



Proyecto MiRobot

Laboratorio de Sistemas Electrónicos – Curso 2015

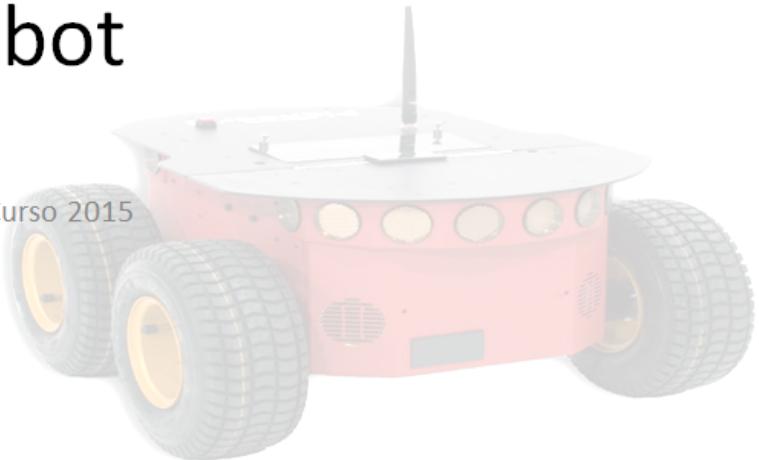
Artés, Diego Onofre

Granda, Juan Manuel

Muñoz, Daniel

Ramos, Sergio

Real, Santiago Isidro



[1 Introducción al proyecto realizado](#)

[2 Desarrollo de los sprints](#)

[3 Cambios Hardware](#)

[Extracción del PC on board](#)

[Introducción de Raspberry Pi Modelo B](#)

[Interfaz RS232](#)

[Modulo WiPi](#)

[Caja protectora impresa](#)

[4 Desarrollo Software](#)

[Esquema general](#)

[Modelo de concurrencia del programa](#)

[Breve descripción de la interfaz de comunicación.](#)

[Máquina de estado TX y RX.](#)

[Máquina de estado TX](#)

[Máquina de estado RX](#)

[5 Funcionalidades y uso](#)

[testMotors](#)

[testBumpers](#)

[subsumption](#)

[6 Propuestas de futuro](#)

1 Introducción al proyecto realizado

El proyecto realizado se enmarca dentro de la asignatura LSEL (Laboratorio de Sistemas Electrónicos) de Ingeniería de Telecomunicaciones. El proyecto ha tenido una duración aproximada de 5 semanas y en ella ha participado un grupo de 5 alumnos:

- Daniel Muñoz
- Diego Onofre Artés
- Juan Manuel Granda
- Santiago Isidro Real
- Sergio Ramos

Además de estos alumnos también hemos contado con la participación de dos profesores que nos han servido como apoyo profesional a cualquiera de las vicisitudes que pudieron surgir en el transcurso del proyecto (Álvaro Araujo y José Manuel Moya), y otro profesor que jugó el rol de cliente (Octavio Nieto).

El objetivo principal del proyecto ha sido desarrollar un sistema electrónico propio para el control y manejo de un robot proporcionado por los profesores de la asignatura. Para ello hemos contado con el propio vehículo (Pioneer 3 de Mobile Robots) y una Raspberry Pi como sistema empotrado de control, además de material auxiliar para completar alguna de sus funcionalidades. Como material de apoyo hemos contado con el manual del fabricante del propio vehículo.

Uno de los requisitos imprescindibles del proyecto era sustituir el ordenador de propósito general con el que contaba el robot, por un sistema empotrado propio que nos permitiera diseñar e implementar funcionalidades adicionales, como ejecución de tareas en tiempo real. Por lo tanto, el grupo de trabajo se vió obligado a participar en todos los aspectos que abarca el desarrollo de un sistema profesional, desde hardware hasta software para lograr los objetivos pedidos. Además, el software implementado proporciona un sistema de navegación autónomo así como una arquitectura que permite añadir nuevos módulos con nuevas funcionalidades de forma fácil.

La metodología de trabajo que ha seguido el grupo consiste en una metodología scrum. En esta metodología definimos los objetivos y los tests que queremos desarrollar de cara al siguiente sprint, siempre cumpliendo las exigencias del cliente, con el que tendremos plena comunicación a lo largo del transcurso de todo el proyecto. Definimos un tiempo dedicado por cada alumno (un mínimo de 16 horas por alumno cada semana) y nunca comprometemos la calidad de la solución presentada. De forma paralela a esta vía de trabajo, también desarrollamos una serie de mensajes que recibieron el nombre de “Daily Meeting” que nos fueron muy útiles para mantener la comunicación interna del equipo.

