearpath Robotics, *ROS 101*,
<https://www.clearpathrobotics.com/2014/01/how-to-guide-ros-101/>, Consulta: noviembre 2015.
- [9] Farnell, *IR Receiver TSOP 4838*,
<http://www.farnell.com/datasheets/1693301.pdf>, Consulta: noviembre 2015.
- [10] Fetch Robotics, *Freight*, <http://fetchrobotics.com/freight/>, Consulta: octubre 2015.
- [11] GitHub, *Charging system for mobile robots with ROS*,
<https://github.com/jaimevera>, Consulta: junio 2016.

- [12] Gómez de Gabriel, *PIERO*, <http://gomezdegabriel.com/projects/piero-mobile-robot-platform/>, Consulta: octubre 2015.
- [13] IEEE, *IROS 2015 Conference Digest*, Universität Hamburg, Germany.
- [14] IEEE Spectrum, *iRobot Roomba 980*,
<http://spectrum.ieee.org/automaton/robotics/home-robots/review-irobot-roomba-980>, Consulta: diciembre 2015.
- [15] IEEE Spectrum, *Jibo is as good as social robots get. But is that good enough?*,
<http://spectrum.ieee.org/robotics/home-robots/jibo-is-as-good-as-social-robots-get-but-is-that-good-enough>, Consulta: marzo 2016.
- [16] IEEE Spectrum, *The search for a better battery*,
<http://spectrum.ieee.org/at-work/innovation/the-search-for-a-better-battery>, Consulta: mayo 2016.
- [17] IEEE Spectrum, *Would you trust a robot surgeon to operate on you*,
<http://spectrum.ieee.org/robotics/medical-robots/would-you-trust-a-robot-surgeon-to-operate-on-you>, Consulta: mayo 2016.
- [18] iRobot, *Create 2*,
<http://www.irobot.com/About-iRobot/STEM/Create-2.aspx>, Consulta: octubre 2015.
- [19] iRobot, *Roomba*,
<http://www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx>, Consulta: octubre 2015.
- [20] Lelong, *Rechargeable lithium battery protection board*,
<http://www.lelong.com.my/12v-8a-rechargeable-lithium-battery-18650-protection-board-bee-150182822-2015-05-Sale-P.htm>, Consulta: octubre 2015.
- [21] A. Martín, *Robot móvil con comportamientos sociales*, Proyecto final de carrera. Universidad de Málaga, 2014.
- [22] A. Martínez Romero, E. Fernández, *Learning ROS for Robotics Programming: A Practical, Instructive, and Comprehensive Guide to Introduce Yourself to ROS, the Top-Notch, Leading Robotics Framework* Birmingham, UK: Packt Publishing, 2013.
- [23] OSRF, *Open Source Robots Foundation*, <http://www.osrfoundation.org/>, Consulta: noviembre 2015.
- [24] Port Tarragona, *Manual Sistemas Ayudas Navegación*,
http://www.porttarragona.cat/dmdocuments/Manual_Sistemas_Ayudas_Navegacion.pdf, Consulta: noviembre 2015.

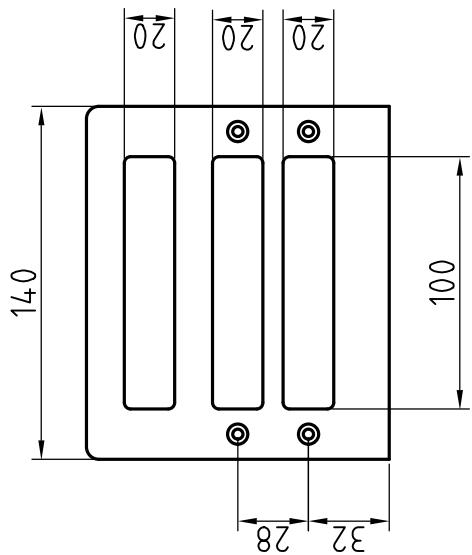
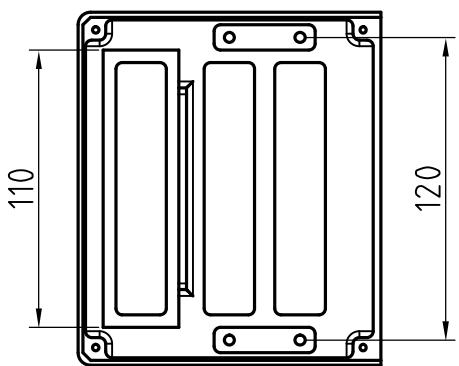
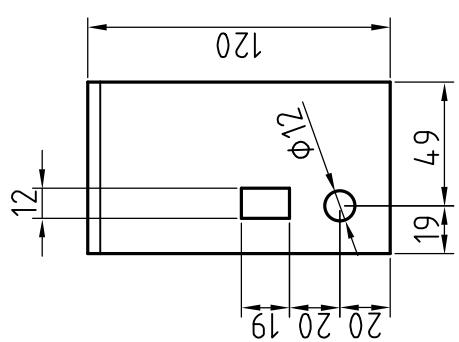
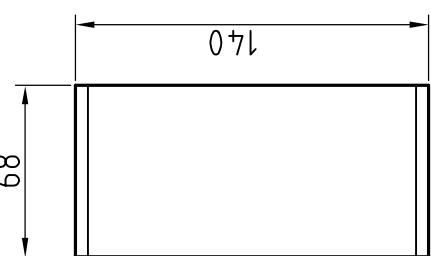
- [25] Cinemática de robots móviles *Prácticas Proyecto PIE 15-180*, Universidad de Málaga
- [26] Prometec, *Bus I2C*, <http://www.prometec.net/bus-i2c/>, Consulta: octubre 2015.
- [27] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, *ROS: an open-source Robot Operating System*, Willow Garage, CA. Stanford University, Stanford, CA.
- [28] Robot electronics, *Motores EMG30*,
<http://www.robot-electronics.co.uk/htm/emg30.htm>, Consulta: octubre 2015.
- [29] Robot electronics, *SRF 08*,
<http://www.robot-electronics.co.uk/htm/srf08tech.html>, Consulta: octubre 2015.
- [30] Robomow, *Robomow RS 612*,
<http://robomow.com/es-ES/shop/mowers/rs-612/>, Consulta: octubre 2015.
- [31] Robot Store HK, *MD23*,
<http://www.robotstorehk.com/motordrivers/doc/md23tech.pdf>, Consulta: octubre 2015.
- [32] Robotix, *Differential drive*,
<https://www.robotix.in/tutorial/mechanical/drivemechtut/>, Consulta: octubre 2015.
- [33] Robotnik, *Plataforma para logística*,
<http://www.robotnik.es/robots-moviles/autonomo-agvs/>, Consulta: octubre 2015.
- [34] B. Siciliano...[et.al.], *Robotics. Modelling, Planning and Control*. London: Springer, 2009.
- [35] Telerobotic and Interactive Systems, Universidad de Málaga, *ROS Projects*, <http://taislab.uma.es/INDEX.PHP/category/ros/>, Consulta: abril 2016.
- [36] Tesla Motors, *Tesla Model S*, <https://www.teslamotors.com/models>, Consulta: abril 2016.
- [37] Trastejant, *Protocolo NEC*,
<http://www.trastejant.es/tutoriales/electronica/ir.html>, Consulta: octubre 2015.
- [38] R. Triviño, *Navegación en edificios públicos mediante SLAM*, Trabajo fin de grado. Universidad de Málaga, 2014.

- [39] TurtleBot, *Learn TurtleBot and ROS*, <http://learn.turtlebot.com/>, Consulta: noviembre 2015.
- [40] Ubuntu, *Ubuntu 14.04.4 LTS (Trusty Tahr)*, <http://releases.ubuntu.com/14.04/>, Consulta: noviembre 2015.
- [41] Vishay, *IR Emitter TSAL 6400*, <http://www.vishay.com/docs/81011/tsal6400.pdf>, Consulta: noviembre 2015.
- [42] J.D. Warren, J. Adams, H. Molle, *Arduino Robotics*. Berkeley, CA: Apress, 2011.
- [43] Wikipedia, *Enfilación náutica*, [https://es.wikipedia.org/wiki/Enfilaci%C3%B3n_\(n%C3%A1utica\)](https://es.wikipedia.org/wiki/Enfilaci%C3%B3n_(n%C3%A1utica)), Consulta: noviembre 2015.
- [44] Wikiros, *ROS Tutorials*, <http://wiki.ros.org/ROS/Tutorials>, Consulta: noviembre 2015.
- [45] Xaxxon, *Oculus Prime*, <http://www.xaxxon.com/documentation/view/oculus-prime-contents>, Consulta: octubre 2015.

