UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
UNIVERSIDAD COMPLUTENSE DE MADRID





MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

TRABAJO FIN DE MÁSTER

"Integración de un robot autónomo basado en Raspberry en el sistema de laboratorios remotos"

Autor: ***Jorge Durán Murillas***

Directores: ***Luis de la Torre Cubillo***

Dictino Chaos García

Daniel Galán Vicente

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
UNIVERSIDAD COMPLUTENSE DE MADRID





MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

TRABAJO FIN DE MÁSTER

"Integración de un robot autónomo basado en Raspberry en el sistema de laboratorios remotos"

Proyecto Tipo A: Proyecto específico propuesto por un profesor.

Autor:	***Jorge Durán Murillas***
Directores:	***Luis de la Torre Cubillo***
	Dictino Chaos García
	Daniel Galán Vicente

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:



Jorge Durán Murillas

Resumen

Un robot autónomo es aquel robot capaz de realizar tareas por sí mismo. Si se trata de un robot móvil esto último significa que el robot tiene la capacidad de determinar su localización y de navegar de un punto a otro del entorno en el cual se encuentra.

La función navegación de un robot se compone de varias sub-tareas: localización, planificación de trayectorias y construcción de mapas e interpretación. Desde el punto de vista del robot, se puede decir que la localización es la capacidad del mismo de determinar su posición y orientación respecto a un sistema de coordenadas dado, la construcción de mapas sirve al robot como una representación del entorno en el que se mueve y la planificación de trayectorias implica la capacidad del robot de calcular cuál es la ruta a seguir para ir de una localización a otra de su entorno.

El desarrollo de este proyecto ha consistido en utilizar un robot autónomo basado en la placa Raspberry Pi (GoPiGo: http://www.dexterindustries.com/gopigo/), para convertirlo en un robot tele-operado que pueda programarse remotamente para realizar acciones autónomas desde un ordenador no conectado físicamente al robot. Ya se ha trabajado con este robot en proyectos anteriores y el objetivo principal de este proyecto es integrarlo con el resto de tecnologías de los laboratorios remotos de la UNED. La finalidad última es convertirlo en una plataforma de experimentación para los estudiantes del Máster.

Como parte del trabajo realizado se ha implementado una interfaz de monitorización, control manual y programación básica del robot GoPiGo. Esta interfaz debe permitir acciones de control simples como por ejemplo mover a la izquierda, mover a la derecha, avanzar o retroceder. Además permite medir variables de estado del robot (traslación efectuada, posición, distancia de obstáculos medidas con los sensores, etc.) así como mostrar el flujo de video procedente de una cámara que permitiese ver el robot en el laboratorio. Dicha interfaz también ofrece la posibilidad de escribir programas de control que doten al robot de una inteligencia para realizar tareas sencillas de forma autónoma. Por otro lado se ha implementado además el software de comunicaciones capaz de enviar órdenes desde la interfaz al robot y se ha realizado la integración de la interfaz de control con el sistema de laboratorios remotos que existe en la UNED.

Abstract

An autonomous robot is a robot capable of performing tasks by itself. If it is a mobile robot, the latter means that the robot has the ability to determine its location and navigate from one point to another in the environment in which it is located.

The navigation task of a robot is composed of several sub-tasks: location, trajectory planning and map construction and interpretation. From the point of view of the robot, the location is the ability of the robot to determine its own position and orientation with respect to a given coordinate system. The construction of maps is used by the robot as a representation of the environment in which it moves. Finally the planning of trajectories implies the capacity of the robot to calculate the route to follow in order to go from one location to another of its environment.

In this project we have worked with an autonomous robot based on the Raspberry Pi board (GoPiGo: http://www.dexterindustries.com/gopigo/), to turn it into a tele-operated robot that can be remotely programmed to perform autonomous actions from a computer not physically connected to the robot. This robot has already been worked on in previous projects and the main objective of this project is to integrate it with the other technologies of the remote laboratories of the UNED. The ultimate goal is to turn it into an experimentation platform for the students of the Master.

In this project, the monitoring, manual control and basic programming interface of the GoPiGo robot have been implemented. This interface allows simple control actions such as the move to the left, move to the right, move forward or backward. It also allows to measure variables of the robot's state (translation made, position, a distance of obstacles measured with the sensors, etc.) as well as to show the video stream from a camera that allows seeing the robot in the laboratory. This interface also allows writing control programs that give the robot an intelligence to perform simple tasks autonomously. On the other hand, the communications software that allows sending orders from the interface to the robot was implemented and the integration of the control interface with the system of remote laboratories that exists in the UNED has been made.

Palabras claves

Raspberry Pi, servidor RIP, GoPiGo, interfaz web, EjsS, Python, Bloxter, Blockly

Contenido

Capítulo 1: Introducción. 1
1.1. Motivación y objetivos del proyecto. 1
1.2. Organización del documento. 2
Capítulo 2: Estado del Arte. 3
2.1. Antecedentes de Investigación. 3
2.2. Necesidades. 7
Capítulo 3: Plataforma Robótica. 8
3.1. Plataforma Robótica. Descripción y Montaje. 10
3.1.1. Descripción de la placa Raspberry Pi. 11
3.1.2. Descripción de la placa Dexter Industries GoPiGo-2. 13
3.2. API Python de programación de la plataforma robótica. 16
3.2.1.1. Funciones de control de movimiento de los motores. 17
3.2.1.2. Funciones de control de velocidad de los motores. 18
3.2.1.3. Funciones de control de los encoders. 19
3.2.1.4. Funciones de control del estado de la plataforma robótica. 20
3.2.1.5. Funciones de control del sensor de ultrasonidos. 21
3.2.1.6. Funciones de control del sensor LED. 21
3.2.1.7. Funciones de control del motor SERVO. 21
Capítulo 4: Arquitectura Desarrollada. 22
4.1. Servidor RIP implementado en lenguaje de programación Python del robot GoPiGo 2. 23
4.1.1. Protocolo RIP. Remote Interoperability Protocol. 23
4.1.2. Servidor RIP. Implementación. 26
4.1.2.1. Arquitectura del Servidor RIP. 27
4.1.2.2. Servidor RIP. Diagrama de flujo del Servidor RIP instalado en el Robot. 33
4.1.2.3. Servidor RIP. Clase GoPiGoContext. 36
4.2. Aplicación Web / EjsS de control del robot GoPiGo 2. 40
4.2.1. Framework EjsS – Easy Java/JavaScript Simulations. 40
4.2.2. Aplicación Web EjsS. Implementación. 45
4.2.2.1. Aplicación Web EjsS. Configuración del componente RIP_Element. 48
4.2.2.2. Aplicación Web EjsS. Panel Superior. 51
4.2.2.3. Aplicación Web EjsS. Panel Central. 57
4.2.2.4. Aplicación Web EjsS. Panel Inferior. 70
4.2.2.5. Aplicación Web EjsS. Configuración del Ciclo Automático de Actualización de la Aplicación. 71

4.3. Programación del robot GoPiGo-2 mediante el editor de código EjsS. 73
4.3.1. Script 1. Control Básico de Evitación de Obstáculos. 73
4.3.2. Script 2. Navegación con Evitación de Obstáculos. 75
4.3.3. Programa 1. Evitación de Obstáculos Simple. 77
4.3.4. Programa 2. Mapeo de Entorno con Evitación de Obstáculos. 78
4.4. Integración de la Aplicación Web / EjsS de control del robot GoPiGo 2 en el framework Blockly. 80
4.4.1. Framework Blockly. 80
4.4.2. Aplicación Web EjsS. Integración con el framework Blockly. 84
4.4.2.1. Definición de las variables públicas en la aplicación EjsS. 85
4.4.2.2. Conexión de bloques del framework Blockly con las variables públicas de EjsS. 88
4.4.2.3. Actualización de la ejecución de acciones del robot GoPiGo-2 a través de la aplicación EjsS. 93
Capítulo 5: Pruebas Realizadas. 98
5.1. Pruebas de Control Manual. 98
5.2. Pruebas de Control Automático. 99
5.3. Pruebas de Control Mediante Blockly. 101
Capítulo 6: Conclusiones y Trabajos Futuros. 104
6.1. Limitaciones en el robot. 104
6.2. Líneas de trabajo futuras. 105
Bibliografía. 107
Listado de siglas, abreviaturas y acrónimos. 109
ANEXO A. 110
Montaje del cuerpo de la plataforma robótica. 110
Montaje del cuerpo de la plataforma robótica. Acoplar el Servo. 117
Montaje del cuerpo de la plataforma robótica. Acoplar el Sensor de Ultrasonidos. 118
Montaje del cuerpo de la plataforma robótica. Instalación de la cámara modelo *Raspberry Pi Camera Board*. 119
Instalación de la Imagen de Raspbian y actualización de paquetes y librerías. 120
Instalación de la Imagen de Raspbian. 120
Conexión con el Robot y actualización de librerías. 122
ANEXO B. 136
Clase HttpServer (HttpServer.py). 136
Clase HttpServerGoPiGo (HttpServerGoPiGo.py). 140
Clase HttpServerGoPiGo (HttpServerGoPiGo.py). 141
Clase RIPGeneric (RIPGeneric.py). 143

Clase GoPiGoContext (GoPiGoContext.py). 149
Clase GoPiGoService (GoPiGoService.py). 150
Clase testHttpServerGoPiGo-ripinfoserver (testHttpServerGoPiGo-ripinfoserver.py). 162
Clase testHttpServerGoPiGo-connectmethods (testHttpServerGoPiGo-connectmethods.py). 163
Clase testHttpServiceGoPiGo-getmethods (testHttpServerGoPiGo-getmethods.py). 164
Clase GoPiGoService (GoPiGoService.py). 165
ANEXO C. 167
ANEXO D. 173
Código Fuente. Script 1. Control Básico de Evitación de Obstáculos. 173
Código Fuente. Script 2. Navegación con Evitación de Obstáculos. 176
Código Fuente. Programa 1. Evitación de Obstáculos Simple. 181
Código Fuente. Programa 2. Mapeo de Entorno con Evitación de Obstáculos. 182

Figuras

Figura 1. Coordenadas y componentes de velocidad del modelo de robot diferencial. Modelo 1. 4
Figura 2. Esquema de movimientos en un robot con tracción y direccionamiento diferencial. 4
Figura 3. Coordenadas y componentes de velocidad del modelo de robot diferencial. Modelo 2. 5
Figura 4. Arquitectura de comunicaciones entre los diferentes componentes del trabajo. 9
Figura 5. Robot GoPiGo-2 de Dexter Industries. 10
Figura 6. Raspberry Pi 3 Model B. 11
Figura 7. Placa Dexter Industries GoPiGo-2. Vista principal. 13
Figura 8. Placa Dexter Industries GoPiGo-2. Vista trasera. 13
Figura 9. Placa Dexter Industries GoPiGo-2. Componentes en la Vista principal. 14
Figura 10. Placa Dexter Industries GoPiGo-2. Componentes en la Vista trasera. 15
Figura 11. Diagrama de secuencia de una sesión del protocolo RIP basado en los servicios del protocolo JSON-RPC. 24
Figura 12. Elementos básicos que componen el patrón MVC. 28
Figura 13. Interacción básica entre un cliente y un servidor a través de una interfaz REST. 29
Figura 14. Servidor RIP. Diagrama de Flujo del Ciclo de trabajo completo del robot GoPiGo 2. 33
Figura 15. Servidor RIP. Diagrama de Flujo del Ciclo de trabajo global del robot GoPiGo 2. 34
Figura 16. Servidor RIP. Diagrama de Flujo Ciclo de trabajo en modo automático del robot GoPiGo 2. 35
Figura 17. Ventana principal del espacio de trabajo del framework EjsS. 40
Figura 18. Panel de trabajo “Description” del interfaz del framework EjsS. 41
Figura 19. Panel de trabajo “Modelo” del interfaz del framework EjsS. 42
Figura 20. Panel de trabajo “Vista” del interfaz del framework EjsS. 43
Figura 21. Ejemplo del componente TabbedPane utilizado para el desarrollo de la aplicación EjsS de control del robot GoPiGo 2. 45
Figura 22. Distribución de la aplicación EjsS de control del robot GoPiGo-2 en componentes. 45
Figura 23. Vista general de la aplicación EjsS de control del robot GoPiGo-2. 46
Figura 24. Vista general del componente TabbedPane incluido en el “Panel Central” con sus diferentes pestañas. 47
Figura 25. Pestaña “Elementos” del interfaz de desarrollo del framework EjsS donde añadir el componente RIP_element. 48
Figura 26. Configuración del componente “RIP_Element” renombrado como “ripclient” en la aplicación EjsS. (I). 49
Figura 27. Configuración del componente “RIP_Element” renombrado como “ripclient” en la aplicación EjsS. (II). 49
Figura 28. Métodos expuestos a través del end-point http://<dir_IP>:2055/RIP/POST. 50
Figura 29. Vista general del componente “Panel Superior”. 51
Figura 30. Vista de diseño de elementos del componente “Panel Superior”. 51
Figura 31. Vista del Sub-Panel de Estado (pnRobot). 52
Figura 32. Vista del sub-panel de Control de opciones de mapa “pnopcionesmapa”. 53
Figura 33. Vista del sub-panel de Control de Movimiento “pnopcionesmoves”. 54
Figura 34. Descomposición del panel de Control de Movimiento (pnopcionesmoves) en elementos gráficos simples 55
Figura 35. Vista de diseño de elementos del componente “Panel Central”. 57
Figura 36. Vista de los componentes gráficos que forman parte del elemento “SubPanelCentral1”. 58
Figura 37. Vista de los componentes gráficos que forman parte del elemento “pnservo”. 59
Figura 38. Vista de los componentes gráficos que forman parte del elemento “pnservo”. 61
Figura 39. Vista de los componentes gráficos que forman parte del elemento “pnultrasonic”. 62
Figura 40. Vista de los componentes gráficos que forman parte del elemento “SubPanelCentral2”. 63
Figura 41. Vista de los componentes gráficos que forman parte del elemento “SubPanelCentral3”. 64
Figura 42. Descomposición del panel de Visualización de Mapa de Ocupación (pnmapa) en elementos gráficos simples. 64

Figura 43. Configuración de los componentes “posicionRobot” y “obstaculos”. 65
Figura 44. Vista de los componentes gráficos que forman parte del elemento “SubPanelCentral4”. 66
Figura 45. Descomposición del panel de Visualización de WebCam (pncamara) en elementos gráficos simples. 66
Figura 46. Configuración del componente “htmlWebCamRenderer”. 67
Figura 47. Recepción del flujo de video en el componente “htmlWebCamRenderer” de la cámara del robot GoPiGo-2. 67
Figura 48. Vista de los componentes gráficos que forman parte del elemento “SubPanelCentral5”. 68
Figura 49. Descomposición del componente “SubPanelCentral5” en elementos gráficos simples. 69
Figura 50. Vista general del componente “Panel Inferior”. 70
Figura 51. Descomposición en componentes del elemento “Panel Inferior”. 70
Figura 52. Listado de los métodos que actualizan de forma automática el estado de la aplicación EjsS. . 71
Figura 53. Script 1. Segmento de código de obtención de los parámetros del robot desde el contexto. 73
Figura 54. Script 1. Segmento de código para el cálculo de la posición del robot. 74
Figura 55. Script 1. Segmento de código para el cálculo de la posición del obstáculo. 75
Figura 56. Script 1. Segmento de código para el cálculo de la parada del robot. 75
Figura 57. Script 2. Segmento de código para el cálculo de las velocidades del robot. 76
Figura 58. Programa 1. Segmento de código para el control del robot en avance y detección de obstáculo con parada. 77
Figura 59. Programa 2. Segmento de código para el control del robot en avance mediante cuentas del encoder. 78
Figura 60. Programa 2. Segmento de código para la obtención de medidas con el sensor de ultrasonidos y movimiento del servo. 79
Figura 61. Programa 2. Segmento de código para la detección de obstáculo con parada del robot en caso de que la distancia medida sea inferior al umbral. 79
Figura 62. Ejemplo de programa definido mediante bloques del Framework Blockly y su equivalencia en lenguaje JavaScript. 84
Figura 63. Definición de variables públicas en la aplicación EjsS de control del Robot GoPiGo-2. 85
Figura 64. Bloque “establecer” del framework Blockly en el entorno de laboratorios. Ejemplo de uso. ... 88
Figura 65. Equivalencia entre la combinación de bloques “establecer” del framework Blockly de los laboratorios remotos y el bloque “ideal” de la hipotética librería de control del robot GoPiGo-2. ... 88
Figura 66. Configuración de la evolución del estado del robot GoPiGo-2 a través del método de actualización de estado para el framework Blockly. 93
Figura 67. Captura del video de demostración 1. 98
Figura 68. Test de control automático del robot GoPiGo-2 mediante el uso de un script Python. 100
Figura 69. Componentes incluidos en la caja del robot GoPiGo-2 de Dexter Industries. 111
Figura 70. Montaje del cuerpo de la plataforma robótica. Paso 1. 111
Figura 71. Montaje del cuerpo de la plataforma robótica. Fijación de la placa en la base de plástico. Paso 1. 112
Figura 72. Montaje del cuerpo de la plataforma robótica. Fijación de la tornillería de los motores. Paso 2. 112
Figura 73. Montaje del cuerpo de la plataforma robótica. Vista de los motores conectados al chasis. Paso 2. 113
Figura 74. Montaje del cuerpo de la plataforma robótica. Conexión de las ruedas internas. Paso 3. 114
Figura 75. Montaje del cuerpo de la plataforma robótica. Montaje de la rueda giratoria. Paso 4. 115
Figura 76. Montaje del cuerpo de la plataforma robótica. Montaje del adaptador de baterías. Paso 5. 115
Figura 77. Montaje del cuerpo de la plataforma robótica. Montaje de las ruedas. Paso 6. 116
Figura 78. Montaje del cuerpo de la plataforma robótica. Conexión de la placa Raspberry Pi. Paso 7. .. 116
Figura 79. Elementos necesarios para el montaje del servo en la plataforma robótica. 117
Figura 80. Montaje del servo acoplado a la plataforma robótica. Paso 8. 117
Figura 81. Montaje del servo acoplado a la plataforma robótica. Conexión eléctrica a la palca GoPiGo-2. Paso 9. 118

Figura 82. Montaje del sensor de ultrasonidos acoplado a la plataforma robótica. Conexión eléctrica a la palca GoPiGo-2. Paso 10. 118
Figura 83. Cámara del robot GoPiGo-2. Modelo “Raspberry Pi Camera Board”. 119
Figura 84. Cámara acoplada en la base de plástico acrílico del robot GoPiGo-2. 119
Figura 85. Ventana principal de la aplicación Etcher para el grabado de la imagen de Raspbian. 120
Figura 86. Ventana principal de la aplicación Etcher preparada para grabar una imagen en un una unidad de destino seleccionada. 121
Figura 87. Ventana principal de la aplicación Etcher. Proceso de verificación / validación de la imagen grabada. 121
Figura 88. Conexión del robot (placa Raspberry Pi 3) con el PC de desarrollo mediante cable Ethernet. 122
Figura 89. Vista principal de la url dex.local una vez que se ha establecido la comunicación con el robot GoPiGo 2. 123
Figura 90. Escritorio Dexter Industries del robot GoPiGo 2. 123
Figura 91. Vista de la aplicación de control del robot GoPiGo 2 – “GoPiGo Control Panel”. 124
Figura 92. Ventana principal del panel de control “WiFi Setup” para la configuración WiFi del robot. 124
Figura 93. Escaneo y configuración de la red WiFi a la que se conectará el robot en cada arranque. 125
Figura 94. Establecimiento de la conexión del robot con la red WiFi (Status: Completed). 125
Figura 95. Prueba de conectividad del robot con una URL externa usando el comando “ping”. 126
Figura 96. Actualización / instalación de la última versión de Python en el robot GoPiGo 2. 127
Figura 97. Instalación de la librería jsonrpc en el robot GoPiGo 2. 127
Figura 98. Instalación de la librería ujson en el robot GoPiGo 2. 127
Figura 99. Instalación de la librería cherrypy en el robot GoPiGo 2. 128
Figura 100. Crear el directorio virtual “venv” dentro del directorio del servidor RIP. 128
Figura 101. Instalación de la librería del robot “GoPiGo” en el directorio virtual (I). 129
Figura 102. Instalación de la librería del robot “GoPiGo” en el directorio virtual (II). 129
Figura 103. Resultado de la instalación de la librería “GoPiGo” en el directorio virtual “venv”. 129
Figura 104. Instalación de la librería RPi.GPIO en el robot GoPiGo-2 130
Figura 105. Instalación de la librería smbus en el robot GoPiGo-2. 130
Figura 106. Instalación de la librería ujson en el directorio virtual “venv” de la implementación base del servidor RIP. 130
Figura 107. Instalación de la librería cherrypy en el directorio virtual “venv” de la implementación base del servidor RIP. 131
Figura 108. Instalación de la librería jsonrpc en el directorio virtual “venv” de la implementación base del servidor RIP. 131
Figura 109. Instalación de la librería ws4py en el directorio virtual “venv” de la implementación base del servidor RIP. 131
Figura 110. Comprobación de la correcta instalación de la librería python3-cherrypy3 y prueba de ejecución del servidor RIP de prueba. 132
Figura 111. Apagado ordenado del robot GoPiGo 2. 133
Figura 112. Desconexión del cable de red Ethernet de la tarjeta Raspberry Pi 3 del robot GoPiGo 2. 134
Figura 113. Comprobación de la conectividad entre el PC de desarrollo y el robot GoPiGo 2 mediante conexión WiFi. 134

Tablas

Tabla 1. API de programación de la plataforma robótica. Funciones para el control del movimiento de los motores. 17
Tabla 2. API de programación de la plataforma robótica. Funciones para el control de velocidad de los motores. 18
Tabla 3. API de programación de la plataforma robótica. Funciones para el control de los encoders. 19
Tabla 4. API de programación de la plataforma robótica. Funciones para el control del estado de la plataforma robótica. 20
Tabla 5. API de programación de la plataforma robótica. Funciones para el control del sensor de ultrasonidos. 21
Tabla 6. API de programación de la plataforma robótica. Funciones para el control del sensor LED. 21
Tabla 7. API de programación de la plataforma robótica. Funciones para el control del servo-motor. 21
Tabla 8. Librerías necesarias para poder utilizar el servidor RIP. 26
Tabla 9. Clases que forman parte de la implementación del Servidor RIP. 27
Tabla 10. Relación de endpoints ofrecidos por la implementación del servidor RIP / GoPiGo-2 realizada. 29
Tabla 11. Variables del servidor RIP actualizables a través del método SET (I). 30
Tabla 12. Variables del servidor RIP actualizables a través del método SET (II). 31
Tabla 13. Variables del servidor RIP consultadas a través del método GET. 32
Tabla 14. Propiedades de la clase GoPiGoContext. 36
Tabla 15. Relación propiedades de la clase GoPiGoContext - API del robot 39
Tabla 16. Párametros de configuración del componente “RIP_Element”. 49
Tabla 17. Configuración de los componentes “btnconectar” y “btndesconectar”. 52
Tabla 18. Configuración de los componentes “btnresetmap” y “btnupdatemap”. 53
Tabla 19. Configuración de los componentes “rbtmanual”, “rbtautomatico” y “chkactwebcam”. 54
Tabla 20. Configuración de los componentes “btnavanzar” y “btnretroceder”. 55
Tabla 21. Configuración de los componentes “btnizquierda”, “btnderecha”, “btnstop”, “btngiroizquierda”, “btngirostop” y “btngiroderecha”. 56
Tabla 22. Configuración de los componentes “btnsetrightspeed2” y “btnsetrightspeed23”. 58
Tabla 23. Configuración de los componentes “btnspeedleft”, “btnspeedright”, “btnincrementar” y “btndecrementar”. 59
Tabla 24. Configuración de los componentes “btnenableserbo”, “btndisableservo”, “btnreset”, “btnscan” y “btnservoizquierda”. 60
Tabla 25. Configuración de los componentes “btnservoderecha” , “btnonoffled1” y “btnonoffled2”. 61
Tabla 26. Configuración de los componentes “btnsetencoder” y “btnresetencoder”. 62
Tabla 27. Configuración del componente “btnultrasonic”. 62
Tabla 28. Configuración de los componentes “chkupdate” y “btnactualizar”. 63
Tabla 29. Configuración de los componentes “btnuploadscript”, “btnstopscript”, “btnuploadprogram” y “btnstopprogram”. 69
Tabla 30. Configuración del componente “btncleareditor”. 70
Tabla 31. Framework Blockly. Bloques Lógicos. 80
Tabla 32. Framework Blockly. Bloques de Bucle. 81
Tabla 33. Framework Blockly. Bloques de Operaciones Matemáticas. 81
Tabla 34. Framework Blockly. Bloques de Operaciones sobre cadenas de caracteres. 82
Tabla 35. Framework Blockly. Bloques de Operaciones sobre listas de elementos. 82
Tabla 36. Framework Blockly. Bloques de Ejecución de Funciones. 83
Tabla 37. Framework Blockly. Bloques propios del entorno de UniLabs. 83
Tabla 38. Definición de las variables públicas de la aplicación EjsS accesibles desde el framework Blockly. 86
Tabla 39. Valores que puede tomar la variable “bStateMachineCommand” 87

Tabla 40. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (I).....89
Tabla 41. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (II). 90
Tabla 42. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (III).....91
Tabla 43. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (IV).....92
Tabla 44. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (V).....97
Tabla 45. Programas de control definidos en Blockly y probados con el robot GoPiGo-2 (I). 101
Tabla 46. Programas de control definidos en Blockly y probados con el robot GoPiGo-2 (II).....102
Tabla 47. Programas de control definidos en Blockly y probados con el robot GoPiGo-2 (III). 103
Tabla 48. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – I.....167
Tabla 49. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – II.....168
Tabla 50. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – III. 169
Tabla 51. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – IV. 170

Capítulo 1: Introducción.

1.1. Motivación y objetivos del proyecto.

El objetivo de este Trabajo Fin de Máster es integrar un robot basado en la Raspberry Pi para convertirlo en un robot tele-operado que pueda programarse remotamente desde un ordenador no conectado físicamente al robot. Ya se ha trabajado con este robot en proyectos anteriores y el objetivo principal de este proyecto es integrarlo con el sistema de laboratorios remotos que existe en la UNED. La finalidad última es convertirlo en una plataforma de experimentación para los estudiantes del Máster. El robot utilizado para este proyecto es el robot GoPiGo 2 (http://www.dexterindustries.com/gopigo/), 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 explicará en capítulos posteriores.

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

En trabajos anteriores a este 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] se puede encontrar una solución, muy parecida a la desarrollada en este Trabajo Fin de Master, 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.

Sin embargo, en este trabajo se han 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 a través del entorno remoto de los laboratorios UNILabs de la UNED. 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.

Para la integración del robot en el sistema de laboratorios remotos UNILabs de la UNED se ha hecho uso de dos soluciones: por un lado la programación de scipts de control enviados al robot en lenguaje Python mediante el editor ACE integrado en la aplicación EjsS, y por otro el framework *Blockly* que permite programar de forma gráfica (mediante la concatenación de bloques funcionales) las acciones del robot para posteriormente traducir dicha definición a código Python entendible por el robot.

1.2. Organización del documento.

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

- En el capítulo 2 se presenta el estado del arte asociado a la teoría detrás de los robots con tracción diferencial.
- En el capítulo 3 se presentan los pasos necesarios para la puesta en funcionamiento de la plataforma robótica comercial utilizada en este trabajo.
- En el capítulo 4 se presenta la arquitectura desarrollada en el presente trabajo, así como sus componentes implementados.
- En el capítulo 5 se describen las pruebas experimentales realizadas para la demostración del funcionamiento del trabajo realizado.
- Finalmente, en el capítulo 6 se presentan las conclusiones y las posibles líneas futuras de desarrollo.

Además, se ha añadido un apartado de "Anexos" donde se incluye información complementaria: código fuente de los elementos software implementados, así como consideraciones adicionales de despliegue de dichos componentes.

Capítulo 2: Estado del Arte.

2.1. Antecedentes de Investigación.

La locomoción es el proceso por el cual un robot autónomo puede desplazarse. Existe una gran variedad de modos de moverse sobre una superficie sólida, entre los robots las más comunes son las ruedas, las cadenas y las patas.

Los vehículos de ruedas son, con mucho, los más populares por varias razones prácticas. Los robots con ruedas son más sencillos y más fáciles de construir, la carga que pueden transportar es mayor relativamente. Tanto los robots basados en cadenas como en patas se pueden considerar más complicados y pesados generalmente que los robots de ruedas para una misma carga útil.

La principal desventaja de las ruedas es su empleo en terreno irregular, en el que se comportan bastante mal. Normalmente un vehículo de ruedas podrá sobrepasar un obstáculo que tenga una altura no superior al radio de sus ruedas, entonces una solución es utilizar ruedas mayores que los posibles obstáculos a superar; sin embargo, esta solución, a veces, puede no ser práctica.

Para los robots que vayan a funcionar en un entorno natural las cadenas (tipo oruga) son una opción muy buena porque las cadenas permiten al robot superar obstáculos relativamente mayores y son menos susceptibles que las ruedas de sufrir daños por el entorno, como piedras o arena. El principal inconveniente de las cadenas es su ineficacia, puesto que se produce deslizamiento sobre el terreno al avanzar y al girar. Si la navegación se basa en el conocimiento del punto en que se encuentra el robot y el cálculo de posiciones futuras sin error, entonces las cadenas acumulan tal cantidad de error que hace inviable la navegación por este sistema.

Potencialmente los robots con patas pueden superar con mayor facilidad que los otros los problemas de los terrenos irregulares. A pesar de que hay un gran interés en diseñar este tipo de robots, su construcción plantea numerosos retos. Estos retos se originan principalmente en el gran número de grados de libertad que requieren los sistemas con patas. Cada pata necesita como mínimo un par de motores lo que produce un mayor coste, así como una mayor complejidad y menor fiabilidad. Es más, los algoritmos de control se vuelven mucho más complicados por el gran número de movimientos a coordinar, los sistemas de patas son un área de investigación muy activo.

La precisión de las medidas de odometría para posicionar el robot es en gran manera una función directa del diseño cinemático del vehículo. A causa de esta cercana relación entre el diseño cinemático y la precisión de la posición, uno debe considerar el diseño cinemático atentamente antes de intentar mejorar la precisión de posicionamiento. Existen varios diseños de ruedas para elegir cuando se quiere construir una plataforma móvil sobre ruedas, diferencial, sincronizada, triciclo y de coche. Para el desarrollo de éste trabajo, se utilizará el diseño con tracción diferencial.

El robot con tracción diferencial (cuyo modelo se puede apreciar en la Figura 1) cuenta con un tipo de direccionamiento que viene dado por la diferencia de velocidades de las ruedas laterales (ambas de radio *"R"*). Dos ruedas montadas en un único eje son independientemente propulsadas y controladas, proporcionando ambas, tracción y direccionamiento. La combinación del movimiento de las dos ruedas provoca el movimiento alrededor del CIR (centro inercial de rotación) en el cual también se toman en consideración los parámetros *"L"* o longitud del robot, *"ω"* o velocidad angular del robot y *"θ"* o ángulo aproximación del robot al punto de destino.

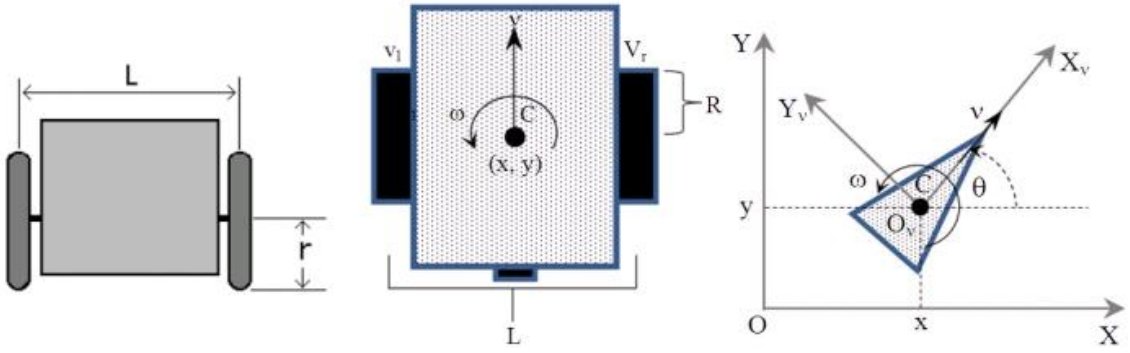


Figura 1. Coordenadas y componentes de velocidad del modelo de robot diferencial. Modelo 1.

Este sistema es muy útil si se considera la habilidad del movimiento del móvil, presentando la posibilidad de cambiar su orientación sin movimientos de traslación. Las variables de control de este sistema son las velocidades angulares de las ruedas izquierda y derecha las cuales permiten imprimir sobre el robot los movimientos que se pueden apreciar en la Figura 2.

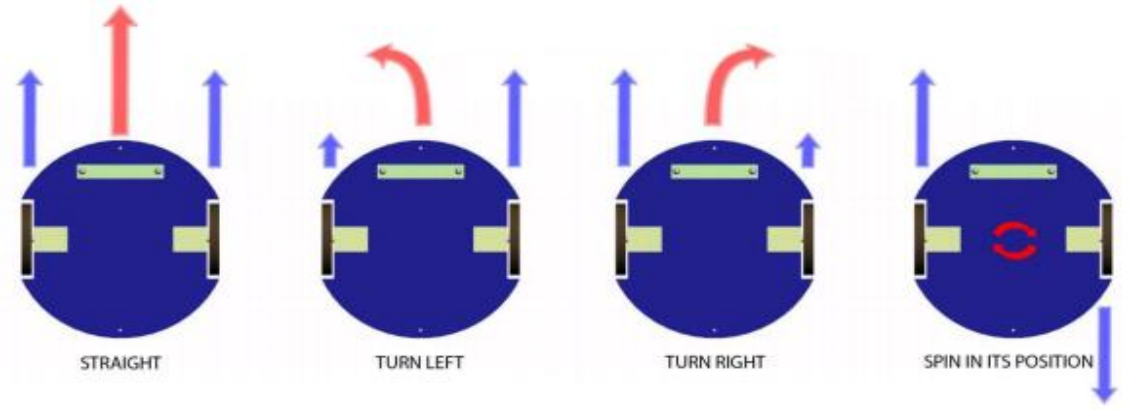


Figura 2. Esquema de movimientos en un robot con tracción y direccionamiento diferencial.

Tanto desde el punto de vista de la programación como de la construcción, el diseño diferencial es uno de los sistemas de locomoción más sencillos. El robot puede ir recto, girar sobre sí mismo y trazar curvas. Un problema importante es cómo resolver el equilibrio del robot, ya que hay que buscarle un apoyo adicional a las dos ruedas ya existentes, esto se consigue mediante una o dos ruedas de apoyo añadidas en un diseño triangular o romboidal. El diseño triangular puede no ser suficiente dependiendo de la distribución de pesos del robot, y el romboidal puede provocar inadaptación al terreno si éste es irregular, lo que puede exigir alguna clase de suspensión.

Otra consideración a hacer en este diseño es cómo conseguir que el robot se mueva recto, para que el robot se mueva en línea recta sus ruedas tienen que girar a la misma velocidad. Cuando los motores encuentran diferentes resistencias (una rueda sobre moqueta y la otra sobre terraza) las velocidades de los motores varían y el robot girará incluso aun cuando se le haya ajustado inicialmente para que vaya recto. Esto quiere decir que la velocidad debe ser controlada dinámicamente, es decir, debe existir un medio de monitorizar y cambiar la velocidad del motor mientras el robot avanza. De esta manera la simplicidad del diseño queda minimizada por la complejidad del sistema de control de la velocidad.

Un aspecto importante en este tipo de diseño consiste en el uso de las ecuaciones que rigen la cinemática del robot móvil para determinar la posición que este en el entorno en que se mueve, el cual tiene como referencia un sistema coordenado (x,y,z). La posición se calcula a partir de los cambios que se den en los sistemas de locomoción del robot (motores).

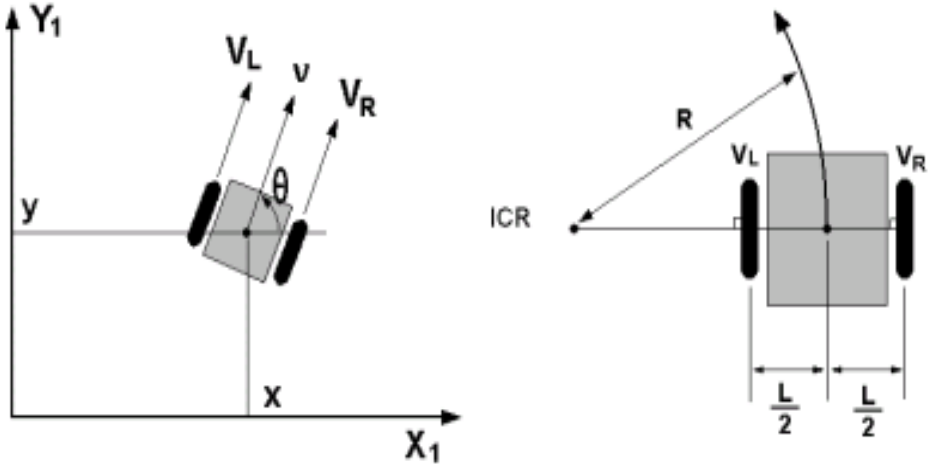


Figura 3. Coordenadas y componentes de velocidad del modelo de robot diferencial. Modelo 2.

Para el análisis cinemático de un robot móvil de tracción diferencial se tiene en cuenta las dimensiones: L que es la longitud entre las dos ruedas del vehículo y r es el radio de las ruedas como se ve en la figura de la derecha. Para el modelo se supone que su desplazamiento es en dos dimensiones, en la figura 3 se muestra la localización del robot en el punto (x,y) donde "v" es la velocidad lineal del móvil y, "V_L" y "V_R" la velocidad tangencial de cada una de las ruedas.