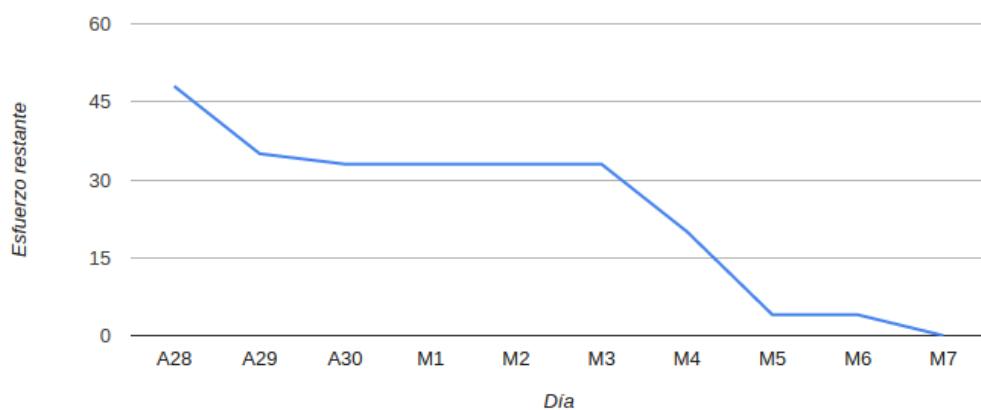
Sobre el robot, se trata del Pioneer 3AT de MobileRobots. La empresa encargada de su fabricación es una empresa estadounidense que trabaja en este campo desde 1995. Los productos que fabrica se destinan a investigación, desarrollo y formación. Sobre el modelo con el que hemos trabajado se trata de un vehículo de cuatro ruedas, fabricado en aluminio con unas dimensiones de 51x50x28 (cm). Tiene un peso de 12 kg y puede soportar una carga de hasta 35 kg. Dispone de cuatro motores y tres baterías que proporcionan una autonomía de hasta 4 horas. Como sensores dispone de bumpers (o sensores de contacto) tanto en la parte delantera como en la parte trasera, y sonars, dispuestos también tanto en la parte frontal como en la parte posterior. Además, el fabricante proporciona otros dispositivos como distintos tipos de cámaras, brazos robóticos, GPS...

En este documento se realiza una breve introducción al trabajo realizado durante este tiempo. Este texto no sólo está destinado a los profesores de la asignatura, para facilitarles la evaluación del proyecto llevado a cabo, sino que también está destinado a todos aquellos alumnos que pudieran trabajar con el código desarrollado con el fin de implementar nuevas funcionalidades y ampliar las oportunidades que ofrece el vehículo y su sistema de control. Una vez realizada esta pequeña introducción, procedemos a comentar en el punto 2 la metodología scrum y el desarrollo de los sprints de forma individual. En los capítulos 3 y 4 comentamos los cambios hardware y el desarrollo software respectivamente. En el capítulo 5 hablamos de las funcionalidades alcanzadas y del uso y el manejo del vehículo. En el apartado 6 comentamos posibilidades futuras, aquellas que no pudimos desarrollar por restricciones temporales del curso, pero fueron propuestas por el grupo de trabajo. Para finalizar, en el séptimo y último capítulo tratamos el control de versiones que se ha seguido y el uso de la herramienta github.

2 Desarrollo de los sprints

A continuación se proceden a detallar los objetivos propuestos y alcanzados durante el desarrollo de los diferentes sprints que tuvieron lugar a lo largo del curso:

- **Sprint 1:** El primero de los sprints contó con dos objetivos. Uno de ellos fue conseguir el control sobre los motores para poder producir movimiento en el vehículo. El otro era poder leer la información recibida por los bumpers, de manera que supiéramos en todo momento si alguno de los bumpers había sido pulsado. Desafortunadamente, ninguno de estos objetivos se pudieron conseguir durante la primera semana de trabajo. El grupo se centró en familiarizarse con el producto con el que tendría que trabajar a lo largo del mes siguiente, consiguiendo buenos resultados. También se modificó la fecha de reunión con el cliente de manera que hubiera una mejor coordinación y comunicación de cara a la presencia en el laboratorio.
- **Sprint 2:** Una vez finalizado el primero de los sprints, el objetivo marcado para el siguiente sprints fue conseguir realizar de forma correcta un test para comprobar el funcionamiento y manejo de los motores. También fue imprescindible conocer e implementar en nuestro sistema una correcta comunicación con el sistema ARCOS. A partir de este momento los resultados de todos los sprints fueron adecuados, cumpliendo siempre con las exigencias del cliente. A continuación podemos ver la gráfica correspondiente al esfuerzo restante dentro del segundo sprint. El resto serán parecidas. Se puede apreciar un trabajo inicial, un parón debido al fin de semana y un trabajo posterior para finalizar sin trabajo restante al final del sprint:



- **Sprint 3:** Este fue sin duda el sprint en el que se consiguieron cumplir más objetivos. Se consiguió leer la información recibida acerca de los bumpers y los sonars. Además se consiguió realizar un giro y comunicación inalámbrica con el robot.

- **Sprint 4:** En este sprint se desarrolló la arquitectura de subsunción para sistema autónomo y se realizó una integración de funcionalidades, de manera que pudimos presentar un código robusto. Además se preparó el producto para su entrega final al cliente.
- **Sprint 5:** Este último sprint no contó con horas de trabajo en el laboratorio como los anteriores. En este caso estuvimos preparando la presentación del proyecto, así como la documentación que se presenta en este documento.

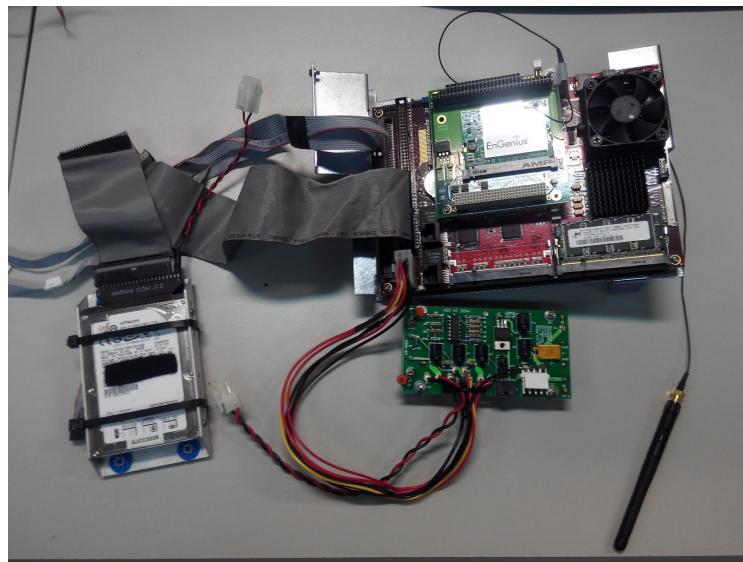
3 Cambios Hardware

Para la realización del proyecto se hace necesario la realización de una serie de cambios en el hardware del dispositivo con el objetivo de mejorar el dispositivo en sí. Esto se consigue con los puntos que veremos a continuación, donde se explicarán los cambios acometidos, el por qué y cómo están situados en el sistema actual.

Extracción del PC on board

El robot Pionner 3AT viene con un ordenador de abordo, una especie de placa base de sobre mesa customizada para ser incluida dentro del robot, con ranuras de expansión PCMCIA, cable para antena wifi, disco duro, diversos buses para comunicaciones... Dada la antigüedad del ordenador, el consumo, la temperatura generada por la fuente de alimentación y el peso que implica, se propone la sustitución de dicho sistema.

Se examina con detenimiento las conexiones del ordenador y se procede a su extracción, poniendo especial atención en que las dos principales líneas de conexión que son la alimentación a la fuente y un cable serie terminado en un conector DB9 a una placa electrónica del robot, conocida como ARCOS.



Introducción de Raspberry Pi Modelo B

El antiguo ordenador es sustituido por un sistema empotrado Raspberry Pi, el cual nos va a permitir mejorar el sistema en cuanto a consumo, tamaño ocupado, peso, y nos añade la posibilidad de realizar una gestión del sistema mediante el uso de tareas en tiempo real.

La alimentación de la Raspberry se consigue desmontando un cable usb a micro-usb, y conectando la parte del usb a un par de pines que ofrece la placa ARCOS para conectar dispositivos auxiliares. Tomamos GND y +5V. El robot ofrece una interfaz Ethernet en la

tapadera superior, por lo que cogemos y conectamos con un alargador que tenía el anterior sistema el puerto Ethernet de la Raspberry al panel superior del robot, esto nos permitirá en un futuro conectados a la Raspberry por Ethernet.

