UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

UNIVERSIDAD COMPLUTENSE DE MADRID

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

TRABAJO FIN DE MÁSTER

"Control de un robot basado en Raspberry"

Autor:	***Juan José Romero Marras***
Directores:	***Luis de la Torre Cubillo***
	Dictino Chaos García
	Jesús Chacón Sombría

Curso 2016/2017

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

UNIVERSIDAD COMPLUTENSE DE MADRID

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

TRABAJO FIN DE MÁSTER

"Control de un robot basado en Raspberry"

Proyecto Tipo A: Proyecto específico propuesto por un profesor.

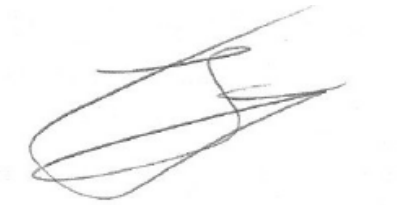
Autor:	***Juan José Romero Marras***
Directores:	***Luis de la Torre Cubillo***
	Dictino Chaos García
	Jesús Chacón Sombría

Curso 2016/2017

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:



Juan José Romero Marras

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENFA DEL TRABAJO FIN DE MASTER

Fecha: 07/06/2017

Quién se suscribe:

Autor:	Juan José Romero Marras
D.N.I.:	11083316-F

Hace constar que es la autor(a) del trabajo:

Titulo completo del trabajo.	Control de un robot basado en Raspberry

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente, en otra revista.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

Resumen

El presente trabajo de fin de máster tiene como objetivo el uso de un robot, basado en la arquitectura Raspberry Pi, para convertirlo en un robot tele-operado que pueda programarse remotamente a través de una interfaz web. El robot utilizado es el modelo comercial GoPiGo 2 de la compañía Dexter Industries.

Para conseguir que el robot sea tele-operado se ha creado una interfaz gráfica en JavaScript, mediante el programa EjsS, que permite al usuario:

1. Controlar el robot tanto manual como automáticamente.
2. Mostrar la lectura de todos los sensores y estado del robot en tiempo real.
3. Configurar cualquier parámetro del robot.
4. Visualizar en todo momento, a través de un mapa, la posición del robot y los obstáculos.
5. Poder implementar un programa, en Python, que sea ejecutado de forma autónoma por el robot. A través de este programa, el usuario podrá desarrollar cualquier algoritmo para dotar de inteligencia al robot, pudiendo controlar tanto sus movimientos como la construcción del mapa que se visualizará en la interfaz web.

Para poder comunicarnos con el robot, se ha desarrollado un servidor Remote Interoperability Protocol (RIP) implementado en Python. Este servidor, que funciona dentro del sistema operativo de la Raspberry Pi del robot, es el encargado de acceder al API del robot para enviarle los comandos necesarios y de atender las peticiones de la interfaz web.

Palabras claves

Raspberry Pi, servidor RIP, GoPiGo, interfaz web, EjsS, Python.

Contenido

Índice de ilustraciones

Índice de tablas

Capítulo 1: Introducción

El objetivo de este Trabajo Fin de Master es conseguir tele-operar desde una interfaz web un robot basado en la arquitectura Raspberry Pi, comunicándose con el robot a través de una red Wifi.

El robot utilizado para este proyecto es el robot GoPiGo 2 desarrollado por la compañía Dexter Industries [1]. Se trata de un kit para robots, basado en la arquitectura Raspberry Pi 3. La programación de este robot se realiza a través de una API en Python, tal y como se explica en capítulos posteriores.

Para poder comunicarnos con el robot se ha aprovechado la arquitectura que ofrece su Raspberry Pi para desarrollar un servidor Remote Interoperability Protrocol (RIP) [2], basado en Python, que permite tanto la comunicación a través de la Wifi con el interfaz web como la comunicación con la API del robot.

En anteriores trabajos ya se ha utilizado la combinación del EjsS y el servidor RIP tanto para la creación de laboratorios virtuales remotos [3] [4] como para el control de robots autónomos. En [5] podemos encontrar una solución, muy parecida a la desarrollada en este trabajo, donde se permite el control de un robot de forma remota, tanto a través del uso de tecnología Wifi como a través de un módulo Xbee. En [6], por ejemplo, se ha utilizado una tableta con una interfaz Android que controla un robot basado en una placa Raspberry Pi estableciendo una comunicación Wifi entre ambos dispositivos.

También podemos encontrar otros trabajos donde se han empleado otros tipos de tecnología para la comunicación con el mismo robot que se utiliza en este trabajo. Por ejemplo, en [7] esta comunicación se estable a través de dispositivos Bluetooth.

Sin embargo, en este trabajo se ha querido ampliar estas soluciones, para no solo permitir tele-operar al robot de forma manual, sino también, dotar al usuario de la posibilidad de poder crear sus propios programas y que éstos sean ejecutados por el robot. De esta forma se dotará al robot de inteligencia y éste podrá operar de forma autónoma. Para realizar

esta tarea se ha incluido en la interfaz un editor de código Python, donde el usuario puede crear sus propios programas, que posteriormente cargará en el robot.

Por otro lado, a diferencias de otros trabajos en los cuales el servidor RIP está desarrollado en Node.js, en este trabajo se ha utilizado un nuevo servidor desarrollado en Python [8], perfecto para su integración con el API del robot.

Como tarea secundaría, además, se implementará un programa de ejemplo para demostrar la funcionalidad de la interfaz. Dicho programa hará que el robot explore de forma autónoma un recinto y cree un mapa a partir de la información de odometría y sus sensores.

Tras esta primera introducción, el resto del trabajo se organiza de la siguiente manera:

- En el siguiente apartado, se presenta la arquitectura desarrollada y sus componentes.
- En el capítulo 3 se describen las pruebas experimentales realizadas para la demostración del funcionamiento del trabajo.
- Finalmente, en el capítulo 4 se presentan las conclusiones y las posibles líneas futuras de desarrollo.

Capítulo 2: Arquitectura

Los principales elementos que componen este trabajo son:

- **El Robot:** El robot utilizado es el modelo GoPiGo 2 de la compañía Dexter Industries. Veremos sus características más adelante.
- **La placa Raspberry Pi:** Entre otros componentes, el robot incluye una placa Raspberry Pi, donde se ejecutará el servidor RIP desarrollado.
- **La placa Dexter Industries GoPiGo 2:** Esta placa se conecta directamente a la Raspberry Pi y sirve de interfaz entre ella y los sensores y actuadores del robot. Gracias a su API permite la programación del robot.
- **El servidor RIP:** Para realizar la comunicación entre la interfaz web y el robot se ha desarrollado un servidor RIP. Este servidor se ejecutará en la propia placa Raspberry Pi.
- **La interfaz web:** La interfaz web se ejecutará en el pc cliente, a través de ella el usuario podrá controlar al robot.

Uno de los principales requisitos es que la comunicación entre el pc cliente, donde se ejecuta la interfaz web, y el robot sea a través de una red Wifi. Gracias a la placa Raspberry PI esto es posible, ya que esta placa incluye un módulo de comunicaciones Wifi. Esto permite ejecutar el servidor RIP directamente en el robot bajo una dirección IP (bien pública o bien privada).

En la ilustración 1 se muestra el esquema de comunicaciones que se han desarrollado en este trabajo.

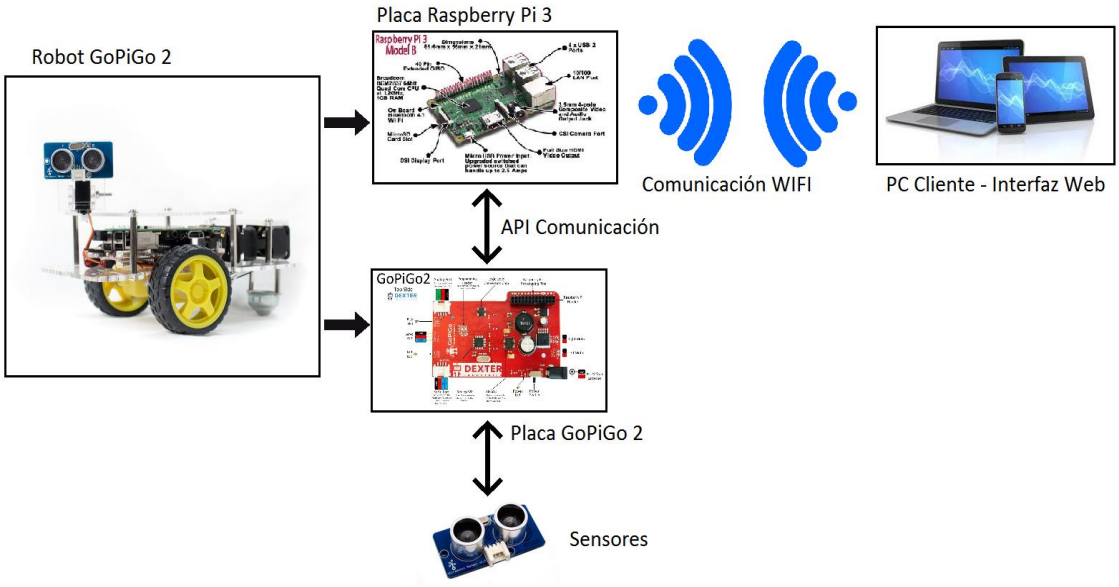


Ilustración 1 - Arquitectura de comunicaciones

La comunicación es la siguiente.

1. A través de la interfaz web, el usuario envía los comandos al robot. Esto es posible ya que la interfaz tiene un cliente RIP que se comunica con el servidor.
2. El servidor RIP está funcionando en la placa Raspberry Pi, escuchando las peticiones de la interfaz web. Cuando le llega una petición la procesa y utilizando el API del robot le envía un comando a través de la placa Dexter Industries GoPiGo 2.
3. La placa Dexter Industries GoPiGo 2, procesa el comando y envía las órdenes a los actuadores y sensores conectados a ella.

A continuación, veremos en detalle cada uno de los diferentes componentes de la arquitectura.

2.1 Descripción y montaje del robot

En este apartado se describe el robot GoPiGo 2, el cual se ha utilizado para el desarrollo de este trabajo de fin de master.

El robot GoPiGo 2 (ilustración 2) es un robot comercial desarrollado por la empresa

Dexter Industries [1]. Se trata de un kit que permite construir nuestro propio robot en forma de “coche”.

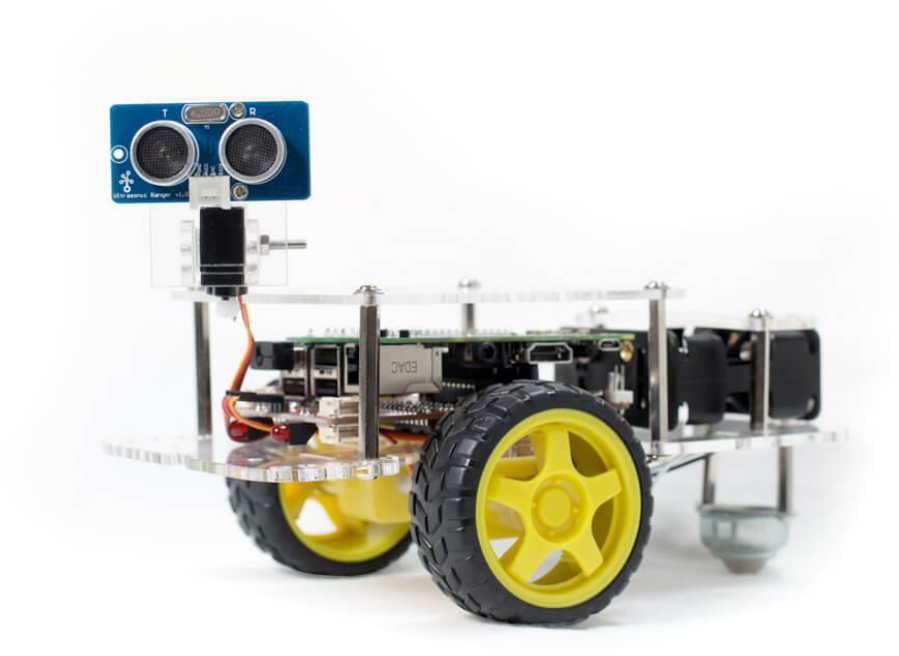


Ilustración 2- Robot GoPiGo 2

El robot facilita al usuario una plataforma móvil, en una arquitectura diferencial, donde tenemos dos ruedas motrices, cada una conectada a un motor, y una tercera rueda loca de apoyo. Sobre esta plataforma se posiciona el resto de elementos del robot.

Dispone de un API, en diferentes lenguajes de programación, mediante el cual podemos desarrollar nuestros propios programas para el robot.

2.1.2 Componentes del robot GoPiGo 2

Como se ha indicado anteriormente, el robot GoPiGo 2 es un kit de desarrollo (ilustración 3) en el cual podemos encontrar varios componentes, los principales son:

Ilustración 3 - Kit desarrollo GoPiGo 2

- **Placa Raspberry Pi**: Se trata de la placa Raspberry Pi 3 Model B, la veremos en detalle más adelante.
- **Tarjeta micro SD**: Tarjeta con una imagen Raspbian preparada para el desarrollo con el robot.
- **Placa Dexter Industries:** Esta placa está desarrollada por la propia compañía Dexter Industries, la explicaremos en detalle más adelante. Sirve de interfaz entre la placa Raspberry Pi y los diferentes dispositivos que se pueden conectar al robot.
- **Sensor ultrasonido:** el kit incorpora un sensor de ultrasonido, que puede ser montando sobre un motor servo para poder girarlo. Permite detectar cualquier obstáculo y facilita la navegación del robot.
- **Un servomotor**: sobre el cual se puede montar el sensor de ultrasonido para poder girarlo.
- **Dos motores:** donde se acoplarán las ruedas del robot.
- **Dos encoders**: Los cuales se pueden acoplar fácilmente a los motores del robot. Gracias a ellos, será posible realizar los cálculos de odometría para poder estimar la posición del robot.

- **Pack de baterías**: Se puede o bien conectar el robot directamente a la corriente eléctrica, conectando la placa Raspberry Pi, o bien utilizar el pack para pilas que incorpora el robot, el cual se conecta directamente a la placa de Dexter Industries.

2.1.3 Montaje del robot GoPiGo 2

En la propia página de Dexter Industries [9] se pueden encontrar las instrucciones para realizar el montaje del robot. Básicamente consiste en acoplar la tarjeta de Dexter Industries a la tarjeta Raspberry Pi y después conectar los sensores a esta tarjeta. Cada sensor solamente se puede conectar una posición determinada, lo que facilita el montaje. Hay que tener presente que este tipo de robot está pensado para estudiantes a partir de 10 años.

Ilustración 3 - Montaje final del robot GoPiGo 2

2.1.3 La placa Raspberry Pi

Entre los principales componentes del robot GoPiGo 2 se encuentra la placa Raspberry Pi.

Raspberry Pi [10] es en realidad un computador de placa reducida o computador de una sola placa (SBC) de bajo coste desarrollado en el Reino Unido, con el objetivo de fomentar la enseñanza de ciencias de la computación en las escuelas. Aunque no se indica si el hardware es open source, si lo es el software, siendo su sistema operativo oficial denominado Raspbian.

Raspbian [11] es una distribución libre basada en el sistema operativo Debian y optimizado para su uso en las placas Raspberry Pi. Contiene herramientas de desarrollo para diferentes IDLE, entre ellos Python, que hemos utilizado en el desarrollo de este trabajo.

Existen diferentes modelos de Raspberry Pi, el modelo que utiliza este robot es la tercera generación de Raspberry Pi.