Se puede obtener el modelo matemático para determinar la posición de un móvil de tracción diferencial a partir de las velocidades "V_L" y "V_R", las cuales se obtienen con la siguiente ecuación.

$$V_L = \omega_l \cdot r;\ V_R = \omega_r \cdot r$$

En donde "w_l" y "w_r" son las velocidades angulares de cada rueda. Las velocidades lineales y angulares del robot se obtienen con las siguientes ecuaciones respectivamente.

$$v = \frac{V_R + V_L}{2} = \frac{(\omega_l + \omega_r) \cdot r}{2}$$

$$w = \frac{V_R - V_L}{L} = \frac{(\omega_l - \omega_r) \cdot r}{L}$$

Sabiendo que el robot se mueve en una superficie plana, sin deslizamiento y que los ejes de las ruedas son perpendiculares a la superficie plana, se puede demostrar que si $p = [x\ y\ \theta]$ es el vector de coordenadas del punto guía del robot y la orientación del mismo y si $q' = [v\ w]$ es el vector de la velocidad lineal y angular del móvil, se puede escribir la siguiente ecuación en forma matricial.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -\sin\theta & 0 \\ \cos\theta & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} v \\ w \end{bmatrix}$$

Ahora sustituyendo las ecuaciones anteriores en la nueva ecuación con forma matricial se tiene.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -\mathrm{r} \cdot \sin\theta/2 & -\mathrm{r} \cdot \sin\theta/2 \\ r \cdot \cos\theta/2 & r \cdot \cos\theta/2 \\ r/L & r/L \end{bmatrix} \times \begin{bmatrix} w_l \\ w_r \end{bmatrix}$$

La velocidad lineal "v" es la media de las velocidades de ambas ruedas $v = (\frac{v_l + v_r}{2})$ y la velocidad angular w depende del radio de ambas ruedas y la distancia entre ellas L. De este modo la suma de las velocidades controla la velocidad total de avance y su diferencia controla la velocidad de giro. De acuerdo a lo expuesto en la figura 3 se puede deducir que la posición del robot diferencial viene dada por las siguientes expresiones:

$$\dot{x} = \frac{(V_R + V_L)}{2} \cdot \cos\theta$$

$$\dot{y} = \frac{(V_R + V_L)}{2} \cdot \sin\theta$$

$$\theta = \frac{(V_R - V_L)}{L}$$

Aunque el presente trabajo no contempla el diseño desde cero de un robot con tracción diferencial si es útil tener como referencia las consideraciones y formulaciones matemáticas anteriormente expuestas de cara a la construcción del robot comercial utilizado en este trabajo y su integración y pruebas en el espacio de trabajo de laboratorios remotos proporcionado por la UNED.

2.2. Necesidades.

En la actualidad el Departamento de Informática y Automática de la UNED, está centrado entre otras líneas de investigación en los robots móviles. En dicho grupo se utiliza entre otros una plataforma de laboratorios remotos UNILabs que permite a los alumnos matriculados en los cursos de grado y máster la realización de prácticas en modo remoto con robots autónomos. Ante la necesidad de mejorar e incrementar la oferta de prácticas con nuevos modelos de robots se consideró oportuno la integración de un nuevo tipo de robot comercial con esquema de tracción diferencial dentro de la oferta de laboratorios remotos ofrecida por la plataforma UNILabs.

Capítulo 3: Plataforma Robótica.

Los principales elementos que componen el trabajo se pueden apreciar en la Figura 4:

- **Plataforma Robótica:** El robot utilizado es el modelo GoPiGo 2 de la compañía Dexter Industries. Se verán sus características más adelante.
- **Placa Raspberry Pi:** Entre otros componentes, el robot incluye una placa Raspberry Pi, donde se ejecutará el servidor RIP implementado.
- **Placa Dexter Industries GoPiGo 2:** Esta placa se conecta directamente a la Raspberry Pi y sirve de interfaz entre esta y los sensores y actuadores del robot. Gracias a su API, permite una programación muy sencilla del robot.
- **Servidor RIP:** Para realizar la comunicación entre la interfaz web desplegado en el entorno de laboratorios remotos UNILabs y el robot se ha extendido el desarrollo de la implementación de un servidor RIP ya existente. Este servidor se ejecutará en la propia placa Raspberry Pi.
- **Interfaz EjsS (Web):** La interfaz EjsS se ejecutará en el servidor de laboratorios remotos UNILabs integrada con *Blockly* y será accedida a través de un PC al tratarse de una aplicación con interfaz web. Por tanto la aplicación se podrá controlar desde el PC cliente y a través de ella el usuario podrá manejar el robot.

Como se ha explicado con anterioridad la aplicación EjsS es en el fondo una herramienta que permite crear una apliación web que se aloja en el servidor de laboratorios remotos UNILabs y se ejecuta en el navegador del cliente, la cual está integrada con el framework *Blockly* para facilitar la interacción del usuario con el robot y tener de esta forma un entorno de desarrollo / aprendizaje de entrada más sencillo que el lenguaje de programación Python. Sin embargo, y para facilitar el desarrollo de aplicaciones en dicho lenguaje de programación de alto nivel se ha incluido un editor de código Python en la aplicación EjsS que permita al usuario desarrollar su propio código de control del robot y enviarlo al robot para su ejecución. Gracias a la placa Raspberry PI se puede establecer la comunicación entre el servidor y la plataforma robótica, 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 los apartados posteriores se explicará con detalle como son los componentes software desarrollados y que en que elemento de la siguiente figura se encuentran instalados y operativos.

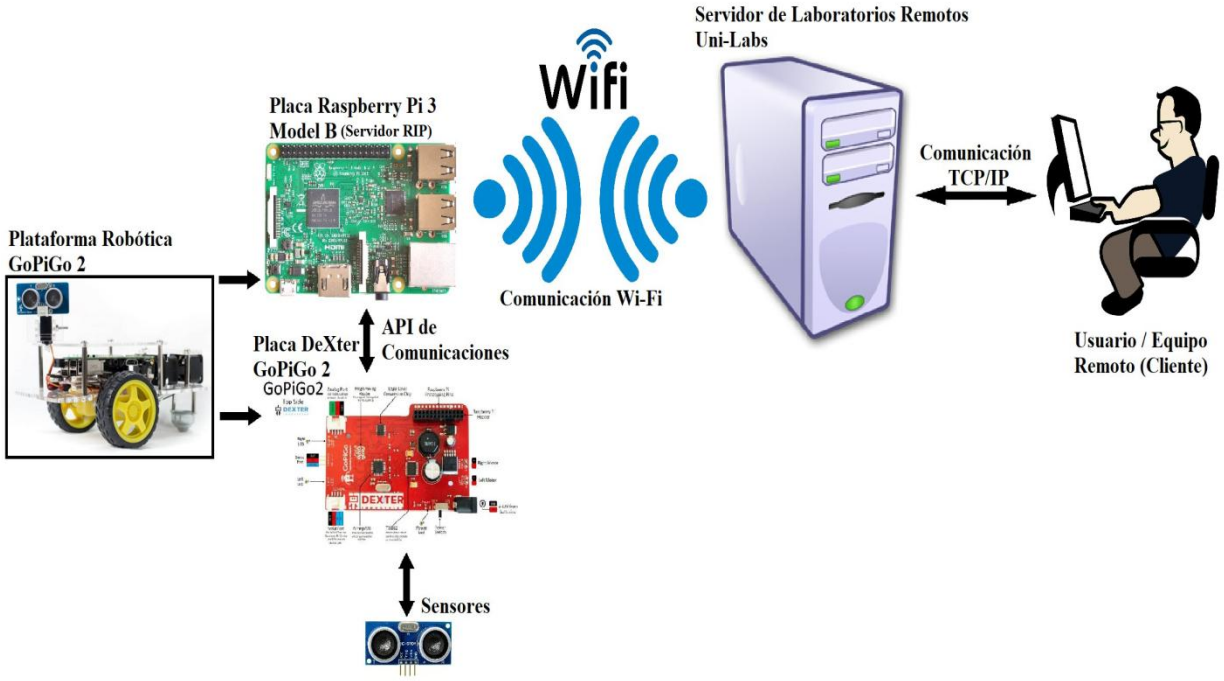


Figura 4. Arquitectura de comunicaciones entre los diferentes componentes del trabajo.

La arquitectura de comunicaciones es la siguiente:

1. A través de la interfaz web proporcionada por la aplicación EjsS desplegada en el servidor de laboratorios remotos, el usuario envía los comandos al robot. Esto es posible ya que la aplicación EjsS tiene un cliente RIP que se comunica con el servidor RIP desarrollado y desplegado en el robot.
2. El servidor RIP está funcionando en la placa Raspberry Pi, escuchando las peticiones de la aplicación EjsS. 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, se verán en detalle cada uno de los diferentes componentes de la arquitectura.

3.1. Plataforma Robótica. Descripción y Montaje.

En este apartado se describe la plataforma robótica GoPiGo 2, la cual se ha utilizado para el desarrollo de este trabajo de fin de máster. El robot GoPiGo 2 es un robot comercial desarrollado por la empresa Dexter Industries [1]. Se trata de un kit que permite construir un robot en forma de "coche" tal y como se puede apreciar en la Figura 5.

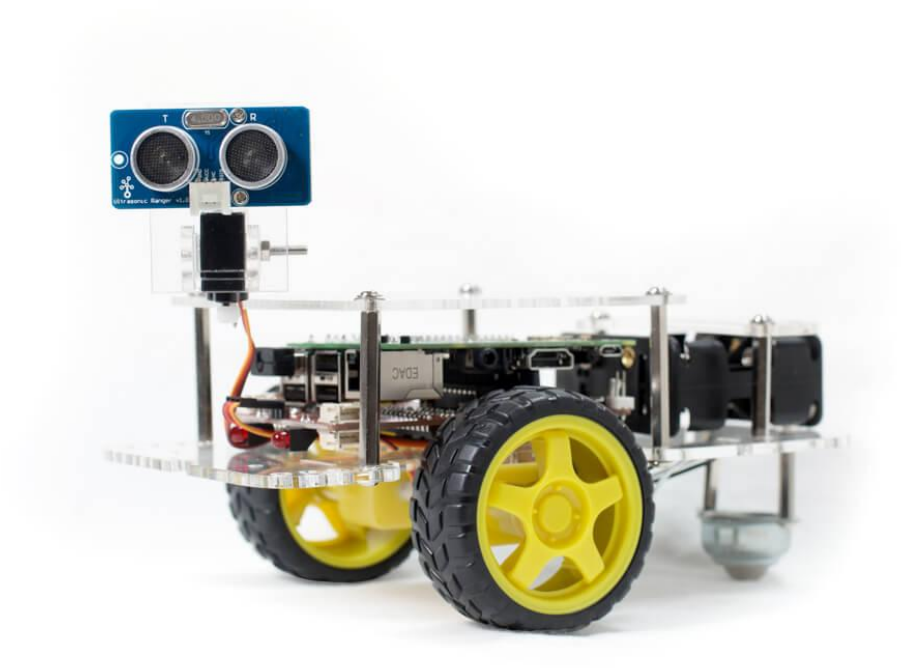


Figura 5. Robot GoPiGo-2 de Dexter Industries.

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 bola loca de apoyo. Sobre esta plataforma se posiciona el resto de elementos del robot. El robot dispone de un API, en diferentes lenguajes de programación, mediante el cual se pueden desarrollar programas de control para el robot.

3.1.1. Descripción de la placa Raspberry Pi.

Como se ha visto al comienzo de este capítulo uno de los principales componentes del robot GoPiGo-2 es la placa Raspberry Pi. Raspberry Pi [10] es un computador montado sobre una placa de tamaño reducido SBC (*"Single-Board Computer"*) de bajo coste. El sistema operativo instalado en esta placa es open-source y se corresponde con una versión libre del sistema operativo Debian optimizado para su uso en las placas Raspberry Pi y recibe el nombre de Raspbian [11]. Esta distribución incluye herramientas de desarrollo para diferentes IDE, entre ellos el lenguaje de programación Python, el cual ha sido utilizado en el desarrollo del presente trabajo. Existen diferentes modelos de Raspberry Pi, el modelo que utiliza este robot es en concreto la versión Raspberry Pi 3 Model B.

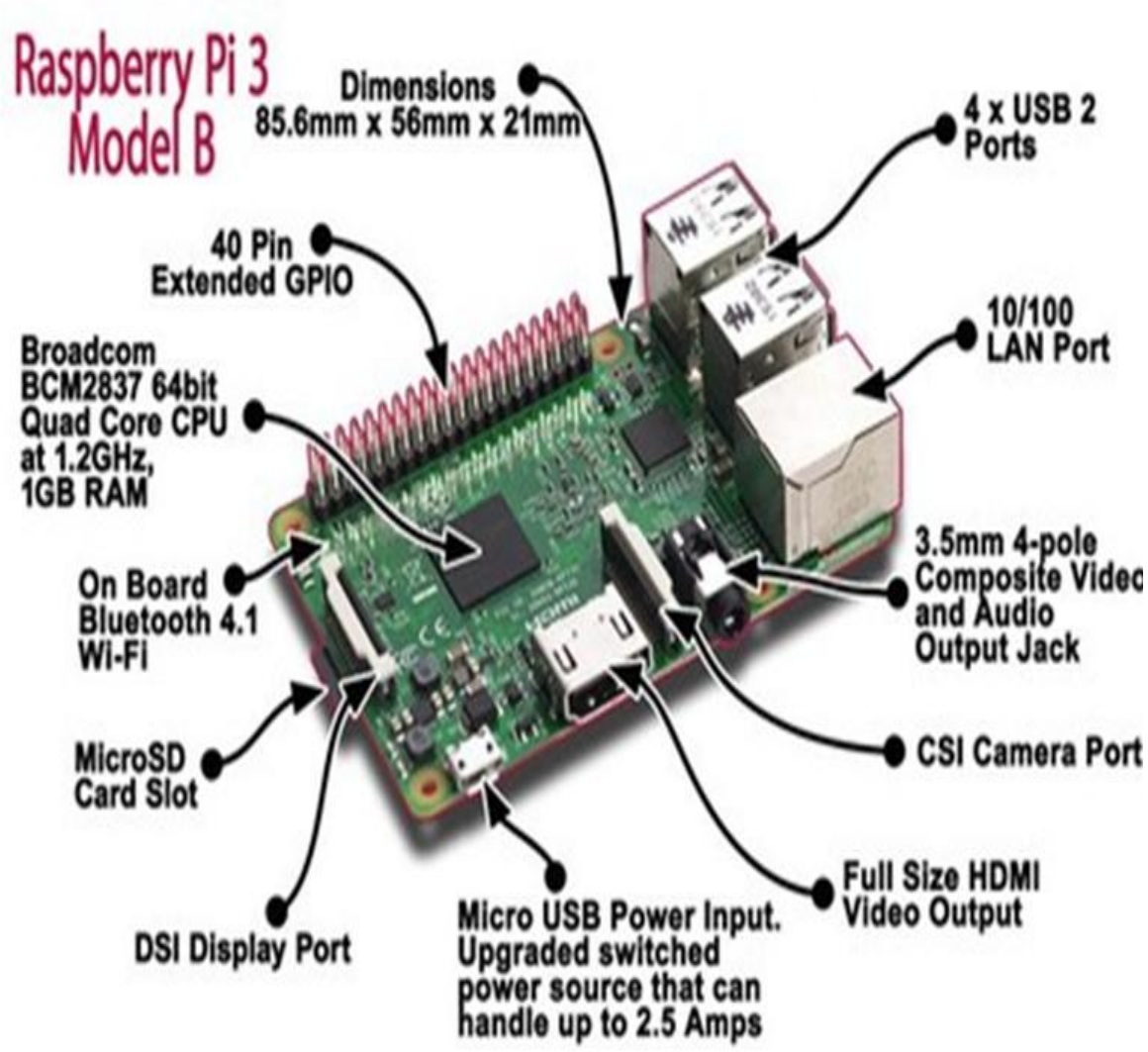


Figura 6. Raspberry Pi 3 Model B. [1].

Las especificaciones técnicas principales de la placa Raspberry Pi 3 Model B son las siguientes:

- Procesador a 1,2 Ghz quad-core de 64-bits con arquitectura ARMv8.
- Módulo de comunicaciones inalámbricas 802.11 Wireless LAN (Wi-Fi).
- Módulo de comunicaciones inalámbricas 802.15.1 Bluetooth 4.1.
- 1 GB de memoria RAM.
- 4 puertos USB.
- 40 pines GPIO (pines de propósito general I/O).
- 1 puerto HDMI.
- 1 puerto Ethernet.
- Interfaz de conexión para cámara (CSI).
- Interfaz de conexión para pantalla / display (DSI).
- Slot para tarjeta Micro SD.

La compañía Dexter Industries proporciona dos tipos de imágenes para trabajar con la placa Raspberry Pi. La primera es una imagen de Raspbian expresamente adaptada al robot GoPiGo-2. Esta imagen (*Raspbian for Robots*), 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.

La segunda es una imagen conocida con el nombre de DexterOS. Esta imagen monta una versión cerrada de Debian en la placa Raspberry Pi que permite a los usuarios menos experimentados empezar a desarrollar programas en Python o bien usando un framework propietario conocido como Bloxter el cual permite realizar la programación de las acciones del robot de forma gráfica mediante la composición de diagramas de bloques tal y como lo permite el framework *Blockly*. El hecho de no usar esta imagen y Bloxter en el presente trabajo es debido a que esta imagen DexterOS no puede ser modificada por el usuario para actualizar los paquetes base con los que viene de serie. Esto último es necesario como se verá en el apartado correspondiente (4.4) ya que para poder usar el servidor RIP es necesario disponer de permisos de administrador y actualizar los paquetes del lenguaje de programación Python a la versión 3, acciones que se quedan fuera de las capacidades de la imagen DexterOS. Por todo ello se usará la imagen Raspbian que ofrece mejores opciones de configuración al desarrollador.

3.1.2. Descripción de la placa Dexter Industries GoPiGo-2.

Como se ha explicado con anterioridad, esta placa sirve de interfaz entre los sensores y actuadores conectados al robot y la placa Raspberry Pi. Tiene diferentes conexiones, que permiten acoplar diferentes dispositivos al robot por medio de conexiones únicas. El principal inconveniente que tiene la placa es que no se puede conectar cualquier dispositivo a la misma, sino que estos dispositivos deben de estar preparados para trabajar con ella.

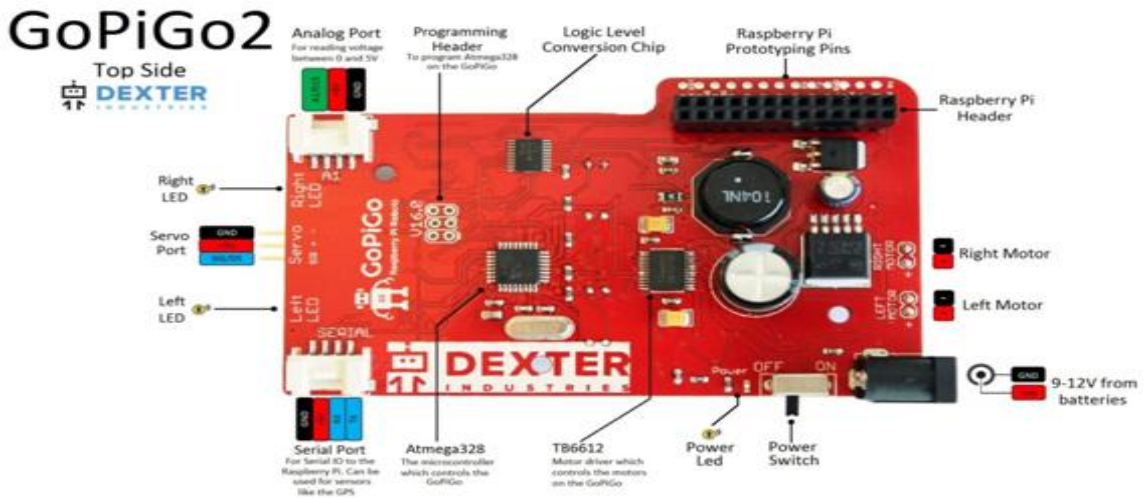


Figura 7. Placa Dexter Industries GoPiGo-2. Vista principal. [1].

Esto reduce el número de dispositivos que se pueden conectar al robot. En las figuras 7 y 8 se puede observar el aspecto de la placa y los diferentes tipos de conexiones que dispone.



Figura 8. Placa Dexter Industries GoPiGo-2. Vista trasera. [1].

Las características principales de la placa de Dexter Industries *"GoPiGo-2"* son las que se incluyen a continuación:

- **Puerto analógico:** la placa dispone de un puerto analógico el cual permite leer valores de voltaje comprendidos entre 0v y 5v. Permite conectar el sensor de ultrasonidos al robot.

- **Puerto serie:** este puerto puede ser utilizado para conectar diferentes tipos de sensores, entre ellos, un GPS.

- **Puerto digital/PWM:** este puerto es utilizado para leer o escribir datos digitales con modulación PWM (modulación por ancho de pulsos) en un sensor.

- **Puerto I2C:** este puerto puede ser utilizado para acoplar al robot sensores del tipo giro-compás o acelerómetros.

- **Encoders ópticos:** la placa incluye dos encoders ópticos que permiten medir la rotación producida en cada una de las ruedas. La precisión es de 18 pulsos de cuenta por cada rotación en las ruedas.

- **Conector para servo-motor:** la placa incluye un conector para el servo-motor. Sobre este servo-motor se pueden acoplar otros elementos como el sensor de ultra-sonidos o la pi-camera (este último elemento va conectado al puerto CSI de la placa Raspberry Pi 3 Model B).

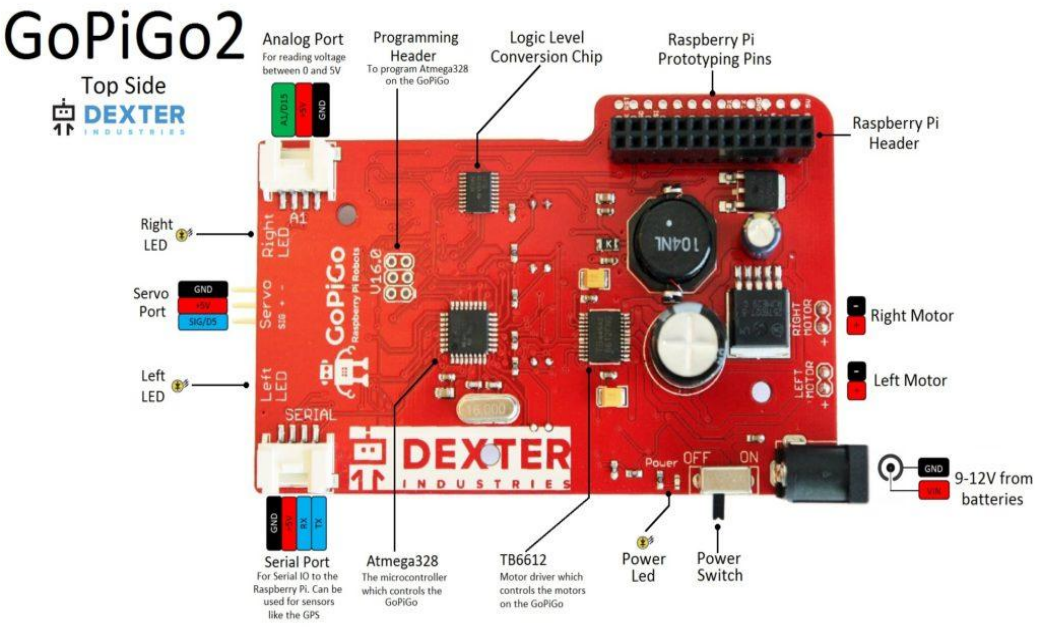


Figura 9. Placa Dexter Industries GoPiGo-2. Componentes en la Vista principal. [1].

- **Conectores para LED:** la placa incluye dos conectores para acoplar dos diodos LED en su panel frontal.

- **Microcontrolador Atmega 328:** este microcontrolador es el encargado de llevar a cabo todo el control de los motores así como la escritura y lectura de los sensores. El microcontrolador hace de intérprete entre la placa Raspberry Pi y los sensores y actuadores del robot GoPiGo 2. Por tanto se encarga del envío, recepción y ejecución de los comandos enviados por la placa Raspberry Pi a través de su API.

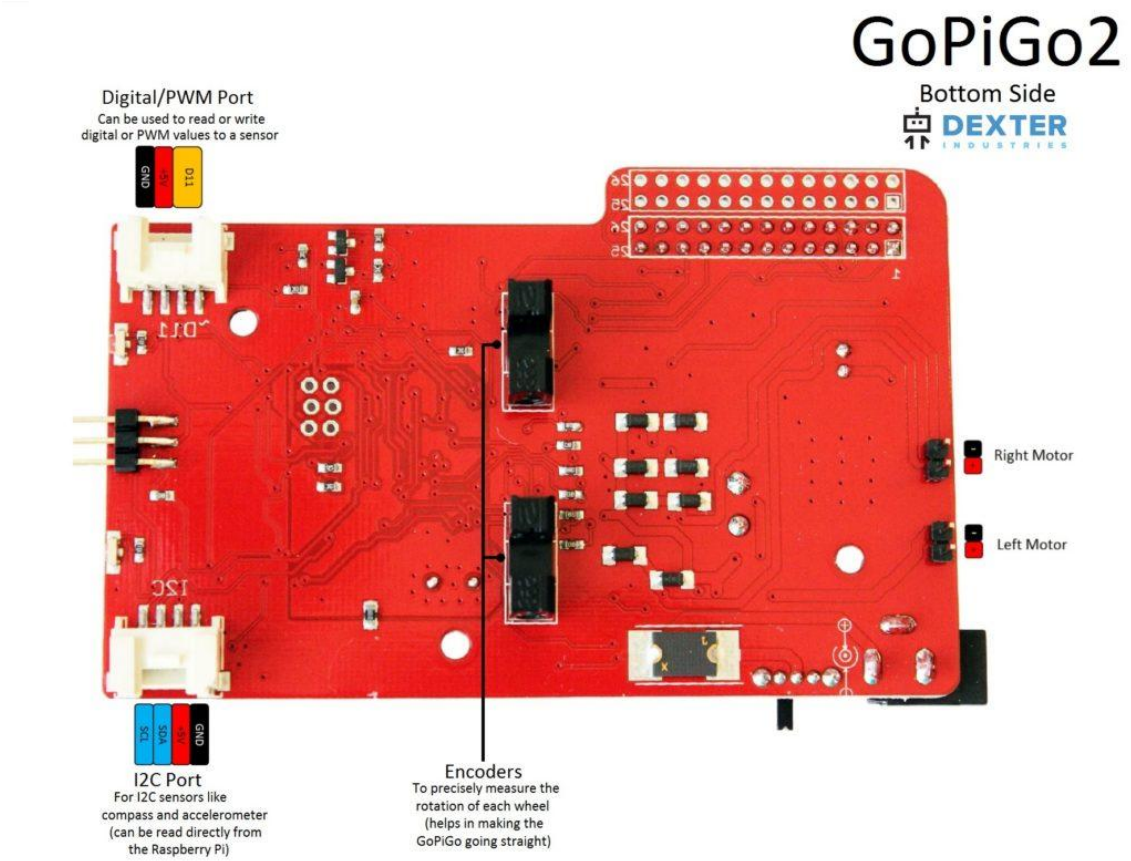


Figura 10. Placa Dexter Industries GoPiGo-2. Componentes en la Vista trasera. [1].

Aunque el montaje de la plataforma robótica no conlleva excesiva complejidad se han incluido en el Anexo A los pasos necesarios para montar el robot así como para instalar el firmware del mismo y actualizar/instalar las librerías necesarias para el desarrollo posterior del servidor RIP.

Una vez vistos los principales componentes hardware que forman parte de la plataforma robótica, se va a proceder en el siguiente apartado a describir el API de programación del robot.

3.2. API Python de programación de la plataforma robótica.

En el presente apartado se va a proceder a presentar la API de programación de la plataforma robótica GoPiGo 2. El robot incluye una API de programación que se puede utilizar por parte de los desarrolladores para definir programas de control del robot (ya sean básicos o más complejos). A través de esta API, se puede enviar, de forma muy sencilla, cualquier orden al robot.

La API de programación está disponible en diferentes lenguajes de programación aunque los recomendados en la propia web del robot son dos: Python o Scratch. Para el desarrollo de este trabajo se ha utilizado el lenguaje de programación Python. Los motivos de utilizar este lenguaje de programación son básicamente dos:

- Se trata de un lenguaje de programación sencillo de utilizar y cuya curva de aprendizaje es relativamente suave si se tienen conocimientos previos de programación. Además, existen multitud de ejemplos de programación asociados a este lenguaje por lo que se hace más sencillo el inicio en el desarrollo de software de control para la plataforma robótica.

- Como ya se ha comentado con anterioridad en el presente trabajo se pretende utilizar el framework *Blockly* para describir de forma sencilla y mediante bloques funcionales (programación gráfica) el software de control de la plataforma robótica. El framework *Blockly* es compatible con el lenguaje de programación Python y permite definir nuevos bloques funcionales empleando dicho lenguaje de programación. Por ello es una buena opción el utilizar el lenguaje de programación Python con el framework *Blockly*.

En los sub-apartados siguientes se va a proceder a comentar las funciones más importantes del API de programación en lenguaje Python. No obstante, en la propia página de Dexter Industries [10] se dispone de toda la documentación y tutoriales necesarios para empezar a utilizar la API de programación.

3.2.1.1. Funciones de control de movimiento de los motores.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control del movimiento de los motores.

Función	Descripción
fwd ()	Función del API que permite mover el robot GoPiGo-2 hacia adelante en línea recta mediante el uso de un controlador PID [13].
motor_fwd()	Función del API que permite desplazar el robot hacía adelante pero sin utilizar el controlador PID.
bwd()	Función del API que permite mover el robot GoPiGo hacia atrás en línea recta mediante el uso de un controlador PID.
motor_bwd()	Función del API que permite mover el robot hacia atrás sin utilizar el controlador PID.
left()	Función del API que permite desplazar el robot hacia la izquierda (el motor derecho estaría parado).
left_rot()	Función del API que permite rotar el robot hacía la izquierda (ambos motores estarían moviéndose en dirección contraria).
right()	Función del API que permite mover el robot hacía la derecha (el motor izquierdo estaría parado).
right_rot()	Función del API que permite rotar el robot hacía la derecha (ambos motores estarían moviéndose en dirección contraria).
stop()	Función del API que permite detener el robot GoPiGo.

Tabla 1. API de programación de la plataforma robótica. Funciones para el control del movimiento de los motores.

3.2.1.2. Funciones de control de velocidad de los motores.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control de velocidad de los motores.

Función	Descripción
increase_speed()	Función del API que permite incrementar la velocidad de los motores en 10 unidades
decrease_speed()	Función del API que permite decrementar la velocidad de los motores en 10 unidades.
set_left_speed(speed)	Función del API que permite configurar / modificar la velocidad del motor izquierdo. El valor del parámetro speed es un número entero que puede variar entre 0 y 255.
set_rigth_speed(speed)	Función del API que permite configurar / modificar la velocidad del motor derecho. El valor del parámetro speed es un número entero que puede variar entre 0 y 255.
set_speed(speed)	Función del API que permite modificar la velocidad de ambos motores. El valor del parámetro speed es un número entero que puede variar entre 0 y 255.

Tabla 2. API de programación de la plataforma robótica. Funciones para el control de velocidad de los motores.

3.2.1.3. Funciones de control de los encoders.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control de los encoders.

Función	Descripción
enc_tgt(m1,m2,target)	Función del API que permite configurar los encoders para que el robot se desplace una distancia, siendo: • **m1:** parámetro de entrada que indica si el encoder del motor 1 se debe de mover la distancia configurada. 0 no se debe de mover, 1 se debe de mover. • **m2:** parámetro de entrada que indica si el encoder del motor 2 se debe de mover la distancia configurada. 0 no se debe de mover, 1 se debe de mover. • **target:** parámetro de entrada que indica el número de pulsos que deben de recorrer los encoders.
enable_encoders()	Función del API que permite activar los encoders.
disable_encoders()	Función del API que permite desactivar los encoders.

Tabla 3. API de programación de la plataforma robótica. Funciones para el control de los encoders.

3.2.1.4. Funciones de control del estado de la plataforma robótica.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control del estado de la plataforma robótica.

Función	Descripción
volt()	Función del API que permite obtener el estado de la batería en Voltios.
fw_ver ()	Función del API que permite obtener el número de versión del firmware del robot.
enable_com_timeuot(time)	Función del API que permite activar el time-out de las comunicaciones. El parámetro de entrada *"time"* indica el tiempo en milisegundos, y su valor es un número entero que debe estar comprendido entre 0 y 65536.
disable_com_timeout()	Función del API que permite desactivar el time-out en las comunicaciones.
read_status()	Función del API que permite obtener el estado del robot.
read_timeout_status()	Función del API que permite obtener el estado del time-out.

Tabla 4. API de programación de la plataforma robótica. Funciones para el control del estado de la plataforma robótica.

3.2.1.5. Funciones de control del sensor de ultrasonidos.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control del sensor de ultrasonidos.

Función	Descripción
us_dist()	Función del API que permite obtener la distancia leída en centímetros por el sensor de ultrasonidos.

Tabla 5. API de programación de la plataforma robótica. Funciones para el control del sensor de ultrasonidos.

3.2.1.6. Funciones de control del sensor LED.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control del sensor LED.

Función	Descripción
led_on()	Función del API que permite encender el led indicado como parámetro.
led_off()	Función del API que permite apagar el led indicado como parámetro

Tabla 6. API de programación de la plataforma robótica. Funciones para el control del sensor LED.

3.2.1.7. Funciones de control del motor SERVO.

En este apartado se va a proceder a presentar las funciones del API de programación de la plataforma robótica que permiten realizar el control del servo-motor.

Función	Descripción
enable_servo()	Función del API que permite activar el motor del servo.
disable_servo()	Función del API que permite desactivar el motor del servo.
servo(angle)	Función del API que permite girar el motor del servo hasta la posición indicada (en grados). El parámetro de entrada *"angle"* es el ángulo que debe girar el motor siendo su valor un número entero que debe estar comprendido entre 0 y 180.

Tabla 7. API de programación de la plataforma robótica. Funciones para el control del servo-motor.

Capítulo 4: Arquitectura Desarrollada.

En este capítulo se va a proceder a presentar el software de control y de comunicaciones desarrollado en el presente trabajo fin de máster con el objetivo de proveer de una aplicación de control remoto a través del servidor de laboratorios remotos *UNILabs* que ofrezca de forma sencilla la capacidad de desarrollar software de control para el robot GoPiGo 2 de Dexter Industries presentado en el capítulo anterior.

Como se definió en la presentación del presente trabajo se han desarrollado tres componentes software a diferentes niveles que son los que permiten la comunicación con el robot. Estos tres componentes son:

- **Servidor RIP**: se trata de un componente que implementa la parte servidora del protocolo RIP (*"Remote Interoperavility Protocol"*). Este elemento en realidad está basado en un servidor RIP previo cuya implementación ha sido provista por los tutores del trabajo y que ha sido adaptado para funcionar en el robot GoPiGo 2 añadiendo nuevas funcionalidades (operaciones de tipo *"get"*, *"set"* y *"connect"* para la operación remota del robot) de las que no disponía la versión original.

- **Aplicación de control EjsS**: se trata de una aplicación Web desarrollada con el framework EjsS (*"Easy Java/Javascript Simulations"*). Esta aplicación dispone de una interfaz gráfica a través de la cual se pueden monitorizar los parámetros más significativos del robot así como controlar sus movimientos. Además dispone de un editor integrado que permite escribir código Python, empleando el API de programación del robot, para escribir software de control más elaborado de tal forma que el robot pueda funcionar en modo automático desarrollando las acciones descritas en dichos programas.

- **Pieza de Integración con el framework Blockly:** se trata de un componente software que permite integrar la aplicación EjsS desarrollada con el framework *Blockly* que está operativo en el servidor de laboratorios remotos de UNILabs.

En los siguientes apartados se presentan los componentes desarrollados así como una introducción a las tecnologías y frameworks empleados para la implementación de dichas soluciones.

4.1. Servidor RIP implementado en lenguaje de programación Python del robot GoPiGo 2.

En este apartado se va a proceder a describir como es la implementación del servidor RIP realizada para el presente trabajo. El apartado está dividido en dos secciones. En la primera se va a proceder a explicar los fundamentos teóricos del protocolo RIP *"Remote Interoperability Protocol"*, mientras que la segunda se centra en los pormenores del desarrollo realizado.

4.1.1. Protocolo RIP. Remote Interoperability Protocol.

El protocolo RIP (*"Remote Interoperability Protocol"*) surge en el contexto actual en el cual han aparecido multitud de nuevos servicios en el ámbito de la educación que permiten la formación y el aprendizaje de diversas materias por medio de aplicaciones distribuidas. Estas aplicaciones distribuidas conocidas como *"Laboratorios Virtuales"* permiten a los alumnos desarrollar experimentos de diversa índole sin necesidad de estar físicamente en las dependencias académicas.

De esta forma usando un ordenador personal (cliente) los alumnos pueden establecer una conexión con los servicios provistos por dichos *"Laboratorios Virtuales"* para realizar experimentos en remoto que pueden ir desde realizar simulaciones a controlar plataformas robóticas.

En un escenario común, el cliente se conecta al recurso y adquiere información básica sobre las capacidades del sistema. Después de la fase de negociación, el cliente debe haber obtenido suficiente conocimiento sobre las formas de interactuar con el sistema. Las funcionalidades básicas que debe soportar cualquier aplicación distribuida planteada como *"Laboratorio Virtual"* son las siguientes:

- **Conexión y control de sesión**: estos métodos permiten establecer la comunicación con el servidor, configurar la ejecución y, finalmente, terminar la conexión correctamente.

- **Control de transmisión de datos:** a esta categoría, pertenecen todos aquellos métodos que pueden leer o modificar el estado del servidor.

- **Ejecución de comandos:** la aplicación del cliente debe ser capaz de ejecutar comandos específicos del servidor. El protocolo sólo define la forma de enviar los comandos. La interpretación de estos comandos es responsabilidad del servidor y estos pueden ser tan simples o tan complejos como las necesidades del servidor.

Desde el punto de vista tecnológico, el protocolo RIP está basado en el uso del protocolo JSON-RPC el cual es un protocolo de llamada a procedimiento remoto codificado en JSON. Es un protocolo muy simple (y muy similar a XML-RPC), que define solo algunos tipos de datos y comandos. JSON-RPC permite notificaciones (datos enviados al servidor que no requieren una respuesta) y que se envíen varias llamadas al servidor que pueden ser contestadas después de la recuperación de una caída del servicio.

En la Figura 11 muestra un diagrama de secuencia correspondiente a una sesión típica en el seno del protocolo RIP empleando para ellos las facilidades que le ofrece el protocolo JSON-RPC en el que está basado. Después de establecer la conexión, generalmente el cliente entra en un bucle donde actualiza periódicamente los valores de algunas variables en el servidor, ejecuta uno o más comandos y pregunta por los nuevos valores de las variables de interés.