Interfaz RS232

La forma de manejar el robot es mediante el envío y recepción de información a la plataforma controladora ARCOS del robot. Se ha comprado una placa conversora de niveles de tensión que nos va a permitir conectar la interfaz serie del sistema ARCOS a los GPIOs de la Raspberry. El módulo elegido para tal fin es el siguiente:

<http://ie.farnell.com/digilent/210-068/mod-converter-pmodrs232-adm3232e/dp/2061858>

Un ejemplo de uso de este tipo de conversores se puede encontrar en el siguiente enlace:

<http://www.davidhunt.ie/add-a-9-pin-serial-port-to-your-raspberry-pi-in-10-minutes/>

<http://www.savagehomeautomation.com/projects/raspberry-pi-installing-a-rs232-serial-port.html>

Modulo WiPi

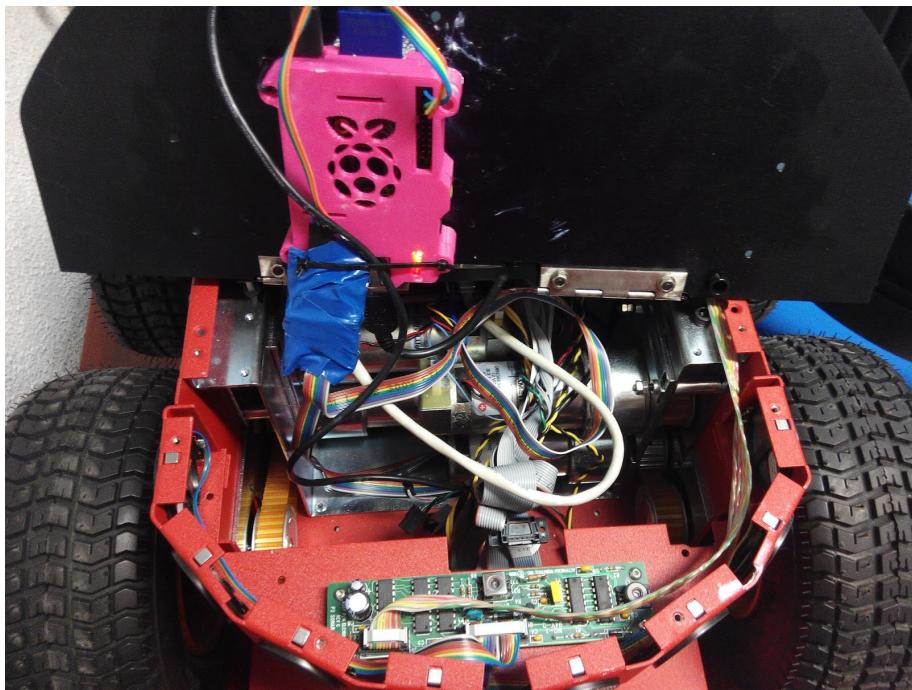
Con el objetivo de poder configurar el funcionamiento del robot y manejarlo de manera inalámbrica se procede a la instalación de un módulo WiPi con el que podremos conectarnos a una red inalámbrica generada con un dispositivo móvil como un tablet o un móvil.

<http://es.farnell.com/element14/wipi/dongle-wifi-usb-for-raspberry/dp/2133900?ost=wipi&categoryid=700000005571>

Para conectarnos al robot, deberemos generar con nuestro dispositivo móvil una red wifi con el nombre rospimola y abierta. Esta configuración se puede cambiar en la raspberry pi desde el archivo interfaces situado en /etc/network/interfaces.

Caja protectora impresa

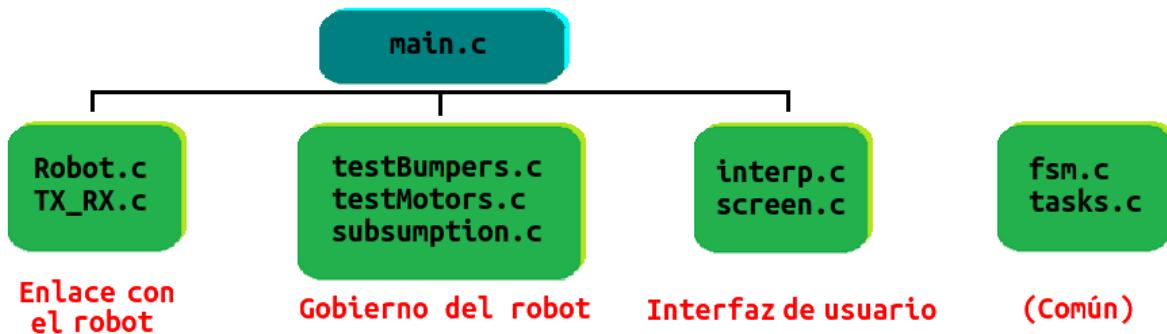
Para colocar la Raspberry en un lugar seguro y poder colocarla de la mejor forma posible en la cavidad del robot, se imprime con una impresora 3D Witbox una caja protectora como la que podemos ver en la imagen, de este modo se ve la conexión de la alimentación, el módulo conversor de RS232, envuelto en cinta aislante para tener el módulo aislado y cómo el bus de colores va hacia el conector de la plataforma ARCOS. A destacar que el módulo ARCOS actúa como DCE y el módulo RS232 también por lo que se tuvo que cruzar el cable para convertir el módulo RS232 en DTE. Por otro lado, para tener activa la conexión se extraen 12V de la plataforma arcos para el pin de habilitación de conexión DTR.



