Plataformas para Sistemas Empotrados

PROYECTO FINAL

SISTEMA DE CONTROL DE UN COCHE RC

GRUPO 2461 PAREJA 08
Fernando Jesús López Colino

Elaborado por
Carlos González Sacristán < carlos.gonzalez@estudiante.uam.es >
Sarah Dobber < sarah.dobber@estudiante.uam.es >

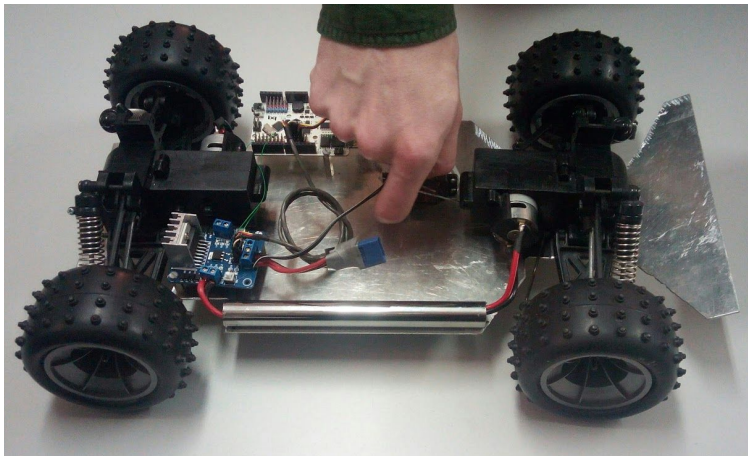
Índice

1. Introducción

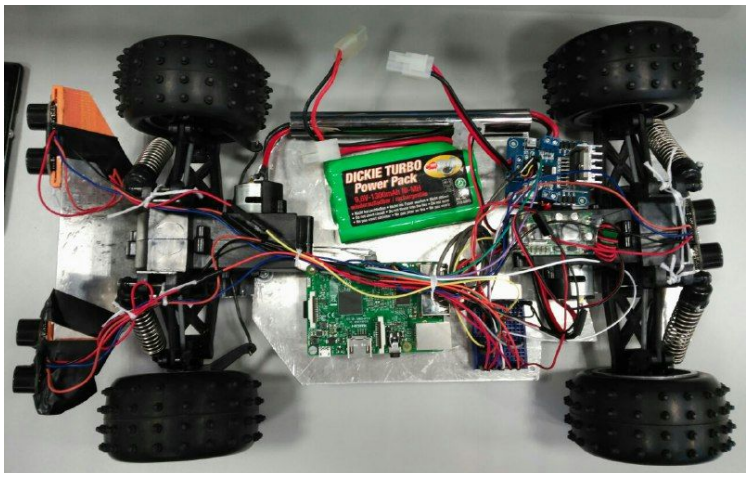
Este proyecto consiste en la adaptación de un coche RC ya existente para ser controlado por una Raspberry Pi 3. La interacción entre la Raspberry Pi y el usuario se hará mediante un ordenador conectado a ella por WiFi.

Hemos partido de un proyecto proporcionado por el Club de Robótica-Mecatrónica de la UAM. El proyecto, en el comienzo, constaba de la base de un coche RC, dos motores y su driver, y un servo para controlar la dirección. El objetivo, tal y como se describió en el ante-proyecto era la implementación de un control más fino y avanzado del coche. El control se realizará a través de un PC, pudiendo controlar desde él la dirección y velocidad del coche. Además, se añadieron sensores de proximidad por ultrasonidos para evitar choques.

Antes:



Después:



2. Configuración y ejecución

Para la ejecución del cliente y servidor se deberán instalar las librerías de las que dependen. Primero hemos configurado la Raspberry Pi 3 pero dicha configuración no se incluirá en esta memoria. Primero se deberá conectar la Raspberry Pi a la batería a través del regulador. Cuando la Raspberry Pi arranca, ejecuta de inmediato el programa servidor, que se encuentra dentro de la misma. Cuando las ruedas del coche se ponen rectas hacia delante, podemos ejecutar el cliente.