ANEXOS

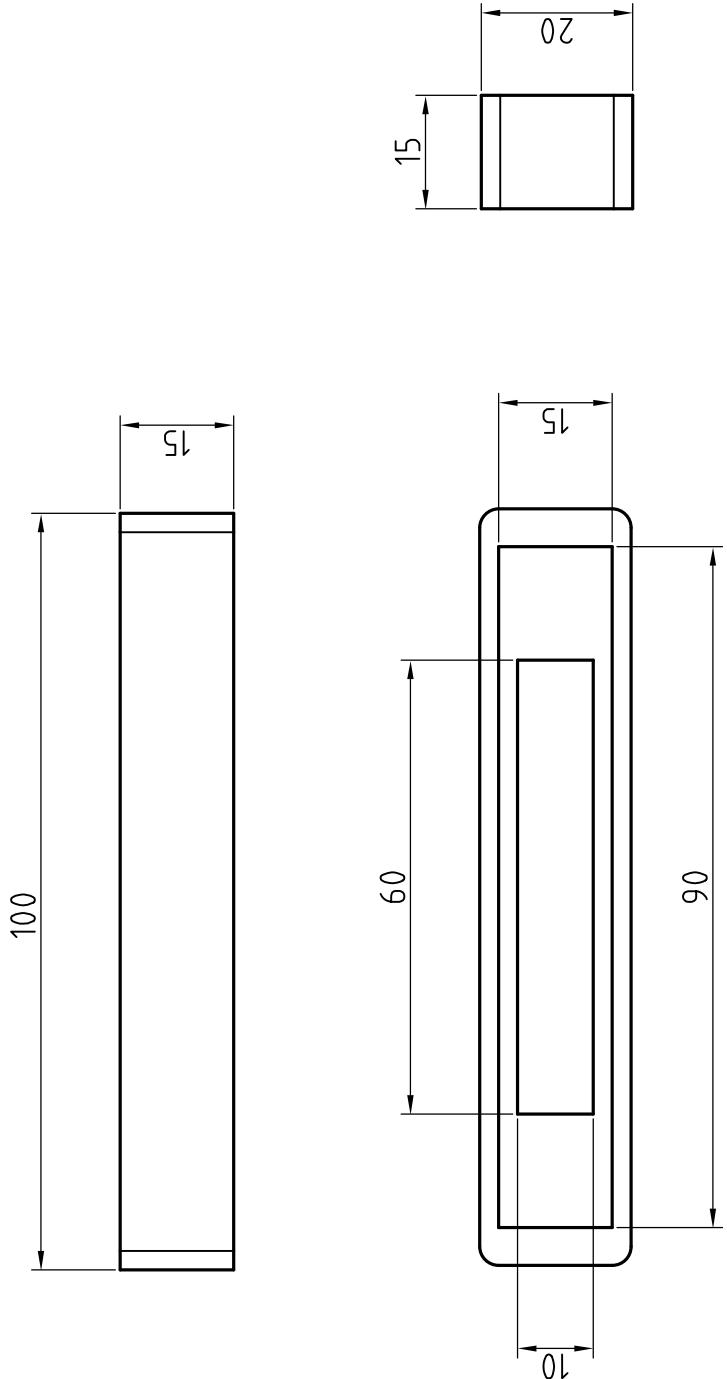
Planos

- Plano n° 1 - Caja estación de carga
- Plano n° 2 - Conector frontal
- Plano n° 3 - Soporte interior
- Plano n° 4 - Receptor frontal
- Plano n° 5 - Receptor lateral



UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS			
DRAWN	NAME Jaime Vera	SIGNATURE	DATE 16/04/2016
CHECKED			
SURFACE FINISH	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE			
BURRS			
FIRST ANGLE PROJECTION	MATERIAL	FINISH	

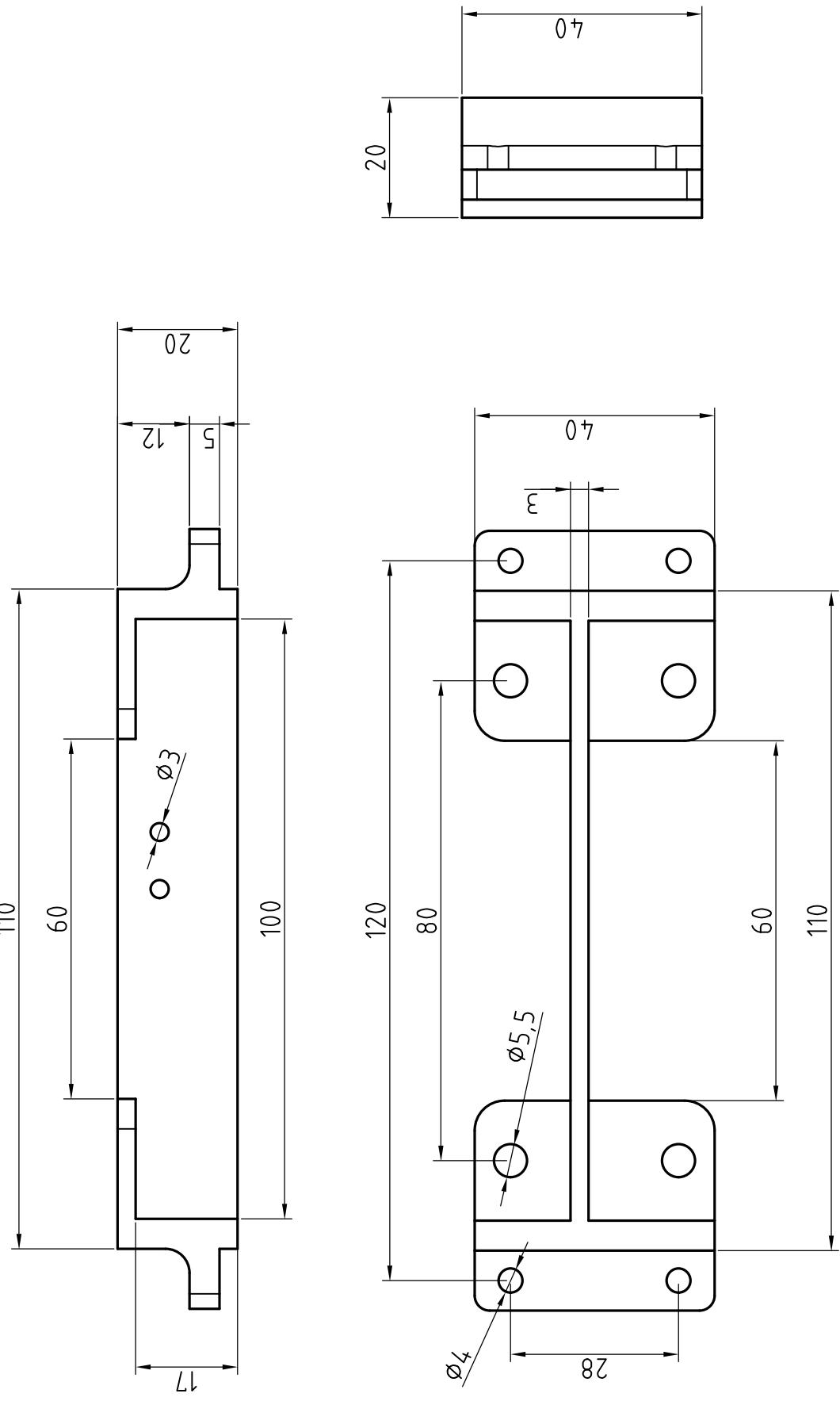
Plano 1	REV.
Caja estación de carga	
A4	DMG NO.
1	1 of 1
SCALE 1:3	WEIGHT
SHEET	



UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS			
DRAWN	NAME Jaime Vera	SIGNATURE	DATE 16/04/2016
CHECKED			TITLE
SURFACE FINISH	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE			
BURRS			
FIRST ANGLE PROJECTION	MATERIAL	FINISH	