4 Desarrollo Software

Esquema general

El programa está dividido en una serie de módulos con funcionalidades específicas, según el esquema:



- **Enlace con el robot:** se encarga de transmitir los comandos al Robot, y de recibir información del mismo (sensores - sónares, bumpers, baterías,... -, configuración - parámetros de velocidad y aceleración, velocidad de transmisión,... -, etc.). Robot.c construye los comandos a enviar y descodifica los paquetes recibidos. TX_RX.c se encarga de transmitir y recibir datos por el puerto serie.
- **Gobierno del robot:** esta es la parte “inteligente” del código. Aquí se implementaron los test del robot (para probar comandos simples de movimiento y recibir información de los sensores), y una arquitectura de subsunción.
- **Interfaz de usuario:** intérprete de comandos (interp.c) y representación simple de gráficos (screen.c).
- **Ejecutable:** main.c.

En caso de querer mejorar el código:

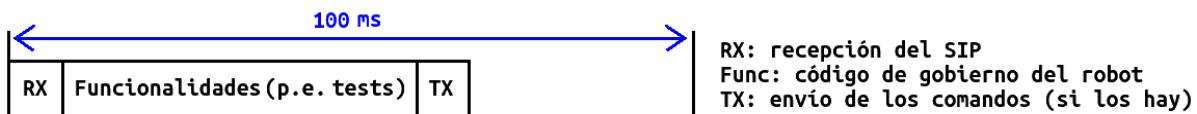
- Se pueden añadir nuevos comandos del manual, y recoger información no considerada hasta ahora de los SIP, en Robot.c.
- TX_RX puede dar problemas en caso de desconexión brusca con el robot, tanto a nivel físico (UART) como a nivel de enlace (p.e. en caso de que un error de transmisión provoque que el robot corte la conexión, en cuyo caso se han de enviar de nuevo las tramas sync); habría que completar la máquina de estados.

- Para añadir nuevas funcionalidades, basta con incluir un nuevo archivo .c, y llamarlo desde main.c. Recomendamos asociar a cada nueva funcionalidad un nuevo comando del intérprete.

Modelo de concurrencia del programa

El programa se divide en dos hebras (mirar main.c):

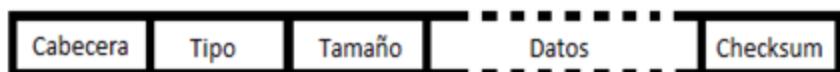
- La primera ejecuta el intérprete de comandos
- La segunda, de mayor prioridad, ejecuta el código cumpliendo restricciones de tiempo real. Esta hebra ejecuta las tareas siguiendo el modelo de concurrencia “ejecutivo cíclico” (ver “Máquinas de estado TX y RX” más adelante):



Breve descripción de la interfaz de comunicación.

Para establecer la conexión con el Robot se ha implementado una interfaz de comunicaciones a través del puerto UART con protocolo RS232 con una velocidad de 9600 baudios.

Para un control básico de Robot nos era necesario obtener información acerca del estado de los bumpers, sonars, motores,... Toda esta información que nos era necesaria viene especificada dentro del paquete de estado SIP estándar que el robot envía periódicamente cada 100 ms (configuración por defecto, modificable). En estos paquetes, aparte de la información dicha, podemos obtener otros datos que pueden ser de utilidad para un futuro, como es el caso de las posiciones relativas del robot o su orientación, así como el estado de las baterías. La estructura, a rasgos generales, del paquete SIP es la siguiente:



La Cabecera, tanto para escritura como para lectura, está compuesta siempre por los dos mismos bytes (250,251).