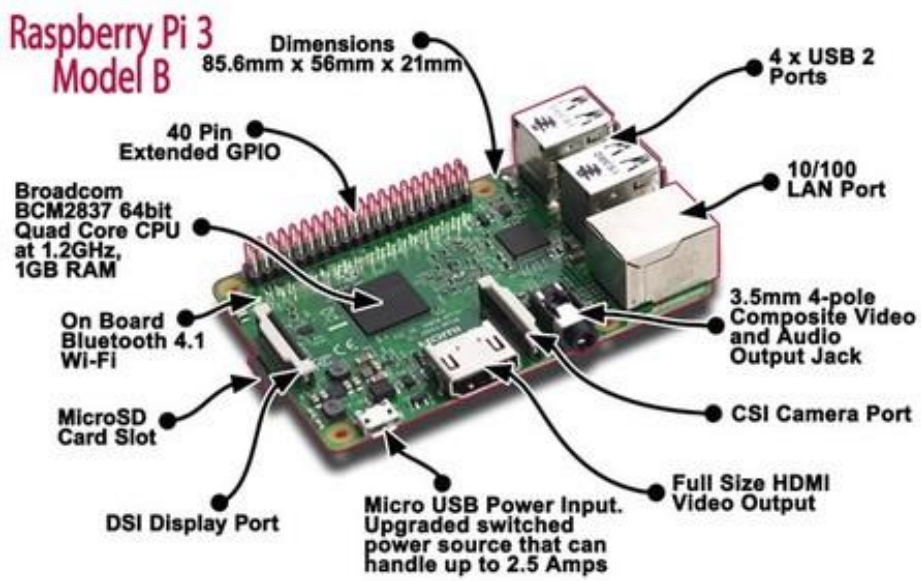


Ilustración 4 - Raspberry Pi 3 Model B

Sus características son:

- Procesador a 1,2 Ghz 64-bit-quad-core ARMv8
- 802.11 Wireless LAN
- Bluetooth 4.1
- 1 GB Ram
- 4 USB puertos
- 40 GPI pins
- Full HDMI puertos
- Ethernet puertos
- Camera Interface (CSI)
- Display Interface (DSI)
- Micro SD

La compañía Dexter Industries proporciona una imagen de Raspbian expresamente adaptada al robot GoPiGo 2. Esta imagen, se puede encontrar en la propia página web de Dexter Industries y contiene todos los programas necesarios para trabajar con el robot. Desde diferentes lenguajes y compiladores para utilizar la API del robot, hasta ejemplos y tutoriales para poder comenzar con el robot desde el primer momento.

2.1.4 La placa Dexter Industries GoPiGo 2

La principal ventaja de este tipo de robot es la propia placa desarrollada por la compañía Dexter Industries.

Esta placa, tal y como ya se ha indicado anteriormente, sirve de interfaz entre los sensores y actuadores conectados al robot y la placa Raspberry Pi. Tiene diferentes conexiones, que permiten al programador acoplar diferentes dispositivos al robot, de forma fácil, ya que cada dispositivo dispone de una conexión única. El principal inconveniente es que no podemos conectar cualquier dispositivo, sino que tienen que estar preparados para trabajar con esta placa. Esto reduce el número de dispositivos que se pueden utilizar con este robot.

En las siguientes ilustraciones se puede observar el aspecto de la placa y los diferentes tipos de conexiones que dispone.

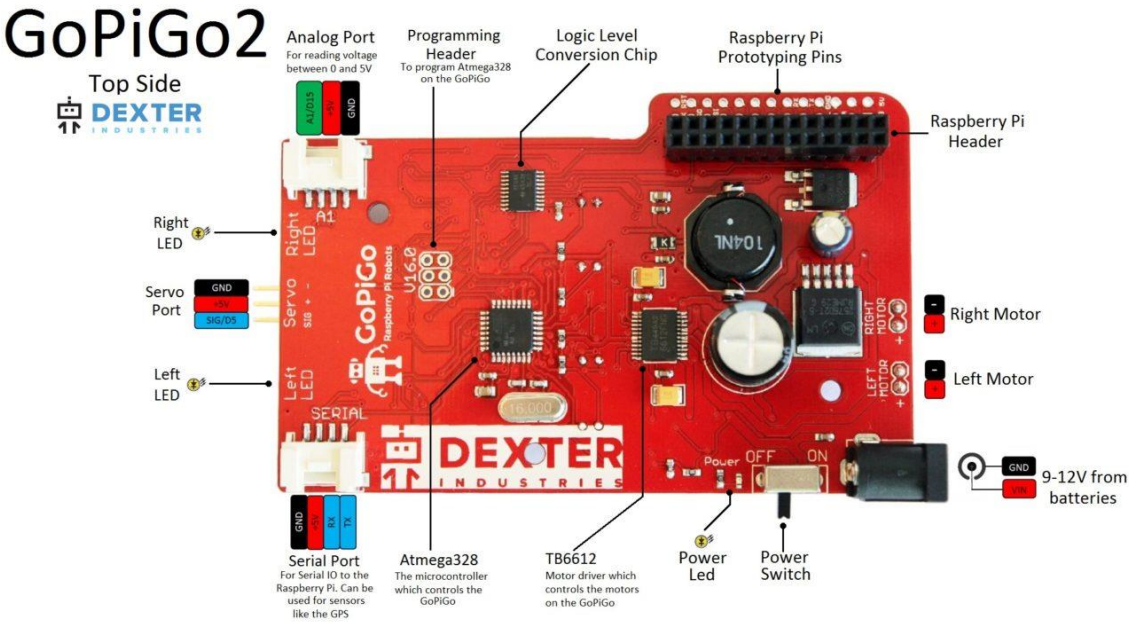


Ilustración 5 - Placa GoPiGo 2 lado de arriba

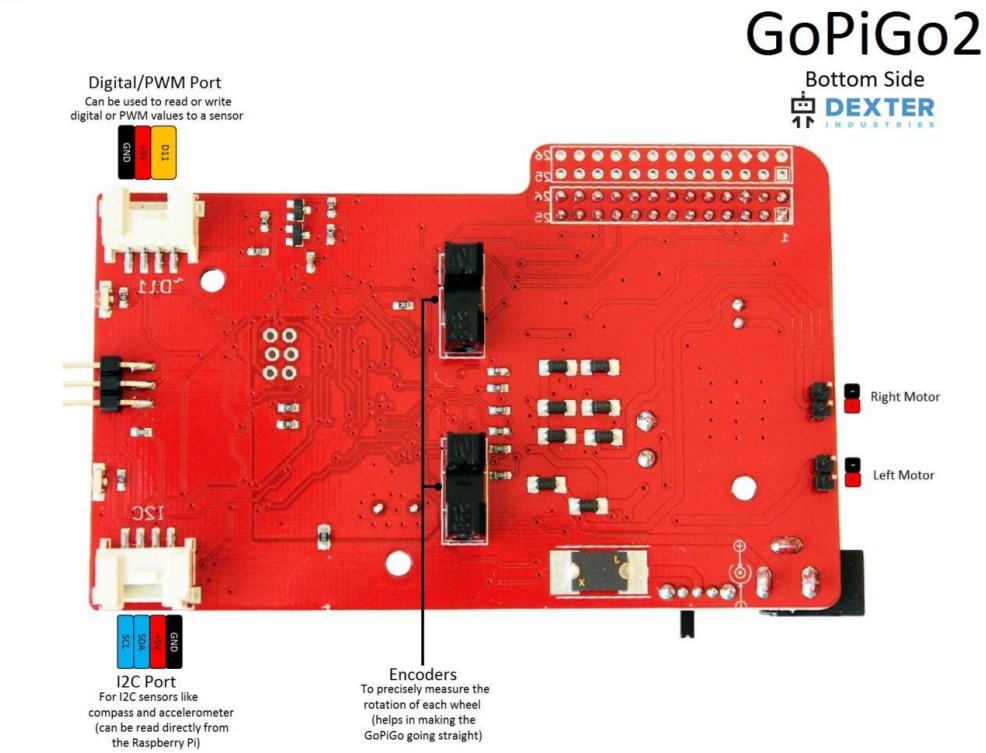


Ilustración 6 - Placa GoPiGo 2 lado de abajo

Sus características son:

- Un puerto analógico: la placa dispone de un puerto analógico el cual permite leer valores de voltaje entre 0v y 5v. Nos permite conectar un sensor analógico, como puede ser el sensor de ultrasonidos.
- Un puerto serie: que puede ser utilizado para conectar diferentes tipos de sensores como un GPS.
- Un puerto digital/PWM: utilizado para leer o escribir datos digitales o PWM (modulación por ancho de pulsos) a un sensor.
- Un puerto I2C: utilizado para sensores como compas o acelerómetros.
- Dos encoders ópticos: la placa contiene dos encoders ópticos para medir la rotación de cada una de las ruedas. La precisión es de 18 pulsos de cuenta por cada rotación.
- Dos conectores, uno para cada motor de las ruedas.
- Un conector para un motor servo: donde podemos conectar el motor servo sobre el cual estará montado el sensor de ultrasonidos.
- Dos conectores para poder conectar dos sensores LED.
- Microcontrolador Atmega 328: todo el control de los motores y escritura y lectura de los sensores es realizada por este microcontrolador. El microcontrolador actúa de intérprete entre la Raspberry Pi y el GoPiGo 2. Envía, recibe y ejecuta los comandos enviados por la Raspberry Pi a través de su API.

2.2 API del robot GoPiGo 2

En este apartado se explicará la API de programación del robot GoPiGo 2.

Como se ha mencionado anteriormente, el robot viene pre-instalado con una API para su funcionamiento. A través de esta API, se puede enviar, de forma muy sencilla, cualquier orden al robot.

Se puede encontrar esta API para diferentes lenguajes (Python o Scratch). Para el desarrollo de este trabajo hemos utilizado Python. En la propia página de Dexter Industries [10] disponen de toda la documentación y tutoriales necesarios para utilizar la

API.

La API proporciona una serie de funciones en Python, encargadas de enviar las órdenes al robot. Las principales funciones son las siguientes:

2.2.1 Funciones para el control de los motores

Función	Descripción
fwd ()	Mueve el robot GoPiGo hacia adelante en línea recta mediante el uso de un controlador PID [13].
motor_fwd()	Mueve el robot hacía adelante pero sin utilizar el controlador PID.
bwd()	Mueve el robot GoPiGo hacia atrás en línea recta mediante el uso de un controlador PID.
motor_bwd()	Mueve el robot hacia atrás sin utilizar el controlador PID.
left()	Mueve el robot hacia la izquierda (el motor derecho estaría parado).
left_rot()	Rota el robot hacía la izquierda (ambos motores estarían moviéndose en dirección contraria).
right()	Mueve el robot hacía la derecha (el motor izquierdo estaría parado).
right_rot()	Rota el robot hacía la derecha (ambos motores estarían moviéndose en dirección contraria).
stop()	Detiene el robot GoPiGo.

Tabla 1 - Funciones para el control de los motores

2.2.2 Funciones de velocidades de los motores

Función	Descripción
increase_speed()	Incrementa la velocidad de los motores en 10 unidades
decrease_speed()	Decrementa la velocidad de los motores en 10 unidades.
set_left_speed(speed)	Configura la velocidad del motor izquierdo. El valor del parámetro speed es un número entero entre 0 y 255.
set_rigth_speed(speed)	Configura la velocidad del motor derecho. El valor del parámetro speed es un número entero entre 0 y 255.
set_speed(speed)	Configura la velocidad de ambos motores. El valor del parámetro speed es un número entero entre 0 y 255.

Tabla 2 - Funciones de velocidades de los motores

2.2.3 Funciones de control de los encoders

Función	Descripción
enc_tgt(m1,m2,target)	Configurar los encoders para que el robot se mueva una distancia, donde: • m1: indica si el encoders del motor 1 se debe de mover la distancia configurada. 0 no se debe de mover, 1 se debe de mover. • m2: indica si el encoders del motor 2 se debe de mover la distancia configurada. 0 no se debe de mover, 1 se debe de mover. • target: número de pulsos que deben de recorrer los encoders.
enable_encoders()	Activa los encoders.
disable_encoders()	Desactiva los encoders.

Tabla 3 - Funciones de control de los encoders

2.2.4 Funciones del sensor de ultrasonidos

Función	Descripción
us_dist()	Devuelve la distancia leída en centímetros por el sensor de ultrasonidos.

Tabla 4 - Funciones del sensor de ultrasonidos

2.2.5 Funciones del sensor led

Función	Descripción
led_on()	Enciende el led.
led_off()	Apaga el led.

Tabla 5 - Funciones del sensor led

2.2.6 Funciones del motor servo

Función	Descripción
enable_servo()	Activa el motor del servo.
disable_servo()	Desactiva el motor del servo.
servo(angle)	Gira el motor del servo hasta la posición, en grados, indicada. El parámetro es el ángulo que girará el motor, su valor es un número entero que debe estar comprendido entre 0 y 180.

Tabla 6 - Funciones del motor servo

2.2.7 Funciones del estado del robot

Función	Descripción
volt()	Indica el estado de la batería en Voltios.
fw_ver ()	Indica el número de versión del firmware del robot.
enable_com_timeuot(time)	Activa el time-out en las comunicaciones (los motores del robot se detendrán solos si no se recibe un comando en el time-out especificado). El parámetro indica el tiempo en milisegundo, y su valor es un número entero debe estará comprendido entre 0 y 65536.
disable_com_timeout()	Desactiva el time-out en las comunicaciones.
read_status()	Indica el estado del robot.
read_timeout_status()	Indica el time-out.

Tabla 7 - Funciones del estado del robot

2.2.8 Ejemplo de uso del API

A continuación, se muestra un ejemplo de uso de esta API. En este ejemplo se ha realizado un programa en Python, en el robot avanza recto durante dos segundos, después retrocede durante otros dos segundos y finalmente se detiene.

```
from gopigo import *
import time

fwd()
time.sleep(2)
bwd()
time.sleep(2)
stop()
```

Ilustración 7 - Ejemplo de uso del API del robot GoPiGo 2

Como se puede comprobar, el uso de la API es ir llamando a las funciones que Dexter Industries pone a nuestra disposición para enviarle las órdenes al robot.

2.3 El servidor RIP

En la tarjeta Raspberry PI se encuentra alojado el servidor RIP, implementado en Python. Este servidor actúa de interfaz entre el robot y la aplicación cliente web. El servidor utilizado está basado en una adaptación del servidor desarrollado por la UNED, cuyo código se puede encontrar en [8].

El servidor utiliza JSON-RPC [14] para el intercambio de información. Recibe y envía la información al robot utilizando su API. Se puede consultar todo su código en el Anexo A.

2.3.1 Preparación del robot

Antes de poder utilizar el servidor RIP en el robot es necesario la instalación de varias librerías en Python para su correcto funcionamiento, las cuales no se encuentran instaladas en la imagen Raspbian que por defecto nos proporciona Dexter Industries.

Las librerías a instalar son las siguientes:

Programa	Comando
jsonrpc	sudo pip install jsonrpc
ujson	sudo pip install ujson
cherrypy	sudo pip install cherrypy

Tabla 8 - Pre-requisitos de funcionamiento del servidor RIP

Todos ellos necesarios para poder ejecutar el servidor RIP correctamente y poder utilizar el protocolo JSON-RPC. También es necesario instalar la **versión 3 de Python,** utilizando el siguiente comando**:**

sudo apt-get install Python

2.3.2 Arquitectura del servidor RIP

El servidor está divido en varias clases, a fin de facilitar su modularidad y futuras ampliaciones. En la siguiente tabla se muestra las principales clases del servidor y cuáles son sus funciones. Su implementación se puede encontrar en el Anexo A.

Clase	Fichero	Descripción
Root	HttpServer.py	Clase principal encargada de levantar el servidor HTTP. Recibe los comandos del interfaz WEB y le envía las respuestas.
RIPGoPiGo	RIPGoPiGo.py	Clase encargada de implementar el servidor JsonRpc. Se encarga de interpretar las tramas recibidas por el servidor HTTP y tratarlas.
GoPiGoConnector	GoPiGoConnector.py	En esta clase está implementada toda la lógica para el funcionamiento con el robot. Contiene el ciclo de ejecución del robot y es la encargada de leer y enviar datos al mismo.

Tabla 9 - Clases del servidor RIP

Una vez iniciado el servidor RIP éste se queda en espera hasta recibir las peticiones desde la interfaz web. La primera petición que debe recibir es la de conectarse con el robot. Esta petición inicia el ciclo de trabajo del robot y devuelve la versión del robot al cual se está conectando. Cualquier otra petición que se reciba antes de iniciar la conexión será ignorada, puesto que el ciclo del robot no está iniciado.

El servidor tiene implementado los métodos de comunicación *connect*, *set* y *get*, cuyo código podemos encontrar en la clase *RIPGoPiGo* del Anexo A.