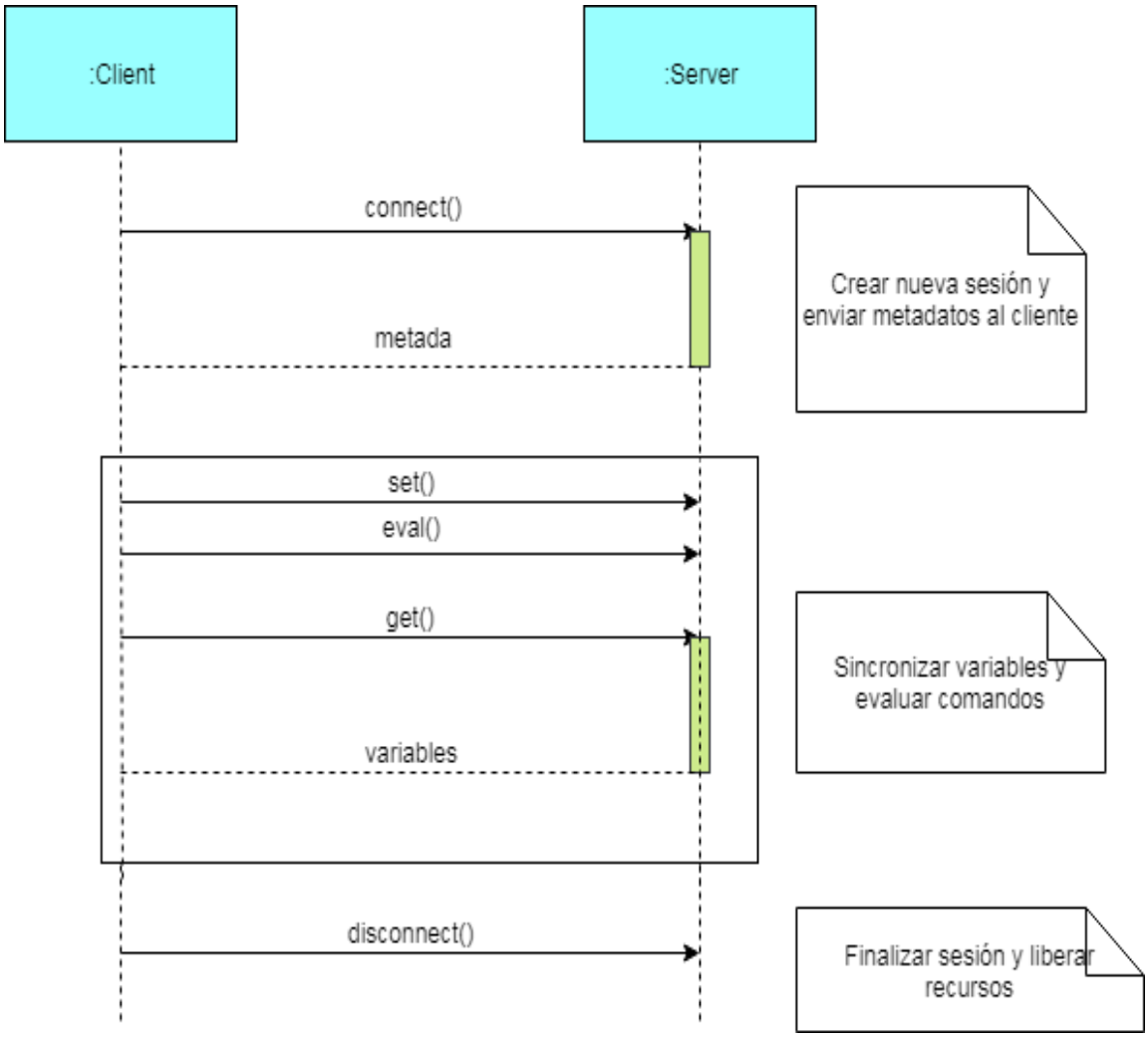


Figura 11. Diagrama de secuencia de una sesión del protocolo RIP basado en los servicios del protocolo JSON-RPC.

Los párrafos siguientes contienen, para cada método, una breve descripción del propósito y los parámetros esperados.

- **Control de conexión**: la implementación de los métodos connect() y disconnect() es obligatoria, porque el sistema debe proporcionar una forma de establecer y de terminar la conexión de forma ordenada.
    - **connect():** para el establecimiento de la conexión el servidor debe informar algunos parámetros al cliente:
        - Una lista de los métodos que se proporcionan por parte del servidor.
        - Una lista de todas las variables disponibles en el servidor, indicando si son de lectura, escritura o de lectura/escritura.
    - **disconnect():** este método permite finalizar el conexión con el servidor de una manera adecuada, cerrando cualquier recurso abierto que el cliente pueda haber abierto.

- **Control de transmisión de datos:** con respecto a la transmisión de datos, es necesario disponer de métodos set() y get(). Puede ocurrir que haya sistemas en los cuales sólo existan variables de lectura o bien variables de sólo escritura. En cualquier caso es obligatorio disponer de un método get() siempre que existan variables de lectura, siendo estas las mismas variables que luego puedan ser modificadas por los métodos set().
    - **get():** recibe una variable o lista o variables de las informadas por el servidor como de lectura y devuelve los valores de dichas variables.
    - **set():** recibe una variable o lista o variables de las informadas por el servidor como de escritura, junto con los valores que se asignarán a dichas variables.

- **Control de sesión**: los métodos y comandos de control de sesión son opcionales, es decir, pueden implementarse o no, porque dependen de las capacidades y la complejidad del servidor. A diferencia de los métodos discutidos anteriormente, no siempre son necesarios.
    - **open():** la invocación del método *open* provoca la 'apertura' de un elemento, que puede ser cualquier recurso que el servidor necesite exponer.
    - **close():** un llamada a este método implica el cierre de un recurso que previamente fue abierto.
    - **start():** en caso de que la ejecución del servidor pueda ser iniciada por el usuario esta se debe iniciar con este método. Por ejemplo iniciar una simulación.
    - **stop():** la llamada a este método detiene la actividad iniciada con el método start().

4.1.2. Servidor RIP. Implementación.

En la tarjeta Raspberry PI 3 se encuentra instalado el servidor RIP, implementado en lenguaje de programación Python. Este servidor actúa de interfaz entre el robot y la aplicación cliente web implementada usando el framework EjsS. El servidor utilizado está basado en una adaptación del servidor desarrollado por la UNED, cuyo código se puede encontrar en [6]. El servidor utiliza el protocolo JSON-RPC [14] para el intercambio de información. Recibe y envía la información al robot utilizando su API cuyo código puede ser consultado en el Anexo B del presente trabajo.

Para poder utilizar el servidor RIP implementado en el robot es necesario realizar la instalación de varias librerías en Python para su correcto funcionamiento, las cuales no se encuentran instaladas en la imagen Raspbian que por defecto proporciona Dexter Industries. El procedimiento de instalación de estas librerías (jsonrpc, ujson y cherrypy) está recogido en el ANEXO A.

LIBRERÍA	COMANDO
jsonrpc	sudo pip install jsonrpc
ujson	sudo pip install ujson
cherrypy	sudo pip install cherrypy

Tabla 8. Librerías necesarias para poder utilizar el servidor RIP.

La librería **jsonrpc** representa la implementación en lenguaje de programación Python del protocolo JSON-RPC que utiliza JSON como formato de representación de los mensajes intercambiados. Es un protocolo muy simple (y parecido a XML-RPC), que define solo algunos tipos de datos y comandos. JSON-RPC permite notificaciones (datos enviados al servidor que no requieren una respuesta) y que se envíen varias llamadas al servidor que pueden ser contestadas tras la recuperación después de una caída de servicio.

Por su parte, la librería **ujson** *"Ultra-JSON"* es un codificador / decodificador de mensajes en formato JSON muy rápido y eficiente, mientras que **cherrypy** es un framework de desarrollo de aplicaciones web orientado a objetos que utiliza el lenguaje de programación Python como base. Está diseñado para el rápido desarrollo de aplicaciones web al utilizar el protocolo HTTP como *wrapper* pero se mantiene en un nivel bajo y no ofrece mucho más de lo que se define en la RFC 7231.

4.1.2.1. Arquitectura del Servidor RIP.

La implementación del servidor RIP se ha divido en varias clases, con el objetivo de facilitar su modularidad y futuras ampliaciones. En la siguiente tabla se muestra las principales clases del servidor con una breve descripción. Su implementación se puede encontrar en el Anexo B.

Clase	Fichero	Descripción
HttpServerGoPiGo	**HttpServerGoPiGo.py**	Clase principal encargada de arrancar el servidor HTTP. Recibe los comandos del interfaz WEB / EJsS y envía las respuestas.
GoPiGoRestController	**GoPiGoRestController.py**	Esta clase implementa el servidor JSON-RPC exponiendo un API REST. Se encarga de interpretar los mensajes recibidos por el servidor HTTP. Expone tres end-points básicos: *"get"*, *"set"* y *"connect"*.
GoPiGoService	**GoPiGoService.py**	Esta clase implementa la lógica de control de la comunicación con el robot GoPiGo2. Controla la lógica de ejecución del robot y es la encargada de leer y enviar datos al mismo.
GoPiGoContext	**GoPiGoContext.py**	Esta clase implementa el acceso directo a ciertas variables de estado del robot. Permite también que el usuario pueda almacenar variables propias que pueda necesitar para la programación del robot. Este contexto funciona como *"HashMap"* con clave y valor.

Tabla 9. Clases que forman parte de la implementación del Servidor RIP.

La implementación del servidor realizada está planteada siguiendo el patrón MVC (Modelo-Vista-Controlador / *"Model, View, Controller"*). MVC es un patrón de arquitectura de software (Figura 12), que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

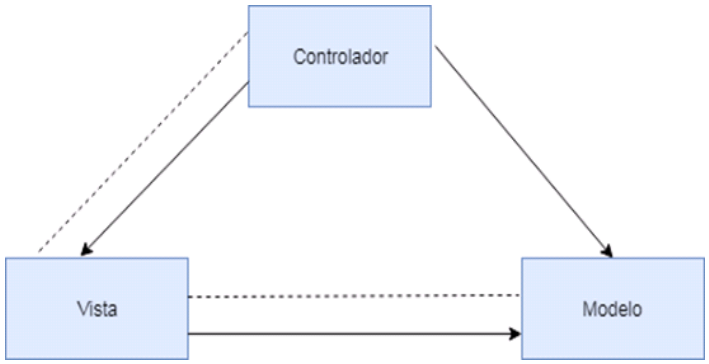


Figura 12. Elementos básicos que componen el patrón MVC.

Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. Los elementos que conforman este patrón son los siguientes:

- **Modelo (*"Model"*):** es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, ya sean consultas o actualizaciones, implementando también los privilegios de acceso a la lógica de negocio. Envía a la *"Vista / View"* aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al *"Modelo"* a través del *"Controlador / Controller"*. En la implementación realizada el *"Modelo"* se corresponde con la clase *"GoPiGoService"*.

- **Controlador (*"Controller"*):** este elemento responde a eventos y realiza peticiones al *"Modelo"* cuando se hace alguna solicitud sobre la información. En la implementación realizada el *"Controlador"* se corresponde con la clase *"GoPiGoRestController"*.

- **Vista (*"View"*):** presenta el *"Modelo"* en un formato adecuado para interactuar con la interfaz de usuario. En la implementación realizada el *"Controlador"* se corresponde con la clase *"GoPiGoRestContext"*.

Como se ha visto el patrón MVC es solo una forma de estructurar el código del lado del servidor. En la tabla 9 se especifica que la clase *"GoPiGoRestController"* ofrece un API REST que se encarga de interpretar los mensajes recibidos por el servidor HTTP, así pues este elemento se ocupa de cómo se comunican el cliente y el servidor. El formato de mensajes intercambiados a través del API REST del servidor RIP es JSON el cual es un formato de representación de información que en los últimos años ha ido sustituyendo de forma paulatina al XML como solución de intercambio de información en los servicios debido a que es más ligero y sencillo de procesar (consume menos recursos).

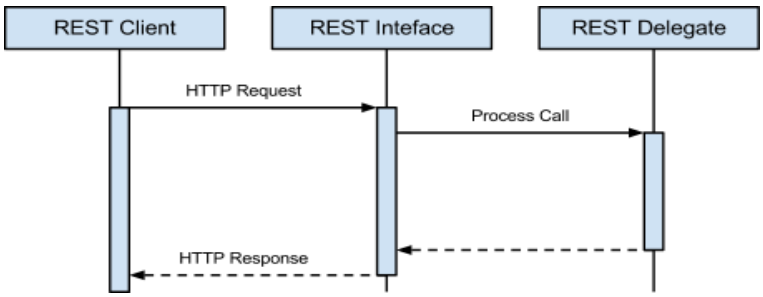


Figura 13. Interacción básica entre un cliente y un servidor a través de una interfaz REST.

Como se puede apreciar en la Figura 13 el ciclo básico de interacción entre un cliente REST (como pueda ser una aplicación EjsS) y un servidor REST (cómo el servidor RIP implementado) compone muy poca lógica y más si se tiene en cuenta que en la implementación realizada se exponen sólo tres end-points por parte del servidor: *"get"*, *"set"* y *"connect"*. Por su parte la implementación del servidor RIP realizada expone tres endpoints que pueden invocados desde un cliente. En el ANEXO B. se puede encontrar la relación de clases que permiten realizar tests de integración contra los endpoints del servidor RIP. Estos endpoints son los siguientes:

ENDPOINT	DESCRIPCIÓN
http://<dir_IP>:2055/RIP	(Petición GET) Permite obtener información sobre el servidor.
http://<dir_IP>:2055/RIP/POST	(Petición POST) Permite realizar las llamadas JSONRPC relativas a los métodos *"connect"*, *"get"* y *"set"*.
http://<dir_IP>:2055/RIP/SSE	(Petición POST) Canal SSE para el envío de variables.

Tabla 10. Relación de endpoints ofrecidos por la implementación del servidor RIP / GoPiGo-2 realizada.

En el momento en el que se arranca la implementación del servidor RIP éste se queda en espera hasta recibir las peticiones desde la interfaz web / EjsS. 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 se puede consultar en la clase *"GoPiGoRestController"* del Anexo B. 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.

Tabla 11. Variables del servidor RIP actualizables a través del método SET (I).

Variable	Tipo	Descripción
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.
setledon	Entero	Indica al robot si se debe encender el led indicado. (Valores: 0 / 1).
setledoff	Entero	Indica al robot si se debe apagar el led indicado. (Valores: 0 / 1).
resetmap	Void	Resetea las variables internas del robot a sus valores iniciales.

Tabla 12. Variables del servidor RIP actualizables a través del método SET (II).

A continuación se muestran las variables que pueden ser consultadas a través del método *get*.

Variable	Tipo	Descripción
getultrasonic	Entero	Devuelve el valor del sensor de ultrasonidos en centímetros.
getstatus	Entero	Devuelve el estado del robot.
getvoltage	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. • "automático" = Si el robot se encuentra en modo automático.
getvelmotorizq	Decimal	Devuelve la velocidad del motor izquierdo.
getmotorder	Decimal	Devuelve la velocidad del motor derecho.
getrobotposx	Decimal	Devuelve la posición X del robot.
getrobotposy	Decimal	Devuelve la posición Y del robot.
getobstacleposx	Decimal	Devuelve una lista con las posiciones X de los obstáculos.
getobstacleposy	Decimal	Devuelve una lista con las posiciones Y de los obstáculos.
getorientacion	Decimal	Devuelve la orientación del robot, en radianes.
getLog	String	Devuelve el último mensaje de traza.

Tabla 13. Variables del servidor RIP consultadas a través del método GET.

4.1.2.2. Servidor RIP. Diagrama de flujo del Servidor RIP instalado en el Robot.

El esquema de funcionamiento del servidor RIP desarrollado e instalado en el robot es el siguiente:

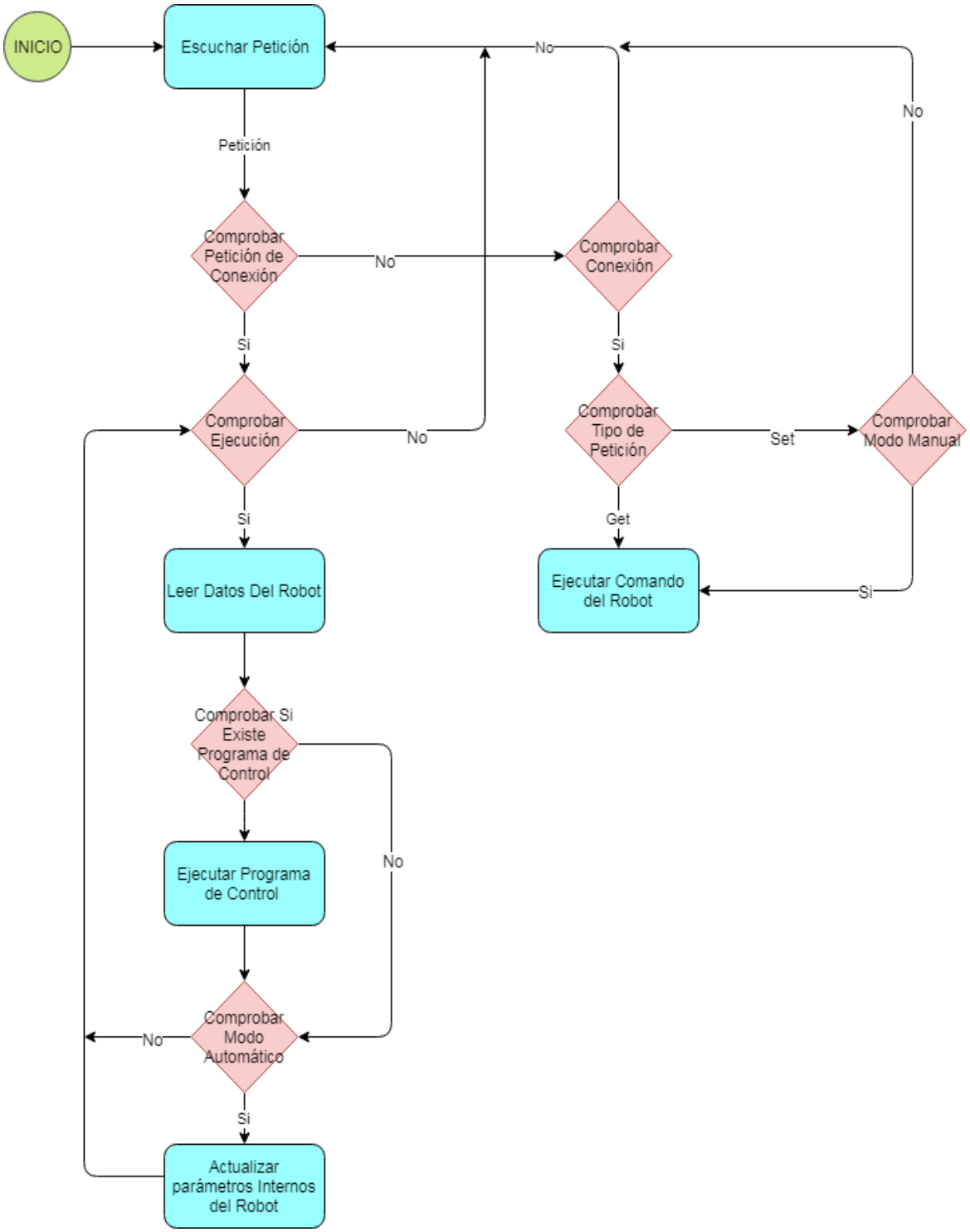


Figura 14. Servidor RIP. Diagrama de Flujo del Ciclo de trabajo completo del robot GoPiGo 2.

Como se puede apreciar en la figura anterior el ciclo de trabajo está dividido en dos partes. Un ciclo de trabajo global y un ciclo de trabajo que se activa cuando el robot está en modo automático en el cual el robot realiza acciones más específicas en función de la existencia de un programa de control. El funcionamiento del robot durante el ciclo de trabajo global es el siguiente:

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

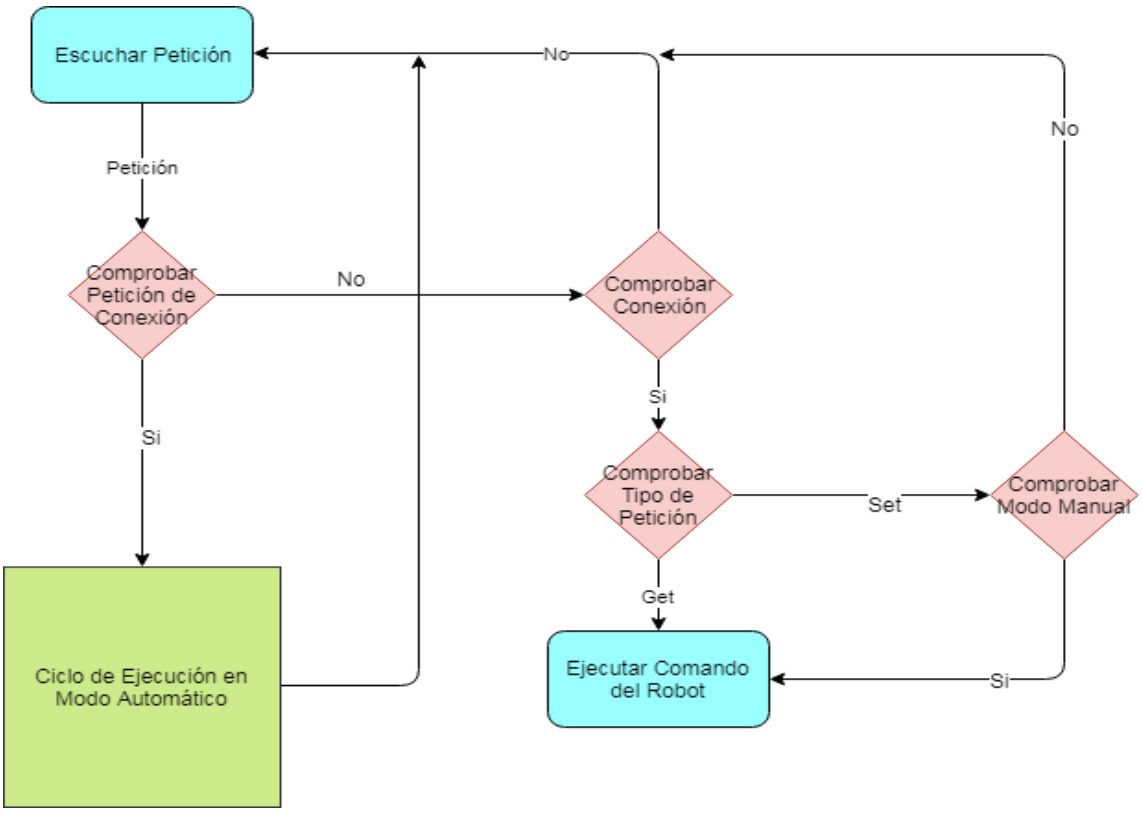


Figura 15. Servidor RIP. Diagrama de Flujo del Ciclo de trabajo global del robot GoPiGo 2.

Por su parte el funcionamiento del ciclo de ejecución en modo automático del robot es el siguiente:

1. El robot se encuentra a la espera de recibir las peticiones enviadas por el usuario desde la interfaz de usuario. En este punto, se deben haber inicializado todos los parámetros del robot.
2. 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 un programa de control cargado en el éste es ejecutado.
5. Finalmente, y si el robot se encuentra en modo automático, se actualiza sus parámetros y se vuelve al inicio del ciclo.

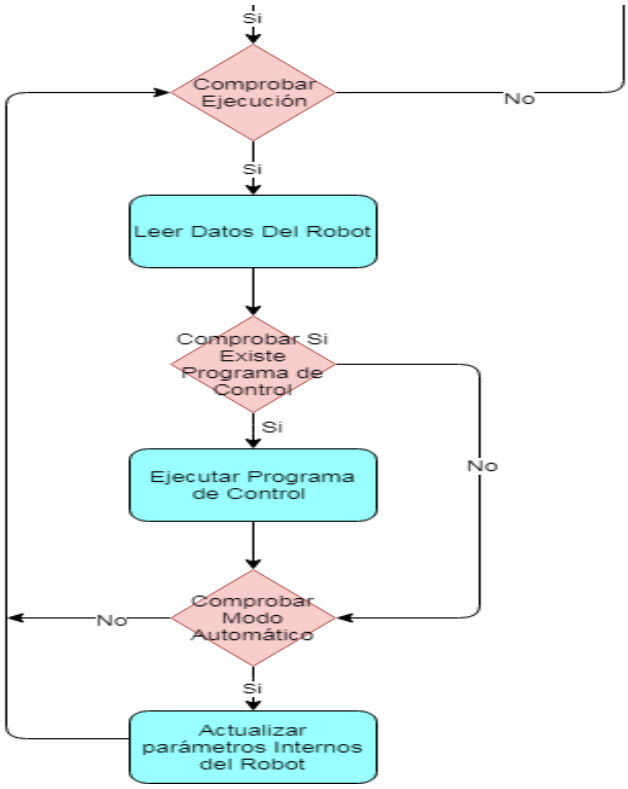


Figura 16. Servidor RIP. Diagrama de Flujo Ciclo de trabajo en modo automático del robot GoPiGo 2.

Como se ha indicado anteriormente, el usuario puede acceder directamente al API del robot desde el programa de control. Para conseguir este propósito se ha creado una clase denominada *GoPiGoContext*, con la cual 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. En los siguientes sub-apartados se presentan los aspectos más relevantes de la clase *GoPiGoContext* y como se debe de emplear para trabajar con el robot.

4.1.2.3. Servidor RIP. Clase GoPiGoContext.

A continuación se va a proceder a comentar en detalle las principales propiedades de la clase *GoPiGoContext*. El código de esta clase se puede encontrar en el Anexo B.

Variable	Descripción
VELMOTORIZQ	Indica la velocidad del motor izquierdo.
VELMOTORDER	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.
ENCMOTORIZQ	Indica la lectura del encoder del motor izquierdo.
ENCMOTORDER	Indica la lectura del encoder del motor derecho.
ENCSTATUS	Indica el estado de los encoders.
robotPosX	Posición X del robot
robotPosY	Posición Y del robot
obstaclePosX[]	Array con las posiciones X de los obstáculos
obstaclePosY[]	Array con las posiciones Y de los obstáculos
orientacion	Orientación del robot
auxiliar	Variable que permite el usuario guardar cualquier tipo de información durante la ejecución del programa de control.
DIRROBOT	Indica la dirección de movimiento del robot. Posibles valores: • 1 = Avanzar. • 2 = Retroceder • 99 = Parar.

Tabla 14. Propiedades de la clase GoPiGoContext.

Para utilizar correctamente la clase *GoPiGoContext* se deben tener en cuenta las siguientes consideraciones:

1. Cuando se entra en el ciclo interno de ejecución se crea un objeto de la clase *GoPiGoContext*. 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" (figura 16) 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 (BATTERY), el valor del sensor de ultrasonido (ULTRASONIC), el valor y estado de los encoders (ENCMOTORDER, ENCMOTORIZQ, ENCSTATUS).
3. Cuando se ejecuta el programa de control, si lo hubiese, se pasa como parámetro el objeto de la clase *GoPiGoContext*.
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.

La clase *GoPiGoContext* provee de métodos para recuperar y guardar información que son necesarios para operar correctamente con el robot. A *grosso modo* las operaciones de guardado y obtención de información sigue las siguientes pautas:

- Una de las primeras acciones que debe realizar el programa de control es recuperar el parámetro que permite acceder al objeto de la clase *GoPiGoContext*. Cuando se ejecuta el programa de control, dentro del ciclo de ejecución interna del robot, se le pasa como parámetro dicho objeto. Para recuperarlo se debe escribir el siguiente código:

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

El objeto se recupera como un *"HashMap"*, en la que a cada propiedad le corresponde una clave. Así, por ejemplo, si se desea acceder a la propiedad robotPosX se debe utilizar el siguiente código:

Xpos = gopigocontext ["robotPosX"]

Los valores del diccionario también los pueden ser modificados. Por ejemplo, si se desea cambiar la velocidad del motor izquierdo se debe utilizar el siguiente código:

gopigocontext ["VELMOTORIZQ"]=100

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

gopigocontext ["robotPosX"] = 120

gopigocontext ["robotPosY"] = 50

- Los obstáculos en el mapa están determinados por el valor de las propiedades *obstaclesPosX* y *obstaclesPosY*. Se trata de dos arrays que indican la posición de cada obstáculo en las coordenadas (x, y). Así, por ejemplo, si se desea indicar que hay un obstáculo en la posición (90, 32) se debe utilizar el siguiente código:

  obstaclePosX[0] = 90

  obstaclePosY[0] = 20

  Si a continuación se necesita añadir un nuevo obstáculo en la posición (70, 120) se debe utilizar el siguiente código:

  obstaclePosX[1] = 70

  obstaclePosY[1] = 120

- Las velocidades de los motores están determinadas por el valor de las propiedades ***VELMOTORIZQ*** y ***VELMOTORDER***. Modificando estas propiedades se pueden modificar tanto la trayectoria del robot como su velocidad siempre y cuando el robot se encuentre en modo automático. La relación entre las propiedades de la clase *GoPiGoContext* y los comandos que se envían al robot cuando este se encuentra en modo automático es la siguiente:

Variable en GoPiGoContext	Comando del robot
gopigocontext [“VELMOTORIZQ”]	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.
gopigocontext [“VELMOTORDER”]	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.
gopigocontext [“DIRROBOT”]	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 15. Relación propiedades de la clase GoPiGoContext - API del robot.

- Cada vez que se activa un ciclo de ejecución en modo automático se vuelve a ejecutar el programa de control del usuario. En cada ejecución se pasa como parámetro el objeto de la clase *GoPiGoContext*, por lo que este objeto es siempre el mismo. Es posible, por tanto, almacenar valores internos en el objeto para tenerlos accesible en cada ejecución. Para poder guardar estos valores el usuario tiene disponible una propiedad definida como ***auxiliar***. Se trata de un *“HashMap”* en la cual el usuario podrá almacenar cualquier valor. Así, por ejemplo, si se desea acceder a un valor almacenado en esta propiedad se puede utilizar el siguiente código:

```
if ‘test_var’ in auxiliar:

        prev_test_var = auxiliar[‘test_var’]
```

O bien si lo que se desea es almacenar un valor en dicha variable se puede utilizar de la siguiente forma:

```
auxiliar[‘test_var’]=test_var
```

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

print(json.dumps(gopigocontext))

4.2. Aplicación Web / EjsS de control del robot GoPiGo 2.

En este apartado se va a proceder a describir como es la implementación de la aplicación Web basada en el framework EjsS (*"Easy Java/Javascript Simulations"*). El apartado está dividido en dos secciones. En la primera se va a proceder a explicar los fundamentos teóricos del framework EjsS, mientras que la segunda se centra en los pormenores del desarrollo realizado.

4.2.1. Framework EjsS – Easy Java/JavaScript Simulations.

EjsS (*"Easy Java/Javascript Simulations"*) es una herramienta de modelado y creación de software de simulación por ordenador en el ámbito académico y ofrece una amplia gama de opciones a la hora de diseñar e implementar estas simulaciones. El software de simulación desarrollado permite obtener datos numéricos de los modelos definidos a medida que avanza el proceso de simulación en el tiempo, y permite también mostrar estos datos mediante gráficas y otros componentes visuales de tal forma que se facilite la comprensión de la información al usuario final. Además el software de simulación que se puede desarrollar usando este framework es fácilmente exportable a código Java o bien a código JavaScript.

EjsS ha sido diseñado para trabajar a un alto nivel conceptual. El objetivo es concentrar la mayor parte del tiempo de trabajo en los aspectos científicos de la simulación a desarrollar, dejando que sea el framework el que se encargue de ejecutar los aspectos repetitivos o automatizables. En la siguiente figura se puede apreciar como es la ventana principal del framework EjsS (una vez que este ha sido iniciado desde línea de comandos).

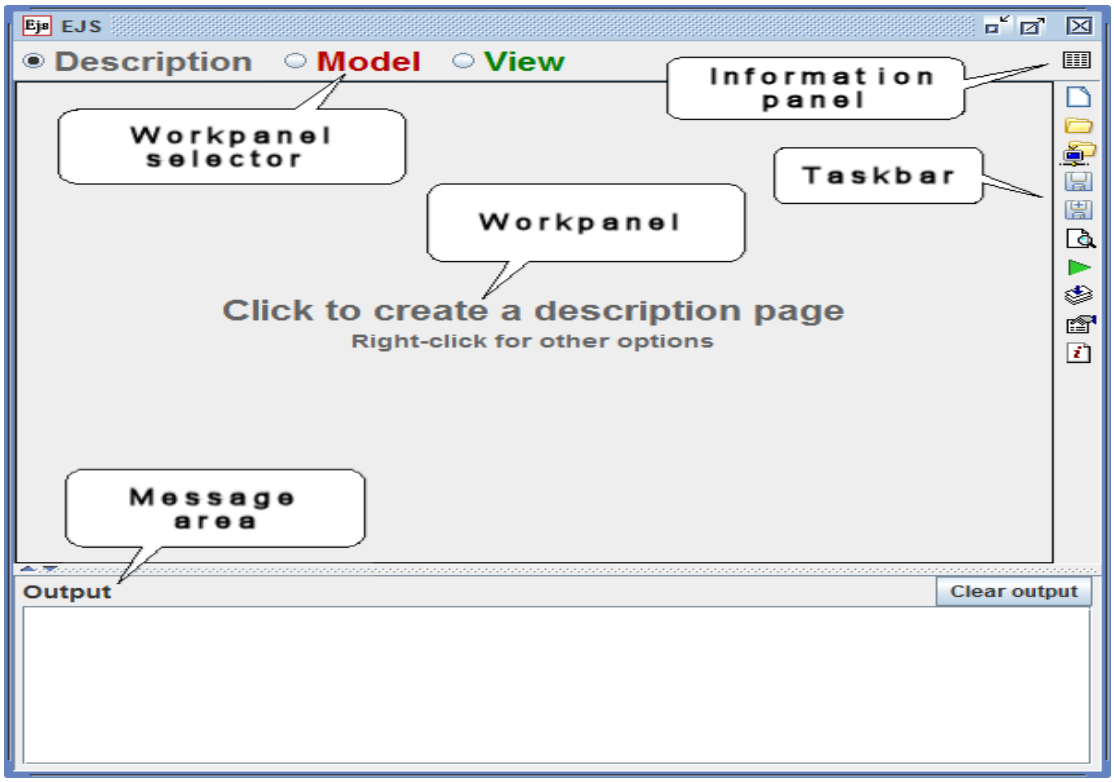


Figura 17. Ventana principal del espacio de trabajo del framework EjsS.

Como se puede apreciar en la figura anterior el interfaz de trabajo es muy sencilla e intuitiva. El interfaz dispone de varias pestañas donde configurar la aplicación a desarrollar (*Descripción*, *Modelo* y *HtmlView*). Dado que la aplicación EjsS desarrollada para este trabajo es una aplicación que debe estar integrada en un servidor para su desarrollo se ha hecho uso de la vista proporcionada por la pestaña *HtmlView*.

A la vista de las opciones ofrecidas en la figura 45 se puede apreciar en la zona de recha la barra de tareas, la cual proporciona una serie de iconos para mostrar un panel de información para la simulación, borrar, abrir y guardar un archivo, buscar el código, configurar EJsS y mostrar la ayuda de EJS. También proporciona iconos para ejecutar una simulación y para empaquetar una o más simulaciones en un archivo *JAR* o exportarlas como applets que serán intradós en páginas web. Al hacer clic con el botón derecho en algunos de los íconos de la barra de tareas, se invocan acciones alternativas (pero relacionadas). La parte inferior de la interfaz contiene un área de salida donde EJS muestra mensajes informativos y logs. La parte central de la interfaz contiene los paneles de trabajo donde se construye la simulación. Como se ha comentado con anterioridad la vista principal del interfaz del framework EJsS proporciona tres paneles de trabajo para crear una simulación:

- El primer panel, *Description* o Descripción, permite crear y editar texto descriptivo en HTML que puede ser utilizado para realizar una descripción de la simulación. Esta información se mostrará al usuario al inicio cuando la simulación se ejecute como una aplicación, o será parte de las páginas HTML que acompañan a la simulación cuando se ejecuta como un *Applet* Java. No es necesario definir la página de descripción empleando para ello las propias herramientas del framework ya que si se desea es posible definir la página HTML de descripción mediante herramientas externas e importar dicha página como página de descripción en el panel *Description*.

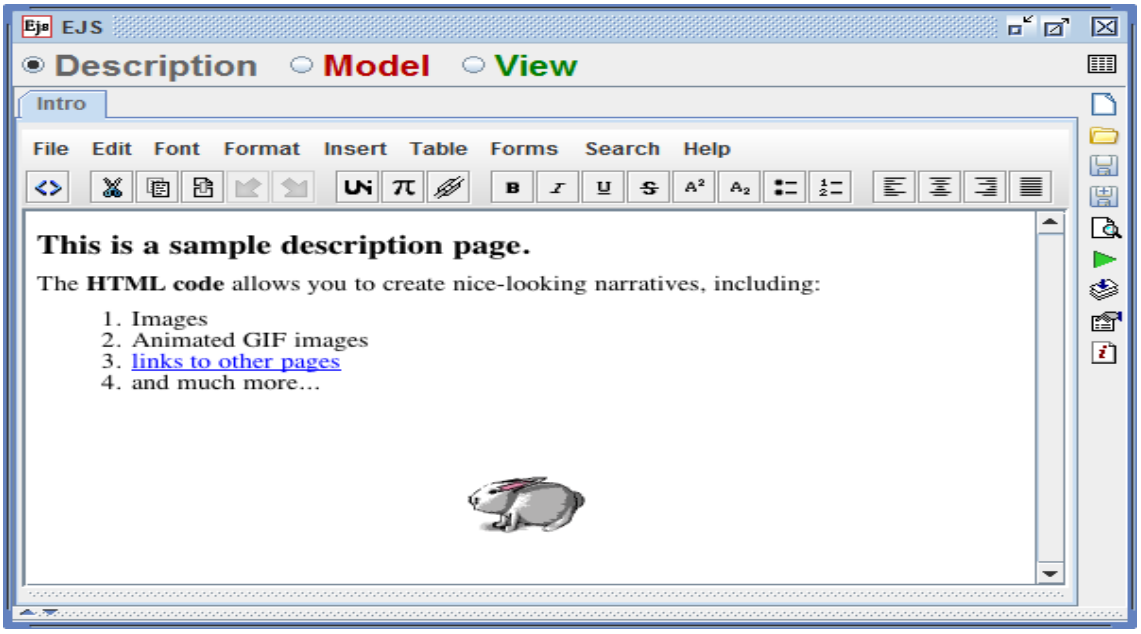


Figura 18. Panel de trabajo "Description" del interfaz del framework EjsS.

- El segundo panel, *Model* o Modelo, está dedicado al proceso de modelado. Este panel se utiliza para crear variables que describan el modelo de la simulación, para inicializar estas variables y para escribir algoritmos que describan cómo el modelo cambia en el tiempo. Dentro del panel *Model*, se encuentran seis sub-paneles: Variables, Inicialización, Evolución, Relaciones fijas, Personalizadas y Elementos.