Como la Raspberry Pi 3 tiene WiFi integrado, la maquina cliente deberá conectarse al punto de acceso WiFi de la Raspberry, llamado Rpi3. A continuación hay que ejecutar el cliente con el siguiente comando (en Linux):

```
$ python3 cliente.py
```

Para la correcta ejecución del cliente se deben instalar las siguientes librerías:

- Python3 (normalmente incluido)
- Pyro4
- Pygame para python3

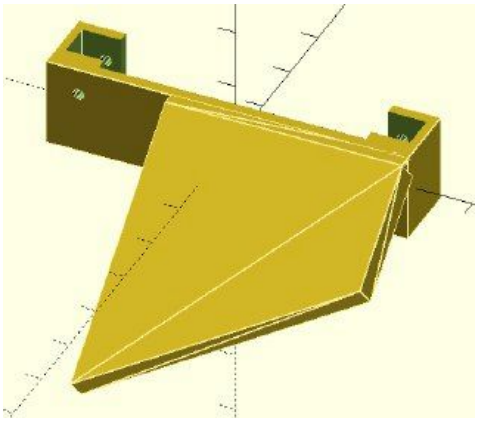
3. Montaje mecánico

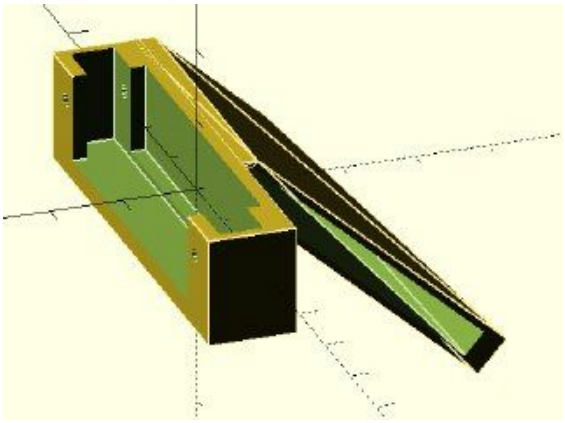
La mayor parte del montaje mecánico estaba construída desde el principio. El coche ya tenía un chasis con dos motores y dos diferenciales. Uno para las ruedas delanteras y otro para las ruedas traseras. Las cuatro ruedas están conectadas a cuatro suspensiones, pudiendo, en principio, circular por terrenos agrestes.

Hemos hecho dos agujeros en el chasis, donde hemos atornillado la Raspberry. Para ahorrar espacio en el chasis, hemos colocado la Raspberry encima del servo.

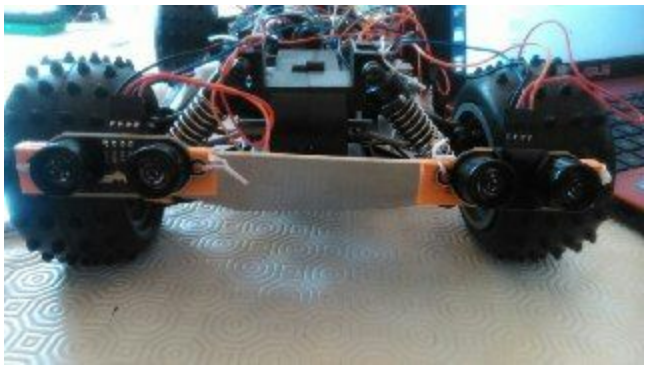
Para evitar que con la vibración se descuelguen las baterías, se han asegurado con cinta adhesiva. El regulador de voltaje también se ha asegurado de la misma manera.

Para la integración de los sensores en la parte frontal del vehículo, hemos diseñado e impreso en 3D unos soportes a medida del coche.