A continuación, se detallan las variables definidas en el servidor, que pueden ser consultadas y actualizadas a través de los métodos *set* y *get*.

En primer lugar, se muestra las variables que pueden ser actualizas a través del método *set*.

Variable	Tipo	Descripción
setmov	Entero	Indica al robot que realice un movimiento. Los posibles valores son: • 1 = el robot avanzará hacia adelante. • 2 = el robot avanzará hacia atrás. • 3 = el robot girará a la izquierda. • 4 = el robot girará a la derecha. • 5 = el robot rotará a la derecha • 6 = el robot rotará a la izquierda • Cualquier otro valor detendrá al robot
setservo	Entero	Gira el motor del servo hasta la posición, en grados, indica.
setservoEnabled	Entero	Activa o desactiva el motor del servo. Los posibles valores son: • 1 = activa el motor del servo • 0 = desactiva el motor del servo
setm1	Entero	Estable si se debe recoger valores del encoder izquierdo. Los posibles valores son: • 1 = se recogen valores. • 0 = no se recogen valores.
setm2	Entero	Estable si se debe recoger valores del encoder izquierdo. Los posibles valores son: • 1 = se recogen valores. • 0 = no se recogen valores.
setenctgt	Entero	Estable una distancia, en milímetros, que debe recorrer el robot.

setincreasespeed	Entero	Incrementa la velocidad del robot en 10 unidades.
setdecreasespeed	Entero	Decrementa la velocidad del robot en 10 unidades.
setleftspeed	Entero	Configura la velocidad del motor izquierdo al valor indicado.
setrightspeed	Entero	Configura la velocidad del motor derecho al valor indicado.
setspeed	Entero	Configura la velocidad de los motores al valor indicado
setportultra	Entero	Configura el puerto donde estará conectado el sensor de ultrasonidos, por defecto su valor será 15.
setclearinfo	Entero	Elimina el último mensaje de información que tuviera almacenado el servidor.
setscript	Texto	Almacena el programa de control desarrollado por el usuario. Además, indica al robot que debe ejecutar el programa de control.
setclearscript	Entero	Indica al robot que no debe ejecutar el programa de control.
resetmap	Void	Resetea las variables internas del robot a sus valores iniciales.

Tabla 10 - Variables del servidor RIP actualizables a través del método SET

Finalmente, las variables que pueden ser consultadas a través del método *get*.

Variable	Tipo	Descripción
getultra	Entero	Devuelve el valor del sensor de ultrasonidos en centímetros.
getstatus	Entero	Devuelve el estado del robot.
getvolt	Decimal	Devuelve el estado de la batería del robot.
getstatusenc	Entero	Devuelve el estado del encoders.
getencread1	Decimal	Devuelve el valor del encoder izquierdo.
getencread2	Decimal	Devuelve el valor del encoder derecho.
getreadmotorspeed	Decimal	Devuelve la velocidad de los motores.
getmodocontrol	String	Devuelve el modo de funcionamiento del robot. Sus posibles valores son: • "manual" = Si el robot se encuentra en modo manual. • "automatico" = Si el robot se encuentra en modo automático.
getvelleftmotor	Decimal	Devuelve la velocidad del motor izquierdo.
getrightmotor	Decimal	Devuelve la velocidad del motor derecho.
getrobotx	Decimal	Devuelve la posición X del robot.
getrobotY	Decimal	Devuelve la posición Y del robot.
getobstacleX	Decimal	Devuelve una lista con las posiciones X de los obstáculos.
getobstacleY	Decimal	Devuelve una lista con las posiciones Y de los obstáculos.
gethead	Decimal	Devuelve la orientación del robot, en radiales.
getInfo	String	Devuelve el último mensaje de traza.

Tabla 11 - Variables del servidor RIP consultables a través del método GET

Se describe, en los siguientes apartados, en detalle el ciclo de trabajo del robot en el servidor, modos de funcionamiento del robot y cómo se pueden crear programas de

control

2.3.4 Ciclo de trabajo del robot

El esquema de funcionamiento es el siguiente:

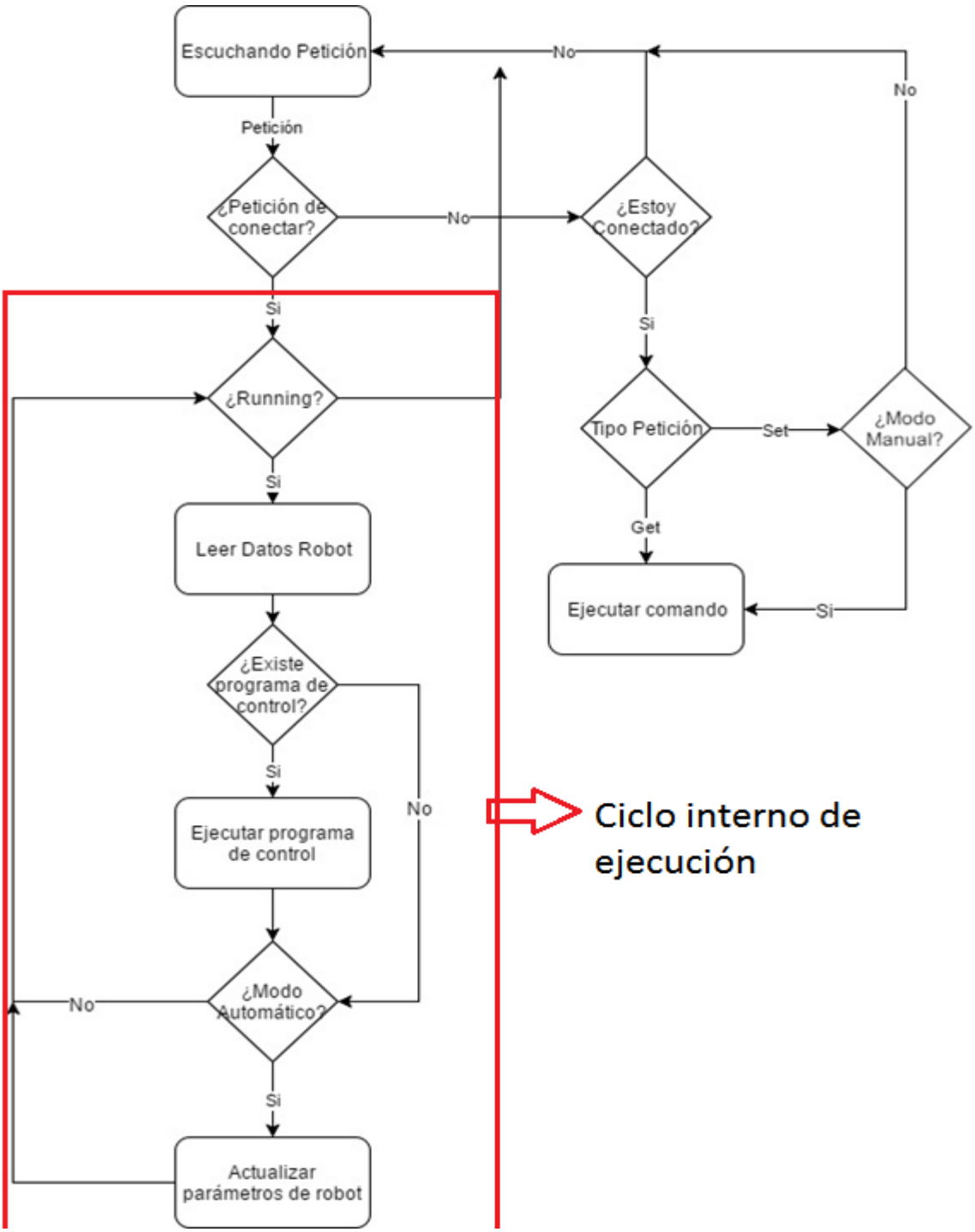


Ilustración 8 – Ciclo de trabajo del robot GoPiGo 2

El funcionamiento es el siguiente:

1. El robot se encuentra a la espera de las peticiones enviadas por el usuario desde la interfaz WEB. Inicialmente, se habrán inicializado todos los parámetros del robot.
2. Al recibir una petición se comprueba si se trata de una petición de conexión.
3. En caso de que se trata de una petición de conexión se inicia el ciclo interno de trabajo del robot.
4. Si la petición es de tipo *get* o *set* se comprueba el robot se encuentra conectado.
5. Si es de tipo *get* y si el robot está conectado se ejecuta el comando *get*.
6. Si es de tipo *set*, si el robot está conectado y se encuentra en modo manual entonces se ejecuta el comando *set*.

El funcionamiento del ciclo interno de trabajo del robot es el siguiente:

1. En la primera petición de conexión se inicia el ciclo.
2. Primero se procede a leer y actualizar toda la información relativa al robot.
3. A continuación, se comprueba si existe un programa de control cargado en el robot.
4. En caso de existir el programa de control éste se ejecuta.
5. Finalmente, y si el robot se encuentra en modo automático, se actualiza sus parámetros y se vuelve al inicio del ciclo.

2.3.6 Modos de funcionamiento del robot

Como se ha indicado anteriormente, existen dos modos de funcionamiento del robot:

- **Modo manual**: Si el robot se encuentra en este modo el usuario podrá controlar el movimiento del robot manualmente desde la interfaz web.
- **Modo automático**: En este estado el robot no se podrá controlar directamente desde el interfaz de usuario, el robot funcionará de forma autónoma. Es a través del programa de control, creado por el usuario, como el robot calcula sus parámetros de ejecución. Estos parámetros son las velocidades del robot y sentido de avance (avanzar, retroceder o detenerse).

2.3.7 Programa de control

Se trata de un programa realizado en Python por el usuario desde la interfaz web. En dicho programa el usuario puede acceder la información del robot, además de indicarle que acciones tiene que realizar ajustando sus parámetros de velocidad y/o sentido de avance.

La interfaz web proporciona:

- Un editor para poder crear el programa de control en Python.
- La posibilidad de poder cargar el programa en el robot de forma remota.
- La posibilidad de poder borrar cualquier programa que tuviera cargado el robot previamente.
- Trazar la información sobre la ejecución del programa de control.

Dentro de su programa, el usuario no puede acceder directamente al API del robot. Pero dispone de toda la información relativa al robot:

- Posición y ángulo actual del robot.
- Velocidades actuales de los motores de las ruedas.
- Estado del robot.
- Estado de la batería del robot.
- Medición del sensor de ultrasonidos.
- Lectura de los encoders del robot.
- Dirección de movimiento del robot.

Con esta información, el usuario puede crear un programa para modificar los parámetros de movimiento del robot:

- Las velocidades de los motores de las ruedas.
- La dirección de avance (avanzar, retroceder o detenerse).

2.3.7.1 La clase Environment

Como se ha indicado anteriormente, el usuario puede acceder directamente al API del robot desde el programa de control. Para solucionar este problema se ha creado una clase *Environment*, donde el usuario puede consultar los datos del robot (velocidades, posición actual etc.), los puede modificar y además puede incluso almacenar sus propias variables. Esta clase se convierte, por tanto, en el punto de intercambio de información entre el programa de control del usuario y el robot.

El código de esta clase se puede encontrar en el Anexo A. Veremos en detalle sus principales propiedades.

Variable	Descripción
VELLEFMOTOR	Indica la velocidad del motor izquierdo.
VELRIGHTMOTOR	Indica la velocidad del motor derecho.
STATUS	Indica el estado del robot.
BATTERY	Indica el estado de la batería.
ULTRASONIC	Indica la medición del sensor de ultrasonidos.
ENCLEFTMOTOR	Indica la lectura del encoder del motor izquierdo.
ENCRIGHTMOTOR	Indica la lectura del encoder del motor derecho.
ENCSTATUS	Indica el estado de los encoders.
robotX	Posición X del robot
robotY	Posición Y del robot
obstacleX[]	Array con las posiciones X de los obstáculos
obstacleY[]	Array con las posiciones Y de los obstáculos
head	Orientación del robot
aux	Diccionario para que el usuario pueda guardar cualquier información entre cada ciclo de ejecución del programa de control.

DIRMOTOR	Indica la dirección de movimiento del robot. Posibles valores: • 1 = Avanzar. • 2 = Retroceder • 99 = Parar.

Tabla 12 - Propiedades de la clase Environment

El uso de esta clase es muy sencillo:

1. Cuando se crea el ciclo interno de ejecución se crea un objeto de la clase *Environment*. Este objeto, es el que será utilizado para el intercambio de información entre el programa del usuario y el robot.
2. Durante la etapa de "Leer Datos del Robot" (ilustración 8) se actualizan las propiedades del objeto. Para actualizar estas propiedades se consulta el estado del robot a través del API. Obviamente, no se actualizan todas las propiedades, solamente las relativas al estado del robot: el estado del robot (STATUS), el estado de la batería (BATERY), el valor del sensor de ultrasonido (ULTRASONIC), el valor y estado de los encoders (ENCRIGHTMOTOR, ENCLEFTMOTOR, ENCSTATUS).
3. Cuando se ejecuta el programa de control, si lo hubiese, se pasa como parámetro el objeto de la clase *Environment*.
4. Dentro del programa de control el usuario puede acceder a las propiedades del objeto, tanto para modificarlas como para consultarlas.
5. Una vez ejecutado el programa de control **y si el robot se encuentra en modo automático**, se procede a configurar las velocidades y dirección de movimiento del robot en función de los valores de las propiedades del objeto.

2.3.7.2 Uso de la clase Environment

El uso y entendimiento de esta clase es fundamental a la hora de poder realizar correctamente el programa de control. A continuación, veremos cómo usar esta clase correctamente, como recuperar la información y como actualizarla.

2.3.7.2.1 Recuperar información del objeto de la clase *Environment*

Lo primero que debemos hacer en nuestro programa es recuperar el parámetro que nos permite acceder al objeto de la clase *Environment*. Cuando se ejecuta el programa de control, dentro del ciclo de ejecución interna del robot, se le pasa como parámetro dicho objeto. Para recuperarlo debemos escribir el siguiente código:

```
while True:
    obj = sys.stdin.read()
    if not obj:
        break;
    enviroment =json.loads(obj)
```

Ilustración 9 – Recuperación del objeto de la clase Environment

El objeto se recupera como un diccionario, en la que cada propiedad es una clave del diccionario. Así, por ejemplo, si queremos acceder a la propiedad robotX debemos escribir el código:

Xpos = environment[“robotX”]

Por supuesto no solamente podemos leer los valores del diccionario, sino que también los podemos modificar. Por ejemplo, si queremos cambiar la velocidad del motor izquierdo escribiremos el siguiente código:

environment[“VELLEFTMOTOR”]=100

2.3.7.2.2 Actualizar la posición del robot

La posición del robot viene determinada por las propiedades *robotX* y *robotY*. Por lo que para actualizar la posición del robot dentro del script debemos de escribir el siguiente código:

environment[“robotX”] = 120

environment[“robotY”] = 50

2.3.7.2.3 Actualizar el mapa de obstáculos

Los obstáculos en el mapa están determinados por el valor de las propiedades *obstaclesX* y *obstaclesY*. Se trata de dos arrays que indican la posición de cada obstáculo en las coordenadas (x, y). Así, por ejemplo, si queremos indicar que hay un obstáculo en la posición (120, 30) escribiríamos el siguiente código:

obstacleX[0] = 120
obstacleY[0] = 30

Si a continuación queremos añadir un nuevo obstáculo en la posición (50, 80) escribiríamos el siguiente código:

obstacleX[1] = 50
obstacleY[1] = 80

2.3.7.2.4 Modificar velocidad y movimiento del robot

Las velocidades de los motores están determinadas por el valor de las propiedades *VELLEFTMOTOR* y *VELRIGHTMOTOR*. Modificando estas propiedades podemos modificar la trayectoria y velocidad del robot, dentro de nuestro programa, siempre y cuando el robot se encuentre en modo automático.