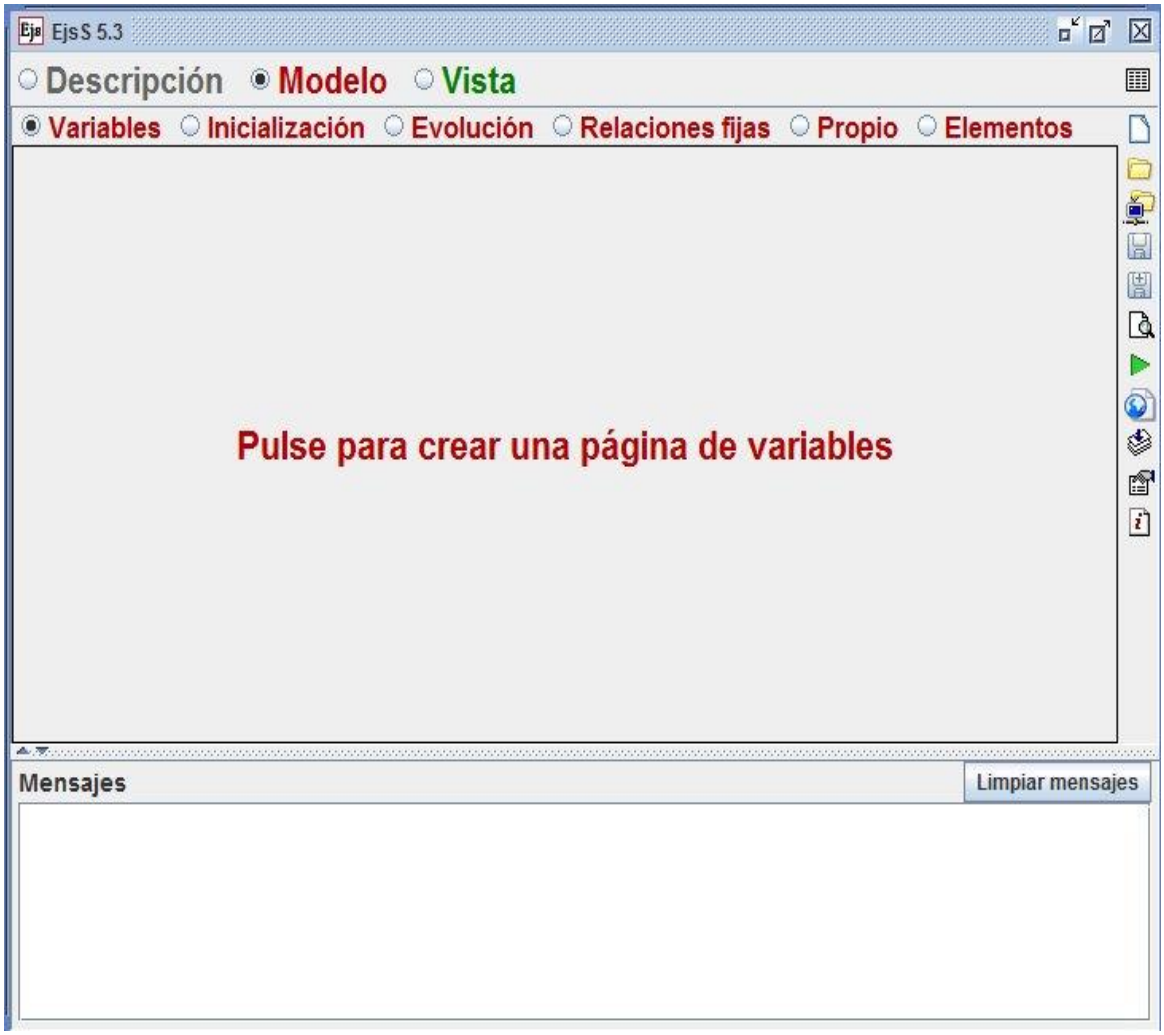


Figura 19. Panel de trabajo "Modelo" del interfaz del framework EjsS.

  - El sub-panel *Variables* se usa para declarar e inicializar las variables del modelo.

  - El sub-panel *Inicialización* del modelo permite describir algoritmos de inicialización adicionales.

  - El sub-panel *Evolución* describe cómo el modelo cambia con en el tiempo según avanza la simulación. Este sub-panel incluye un editor ODE en EJS.

- El sub-panel *Relaciones fijas* establece relaciones fijas entre las variables y cálculos adicionales necesarios para mantener el modelo actualizado.

- Las páginas personalizadas del modelo permiten al autor crear métodos (funciones y subrutinas) para usar en otras partes del modelo y la vista. Estos métodos se agregan a una lista estándar de métodos predefinidos EJS.

- El sub-panel *Elementos* ofrece objetos de modelo (elementos) predefinidos y listos para usar que simplifican el desarrollo algunas tareas avanzadas.

- El último de los paneles principales que forman parte del interfaz del framework EjsS es el panel *"View" o* "Vista".

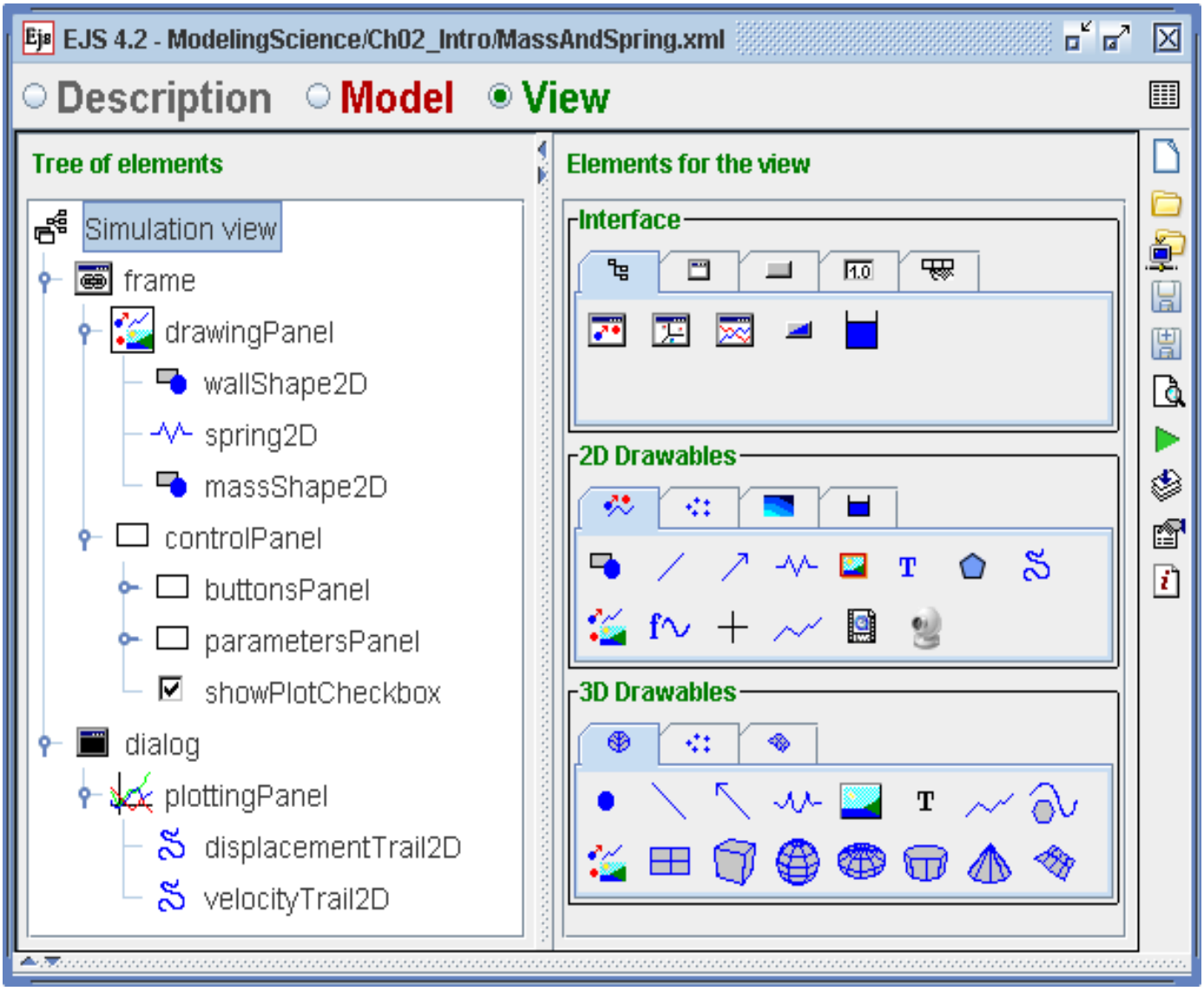


Figura 20. Panel de trabajo "Vista" del interfaz del framework EjsS.

Este panel de trabajo permite al usuario crear una interfaz gráfica que incluye visualización, interacción con el usuario y control de programa mediante un modelo de programación mínima. La forma en que se visualizan los elementos de vista refleja la forma en que se organizan los elementos de la misma en las simulaciones y por defecto tiene una estructura de árbol. Los elementos que van a formar parte de la vista se agregan a la vista activa arrastrando y soltando los iconos sobre la misma. Al hacer clic en un elemento de vista, se selecciona y el cursor se convierte en una "varita mágica". El elemento *view* se arrastra al árbol de elementos y se coloca en un elemento padre apropiado del que se dependiente. En gran medida, el orden en la vista de árbol determina cómo se ubicarán los elementos secundarios dentro de su elemento principal. Esto se complementa mediante la edición de las propiedades de diseño. Las propiedades de diseño guían cómo los elementos secundarios se colocan dentro de sus elementos principales.

Como se ha podido comprobar en los párrafos anteriores la interfaz de desarrollo del framework EjsS es bastante intuitiva de utilizar. En la página web que sirve de *wiki* al framework EjsS (http://www.um.es/fem/EjsWiki/) construir una aplicación de simulación utilizando este framework así como aspectos más avanzados.

Una vez vistos los aspectos fundamentales del framework EjsS y la descripción básica de los componentes básicos de su interfaz de desarrollo, se procede en el siguiente apartado a describir como es la aplicación EjsS desarrollada en el presente trabajo para realizar la gestión y control remoto del robot GoPiGo-2.

4.2.2. Aplicación Web EjsS. Implementación.

En este apartado se va a proceder a presentar la aplicación EjsS desarrollada para llevar a cabo el control remoto del robot GoPiGo 2. La aplicación ha sido diseñada con el objetivo de proveer al usuario final de una interfaz de control fácil de utilizar pero que a la vez dote al usuario de la capacidad de controlar y de gestionar los aspectos más relevantes asociados al robot GoPiGo 2. La aplicación EjsS de control del robot GoPiGo 2 cuenta con un interfaz gráfico que basado en un componente conocido como *"TabbedPane"* en el cual se pueden distribuir diferentes paneles maestros por medio de pestañas. De esta forma en un área reducida se puede disponer de toda la funcionalidad de la aplicación y para acceder a cuadro de mandos o panel concreto basta con cambiar de pestaña.

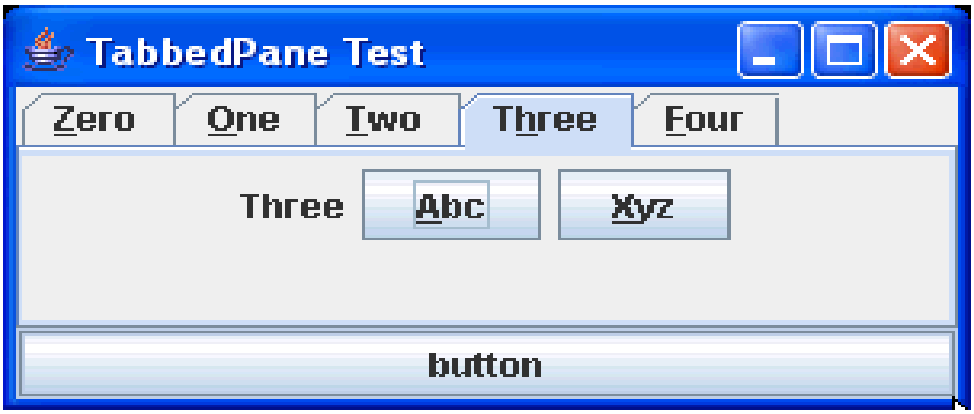


Figura 21. Ejemplo del componente TabbedPane utilizado para el desarrollo de la aplicación EjsS de control del robot GoPiGo 2.

En la Figura 22 se puede apreciar como es la vista general de diseño que se ha definido en el framwork EjsS para la aplicación desarrollada.

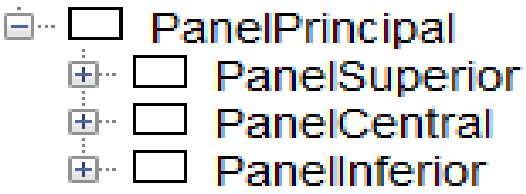


Figura 22. Distribución de la aplicación EjsS de control del robot GoPiGo-2 en componentes.

En concreto, la aplicación EjsS desarrollada está distribuida en tres paneles principales cuyas funciones básicas se describen a continuación:

- **Panel Superior:** este panel está siempre visible al usuario y está compuesto por una serie de sub-paneles que permiten gestionar algunos aspectos relativos a la conexión de la aplicación con el robot, los controles necesarios para permitir al usuario controlar el movimiento del robot en modo manual así como activar el modo de navegación automático del robot en caso de que exista un programa de control cargado en el mismo entre otros.

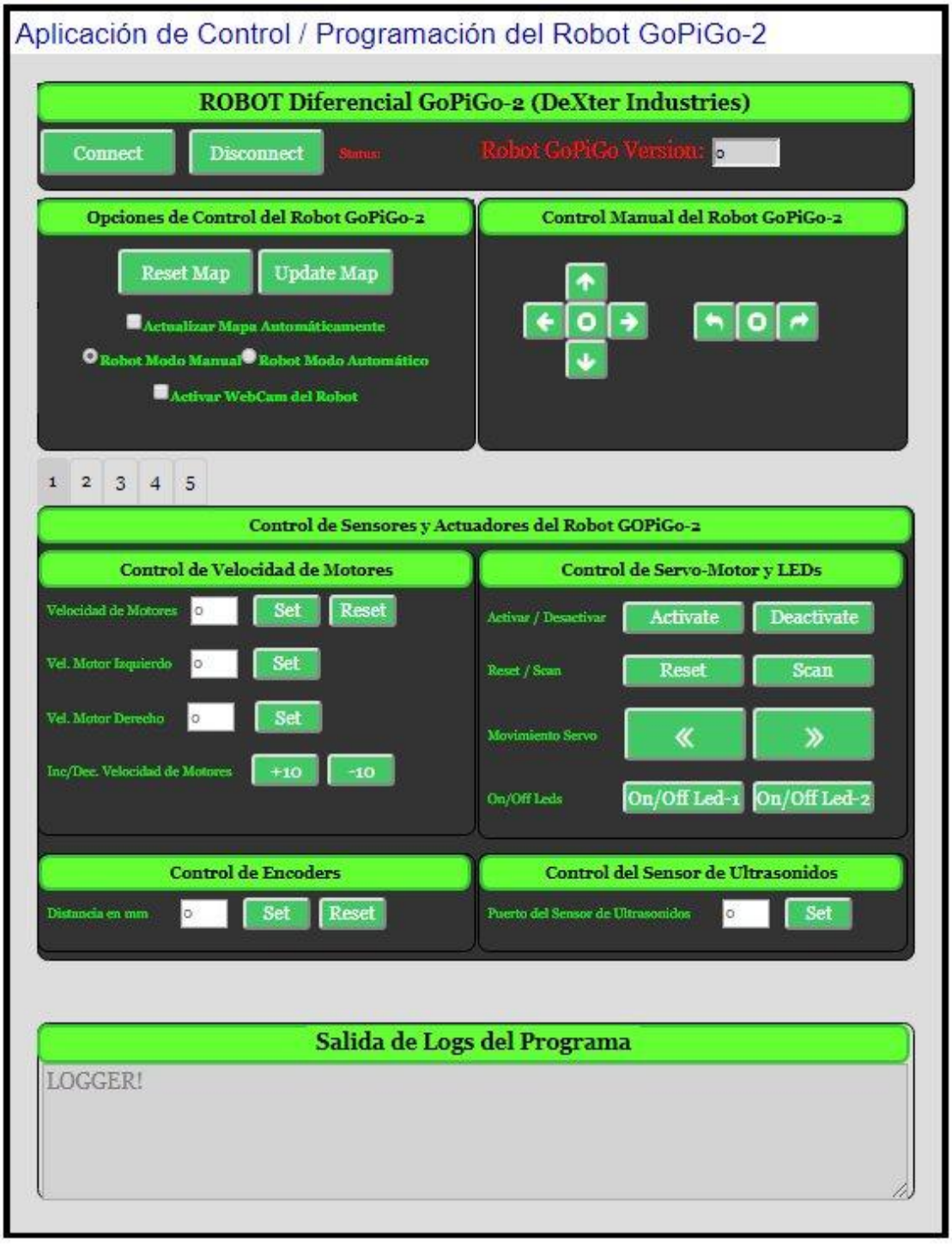


Figura 23. Vista general de la aplicación EjsS de control del robot GoPiGo-2.

- **Panel Central:** este panel incluye un componente del tipo *"TabbedPane"* que permite distribuir en sub-paneles diferentes tipos de controles relativos al manejo del robot. En concreto los sub-paneles disponibles permiten:

    - Configurar el movimiento del motor servo, gestionar y controlar la velocidad de los dos motores de las ruedas del robot, obtener información de los encoders, modificar el puerto del sensor de ultra-sonidos, etc.

    - Obtener mediciones relativas a la velocidad, distancia a obstáculos, estado de la batería del robot, etc.

    - Visualizar la posición del robot así como de los obstáculos en el modo de control automático.

    - Escribir programas de control (editor de textos) para el robot en leguaje de programación Python que luego puede ser enviado al robot para su ejecución.



Figura 24. Vista general del componente TabbedPane incluido en el "Panel Central" con sus diferentes pestañas.

- **Panel Inferior:** este componente incluye un sub-panel en el cual se registran logs de la aplicación en función de las acciones que el usuario realiza sobre el interfaz gráfico.

4.2.2.1. Aplicación Web EjsS. Configuración del componente RIP_Element.

Como se ha visto en secciones anteriores de este trabajo para llevar a cabo la comunicación entre el robot GoPiGo-2 y la aplicación EjsS desarrollada se ha utilizado el protocolo RIP (*"Remote Interoperability Protocol"*). En secciones anteriores se detallaron los pormenores de la implementación del servidor RIP instalado en el robot GoPiGo-2 pero quedaba por explicar la forma en que la aplicación EjsS establecía la conexión con dicho servidor. Para ello el interfaz de desarrollo del framework EjsS provee de un componente denominado *"RIP_Element"* que es el encargado de actuar de cliente y de establecer la comunicación con el servidor RIP así como manejar el intercambio de información con este una vez que la conexión entre ambos elementos ha sido establecida.

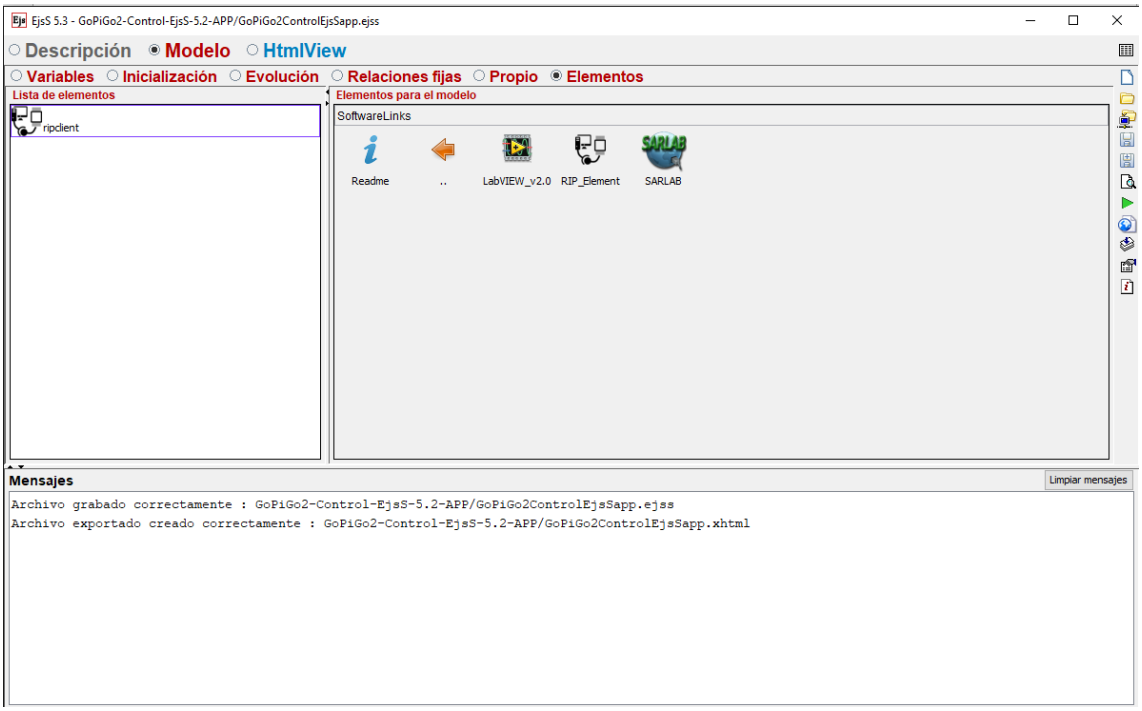


Figura 25. Pestaña "Elementos" del interfaz de desarrollo del framework EjsS donde añadir el componente RIP_element.

Para añadir dicho elemento a la aplicación es necesario buscar en la pestaña *"Elementos"* la sub-opción *"Software Links"* y arrastrar el icono del componente *"RIP_Element"* a la lista de elementos. Al hacer este paso se pide nombrar el nuevo componente siendo el nombre escogido para el caso del desarrollo realizado en este trabajo *"ripclient"*. Mediante este nombre se podrá referenciar el componente *"RIP_element"* para poder intercambiar información con el servidor RIP instalado en el robot GoPiGo-2.

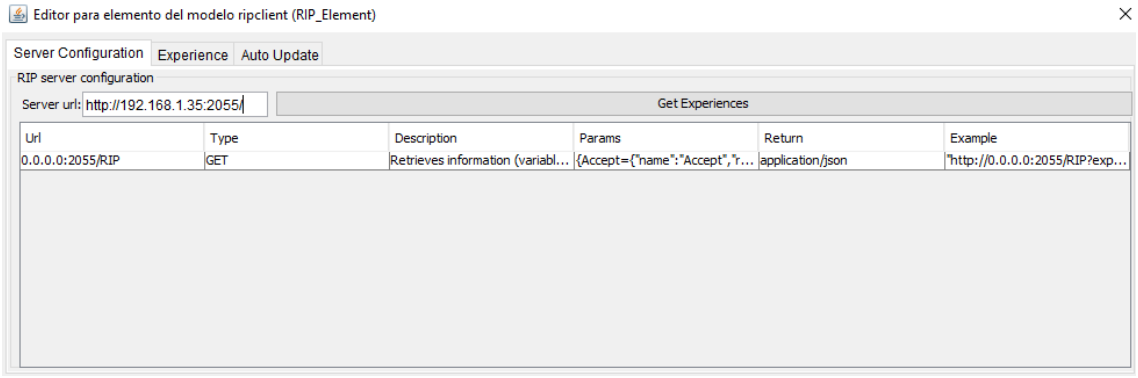


Figura 26. Configuración del componente "RIP_Element" renombrado como "ripclient" en la aplicación EjsS. (I).

Haciendo doble click sobre el icono se abre una ventana donde se puede configurar el componente *"RIP_Element"* con los datos necesarios para establecer la conexión con el servidor RIP. El principal parámetro de configuración para establecer la conexión con el servidor RIP instalado en el robot GoPiGo-2 es *"Server url"*. En el ejemplo de la figura 23 este parámetro toma el siguiente valor: *http://192.168.1.35:2055/*.

Parámetros de Configuración del componente RIP_Element	
Dirección IP	192.168.1.35
Puerto TCP	2055

Tabla 16. Párametros de configuración del componente "RIP_Element".

Al configurar el valor del parámetro *"Server url"* y pulsar sobre el botón *"Get Experiences"* se debe obtener (en caso de que el servidor RIP esté activo) la lista de experiencias activas y configuradas en el servidor RIP del robot GoPiGo-2.

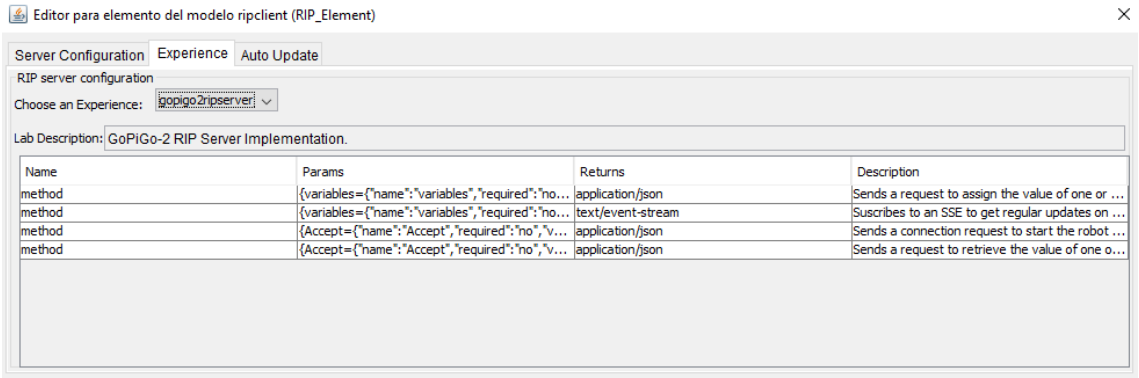


Figura 27. Configuración del componente "RIP_Element" renombrado como "ripclient" en la aplicación EjsS. (II).

Una vez realizado el paso anterior se puede seleccionar la pestaña *"Experience"* para comprobar la lista de experiencias disponibles. En la figura 24 se puede apreciar que la experiencia configurada en el robot GoPiGo-2 es *"gopigo2ripserver"*. Tal y como se puede apreciar en dicha figura esta experiencia expone 3 métodos a través del end-point http://<dir_IP>:2055/RIP/POST (figura 25) y un método adicional a través del end-point http://<dir_IP>:2055/RIP/SSE.

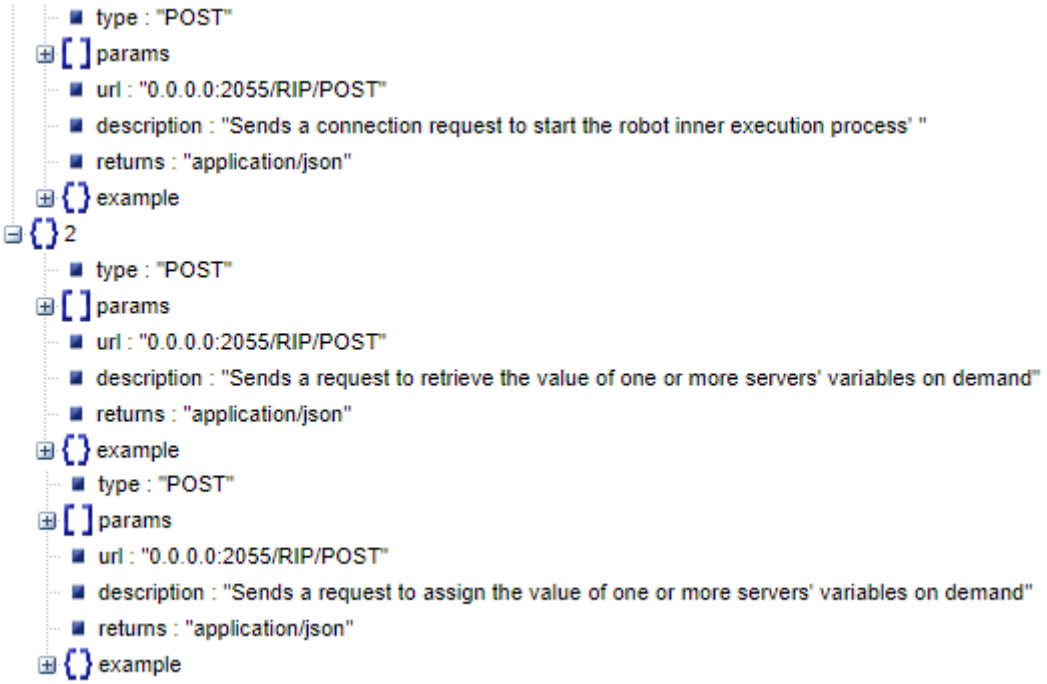


Figura 28. Métodos expuestos a través del end-point http://<dir_IP>:2055/RIP/POST.

Finalmente, la comunicación entre la aplicación web EjsS, a través del elemento RIP, y el servidor RIP se realiza mediante llamadas a los diferentes métodos, los cuales fueron explicados en el apartado referente a la arquitectura del servidor RIP:

- **ripclient.connect()**: realiza una llamada al método *connect* del servidor para iniciar la comunicación con el robot.
- **ripclient.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.
- **ripclient.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.

4.2.2.2. Aplicación Web EjsS. Panel Superior.

Como se ha comentado con anterioridad el *"Panel Superior"* concentra una serie de opciones que siempre están visibles para el usuario. La vista general de este panel se puede apreciar en la Figura 29.

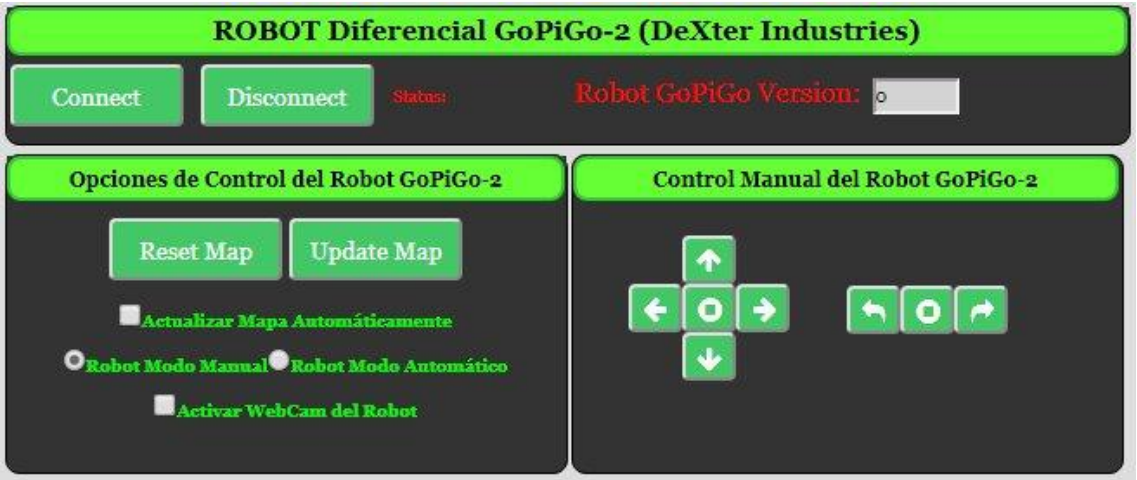


Figura 29. Vista general del componente "Panel Superior".

La distribución de componentes anterior se corresponde con la vista de elementos en el framework EjsS de la Figura 30.

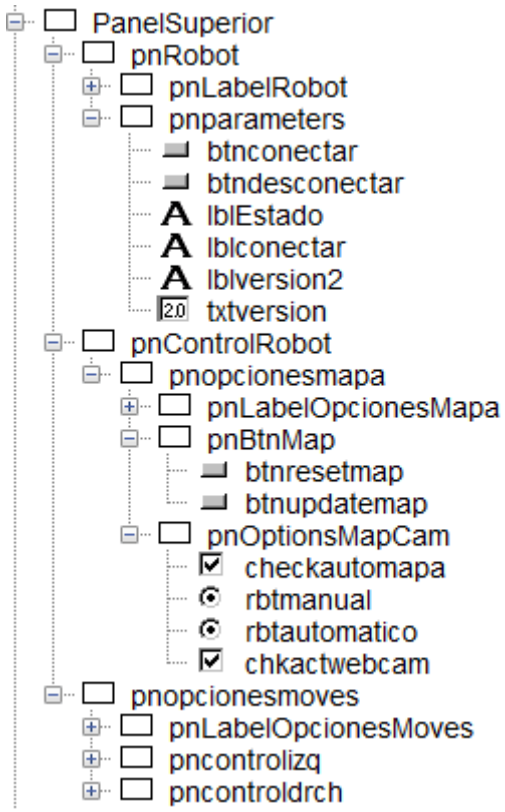


Figura 30. Vista de diseño de elementos del componente "Panel Superior".

A continuación se van a presentar las funcionalidades de los componentes más representativos de cada sub-panel:

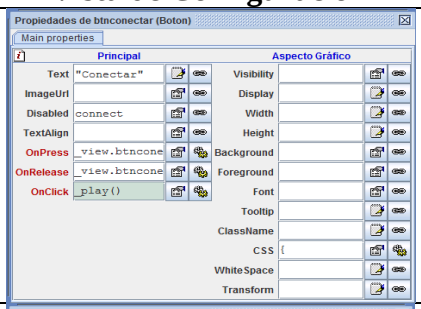
- El primer sub-panel que forma parte del componente *"Panel Superior"* es el sub-panel de *"Estado"* cuyo nombre de componente es *"pnRobot"* dentro de la interfaz de desarrollo del framework EjsS.



Figura 31. Vista del Sub-Panel de Estado (pnRobot).

La misión de este panel es la de proveer de los controles necesarios que permitan gestionar la conexión / desconexión de la aplicación EjsS con el robot GoPiGo-2. De los elementos que componen el sub-panel de estado vistos en la Figura 31 interesa hacer un inciso en los componentes *"btnconectar"* y *"btndesconectar"*. En ambos casos se trata de componentes gráficos asociados a botones cuya función es la de establecer/cerrar la conexión con el robot GoPiGo-2 respectivamente. En la Tabla 17 se incluye la relación existente entre la configuración de los componentes *"btnconectar"* y *"btndesconectar"* y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

btnconectar		_play();
		connect = true;
		updateLogger('Connecting with GoPiGo-2 Robot...');
		updateLogger('Connection stablished...');
		ripclient.get(['connect'], function(result) {
		_view.txtversion.setValue(result[0]);
		ClearObstaclesMap();
		});

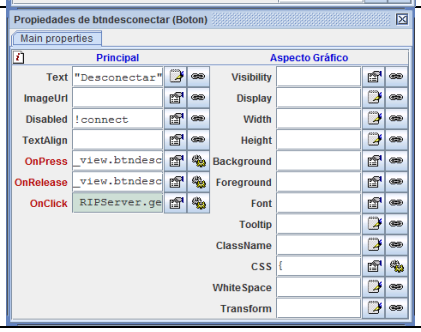
btndesconectar		updateLogger('Disconnecting GoPiGo-2 Robot...');
		updateLogger('Disconnection performed...');
		ripclient.get(['disconnect'], function(result) {
		connect=false;
		_view.txtversion.setValue("0.000");
		//_stop();
		});

Tabla 17. Configuración de los componentes "btnconectar" y "btndesconectar".

- El segundo sub-panel que forma parte del componente *"Panel Superior"* es el sub-panel de *"Control de Opciones de Mapa"* cuyo nombre de componente es *"pnopcionesmapa"* dentro de la interfaz de desarrollo del framework EjsS.



Figura 32. Vista del sub-panel de Control de opciones de mapa "pnopcionesmapa".

La misión de este panel es la de proveer de los controles necesarios que permitan gestionar el cambio entre el modo manual y el modo automático de control del robot así como la actualización del mapa. De los elementos que componen el sub-panel de *"Control de Opciones de Mapa"* vistos en la Figura 30 interesa hacer un inciso en los siguientes componentes: *"btnresetmap"*, *"rbtmanual"* y *"rbtautomatico"*. En el primer caso se trata de un componente gráfico asociado a un elemento de tipo botón mientras que el resto son componentes gráficos del tipo *"radio-button"*. Por su parte los elementos *"checkautomapa"* y *"chactwebcam"* modifican los valores de variables globales de las funciones de control definidas en la pestaña *"Evolución"* de la interfaz de desarrollo. La función de estos componentes es la de modificar el tipo de control que se efectúa sobre el robot: manual o automático. En las siguientes tablas se incluye la relación existente entre la configuración de los componentes *vistos* y las acciones que realizan a través de sus *"handler"* correspondientes.

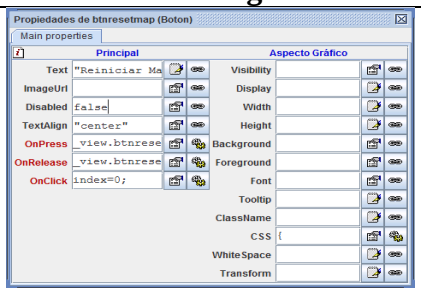
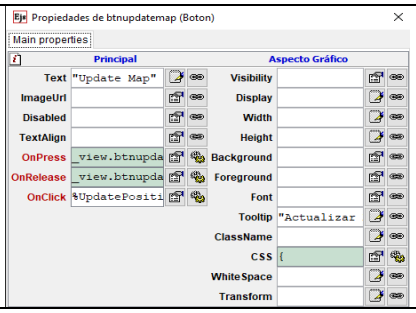
Componente	Vista de Configuración	Código JS / Handler
btnresetmap		function ejsSManualResetObstacleMap() { index = 0; while(index < nPoints){ x[index] = 0; y[index] = 0; index = index + 1; } nPoints = 0; ripclient.set(["setresetmap"],[0]); updateLogger('Reset Obstacle Map...'); }
btnupdatemap		%UpdatePositionRobot()%

Tabla 18. Configuración de los componentes "btnresetmap" y "btnupdatemap".

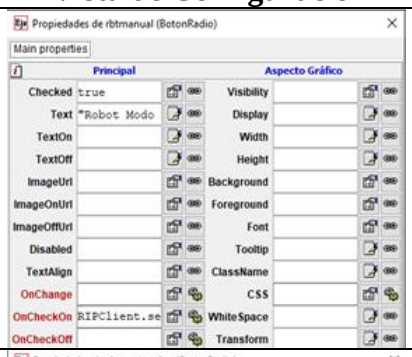
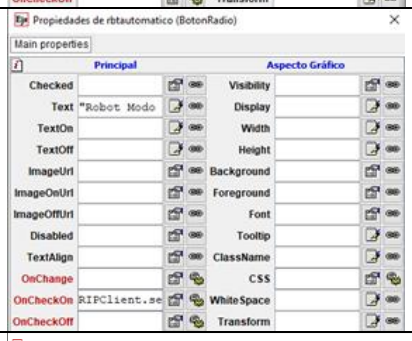
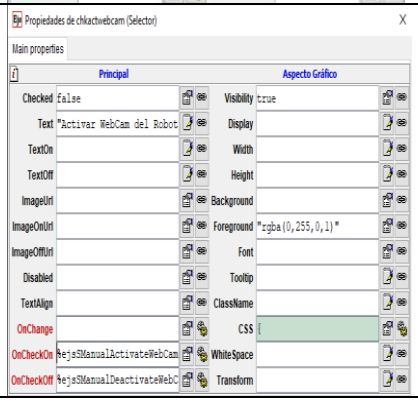
Componente	Vista de Configuración	Código JS / Handler
rbtmanual		function ejsSManualSetManualRobotOperationMode() { ripclient.set(["getmodocontrol"],["manual"]); updateLogger('Robot in Manual Operation Mode...'); }
rbtautomatic		function ejsSManualSetAutomaticRobotOperationMode() { ripclient.set(["getmodocontrol"],["automatico"]); updateLogger('Robot in Automatic Operation Mode...'); }
chkactwebcam		function ejsSManualActivateWebCamStreaming() { activeWebCam = true; updateLogger('WebCam Streaming Activated...'); } ---------- function ejsSManualDeactivateWebCamStreaming() { activeWebCam = false; updateLogger('WebCam Streaming Deactivated...'); }

Tabla 19. Configuración de los componentes "rbtmanual", "rbtautomatico" y "chkactwebcam".