SIZE	DRAWING NO.	REV.
A4	2	
SCALE	1:1	WEIGHT
SHEET	1 of 1	

Plano 2
Conector

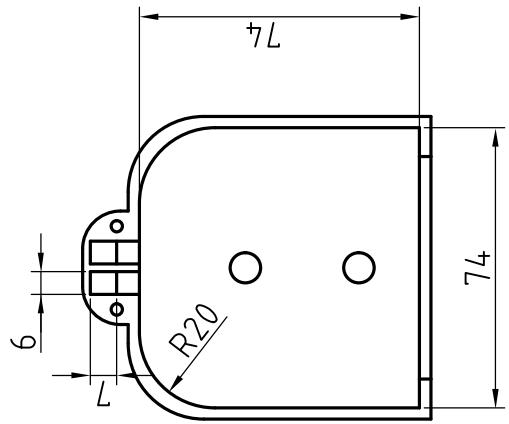
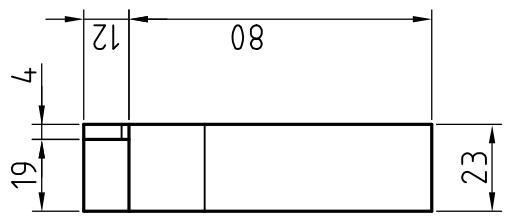
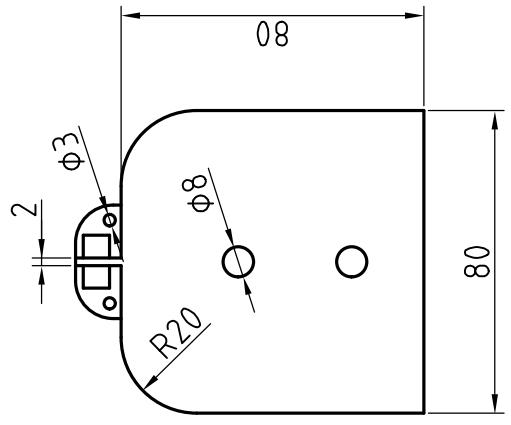
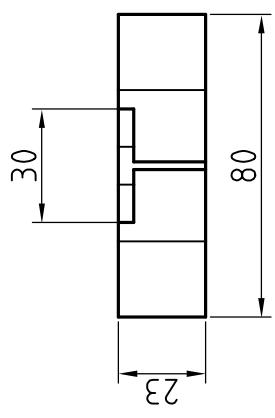


Plano 3

Soporte interior

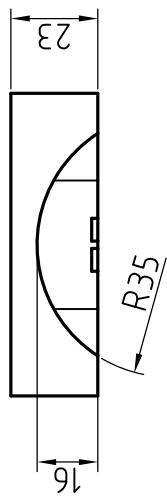
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS			
DRAWN	Jaime Vera	SIGNATURE	DATE
ANGULAR = \pm°			16/04/2016
FRACTIONAL = $\frac{\text{A}}{\text{B}}$			TITLE
CHECKED			
SURFACE FINISH	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE			
BURRS			
FIRST ANGLE PROJECTION	MATERIAL	FINISH	

SIZE	DRWG NO.	SCALE	WEIGHT	SHEET	REV.
A4	3	1:1		1 of 1	

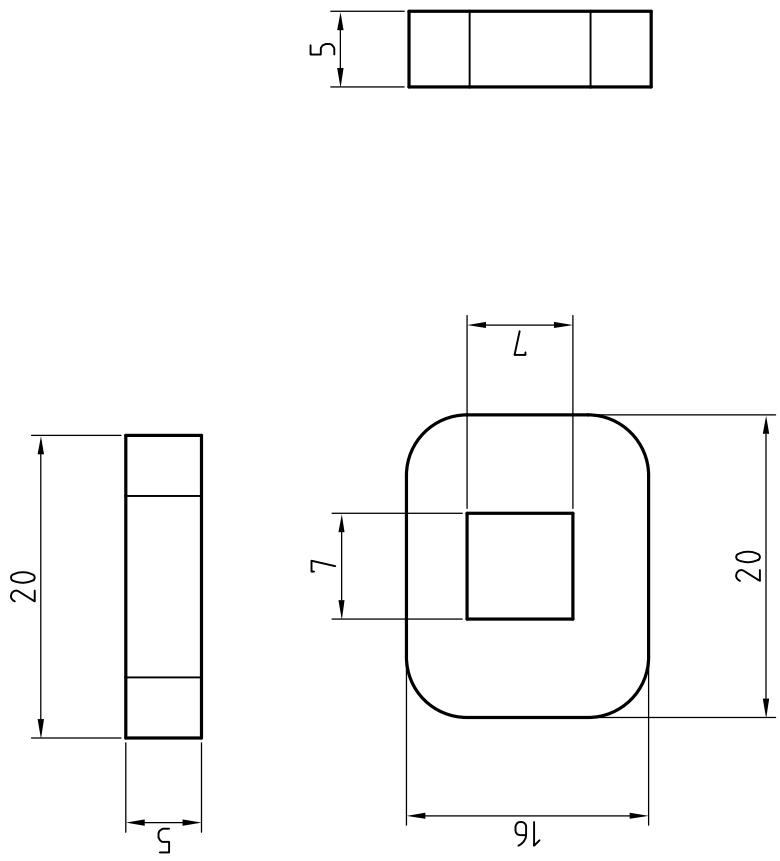


Plano 4
Receptor frontal

UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS			
DRAWN	Jaime Vera	SIGNATURE	DATE
CHECKED			16/04/2016
SURFACE FINISH	APPROVED	TITLE	
DO NOT SCALE DRAWING BREAK ALL SHARP EDGES AND REMOVE BURRS			
FIRST ANGLE PROJECTION	MATERIAL	FINISH	



SIZE	DRAWING NO.	SCALE	WEIGHT	SHEET	REV.
A4	4	1:2		1 of 1	



UNLESS OTHERWISE SPECIFIED,
DIMENSIONS ARE IN MILLIMETERS
ANGULAR = \pm
FRACTIONAL = $\frac{1}{2}$

DRAWN Jaime Vera
CHECKED
TITLE
APPROVED

DO NOT SCALE DRAWING
BREAK ALL SHARP EDGES AND REMOVE
BURRS

FIRST ANGLE PROJECTION
MATERIAL FINISH

SIZE A4 Dwg No. 5
SCALE 2:1 WEIGHT 5
SHEET 1 OF 1 REV 1

Plano 5
Soporte lateral

Anexo A

Código del programa del Arduino que controla los emisores IR

```
1  /* Arduino Code for sending IR messages with 3 IR
   emitters */
2
3 /*
4 PUERTOS
5     Se usan directamente las direcciones de los pines
       correspondientes ,
6     para evitar usar la funcion digital write (que tarda 3
       microsegundos en ejecutarse)
7
8     Digital pin 2 --> PD2 -- Izquierdo
9     Digital pin 3 --> PD3 -- Central
10    Digital pin 4 --> PD4 -- Derecho
11 */
12
13
14 byte portD_Izquierdo = B00000100;      // Declaracion de
   los puertos para los pines de cada emisor
15 byte portD_Central =   B00001000;
16 byte portD_Derecho =   B00010000;
17 byte portD_Todos   =   B00011100;      // Declaracion de
   todos los pines a la vez
18 byte portD_Ninguno =   B00000000;
19
```

```

20
21 void setup() {
22
23     DDRD = DDRD | B11111100; // Se establecen los puertos
24         // 2-7 como salidas
25         // Sin cambiar los valores
26         // del puerto 0 y 1 (RX y TX)
27 }
28 void loop(){
29
30     //Se ejecuta un ciclo con el codigo disenado
31     //total ciclo = 2000+2500+3*1000+2*1500 = 10500
32         microsegundos (sin la pausa final)
33
34     pulseIR(2000);           //Un pulso inicial con
35         todos los leds IR emitiendo
36     delayMicroseconds(2500); //Pausa
37     pulseIRD(1000);        //Emite el led Derecho
38     delayMicroseconds(1500); //Pausa
39     pulseIRC(1000);        //Emite el led Central
40     delayMicroseconds(1500); //Pausa
41     pulseIRI(1000);        //Emite el led Izquierdo
42
43 }
44
45
46 void pulseIR(long microseg){ //Funcion para modular la
47     //senal mientras se produce un pulso en alto para todos
48     //los emisores
49
50     /*38KHz son aprox. 13microseg en HIGH y 13microseg en
51     //LOW
52     la instruccion digitalWrite tarde 3microseg en
53     //ejecutarse
54     por lo que hacemos dos delays de 10 en vez de 13.
55     En total el ciclo dura 26microseg, cuando se completa
56     ,
57     restamos al tiempo que tiene que estar mandando el
58     pulso*/