La relación entre las propiedades de la clase *Environment* y los comandos que se envían al robot cuando nos encontramos en modo automático es la siguiente:

Variable Environment	Comando del robot
environment[“VELLEFMOTOR”]	Se envía al robot el comando setleftspeed con el valor de la variable. Este valor debe ser un número numérico comprendido entre 0 y 255.

environment[“VELRIGHTMOTOR”]	Se envía al robot el comando setrightspeed con el valor de la variable. Este valor debe ser un número numérico comprendido entre 0 y 255.
environment[“DIRMOTOR”]	En función de su valor se enviará un comando u otro al robot: • 1 → se envía el comando fwd() • 2 → se envía el comando bwd() • 99 → se envía el comando stop()

Tabla 13 - Relación propiedades de la clase Environment - API del robot

2.3.7.2.5 Guardar información entre cada ciclo de ejecución

Cada vez que se ejecuta un ciclo interno de ejecución se vuelve a ejecutar el programa de control del usuario. En cada ejecución se pasa como parámetro el objeto de la clase *Environment*, por lo que este objeto es siempre el mismo. Es posible, por tanto, almacenar valores internos en el objeto para tenerlos accesible entre cada ciclo de ejecución.

Para poder guardar estos valores el usuario tiene disponible la propiedad *aux*. Se trata de un diccionario en la cual el usuario podrá almacenar cualquier valor. Así, por ejemplo, si queremos acceder a un valor almacenado en esta propiedad podemos escribir el siguiente código:

```
if 'd_left_wheel' in aux:
    prev_left_wheel = aux['d_left_wheel']
    prev_right_wheel = aux['d_right_wheel']
```

Ilustración 10 - Acceso a la propiedad aux de la clase Environment

O si se quiere almacenar, se debe escribir el siguiente código:

```
aux['d_left_wheel']=d_left_wheel
aux['d_right_wheel']=d_right_wheel
aux['rand']=rand
```

Ilustración 11 - Guardar información en la propiedad aux de la clase Environment

2.3.7.2.6 Retornar la información del objeto de la clase *Environment*

Una vez modificadas las propiedades del objeto se debe de retornar su valor en nuestro programa. En caso contrario todas las modificaciones serán ignoradas. Por lo que al final del programa el usuario debe escribir el siguiente código:

```
print(json.dumps(environment))
```

2.4 Creación, aspectos y funcionalidades de la interfaz web

El interfaz ha sido desarrollado utilizando EjsS [14]. En este capítulo describiremos en qué consiste el programa EjsS y el aspecto y funcionalidad del interfaz desarrollado.

2.4.1 Qué es EjsS

Easy Java Simulations (EjsS) es un programa de libre distribución desarrollado en Java que permite crear simulaciones interactivas tanto en Java como en JavaScript, principalmente para uso académico. Su principal ventaja es que puede ser utilizado por usuarios sin grandes conocimientos de programación.

Podemos descargar el programa desde su página web, disponemos de versiones para Windows, Mac y Linux. Su instalación es muy sencilla, basta con seguir las propias instrucciones que encontraremos en su página web.

Su interfaz es muy sencilla e intuitiva. Disponemos de varias pestañas donde configurar nuestra aplicación (Descripción, Modelo y HtmlView). Es en la pestaña de HtmlView donde iremos creando nuestro interfaz de forma gráfica, arrastrando los elementos desde la paleta de objetos. En la propia página web disponemos de un tutorial y de varios ejemplos de su uso.

Se pueden crear todo tipos de aplicaciones: en Java o JavaScript. Para este proyecto se ha desarrollado la aplicación en JavaScript.

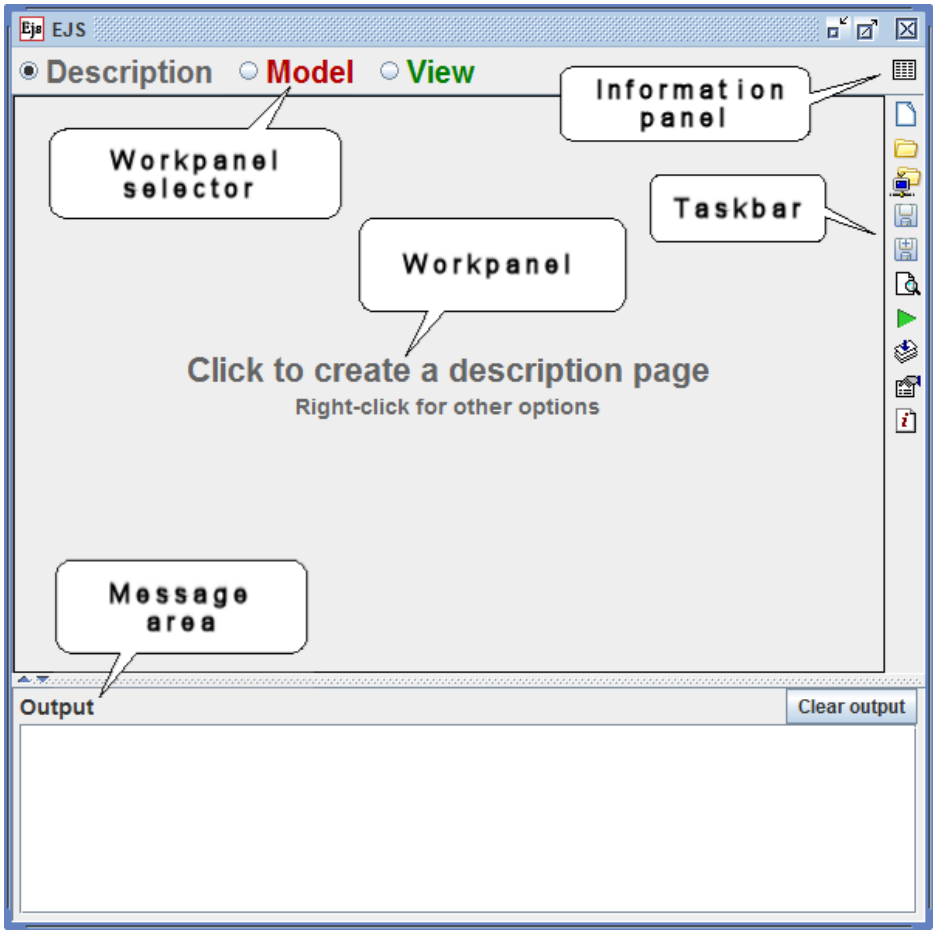


Ilustración 12- Aplicación EjsS

2.4.2 La interfaz de usuario

A continuación, se explicará con detalle la interfaz creada para este proyecto, la cual permite controlar el robot tanto en modo manual como en modo automático.

En el Anexo B se puede encontrar una relación entre las acciones del usuario en el interfaz y el comando que se envía al servidor RIP.

2.4.2.1 Cliente RIP y comunicación con el servidor

La aplicación EjsS dispone de un cliente RIP en JavaScript el cual permite la conexión con el servidor RIP que se encuentra funcionando en la Raspberry Pi del robot.

Podemos encontrar este elemento en la pestaña de Modelo → Elementos (ilustración 13).

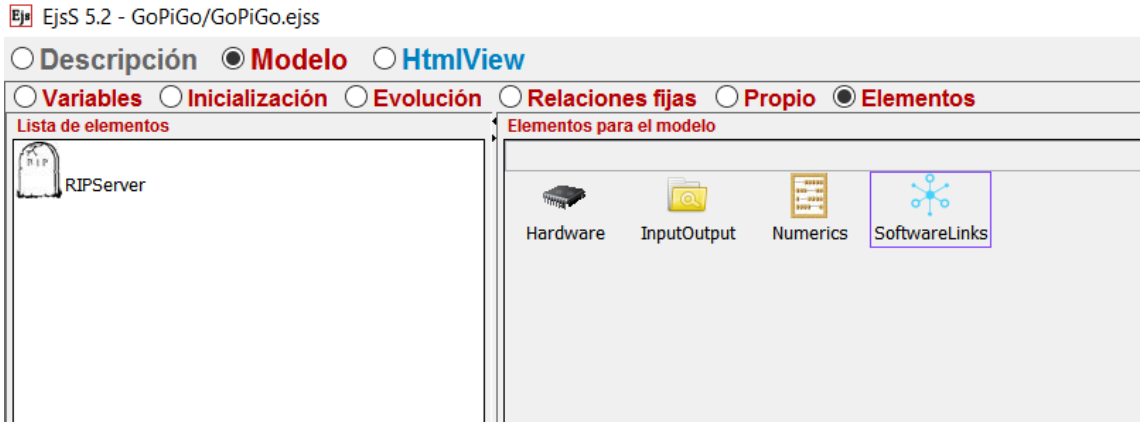


Ilustración 13 – Elemento RIPServer en EjsS

Es necesario configurar la dirección IP de la Raspberry Pi, que es donde se encontrará el servidor RIP, en las propiedades del elemento RIP. Esta configuración se muestra en la ilustración 14.

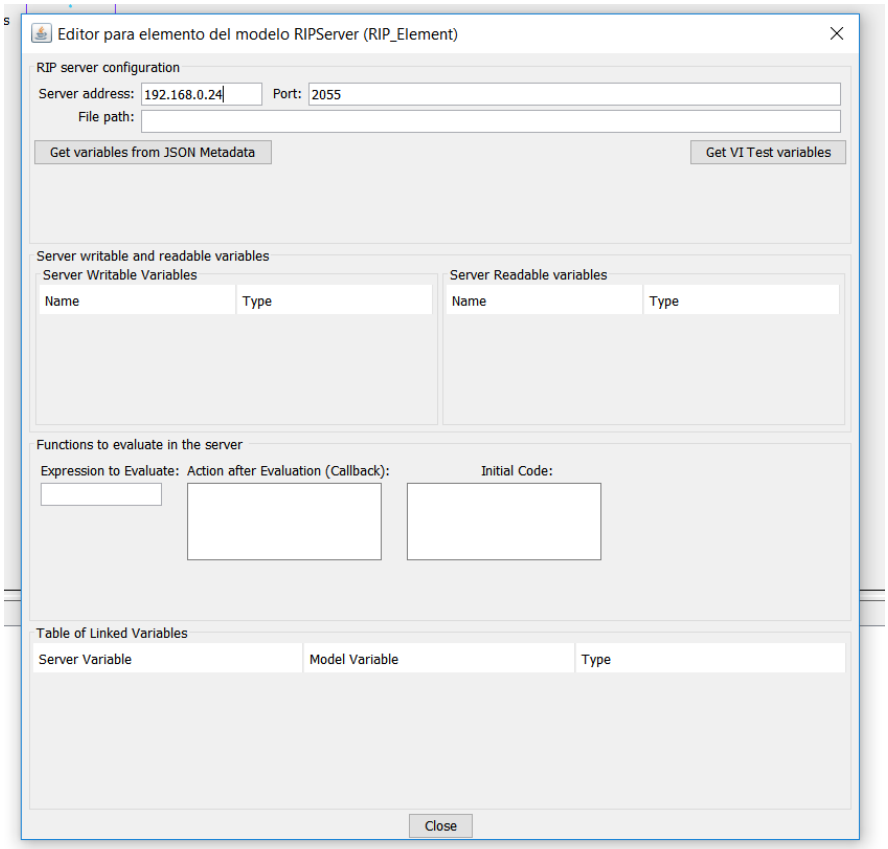


Ilustración 14 - Configuración RIPServer en EjsS

La comunicación entre la aplicación web, a través del elemento RIP, y el servidor RIP se realiza mediante llamadas a los distintos métodos, los cuales se han explicado en la arquitectura del servidor RIP:

- **RIPServer.connect()**: Realiza una llamada al método *connect* del servidor para iniciar la comunicación con el robot.
- **RIPServer.get([variables])**: Realiza una llamada al método *get* del servidor y devuelve su valor. Es utilizado en varias partes de la aplicación para actualizar el interfaz de usuario.
- **RIPServer.set([variables], [values])**: Realiza una llamada al método *set* del servidor actualizando los valores de las variables con los contenidos de los parámetros values. Durante la ejecución del programa se utilizará este método para configurar los parámetros del robot por parte del usuario o para cargar el programa de control en el robot, por ejemplo.

El nombre de las variables es el indicado en las tablas 3 y 4, las cuales podemos consultar en el apartado de la arquitectura del servidor RIP.

A continuación, se verá en detalle cada uno de los diferentes elementos de este interfaz.

2.4.2.2 Panel de Robot

Inicialmente no existirá una conexión con el robot. Las únicas acciones posibles por parte del usuario es la de conectarse al robot a través del botón *Conectar* (ilustración 14). Pulsando dicho botón se enviará un comando de conexión al robot.

Ilustración 15 - Panel Robot

Si la conexión es correcta veremos cómo cambia el interfaz, indicando el número de versión del robot al cual nos estamos conectado. Además, se habilitará el resto del interfaz para poder operar con el robot (ilustración 16).

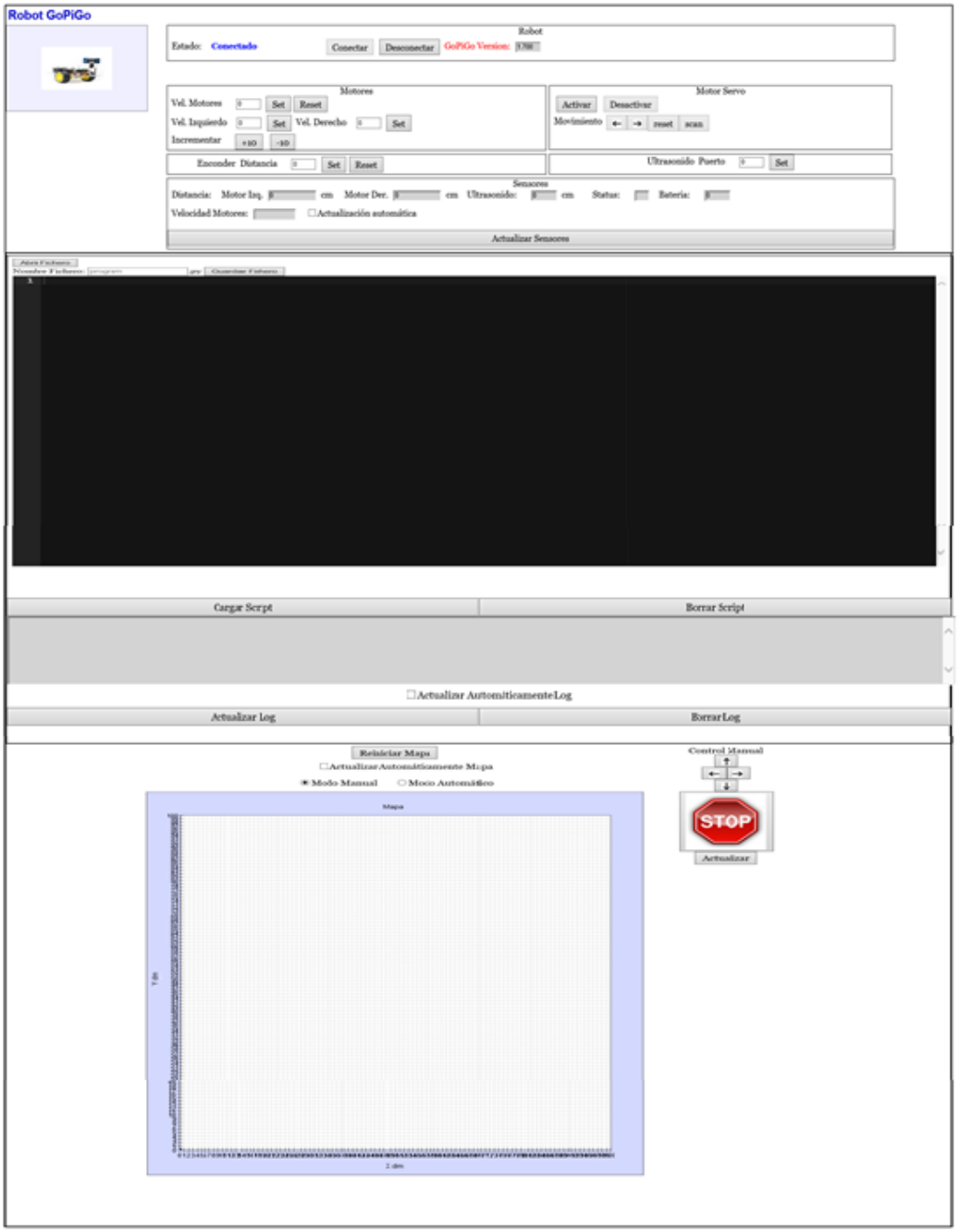


Ilustración 16 - Interfaz web completa

Desde el propio panel podemos desconectarnos del robot, pulsando el botón *Desconectar*. Esta acción no solamente volver a ocultar las acciones del interfaz, sino que también enviará el comando de desconexión al robot. Este comando hará que se detenga cualquier acción que estuviera realizando el robot, quedándose parado inmediatamente.