- El último sub-panel que forma parte del componente *"Panel Superior"* es el sub-panel de *"Control de Movimiento"* cuyo nombre de componente es *"pnopcionesmoves"* dentro de la interfaz de desarrollo del framework EjsS.



Figura 33. Vista del sub-panel de Control de Movimiento "pnopcionesmoves".

La misión de este panel es la de proveer de los controles necesarios que permitan realizar el control manual del robot. En la siguiente figura se puede apreciar como es la descomposición de este panel en los elementos gráficos más simples que lo componen dentro de la interfaz de diseño del framework EjsS.

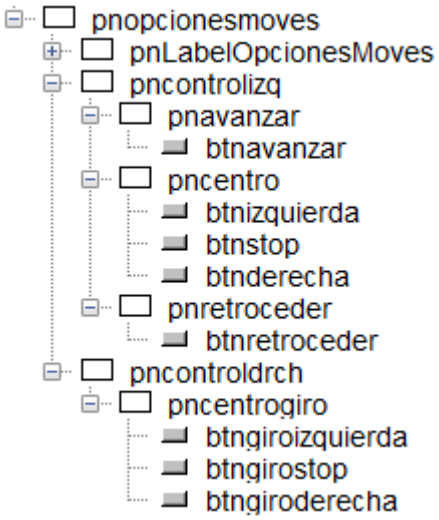
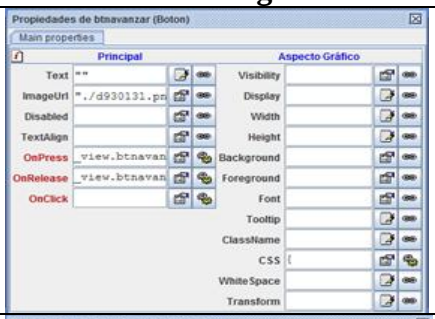


Figura 34. Descomposición del panel de Control de Movimiento (pnopcionesmoves) en elementos gráficos simples

De los elementos que componen el sub-panel de control de movimiento vistos en la Figura 34 interesa hacer un inciso en los siguientes componentes: *"btnavanzar"*, *"btnizquierda"*, *"btnderecha"*, *"btnretroceder"*, *"btnstop"*, *"btngiroizquierda", "btngirostop"* y *"btngiroderecha"*. En todos los casos se trata de componentes gráficos asociados a botones cuya función es la de controlar el robot GoPiGo 2 en modo manual. A continuación se presentan tablas que incluyen la relación existente entre la configuración de los componentes *vistos* y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

btnavanzar		function ejssManualMoveForward() {
		
		ripclient.set(["setmov"],[moveForward]);
		updateLogger('Move GoPiGo-2 Robot forward...');
		}

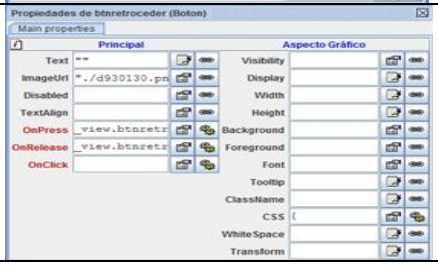
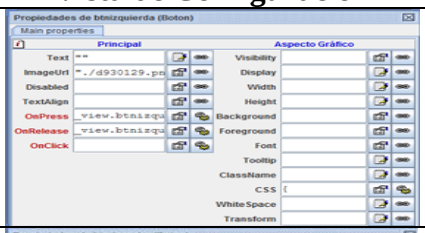
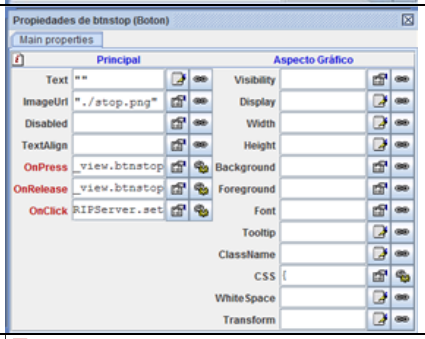
btnretroceder		function ejssManualMoveBackwards() {
		ripclient.set(["setmov"],[moveBackward];
		updateLogger('Move GoPiGo-2 Robot backwards...');
		}

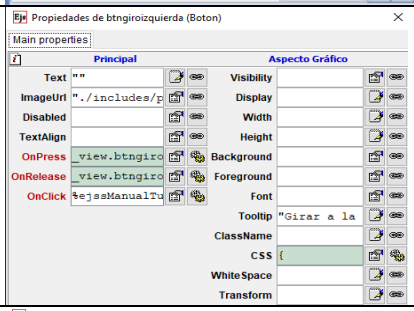
Tabla 20. Configuración de los componentes "btnavanzar" y "btnretroceder".

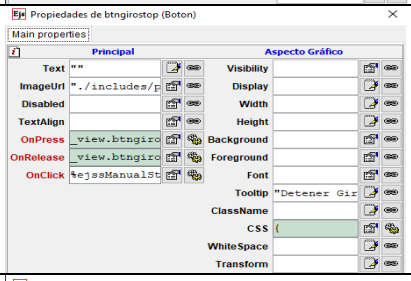
Componente	Vista de Configuración	Código JS / Handler

btnizquierda		function ejssManualMoveToTheLeft() {
		ripclient.set(["setmov"],[moveLeft]);
		updateLogger('Move GoPiGo-2 Robot to the left...');
		}

btnderecha		function ejssManualMoveToTheRight() {
		ripclient.set(["setmov"],[moveRight]);
		updateLogger('Move GoPiGo-2 Robot to the right...');
		}

btnstop		function ejssManualStop() {
		ripclient.set(["setmov"],[99]);
		updateLogger('Stop GoPiGo-2 Robot...');
		}

btngiroizquierda		function ejssManualTurnToTheLeft() {
		ripclient.set(["setmov"],[turnLeft]);
		updateLogger('Turn GoPiGo-2 Robot to the left...');
		}

btngirostop		function ejssManualStop() {
		ripclient.set(["setmov"],[99]);
		updateLogger('Stop GoPiGo-2 Robot...');
		}

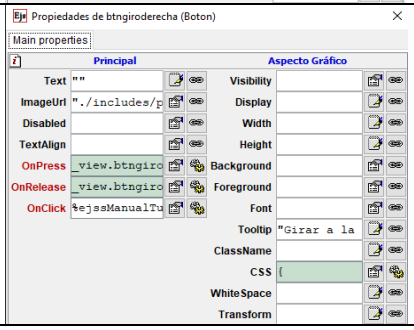
btngiroderecha		function ejssManualTurnToTheRight() {
		ripclient.set(["setmov"],[turnRight]);
		updateLogger('Turn GoPiGo-2 Robot to the right...');
		}

Tabla 21. Configuración de los componentes "btnizquierda", "btnderecha", "btnstop", "btngiroizquierda", "btngirostop" y "btngiroderecha".

4.2.2.3. Aplicación Web EjsS. Panel Central.

Como se ha visto con anterioridad, este panel permite al usuario configurar y gestionar algunos aspectos relativos al robot GoPiGo 2 entre los cuales destacan: gestionar el movimiento del motor servo, gestionar y controlar la velocidad de los dos motores de las ruedas del robot, obtener información de los encoders, modificar el puerto del sensor de ultra-sonidos, obtener mediciones relativas a la velocidad, distancia a obstáculos, estado de la batería del robot, visualizar la posición del robot y codificar programas de control entre otros. En la Figura 35 se puede apreciar la distribución de componentes del *"Panel Central"* que se corresponde con la vista de elementos en el framework EjsS.

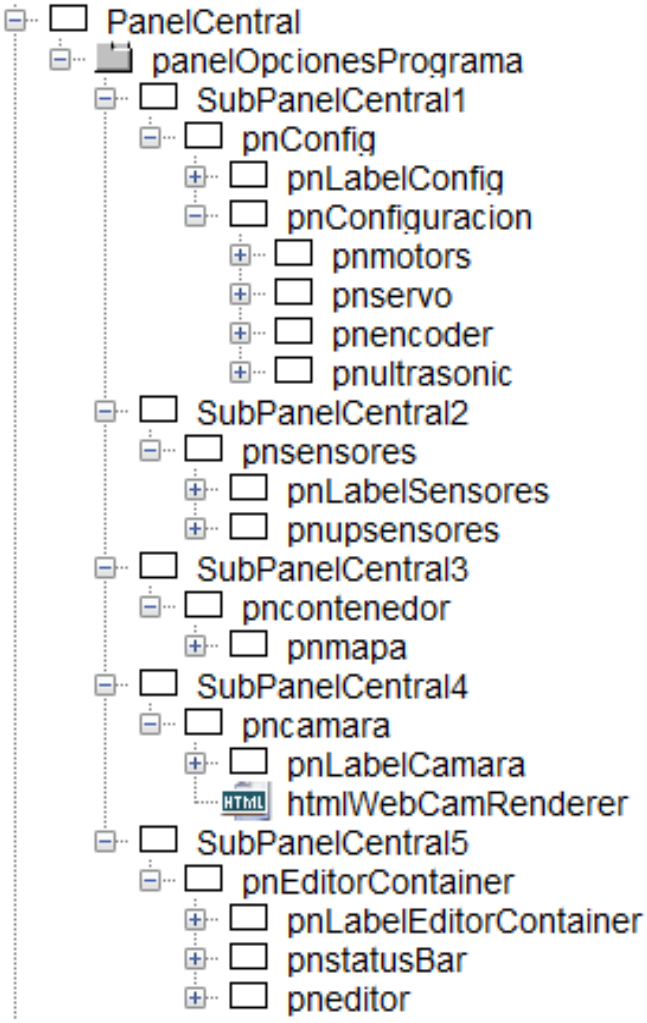


Figura 35. Vista de diseño de elementos del componente "Panel Central".

Como se puede apreciar en la figura anterior el componente *"panelOpcionesPrograma"* contiene la definición de las pestañas principales que contienen los sub-paneles en los cuales se definen los diferentes componentes con los que el usuario puede interactuar. A continuación se van a presentar las funcionalidades de los componentes más representativos de cada sub-panel:

- El componente *"SubPanelCentral1"* incluye todos los elementos que se pueden apreciar en la Figura 36.



Figura 36. Vista de los componentes gráficos que forman parte del elemento "SubPanelCentral1".

En la figura anterior se muestra el componente *"SubPanelCentral1"* que contiene los siguientes elementos:

- **Panel *"pnmotors"***: el nombre de este componente es *"Control de Velocidad de Motores"* y tiene por misión la de proveer de los controles necesarios que permitan gestionar las velocidades del motor izquierdo y del motor derecho del robot GoPiGo-2. En las siguientes tablas se incluye la relación existente entre la configuración de los componentes *vistos* y las acciones que realizan a través de sus *"handler"* correspondientes.

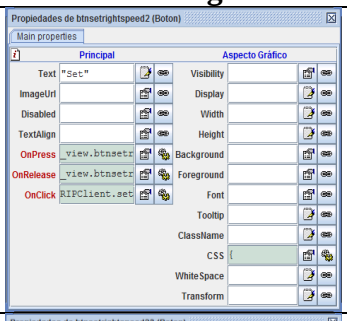
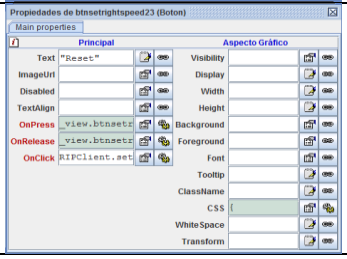
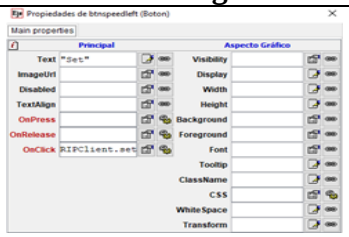
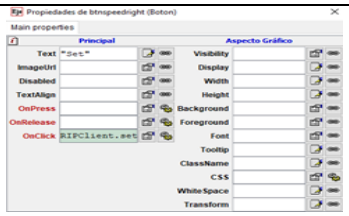
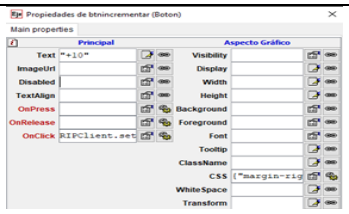
Componente	Vista de Configuración	Código JS / Handler
btnsetrightspeed2 "Velocidad Motores" (Set)		function ejsSManualSetMotorsSpeed() { ripclient.set(["setspeed"],[parseInt(_view.txtmotorspeed.getValue(),10)]); _view.txtmotorspeed.setValue(parseInt(_view.txtmotorspeed.getValue(),10)); updateLogger('Set Motors Speed...'); }
btnsetrightspeed23 "Velocidad Motores" (Reset)		function ejsSManualResetMotorsSpeed() { ripclient.set(["setspeed"],[200]); updateLogger('Reset Motors Speed...'); }

Tabla 22. Configuración de los componentes "btnsetrightspeed2" y "btnsetrightspeed23".

Componente	Vista de Configuración	Código JS / Handler

btnspeedleft
“Vel. Motor Izquierdo”
(Set)
		function ejsSManualSetLeftMotorSpeed() {
		ripclient.set(["setleftspeed"],[parseInt(_view.txtspeedleft.getValue(),10)]);
		_view.txtspeedleft.setValue(parseInt(_view.txtspeedleft.getValue(),10));
		updateLogger('Set Left Motor Speed...');
		}

btnspeedright
“Vel. Motor Derecho”
(Set)
		function ejsSManualSetRightMotorSpeed() {
		ripclient.set(["setrightspeed"],[parseInt(_view.txtspeedright.getValue(),10)]);
		_view.txtspeedright.setValue(parseInt(_view.txtspeedright.getValue(),10));
		updateLogger('Set Right Motor Speed...');
		}

btnincrementar
“Inc./Dec. Velocidad Motores”
(+10)
		function ejsSManualIncreaseMotorsSpeedInTenUnits() {
		ripclient.set(["setincreasespeed"],[10]);
		updateLogger('Increase Motors Speed in 10 units...');
		}

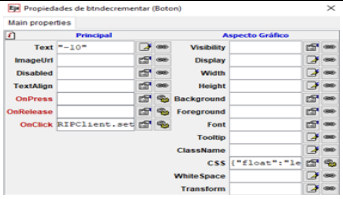
btndecrementar
“Inc./Dec. Velocidad Motores”
(-10)
		function ejsSManualDecreaseMotorsSpeedInTenUnits() {
		ripclient.set(["setdecreasespeed"],[10]);
		updateLogger('Decrease Motors Speed in 10 units...');
		}

Tabla 23. Configuración de los componentes “btnspeedleft”, “btnspeedright”, “btnincrementar” y “btndecrementar”.

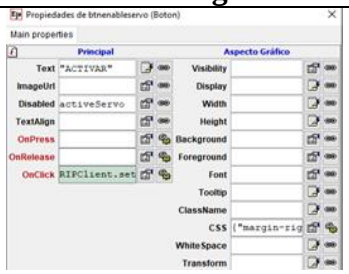
- **Panel *“pnservo”***: el nombre de este componente es *“Control de Servo y Leds”*. La misión de este panel es la de proveer de los controles que permitan gestionar el movimiento del servo-motor que incorpora el robot GoPiGo-2 así como controlar el encendido y apagado de los diodos LED incorporados en el frontal del robot.

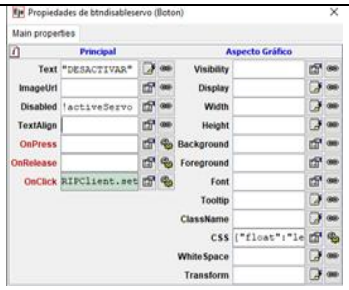


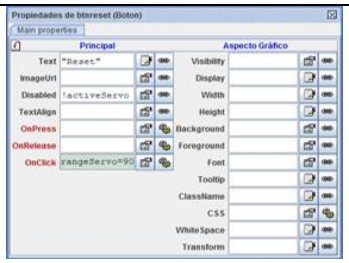
Figura 37. Vista de los componentes gráficos que forman parte del elemento “pnservo”.

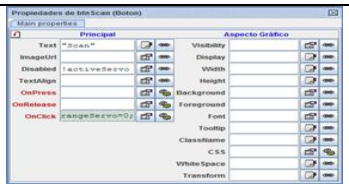
En las siguientes tablas se puede apreciar la relación existente entre la configuración de los componentes vistos en la Figura 37 y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

btnenableservo		function ejsSManualActivateServoMotor() {
		ripclient.set(["setservoEnabled","setservo"],[1,0]
		);activeServo = true;
		updateLogger('Activate Servo-Motor...');
		}

btndisableservo		function ejsSManualDeactivateServoMotor() {
		ripclient.set(["setservoEnabled"],[0]);
		activeServo = false;
		updateLogger('Deactivate Servo-Motor...');
		}

btnreset		function ejsSManualResetServoMotor() {
		rangeServo = 0;
		ripclient.set(["setservo"],[rangeServo]);
		updateLogger('Reset Servo-Motor...');
		}

btnScan		function ejsSManualScanServoMotor() {
		rangeServo = 0;
		do{
		ripclient.set(["setservo"],[rangeServo]);
		(result[0]);}(result));
		rangeServo = rangeServo + 10;
		}while(rangeServo <= 180);
		updateLogger('Scan Servo-Motor...');
		}

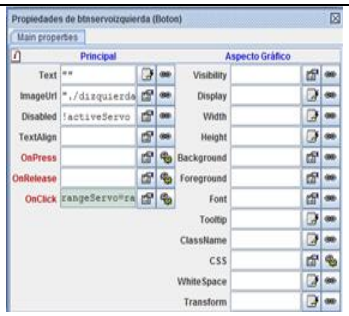
btnservoizquierda		function
		ejsSManualMoveServoMotorToTheLeft() {
		rangeServo = rangeServo + 10;
		if (rangeServo < 0) rangeServo = 0;
		ripclient.set(["setservo"],[rangeServo]);
		updateLogger('Move Servo-Motor to the Left...');
		}

Tabla 24. Configuración de los componentes "btnenableserbo", "btndisableservo", "btnreset", "btnscan" y "btnservoizquierda".

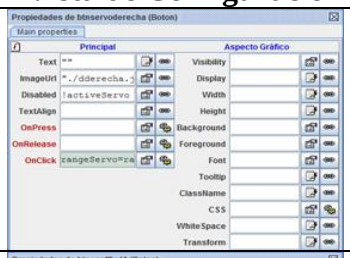
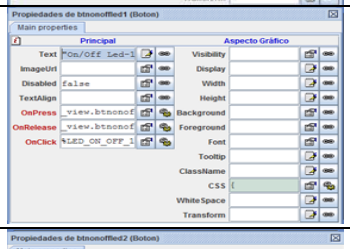
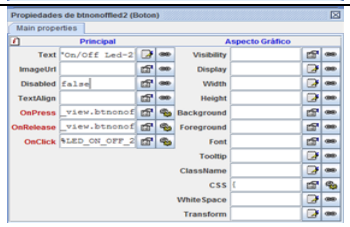
Componente	Vista de Configuración	Código JS / Handler
btnservoderecha		function ejsSManualMoveServoMotorToTheRight() { rangeServo = rangeServo - 10; if (rangeServo < 0) rangeServo = 0; ripclient.set(["setservo"],[rangeServo]); updateLogger('Move Servo-Motor to the Right...'); }
btnonoffled1		function ejsSManualTurnOnOffLed1() { if (led1Status === 0) {led1Status = 1; ripclient.set(["setledon"],[0]); updateLogger('Turn On LED-1...'); } else {led1Status = 0; ripclient.set(["setledoff"],[0]); updateLogger('Turn Off LED-1...'); } }
btnonoffled2		function ejsSManualTurnOnOffLed2() { if (led2Status === 0) {led2Status = 1; ripclient.set(["setledon"],[1]); updateLogger('Turn On LED-2...'); } else {led2Status = 0; ripclient.set(["setledoff"],[1]); updateLogger('Turn Off LED-2...'); } }

Tabla 25. Configuración de los componentes "btnservoderecha" , "btnonoffled1" y "btnonoffled2".

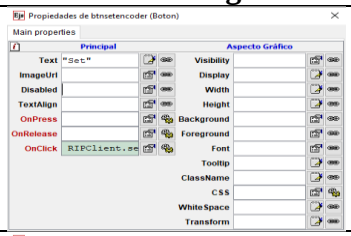
- **Panel *"pnencoder"***: el nombre de este componente es *"Control de Encoders"*. La función de este panel es la de proveer de los controles necesarios que permitan gestionar los valores de los encoders en el robot GoPiGo-2.



Figura 38. Vista de los componentes gráficos que forman parte del elemento "pnservo".

Como se puede apreciar en la Figura 38 este panel dispone una caja de texto en la cual el usuario puede introducir la distancia en milímetros que se desea configurar en los encoders y que una vez fijada delimita la distancia que el robot puede recorrer en cualquier dirección. A continuación se incluye la tabla que relaciona la configuración de los componentes vistos en la Figura 38 y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

btnsetencoder		function ejsSManualSetEncoders() {
		ripclient.set(["setm1"],[encoder1]);
		ripclient.set(["setm2"],[encoder2]);
		ripclient.set(["setenctgt"],[parseInt(_view.txtencoder.getValue(),10)]);
		_view.txtencoder.setValue(parseInt(_view.txtencoder.getValue(),10));
		updateLogger('Set Encoders...');
		}

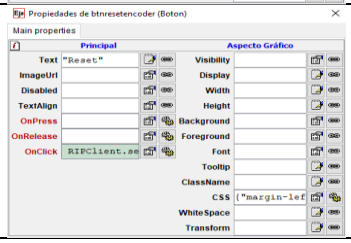
btnresetencoder		function ejsSManualResetEncoders() {
		ripclient.set(["setm1"],[encoder1]);
		ripclient.set(["setm2"],[encoder2]);
		ripclient.set(["setenctgt"],[0]);
		updateLogger('Reset Encoders...');
		}

Tabla 26. Configuración de los componentes "btnsetencoder" y "btnresetencoder".

- **Panel *"pnultrasonic"***: el nombre de este componente es *""Control del Sensor de Ultrasonidos"*. La función principal de este panel es la de ofrecer los controles necesarios que permitan gestionar el valor del puerto de comunicaciones con el sensor de ultrasonidos del robot GoPiGo-2.



Figura 39. Vista de los componentes gráficos que forman parte del elemento "pnultrasonic".

Como se puede apreciar en la Figura 39 este panel dispone una caja de texto en la cual el usuario puede el valor numérico asociado al puerto de ultrasonidos. A continuación se incluye la tabla que relaciona la configuración de los componentes vistos en la Figura 39 y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

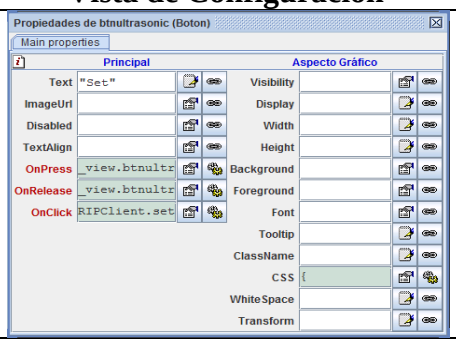
btnultrasonic		function ejsSManualSetUltraSoundSensorPort() {
		ripclient.set(["setportultra"],[parseInt(_view.txtultrasonic.getValue(),10)]);
		_view.txtultrasonic.setValue(parseInt(_view.txtultrasonic.getValue(),10));
		updateLogger('Set UltraSound Sensor Port...');
		}

Tabla 27. Configuración del componente "btnultrasonic".

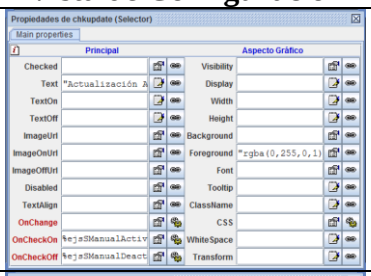
- El componente *"SubPanelCentral2"* incluye todos los elementos que se pueden apreciar en la Figura 40.



Figura 40. Vista de los componentes gráficos que forman parte del elemento "SubPanelCentral2".

Este componente lleva por nombre *"Actualización de Medidas Obtenidas de los Sensores del Robot GoPiGo-2"* e incluye una serie de componentes gráficos cuya misión es la de proveer de los controles necesarios que permitan visualizar los valores de los diferentes sensores conectados al robot GoPiGo-2 así como actualizar su estado. Como se puede apreciar en la figura anterior este componente incluye dos elementos con los que el usuario puede interactuar. El primero es un elemento de tipo *"checkbox"* que permite al usuario activar la actualización automática de las medidas cuando está activo y el segundo es un elemento de tipo *"button"* que permite al usuario activar la actualización de las medidas procedentes del robot cuando es pulsado. A continuación se incluye la tabla que relaciona la configuración de los componentes vistos en la Figura 40 y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

chkupdate		function ejsSManualActivateAutomaticSensorUpdate() {
		updateautomatic = true;
		updateLogger('Automatic Update Activated...');
		}
		--- 	
		function ejsSManualDeactivateAutomaticSensorUpdate() {
		updateautomatic = false;
		updateLogger('Automatic Update Deactivated...');
		}

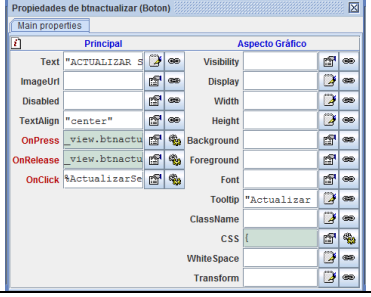
btnactualizar		function UpdateSensor() {
		if (updateautomatic) {
		ActualizarSensores();
		}
		}

Tabla 28. Configuración de los componentes "chkupdate" y "btnactualizar".

- El componente *"SubPanelCentral3"* incluye todos los elementos que se pueden apreciar en la Figura 41, siendo el principal de ellos el panel *"pnmapa"*.

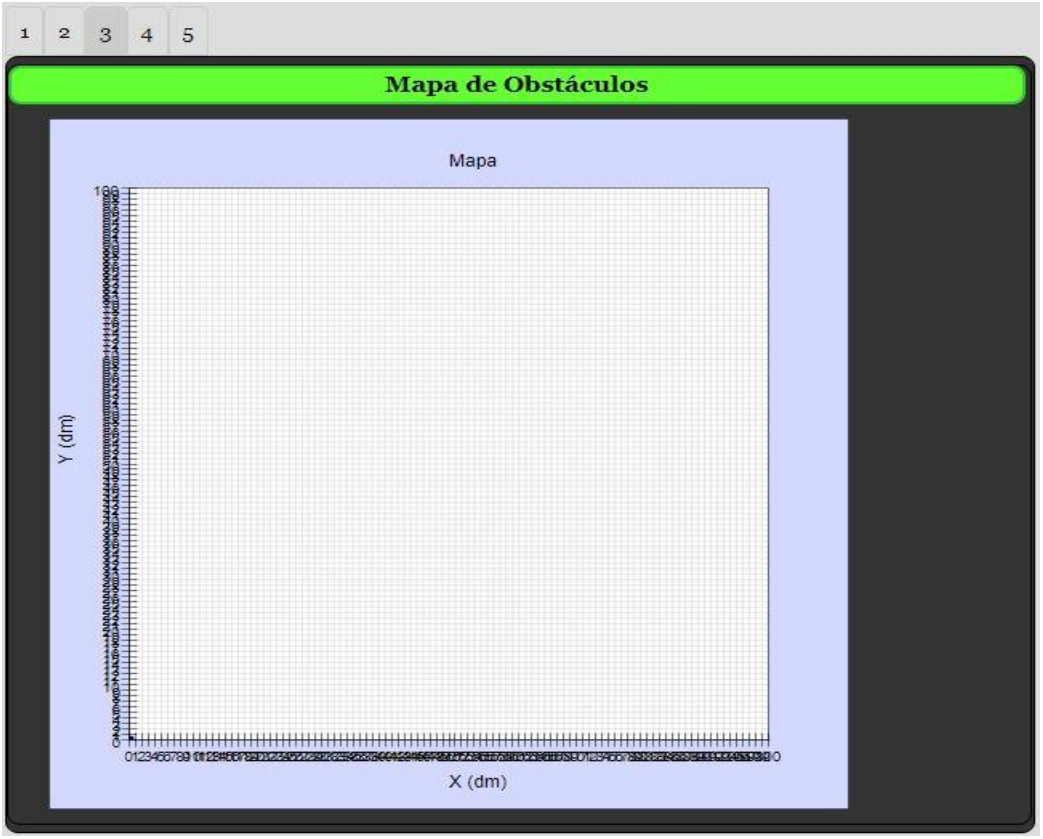


Figura 41. Vista de los componentes gráficos que forman parte del elemento "SubPanelCentral3".

Como se puede apreciar en la figura anterior este componente sólo incluye el mapa de obstáculos cuyo objetivo es el de ofrecer los elementos de visualización necesarios que permitan obtener en el mapa de ocupación la posición actualizada del robot y de los obstáculos detectados por medio del sensor de ultrasonidos instalado en el robot GoPiGo-2 cuando se ha cargado en el robot un script Python con la clase *GoPiGoContext* y este se ha configurado en modo de operación automático. En la Figura 42 se puede apreciar como es la descomposición de este panel en los elementos gráficos más simples que lo componen dentro de la interfaz de diseño del framework EjsS.

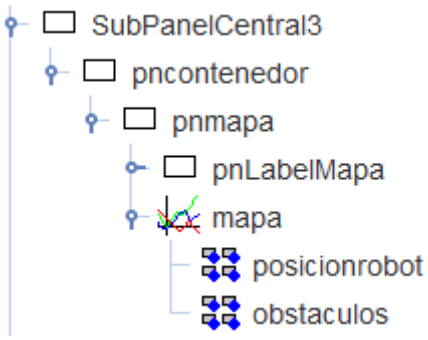


Figura 42. Descomposición del panel de Visualización de Mapa de Ocupación (pnmapa) en elementos gráficos simples.

De los elementos que componen el sub-panel de control de movimiento vistos en la Figura 42 interesa hacer un inciso en los siguientes componentes: *"posiciónRobot"* y *"obstáculos"*. En todos los casos se trata de componentes gráficos asociados a formas geométricas cuya función es la mostrar en el mapa de ocupación los puntos que representan los obstáculos y la posición del robot GoPiGo-2. En la Figura 43 se puede apreciar como es la configuración de los componentes *"posiciónRobot"* y *"obstáculos"* que se corresponden con los botones de flecha izquierda y flecha derecha del sub-panel *"pnmapa"*.

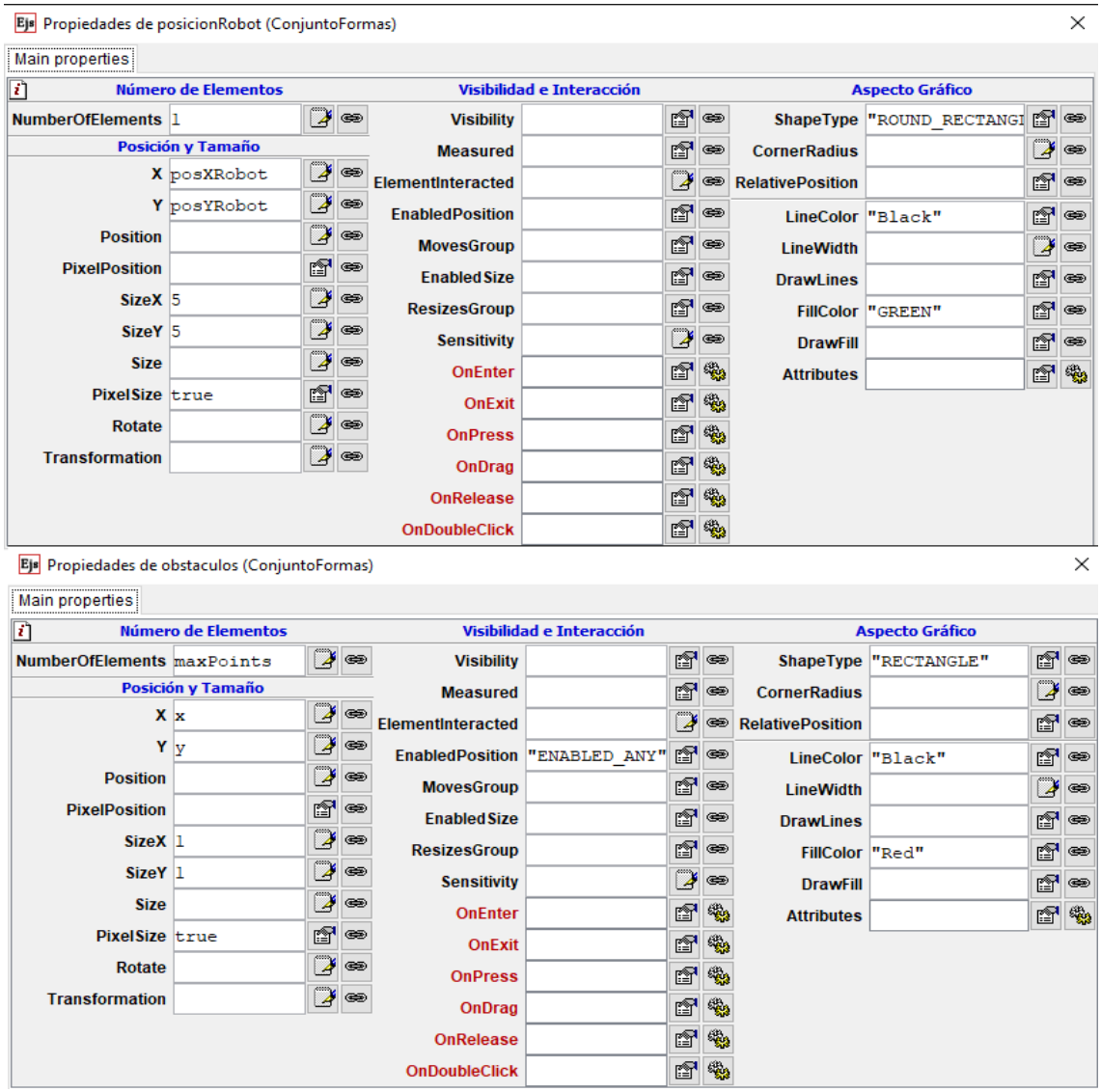


Figura 43. Configuración de los componentes "posicionRobot" y "obstaculos".

Como se puede apreciar en la Figura 43 ambos componentes dependen de variables globales definidas en la pestaña *"Variables"* en la interfaz de desarrollo del framework EjsS.

- El componente *"SubPanelCentral4"* incluye el panel *"pncamara"*.

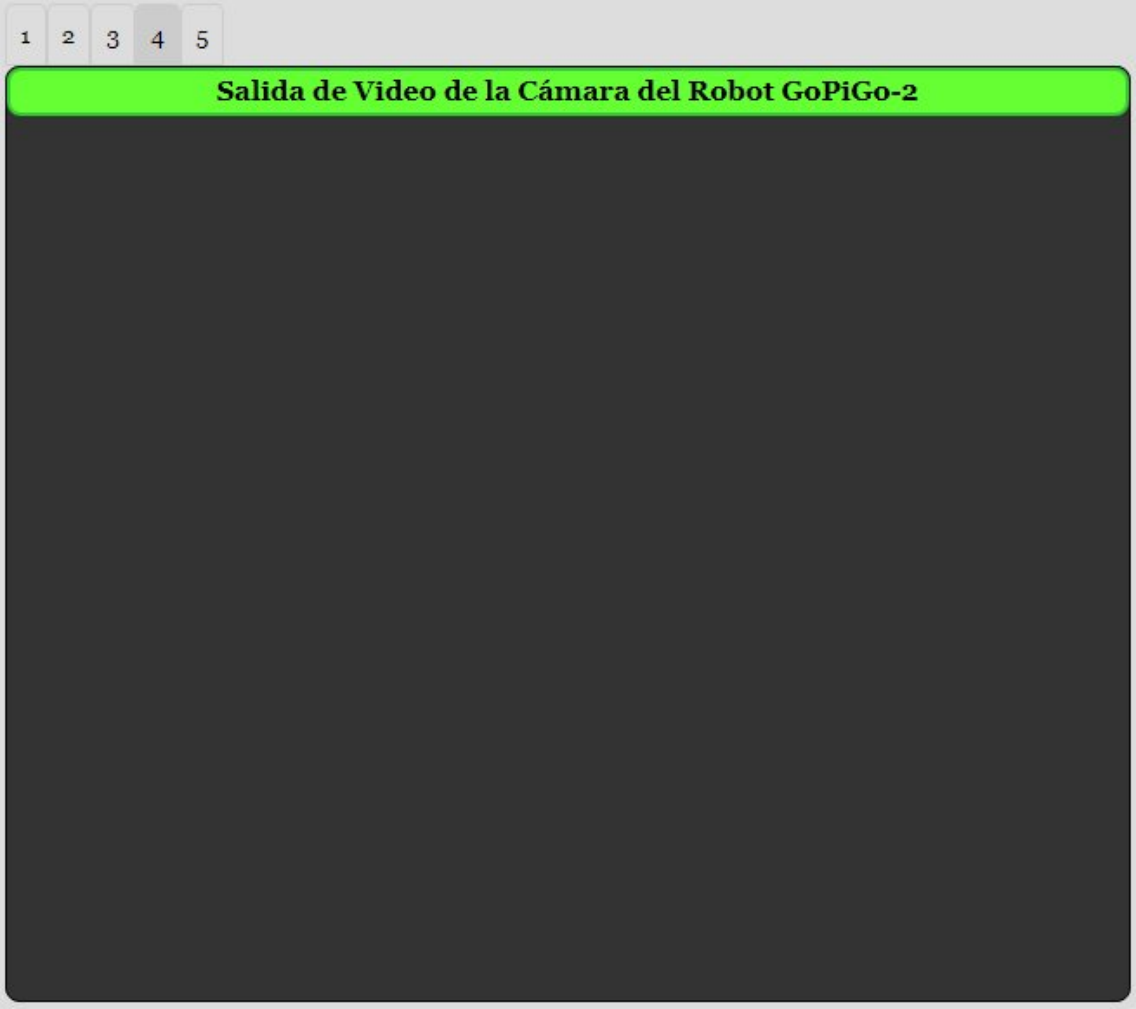


Figura 44. Vista de los componentes gráficos que forman parte del elemento "SubPanelCentral4".

La misión de este panel es la de proveer de los elementos de visualización que permitan recibir y visualizar un flujo de video obtenido mediante una webcam conectada al robot GoPiGo-2. En la Figura 45 se puede apreciar como es la descomposición de este panel en los elementos gráficos más simples que lo componen dentro de la interfaz de diseño del framework EjsS.