```

```

55 cli(); //Desabilita cualquier interrupcion
56 while (microseg > 0) {
57
58     PORTD = portD_Todos;           //Pulso en alto ( valor 1) de todos los emisores
59     delayMicroseconds (13);
60     PORTD = portD_Ninguno;        //Pausa (valor 0) de todos los emisores
61     delayMicroseconds (13);
62     microseg -= 26;
63 }
64 sei(); //Activa las interrupciones
65 }
66
67
68
69 void pulseIRD(long microseg){      //Igual funcion anterior , pero solo para el emisor derecho
70
71     cli();
72     while (microseg > 0) {
73         PORTD = portD_Derecho;
74         delayMicroseconds (13);
75         PORTD = portD_Ninguno;
76         delayMicroseconds (13);
77         microseg -= 26;
78     }
79     sei();
80 }
81
82
83 void pulseIRI(long microseg){      //Igual funcion anterior , pero solo para el emisor izquierdo
84
85     cli();
86     while (microseg > 0) {
87         PORTD = portD_Izquierdo;
88         delayMicroseconds (13);
89         PORTD = portD_Ninguno;
90         delayMicroseconds (13);
91         microseg -= 26;
92     }
93     sei();
94 }
95

```

```
96 void pulseIRC(long microseg){           //Igual funcion  
97   anterior , pero solo para el emisor central  
98   cli();  
99   while (microseg > 0) {  
100     PORTD = portD_Central;  
101     delayMicroseconds (13);  
102     PORTD = portD_Ninguno;  
103     delayMicroseconds (13);  
104     microseg -= 26;  
105   }  
106   sei();  
107 }
```

Anexo B

Código del programa del Arduino que controla los receptores IR

```
1 /* Arduino Code for the reception control of IR signals
   and publication of messages in the ROS topic*/
2
3
4 // Mapeo de los pines desde los puertos http://arduino.cc
   /en/Hacking/PinMapping168 ('raw' pin mapping)
5 // Puertos , pines , mascaras , etc: https://www.arduino.cc/
   en/Reference/PortManipulation
6
7
8 #include <ros.h>                      //Libreria ROSSerial
   para comunicacion con ROS
9 #include <std_msgs/Int32.h>           //El mensaje para la
   comunicacion serie con ROS es un Int32
10
11 ros::NodeHandle nh;
12 std_msgs::Int32 test;
13 ros::Publisher p("codigoIR", &test); //Nombre del topic
   en ROS: "codigoIR"
14
15
16 #define IRpin_PIN PIND //Puerto D, que incluye los pines
   digitales 2, 3, 4 y 5 del Arduino UNO. (B00111100)
```

```

17 // Receptor IR Derecho
18     -> pin 2 (B00000100)      (
19         Visto desde el robot/arriba)
20 // Receptor IR Central Derecho
21     -> pin 3 (B00001000)
22 // Receptor IR Central Izquierdo
23     -> pin 4 (B00010000)
24 // Receptor IR Izquierdo
25     -> pin 5 (B00100000)

26
27 #define MAXPULSE 8000 //Tiempo para detectar las pausas
28     entre mensajes consecutivos
29 #define RESOLUTION 20

30 uint8_t currentpulse = 0; //Indice para el pulso que se
31     esta guardando
32 uint8_t num_pulsos = 0; //Numero de pulsos para cada
33     ciclo

34 void setup(void) {
35
36     nh.initNode();
37     nh.advertise(p);
38
39     DDRB = DDRB | B00001110; // Se establecen los puertos
40         1, 2 y 3 como salidas para encender el led RGB.
41         //Pin 9 para led rojo ->
42             PORTB = B00000010;
43             //Pin 10 para led verde ->
44                 PORTB = B00000100;
45                 //Pin 11 para led azul ->
46                     PORTB = B00001000;
47 }

48 void loop(void) {
49
50     int contador_inicial, leer_codigo; //Contador para
51         el tiempo del flanco positivo de inicio de codigo
52         contador_inicial = leer_codigo = 0; //Leer_codigo->
53             variable igual a 1 si se detecta el inicio de codigo
54                 para leerlo a continuacion
55     int contador2=0;

```

```

48
49     int codigo0 = B00000000;           //Variables para
      guardar las lecturas de los receptores
50     long codigo=0;
51     long codigo1=0;
52     long codigo2=0;
53     long codigo3=0;
54
55     long codigo1b=0;
56     long codigo2b=0;
57     long codigo3b=0;
58
59     long codigo4=0;
60
61
62     //Bucle para detectar el pulso inicial en el que los 3
      emisores estan emitiendo a la vez.
63     //Cuando un receptor IR recibe senal , su pin
      correspondiente esta en LOW, cuando recibe senal en
      HIGH
64
65     while ((IRpin_PIN & B00111100)==60) {    //Si ninguno
      de los receptores IR recibe senal , sus pines estan
      en HIGH (IRpin_PIN=B00111100)
66                                         //Luego la
                                         operacion
                                         AND es
                                         igual a
                                         B00111100
                                         (60 en
                                         binario)
67     contador_inicial++;
68     delayMicroseconds(20);
69
70     if ((20*contador_inicial>= 5000*MAXPULSE) && (
      leer_codigo==0)){    //Para publicar un primer
      mensaje (=0) en el caso de no recibir nada, ya que
      en ese caso, no entra en el bucle y no publica
      nada
71     test.data = 4096;
72
      //Se publica el mensaje igual a 0 cada aprox
      5000 posibles lecturas de codigo
73     p.publish( &test );
74     nh.spinOnce();

```

```

75         delay(10);
76         return;
77     }
78 }
79
80 //Si el tiempo durante el que no reciben senal es
81 //mayor que MAXPULSE y ademas se recibe la senal de
82 //algun led IR, entonces leer_codigo=1
83 if ((20*contador_inicial>= MAXPULSE) && ((IRpin_PIN &
84     B00111100)!=60)){
85     while ((IRpin_PIN & B00111100)!=60){
86         contador2++;
87         delayMicroseconds (20);
88
89         leer_codigo = 1;           //Variable para
90             pasar al siguiente estado
91         num_pulsos = 0;
92     }
93 }
94
95
96
97 if (leer_codigo == 1){          //Mientras leer_codigo
98     //=1, se comprueban las senales en tres tiempos
99     //distintos para detectar que senales recibe cada
100    receptor IR
101    //En cada pulso se leen
102    //las entradas de
103    //cada receptor IR (3
104    //bits) y se guardan
105    //para formar un
106    //numero binario de 9
107    //bits formado
108    //por la concatenacion
109    //de las 3 lecturas de
110    //los 3 receptores de
111    //3 bits cada una.
112 }
```



```

124     codigo2b = codigo2 | 4096L;
125     codigo3b = codigo3 | 4096L;
126
127
128     codigo0 = codigo1b | codigo2b;      //Suma de los
129         codigos leidos anteriormente. Operacion OR que
130         devuelve un 1 si en cualquiera de los operandos
131         hay un 1.
132     codigo = codigo0 | codigo3b;        //El codigo final
133         es la suma de los 3 codigos leidos en los
134         distintos pulsos.
135
136
137     leer_codigo=0;
138
139     delay(10);
140
141
142 //CODIGO ENCENDIDO LED RGB (para un solo receptor y
143     usado para realizar comprobaciones del
144     funcionamiento y ajustar frecuencias)
145
146 //    if ((codigo & 10) != 0) {
147 //        //Si recibe senal del led
148 //        derecho, se enciende el led azul
149 //        PORTB = PORTB | B00001000;      //Color Azul
150 //    }
151 //    if ((codigo & 100000) != 0) {
152 //        //Si recibe senal del led
153 //        central, se enciende el led verde
154 //        PORTB = PORTB | B00000100;      //Color Verde
155 //    }
156 //    if ((codigo & 100000000) != 0) {
157 //        //Si recibe senal del led
158 //        izquierdo, se enciende el led rojo
159 //        PORTB = PORTB | B00000010;      //Color Rojo
160 //    }
161
162
163

```

```
//En el caso de que reciba 2 o las 3 senales , se  
154 //      encienden a la vez dando lugar a los colores  
      secundarios  
154 //      delay(10);           //Tiempo de espera para  
      mostrar led  
155 //      PORTB = B00000000;    //Reinicia los valores  
      del led RGB a 0  
156 //  
157  
158     return ;  
159  
160  
161 }  
162 }
```