2.4.2.3 Panel de Motores

Desde este panel se controla las velocidades de los motores. El usuario puede cambiar la velocidad tanto individualmente, para cada motor, como a ambos motores a la vez. También puede incrementar/decrementar la velocidad de los motores de 10 en 10 o resetear los motores a sus velocidades iniciales, que por defecto es 200.

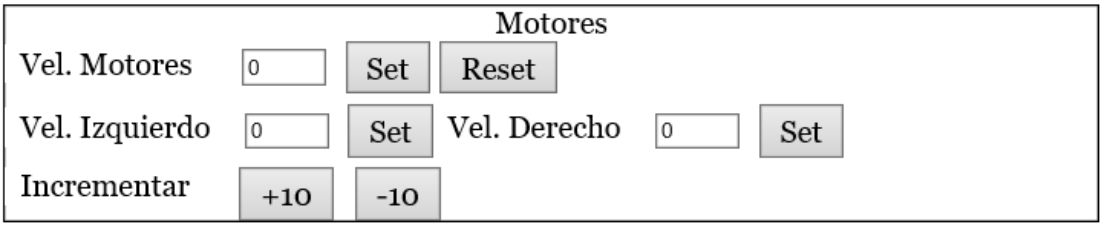


Ilustración 17 - Panel Motores

2.4.2.4 Panel del Motor Servo

A través de este panel se puede controlar el motor servo del robot. En primer lugar, hay que activarlo. Una vez activado se puede: girar el motor hacía la izquierda o la derecha, resetear su posición (el servo se pondrá en ángulo de 90º para orientar el sensor de ultrasonidos hacía delante) o realizar un barrido para girar el motor desde el ángulo 0 hasta el ángulo 180º.

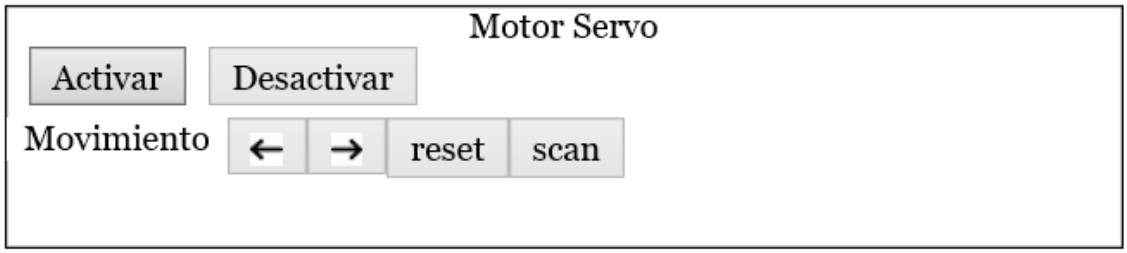


Ilustración 18 - Panel del Motor Servo

2.4.2.5 Panel de Encoders

Mediante este panel el usuario podrá configurar una distancia en milímetros para los encoders que debe de recorrer el robot. Por ejemplo, si el usuario indica un valor de 100 y a continuación indica que el robot se mueva hacía adelante, cuando haya recorrido esa distancia se parará solo.

También es posible resetear el valor de los encoders, a través del botón *Reset*. Esto hará que su cuenta comience otra vez desde 0,

Ilustración 19 – Panel de Enconders

2.4.2.6 Panel de Ultrasonido

En este panel el usuario podrá configurar el puerto donde se encuentra instalado el sensor de ultrasonidos.

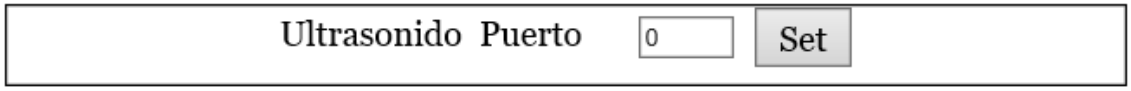


Ilustración 20 - Panel de Ultrasonido

2.4.2.7 Panel de Sensores

En el panel de sensores podemos acceder a toda la información relativa al robot. Esta información consta de: las distancias recorridas por cada rueda, el valor del sensor de ultrasonidos, el estado del robot y la batería y las velocidades de los motores.

El usuario puede o bien refrescar esta información manualmente, mediante el botón de *Actualizar*, o bien activado la opción de *Actualización Automática*, la cual hará que se refresque el panel automáticamente cada segundo.

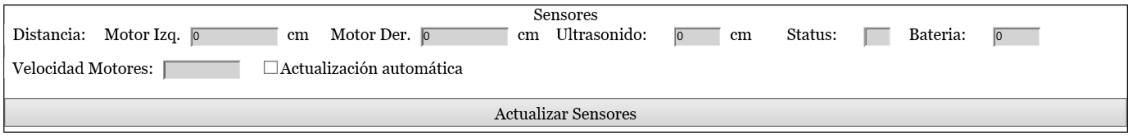


Ilustración 21 - Panel de sensores

2.4.2.8 Editor de Script

El editor de script permite al usuario poder crear sus propios programas en Python para que sean ejecutados por el robot. El editor esta configurador para detectar la sintaxis de Python a fin de facilitar al usuario la programación. Además, dispone de las opciones de G*uardar Fichero*, la cual permite guardar el programa en un fichero en el PC del usuario, y de *Abrir Fichero*, que permite cargar un fichero del PC del usuario en el editor.

Después del editor se encuentran los botones de *Cargar Script* y *Borrar Script*. El primero permite cargar el script en el robot como programa de control. El segundo indicará al robot que debe de dejar de ejecutar el programa de control que actualmente tuviese cargado.

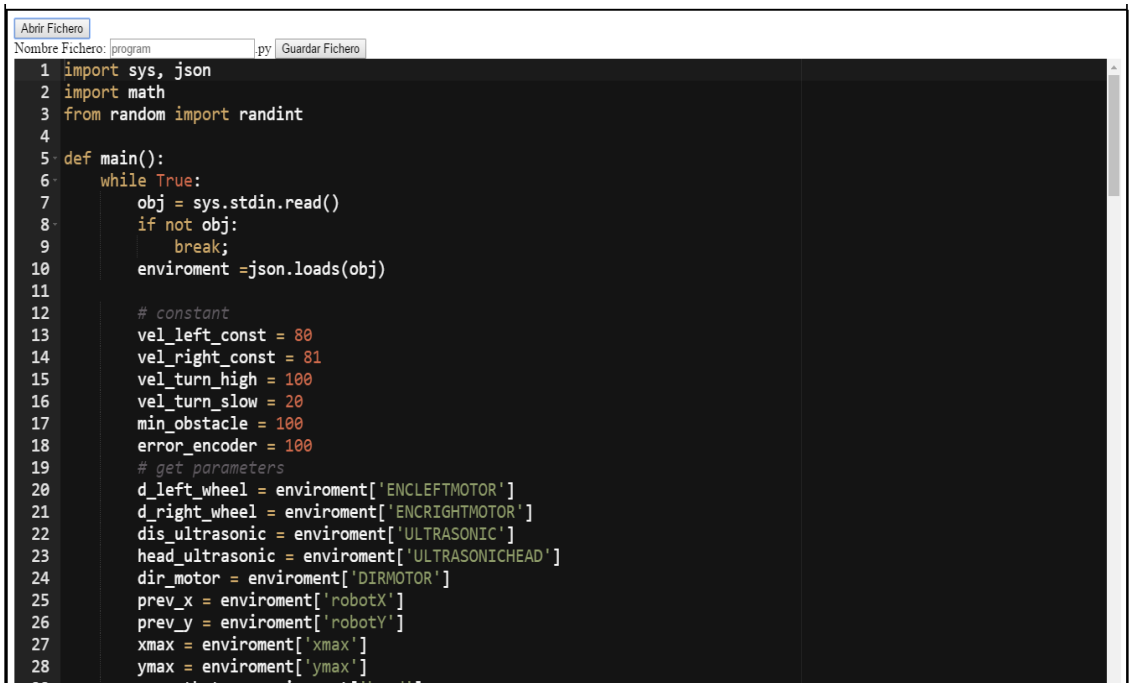


Ilustración 22 - Editor de script

2.4.2.9 Panel de Traza

Debajo del editor del script se encuentra el panel de traza. En este panel se muestra la traza producida tanto en la ejecución del programa de control como en la ejecución del ciclo de trabajo del robot. Cualquier excepción o log es mostrada en este panel.

El usuario puede o bien actualizar la traza manualmente mediante el botón *Actualizar Log*, borrar el panel de traza mediante el botón *Borrar Log* o hacer que la traza se actualice automáticamente marcando la opción *Actualizar Automáticamente Log.*

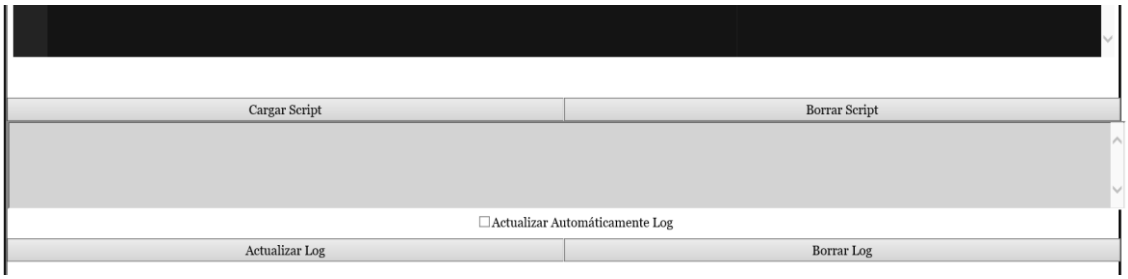


Ilustración 23 - Panel de traza

2.4.2.10 Mapa y control manual del robot

Los últimos controles del interfaz son el panel de mando manual y el mapa de posición del robot.

Con el panel de mando manual el usuario puede:

- Modificar el modo de funcionamiento del robot: Indicando si está en manual o en automático.
- Controlar los movimientos del robot, siempre que se encuentre en modo manual. El usuario podrá hacer que el robot avance, retroceda, gire a izquierda o derecha o se detenga.

En el mapa de posición se refleja tanto la posición del robot como la de los obstáculos detectados. El robot se muestra como un cuadrado de color azul, mientras que los

obstáculos se muestran como un cuadrado de color naranja. Las dimensiones del mapa son de 100dm x100dm. Inicialmente el robot se encontrará en el centro del mapa.

El usuario puede refrescar la información del mapa bien mediante el botón *Actualizar Mapa* o bien activando la opción *Actualiza Mapa Automáticamente*, la cual refrescará el mapa cada segundo.

También es posible resetear el mapa. Esta acción, que se ejecuta pulsando el botón *Reiniciar Mapa*, envía un comando al robot para que:

1. Ajuste la posición del robot a su posición por defecto, situando el robot en mitad de mapa en la posición (50,50).
2. Limpie por completo los obstáculos.

Tal y como se ha explicado en el apartado de la clase Environment, la posición del robot viene determinada por las coordenadas *robotX* y *robotY* y la de los obstáculos por los arrays *obstacleX* y *obstacleY*. Es responsabilidad del usuario el calcular y actualizar correctamente tanto las coordenadas del robot como la de los obstáculos dentro de su programa de control.

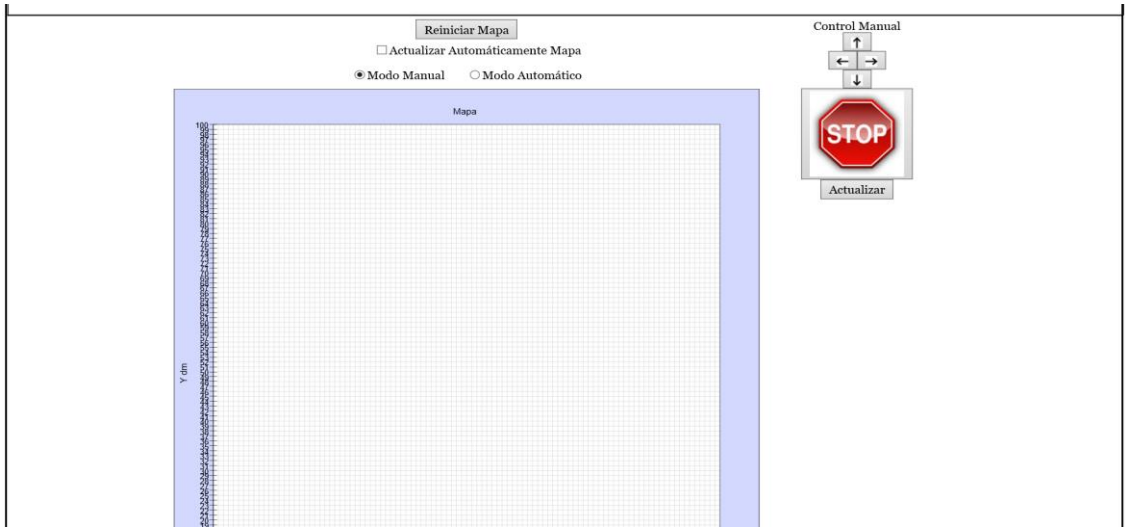


Ilustración 24 – Mapa y control manual

Capítulo 3: Pruebas experimentales

3.1 Objetivo

Como demostración de uso de la interfaz web se ha creado un programa para generar dinámicamente un mapa del entorno del robot. El funcionamiento del programa es muy simple:

- Moverá el robot de manera aleatoria a través del entorno.
- Actualizará la posición del robot en base a los datos de la odometría.
- Comprobará el valor del sensor de ultrasonidos. Si detecta un obstáculo a una distancia menor que la distancia de seguridad lo incluirá dentro del array de obstáculo para que se pueda visualizar en mapa.
- En función de la distancia a la cual se encuentre el obstáculo cambiará la trayectoria del robot (avanzando, girando o retrocediendo) para que pueda seguir explorando el entorno sin chocar.

3.2 Programa

En el Anexo C se puede consultar el código del programa desarrollado. A continuación, se explica en detalle las partes más importantes de su código.

3.2.1 Obtención de los parámetros del robot

Lo primero que hace el programa, en cada ejecución, es obtener los parámetros del robot, consultando las propiedades del objeto de la clase *Environment*.

```
# get parameters
d_left_wheel = enviroment['ENCLEFTMOTOR']
d_right_wheel = enviroment['ENCRIGHTMOTOR']
dis_ultrasonic = enviroment['ULTRASONIC']
head_ultrasonic = enviroment['ULTRASONICHEAD']
dir_motor = enviroment['DIRMOTOR']
prev_x = enviroment['robotX']
prev_y = enviroment['robotY']
xmax = enviroment['xmax']
ymax = enviroment['ymax']
prev_theta = enviroment['head']
aux = enviroment['aux']
wheel_base_lenght = enviroment['wheel_base_lenght']
```

Ilustración 25 - Código para obtener los parámetros del robot

3.2.1 Cálculo de la posición del robot

Para calcular la posición del robot nos hemos basado en los datos de los encoders y los cálculos de odometría [15]. Básicamente en cada ciclo de ejecución se lee los datos de los encoders y se calcula su diferencia con los datos del ciclo anterior.

Para estimar la posición del robot (x, y, α) debemos utilizar la información proporcionada por los encoders, sabiendo que la cinemática del robot es de tipo diferencial. Lo primero, es estimar el desplazamiento realizado por cada rueda en cada pulso registrado por los encoders. Afortunadamente, la API del robot nos proporciona la distancia total recorrida por cada rueda, y esta información es accesible para el usuario a través de las propiedades *ENCLEFTMOTOR* y *ENCRIGHTMOTOR*.