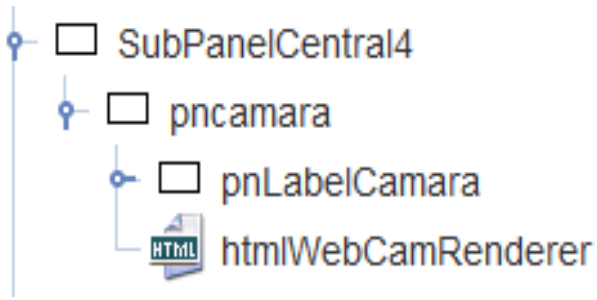


Figura 45. Descomposición del panel de Visualización de WebCam (pncamara) en elementos gráficos simples.

De los elementos que componen el sub-panel visto en la Figura 45 interesa hacer un inciso en el componente *"htmlWebCamRenderer"*. Se trata en este caso de un componente gráfico que permite recibir y visualizar un flujo de video procedente de una webcam conectada al robot GoPiGo-2. En la Figura 46 se puede apreciar como es la configuración del componente *"htmlWebCamRenderer"* del sub-panel *"pncamara"*.

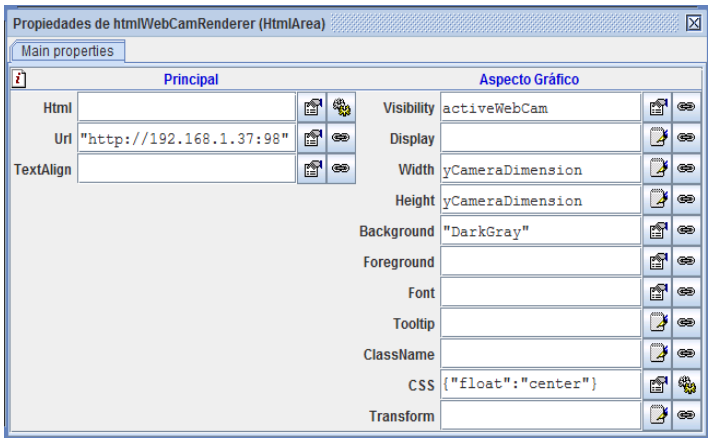


Figura 46. Configuración del componente "htmlWebCamRenderer".

Como se puede apreciar en la figura anterior este componente recebe como parámetro principal la url a través de la cual la cámara está ofreciendo el flujo de video. En este caso el servidor de streaming instalado en el robot ofrece en dicha url una página HTML que es la que se visualiza en el componente *"htmlWebCamRenderer"* (Figura 47).



Figura 47. Recepción del flujo de video en el componente "htmlWebCamRenderer" de la cámara del robot GoPiGo-2.

- El componente *"SubPanelCentral5"* incluye todos los elementos que se pueden apreciar en la Figura 48, siendo el principal de ellos el panel *"pneditorcontainer"*.

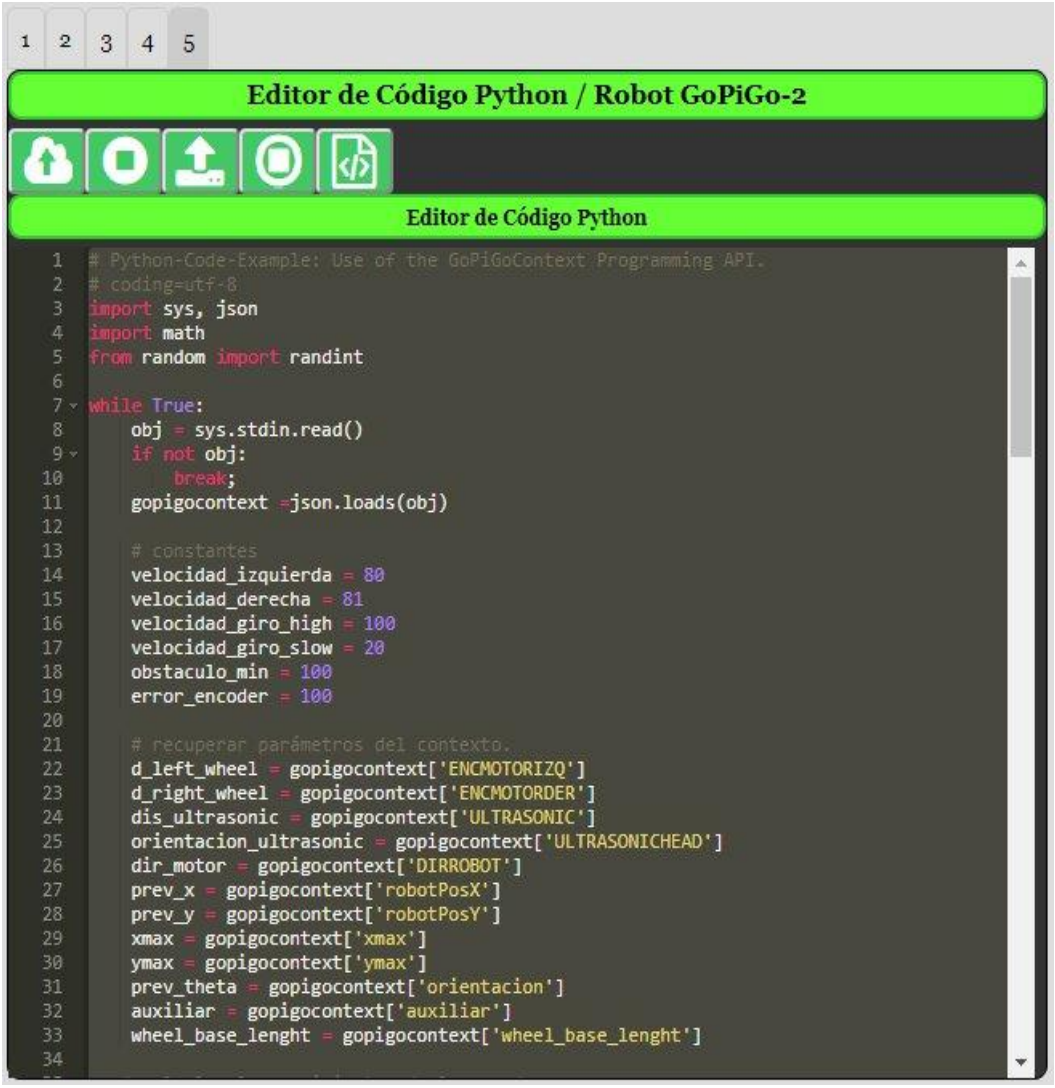


*Figura 48. Vista de los componentes gráficos que forman parte del elemento "*SubPanelCentral5*".*

Como se puede apreciar en la figura anterior en este componente se incluyen todos los elementos que permiten al usuario desarrollar código Python para controlar el robot GoPiGo-2 en modo automático. A parte del área de texto se incluye una barra de botones que permiten desarrollar diferentes acciones como: subir script de código Python, detener ejecución de script, subir programa de control en código Python, detener ejecución de programa y limpiar el área de texto. La diferencia entre subir un script Python o un programa de control Python radica en que en el primer caso se hace uso de la clase *GoPiGoContext* para acceder a las variables del robot en el servidor, mientras que en el segundo caso se envía al robot código Python que hace uso directamente del API nativo de programación del robot. En la Figura 49 se puede apreciar como es la descomposición de este panel en los elementos gráficos más simples que lo componen dentro de la interfaz de diseño del framework EjsS.

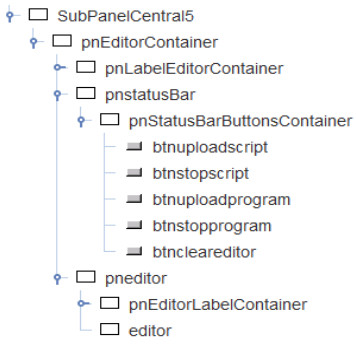
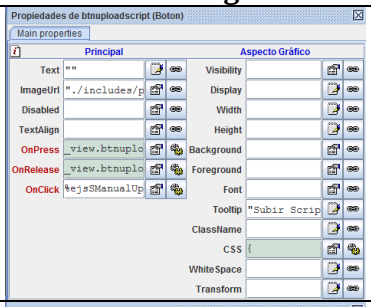
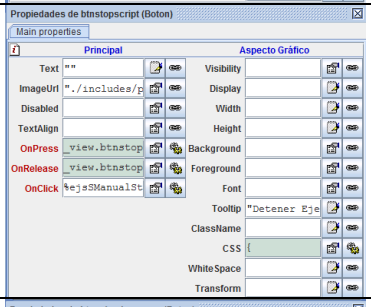


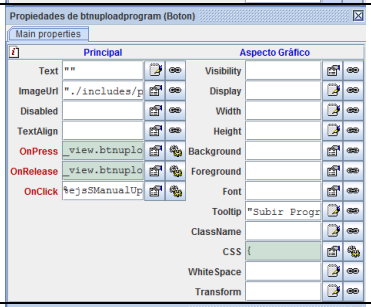
Figura 49. Descomposición del componente "SubPanelCentral5" en elementos gráficos simples.

A continuación se incluyen las tablas que relacionan la configuración de los componentes vistos en la Figura 49 y las acciones que realizan a través de sus *"handler"* correspondientes.

Componente	Vista de Configuración	Código JS / Handler

btnuploadscript		function ejsSManualUploadPythonScriptToGoPiGo2Robot(){
		if (document.getElementById('editor') !== null) {
		ripclient.set(["setscript"],[scriptData]);
		updateLogger('Upload Python Script Code...');
		updateLogger('SENT CODE: ' + scriptData);}
		}

btnstopscript		function ejsSManualStopPythonScriptExecution() {
		ripclient.set(["setclearscript"],[0]);
		ripclient.set(["setmov"],[99]);
		updateLogger('Stop Python Script Code...');
		}

btnuploadprogram		function ejsSManualUploadPythonScriptToGoPiGo2Robot(){
		if (document.getElementById('editor') !== null) {
		ripclient.set(["setscript"],[scriptData]);
		updateLogger('Upload Python Script Code...');
		updateLogger('SENT CODE: ' + scriptData);}
		}

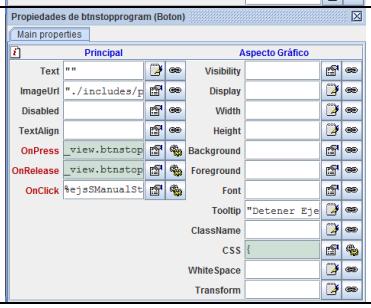
btnstopprogram		function ejsSManualStopPythonScriptExecution() {
		ripclient.set(["setclearscript"],[0]);
		ripclient.set(["setmov"],[99]);
		updateLogger('Stop Python Script Code...');
		}

Tabla 29. Configuración de los componentes "btnuploadscript", "btnstopscript", "btnuploadprogram" y "btnstopprogram".

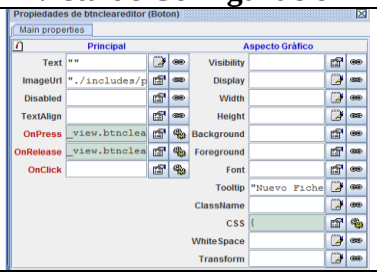
Componente	Vista de Configuración	Código JS / Handler
btnclearеditor		%ejsSclearEditor%

Tabla 30. Configuración del componente "btnclearеditor".

4.2.2.4. Aplicación Web EjsS. Panel Inferior.

Como se ha comentado con anterioridad el *"Panel Inferior"* incluye los componentes necesarios para la visualización de los logs de la aplicación. En la Figura 50 se puede apreciar como es este componente.

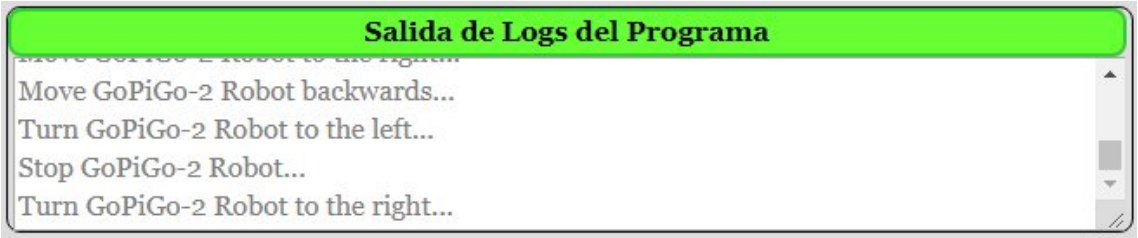


Figura 50. Vista general del componente "Panel Inferior".

La vista de elementos anterior se puede descomponer en la vista de componentes del framework EjsS de la Figura 51.

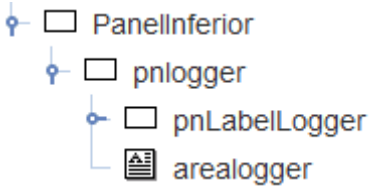


Figura 51. Descomposición en componentes del elemento "Panel Inferior".

Como se puede apreciar en la figura anterior se dispone de un componente denominado *"arealogger"* en el cual se muestran todas las interacciones que el usuario realiza con la aplicación de control EjsS. Lo interesante de este componente es que también está pensado para mostrar las trazas de ejecución cuando se está usando el framework Blockly con la aplicación EjsS por lo que se puede depurar de forma más sencilla un programa escrito en Blockly.

4.2.2.5. Aplicación Web EjsS. Configuración del Ciclo Automático de Actualización de la Aplicación.

Como se ha visto en los apartados anteriores en algunos manejadores de componentes gráficos de la aplicación EjsS hay variables que son actualizadas mediante el uso de una serie de funciones o métodos definidos en el cuadro *"Evolución"* de la pestaña *"Modelo"*. Estas funciones han sido declaradas para que sean ejecutadas de forma cíclica durante el tiempo que la aplicación EjsS está activa.

La misión de estas funciones o métodos es actualizar algunos valores o ejecutar algún procedimiento en concreto que es necesario para el correcto funcionamiento de la aplicación EjsS. En la Figura 52 se pueden apreciar cuales son los métodos añadidos en el cuadro *"Evolución"* de la pestaña *"Modelo"*.

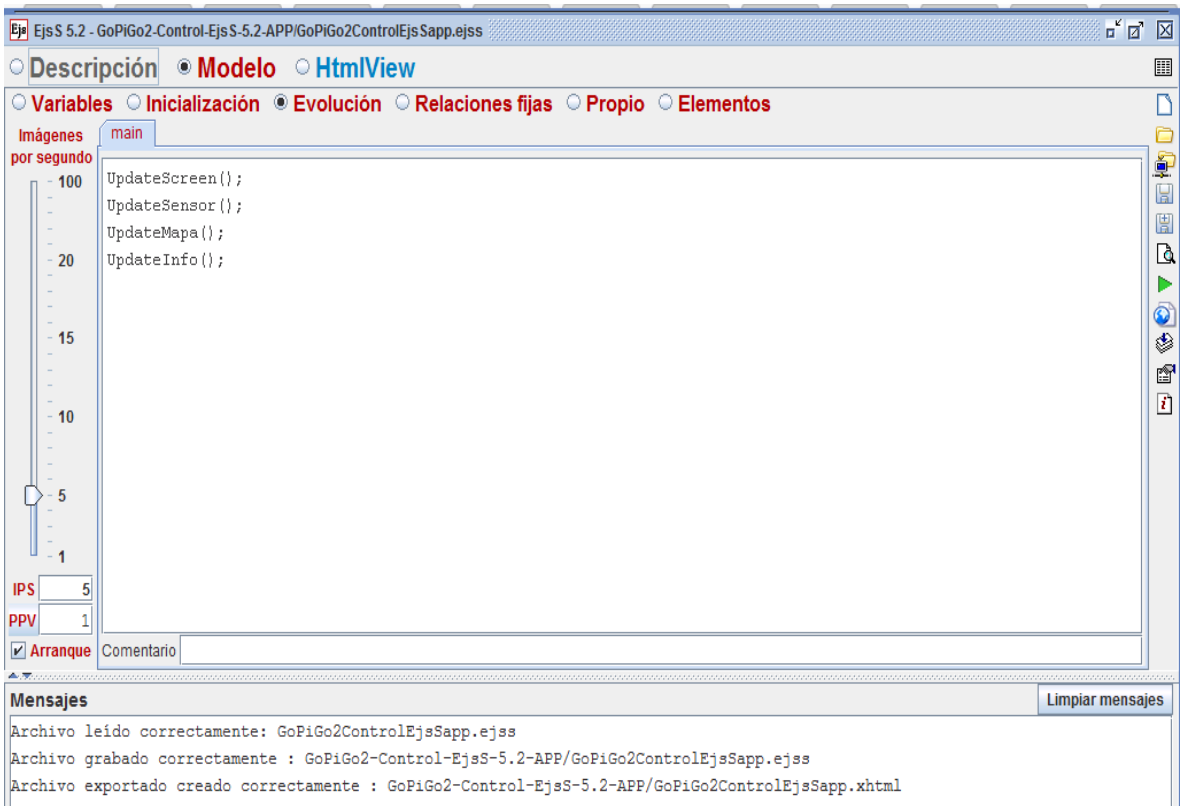


Figura 52. Listado de los métodos que actualizan de forma automática el estado de la aplicación EjsS.

A continuación se incluye una breve descripción de que cometido tienen cada una de las funciones anteriores:

- ***UpdateScreen():*** esta función se encarga comprobar si existe conectividad entre la aplicación EjsS y el robot GoPiGo-2 y en caso afirmativo mostrar la versión del robot en sub-panel *"pnRobot"* incluido en el panel principal de configuración del robot GoPiGo-2.

- ***UpdateSensor():*** esta función se encarga de actualizar los componentes gráficos del sub-panel *"pnsensores"* incluido en el panel principal de configuración del robot GoPiGo-2. De esta forma se tiene una actualización constante de los valores de los encoders y de las lecturas tomadas por el sensor de ultrasonidos del robot.

- ***UpdateMapa():*** esta función se encarga de actualizar el mapa de ocupación incluido en el sub-panel *"pnmapa"* incluido en el panel de control remoto del robot GoPiGo-2. De esta forma se tienen lecturas actualizadas dentro del mapa de ocupación de posibles obstáculos detectados mediante el sensor de ultrasonidos instalado en el robot GoPiGo-2.

- ***UpdateInfo():*** esta función se encarga de actualizar el área de la aplicación dicada a los logs generados por el servidor RIP instalado en el robot GoPiGo-2 y que pueden ser visualizados en el sub-panel *"pnlogger"* incluido en el panel de programación del robot GoPiGo-2.

El código fuente de todas estas funciones (así como de otras sub-funciones de las que dependen) está definido en el cuadro *"Propio"* de la pestaña *"Modelo"* y se puede consultar en el Anexo C de este trabajo.

4.3. Programación del robot GoPiGo-2 mediante el editor de código EjsS.

Como se ha visto en los apartados anteriores, la aplicación EjsS desarrollada para este trabajo incluye una sección en al cual el usuario puede escribir scripts o programas de control para su ejecución en el robot GoPiGo-2. La diferencia entre subir un script Python o un programa de control Python radica en que en el primer caso se hace uso de la clase *GoPiGoContext.py* para acceder a las variables del robot en el servidor, mientras que en el segundo caso se envía al robot código Python que hace uso directamente del API nativo de programación del robot

Es por ello que en esta sección se presentan cuatro ejemplos (dos scripts y dos programas) que permiten explicar los conceptos básicos de la programación del robot GoPiGo-2 por medio del lenguaje de programación Python haciendo uso tanto de la clase *"GoPiGoContext.py"* como haciendo uso del API nativa del robot. El código fuente de los scripts y programas se puede encontrar en el Anexo D de este trabajo.

4.3.1. Script 1. Control Básico de Evitación de Obstáculos.

El primer script que se presenta en esta sección es una sencilla función de control que permite al robot desplazarse por el entorno y detenerse ante la presencia de un obstáculo manteniendo un margen o distancia de seguridad con respecto al obstáculo detectado. El funcionamiento del programa se puede resumir de la siguiente forma:

- Desplazar al robot de forma aleatoria a través del entorno.
- Actualizar la posición del robot en base a los datos de odometría.
- Comprobar 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.
- Determinar la distancia a la cual el robot se encuentra el obstáculo y detiene el movimiento del robot.

Así pues, a la vista de las pautas anteriores lo primero que hace el programa, en cada ejecución, es obtener los parámetros del robot, consultando las propiedades del objeto de la clase *"GoPiGoContext"*.

```
# recuperar parámetros del contexto.
d_left_wheel = gopigocontext['ENCMOTORIZQ']
d_right_wheel = gopigocontext['ENCMOTORDER']
dis_ultrasonic = gopigocontext['ULTRASONIC']
orientacion_ultrasonic = gopigocontext['ULTRASONICHEAD']
dir_motor = gopigocontext['DIRMOTOR']
prev_x = gopigocontext['robotPosX']
prev_y = gopigocontext['robotPosY']
xmax = gopigocontext['xmax']
ymax = gopigocontext['ymax']
prev_theta = gopigocontext['orientacion']
auxiliar = gopigocontext['auxiliar']
wheel_base_lenght = gopigocontext['wheel_base_lenght']
```

Figura 53. Script 1. Segmento de código de obtención de los parámetros del robot desde el contexto.

Para calcular la posición del robot es necesario emplear los datos proporcionados por los encoders y los cálculos de odometría. Así en cada ciclo de ejecución se leen 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) se debe utilizar la información proporcionada por los encoders, sabiendo que la cinemática del robot es de tipo diferencial. Así pues, lo primero es estimar el desplazamiento realizado por cada rueda en cada pulso registrado por los encoders. Una vez obtenida la diferencia se calcula posición del robot por medio de las fórmulas:

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

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

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

La API del robot proporciona la distancia total recorrida por cada rueda, y esta información es accesible para el usuario a través de las variables "*ENCMOTORIZQ"* y "*ENCMOTORDER"* almacenadas en el contexto.

```
# calcular los movimientos de las ruedas.
prev_left_wheel=0
prev_right_wheel=0
if 'd_left_wheel' in auxiliar:
    prev_left_wheel = auxiliar['d_left_wheel']
    prev_right_wheel = auxiliar['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:
    # calcular la nueva posición del robot.
    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)
```

Figura 54. Script 1. Segmento de código para el cálculo de la posición del robot.

Por otro lado, es necesario calcular la posición de los posibles obstáculos una vez que estos son detectados por el sensor de ultrasonidos. Para ello se utiliza la medición del sensor de ultrasonidos de tal forma que si esta es menor a una distancia de seguridad se asume que hay un obstáculo en frente del robot y se procede a calcular su posición.

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

```
# Calcular la posición del obstáculo
xobs = -1
yobs = -1
if dis_ultrasonic < obstaculo_min:
    xobs = new_x + ((dis_ultrasonic+12)*math.cos(new_theta))/10
    yobs = new_y + ((dis_ultrasonic+12)*math.sin(new_theta))/10
```

Figura 55. Script 1. Segmento de código para el cálculo de la posición del obstáculo.

El último cálculo del programa es determinar la velocidad que ha de aplicarse a los motores. Dado que el programa detiene el robot ante la presencia de un obstáculo en este punto el código del programa se encarga de enviar el código de parada (***dir_motor = 99***) con independencia del tipo de movimiento que el robot estuviese realizando en ese momento (avanzar o retroceder).

```
# Calcular la velocidad de los motores
rand = -1
if 'rand' in auxiliar:
    rand = auxiliar['rand']
if rand == -1:
    rand = randint(0,9)
if dir_motor == 0:
    dir_motor = 99

if dir_motor != 2 :
    dir_motor = 99
else:
    dir_motor = 99
```

Figura 56. Script 1. Segmento de código para el cálculo de la parada del robot.

4.3.2. Script 2. Navegación con Evitación de Obstáculos.

El segundo script que se presenta en esta sección es una función de control algo más compleja que la anterior y que permite al robot generar de forma dinámica del entorno que rodea al robot. El funcionamiento del programa se puede resumir de la siguiente forma:

- Desplazar al robot de forma aleatoria a través del entorno.
- Actualizar la posición del robot en base a los datos de odometría.
- Comprobar 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.
- Determinar la distancia a la cual el robot se encuentra el obstáculo y cambiar la trayectoria del robot (avanzando, girando o retrocediendo) para que pueda seguir explorando el entorno sin chocar.

Este segundo programa realiza las mismas funciones básicas que el programa de control con detección de obstáculos presentado en el apartado anterior siendo estas: cargar los datos de contexto, calcular la posición del robot, calcular la posición de los posibles obstáculos y actualizar la velocidad del robot. La última función relativa a la actualización de la velocidad del robot cambia en este segundo programa ya que ahora en lugar de enviar la orden de parada se debe de calcular la velocidad del robot en función de la medición del sensor de ultrasonidos con el fin de poder evitar los posibles obstáculos del entorno. Así pues, si dicha 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. Este cálculo de velocidades se realiza mediante el segmento de código que se presenta a continuación.

```
# Calcular la velocidad de los motores
rand = -1
if 'rand' in auxiliar:
    rand = auxiliar['rand']
if rand == -1:
    rand = randint(0,9)
if dir_motor == 0:
    dir_motor = 1

if dir_motor != 2 :
    # El robot se mueve hacia adelante
    if dis_ultrasonic < 10:
        # girar a la derecha
        rand = -1
        gopigocontext['DIRMOTOR']=2
        gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
        gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
    elif dis_ultrasonic < 20:
        gopigocontext['DIRMOTOR']=1
        if rand%2==0:
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
        else:
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
    elif dis_ultrasonic < 50:
        # retroceder
        gopigocontext['DIRMOTOR']=1
        if rand%2==0:
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
        else:
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
    else:
        rand = -1
        gopigocontext['DIRMOTOR']=1
        gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
        gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
else:
    if dis_ultrasonic<20:
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
    elif dis_ultrasonic < 50:
        # retroceder
        gopigocontext['DIRMOTOR']=1
        if rand%2==0:
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
        else:
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
    else:
        rand = -1
        gopigocontext['DIRMOTOR']=1
        gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
        gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
else:
    if dis_ultrasonic<20:
        # girar a la izquierda
        gopigocontext['DIRMOTOR']=2
        if rand%2==0:
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
        else:
            gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
            gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
    elif dis_ultrasonic < 50:
        gopigocontext['DIRMOTOR']=2
        gopigocontext['VELRIGHTMOTOR'] = velocidad_izquierda
        gopigocontext['VELLEFTMOTOR'] = velocidad_derecha
    else:
        gopigocontext['DIRMOTOR']=1
        gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
        gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
```

Figura 57. Script 2. Segmento de código para el cálculo de las velocidades del robot.

4.3.3. Programa 1. Evitación de Obstáculos Simple.

El primer programa que se presenta en esta sección es una sencilla función de control que permite al robot desplazarse hacia adelante por el entorno y detenerse ante la presencia de un obstáculo manteniendo un margen o distancia de seguridad con respecto al obstáculo detectado. El funcionamiento del programa se puede resumir de la siguiente forma:

- Desplazar al robot hacía adelante en el entorno.
- Comprobar el valor del sensor de ultrasonidos. Si detecta un obstáculo a una distancia menor que la distancia de seguridad entonces se debe determinar la distancia a la cual el robot se encuentra del obstáculo y detiener el movimiento del robot.

Con todo ello el programa de control es el que se puede consultar en la Figura 58.

```
from gopigo import *
import time

#Distancia al obstáculo a la cual se debe detener el robot.
distance_to_stop = 20

#orden de avance del robot.
fwd()

while True:
    # Obtener una lectura del sensor de ultrasonidos
    dist = us_dist(15)
    # Si el objeto detectado se encuentra por debajo del umbral
    # entonces se debe detener el robot GoPiGo-2.
    if dist < distance_to_stop:
        # orden de parada del robot.
        stop()
        break
    time.sleep(.1)
```

Figura 58. Programa 1. Segmento de código para el control del robot en avance y detección de obstáculo con parada.

4.3.4. Programa 2. Mapeo de Entorno con Evitación de Obstáculos.

El segundo programa que se presenta en esta sección es una función de control que permite al robot desplazarse hacia adelante por el entorno, tomar medidas con el sensor de ultrasonidos, pintar la ocupación en un mapa por la salida de la consola de comandos y detenerse ante la presencia de un obstáculo manteniendo un margen o distancia de seguridad con respecto al obstáculo detectado. El funcionamiento del programa se puede resumir de la siguiente forma:

- Desplazar al robot hacía adelante en el entorno fijando como referencia en los encoders 4 vueltas completas de las ruedas del robot.
- Detener el robot para tomar medidas con el sensor de ultrasonidos.
    - Activar el servo-motor.
    - Tomar medidas con el sensor de ultrasonidos aprovechando el movimiento del servo-motor para escanear una región concreta de 180 grados.
    - Tomar las medidas que más se repiten dentro del ángulo de escaneo para pintar la ocupación en una malla por salida de la consola de comandos.
- Comprobar el valor del sensor de ultrasonidos. Si detecta un obstáculo a una distancia menor que la distancia de seguridad entonces se debe determinar la distancia a la cual el robot se encuentra del obstáculo y detienen el movimiento del robot.

A la vista de las pautas anteriores lo primero que hace el programa, en cada ejecución, desplazar el robot hacía adelante completando 4 vueltas completas de las ruedas gracias a la configuración de los encoders (Figura 59).

```
# Mover el robto hacia adelante, parar crear el mapa y volver a mover el robot
# hacia adelante.
enable_com_timeout(2000)
enc_tgt(1,1,18) #SFijar el encoder. Parar después de 4 rotacfiones de las ruedas.
fwd()
time.sleep(.2)
```

Figura 59. Programa 2. Segmento de código para el control del robot en avance mediante cuentas del encoder.

En la Figura 60 se puede apreciar como es el cuerpo del método *"us_map"* desarrollada. Este método hace uso de la función *"us_dist()"* del API nativa del robot GoPiGo-2 para obtener en cada instante una medición del sensor de ultrasonidos a través del puerto 15 (puerto en el que está conectado el robot). Además e comprueba que el posible obstáculo detectado se encuentra dentro de unos límites definidos para el mapa de ocupación y se procede a obtener las muestras que se más se repiten dentro del ángulo de escaneo. El último paso consiste en mover el servo-motor un nuevo ángulo para realizar el escaneo en dicha sección la siguiente vez que se ejecuta el código.

```
while True:
    # Tomar las medidas obtenidas por el sensor de ultrasonidos.
    for i in range(sample):
        dist=us_dist(15)
        if dist<lim and dist>=0:
            buf[i]=dist
        else:
            buf[i]=lim

    # Encontrar la muestra que más se repite dentro del ángulo de escaneo.
    max=Counter(buf).most_common()
    rm=-1
    for i in range (len(max)):
        if max[i][0] <> lim and max[i][0] <> 0:
            rm=max[i][0]
            break
    if rm==-1:
        rm=lim

    if debug==1:
        print index,ang,rm
    ang_l[index]=ang
    dist_l[index]=rm
    index+=1

    # Mover el servo al siguiente ángulo.
    servo(ang)
    time.sleep(delay)
    ang+=incr
    if ang>180:
        break
```

Figura 60. Programa 2. Segmento de código para la obtención de medidas con el sensor de ultrasonidos y movimiento del servo.

En cada iteración de la función *"us_map()"* se devuelve la distancia al posible obstáculo detectado. En caso de que el obstáculo se encuentre a una distancia inferior a un margen de seguridad entonces el robot se detiene (Figura 61).

```
enable_servo()
# Detener el robot si algún objeto se ecneuntra a una distancia inferior a 20 cm.
if us_map() <20:
    break
disable_servo()
```

Figura 61. Programa 2. Segmento de código para la detección de obstáculo con parada del robot en caso de que la distancia medida sea inferior al umbral.

4.4. Integración de la Aplicación Web / EjsS de control del robot GoPiGo 2 en el framework Blockly.

En este apartado se va a proceder a describir como se ha desarrollado la integración de la aplicación Web basada en el framework EjsS desarrollada y el framework *Blockly*. El apartado está dividido en dos secciones. En la primera se va a proceder a explicar los fundamentos teóricos más básicos del framework *Blockly*, mientras que la segunda se centra en los pormenores del desarrollo realizado.

4.4.1. Framework Blockly.

Blockly es un lenguaje de programación visual desarrollado por *Google* compuesto por un sencillo conjunto de comandos que pueden ser combinados como si fueran las piezas de un rompecabezas. Es una herramienta muy útil para el que quiera aprender a programar de una forma intuitiva y simple. Por lo general, se ejecuta en un navegador web y se asemeja visualmente a Scratch. El framework *Blockly* dispone también de versiones para Android e iOS aunque no todas las funciones disponibles en la distribución para navegador web están disponibles en las distribuciones para Android / iOS.

Blockly permite a los programadores noveles el concentrarse en la lógica del programa. El framework ofrece una serie de comandos básicos así cómo una serie de normas que determinan la forma en la cual se pueden colocar juntos. *Blockly* permite arrastrar los distintos componentes de control, lógica, operaciones matemáticas, texto, listados y procesos para crear sencillos scripts que pueden ser exportados a diferentes lenguajes de programación como JavaScript, Dart, Python. Esto último es un interesante punto a tener en cuenta. El framework no está concebido para el desarrollo de aplicaciones muy complejas pero es una herramienta muy potente que puede ser utilizada por los programadores más experimentados para definir un prototipo rápido que posteriormente sea exportado a código.

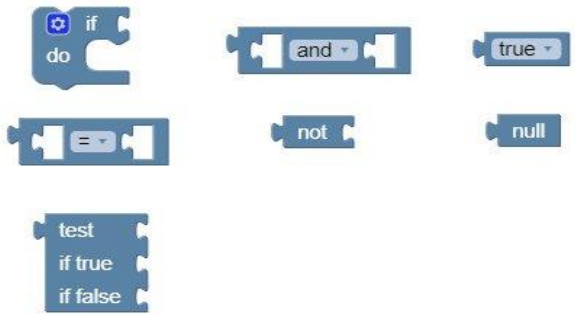
Framework Blockly – Bloques Lógicos	
Bloques Funcionales	**Descripción**
	Bloques de operaciones lógicas. Se trata de bloques que pueden ser utilizados para comprobar condiciones o bien operadores lógicos de tipo AND, OR, NOT.

Tabla 31. Framework Blockly. Bloques Lógicos.

Dentro de las opciones disponibles para el desarrollo de código, el framework *Blockly* ofrece toda una gama de bloques que son equivalentes a las expresiones que se pueden encontrar en cualquier lenguaje de programación actual. Así el framework dispone de bloques que funcionan como operadores lógicos (Tabla 31) utilizados para comprobar condiciones o implementar operadores lógicos. Estos bloques pueden ser utilizados en conjunción con los bloques que representan bucles de ejecución (Tabla 32) para componer lógicas de programa más complejas.

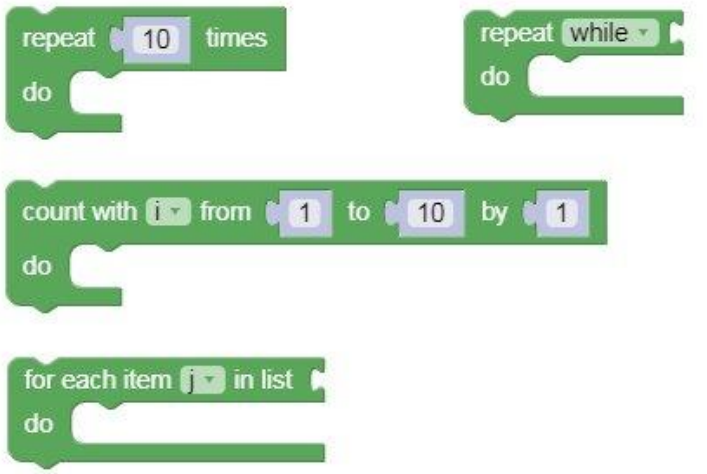
Framework Blockly – Bloques de Bucle	
Bloques Funcionales	**Descripción**
	Bloques de Bucle. Se trata bloques que representan bucles de ejecución.

Tabla 32. Framework Blockly. Bloques de Bucle.

Al igual que ocurre con los lenguajes de programación actuales, el framework *Blockly* define una serie de bloques que permiten realizar operaciones matemáticas empleando para ello los operadores básicos pero además incluye una amplia biblioteca que incluye operaciones trigonométricas y logarítmicas entre otras (Tabla 33).

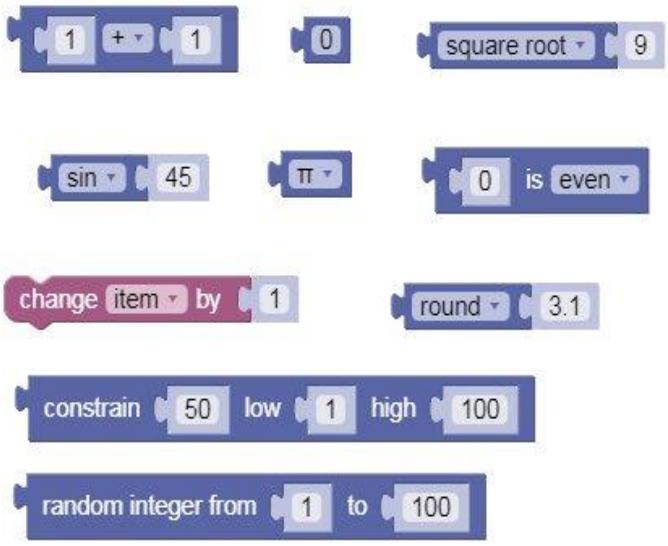
Framework Blockly – Bloques de Operaciones Matemáticas	
Bloques Funcionales	**Descripción**
	Bloques de Operaciones Matemáticas. Se trata bloques que representan operaciones matemáticas.

Tabla 33. Framework Blockly. Bloques de Operaciones Matemáticas.

Otra familia de bloques muy utilizada es la relativa a las operaciones sobre cadenas de caracteres (Tabla 34). Esta familia de bloques permite entre otras operaciones comparar cadenas de caracteres, obtener sub-cadenas, buscar cadenas de caracteres dentro de otras cadenas de caracteres, etc.

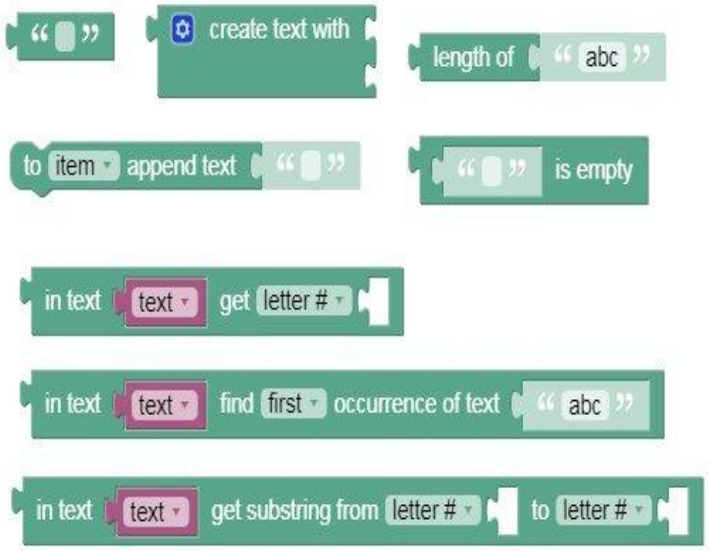
Framework Blockly – Bloques de Operaciones Sobre Cadenas de Caracteres	
Bloques Funcionales	**Descripción**
	Bloques de Operaciones sobre cadenas de caracteres. Se trata bloques que representan operaciones sobre cadenas de caracteres.