El Tipo sirve para distinguir entre distintos tipos de paquetes que se reciben, ya que no sólo se reciben paquetes de tipo SIP ya que, por ejemplo, podemos solicitar la configuración del robot devolviendo este otro paquete con distinto tipo que el SIP.

El campo Tamaño indica la longitud del paquete, en bytes, del paquete recibido sin contabilizar el campo Tipo y Cabecera.

En el campo Datos obtenemos la información requerida hasta el momento (sonars, bumpers, estado de los motores,...).

Finalmente, el campo Checksum sirve para verificar si el paquete recibido es correcto de acuerdo con el contenido de este.

Para actuar en función de esta información (por ejemplo en los tests) es necesario generar comandos dentro de la gran gama de ellos que se dan, como por ejemplo comandos de rotación o de aceleración (consultar manual). Su estructura es parecida a los paquetes recibidos salvo que no incluyen el campo Tipo y que en el campo Datos ponemos los argumentos asociados al comando.



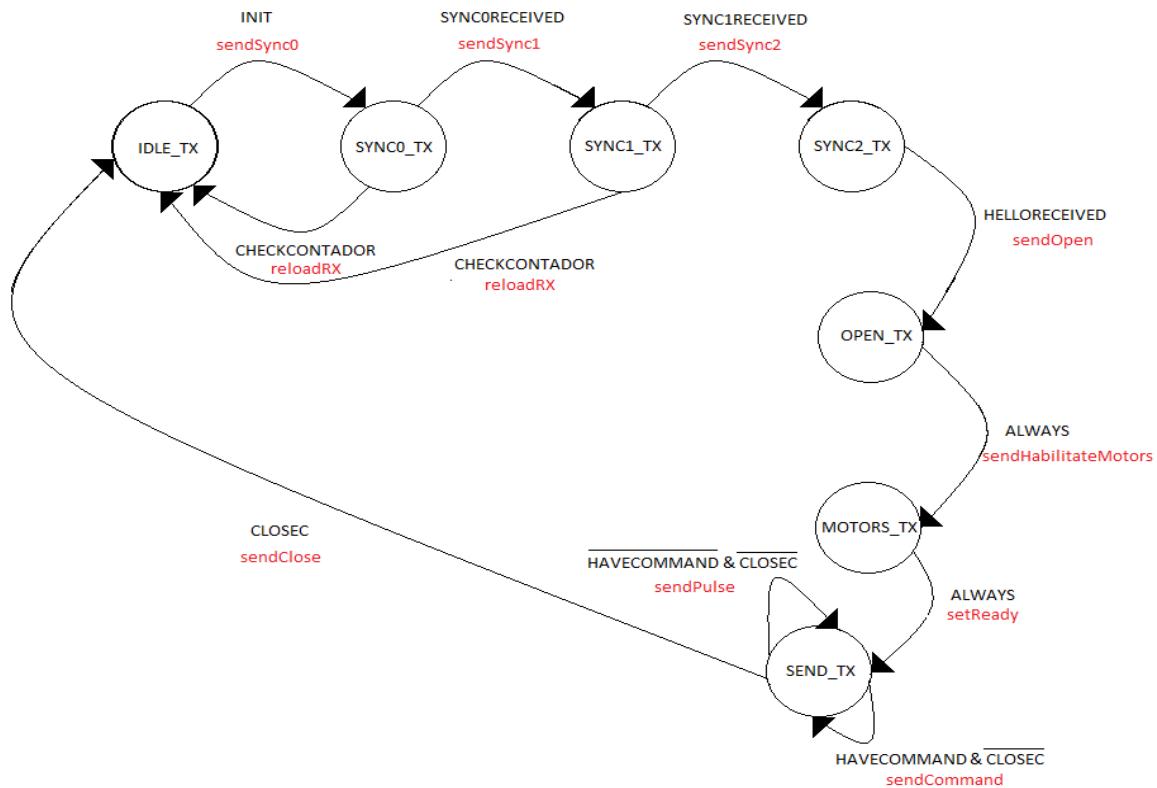
Máquina de estado TX y RX.

A continuación se expondrá, mediante un esquema, las máquinas de estado correspondientes a los módulos de transmisión y recepción de nuestra interfaz de comunicaciones.