Una vez obtenida la diferencia se calcula posición del robot a través de las fórmulas:

$$\theta_i = \theta_{i-1} + \Delta\theta_i$$

$$x_i = x_{i-1} + \Delta s_{c,i} cos\theta_i$$

$$y_i = y_{i-1} + \Delta s_{c,i} sin\theta_i$$

```
# estimate the wheel movements
prev_left_wheel=0
prev_right_wheel=0
if 'd_left_wheel' in aux:
    prev_left_wheel = aux['d_left_wheel']
    prev_right_wheel = aux['d_right_wheel']
d_ticks_left = d_left_wheel-prev_left_wheel
d_ticks_right = d_right_wheel-prev_right_wheel
d_center = 0.5*(d_ticks_left+d_ticks_right)
if d_ticks_left<error_encoder and d_ticks_right<error_encoder:
    # calculate new pose
    new_theta = prev_theta + ((d_ticks_right-d_ticks_left)/wheel_base_lenght)
    new_x=prev_x
    new_y=prev_y
    if dir_motor == 1:
        new_x = prev_x + ((d_center*math.cos(new_theta))/10)
        new_y = prev_y + ((d_center*math.sin(new_theta))/10)
    elif dir_motor == 2:
        new_x = prev_x - ((d_center*math.cos(new_theta))/10)
        new_y = prev_y - ((d_center*math.sin(new_theta))/10)
```

Ilustración 26- Código para el cálculo posición del robot

3.2.2 Cálculo de la posición del obstáculo

Para realizar el cálculo de la posición de los obstáculos se comprueba la medición del sensor de ultrasonidos. Si esta es menor a una distancia de seguridad entendemos que hay un obstáculo en frente del robot y procedemos a calcular su posición.

El cálculo de la posición se realiza mediante el siguiente código:

```
# Read obstacle
xobs = -1
yobs = -1
if dis_ultrasonic < min_obstacle:
    xobs = new_x + ((dis_ultrasonic+12)*math.cos(new_theta))/10
    yobs = new_y + ((dis_ultrasonic+12)*math.sin(new_theta))/10
```

Ilustración 27 - Código para el cálculo de posición de un obstáculo

Finalmente, una vez que obtenemos la posición la añadimos al array de obstáculos.

3.2.3 Cálculo de la velocidad de los motores

El último cálculo del programa es determinar la velocidad que ha de aplicarse a los motores. La velocidad se calcula en función de la medición del sensor de ultrasonidos, a fin de poder evitar los obstáculos del entorno. Si esta medición se encuentra por debajo de un umbral de seguridad entonces de forma aleatoria el robot gira hacía la derecha o izquierda para evitar el obstáculo.

```
if 'rand' in aux:
    rand = aux['rand']
if rand == -1:
    rand = randint(0,9)
if dir_motor == 0:
    dir_motor = 1

if dir_motor != 2 :
    # robot is moving to forward
    if dis_ultrasonic < 10:
        # turn right
        rand = -1
        enviroment['DIRMOTOR']=2
        enviroment['VELLEFTMOTOR'] = vel_left_const
        enviroment['VELRIGHTMOTOR'] = vel_right_const
    elif dis_ultrasonic < 20:
        enviroment['DIRMOTOR']=1
        if rand%2==0:
            enviroment['VELLEFTMOTOR'] = vel_turn_high
            enviroment['VELRIGHTMOTOR'] = vel_turn_slow
        else:
            enviroment['VELLEFTMOTOR'] = vel_turn_slow
            enviroment['VELRIGHTMOTOR'] = vel_turn_high
    elif dis_ultrasonic < 50:
        # back
        enviroment['DIRMOTOR']=1
        if rand%2==0:
            enviroment['VELLEFTMOTOR'] = vel_turn_high
            enviroment['VELRIGHTMOTOR'] = vel_turn_slow
        else:
            enviroment['VELLEFTMOTOR'] = vel_turn_slow
            enviroment['VELRIGHTMOTOR'] = vel_turn_high
    else:
        rand = -1
        enviroment['DIRMOTOR']=1
        enviroment['VELLEFTMOTOR'] = vel_left_const
        enviroment['VELRIGHTMOTOR'] = vel_right_const
else:
    if dis_ultrasonic<20:
        #turn left
        enviroment['DIRMOTOR']=2
        if rand%2==0:
            enviroment['VELRIGHTMOTOR'] = vel_turn_high
            enviroment['VELLEFTMOTOR'] = vel_turn_slow
        else:
            enviroment['VELRIGHTMOTOR'] = vel_turn_slow
            enviroment['VELLEFTMOTOR'] = vel_turn_high
    elif dis_ultrasonic < 50:
        enviroment['DIRMOTOR']=2
        enviroment['VELRIGHTMOTOR'] = vel_left_const
        enviroment['VELLEFTMOTOR'] = vel_right_const
    else:
        enviroment['DIRMOTOR']=1
        enviroment['VELLEFTMOTOR'] = vel_left_const
        enviroment['VELRIGHTMOTOR'] = vel_right_const
```

Ilustración 28 - Código para el cálculo de la velocidad de los motores

3.3 Pruebas experimentales

El resultado de la ejecución de la interfaz y del programa desarrollado se puede ver en: https://www.youtube.com/watch?v=EhPwDsPvI3U.

En las siguientes ilustraciones se muestran algunos fotogramas de la ejecución.

Inicialmente (ilustración 29) se puede observar cómo se realiza la conexión al robot desde la interfaz Web pulsando el botón de conectar.

Ilustración 29 - Conexión al robot desde la interfaz Web

Si la conexión ha sido satisfactoria (ilustración 30) se muestra el resto de la interfaz, tal y como se puede observar en el siguiente fotograma.

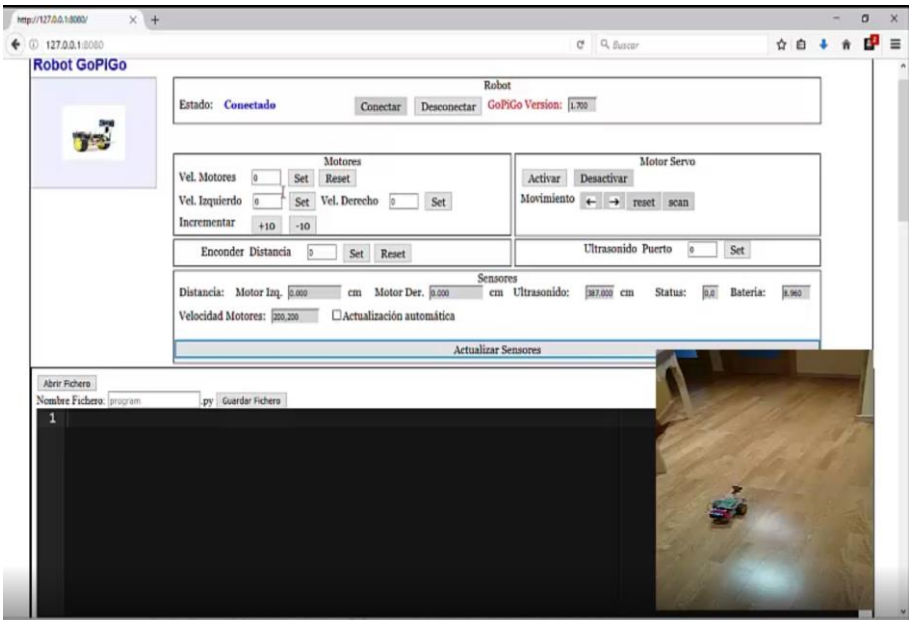


Ilustración 30 - Conexión establecida con el robot GoPiGo desde la interfaz Web

A continuación, se abre, con el editor de scripts (ilustración 31), el programa de control desarrollado.

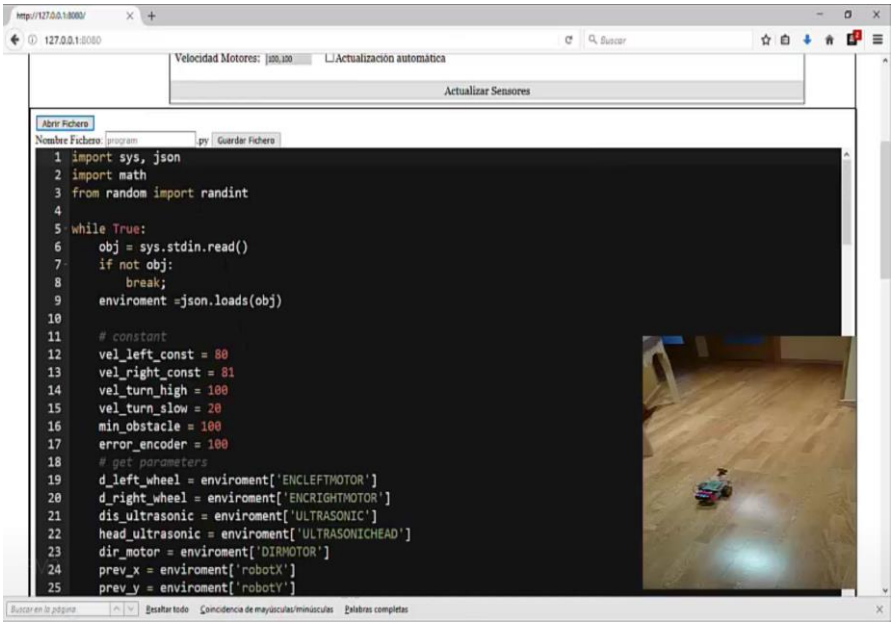


Ilustración 31 - Programa de control en el editor de scripts

Pulsando el botón de *Cargar Script*, se carga el programa en el robot (ilustración 32). En el fotograma se puede observar como en la traza se muestra el mensaje indicando que se ha cargado correctamente el programa en el robot.

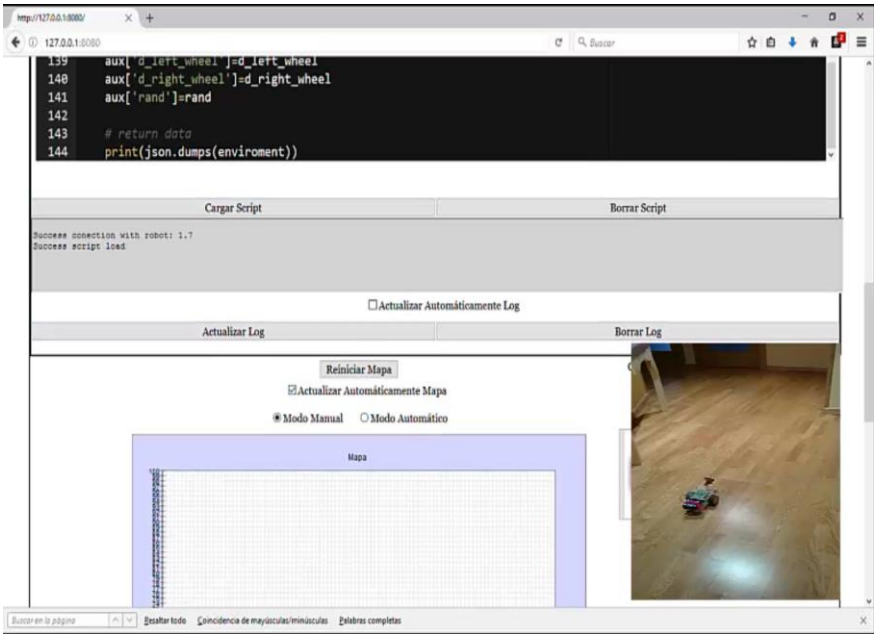


Ilustración 32 - Carga del programa de control en el robot

Inicialmente el robot se encuentra en el centro del mapa (ilustración 33). Se marca la casilla de *Actualizar Automáticamente Mapa* para visualizar en tiempo real su posición.

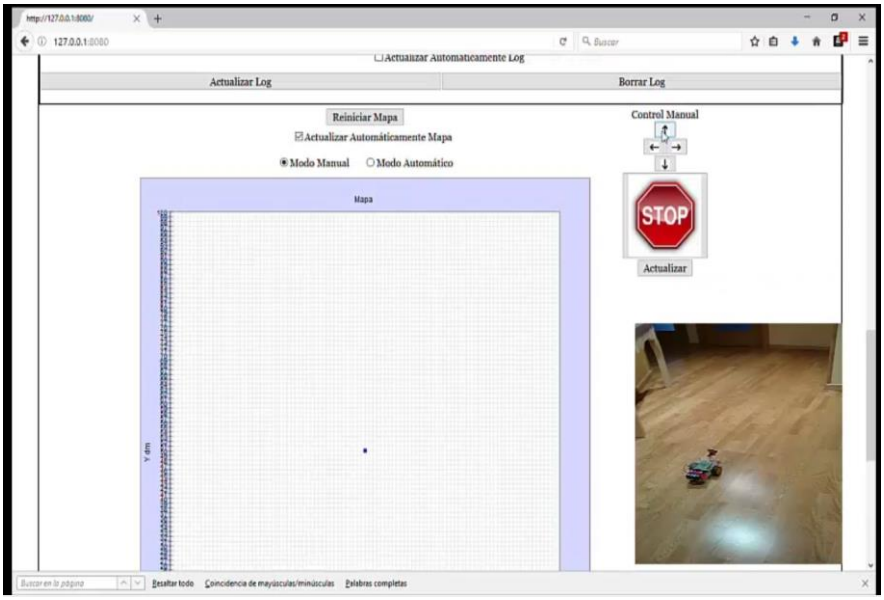


Ilustración 33 - Mapa inicial

A través del *Control Manual*, usando las flechas, se comienza a mover al robot. En el siguiente fotograma (ilustración 34) se puede observar como a medida que se mueve el robot, también se actualiza el mapa con los obstáculos detectados y la posición del mismo.

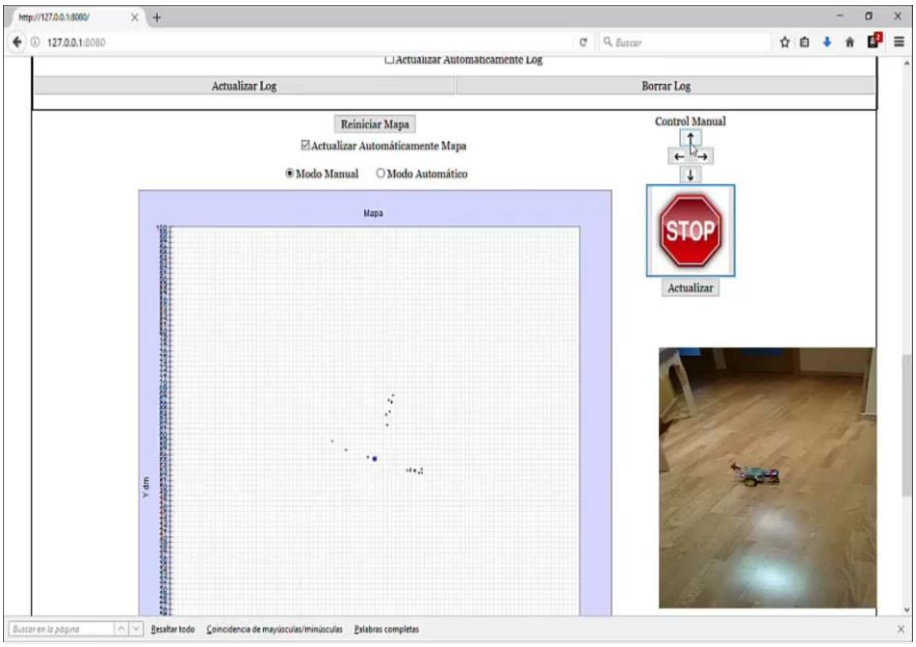


Ilustración 34- Control manual del robot

Finalmente, se reinicia el mapa y se activa el modo automático (ilustración 35). Al reiniciar el mapa se limpia todos los obstáculos detectado anteriormente, y al ponerlo en modo automático el robot comienza a ejecutar la rutina del programa control. En el siguiente fotograma, se aprecia como el robot se encuentra al fondo de la habitación y como el mapa se ha actualizado.