Tabla 34. Framework Blockly. Bloques de Operaciones sobre cadenas de caracteres.

Finalmente, el framework define dos conjuntos de bloques muy interesantes para el usuario. Por un lado los bloques relativos al manejo de estructuras de datos más complejas, concretamente *"listas de elementos"* (Tabla 35) lo que implica disponer de bloques que permiten realizar toda clase de operaciones sobre listas (crear listas, añadir elementos, borrar elementos, etc).

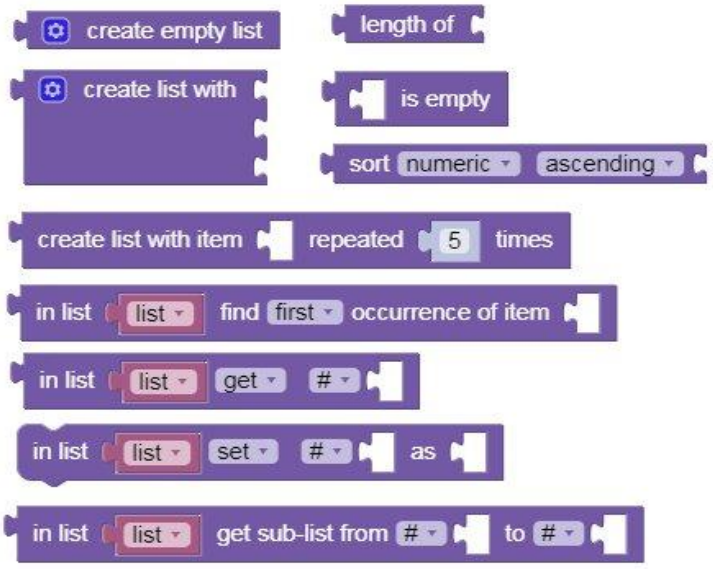
Framework Blockly – Bloques de Operaciones sobre Listas de Elementos	
Bloques Funcionales	**Descripción**
	Bloques de Operaciones sobre Listas de Elementos. Se trata bloques que representan operaciones sobre listas de elementos.

Tabla 35. Framework Blockly. Bloques de Operaciones sobre listas de elementos.

El otro conjunto de bloques más interesantes para el usuario está representado por aquellos bloques que permiten definir funciones propias por parte del usuario (Tabla 36) o ejecutar segmentos de código en algún lenguaje de programación soportado (por ejemplo Python).

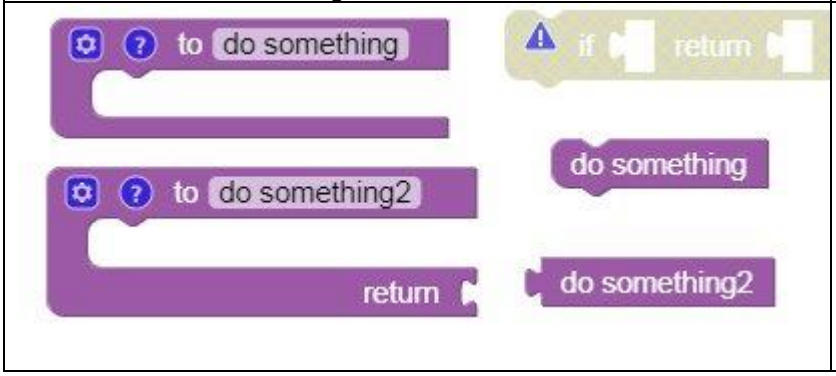
Framework Blockly – Bloques de Ejecución de Funciones	
Bloques Funcionales	**Descripción**
	Bloques de ejecución de Funciones y código externo.

Tabla 36. Framework Blockly. Bloques de Ejecución de Funciones.

Por otro lado, la versión del framework Blockly desplegada en el entorno remoto de UniLabs incluye una serie de bloques propios que permiten realizar, entre otras funciones, la integración de un programa Blockly con una aplicación EjsS (Tabla 37).

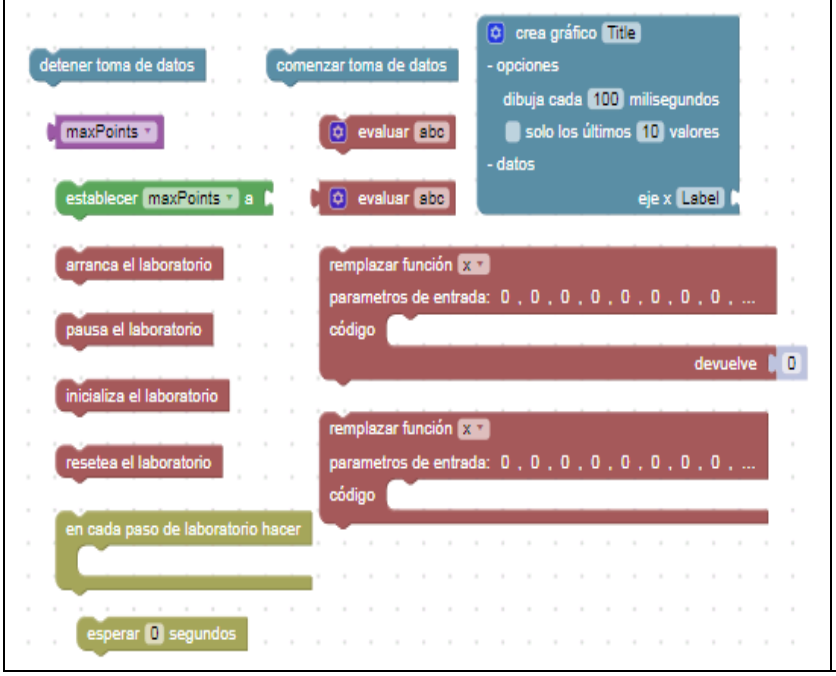
Framework Blockly – Bloques propios del entorno de UniLabs	
Bloques Funcionales	**Descripción**
	Bloques propios del entorno de laboratorios remotos UniLabs.

Tabla 37. Framework Blockly. Bloques propios del entorno de UniLabs..

En la Figura 62 se puede apreciar como es un programa simple definido mediante bloques básicos del framework *Blockly* y su correspondencia en código *JavaScript*.

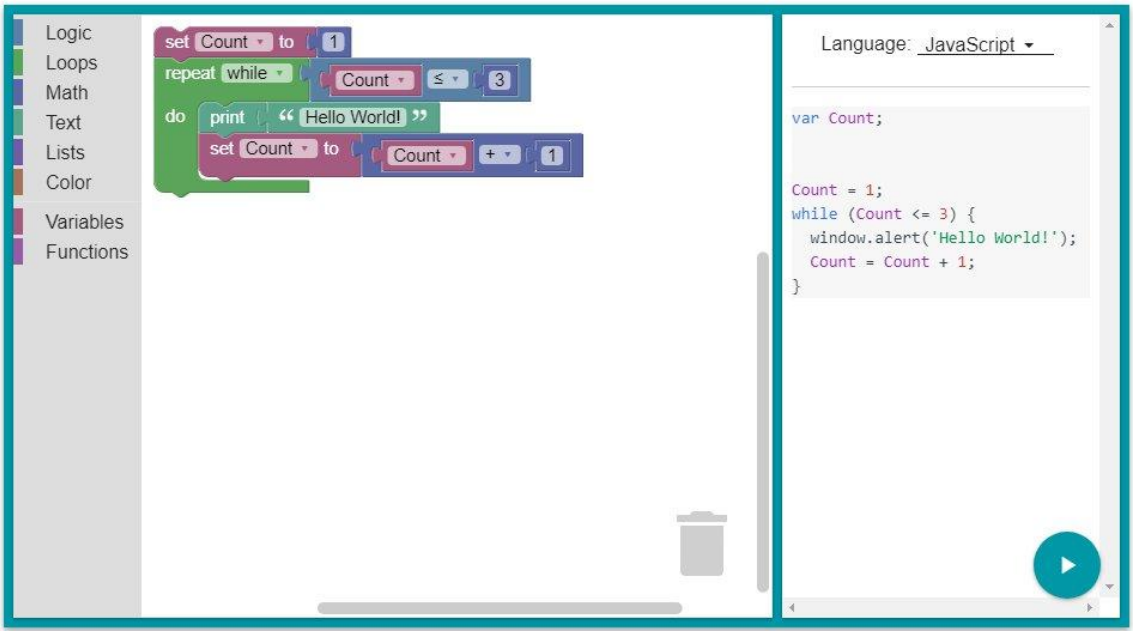


Figura 62. Ejemplo de programa definido mediante bloques del Framework Blockly y su equivalencia en lenguaje JavaScript.

En el siguiente apartado se va a proceder a describir como es el proceso de integración entre la aplicación EjsS desarrollada para el control del robot GoPiGo-2 y el framework *Blockly* dentro del marco de la plataforma de laboratorios remotos *UNILabs*.

4.4.2. Aplicación Web EjsS. Integración con el framework Blockly.

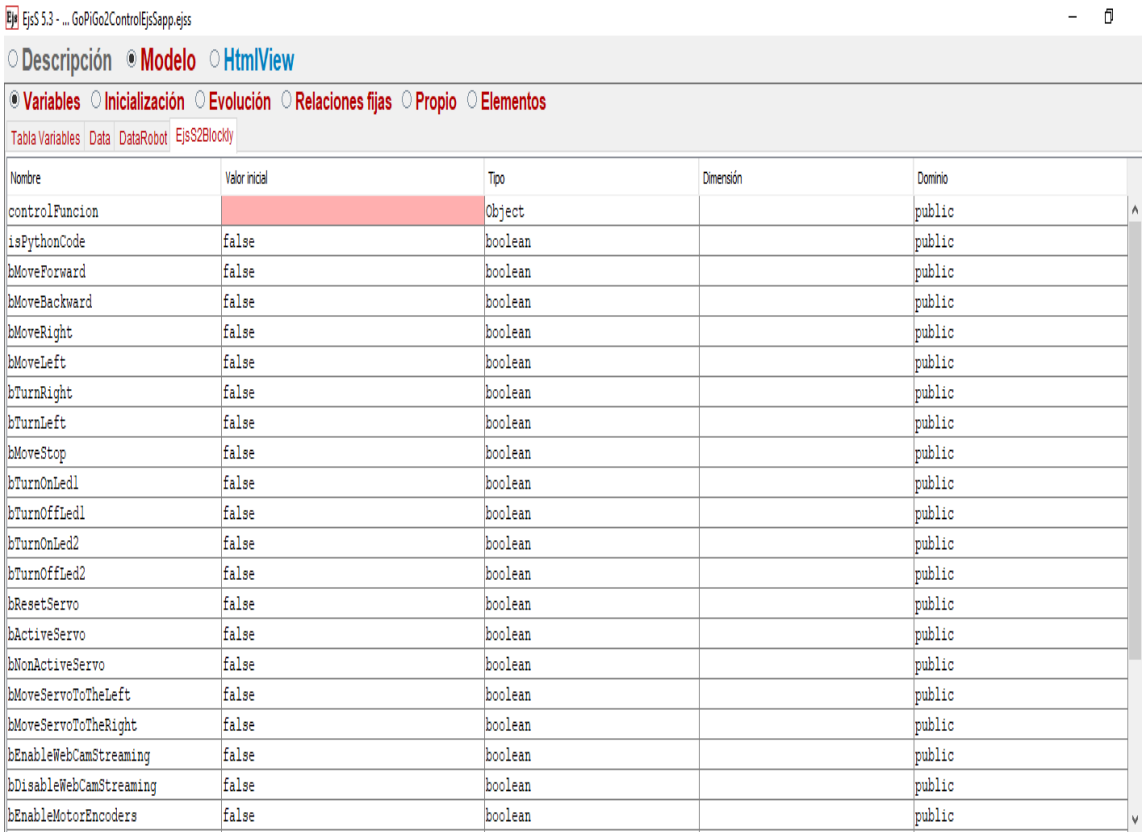
En este apartado se va a proceder a presentar el método empleado para realizar la integración de la aplicación EjsS de control del robot GoPiGo-2 con el framework Blockly desplegado en los entornos de laboratorios remotos. El objetivo de esta integración es poder usar las facilidades de Blockly para desarrollar programas de control para el robot GoPiGo-2 haciendo uso de los bloques que ofrece este framework. En este sentido el proceso de integración consta de los siguientes pasos:

- Definición de un conjunto de variables *"públicas"* en la aplicación EjsS. Este tipo de variables son accesibles tanto desde la aplicación EjsS como desde el entorno de laboratorios remotos, y por tanto, desde Blockly una vez que la aplicación EjsS está desplegada.
- Definición de un conjunto de métodos de *"acción"* que permitan realizar operaciones sobre el robot cuando las variables *"públicas"* cambian de estado.
- Utilización de los bloques de la sección *"Laboratorio"* del framework Blockly instalado en el sistema de laboratorios remotos para acceder y cambiar el estado de las variables *"públicas"*.
- Definición de un método principal en la aplicación EjsS que continuamente esté comprobando el estado de las variables *"públicas"* para tomar la decisión de que método de *"acción"* ejecutar en cada caso.

A la vista de los pasos anteriores, una vez cumplidos todos ellos y definido un conjunto de bloques de *"Laboratorio"* basta con arrancar el laboratorio remoto para que comience la ejecución de los bloques que hacen uso de las variables *"públicas"* para que el robot empiece a seguir las órdenes definidas en dicho conjunto de bloques.

4.4.2.1. Definición de las variables públicas en la aplicación EjsS.

Como se ha explicado con anterioridad uno de los pasos esenciales para poder comunicar la aplicación EjsS con el framework Blockly desplegado en el entorno de laboratorios remotos es la definición de las variables *"públicas"* accesibles desde EjsS y desde Blockly. En la Figura 63 se incluye un ejemplo de definición de variables públicas para la aplicación EjsS de control del robot GoPiGo-2.



Nombre	Valor inicial	Tipo	Dimensión	Dominio
controlFuncion		Object		public
isPythonCode	false	boolean		public
bMoveForward	false	boolean		public
bMoveBackward	false	boolean		public
bMoveRight	false	boolean		public
bMoveLeft	false	boolean		public
bTurnRight	false	boolean		public
bTurnLeft	false	boolean		public
bMoveStop	false	boolean		public
bTurnOnLed1	false	boolean		public
bTurnOffLed1	false	boolean		public
bTurnOnLed2	false	boolean		public
bTurnOffLed2	false	boolean		public
bResetServo	false	boolean		public
bActiveServo	false	boolean		public
bNonActiveServo	false	boolean		public
bMoveServoToTheLeft	false	boolean		public
bMoveServoToTheRight	false	boolean		public
bEnableWebCamStreaming	false	boolean		public
bDisableWebCamStreaming	false	boolean		public
bEnableMotorEncoders	false	boolean		public

Figura 63. Definición de variables públicas en la aplicación EjsS de control del Robot GoPiGo-2.

Como se puede apreciar en la figura anterior el dominio de todas las variables definidas en la pestaña *"EjsS2Blockly"* es de tipo *"public"*. A continuación se presenta la Tabla 38 con la definición completa de todas las variables *"públicas"* utilizadas en este trabajo para integrar la aplicación EjsS con el framework Blockly.

Nombre de la Variable	Valor Inicial	Tipo	Dimensión	Descripción
bMoveForward	false	Booleano	pública	Activa / Desactiva el movimiento hacia adelante del robot.
bMoveBackwards	false	Booleano	pública	Activa / Desactiva el movimiento hacia atrás del robot.
bMoveRight	false	Booleano	pública	Activa / Desactiva el movimiento hacia la derecha del robot.
bMoveLeft	false	Booleano	pública	Activa / Desactiva el movimiento hacia la izquierda del robot.
bTurnRight	false	Booleano	pública	Activa / Desactiva el giro hacia la derecha del robot.
bTurnLeft	false	Booleano	pública	Activa / Desactiva el giro hacia la izquierda del robot.
bMoveStop	false	Booleano	pública	Activa / Desactiva la parada del robot.
bTurnOnLed1	false	Booleano	pública	Enciende el LED-1 del robot.
bTurnOffLed1	false	Booleano	pública	Apaga el LED-1 del robot.
bTurnOnLed2	false	Booleano	pública	Enciende el LED-2 del robot.
bTurnOffLed2	false	Booleano	pública	Apaga el LED-2 del robot.
bResetServo	false	Booleano	pública	Aplica un reset a la posición del servo.
bActiveServo	false	Booleano	pública	Activa el Servo-Motor.
bNonActiveServo	false	Booleano	pública	Desactiva el Servo-Motor.
bMoveServoTheTheLeft	false	Booleano	pública	Activa / Desactiva el movimiento del servo hacía izquierda.
bMoveServoTheTheRight	false	Booleano	pública	Activa / Desactiva el movimiento del servo hacía derecha.
bEnableWebCamStreaming	false	Booleano	pública	Activa la web-cam.
bDisableWebCamStreaming	false	Booleano	pública	Desactiva la web-cam.
bEnableMotorEncoders	false	Booleano	pública	Activa / Desactiva los encoders.
bSetEncodersValue	0	Entero	pública	Establece el valor de los encoders.
bSetUltraSoundPort	15	Entero	pública	Establece el valor del puerto del sensor de ultrasonidos.
bGetUltraSoundMeasure	0	Entero	pública	Obtiene la medida del sensor de ultrasonidos.
bSetMotorLeftSpeed	100	Entero	pública	Establece la velocidad en el motor izquierdo.
bSetMotorRightSpeed	100	Entero	pública	Establece la velocidad en el motor derecho.
bStateMachineCommand	-1	Entero	pública	Control de ejecución
bCloseConnectionWithTheRobot	false	Booleano	pública	Cierra la conexión con el robot.
bStablishConnectionWithTheRobot	false	Booleano	publica	Establece la conexión con el robot.

Tabla 38. Definición de las variables públicas de la aplicación EjsS accesibles desde el framework Blockly.

Todas las variables definidas en la Tabla 38 tienen su importancia para controlar las acciones del robot por medio del framework Blockly, siendo la variable *"bStateMachineCommand"* la más importante desde el punto de vista del ciclo de ejecución de acciones del robot pues está variable es la que permite que se ejecute una orden concreta en el robot. En la Tabla 39 se presentan los posibles valores que puede tomar la variable *"bStateCommand"* y que pueden ser usados para programar el robot.

Nombre de la Variable	Valor	Descripción
bStateMachineCommand	0	Envía la orden de parada al robot.
	1	Envía la orden de avance al robot.
	2	Envía la orden de retroceso al robot.
	3	Envía la orden de movimiento hacía la derecha al robot.
	4	Envía la orden de movimiento hacía la izquierda al robot.
	5	Envía la orden de giro hacia la derecha al robot.
	6	Envía la orden de giro hacia la izquierda al robot.
	7	Envía la orden de encendido del LED-1 al robot.
	8	Envía la orden de apagado del LED-1 al robot.
	9	Envía la orden de encendido del LED-2 al robot.
	10	Envía la orden de apagado del LED-2 al robot.
	11	Envía la orden de activación de la webcam al robot.
	12	Envía la orden de desactivación de la webcam al robot.
	13	Envía la orden de activación del servo-motor al robot.
	14	Envía la orden de desactivación del servo-motor al robot.
	15	Envía la orden de giro hacia la derecha del servo-motor al robot.
	16	Envía la orden de giro hacia la izquierda del servo-motor al robot.
	17	Envía la orden de reset del servo-motor al robot.
	18	Envía la orden de establecimiento del valor del puerto de ultrasonidos en el robot.
	19	Envía la orden de obtención de la medida recogida por el sensor de ultrasonidos.
	20	Envía la orden para actualizar el valor de la velocidad en el motor izquierdo.
	21	Envía la orden para actualizar el valor de la velocidad en el motor derecho.
	22	Envía la orden de activación y establecimiento del valor de cuenta en los encoders.
	23	Envía la orden de restablecimiento del valor de cuenta en los encoders.
	24	Envía la orden de establecimiento de conexión con el robot.
	25	Envía la orden de cierre de la conexión con el robot.

Tabla 39. Valores que puede tomar la variable "bStateMachineCommand".

4.4.2.2. Conexión de bloques del framework Blockly con las variables públicas de EjsS.

Como se definió durante la introducción de esta sección del trabajo para poder enviar órdenes al robot se han utilizado los bloques del framework Blockly que están disponibles en la pestaña *"Laboratorio"* del entorno de laboratorios remotos UniLabs. La forma más sencilla de conectar las variables públicas definidas en la aplicación EjsS con el framework Blockly de los laboratorios remotos es usando el bloque *"establecer"* que se encuentra disponible la ruta: *"Laboratorio → variables → establecer"*. En la Figura 64 se puede apreciar como es este bloque.

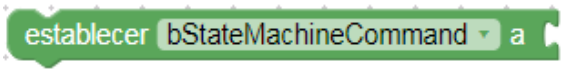


Figura 64. Bloque "establecer" del framework Blockly en el entorno de laboratorios. Ejemplo de uso.

Como se puede apreciar en la figura anterior el bloque *"establecer"* dispone de un menú desplegable en el que aparecen todas las variables públicas (y por tanto accesibles desde Blockly) definidas en la aplicación EjsS. De esta forma es muy sencillo modificar el estado de las variables en el robot para que este modifique su comportamiento en función de cómo se combinen sucesivos bloques *"establecer"* dentro del modelo. Uno de los aspectos destacables del framework Blockly es que permite extender sus funcionalidades pudiendo crear nuevos tipos de bloques a partir de código fuente o bloques ya existentes. Aunque esta es una tarea que queda fuera del presente trabajo, la idea de tener una librería de bloques exclusiva (Figura 65) para controlar el robot GoPiGo-2 en el entorno de laboratorios remotos es lo suficientemente atractiva como para plantear una evolución en el desarrollo realizado en el presente trabajo. Realizar esa tarea de adaptación a los bloques *"ideales"* no se debería ser muy complicado pues ya se dispone del modelo de los bloques actuales así como del código fuente en el framework EjsS.

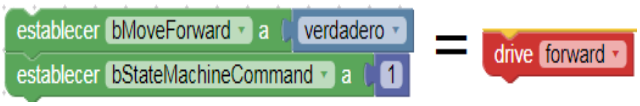
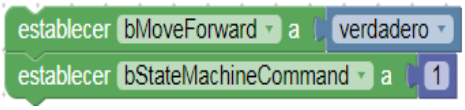
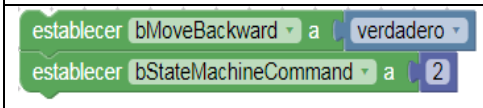


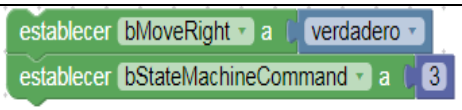
Figura 65. Equivalencia entre la combinación de bloques "establecer" del framework Blockly de los laboratorios remotos y el bloque "ideal" de la hipotética librería de control del robot GoPiGo-2.

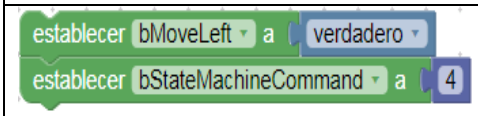
El framework utilizado es general para cualquier laboratorio que se use, ya sea de robótica, física, química y, por tanto, usa bloques generales. Desarrollar los bloques *"ideales"* es un trabajo futuro que se puede hacer para simplificar laboratorios de una misma temática como puede ser la *"Robótica"*. Sin embargo, no se ha hecho porque lo que se pretende en este trabajo es que cualquier alumno de la UNED pueda usar cualquier laboratorio sin tener que aprender bloques o estructuras nuevas, es decir, homogeneizar la oferta. Por ello, a continuación se incluyen una serie de tablas (Tabla 40, Tabla 41, Tabla 42 y Tabla 43) con las combinaciones de bloques que permiten enviar órdenes concretas al robot y que pueden servir de guía de programación del robot GoPiGo-2 por medio del framework Blockly.

Combinación de Bloques en el entorno de laboratorios remotos (UniLabs).	**Descripción**	**Código fuente de las acciones definido en la aplicación EjsS.**

	Envía la orden de avance al robot.	function blocklyMoveRobotForward(){
		bMoveBackward = bMoveRight = bMoveLeft = bTurnRight = bTurnLeft = bMoveStop = false;
		if (bMoveForward) {
		ripclient.set(["setmov"],[moveForward]);
		console.log("---> GoPiGo-2 Robot: moveForward");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: moveForward");
		} else {
		ripclient.set(["setmov"],[moveStop]);
		console.log("---> GoPiGo-2 Robot: moveStop");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: moveStop");
		}
		}

	Envía la orden de retroceso al robot.	function blocklyMoveRobotBackwards(){
		bMoveForward = bMoveRight = bMoveLeft = bTurnRight = bTurnLeft = bMoveStop = false;
		if (bMoveBackward) {
		ripclient.set(["setmov"],[moveBackward]);
		console.log("---> GoPiGo-2 Robot: moveBackward");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: moveBackward");
		} else {
		ripclient.set(["setmov"],[moveStop]);
		}
		}

	Envía la orden de movimiento hacía la derecha al robot.	function blocklyMoveRobotToTheRight(){
		bMoveForward = bMoveBackward = bMoveLeft = bTurnRight = bTurnLeft = bMoveStop = false;
		if (bMoveRight) {
		ripclient.set(["setmov"],[moveRight]);
		console.log("---> GoPiGo-2 Robot: moveRight");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: moveRight");
		} else {
		ripclient.set(["setmov"],[moveStop]);
		}
		}

	Envía la orden de movimiento hacía la izquierda al robot.	function blocklyMoveRobotToTheLeft(){
		bMoveForward = bMoveBackward = bMoveRight = bTurnRight = bTurnLeft = bMoveStop = false;
		if (bMoveLeft) {
		ripclient.set(["setmov"],[moveLeft]);
		console.log("---> GoPiGo-2 Robot: moveLeft");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: moveLeft");
		} else {
		ripclient.set(["setmov"],[moveStop]);
		}
		}

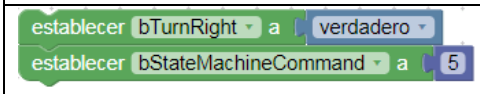
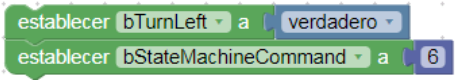
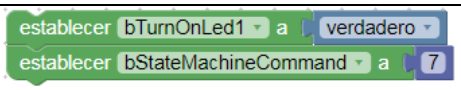
	Envía la orden de giro hacia la derecha al robot.	function blocklyTurnRobotToTheRight(){
		bMoveForward = bMoveBackward = bMoveRight = bMoveLeft = bTurnLeft = bMoveStop = false;
		if (bTurnRight) {
		ripclient.set(["setmov"],[turnRight]);
		console.log("---> GoPiGo-2 Robot: turnRight");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnRight");
		} else {
		ripclient.set(["setmov"],[moveStop]);
		}
		}

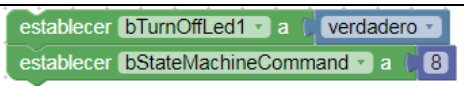
Tabla 40. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (I).

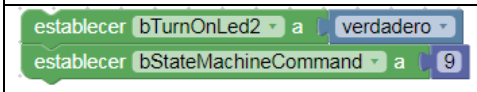
Combinación de Bloques en el entorno de laboratorios remotos (UniLabs).	**Descripción**	**Código fuente de las acciones definido en la aplicación EjsS.**

	Envía la orden de giro hacia la izquierda al robot.	function blocklyTurnRobotToTheLeft(){
		bMoveForward = bMoveBackward = bMoveRight = bMoveLeft = bTurnRight = bMoveStop = false;
		if (bTurnLeft) {
		ripclient.set(["setmov"],[turnLeft]);
		console.log("---> GoPiGo-2 Robot: turnLeft");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnLeft");
		} else {
		ripclient.set(["setmov"],[moveStop]);
		}
		}

	Envía la orden de parada al robot.	function blocklyStopRobot(){
		bMoveForward = bMoveBackward = bMoveRight = bMoveLeft = bTurnRight = bTurnLeft = false;
		if (bMoveStop) {
		ripclient.set(["setmov"],[moveStop]);
		console.log("---> GoPiGo-2 Robot: moveStop");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: moveStop");
		}
		}

	Envía la orden de encendido del LED-1 al robot.	function blocklyTurnOnLED1() {
		if (bTurnOnLed1) {
		if (led1Status === 0) {
		led1Status = 1;
		ripclient.set(["setledon"],[0]);
		console.log("---> GoPiGo-2 Robot: turnOn LED-1");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnOn LED-1");
		}bTurnOffLed1 = false;
		}
		}

	Envía la orden de apagado del LED-1 al robot.	function blocklyTurnOffLED1() {
		if (bTurnOffLed1) {
		if (led1Status === 1) {
		led1Status = 0; ripclient.set(["setledoff"],[0]);
		console.log("---> GoPiGo-2 Robot: turnOff LED-1");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnOff LED-1");
		}bTurnOnLed1 = false;
		}
		}

	Envía la orden de encendido del LED-2 al robot.	function blocklyTurnOnLED2() {
		if (bTurnOnLed2) {
		if (led2Status === 0) {
		led2Status = 1; ripclient.set(["setledon"],[1]);
		console.log("---> GoPiGo-2 Robot: turnOn LED-2");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnOn LED-2");
		}bTurnOffLed2 = false;
		}
		}

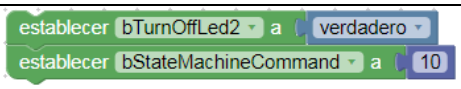
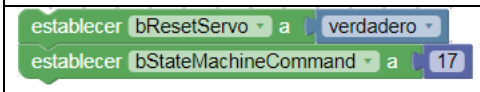
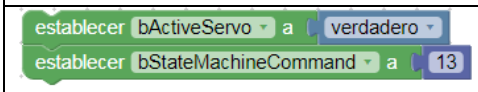
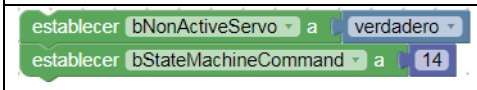
	Envía la orden de apagado del LED-2 al robot.	function blocklyTurnOffLED2() {
		if (bTurnOffLed2) {
		if (led2Status === 1) {
		led2Status = 0; ripclient.set(["setledoff"],[1]);
		console.log("---> GoPiGo-2 Robot: turnOff LED-2");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnOff LED-2");
		}bTurnOnLed2 = false;
		}
		}

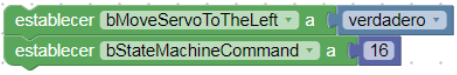
Tabla 41. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (II).

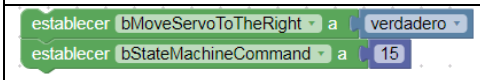
Combinación de Bloques en el entorno de laboratorios remotos (UniLabs).	**Descripción**	**Código fuente de las acciones definido en la aplicación EjsS.**

	Aplica un reset a la posición del servo.	function blocklyResetServoEngine() {
		if (bResetServo) {
		rangeServo = 0;
		ripclient.set(["setservo"],[rangeServo]);
		console.log("---> GoPiGo-2 Robot: reset Servo-Engine");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: reset Servo-Engine");
		} bResetServo = false;
		}

	Activa el Servo-Motor.	function blocklyTurnOnServo() {
		if (bActiveServo) {
		ripclient.set(["setservoEnabled","setservo"],[1,90]);
		activeServo = true;
		console.log("---> GoPiGo-2 Robot: turnOn Servo-Engine");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnOn Servo-Engine");
		}bActiveServo = false;
		}

	Desactiva el Servo-Motor.	function blocklyTurnOffServo() {
		if (bNonActiveServo) {
		ripclient.set(["setservoEnabled"],[0]);
		activeServo = false;
		console.log("---> GoPiGo-2 Robot: turnOff Servo-Engine");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: turnOff Servo-Engine");
		} bNonActiveServo = false;
		}

	Activa / Desactiva el movimiento del servo hacía izquierda.	function blocklyMoveServoToTheLeft() {
		if (bMoveServoToTheLeft && activeServo) {
		rangeServo = rangeServo + 10;
		if (rangeServo < 0) {
		rangeServo = 0;
		}
		ripclient.set(["setservo"],[rangeServo]);
		console.log("---> GoPiGo-2 Robot: move Servo-Engine to the Left");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: move Servo-Engine to the Left");
		} bMoveServoToTheLeft = false;
		}

	Activa / Desactiva el movimiento del servo hacía derecha.	function blocklyMoveServoToTheRight() {
		if (bMoveServoToTheRight && activeServo) {
		rangeServo = rangeServo - 10;
		if (rangeServo < 0) {
		rangeServo = 0;
		}
		ripclient.set(["setservo"],[rangeServo]);
		console.log("---> GoPiGo-2 Robot: move Servo-Engine to the Right");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: move Servo-Engine to the Right");
		} bMoveServoToTheRight = false;
		}

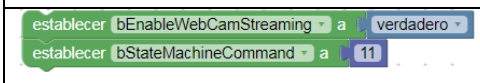
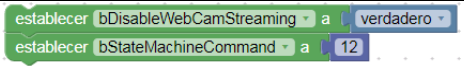
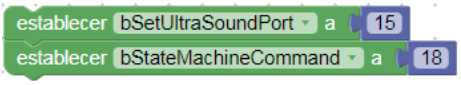
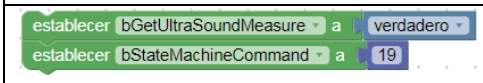
	Activa la web-cam.	function blocklyEnableWebCamStreaming() {
		if (bEnableWebCamStreaming) {
		activeWebCam = true;
		console.log("---> GoPiGo-2 Robot: ENABLE WebCam Streaming");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: ENABLE WebCam Streaming");
		}
		bDisableWebCamStreaming = false;
		}

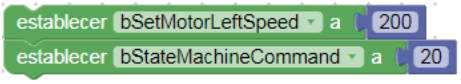
Tabla 42. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (III).

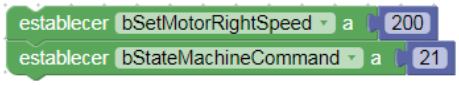
Combinación de Bloques en el entorno de laboratorios remotos (UniLabs).	**Descripción**	**Código fuente de las acciones definido en la aplicación EjsS.**

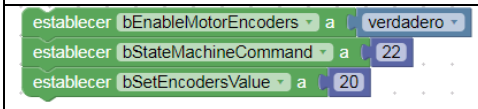
	Envía la orden de desactivación de la webcam al robot.	function blocklyDisableWebCamStreaming() {
		if (bDisableWebCamStreaming) {
		activeWebCam = false;
		console.log("---> GoPiGo-2 Robot: DISABLE WebCam Streaming");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: DISABLE WebCam Streaming");
		} bEnableWebCamStreaming = false;
		}

	Envía la orden de establecimiento del valor del puerto de ultrasonidos en el robot.	function blocklySetUltraSoundSensorPort() {
		if (bSetUltraSoundPort > 0) {
		ripclient.set(["setportultra"],[bSetUltraSoundPort]);
		console.log("---> GoPiGo-2 Robot: Set UltraSound Sensor Port");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: Set UltraSound Sensor Port");
		}
		}

	Envía la orden de obtención de la medida recogida por el sensor de ultrasonidos.	function blocklyGetUltraSoundSensorMeasure() {
		ripclient.get(['getultra'], function(result) {
		bGetUltraSoundMeasure = result[0];
		console.log("---> GoPiGo-2 Robot: Get UltraSound Sensor Measure");
		updateLogger("---> {Blockly} GoPiGo-2 Robot: Get UltraSound Sensor Measure");
		});
		}

	Envía la orden para actualizar el valor de la velocidad en el motor izquierdo.	function blocklySetLeftMotorSpeed() {
		if (bSetMotorLeftSpeed >= 0) {
		ripclient.set(["setleftspeed"],[bSetMotorLeftSpeed]);
		console.log("---> GoPiGo-2 Robot: Set Left Motor Speed"); updateLogger("---> {Blockly} GoPiGo-2 Robot: Set Left Motor Speed");
		}
		}

	Envía la orden para actualizar el valor de la velocidad en el motor derecho.	function blocklySetRightMotorSpeed() {
		if (bSetMotorRightSpeed >= 0) {
		ripclient.set(["setrightspeed"],[bSetMotorRightSpeed]); console.log("---> GoPiGo-2 Robot: Set Right Motor Speed"); updateLogger("---> {Blockly} GoPiGo-2 Robot: Set Right Motor Speed");
		}
		}

	Envía la orden de activación y establecimiento del valor de cuenta en los encoders.	function blocklySetEncoders() {
		if (bEnableMotorEncoders) {
		ripclient.set(["setm1"],[encoder1]);
		ripclient.set(["setm2"],[encoder2]);
		ripclient.set(["setenctgt"],[parseInt(bSetEncodersValue,10)]); console.log("---> GoPiGo-2 Robot: Enable Encoders. Distance in mm"); updateLogger("---> {Blockly} GoPiGo-2 Robot: Enable Encoders. Distance in mm");
		}
		}

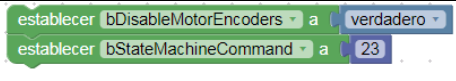
	Envía la orden de restablecimiento del valor de cuenta en los encoders.	function blocklyResetEncoders() {
		if (bDisableMotorEncoders) {
		ripclient.set(["setm1"],[encoder1]);
		ripclient.set(["setm2"],[encoder2]);
		ripclient.set(["setenctgt"],[0]);
		console.log("---> GoPiGo-2 Robot: Disable Encoders."); updateLogger("---> {Blockly} GoPiGo-2 Robot: Disable Encoders.");
		}
		}

Tabla 43. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (IV).

4.4.2.3. Actualización de la ejecución de acciones del robot GoPiGo-2 a través de la aplicación EjsS.

En el apartado anterior se ha presentado como es la estructura y la configuración de los bloques que permiten utilizar el framework Blockly desde el entorno de laboratorios UniLabs para enviar órdenes al robot GoPiGo-2. Sin embargo, no basta con tener la definición de estos bloques y el código fuente en EjsS que se ejecuta cuando se usan dichos bloques para poder interactuar con el robot, ya que además es necesario definir un método principal en la aplicación EjsS que continuamente esté comprobando el estado de las variables *"públicas"* para tomar la decisión de que método de *"acción"* ejecutar en cada caso. Es por ello que en este apartado se presenta el método principal de actualización del estado del robot. En la Figura 66 se presenta como es la configuración de la actualización de estado desde la aplicación EjsS de control.