Anexo C

Código del programa del Arduino que controla la tarjeta MD23 y los sensores ultrasónicos

```
1 /*Arduino code for MD23 and RFR08 control , using the wire  
and ROSSerial librarys*/  
2  
3 #include <Wire.h> //Libreria necesaria  
para la comunicacion mediante el protocolo I2C  
4 #include <ros.h> //Libreria para la  
comunicacion con ROS  
5 #include <std_msgs/Int32.h> //Libreria para el  
tipo de mensaje intercambiado con ROS  
6  
7 ros::NodeHandle nh; //Inicializa y  
establece las conexiones con los nodos  
correspondientes  
8 std_msgs::Int32 enc_1;  
9 ros::Publisher pe1("encoder1", &enc_1); //Declaracion de  
los topicos en los que se publica desde Arduino.  
Nombre del topic en ROS: "encoder1"  
10 std_msgs::Int32 enc_2;  
11 ros::Publisher pe2("encoder2", &enc_2);  
12 std_msgs::Int32 batt;  
13 ros::Publisher pbv("batteryVolt", &batt);
```

```

14 std_msgs::Int32 RF_L;
15 ros::Publisher pL("sensorRFL", &RF_L);
16 std_msgs::Int32 RF_C;
17 ros::Publisher pC("sensorRFC", &RF_C);
18 std_msgs::Int32 RF_R;
19 ros::Publisher pR("sensorRFR", &RF_R);
20
21
22
23
24 int x = 0;
25 int y = 0;
26 int reading=0;
27
28 void messageCb1(const std_msgs::Int32& msg1){           // Establecen el tipo de mensajes que se reciben de ROS para los topicos de velocidades
29     x=msg1.data;
30 }
31
32 void messageCb2(const std_msgs::Int32& msg2){
33     y=msg2.data;
34 }
35
36 ros::Subscriber<std_msgs::Int32> sub1("speed1", &messageCb1); //Establecen los nombres de los topicos a los que se subscribe Arduino y el tipo de mensaje anteriormente definido
37
38 ros::Subscriber<std_msgs::Int32> sub2("speed2", &messageCb2);
39
40
41 #define md23Address 89                                // Direccion de la tarjeta MD23
42
43
44
45 #define softwareReg 13    //0x0D      // Byte to read the software version
46 #define speed1 0   //0x00          // Byte to send speed to first motor

```

```

47 #define speed2 1 //0x01           // Byte to send
      speed to second motor
48 #define cmdByte 16 //0x10         // Command byte
49 #define encoderOne 2 //0x02       // Byte to read
      motor encoder 1
50 #define encoderTwo 6 //0x06       // Byte to read
      motor encoder 2
51 #define voltRead 10 //0x0A        // Byte to read
      battery volts
52
53
54 #define modeOp 15 //Modo
      operacion.
55
56
57 void setup() {                  //Funcion inicial , se
      ejecuta al principio
58
59
60 nh.initNode();                 //Inicializa los nodos y
      la publicacion de los diferentes topicos.
61 nh.advertise(pe1);
62 nh.advertise(pe2);
63 nh.advertise(pbv);
64
65 nh.advertise(pL);
66 nh.advertise(pC);
67 nh.advertise(pR);
68
69 nh.subscribe(sub1);           //Inicializa la
      subscripcion a los mensajes con las velocidades para
      los motores.
70 nh.subscribe(sub2);
71
72
73 Wire.begin();                  //Inicializa las
      comunicaciones I2C.
74
75
76 Wire.beginTransmission(md23Address); // Establece el modo de
      funcionamiento de la MD23 en modo 3.
77 Wire.write(modeOp);
78 Wire.write(3);
79 Wire.endTransmission();
80

```

```

81     encodeReset();      // Llamada a la funcion que resetea
82     los valores de los encoders.
83 }
84
85
86
87 void loop(){           //Bucle principal
88
89
90
91     RF_C.data = lectura(127);    //Lectura sensor RFR08 (
92         Central , con direccion 127)
93     pC.publish( &RF_C );
94     nh.spinOnce();
95
96     RF_L.data = lectura(113);    //Lectura sensor RFR08 (
97         Izquierdo , con direccion 113)
98     pL.publish( &RF_L );
99     nh.spinOnce();
100
101     RF_R.data = lectura(114);    //Lectura sensor RFR08 (
102         Derecho , con direccion 114)
103     pR.publish( &RF_R );
104     nh.spinOnce();
105
106
107
108     Wire.beginTransmission(md23Address);
109             // Avance del robot con la
110             velocidad x
111     Wire.write(speed1);
112     Wire.write(x);
113     Wire.endTransmission();
114
115     Wire.beginTransmission(md23Address);
116             // Giro del robot con la
117             velocidad y
118     Wire.write(speed2);
119     Wire.write(y);
120     Wire.endTransmission();
121
122
123
124     encoder1();                //Llamada a las funciones
125     que realizan las lecturas en los registros de
126     encoders y tension , y que realizan que publican
127     sus valores en el topico correspondiente

```

```

116     encoder2();
117     volts();
118 }
119
120 int getSoft() {
121     // Funcion para obtener la version del software
122     Wire.beginTransmission(md23Address);
123     Wire.write(softwareReg);
124     Wire.endTransmission();
125     Wire.requestFrom(md23Address, 1);
126     while(Wire.available() < 1);
127     int software = Wire.read();
128
129     return(software);
130 }
131
132 void encodeReset() {
133     // Funcion
134     para resetear los encoders
135     Wire.beginTransmission(md23Address);
136     Wire.write(cmdByte);
137     Wire.write(32);
138     Wire.endTransmission();
139 }
140
141 void encoder1() {
142     // Funcion
143     para leer el registro con el valor del encoder 1
144     Wire.beginTransmission(md23Address);
145     Wire.write(encoderOne);
146     Wire.endTransmission();
147
148     Wire.requestFrom(md23Address, 4);
149     while(Wire.available() < 4);
150     long firstByte = Wire.read();
151     long secondByte = Wire.read();
152     long thirdByte = Wire.read();
153     long fourthByte = Wire.read();
154
155     long poss1 = (firstByte << 24) + (secondByte << 16) +
156                 (thirdByte << 8) + fourthByte;

```

```

155 enc_1.data =poss1;
156 pe1.publish( &enc_1 );
157                                     // Publicacion en
158                                     el topico encoder1 con el valor del encoder1
159 nh.spinOnce();
160 }
161 void encoder2(){
162                                     // Funcion
163                                     igual a la anterior para el encoder2
164 Wire.beginTransmission(md23Address);
165 Wire.write(encoderTwo);
166 Wire.endTransmission();
167
168 Wire.requestFrom(md23Address, 4);
169 while(Wire.available() < 4);
170 long firstByte = Wire.read();
171 long secondByte = Wire.read();
172 long thirdByte = Wire.read();
173 long fourthByte = Wire.read();
174
175 long poss2 = (firstByte << 24) + (secondByte << 16) +
176             (thirdByte << 8) + fourthByte;
177
178 enc_2.data =poss2;
179 pe2.publish( &enc_2 );
180 nh.spinOnce();
181 }
182 void volts(){
183                                     //
184                                     Funcion para realizar la lectura de la tension
185 Wire.beginTransmission(md23Address);
186 Wire.write(voltRead);
187 Wire.endTransmission();
188
189 Wire.requestFrom(md23Address, 1);
190 while(Wire.available() < 1);
191 int batteryVolts = Wire.read();
192
193 batt.data =batteryVolts;
194                                     // Publicacion en el
195                                     topico batteryVolts con el valor de la tension
196 pbv.publish( &batt );

```

```

192 nh.spinOnce();
193 }
194
195 void stopMotor() {
196     // Funcion
197     para parar los motores
198     Wire.beginTransmission(md23Address);
199     Wire.write(speed2);
200     Wire.write(0);
201     Wire.endTransmission();
202
203     Wire.beginTransmission(md23Address);
204     Wire.write(speed1);
205     Wire.write(0);
206     Wire.endTransmission();
207
208
209
210
211 int lectura (int address) {
212     //Funcion para las
213     lecturas de los sensores ultrasonicos RFR08
214
215     int reading;
216
217     // step 1: instruct sensor to read echoes
218     Wire.beginTransmission(address); // transmit to device
219     #112 (0x70)
220     // the address specified in the datasheet is 224 (0xE0)
221     // but i2c addressing uses the high 7 bits so it's 112
222     Wire.write(byte(0x00)); // sets register pointer
223     to the command register (0x00)
224     Wire.write(byte(0x51)); // command sensor to
225     measure in "inches" (0x50)
226     // use 0x51 for centimeters
227     // use 0x52 for ping microseconds
228     Wire.endTransmission(); // stop transmitting
229
230     // step 2: wait for readings to happen
231     delay(70); // datasheet suggests at
232     least 65 milliseconds
233
234     // step 3: instruct sensor to return a particular echo
235     reading