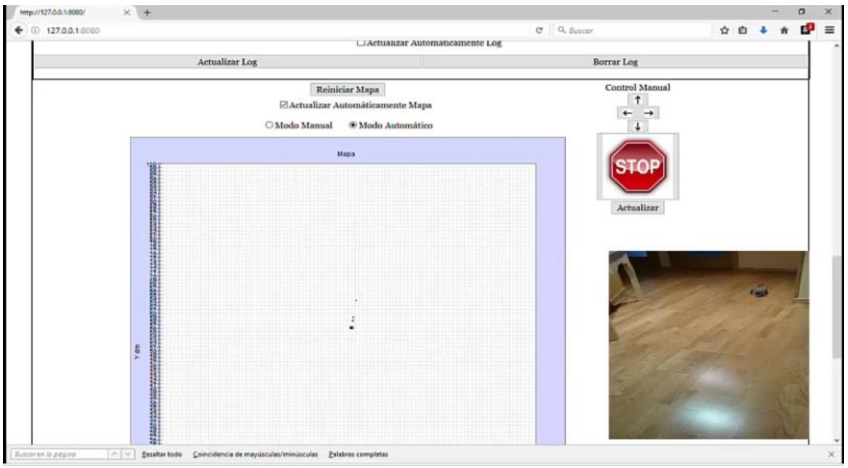


Ilustración 35 - Modo de funcionamiento automático

Capítulo 4: Conclusiones

En el apartado anterior se demostró el correcto funcionamiento de la interfaz web, tanto en modo manual como cuando se ejecuta en modo automático.

En el modo manual la respuesta del robot a las peticiones desde la interfaz es prácticamente inmediata.

En el en modo automático, que es cuando se ejecuta el programa diseñado para la generación del mapa dinámico, se puede comprobar como el robot se mueve por la habitación, intentando esquivar las paredes y construyendo el mapa.

La actualización en tiempo real del mapa de la posición del robot y de los obstáculos encontrados también ha resultado correcta, tal y como se muestra en el apartado anterior (y puede comprobarse mejor en el video adjunto) como a medida que el robot se mueve el mapa se actualiza correctamente.

El objetivo de este trabajo era desarrollar una interfaz remota para controlar el robot, tanto de forma manual como automática, permitiendo crear un programa para que el robot lo ejecutase de manera autónoma.

La comunicación entre el servidor RIP y la interfaz web a través de una red Wifi también ha funciona correctamente y de forma estable. De hecho, aunque no se haya indicado en el apartado anterior, el robot se encontraba en otra habitación distinta a la cual estaba el pc cliente, con el fin de comprobar que la interactividad y realimentación ofrecida por la herramienta era suficiente para controlar el robot.

De este modo se han cumplido satisfactoriamente todos los objetivos del proyecto.

4.1 Limitaciones en el robot

Por supuesto, el programa de control realizado presenta algunas limitaciones. Por un lado, nos encontramos, al ejecutarlo, con los típicos problemas de odometría, como, por ejemplo, cuando las ruedas se deslizan y hace alterar los resultados en el cálculo de posición del robot. Estos problemas son debido a las limitaciones físicas del robot.

Las principales limitaciones detectadas son las debidas a la precisión de sus sensores y actuadores, sobre todo en la calidad de los materiales. Las limitaciones han sido las siguientes:

- El número de sensores de ultrasonidos utilizado, el kit solamente dispone de uno, hace muy difícil realizar una correcta navegación. Es posible rotar el sensor para detectar en radio de 0º a 180º los obstáculos del robot, pero esto haría incompatible que el robot se estuviese moviendo a la vez.
- El sensor de ultrasonidos es de baja precisión y en ocasiones presenta falsos positivos.
- Se ha comprobado que, a pesar de configurar ambos motores con la misma velocidad en ocasiones, sin causa aparente, un motor gira más rápido que el otro. Esto distorsiona la navegación. Por suerte la lectura de los encoders se mantiene coherente.
- Otro problema detectado se encuentra en la lectura de los datos del robot. Si se realiza de forma constante y muy rápida el acceso a los datos del robot, a través de su API, los valores devueltos comienzan o bien a ser incoherentes o a fallar.

Afortunadamente, muchas de estas limitaciones son salvables por parte del usuario a través de la lógica del programa de control que implemente.

4.2 Líneas de desarrollo futuras

La finalidad de este trabajo es realizar una base a partir de la cual se pueda crecer en el futuro.

El desarrollo de la interfaz web permitirá a los usuarios poder desarrollar sus propios programas y algoritmos para ser ejecutados en el robot. En este sentido se puede dotar al robot de un mayor número de sensores para facilitar la navegación. La tarjeta GoPiGo 2 permite conectar GPS, acelerómetros o más sensores ultrasonidos, por ejemplo, y así paliar los problemas encontrados en el cálculo de la posición robot a través de la odometría.

Otra línea desarrollo sería poder incluir una cámara al robot para realizar programas de visión por computador. Tanto la tarjeta GoPiGo 2 como la propia Raspberry Pi permiten la conexión de una cámara, ampliando la interfaz web se propia mostrar lo que está viendo el robot, y mediante el programa de control desarrollar algoritmos de visión por computador.

Bibliografía

[1] Dexter Industries, «GoPiGo Robot Starte Kit,» [En línea]. Available: https://www.dexterindustries.com/shop/gopigo-starter-kit-2/.

[2] J. Chacón , G. Farias, H. Vargas, A. Visioli y S. Dormido, «Remote Interoperability Protocol: A bridge between interactive interfaces and engineering systems,» *IFAC-PapersOnLine*, vol. 48, nº 29, pp. 247-252, 2015.

[3] J. Chacón, . J. Saenz, L. de la Torre y S. Dormido, Air Levitation Virtual andRemote Lab for Teaching Physics, 2016.

[4] J. Saenz, J. Chacón, L. de la Torre, S. Dormido y A. Visioli, Open and Low CostVirtual and Remote Labs on control Engineering, 2016.

[5] A. R. Lopez, Estudio de aplicabilidad de tecnologías inalámbricas para el control de un robot basado en arduino, 2016.

[6] C. Hernández, R. Poot, L. Narváez, E. Llanes y V. Chi, Design andImplementation of a System for Wireless Control of a Robot,» InternationalJournal of Computer Science Issues, 2010.

[7] G. Mester, Wireless Sensor-based Control of Mobile Robots Motion, 2009.

[8] Jesús Chacón Sombría, «UNED, PY-RIPSERVER,» [En línea]. Available: https://github.com/jcsombria/py-ripserver.

[9] D. Industries, «Montaje del robot GoPiGo,» [En línea]. Available: https://www.dexterindustries.com/GoPiGo/getting-started-with-your-gopigo-raspberry-pi-robot-kit-2/1-assemble-the-gopigo-2/assemble-gopigo-raspberry-pi-robot/.

[10] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Raspberry_Pi.

[11] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Raspbian.

[12] D. Industries, «Programación Python robot GoPiGo,» 2016. [En línea]. Available: https://www.dexterindustries.com/GoPiGo/programming/python-programming-for-the-raspberry-pi-gopigo/.

[13] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Controlador_PID.

[14] M. Morley, «JSON-RPC 2.0 Specification,» 2013. [En línea]. Available: http://www.jsonrpc.org/specification.

[15] EajsS, «http://www.um.es/fem/EjsWiki,» [En línea].

[16] R. Siegwart, I. R. Nourbakhsh y D. Scaramuzza, Autonomous Mobile Robots - seconde edition.

Listado de siglas, abreviaturas y acrónimos

API	Application Programming Interface
EjsS	Easy Java/JavaScript Simulations
HTML	HyperText Markup Language
RIP	Routing Information Protocol
Wifi	Wireless Fidelity

Anexo A

HttpServer.py

```
'''
Created on 10/11/2015



@author: jcsombria
'''
import time
import cherrypy
import json
from ws4py.websocket import WebSocket, EchoWebSocket

from app.RIPGoPiGo import RIPGoPiGo

class Root(object):
  exposed = True
  server = RIPGoPiGo()

  @cherrypy.tools.accept(media='application/json')

  def POST(self):
    socket = cherrypy.request.body.fp
    message = socket.read()
    print(message)
    response = self.server.parse(message)
    print(response)
    response = json.dumps(response)
    return response.encode("utf-8")
```

```
if __name__ == '__main__':
  cherrypy.config.update({
    'server.socket_host':'192.168.0.24',
    'server.socket_port': 2055,
    'log.access_file' : './log/access.log',
    'log.errors_file' : './log/error.log'
  })
  config = {
    '/': {
      'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
      'tools.sessions.on': True,
      'tools.response_headers.on': True,
      'tools.response_headers.headers': [
        ('Content-Type', 'application/json'),
        ('Access-Control-Allow-Origin', '*'),
      ],
      'tools.encode.on': True,
      'tools.encode.encoding': 'utf-8',
    },
  }

  cherrypy.quickstart(Root(), '/', config)
```

RIPGoPiGo.py

```
'''
Created on 07/02/2016

@author: Juan J. Romero <jjromeromarras@gmail.com>
'''
from jsonrpc.JsonRpcServer import JsonRpcServer
from app.GoPiGoConnector import GoPiGoConnector
```

```
class RIPGoPiGo(JsonRpcServer):
 '''
 classdocs
 '''

 def __init__(self):
  '''
  Constructor
  '''
  super().__init__()
  self.on('get', 1, self._get)
  self.on('set', 2, self._set)
  self.on('connect', 3, self._connect)
  self.gopigo = GoPiGoConnector()

 # REST API
 def _set(self, variables, values):
  return self.gopigo.set(variables, values)



 def _get(self, variables):
  return self.gopigo.get(variables)

 def _connect(self):
  return self.gopigo.connect()
```

GoPiGoConnector.py

```
'''
Created on 07/02/2017

@author: Juan J. Romero <jjromeromarras@gmail.com>
'''
import os.path
import subprocess, json
import threading
from gopigo import *
from app.Environment import *

class GoPiGoConnector(object):
  '''
  classdocs
  '''

  def __init__(self):
    '''
    Constructor
    '''
    self.modoControl = "manual"
    self.running = True
    self.m1=1
    self.m2=1
    self.environment = Environment(100,100)
    self.velrightmotor = 200
    self.velleftmotor = 200
    self.portultra=15
    self.procpid = -1
    self.script = False
    self.mapping = {
      # Set Methods
```

```
'setmov':self.mov,
'setservo': self.servo,
'setservoEnabled':self.servoenabled,
'setm1':self.setm1,
'setm2':self.setm2,
'setenctgt':self.enctgt,
'setincreasespeed':self.increasespeed,
'setdecreasespeed':self.decreasespeed,
'setleftspeed':self.setleftspeed,
'setrightspeed':self.setrightspeed,
'setspeed':self.setspeed,
'setportultra':self.setportultra,
'setclearinfo':self.clearinfo,
'setscript':self.savefile,
'setclearscript':self.clearscript,
'setresetmap':self.resetmap,
# Get Methods
'getultra': self.getultra,
'getstatus':self.getstatus,
'getvolt':self.getvolt,
'getstatusenc':self.getstatusenc,
'getencread1':self.getencread1,
'getencread2':self.getencread2,
'getreadmotorspeed':self.getmotorspeed,
'getmodocontrol':self.setmodocontrol,
'getvelleftmotor':self.getvelleftmotor,
'getvelrightmotor':self.getvelrightmotor,
'getrobotx':self.getrobotx,
'getroboty':self.getroboty,
'getobstaclex':self.getobstaclex,
'getobstacley':self.getobstacley,
'gethead':self.gethead,
'getinfo':self.getinfo,
# Connection Methods
```

```
    'disconnect':self.disconnect,
    'connect':self.connect
    }
  self.readData()


# Behavior Robot
def setmodocontrol(self, modo):
  self.modoControl = modo
  if modo == "automatico":
    self.servoenabled(1)

def worker(self):
  while(self.running):
      self.readData()
      self.control()
      if self.modoControl == "automatico":
        # update robot's parameters
        self.setleftspeed(self.environment.VELLEFTMOTOR)
        self.setrightspeed(self.environment.VELRIGHTMOTOR)
        self.mov(self.environment.DIRMOTOR)
        self.servo(self.environment.ULTRASONICHEAD)
      time.sleep(0.1)


def connect(self):
  try:
    enable_com_timeout(2000)
    self.mov(99)
    self.environment.reset()
    self.setleftspeed(self.velleftmotor);
    self.setrightspeed(self.velrightmotor);
    self.disconnect()
    self.enctgt(0)
```

```
    self.servo(90)
    self.running = True
    self.thread = threading.Thread(target=self.worker)
    self.thread.start()
    version = fw_ver()
    self.environment.msg = str("Conexion establcida con el robot: "+ str(version))
    return version
  except:
    msg = str("Error inexperado en la conexion:"+ str(sys.exc_info()[0]))
    self.environment.msg = msg.encode("utf-8")
    print(msg)

def disconnect(self):
  self.running = False
  self.modoControl == "manual"
  self.script = False
  self.stopcontrol()
  return stop()

def readData(self):
  try:
    self.environment.STATUS = read_status()
  except OSError as e:
    msg = str("Error leyendo STATUS OSERROR:"+ str(e.strerror))
    self.environment.msg = msg
    print(msg)
    pass
  except:
    msg = str("Error inexperado leyendo STATUS:"+ str(sys.exc_info()[0]))
    self.environment.msg = msg
    print(msg)
    pass

  try:
```

```
      self.environment.BATTERY = volt()
    except OSError as e:
      msg = str("Error leyendo BATTERY OSERROR error:"+ str(e.strerror))
      self.environment.msg = msg
      print(msg)
      pass
    except:
      msg = str("Error inexperado leyendo BATTERY :"+ str(sys.exc_info()[0]))
      self.environment.msg = msg
      print(msg)
      pass

    try:
      self.environment.ULTRASONIC = us_dist(self.portultra)
    except OSError as e:
      msg = str("Error leyendo ULTRASONIC OSERROR error:"+ str(e.strerror))
      self.environment.msg = msg
      print(msg)
      pass
    except:
      msg = str("Error inexperado leyendo ULTRASONIC :"+ str(sys.exc_info()[0]))
      self.environment.msg = msg
      print(msg)
      pass
    try:
      self.environment.ENCRIGHTMOTOR = enc_read(1)
    except OSError as e:
      msg = str("Error leyendo ENCRIGHTMOTOR OSERROR error:"+ str(e.strerror))
      self.environment.msg = msg
      print(msg)
      pass
    except:
      msg = str("Error inexperado leyendo ENCRIGHTMOTOR :"+
str(sys.exc_info()[0]))
```

```
    self.environment.msg = msg
    print(msg)
    pass
  try:
    self.environment.ENCLEFTMOTOR = enc_read(0)
  except OSError as e:
    msg = str("Error leyendo ENCLEFTMOTOR OSERROR error:"+ str(e.strerror))
    self.environment.msg = msg
    print(msg)
    pass
  except:
    msg = str("Error inexperado leyendo ENCLEFTMOTOR :"+ str(sys.exc_info()[0]))
    self.environment.msg = msg
    print(msg)
    pass
  try:
    self.environment.ENCSTATUS = read_enc_status()
  except OSError as e:
    msg = str("Error leyendo ENCSTATUS OSERROR error:"+ str(e.strerror))
    self.environment.msg = msg
    print(msg)
    pass
  except:
    msg = str("Error inexperado leyendo ENCSTATUS :"+ str(sys.exc_info()[0]))
    self.environment.msg = msg
    print(msg)
    pass

def control(self):
  try:
    if self.script:
      self.procpid = subprocess.Popen(['python','-u', 'control.py'],
                    stdin=subprocess.PIPE,
                    stdout=subprocess.PIPE
```

```
                    )
      output = self.procpid.communicate(input=self.environment.toJSON().encode())[0]
      env = json.loads(output.decode('utf-8'))
      self.environment.robotX=env['robotX']
      self.environment.robotY=env['robotY']
      self.environment.obstacleX=env['obstacleX']
      self.environment.obstacleY=env['obstacleY']
      self.environment.VELLEFTMOTOR=env['VELLEFTMOTOR']
      self.environment.VELRIGHTMOTOR=env['VELRIGHTMOTOR']
      self.environment.DIRMOTOR=env['DIRMOTOR']
      self.environment.head=env['head']
      self.environment.aux=env['aux']
      self.environment.ULTRASONICHEAD=env['ULTRASONICHEAD']
      self.environment.msg=env['msg']

    except ValueError as e:
      msg = str("Control ValueError error:"+ str(e))
      self.environment.msg = msg
      print(msg)
      pass
    except KeyError as e:
      msg = str("Control KeyError error:"+ str(e))
      self.environment.msg = msg
      print(msg)
      pass
    except:
      msg = str("Control Unexpected error:"+ str(sys.exc_info()[0]))
      self.environment.msg = msg
      print(msg)
      pass



def stopcontrol(self):
  if self.procpid != -1:
```

```
    try:
      self.procpid.kill()
      self.procpid = -1
    except OSError:
      self.environment.msg = sys.exc_info()[0]
      pass

def getinfo(self):
  return self.environment.msg

def clearinfo(self, value):
  self.environment.msg = ''

def resetmap(self, value):
  self.enctgt(0);
  self.environment.reset();

# REST API
def set(self, variables, values):
  size = len(variables)
  for i in range(size):
    try:
       action = self.mapping[variables[i]]
       action(values[i])
    except:
      msg = str("Set Unexpected error:" + str(sys.exc_info()[0]))
      self.environment.msg = msg
      print(msg)


def get(self, variables):
  result = []
  for name in variables:
    try:
```

```
      action = self.mapping[name]
      result.append(action())
    except TypeError as e:
      msg = str("Get TypeError error:"+ str(e))
      self.environment.msg = msg
      print(msg)
    except:
      msg = str("Get Unexpected error:"+ str(sys.exc_info()[0]))
      self.environment.msg = msg
      print(msg)
  return result

# Motor control
def mov(self, type):
  # reset encoders

  self.environment.DIRMOTOR=1
  if type==1:
    return fwd()
  elif type == 2:
    self.environment.DIRMOTOR=2
    return bwd()
  elif type == 3:
    return left()
  elif type == 4:
    return right()
  elif type == 5:
    return right_rot()
  elif type == 6:
    return left_rot()
  else:
    self.environment.DIRMOTOR=0
    return stop()
```

```
# Servo control functions
def servo(self, range):
  self.environment.ULTRASONICHEAD=range
  servo(range)

def servoenabled(self, enabled):
  if enabled==1:
    enable_servo()
  else:
    disable_servo()

# Enconder functions
def setm1(self, value):
  self.m1=value

def setm2(self, value):
  self.m2=value

def enctgt(self, target):
  if target==0:
    self.environment.reset()
  return enc_tgt(self.m1,self.m2,target)

def getencread1(self):
  return self.environment.ENCRIGHTMOTOR

def getencread2(self):
  return self.environment.ENCLEFTMOTOR

# Motor speed functions
def increasespeed(self, value):
  self.environment.VELRIGHTMOTOR = self.environment.VELRIGHTMOTOR + 10
  self.environment.VELLEFTMOTOR = self.environment.VELLEFTMOTOR + 10
  return increase_speed()
```

```
  def decreasespeed(self, value):
    self.environment.VELRIGHTMOTOR = self.environment.VELRIGHTMOTOR - 10
    self.environment.VELLEFTMOTOR = self.environment.VELLEFTMOTOR - 10
    return decrease_speed()

  def setleftspeed(self,value):
    self.environment.VELLEFTMOTOR = value
    set_left_speed(value)

  def setrightspeed(self,value):
    self.environment.VELRIGHTMOTOR = value
    set_right_speed(value)

  def setspeed(self,value):
    self.environment.VELLEFTMOTOR = self.environment.VELRIGHTMOTOR =
value
    set_speed(value)

  def getmotorspeed(self):
    return (self.environment.VELLEFTMOTOR, self.environment.VELRIGHTMOTOR)

  # Ultrasonic ranger
  def setportultra(self, port):
    self.portultra=port

  def getultra(self):
    return self.environment.ULTRASONIC

  # Status from the GoPiGo
  def getvolt(self):
    return self.environment.BATTERY

  def getstatus(self):
```

```
  return self.environment.STATUS

def getstatusenc(self):
  return self.environment.ENCSTATUS

def getvelleftmotor(self):
  return self.environment.VELLEFTMOTOR

def getvelrightmotor(self):
  return self.environment.VELRIGHTMOTOR

def getrobotx(self):
  return self.environment.robotX

def getroboty(self):
  return self.environment.robotY

def gethead(self):
  return self.environment.head

def getobstaclex(self):
  return self.environment.obstacleX

def getobstacley(self):
  return self.environment.obstacleY

def getfwver(self):
  return fw_ver()

#LED control
def ledon(self, led):
  return led_on(led)

def ledoff(self, led):
```

```
  return led_off(led)


# Script
def savefile(self, text):
  try:
    fobj = open(".//control.py","w")
    fobj.write(text)
    fobj.close()
    self.script = True;
    self.environment.msg = str("Success script load ")
  except TypeError as e:
      msg = str("SaveFile TypeError error:"+ str(e))
      self.environment.msg = msg
      print(msg)
  except:
    msg = str("SaveFile Unexpected error:"+ str(sys.exc_info()[0]))
    self.environment.msg = msg
    print(msg)

def clearscript(self, value):
  self.script=False;
```