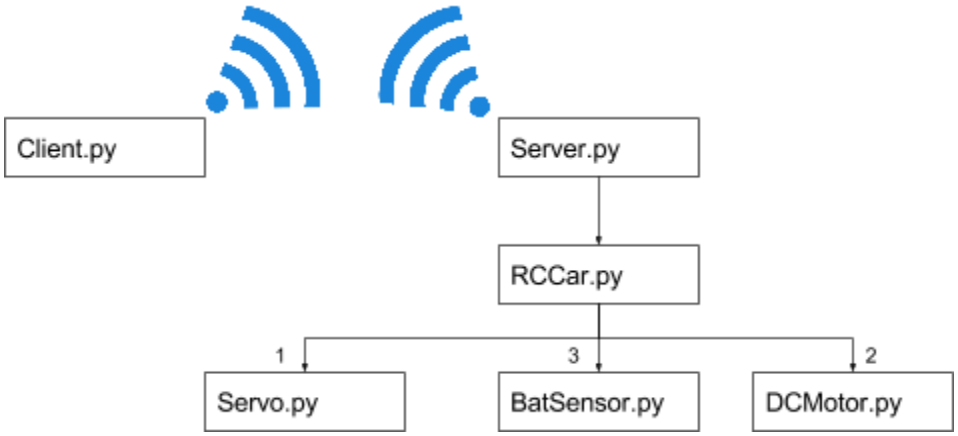
Como se puede observar en la imagen inferior, las piezas están sujetas a la parte delantera del chasis mediante la ranura de las piezas. Los sensores están sujetos a la parte frontal de las mismas.



4. Montaje eléctrico y funcionamiento

Para el desarrollo del coche radiocontrol se han realizado las acciones descritas a continuación, que incluyen una parte hardware, donde se han montado los circuitos necesarios y una parte software que comprueba el funcionamiento de los mismos y hace funcionar el coche.

El proyecto consiste de seis módulos software: client.py, server.py, rccar.py, servo.py, dcmotor.py y batsensor.py y tienen la siguiente estructura simplificada.



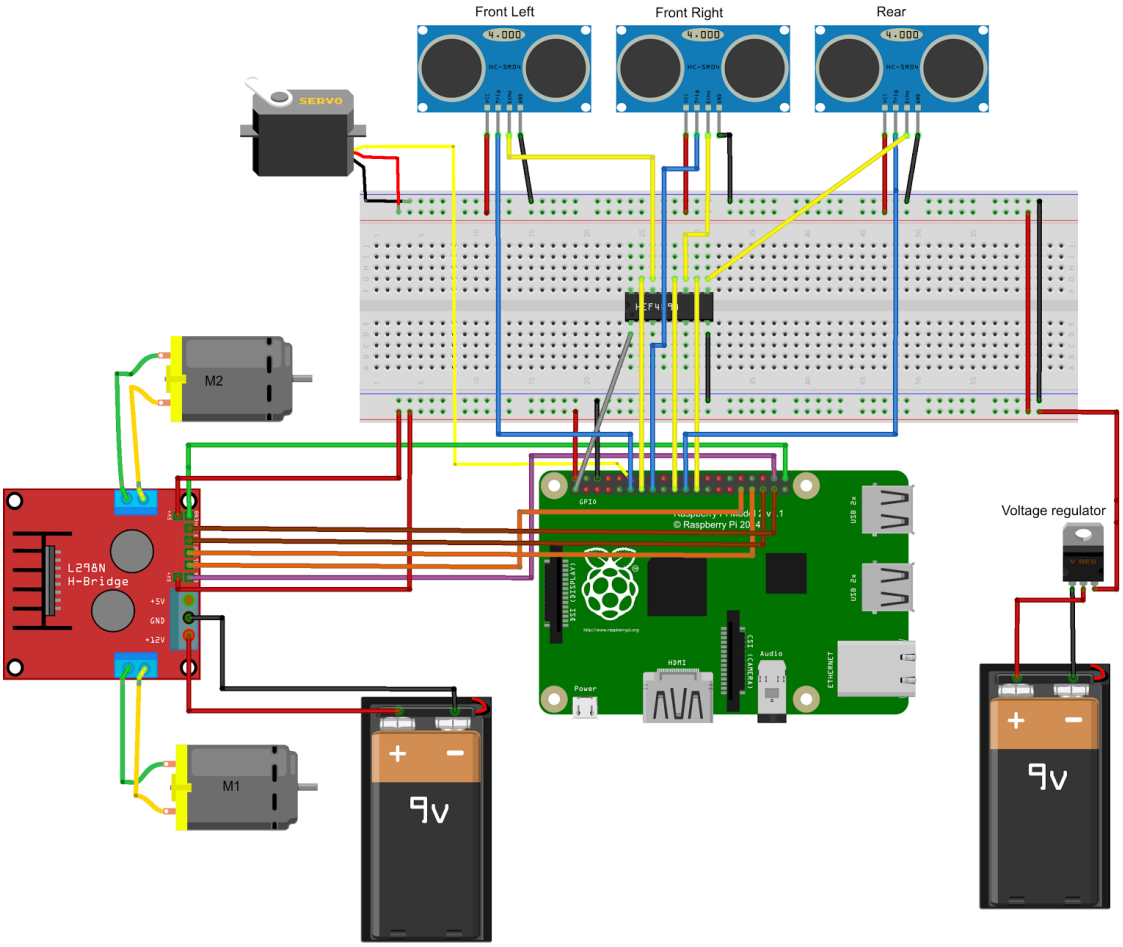
El montaje eléctrico del coche se muestra a continuación. Como se puede observar, todas las conexiones van a la Raspberry Pi. Para la alimentación de la Raspberry se ha utilizado un regulador (UBEC. Universal Battery Elimination Circuit) debido a que sólo poseíamos baterías de 9,6 voltios y la alimentación que necesita la Raspberry es de 5 voltios, y el regulador interno del puente H, que puede proporcionar 5V estables, no daba la suficiente corriente para alimentar el resto del sistema. Por tanto, la batería se conecta con el regulador de voltaje, que a su vez se conecta con la Raspberry en el pin 2.