Ambas máquinas comparten una serie de variables que se enumeran a continuación:

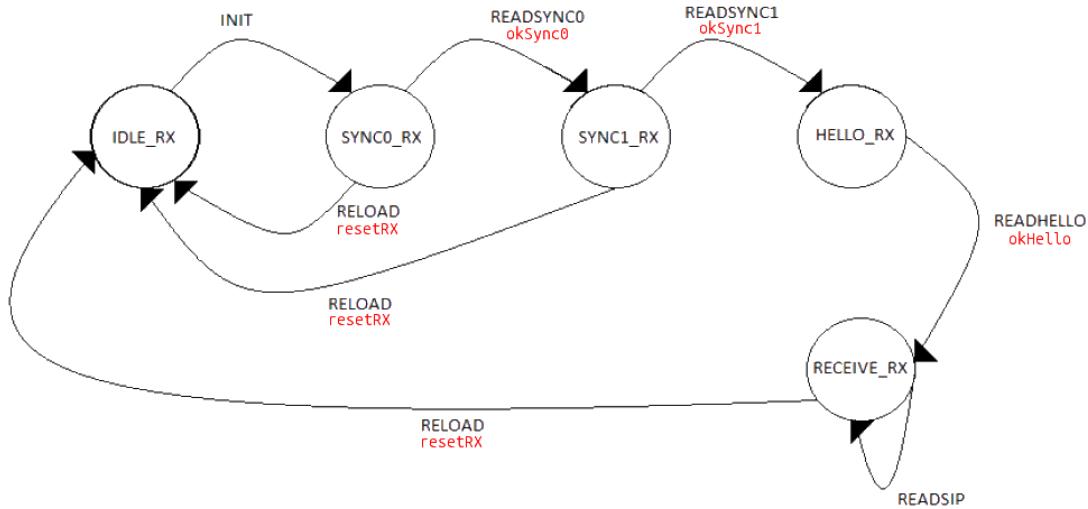
- **initR**: Variable que sirve para inicializar ambas máquinas de estado a la vez (INIT).
- **sync0R**: Variable que determina si se ha recibido la respuesta al envío del comando SYNC0 (SYNC0RECEIVED - READSYNC0).
- **sync1R**: Variable que determina si se ha recibido la respuesta al envío del comando SYNC1 (SYNC1RECEIVED - READSYNC1).
- **helloR** : Variable que determina si se ha recibido la respuesta al envío del comando SYNC2 (HELLORECEIVED - READHELLO).
- **reload_RX**: Variable que sirve para cerrar conexión de ambas máquinas de estado a la vez (CLOSEC - RELOAD).

Máquina de estado TX



(Nota: funciones de salida en rojo)

Máquina de estado RX



5 Funcionalidades y uso

Tras lanzar el programa con “sudo ./main” dispondremos de la interfaz que aparece en la imagen:

```
*****  
*          ROSPI APPLICATION MANAGER          *  
*****  
  
)))))  FRONT SONAR  (((((  
|||||  
|||  
||  
||  
|||  
|||  
)))))  REAR SONAR  (((((  
|||||  
|||||  
|||||  
|||||  
|||||  
|||||  
  
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----  
^ ^ ^     FRONT BUMPERS     ^ ^ ^ | | ^ ^ ^     REAR BUMPERS     ^ ^ ^  
  
>>> sonars 1  
>>> subsumption 1  
>>> subsumption  
>>> q  
>>> vel -10  
>>> vel -100  
>>> vel 0  
>>> subsumption  
>>> q  
>>> bumpers 1  
>>> █
```

En primer lugar se establece la conexión con el robot, y rápidamente comenzamos a recibir lecturas de los sensores (por ahora se manejan únicamente “sonars” y “bumpers”).

Los comandos que se ofrecen son los siguientes:

- **help** (or “?”): muestra la lista de comandos con su función.
- **sleep**: duerme el intérprete durante x segundos.
- **quit**: cierra la conexión con el robot, y acaba la ejecución.
- **task**: muestra las tareas ejecutándose.
- **setvel**: fija el límite de velocidad del robot en mm/s.
- **setacc**: fija el límite de aceleración del robot en mm/s².
- **rotate**: fija una velocidad de rotación constante en mm/s.
- **vel**: fija una velocidad de desplazamiento lineal constante en mm/s.
- **radiocontrol**: entra en el modo “radiocontrol”, pudiendo manejar el robot con las teclas “w”, “a”, “s”, “d”, “e” y “q” (aparecerá una nota para su uso en pantalla)
- **testMotors**: ejecuta el test de los motores
- **testBumpers**: ejecuta el test de los bumpers
- **subsumption**: ejecuta el algoritmo de subsunción
- **q**: detiene la ejecución del test
- **sonars**: muestra (1) u oculta (0) las medidas de los sonars
- **bumpers**: muestra (1) u oculta (0) las medidas de los bumpers

Nota: el software está preparado para incluir nuevos comandos de forma sencilla.