Environment.py

```
'''
Created on 14/04/2017


@author: Juan J. Romero <jjromeromarras@gmail.com>
'''
import json


class Environment(object):
```

```
def __init__(self, xmax, ymax):
    self.xmax = xmax
    self.ymax = ymax
    self.VELLEFTMOTOR = 0
    self.VELRIGHTMOTOR = 0
    self.STATUS= 0
    self.BATTERY= 0
    self.ULTRASONIC= 0
    self.ENCLEFTMOTOR= 0
    self.ENCRIGHTMOTOR= 0
    self.ENCSTATUS= 0
    self.wheel_base_lenght=12
    self.reset()

def reset(self):
    self.robotX = self.xmax/2
    self.robotY = self.ymax/2
    self.obstacleX = []
    self.obstacleY = []
    self.head = 1.570796
    self.aux = {}
    self.DIRMOTOR = 0
    self.ULTRASONICHEAD = 0
    self.msg = ""

def toJSON(self):
    return json.dumps(self, default=lambda o: o.__dict__,
        sort_keys=True, indent=4)
```

Anexo B

Panel	Acción	Comando

Robot	“Connectar”	RIPServer.get(['connect'], function(result) {
		_view.txtversion.setValue(result[0]);
		ClearObstaclesMap();
		})

Robot	“Desconectar”	RIPServer.get(['disconnect'], function(result)
		{
		connect=false;
		_view.txtversion.setValue("0.000");
		_stop()
		});

Motores	Vel. Motores → “set”	RIPServer.set(["setspeed"],[parseInt(_view.txtmotorspeed.getValue())]); _view.txtmotorspeed.setValue(parseInt(_view.txtmotorspeed.getValue()));
Motores	Vel. Motores → “reset”	RIPServer.set(["setspeed"],[200]);
Motores	Vel. Izquierdo → “set”	RIPServer.set(["setleftspeed"],[parseInt(_view.txtspeedleft.getValue())]); _view.txtspeedleft.setValue(parseInt(_view.txtspeedleft.getValue()))
Motores	Vel. Derecho → “set”	RIPServer.set(["setrightspeed"],[parseInt(_view.txtspeedright.getValue())]); _view.txtspeedright.setValue(parseInt(_view.txtspeedright.getValue()))
Motores	Incrementar → “+10”	RIPServer.set(["increasespeed"],[10]);
Motores	Decrementar → “-10”	RIPServer.set(["decreasespeed"],[10]);
Motor Servo	“Activar”	RIPServer.set(["servoEnabled","servo"],[1,90]);
Motor Servo	“Desactivar”	RIPServer.set(["servoEnabled"],[0]);

Motor Servo	Movimiento → “←”	rangeServo=rangeServo+10;
		if (rangeServo<0)
		rangeServo=0;
		RIPServer.set(["servo"],[rangeServo]);

Motor Servo	Movimiento → "→"	rangeServo=rangeServo-10;
		if (rangeServo<0)
		  rangeServo=0;
		RIPServer.set(["servo"],[rangeServo])

Motor Servo	Movimiento → "reset"	rangeServo=90; RIPServer.set(["servo"],[rangeServo])

Motor Servor	Movimiento → "scan"	rangeServo=0;
		do
		{
		  RIPServer.set(["servo"],[rangeServo]);
		  RIPServer.get(['ultra'], function(result) {
		  _view.txtultrasensor.setValue(result[0]);
		  });
		  rangeServo=rangeServo+10;
		
		}while(rangeServo<=180);

Enconders	"set"	RIPServer.set(["setm1"],[encoder1]); RIPServer.set(["setm2"],[encoder2]); RIPServer.set(["enctgt"],[parseInt(_view.txtencoder.getValue())]); _view.txtencoder.setValue(parseInt(_view.txtencoder.getValue()));
Enconders	"reset"	RIPServer.set(["setm1"],[encoder1]); RIPServer.set(["setm2"],[encoder2]); RIPServer.set(["enctgt"],[0]);
Ultrasonido	"set"	RIPServer.set(["setportultra"],[parseInt(_view.txtultrasonic.getValue())]); _view.txtultrasonic.setValue(parseInt(_view.txtultrasonic.getValue()))
Sensores	"Actualizar Sensor"	RIPServer.get(['ultra','status','volt','encread1','encread2','readmotorspeed'])
Script	"Cargar Script"	RIPServer.set(["script"],[_view.content.getValue()]);
Script	"Borrar Script"	RIPServer.set(["clearscript"],[0]);

Traza	"Actualizar Log"	RIPServer.get(['getinfo',], function(result) {
		  trace = _view.log.getValue();
		  msg = result[0];
		  if (msg!=''){
		  trace = trace +'\n'+msg;

		_view.log.setValue(trace);
		}
		});

Mapa	“Reiniciar Mapa”	index=0;
		while(index<nPoints){
		x[index]=0;
		y[index]=0;
		index=index+1;
		}
		nPoints=0;
		RIPServer.set(["resetmap"],[0]);

Mapa	“Actualizar Automáticamente Mapa”	RIPServer.get(['getrobotx','getroboty','gethead','getobstaclex','getobstacley'])
Mapa	“Modo Manual”	RIPServer.set(["modocontrol"],["manual"]);
Mapa	“Modo Automático”	RIPServer.set(["modocontrol"],["automatico"]);
Control Manual	“←”	RIPServer.set(["mov"],[movLeft]);
Control Manual	“→”	RIPServer.set(["mov"],[movRight]);
Control Manual	“ ↑ “	RIPServer.set(["mov"],[movForward]);
Control Manual	“ ↓ “	RIPServer.set(["mov"],[movBack]);
Control Manual	“Stop”	RIPServer.set(["mov"],[99]);

Control Manual	“Actualizar”	RIPServer.get(['getrobotx','getroboty','gethead','getobstaclex','getobstacley'],
		function(result) {
		// Robot Position
		posXRobot = result[0];
		posYRobot = result[1];
		// Obstacles Position
		obstaclesX = result[3];
		obstaclesY = result[4];
		num = obstaclesX.length;
		ClearObstaclesMap();
		index = 0;
		while(index<num && index<maxPoints){
		x[index]=obstaclesX[index];
		y[index]=obstaclesY[index];
		index+=1;
		}
		nPoints = num;

		_view.log.setValue(nPoints); });

Tabla 14 - Relación entre las acciones del interfaz de usuario y los comandos que se envía y reciben.

Anexo C

```
import sys, json
import math
from random import randint

while True:
    obj = sys.stdin.read()
    if not obj:
        break;
    environment =json.loads(obj)

    # constant
    vel_left_const = 80
    vel_right_const = 81
    vel_turn_high = 100
    vel_turn_slow = 20
    min_obstacle = 100
    error_encoder = 100
    # get parameters
    d_left_wheel = environment['ENCLEFTMOTOR']
    d_right_wheel = environment['ENCRIGHTMOTOR']
    dis_ultrasonic = environment['ULTRASONIC']
    head_ultrasonic = environment['ULTRASONICHEAD']
    dir_motor = environment['DIRMOTOR']
    prev_x = environment['robotX']
    prev_y = environment['robotY']
    xmax = environment['xmax']
    ymax = environment['ymax']
    prev_theta = environment['head']
    aux = environment['aux']
    wheel_base_lenght = environment['wheel_base_lenght']
```

```
# estimate the wheel movements
prev_left_wheel=0
prev_right_wheel=0
if 'd_left_wheel' in aux:
    prev_left_wheel = aux['d_left_wheel']
    prev_right_wheel = aux['d_right_wheel']
d_ticks_left = d_left_wheel-prev_left_wheel
d_ticks_right = d_right_wheel-prev_right_wheel
d_center = 0.5*(d_ticks_left+d_ticks_right)
if d_ticks_left<error_encoder and d_ticks_right<error_encoder:
    # calculate new pose
    new_theta = prev_theta + ((d_ticks_right-d_ticks_left)/wheel_base_lenght)
    new_x=prev_x
    new_y=prev_y
    if dir_motor == 1:
        new_x = prev_x + ((d_center*math.cos(new_theta))/10)
        new_y = prev_y + ((d_center*math.sin(new_theta))/10)
    elif dir_motor == 2:
        new_x = prev_x - ((d_center*math.cos(new_theta))/10)
        new_y = prev_y - ((d_center*math.sin(new_theta))/10)

    if new_x>=xmax:
        while new_x>xmax:
            new_x=new_x-xmax
    elif new_x<0:
        new_x=new_x+xmax

    if new_y>=ymax:
        while new_y>ymax:
            new_y=new_y-ymax
    elif new_y<0:
        new_y=new_y+ymax
```

```
# Read obstacle
xobs = -1
yobs = -1
if dis_ultrasonic < min_obstacle:
    xobs = new_x + ((dis_ultrasonic+12)*math.cos(new_theta))/10
    yobs = new_y + ((dis_ultrasonic+12)*math.sin(new_theta))/10

# Set Motor Velocity
rand = -1
if 'rand' in aux:
    rand = aux['rand']
if rand == -1:
    rand = randint(0,9)
if dir_motor == 0:
    dir_motor = 1

if dir_motor != 2 :
    # robot is moving to forward
    if dis_ultrasonic < 10:
        # turn right
        rand = -1
        environment['DIRMOTOR']=2
        environment['VELLEFTMOTOR'] = vel_left_const
        environment['VELRIGHTMOTOR'] = vel_right_const
    elif dis_ultrasonic < 20:
        environment['DIRMOTOR']=1
        if rand%2==0:
            environment['VELLEFTMOTOR'] = vel_turn_high
            environment['VELRIGHTMOTOR'] = vel_turn_slow
        else:
            environment['VELLEFTMOTOR'] = vel_turn_slow
            environment['VELRIGHTMOTOR'] = vel_turn_high
    elif dis_ultrasonic < 50:
        # back
```

```
            environment['DIRMOTOR']=1
            if rand%2==0:
                environment['VELLEFTMOTOR'] = vel_turn_high
                environment['VELRIGHTMOTOR'] = vel_turn_slow
            else:
                environment['VELLEFTMOTOR'] = vel_turn_slow
                environment['VELRIGHTMOTOR'] = vel_turn_high
        else:
            rand = -1
            environment['DIRMOTOR']=1
            environment['VELLEFTMOTOR'] = vel_left_const
            environment['VELRIGHTMOTOR'] = vel_right_const
    else:
        if dis_ultrasonic<20:
            #turn left
            environment['DIRMOTOR']=2
            if rand%2==0:
                environment['VELRIGHTMOTOR'] = vel_turn_high
                environment['VELLEFTMOTOR'] = vel_turn_slow
            else:
                environment['VELRIGHTMOTOR'] = vel_turn_slow
                environment['VELLEFTMOTOR'] = vel_turn_high
        elif dis_ultrasonic < 50:
            environment['DIRMOTOR']=2
            environment['VELRIGHTMOTOR'] = vel_left_const
            environment['VELLEFTMOTOR'] = vel_right_const
        else:
            environment['DIRMOTOR']=1
            environment['VELLEFTMOTOR'] = vel_left_const
            environment['VELRIGHTMOTOR'] = vel_right_const



# Save data
environment['robotX']=new_x
```

```
environment['robotY']=new_y
environment['head']=new_theta
environment['obstacleX']=xobs
environment['obstacleY']=yobs
environment['DIRMOTOR']=dir_motor
environment['ULTRASONICHEAD']=90
aux['d_left_wheel']=d_left_wheel
aux['d_right_wheel']=d_right_wheel
aux['rand']=rand

# return data
print(json.dumps(environment))
```