Para la alimentación de los motores se ha empleado otra batería a parte, ya que con una sola batería no se podía alimentar ambos componentes. La batería que se ha empleado para alimentar los motores también es de 9,6V y va conectada al driver del motor L298N, que es el que alimenta los motores. En este caso no es necesario el uso de un regulador, ya que los motores no tienen un límite de voltaje tan estricto.

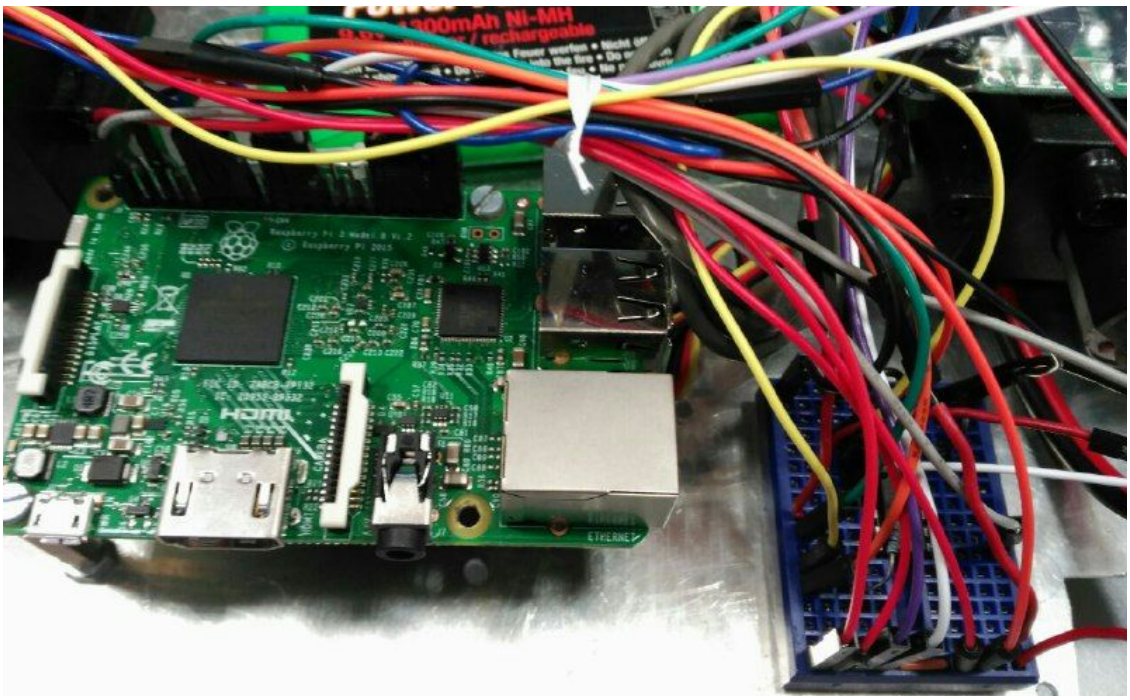
Para el control del movimiento del coche, el driver va conectado a la Raspberry Pi, desde la cual se indicará al driver la dirección a tomar (si los motores deben girar en un sentido u otro). Para ello se emplean los pines 38, 31, 33 para el primer motor y 40, 35, 37 para el segundo motor. Las funciones de los pines son enable, in1 e in2 y se explican en el apartado 3.d.