```

```
229  Wire.beginTransmission(address); // transmit to device
      #112
230  Wire.write(byte(0x02));        // sets register pointer
      to echo #1 register (0x02)
231  Wire.endTransmission();       // stop transmitting
232
233 // step 4: request reading from sensor
234 Wire.requestFrom(address, 2);   // request 2 bytes
      from slave device #112
235
236 // step 5: receive reading from sensor
237 if (2 <= Wire.available()) { // if two bytes were
      received
238     reading = Wire.read(); // receive high byte (
      overwrites previous reading)
239     reading = reading << 8; // shift high byte to be
      high 8 bits
240     reading |= Wire.read(); // receive low byte as lower
      8 bits
241 }
242
243 return reading;
244 }
```

Anexo D

Código del archivo package.xml

```
1 <?xml version="1.0"?>
2 <package>
3   <name>pack1</name>
4   <version>1.0.0</version>
5   <description>The pack1 package: Control of the charging
       dock system for mobile robots</description>
6
7   <maintainer email="jaimeveraduran@gmail.com">Jaime Vera
       Duran</maintainer>
8
9
10  <!-- One license tag required , multiple allowed , one
      license per tag -->
11  <!-- Commonly used license strings: -->
12  <!-- "TODO" BSD, MIT, Boost Software License , GPLv2,
      GPLv3, LGPLv2.1 , LGPLv3 -->
13  <license>GPLv3</license>
14
15  <!-- <url type="repository">github.com</url> -->
16  <!-- <url type="website">http://wiki.ros.org/pack1</url
       > -->
17  <url type="website">http://taislab.uma.es/</url>
18
19  <author email="jaimeveraduran@gmail.com">Jaime Vera
       Duran</author>
20
```

```

21 <!-- The *_depend tags are used to specify dependencies
22   -->
23 <!-- Dependencies can be catkin packages or system
24   dependencies -->
25 <!-- Examples: -->
26 <!-- Use build_depend for packages you need at compile
27   time: -->
28 <build_depend>message_generation</build_depend>
29 <!-- Use buildtool_depend for build tool packages: -->
30 <!-- <buildtool_depend>catkin</buildtool_depend> -->
31 <!-- Use run_depend for packages you need at runtime:
32   -->
33 <run_depend>message_runtime</run_depend>
34 <!-- Use test_depend for packages you need only for
35   testing: -->
36 <!-- <test_depend>gtest</test_depend> -->
37 <buildtool_depend>catkin</buildtool_depend>
38 <build_depend>roscpp</build_depend>
39 <build_depend>rospy</build_depend>
40 <build_depend>std_msgs</build_depend>
41 <run_depend>roscpp</run_depend>
42 <run_depend>rospy</run_depend>
43 <run_depend>std_msgs</run_depend>
44
45 <!-- The export tag contains other, unspecified, tags
46   -->
47 <export>
48   <!-- Other tools can request additional information
49     be placed here -->
50
51 </export>
52 </package>

```

Anexo E

Código del archivo CMakeList.txt

```
1 cmake_minimum_required(VERSION 2.8.3)
2 project(pack1)
3
4 ## Find catkin macros and libraries
5 ## if COMPONENTS list like find_package(catkin REQUIRED
6 ## COMPONENTS xyz)
7 ## is used, also find other catkin packages
7 find_package(catkin REQUIRED COMPONENTS
8   roscpp
9   rospy
10  std_msgs
11  message_generation
12 )
13
14 ## System dependencies are found with CMake's conventions
15 # find_package(Boost REQUIRED COMPONENTS system)
16
17
18 ## Uncomment this if the package has a setup.py. This
19 ## macro ensures
20 ## modules and global scripts declared therein get
21 ## installed
22 ## See http://ros.org/doc/api/catkin/html/user_guide/
22 ## setup_dot_py.html
22 # catkin_python_setup()
```

```

23 ##### Declare ROS messages, services and actions #####
24 ## Declare ROS messages, services and actions ##
25 #####
26
27 ## To declare and build messages, services or actions
   from within this
28 ## package, follow these steps:
29 ## * Let MSG_DEP_SET be the set of packages whose message
   types you use in
30 ##   your messages/services/actions (e.g. std_msgs,
   actionlib_msgs, ...).
31 ## * In the file package.xml:
32 ##   * add a build_depend tag for "message-generation"
33 ##   * add a build_depend and a run_depend tag for each
   package in MSG_DEP_SET
34 ##   * If MSG_DEP_SET isn't empty the following
   dependency has been pulled in
35 ##     but can be declared for certainty nonetheless:
36 ##       * add a run_depend tag for "message_runtime"
37 ## * In this file (CMakeLists.txt):
38 ##   * add "message-generation" and every package in
   MSG_DEP_SET to
39 ##     find_package(catkin REQUIRED COMPONENTS ...)
40 ##   * add "message_runtime" and every package in
   MSG_DEP_SET to
41 ##     catkin_package(CATKIN_DEPENDS ...)
42 ##   * uncomment the add_*_files sections below as needed
43 ##     and list every .msg/.srv/.action file to be
   processed
44 ##   * uncomment the generate_messages entry below
45 ##   * add every package in MSG_DEP_SET to
   generate_messages(DEPENDENCIES ...)

46
47 ## Generate messages in the 'msg' folder
48 # add_message_files(
49 #   FILES
50 #   Message1.msg
51 #   Message2.msg
52 # )
53
54 ## Generate services in the 'srv' folder
55 add_service_files(
56   FILES
57   Service1.srv
58 )
59

```

```

60 ## Generate actions in the 'action' folder
61 # add_action_files(
62 #   FILES
63 #   Action1.action
64 #   Action2.action
65 # )
66
67 ## Generate added messages and services with any
68 ## dependencies listed here
68 generate_messages(
69   DEPENDENCIES
70   std_msgs
71 )
72
73 #####
74 ## Declare ROS dynamic reconfigure parameters ##
75 #####
76
77 ## To declare and build dynamic reconfigure parameters
77 ## within this
78 ## package, follow these steps:
79 ## * In the file package.xml:
80 ##   * add a build_depend and a run_depend tag for "
80 ##     dynamic_reconfigure"
81 ## * In this file (CMakeLists.txt):
82 ##   * add "dynamic_reconfigure" to
83 ##     find_package(catkin REQUIRED COMPONENTS ... )
84 ##   * uncomment the "
84 ##     generate_dynamic_reconfigure_options" section below
85 ##     and list every .cfg file to be processed
86
87 ## Generate dynamic reconfigure parameters in the 'cfg'
87 ## folder
88 # generate_dynamic_reconfigure_options(
89 #   cfg/DynReconf1.cfg
90 #   cfg/DynReconf2.cfg
91 # )
92
93 #####
94 ## catkin specific configuration ##
95 #####
96 ## The catkin_package macro generates cmake config files
96 ## for your package
97 ## Declare things to be passed to dependent projects
98 ## INCLUDE_DIRS: uncomment this if you package contains
98 ## header files

```

```

99 ## LIBRARIES: libraries you create in this project that
100    ## dependent projects also need
101 ## CATKIN_DEPENDS: catkin_packages dependent projects
102    ## also need
103 ## DEPENDS: system dependencies of this project that
104    ## dependent projects also need
105 catkin_package(
106 #  INCLUDE_DIRS include
107 #  LIBRARIES pack1
108 #  CATKIN_DEPENDS roscpp rospy std_msgs
109 #  DEPENDS system_lib
110 )
111 #####
112 #### Build ####
113 #####
114 ## Specify additional locations of header files
115 ## Your package locations should be listed before other
116 ## locations
117 # include_directories(include)
118 include_directories(
119     ${catkin_INCLUDE_DIRS}
120 )
121 ## Declare a C++ library
122 # add_library(pack1
123 #   src/${PROJECT_NAME}/pack1.cpp
124 # )
125 ## Add cmake target dependencies of the library
126 ## as an example, code may need to be generated before
127 ## libraries
128 ## either from message generation or dynamic reconfigure
129 # add_dependencies(pack1 ${${PROJECT_NAME}
130                   _EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
131 ## Declare a C++ executable
132 # add_executable(pack1_node src/pack1_node.cpp)
133 ## Add cmake target dependencies of the executable
134 ## same as for the library above
135 # add_dependencies(pack1_node ${${PROJECT_NAME}
136                   _EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