A continuación desarrollaremos brevemente el comportamiento automático del robot con los comandos “testMotors”, “testBumpers” y “subsumption”:

testMotors

Primer test del movimiento del robot. Mantiene 3 velocidades distintas (lenta, media, rápida) en el sentido que se le pasa como parámetro (1, -1) durante intervalos regulares de 3 segundos. Al acabar, repite el proceso en sentido contrario, volviendo al punto de partida.

testBumpers

Test de los bumpers del robot. En caso de choque se inhabilita automáticamente - vía hardware - el movimiento del robot en esa dirección. Éste test recibe como parámetros el sentido inicial (1, -1), y el número de choques N; el robot avanza en línea recta hasta chocar, momento en el que cambia de sentido. Una vez completados los N choques el robot se detiene y finaliza el test.

subsumption

El comando subsumption pone en marcha el modo de navegación autónomo. Se ha implementado una arquitectura de subsunción, de manera que el robot pueda navegar sin la supervisión o el control de un operario. Este algoritmo tiene como objetivo el desplazamiento del robot sin chocar contra ningún obstáculo, por lo que el robot puede moverse hacia delante, detectar obstáculos a distancia y frenar para prevenir un choque y, en caso de choque, retroceder una breve distancia.

La arquitectura de subsunción permite implementar diferentes comportamientos de manera jerárquica, de modo que se evalúan los datos provenientes de los sensores (del entorno) y, conforme a ello, se ejecuta uno de los comportamientos. Cada uno de estos comportamientos constituye un módulo que es totalmente independiente de los demás. Gracias a esto pueden añadirse nuevos módulos que permitan que el vehículo pueda realizar diversas actividades. La arquitectura de subsunción es fácilmente implementable y además es escalable, por lo que consideramos que ha sido una elección correcta el hacer uso de ella para este proyecto en lugar de otras arquitecturas más complejas como robótica evolutiva.

Los módulos que se han implementado son los siguientes:

- Navegar: Es el comportamiento de menor prioridad. Si no se detecta ninguna amenaza para la supervivencia del vehículo, éste continuará con velocidad constante hacia delante.
- Sonars: Este comportamiento se encuentra un nivel por encima del anterior en el orden de prioridades. Es el comportamiento encargado de detectar algún objeto a distancia. Si esto ocurriera, el movimiento del robot se detendría.
- Bumpers: Este último comportamiento es el de mayor prioridad. Es el encargado de detectar, a través de los bumpers o sensores de contacto, si el vehículo ha impactado con algún objeto. Si esto ocurriera, el vehículo retrocederá una breve distancia para guardar una separación entre su parte frontal y el objeto impactado.

En caso de continuar este trabajo, sería interesante implementar nuevos comportamientos o mejorar los anteriores. En este proyecto no pudo llevarse a cabo por falta de tiempo. Sin embargo, sería interesante y básico en cualquier sistema autónomo de estas características que el vehículo, no sólo detectara un objeto a distancia, si no que además lo esquivara sin detener su marcha.

6 Propuestas de futuro

El proyecto acabó antes de poder aprovechar el potencial de las herramientas que se han desarrollado. Con el software implementado y los comandos del manual muchas de las opciones barajadas en el “product backlog” tienen una base sólida, pudiendo completarse con un relativo menor esfuerzo.

Las ideas propuestas, por orden de prioridad (extraídas del “product backlog”) serían:

1	Sensor de temperatura, humedad, brújula
2	Seguir una trayectoria alrededor de un contorno
3	Información (log) de recorrido (distancia-tiempo; parada-tiempo)
4	Realización de recorrido de 25m (Salida-Meta) sorteando hasta 5 obstáculos
5	Transportar un vaso de agua sin derramar nada con obstáculos
6	Describir una trayectoria predefinida
7	Aparcar el vehículo de forma automática en batería
8	Diseño y Desarrollo de Interfaz Gráfica de Usuario
9	Enviar imágenes del recorrido realizado
10	Buscar un objetivo definido con una imagen

Recomendamos para propuestas como (2), (3), (4), (5) o (6) emplear la información de la orientación y la posición en coordenadas respecto a un punto origen que ofrece el robot; llega la información en los paquetes SIP estándar cada 100 ms, y se puede acceder a la misma con las funciones `getOrientation()`, `getXPos()` y `getYPos()` respectivamente (`Robot.c`).