El servo tiene tres pines, un Vcc que va conectado a la salida de 5V de la Raspberry Pi (pin 2), un GND que va conectado al pin GND de la Raspberry (pin 6) y un pin con el que se le indica al servo el ángulo a tomar. Este pin va conectado al pin 12 de la placa, configurado como PWM, por lo que se le indicará mediante este pin la posición a tomar.

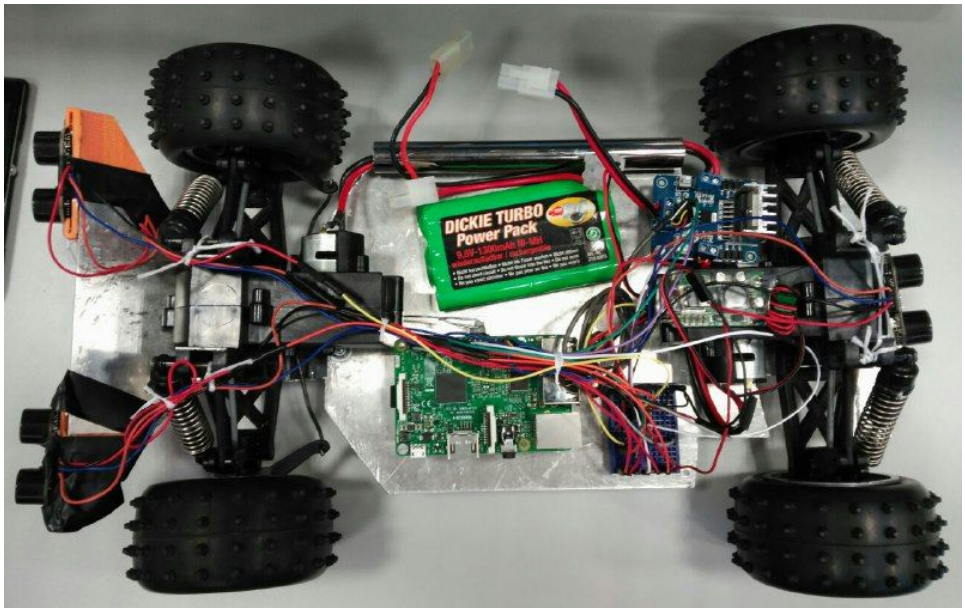
Para detectar y evitar obstáculos, hemos empleado tres sensores de proximidad por ultrasonidos. Hemos colocado dos en la parte frontal del coche y uno en la parte trasera. Los tres sensores tienen cuatro pines, dos de ellos de alimentación, conectados a la salidad el regulador, y dos de datos, conectados a la Raspberry Pi. La patilla de disparo (TRIG) va conectada directamente a la Raspberry Pi (es un GPIO de salida de la placa), que es el encargado de emitir el pulso y la patilla ECO que lo recibe y mide el tiempo que la señal tarda en volver. Así, es posible medir la distancia entre el sensor y el obstáculo. El sensor frontal izquierdo tiene los pines 11 y 13 como trigger y eco respectivamente. El sensor frontal derecho está conectado a los pines 15 y 19 y el sensor trasero está conectado a los pines 21 y 23. Es importante notar que las entradas a la Raspberry Pi están protegidas mediante un buffer, por lo que todos los pines de eco se conectarán primero al buffer y la salida de este se conectarán las entradas de la Raspberry Pi.