```

```

137 ## Specify libraries to link a library or executable
138 # target_link_libraries(pack1_node
139 #   ${catkin_LIBRARIES}
140 # )
141 #####
142 ## Install ##
143 #####
144 #####
145
146 # all install targets should use catkin DESTINATION
147 #   variables
148 # See http://ros.org/doc/api/catkin/html/adv\_user\_guide/variables.html
149 #####
150 ## Mark executable scripts (Python etc.) for installation
151 ## in contrast to setup.py, you can choose the
152 #   destination
153 # install(PROGRAMS
154 #   scripts/my-python-script
155 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
156 # )
157 #####
158 ## Mark executables and/or libraries for installation
159 # install(TARGETS pack1 pack1_node
160 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
161 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
162 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
163 # )
164 #####
165 ## Mark cpp header files for installation
166 # install(DIRECTORY include/${PROJECT_NAME}/
167 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
168 #   FILES_MATCHING PATTERN "*.h"
169 #   PATTERN ".svn" EXCLUDE
170 # )
171 #####
172 ## Mark other files for installation (e.g. launch and bag
173 #   files, etc.)
174 # install(FILES
175 #   # myfile1
176 #   # myfile2
177 #   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
178 # )
179 #####

```

```
178 ## Testing ##
179 #####
180
181 ## Add gtest based cpp test target and link libraries
182 # catkin_add_gtest(${PROJECT_NAME}-test test/test_pack1.
183 #   cpp)
184 # if(TARGET ${PROJECT_NAME}-test)
185 #   target_link_libraries(${PROJECT_NAME}-test ${{
186 #     PROJECT_NAME}})
187 # endif()
188
189 ## Add folders to be run by python nosetests
190 # catkin_add_nosetests(test)
```

Anexo F

Código del archivo control.launch

```
1 <launch>
2
3   <group ns="ArduinoMotor">
4     <node pkg="rosserial_python" name="serial" type="
5       serial_node.py" args="/dev/ttyUSB0"
6   </group>
7
8   <group ns="ArduinoIR">
9     <node pkg="rosserial_python" name="serial" type="
10    serial_node.py" args="/dev/ttyACM0"
11 </group>
12
13 <node name="listener1" pkg="pack1" type="listener.py"
14   output="screen" />
15
16 </launch>
```

Anexo G

Código del archivo listener.py

```

1 #!/usr/bin/env python
2 # coding=utf-8
3
4
5 import rospy
6 import time #libreria para las pausas
7
8 from std_msgs.msg import Int32 #libreria para los
9     mensajes enviados.
10
11 codeL = [0,0,0] #codeL guarda el codigo recibido
12     por el receptor izquierdo (visto desde el robot).
13 codeLC = [0,0,0] #codeLC guarda el codigo recibido
14     por el receptor izquierdo central.
15 codeRC = [0,0,0] #codeRC guarda el codigo recibido
16     por el receptor derecho central.
17 codeR = [0,0,0] #codeR guarda el codigo recibido
18     por el receptor derecho.
19
20 codigo = 0b10000000000000 #valores iniciales para los
21     mensajes de las senales IR.
22 code = 0b10000000000000
23
24 distanciaC=1 #valores iniciales para las
25     distancias de los sensores ultrasonicos en cm.

```

```

19 | distanciaR=1
20 | distanciaL=1
21 | volt=0
22 | tension=14           #Variable para guardar el valor
   | de la tension antes de la conexion a la base y
   | realizar la comparacion.
23 | #recargar = 1         #copiada mas abajo #Variable que
   | indica si el robot necesita recarga (=1) o si no la
   | necesita (=0).
24 |
25 | tiempo_recarga = 0
26 |
27 |
28 | giro=0                #guarda el sentido del giro que
   | realiza.
29 | encoder_inicio = 0     #valor para la lectura del
   | encoder.
30 |
31 | ##Funciones que declaran las lecturas que se realizan de
   | las tarjetas Arduino.
32 | ##Cada una de ellas corresponde a un topico.
33 |
34 | def callbackC(data):      #Lecturas de las
   | distancias de los sensores ultrasonicos.
35 |     global distanciaC
36 |     distanciaC=data.data
37 |
38 | def callbackR(data):
39 |     global distanciaR
40 |     distanciaR=data.data
41 |
42 | def callbackL(data):
43 |     global distanciaL
44 |     distanciaL=data.data
45 |
46 | def callbackEnc1(data):    #Lecturas de los encoders
47 |
48 |     global encoder1
49 |     encoder1 = data.data
50 |
51 | def callbackEnc2(data):
52 |     global encoder2
53 |     encoder2 = data.data
54 |
55 | def callbackVolt(data):    #Lectura de la tension.
   |     global volt

```

```

56     volt = data.data
57     rospy.loginfo("## Battery Volt %", data.data)
58
59 def callback(data):           #Lectura del codigo con
60     la informacion de todos los receptores IR.
61     global codigo
62     rospy.loginfo("## IR data %", bin(data.data))      #
63     Funcion para ver en pantalla los mensajes
64     recibidos.
65
66
67
68     codigo = data.data          #data.data es un
69     binario de 9 bits.
70     code = 0b1000000000000 | data.data #se anade un 1
71     delante para tener siempre el mismo tamano (ahora
72     3*4+1=13 bits).
73
74
75
76     ## Orden Onda emisores: Derecho – Central – Izquierdo
77     ##CODIGO EN EL SKETCH DE ARDUINO:
78     #//Puerto D, que incluye los pines digitales 2,
79     #      3, 4 y 5 del Arduino UNO. (B00111100)
80     #           //Receptor IR Derecho
81     #           -> pin 2 (B00000100)           "visto desde el
82     #           robot"
83     #           //Receptor IR Central Derecho
84     #           -> pin 3 (B00001000)
85     #           //Receptor IR Central Izquierdo
86     #           -> pin 4 (B00010000)
87     #           //Receptor IR Izquierdo
88     #           -> pin 5 (B00100000)
89
90
91
92     ##En las siguientes lineas se guardan las lecturas de
93     #cada sensor en un vector separado, tomadas del numero
94     #binario de 13 bits.
95
96     ##Comienza en la posicion 3 porque la 0,1 y 2 son las
97     #correspondientes a 0b1 del numero binario recibido ,
98     #parte que no nos interesa .
99     codeL[0]= bin(code)[3]
100    codeL[1]= bin(code)[7]
101    codeL[2]= bin(code)[11]
102
103    codeLC[0]= bin(code)[4]
104    codeLC[1]= bin(code)[8]
105    codeLC[2]= bin(code)[12]

```

```

86     codeRC[0]= bin(code)[5]
87     codeRC[1]= bin(code)[9]
88     codeRC[2]= bin(code)[13]
89
90     codeR[0]= bin(code)[6]
91     codeR[1]= bin(code)[10]
92     codeR[2]= bin(code)[14]
93
94
95
96 def listener():
97
98
99     rospy.init_node('listener', anonymous=True)
100                #Se inicializa el nodo.
101
102 ##Se establecen los topics en los que se publican y el
103      tipo de mensaje que se envia desde este nodo. En este
104      caso, las dos velocidades enviadas a la controladora
105      de los motores.
106
107 ##Se establecen los topics a los que se subscribe este
108      nodo, correspondientes a los declarados en las
109      funciones callback anteriores.
110
111     rospy.Subscriber("ArduinoIR/codigoIR", Int32,
112                         callback)
113     rospy.Subscriber("ArduinoMotor/sensorRFC", Int32,
114                         callbackC)
115     rospy.Subscriber("ArduinoMotor/sensorRFR", Int32,
116                         callbackR)
117     rospy.Subscriber("ArduinoMotor/sensorRFL", Int32,
118                         callbackL)
119
120     rospy.Subscriber("ArduinoMotor/encoder1", Int32,
121                         callbackEnc1)
122     rospy.Subscriber("ArduinoMotor/encoder2", Int32,
123                         callbackEnc2)
124     rospy.Subscriber("ArduinoMotor/batteryVolt", Int32,
125                         callbackVolt)