Para evitar el wrapinado de los cables, se han utilizado cables macho-macho, macho-hembra y hembra-hembra. Así se aseguran conexiones más seguras y fuertes. Debido a la abundancia de cables, que cruzan el coche entero, nos hemos visto obligados a hacer empalmes de cables, que le restan seguridad a las conexiones, pero están aseguradas igualmente con cinta adhesiva, y para evitar que se metan entre las ruedas al girar, los hemos atado al chasis.



El montaje realizado se muestra a continuación. Se puede apreciar claramente, la colocación de los sensores, las baterías, el driver de los motores, el regulador de voltaje y la Raspberry Pi 3. El servo se encuentra debajo de la Raspberry.



A continuación se comentan las distintas fases del proyecto, por orden de elaboración.

a. Configuración de la Raspberry Pi y conexión mediante WiFi

El diseño de la aplicación sigue el modelo cliente-servidor. Se ha configurado la Raspberry Pi 3 como servidor. Proporciona su propio punto de acceso WiFi, al que el ordenador que haga las veces de cliente se conectará.

b. Cliente

Dada la naturaleza del proyecto, el tiempo de respuesta es importante, por lo que decidimos utilizar una librería de videojuegos (pygame) para capturar las pulsaciones de teclado en "tiempo real", sin esperar a que el usuario pulse Intro para ejecutar el comando. Pygame necesita una ventana de interfaz gráfica para reconocer las teclas cuando ésta tiene el foco del sistema, por lo que diseñamos una interfaz gráfica sencilla que representa el estado del coche en cada momento, usando los valores de retorno de las ejecuciones de los métodos del proxy para cambiar el estado de las imágenes.

c. Servo

Primero, se ha conectado el servo a la Raspberry Pi y se ha codificado una clase de Python, que se encuentra en el archivo servo.py. Si se ejecuta dicho archivo individualmente, se lanza la prueba de la clase, que consiste la introducción por el teclado de números entre -45 y 45, que representan los ángulos del servo, relativos a la posición actual. Con esta prueba

hemos visto que el rango de funcionamiento del servo, antes de empezar a forzarse, es desde 25º a 210º. Como el servo acciona la dirección del coche, vamos a seleccionar un ángulo que nos resulte cómodo. Una vez centrado el servo “manualmente”, vamos a tomar como el rango 72.5º - 162.5º, centrado en 117.5º.

La clase Servo tiene como atributos:

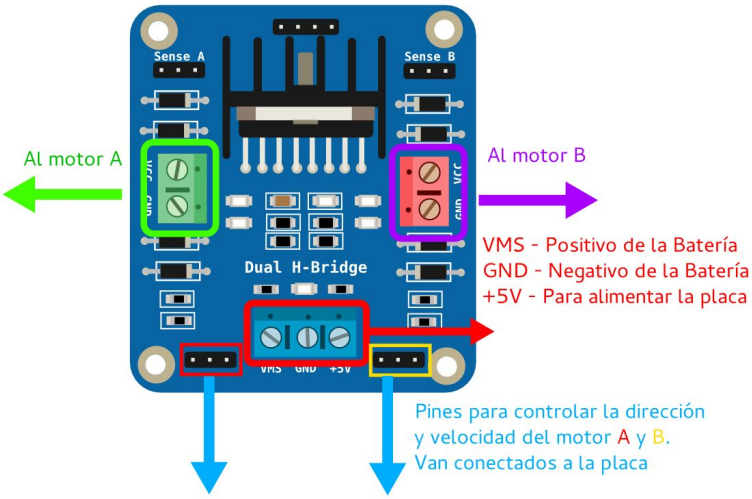
- Un GPIO de control (salida), que actúa como PWM por software, y cuyo duty cycle se calcula en función del ángulo deseado.
- Un rango de actuación (el mencionado anteriormente) con los ángulos válidos.
- Un rango de entrada, que va del -45 al 45 y que se mapeará al rango de actuación. Si se sale de ese rango se tomará uno de los valores mínimo o máximo.
- Un pin configurado como PWM (mismo pin de salida), al que se le tiene especificar la frecuencia.
- Un ángulo.
- Trim. La corrección del error sistemático que pueda tener el servo.

El método `update(angulo)` es el encargado de actualizar la posición del servo al ángulo deseado, relativo al ángulo actual en que se encuentre. Dicho ángulo deberá estar entre [-45,45], el cual se mapeará a los ángulos efectivos del servo, mediante el método `_map`. El cambio del ángulo de las ruedas se realiza con la acción `self.pwm.ChangeDutyCycle(self._duty())`, donde `self._duty()` se encarga de hacer el mapeo del ángulo con la llamada a `_map` y de calcular el duty cycle adecuado para el ángulo deseado.

Ejecución	`$ sudo python servo.py`
Funcionamiento	Introducir números del -45 al 45
GPIO utilizado	18 (pin 12 en la placa)

d. Motores

A continuación implementamos la funcionalidad de los motores. Hemos utilizado el driver L298N para controlar ambos motores.



La tabla de verdad del driver de motores que hemos encontrado en la documentación es la siguiente, aunque después de probar, hemos visto que las opciones de frenado son falsas: no hay un freno activo, sólo la no-aceleración y liberación del motor, por lo que el coche mantiene su inercia.

Tabla de control del motor A:

ENA	IN1	IN2	Description
0	N/A	N/A	Motor A is off
1	0	0	Motor A is stopped (brakes)
1	0	1	Motor A is on and turning backwards
1	1	0	Motor A is on and turning forwards
1	1	1	Motor A is stopped (brakes)

Tabla de control del motor B:

ENB	IN3	IN4	Description
0	N/A	N/A	Motor B is off
1	0	0	Motor B is stopped (brakes)
1	0	1	Motor B is on and turning backwards
1	1	0	Motor B is on and turning forwards
1	1	1	Motor B is stopped (brakes)

En el archivo dcmotor.py se implementa la clase DCMotor, que controla los motores de nuestro coche. El coche tiene dos motores, controlados por un doble puente H. Los GPIO utilizados en la clase son los que conectan la Raspberry Pi con el puente H, no hay nada conectado directamente a los elementos de potencia. Si se ejecuta individualmente este archivo, podemos probar el funcionamiento de los motores, introduciendo números positivos para ir hacia delante, negativos para atrás, y 0 para parar.

Ejecución	`$ sudo python dcmotor.py`
Funcionamiento	Introducir números del positivos, negativos, 0 para salir.
GPIOs utilizados	20, 6, 13 (38, 31, 33 en la placa) para el enable, in1 e in2

e. Coche radiocontrol

La clase RCCar, en RCCar.py, integra todas las demás clases del proyecto, dando funcionalidad a más alto nivel, abstrayendo la funcionalidad del coche de la acción de sus componentes. Esta es la clase que se va a publicar como objeto remoto, y de la que se hará un proxy en el cliente. Se encarga de la inicialización de la librería RPi.GPIO y de la configuración de los demás elementos, asignando a cada componente los gpio que les corresponden.

f. Servidor

La comunicación entre el cliente y el servidor se hace mediante una librería de objetos remotos, Pyro4, que facilita la creación y comunicación remota de objetos, publicando un objeto en un servidor disponible en la red local, al que accedemos mediante un Proxy en el cliente, y utilizamos dicho objeto como si fuera un objeto local, por lo que la invocación de métodos en el cliente se transmite por el proxy y se ejecuta en el servidor, transmitiéndose la salida de la ejecución al cliente.