```

```

117
118     lost=1                      #Variable que
119         indica el estado del robot.
120     recargar=1                  #Indica si
121         necesita recarga de la bateria.
122     rate = rospy.Rate(10)        #Indica la tasa
123         de refresco en Hz.
124
125
126     while not rospy.is_shutdown():    #Mientras el nodo
127         este funcionando.
128         time.sleep(1) #Pausa en segundos.
129
130     print "----Distancia C:      %s" % distanciaC
131         #Se imprimen en pantalla algunos de
132         los mensajes recibidos.
133     print "----Distancia R:      %s" % distanciaR
134     print "----Distancia L:      %s" % distanciaL
135     print "----CODE-----:      %s" % code
136     print "----CODIGO---:      %s" % bin(codigo)
137     print "----Battery ---:      %s" % volt
138
139 ##Estado en busqueda de senales IR, sigue trayectorias
140     evitando obtaculos segun las distancias detectadas por
141     los sensores ultrasonicos.
142
143
144
145     if lost==1:
146         rate = rospy.Rate(10)
147
148         print "          --LOST-- "
149
150         if distanciaC < 40 and distanciaR < 40
151             and distanciaL < 40:
152                 x=-3
153                 y=1
154                 pub1.publish(x)
155                     #Publicacion de las
156                     #velocidades. "x" es el avance,
157                     # "y" es el giro.
158                 pub2.publish(y)
159
160             elif distanciaC < 20:
161                 if distanciaL < distanciaR:
162                     x=0
163                     y=5

```

```

151                               pub1.publish(x)
152                               pub2.publish(y)
153             else:
154                 x=0
155                 y=-5
156                 pub1.publish(x)
157                 pub2.publish(y)
158         elif distanciaL < 20:
159             x=0
160             y=5
161             pub1.publish(x)
162             pub2.publish(y)
163
164         elif distanciaL < 40:
165             x=5
166             y=3
167             pub1.publish(x)
168             pub2.publish(y)
169         elif distanciaR < 20:
170             x=0
171             y=-5
172             pub1.publish(x)
173             pub2.publish(y)
174         elif distanciaR < 40:
175             x=5
176             y=-3
177             pub1.publish(x)
178             pub2.publish(y)
179     else:
180         x=25
181         y=0
182         pub1.publish(x)
183         pub2.publish(y)
184     rate.sleep()
185
186     if codigo != 4096 and recargar == 1:
#4096 corresponde al numero binario
del mensaje cuando no se recibe
ninguna senal IR (Un 1 seguido de 12
ceros). Si se recibe alguna senal y
recargar es igual a 1, se pasa al
estado 0. (100000000000 = bin(4096)).
187         lost=0
188         giro=0
189         girando=0
190

```

```

191
192         print "-----Speed x:      %s" %x
193         print "-----Speed y:      %s" %y
194
195 ##Estado Localizado Region Lejana. Tras detectar algunas
196     de las senales IR, se analiza cual es y se realizan
197     los movimientos necesarios para el acercamiento a la
198     zona central cercana.
199
200     if lost==0:
201
202         print "----- Localized -----"
203         rate = rospy.Rate(10)
204
205         if distanciaC < 15 or distanciaR < 15 or
206             distanciaL < 15:          #Si aparece
207                 algun obstaculo en la zona alrededor
208                 de la estacion a menos de 15 cm del
209                 robot , espera hasta que desaparezca.
210                 x=0
211                 y=0
212                 pub1.publish(x)
213                 pub2.publish(y)
214
215         elif codeRC[0] == '1' and codeLC[2]=='0':
216             #Segun las senales detectadas ,
217             realiza un avance con un giro u otro.
218             girando=0
219             giro=1
220             if codeRC[1]==='1' or codeLC
221                 [1]==='1':    #Si se encuentra
222                     en la region cercana a la base
223                     , se pasa al siguiente estado.
224                     Igual si ocurre en las
225                     siguientes sentencias elif .
226                     lost=2
227             else:
228                 x=15
229                 y=5
230                 pub1.publish(x)
231                 pub2.publish(y)
232
233         elif codeRC[0] == '0' and codeLC[2]==='1':
234             girando=0
235             giro=2

```

```

222         if codeRC[1]=='1' or codeLC
223             [1]=='1':
224                 lost=2
225             else:
226                 x=15
227                 y=-5
228                 pub1.publish(x)
229                 pub2.publish(y)
230             elif codeRC[0] == '1' and codeLC[2]=='1':
231                 girando=0
232                 giro=0
233                 if codeRC[1]=='1' or codeLC
234                     [1]=='1':
235                         lost=2
236                     else:
237                         x=10
238                         y=0
239                         pub1.publish(x)
240                         pub2.publish(y)
241
242             elif codeR[0] == '1' and codeR[2]=='1':
243                 #Zona central detectada por el
244                 sensor derecho , realiza un giro .
245                 girando=1
246                 encoder_inicio= encoder1
247                 x=0
248                 y=5
249                 pub1.publish(x)
250                 pub2.publish(y)
251
252             elif codeL[0] == '1' and codeL[2]=='1':
253                 #Zona central detectada por el
254                 sensor izquierdo , realiza el giro
255                 opuesto .
256                 girando=1
257                 encoder_inicio = encoder1
258                 x=0
259                 y=-5
260                 pub1.publish(x)
261                 pub2.publish(y)
262
263             elif codeR[2] == '1':
264                 #Sensor derecho recibe de solo
265                 un emisor (regiones laterales
266                 externas )

```

```

258                         x=10
259                         y=3
260                         pub1.publish(x)
261                         pub2.publish(y)
262
263             elif codeL[0] == '1':
264                 #Sensor izquierdo recibe de
265                 #solo un emisor.
266                 x=10
267                 y=-3
268                 pub1.publish(x)
269                 pub2.publish(y)
270
271             rate.sleep()
272             if codigo==4096 and girando==0:
273                 #Si se deja de recibir alguna
274                 #senal IR, se vuelve al estado inicial.
275                 lost=1
276             if codigo==4096 and girando==1 and (abs(
277                 encoder_inicio-encoder1) > 550):
278                 #Esto evita entrar en un bucle
279                 #en el caso de que se reciba una senal
280                 #por uno de los lados, se ponga a
281                 #girar y no vuelva a recibir ninguna
282                 #senal que haga que salga de ese estado
283
284                 lost=1
285
286             print "-----Speed x:      %s" %x
287             print "-----Speed y:      %s" %y
288
289             tension = volt          #Guarda en la
290             #variable "tension" el valor de la
291             #tension de las baterias para usarlo en
292             #el estado siguiente.
293
294             ##Estado Localizado Region Cercana. Se encuentra en la
295             #zona central y ha detectado la senal del emisor de
296             #region cercana.
297
298             if lost == 2:
299
300                 print "-----"
301                 print "Localized ZONA CERCANA"
302                 print "CENTRAL--"

```

```

286         rate = rospy.Rate(10)
287
288
289     if codeRC[0] == '1' and codeLC[0] == '1':
290         girando=0
291         giro=1
292         x=6
293         y=0
294         pub1.publish(x)
295         pub2.publish(y)
296
297     elif codeRC[0] == '1':
298         girando=0
299         giro=1
300         x=6
301         y=3
302         pub1.publish(x)
303         pub2.publish(y)
304     elif codeLC[0] == '1':
305         girando=0
306         giro=1
307         x=6
308         y=-3
309         pub1.publish(x)
310         pub2.publish(y)
311
312     if volt > (tension + 2):          #Si el
313                                         voltmetro detecta una diferencia de
314                                         tension entre el estado anterior y la
315                                         lectura actual, significa que se ha
316                                         realizado la conexion correctamente.
317                                         Se pasa al siguiente estado.
318                                         lost=3
319                                         tiempo_recarga=0          #Se
320                                         iguala a 0 el tiempo de la
321                                         recarga.
322     else:
323         rate.sleep()
324
325 ##Estado de carga y desconexion.
326
327
328     if lost == 3:
329         rate = rospy.Rate(10)

```

```

325
326     print "                    -- RECARGA!!--"
327     x=0
328     y=0
329     pub1.publish(x)           #Se paran
330             los motores.
331     pub2.publish(y)
332     rate.sleep()
333     time.sleep(10)

334     tiempo_recarga += 1
335
336     if recargar == 0 or tiempo_recarga >= 2:
337             #Se realiza un bucle para
338             controlar el tiempo durante el que se
339             realiza la recarga.

340
341     print "                    --"
342             DESCONEXION--"
343     x=-10                      #
344             Movimiento hacia atras de
345             desconexion.
346
347     y=0
348     pub1.publish(x)
349     pub2.publish(y)
350     rate.sleep()
351     time.sleep(4)

352
353     x=0                      #
354             Giro sobre si mismo para
355             seguir su camino.
356
357     y=10
358     pub1.publish(x)
359     pub2.publish(y)
360     rate.sleep()
361     time.sleep(5)

362
363     lost=1                      #
364             Vuelve al estado inicial.
365     recargar = 0                  #
366             Se establece igual a 0 para
367             que no vuelva a buscar las
368             senales IR hasta que se
369             indique lo contrario.

370

```

```
357
358     rospy.spin()
359
360
361 if __name__ == '__main__':
362     listener()
```