El servidor se ejecuta con el siguiente comando:

```
$ sudo python server.py
```

La clase server.py contiene el main que pondrá en marcha el coche. Este archivo se ejecuta automáticamente en el arranque de la Raspberry Pi.

g. Sensores de proximidad

Los sensores de proximidad están integrados en la estructura del coche. Dispone de dos al frente y otro en la parte trasera. La parte software de los sensores es la más compleja.

Debido al sistema de comunicaciones por proxy, el coche es un sujeto pasivo, no hace nada a menos que el cliente lo ordene, por lo que los sensores no pueden sensar autónomamente, dado que necesitan la acción del trigger para disparar el pulso ultrasónico y recibir el eco en el receptor. Para evitar esto, se ha implementado un sistema de timing propio en cada sensor, que consiste en lanzar un subproceso por cada sensor que se encarga de dispararlo a intervalos regulares.

La librería RPi.GPIO usada nos permite ejecutar una función cuando se detecta un flanco (tanto de subida como de bajada) en un GPIO particular. Para nuestros propósitos, la hemos configurado para que se ejecute la función en cualquiera de los dos casos. Cuando se ejecuta dicha función, lee el dato del GPIO, y si es nivel ALTO, significa que es el comienzo del eco, guarda el timestamp y sale. Cuando lo que se detecta es un nivel bajo, significa fin de pulso, se guarda el timestamp igual y se procede al cálculo de la distancia. El sistema de toma de medidas se ha implementado usando una cola FIFO de longitud fija (5 elementos), donde el más antiguo desaparece cuando entra un dato más nuevo. Esto nos permite no depender únicamente de una medida, y cuando se le pide al sensor que nos dé el dato, hace una media aritmética de los valores que tenga en ese momento almacenados en la cola FIFO, lo que suaviza las posibles imprecisiones y los errores de medición.

Si se ejecuta el archivo batsensor.py individualmente, se pueden ver los tres sensores en acción, cada uno reaccionando independientemente de los demás, tomando medidas aunque nadie se lo ordene desde fuera, dando unos valores de distancia suavizados y bastante fiables.

5. Conclusión

Hemos conseguido todos los objetivos en la entrega de la memoria del ante-proyecto. Nos hemos encontrado con algunas dificultades con los sensores, ya que al ejecutar los tres sensores a la vez aparecía un problema eléctrico o de interferencias de uno con otro, donde había "ráfagas" de números incorrectos.

Después de solucionar el problema (implementando un timer por sensor) nos hemos encontrado con otro problema al evitar obstáculos. El coche tarda demasiado tiempo en frenar, principalmente porque no tiene freno, por lo que si se va a una velocidad medianamente alta, se chocará igualmente. La solución es ir a una velocidad moderada para evitar el choque. Aun siendo tarde y sin más tiempo para hacer pruebas y experimentos, se nos ha ocurrido una posible forma de solucionarlo: usando un PWM para hacer un cambio de sentido en los motores, a una frecuencia más baja y forzar una detención más rápida.

Con más tiempo este proyecto se puede mejorar mucho, pero ha sido una buena toma de contacto con el mundo de la robótica. Nos ha gustado este proyecto.

6. Referencias

Librería Pygame: http://www.pygame.org/wiki/CompileUbuntu
Librería Pyro4: https://pypi.python.org/pypi/Pyro4
Control motores: http://www.instructables.com/id/Raspberry-PI-L298N-Dual-H-Bridge-DC-Motor/step4/Code-Snippet-Python-sample-make-motor-turn-one-way/
Sensor de proximidad (documentación y ejemplo de código para Arduino): http://www.elecfreaks.com/store/bat-ultrasonic-sensor-distance-measuring-p-743.html
Servo (ejemplo de control): http://razzpisampler.oreilly.com/ch05.html