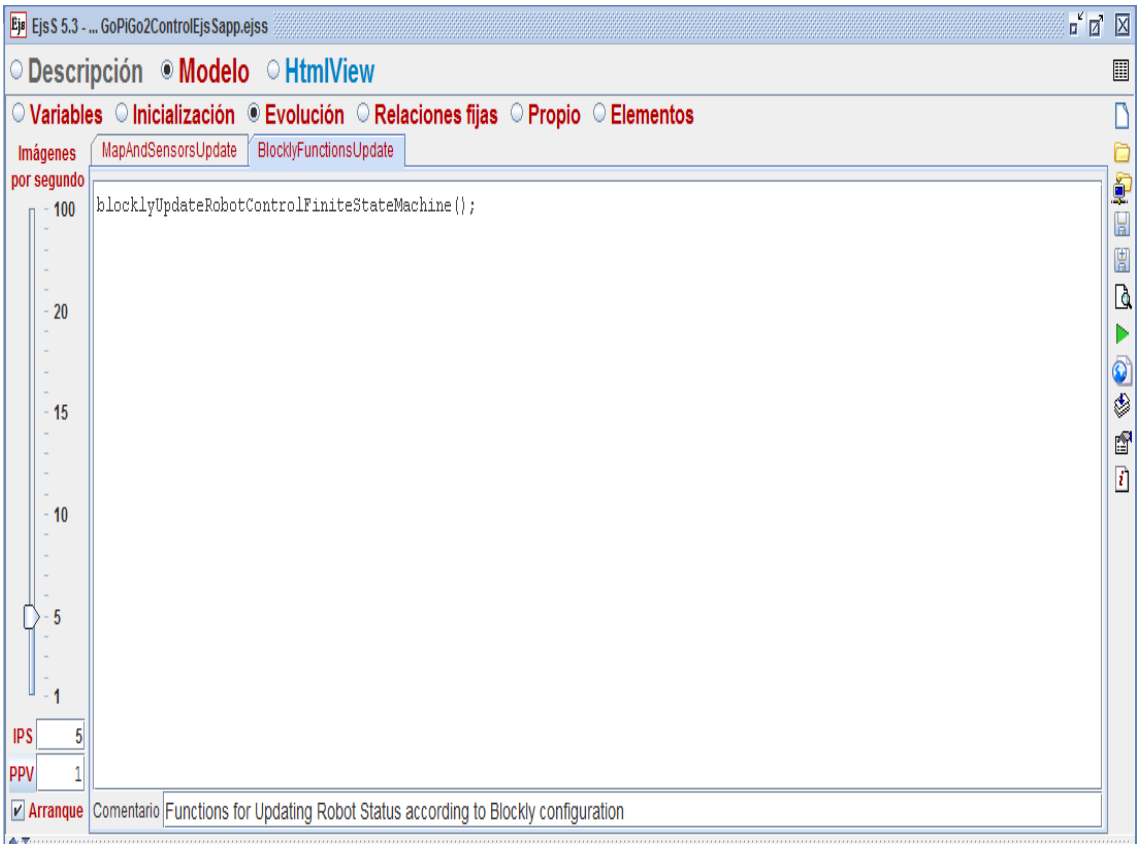


Figura 66. Configuración de la evolución del estado del robot GoPiGo-2 a través del método de actualización de estado para el framework Blockly.

Como se puede apreciar en la figura anterior para definir un método que se ejecute con cierta periodicidad basta con añadir la llamada al método (que tiene que estar definido en la pestaña *"Propio"*) y configurar el tiempo de refresco. Finalmente, se presenta el código fuente de dicho método. El cuerpo del método es una estructura de control basada en un bloque *"switch"* que captura los cambios en las variables públicas definidas en la Tabla 38.

```
// General Update Robot Control Method. This method allows updating the status of the robot control functions based
// on the state of the public variables accessible from Blockly in the Uni-Labs remote laboratories environment.
function blocklyUpdateRobotControlFiniteStateMachine() {
  switch (bStateMachineCommand) {
    case 0:
      // Stop GoPiGo-2 Robot.
      blocklyStopRobot();
      break;
    case 1:
      // Move GoPiGo-2 Robot Forward.
      blocklyMoveRobotForward();
      break;
    case 2:
      // Move GoPiGo-2 Robot Backwards.
      blocklyMoveRobotBackwards();
      break;
    case 3:
      // Move GoPiGo-2 Robot to the Right.
      blocklyMoveRobotToTheRight();
      break;
    case 4:
      // Move GoPiGo-2 Robot to the Left.
      blocklyMoveRobotToTheLeft();
      break;
    case 5:
      // Turn GoPiGo-2 Robot to the Right.
      blocklyTurnRobotToTheRight();
      break;
    case 6:
      // Turn GoPiGo-2 Robot to the Left.
      blocklyTurnRobotToTheLeft();
      break;
```

```
case 7:
    // Turn On GoPiGo-2 Robot Led-1.
    blocklyTurnOnLED1();
    break;
  case 8:
    // Turn Off GoPiGo-2 Robot Led-1.
    blocklyTurnOffLED1();
    break;
  case 9:
    // Turn On GoPiGo-2 Robot Led-2.
    blocklyTurnOnLED2();
    break;
  case 10:
    // Turn Off GoPiGo-2 Robot Led-2.
    blocklyTurnOffLED2();
    break;
  case 11:
    // Enable GoPiGo-2 Robot WebCam Streaming.
    blocklyEnableWebCamStreaming();
    break;
  case 12:
    // Disable GoPiGo-2 Robot WebCam Streaming.
    blocklyDisableWebCamStreaming();
    break;
  case 13:
    // Turn On Servo-Motor.
    blocklyTurnOnServo();
    break;
  case 14:
    // Turn Off Servo-Motor.
    blocklyTurnOffServo();
    break;
```

```
case 15:
      // Move Servo-Motor to the Right.
      blocklyMoveServoToTheRight();
      break;
   case 16:
      // Move Servo-Motor to the Left.
      blocklyMoveServoToTheLeft();
      break;
   case 17:
      // Reset Servo-Motor.
      blocklyResetServoEngine();
      break;
   case 18:
      // Set UltraSound Sensor Port.
      blocklySetUltraSoundSensorPort();
      break;
   case 19:
      // Get UltraSound Sensor Measure.
      blocklyGetUltraSoundSensorMeasure();
      break;
   case 20:
      // Set Left-Motor Speed.
      blocklySetLeftMotorSpeed();
      break;
   case 21:
      // Set Right-Motor Speed.
      blocklySetRightMotorSpeed();
      break;
   case 22:
      // Enable and Set Encoders Value. Distance in mm.
      blocklySetEncoders();
      break;
```

```
case 23:
      // Disable and Reset Encoders Value. Distance in mm.
      blocklyResetEncoders();
      break;
   case 24:
      // Stablish Connection With the Robot GoPiGo-2.
      blocklyStablishConnectionWithTheRobot();
      break;
   case 25:
      // Close Connection With the Robot GoPiGo-2.
      blocklyCloseConnectionWithTheRobot();
      break;
   default:
      break;
  }
}
```

A la vista del código anterior, se puede apreciar que en el caso de tener que realizar modificaciones en el método de actualización de estado bastaría con añadir tantas condiciones nuevas en el bloque *"switch"* como estados nuevos se tuviesen que tratar por lo que el mantenimiento de este software de control no es excesivamente complejo. Por otro lado, se puede apreciar en el código anterior la definición de dos acciones en las que no se había hecho hincapié hasta ahora. Se trata de las acciones *"bEstablishConnectionWithTheRobot"* y *"bCloseConnectionWithTheRobot"* que permiten establecer y cerrar la conexión con el robot respectivamente.

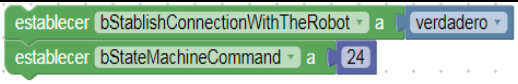
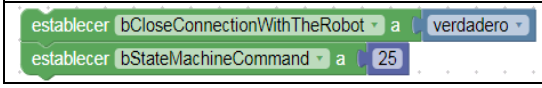
Combinación de Bloques en el entorno de laboratorios remotos (UniLabs).	**Descripción**	**Código fuente de las acciones definido en la aplicación EjsS.**
	Envía la orden de establecimiento de conexión con el robot.	blocklyStablishConnectionWithTheRobot()
	Envía la orden de cierre de conexión con el robot	blocklyCloseConnectionWithTheRobot()

Tabla 44. Relación entre los bloques del entorno de laboratorios remotos y las acciones realizadas en la aplicación EjsS (V).

Estos métodos pueden utilizarse sólo si es necesario obtener en el programa definido con Blockly una actualización constate de las medidas obtenidas con los sensores (en este caso con el sensor de ultrasonidos). En caso contrario no es necesario su uso para operar con el robot.

Capítulo 5: Pruebas Realizadas.

En este capítulo se presentan las pruebas realizadas con el robot GoPiGo-2 y el software de control desarrollado e integrado en el entorno de laboratorios remotos UniLabs. El objetivo de estas pruebas es el de validar el software desarrollado así como encontrar limitaciones al mismo para proponer futuras líneas de investigación.

Las pruebas que se presentan en este capítulo se han distribuido de la siguiente manera:

- **Pruebas de control manual:** este tipo de pruebas consisten en testear el interfaz web de la aplicación de control EjsS y observar el comportamiento del robot GoPiGo-2.

- **Pruebas de control automático:** estas pruebas consisten en testear el comportamiento del robot GoPiGo-2 cuando desde la aplicación de control EjsS se envía un programa o un script de control para su ejecución.

- **Pruebas de control mediante Blockly:** estas pruebas consisten en definir un programa mediante bloques del framework Blocky para comprobar el comportamiento del robot en base a dicho programa de control.

5.1. Pruebas de Control Manual.

En esta sección se presentan las pruebas de control manual realizadas con la aplicación web EjsS y el robot GoPiGo-2. El resultado de las pruebas se puede consultar en el siguiente video: https://www.dropbox.com/s/accxh969h4g65vz/TFM-Jorge-Dur%C3%A1n-Murillas-VIDEO-DEMO1-Control-Manual.mp4?dl=0 .

Como se puede apreciar las pruebas han consistido en probar los controles básicos del robot. En la Figura 67 se puede apreciar una captura del video enlazado con anterioridad.

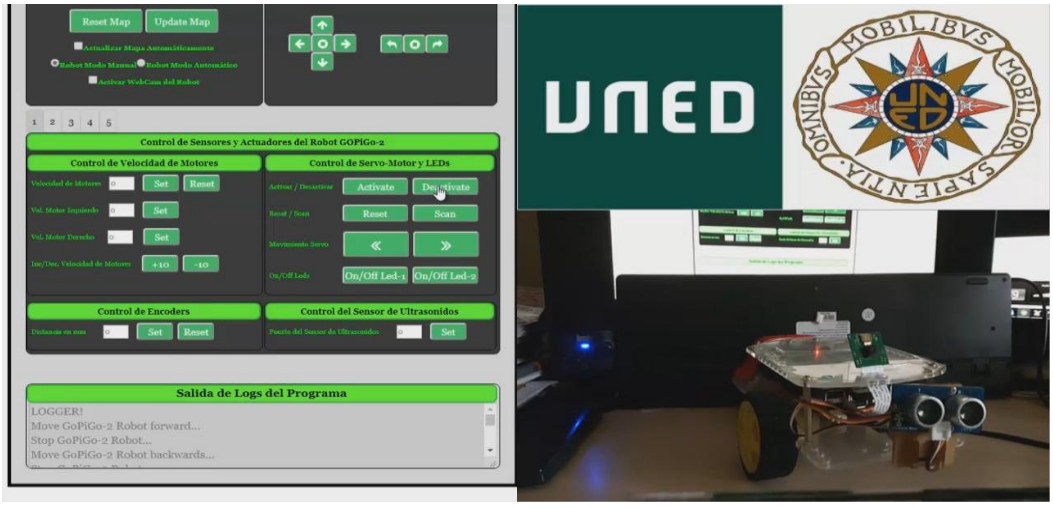


Figura 67. Captura del video de demostración 1.

En el video de demostración se puede apreciar como el robot GoPiGo-2 responde de forma satisfactoria a los siguientes controles manuales:

- Establecimiento de la conexión entre la aplicación EjsS y el servidor RIP instalado en el robot.
- Movimiento del robot mediante el empleo de los controles básicos: avanzar, retroceder, mover a la izquierda, mover a la derecha, girar a la izquierda, girar a la derecha, detener movimiento, etc.
- Modificación de la velocidad de los motores del robot.
- Activación y desactivación del movimiento del servo-motor.
- Movimiento del sensor de ultrasonidos mediante el servo-motor hacia la derecha y hacia la izquierda.
- Encendido y apagado de los LEDs integrados en el robot GoPiGo-2.
- Movimiento del robot en varias direcciones una distancia concreta mediante el establecimiento de la cuenta en los encoders.
- Establecimiento del puerto del sensor de ultrasonidos.
- Actualización automática y manual de las lecturas procedentes del sensor de ultrasonidos, estado de la batería y velocidad de los motores.
- Inicialización del mapa de obstáculos.
- Encendido y apagado de la webcam acoplada al robot GoPiGo-2.
- Interacción con el editor de código Python y los controles asociados.
- Actualización automática de las trazas del programa.

5.2. Pruebas de Control Automático.

En esta sección se presentan las pruebas de control automáticas realizadas con la aplicación web EjsS y el robot GoPiGo-2. El resultado de las pruebas se puede consultar en el siguiente video: https://www.dropbox.com/s/m2k50wga67coox5/TFM-Jorge-Dur%C3%A1n-Murillas-VIDEO-DEMO2-Control-Autom%C3%A1tico.mp4?dl=0 .

Como se puede apreciar en el video enlazado con anterioridad las pruebas se han dividido en dos partes:

- La primera parte se basa en el envío al robot de un programa *Python* que hace uso exclusivo del API nativa de programación del robot GoPiGo-2 para definir una función de control que permite al robot avanzar en una única dirección y un distancia determinada (definida por la cuenta de los encoders) para detenerse y realizar un escaneo del área con el sensor de ultrasonidos. En caso de que se detecte un obstáculo a una distancia inferior a un umbral marcado (de 20 cm) el robot se detiene definitivamente.

- La segunda parte se basa en el envío al robot de un script *Python* que hace uso de las facilidades proporcionadas por la clase *GoPiGoContext.py* para definir una función de control con navegación y evitación de obstáculos (Figura 68). Como se puede apreciar en esta prueba se hace uso del panel que contiene el mapa de obstáculos para visualizar la posición del robot así como los obstáculos que es capaz de detectar mediante el uso del sensor de ultrasonidos.

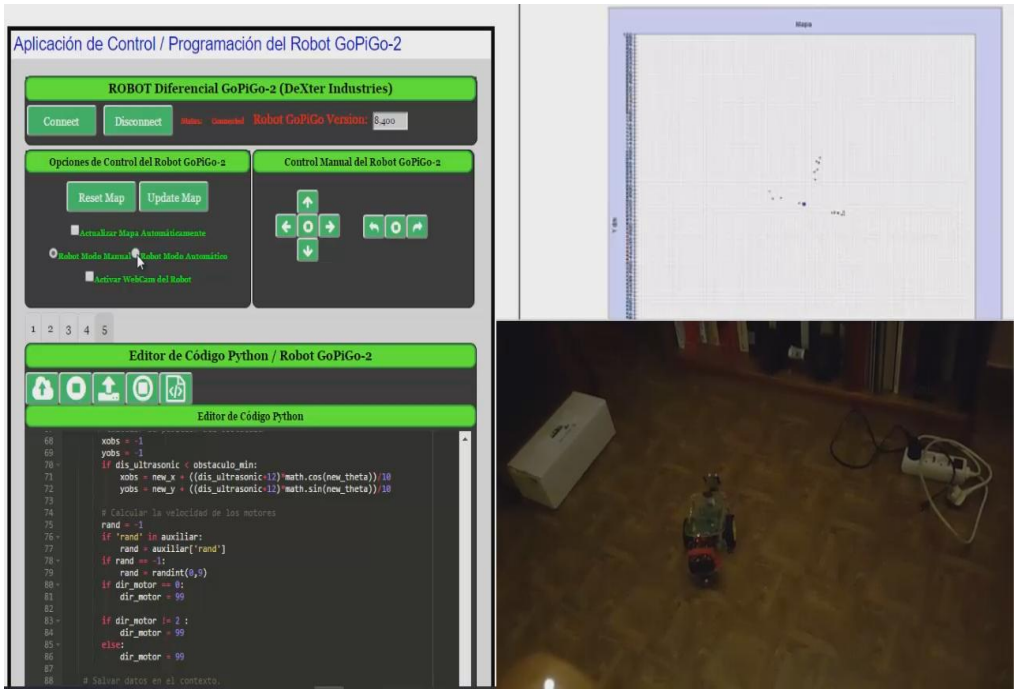


Figura 68. Test de control automático del robot GoPiGo-2 mediante el uso de un script Python.

5.3. Pruebas de Control Mediante Blockly.

En esta sección se presentan las pruebas de control mediante el uso del framework Blockly. El resultado de las pruebas se puede consultar en el siguiente video: https://www.dropbox.com/s/ygcwk5ob8jsfwti/TFM-Jorge-Dur%C3%A1n-Murillas-VIDEO-DEMO3-Control-Autom%C3%A1tico-BLOCKLY.mp4?dl=0 .

En las siguientes tablas (Tabla 45, Tabla 46 y Tabla 47) se puede encontrar la relación de los programas de control testeados en el video así como una descripción de las acciones realizadas.

Programa definido con el framework Blockly	Descripción del programa.
	Programa básico de control que permite testear todas las combinaciones posibles de movimiento del robot GoPiGo-2. Entre movimiento y movimiento se realiza una espera de 1 segundo. Nombre del archivo de control: **finaltest-Blockly-all-basic-moves-and-turns.blk**
	Programa básico de control que permite testear el encendido y apagado de los LEDs integrados en el robot GoPiGo-2. Entre encendido y apagado hay una espera de 2 segundos. Nombre del archivo de control: **finaltest-Blockly-LEDs-control.blk**

Tabla 45. Programas de control definidos en Blockly y probados con el robot GoPiGo-2 (I).

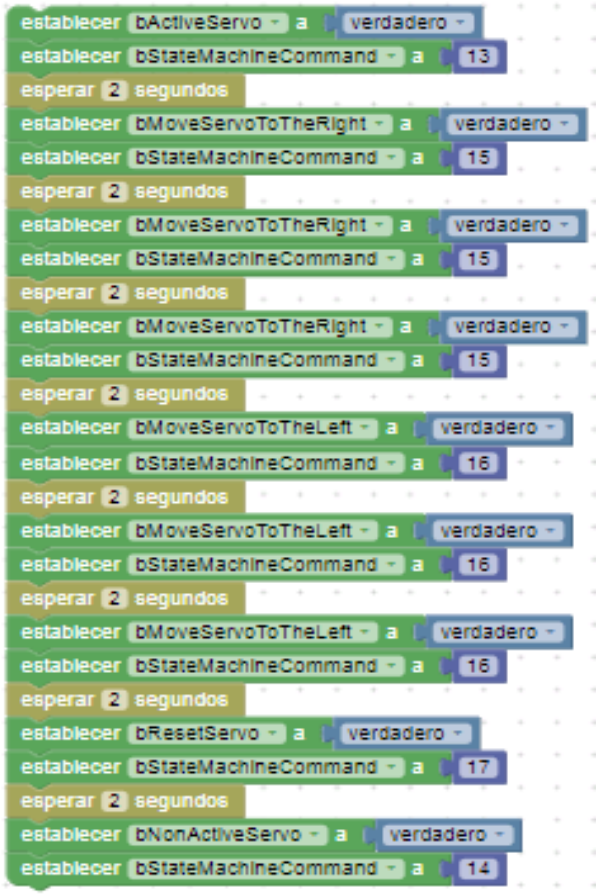
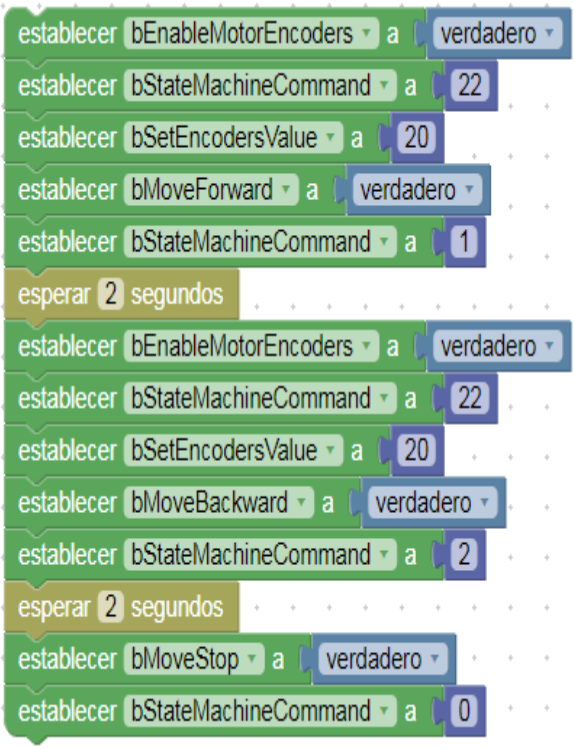
Programa definido con el framework Blockly	Descripción del programa.
	Programa básico de control que permite testear todas las combinaciones posibles de movimiento del servo-motor instalado en el robot GoPiGo-2. Entre movimiento y movimiento se realiza una espera de 2 segundos. Nombre del archivo de control: **finaltest-Blockly-control-moves-Servo-motor.blk**
	Programa básico de control que permite testear el movimiento del robot GoPiGo-2 mediante la definición de la distancia a recorrer (en milímetros) como cuentas de los encoders. Entre movimiento y movimiento hay una espera de 2 segundos. Nombre del archivo de control: **finaltest-Blockly-Moves-With-Encoders-distance.blk**

Tabla 46. Programas de control definidos en Blockly y probados con el robot GoPiGo-2 (II).

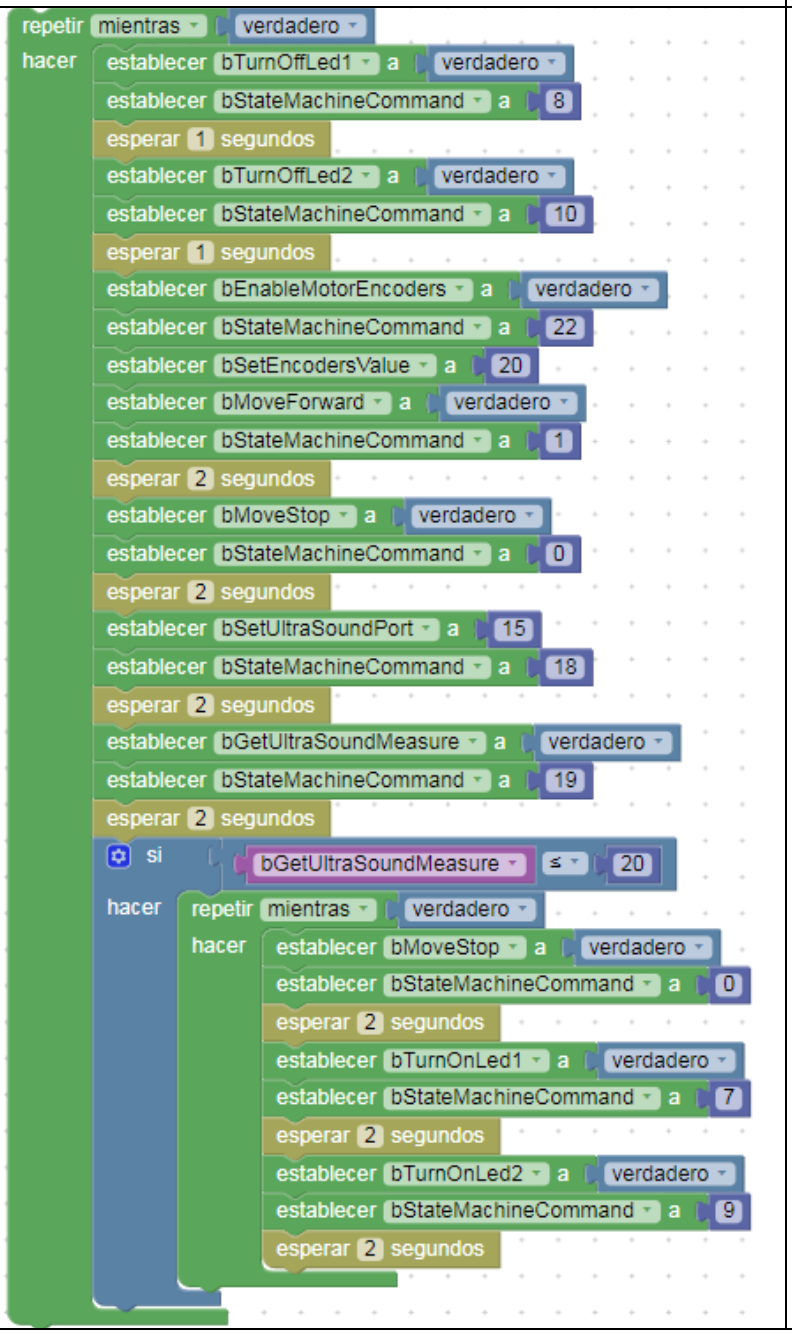
Programa definido con el framework Blockly	Descripción del programa.
	Programa de control que indica al robot avanzar mientras la lectura tomada por el sensor de ultrasonidos sea superior a un umbral. En caso de que dicha lectura sea inferior al umbral (de 20 cm) el robot se detiene y mantiene encendidos los dos LEDs del frontal del robot. Nombre del archivo de control: **finaltest-Blockly-control-fwd-avoid-obstacles-leds-loop.blk**

Tabla 47. Programas de control definidos en Blockly y probados con el robot GoPiGo-2 (III).

Capítulo 6: Conclusiones y Trabajos Futuros.

En los videos de pruebas presentados en el capítulo anterior se ha podido constatar que el correcto funcionamiento del robot tanto en modo manual como en modo automático.

En el modo manual la respuesta del robot a las peticiones desde la aplicación EjsS es prácticamente inmediata. En el en modo automático se puede comprobar como el robot se mueve por la zona de trabajo, intentando esquivar obstáculos y construyendo el mapa. Finalmente, cuando se define un programa de control en Blockly, se puede apreciar en las pruebas realizadas que la respuesta a este tipo de programas es correcta y prácticamente inmediata a pesar de tener la aplicación EjsS desplegada en el entorno de laboratorios remotos.

Por otro lado, se ha podido comprobar que la actualización en tiempo real del mapa que muestra los obstáculos y la posición del robot también ha resultado correcta, apreciándose en el vídeo demostrativo como a medida que el robot se mueve el mapa se actualiza correctamente.

El objetivo era demostrar el funcionamiento de la aplicación EjsS desplegada en el entorno de laboratorios remotos UniLabs y como el usuario podría crear un programa para que el robot lo ejecutase de manera autónoma. En ese sentido la prueba ha resultado satisfactoria.

La comunicación entre el servidor RIP y la aplicación EjsS a través de una red Wi-Fi también ha funcionado correctamente, aunque en el vídeo no se pueda apreciar, el robot se encontraba en otra habitación distinta a la cual estaba el pc cliente.

6.1. Limitaciones en el robot.

Las pruebas realizadas con el robot no sólo han constatado los casos de éxito sino que también han puesto de manifiesto algunas limitaciones. Las principales limitaciones detectadas son aquellas debidas a la precisión de los sensores y actuadores del robot, sobre todo en la calidad de los materiales. Las limitaciones que se han encontrado son las siguientes:

- Se han detectado problemas de odometría cuando las ruedas se deslizan por la superficie alterando así el cálculo de la posición del robot.
- El número de sensores de ultrasonidos utilizado, ya que el kit dispone sólo de un sensor de ultrasonidos, 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 de forma simultánea.

- El sensor de ultrasonidos tiene baja precisión y en ocasiones presenta falsos positivos.
- A pesar de configurar ambos motores con la misma velocidad puede suceder que un motor gire a más velocidad que el otro, distorsionando con ello la función de navegación. Sin embargo la lectura de los encoders se mantiene coherente.

Muchas de las limitaciones presentadas son salvables por parte del usuario a través de la lógica del programa de control que sea implementado.

6.2. Líneas de trabajo futuras.

Una de las finalidades de este trabajo es crear una base a partir de la cual se pueda crecer en el futuro. Es por ello que a continuación se hacen una serie de recomendaciones en forma de posibles líneas de actuación que mejoren o añadan nuevas funcionalidades al trabajo realizado:

- Se puede dotar al robot GoPiGo-2 de un mayor número de sensores de ultrasonidos en su chasis para mejorar la calidad de las medidas y mejorar el algoritmo de mapeo del entorno. En este sentido, el robot GoPiGo-2 soporta también la conexión mediante GPS o el montaje de acelerómetros lo que permitiría mejorar el cálculo de su posición evitando problemas detectados relacionados con la odometría.

- En este trabajo se ha incluido una cámara web que permite visualizar el entorno desde la aplicación EjsS. Una nueva línea de trabajo podría ser utilizar dicha cámara para desarrollar algoritmos de visión por computador.

- Como se ha visto a lo largo de este trabajo la integración con el framework Blockly ha sido satisfactoria. Tal y como se propuso en el capítulo correspondiente se podría extender la versión del framework Blockly instalada en el entorno de laboratorios remotos para crear una librería de control del robot GoPiGo-2 que tome como base la implementación realizada y basada en el código EjsS y de los bloques de laboratorio utilizados.

- En el momento en el que se ha desarrollado este trabajo han aparecido en el mercado otros robots del fabricante *DeXter Indsutries* (incluido el nuevo GoPiGo-3) que usan un API de programación muy similar a la del robot GoPiGo-2 por lo que para mejorar la oferta de prácticas con robots autónomos se podría pensar en ampliar el stock de robots disponibles para realizar prácticas desde la plataforma UniLabs y realizar su integración tomando como base el trabajo realizado en este proyecto.

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

[7] 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/.

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

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

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

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

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

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

[14] 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
IP	Internet Protocol
LED	Light-Emitting Diode
MVC	Model-View-Controller
PID	Proportional-Integral-Derivative Controller
REST	Representational State Transfer
RIP	Remote Interoperability Protocol
SBC	Single-Board Computer
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
Wifi	Wireless Fidelity

ANEXO A.

Montaje del cuerpo de la plataforma robótica.

Al abrir la caja deben estar presentes los siguientes elementos:

- **Placa Raspberry Pi**: Se trata de la placa Raspberry Pi 3 Model B, la cual se detallará 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, y sus características se detallarán más adelante. Esta placa sirve de interfaz entre la placa Raspberry Pi y los diferentes dispositivos que se pueden conectar al robot.
- **Sensor de ultrasonidos:** el kit incorpora un sensor de ultrasonidos, que puede ser montando sobre un motor servo para poder girarlo. Permite detectar cualquier obstáculo y facilita la navegación del robot.
- **Servomotor**: elemento sobre el cual se puede montar el sensor de ultrasonido para poder girarlo. También permite montar sobre él una cámara.
- **Dos motores:** elementos en los cuales se acoplarán las ruedas del robot.
- **Dos encoders**: estos elementos 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**: 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.
- **Piezas de tornillería y plástico**: el kit incorpora toda la tornillería necesaria para el ensamblaje de la plataforma robótica así como piezas de plástico acrílico de diferentes tamaños que sirven de base para el robot y para la disposición de otros elementos.

Las piezas de plástico acrílico incorporan una capa protectora para evitar que se rayen. Como se puede apreciar en la siguiente figura las piezas de plástico acrílico son transparentes.



Figura 69. Componentes incluidos en la caja del robot GoPiGo-2 de Dexter Industries.

Esta sección se centra en el montaje del cuerpo de la plataforma robótica y se corresponde con las piezas de plástico acrílico más grandes incluidos en la caja. La forma más sencilla de averiguar cómo encaja la base de plástico acrílico con la placa GoPiGo-2 es colocando la placa sobre la base de plástico. En este caso se deberían apreciar dos agujeros alineados entre ambos elementos. Una vez localizados se pueden emplear los espaciadores hexagonales en ellos para recordar cual es la dirección correcta de ensamblaje (PASO-1).

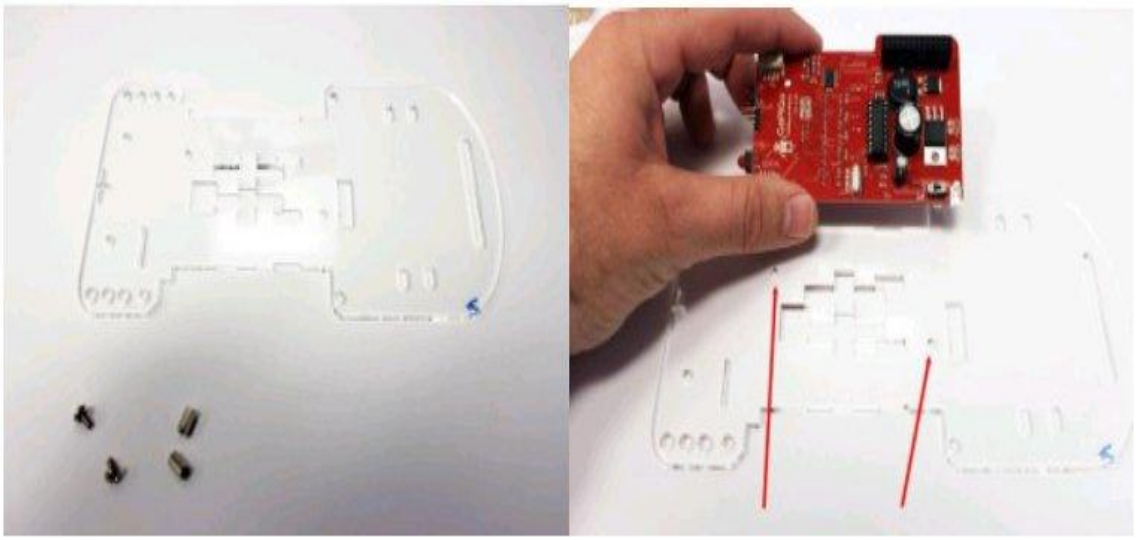


Figura 70. Montaje del cuerpo de la plataforma robótica. Paso 1.



Figura 71. Montaje del cuerpo de la plataforma robótica. Fijación de la placa en la base de plástico. Paso 1.

A continuación, se debe proceder a conectar los motores. Para conectar los motores al chasis, primero hay que identificar los 4 tornillos largos incluidos en el kit, posteriormente hay que prepararlos con sus respectivas tuercas, y luego usar las piezas con forma de "T" de plástico acrílico incluidas en el kit para unir los motores. Hay que tener cuidado en este punto al enroscar el motor en su lugar ya que si se aprietan demasiado los tornillos se puedes agrietar la pieza de plástico acrílico (PASO-2).

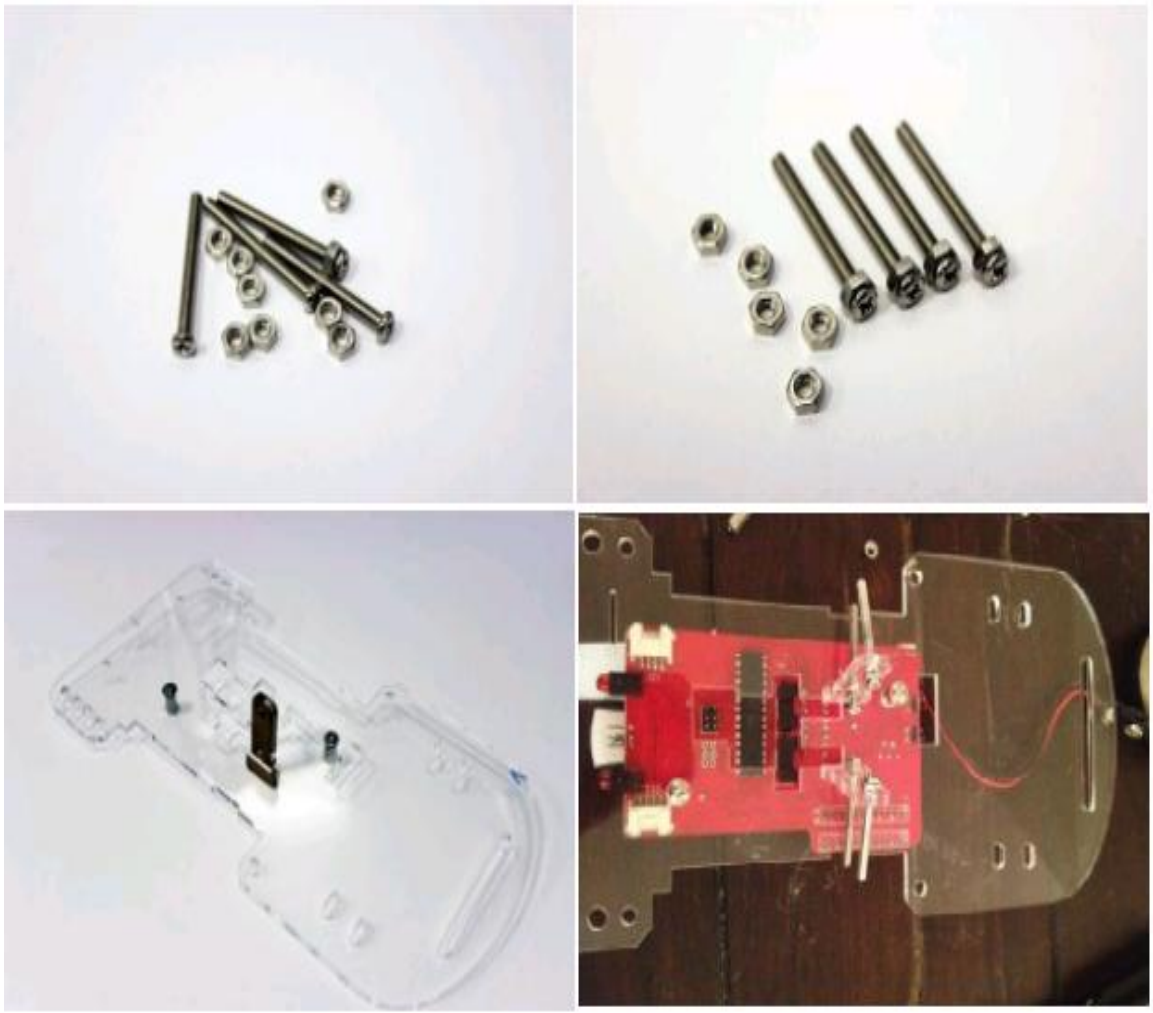


Figura 72. Montaje del cuerpo de la plataforma robótica. Fijación de la tornillería de los motores. Paso 2.

Para realizar el paso 2 reflejado en la figura 13, se recomienda lo siguiente:

- Preparar los tornillos colocando una tuerca en cada tornillo.
- Encontrar las piezas con forma “T” de plástico acrílico en el kit. Debería haber al menos cuatro.
- Insertar una T desde la parte superior del chasis. Después de que la T haya sido deslizada a través del chasis, deslizar los tornillos a través de la pieza con forma de T. Los tornillos deberían quedar en dirección de adentro hacia afuera. Las cabezas de los tornillos debe estar mirando hacia afuera.
- Conectar el motor deslizándolo hacia un lado en los tornillos. Deslizar el segundo motor.
- Finalmente, enroscar una tuerca en cada tornillo para que quede fija la parte exterior del chasis.

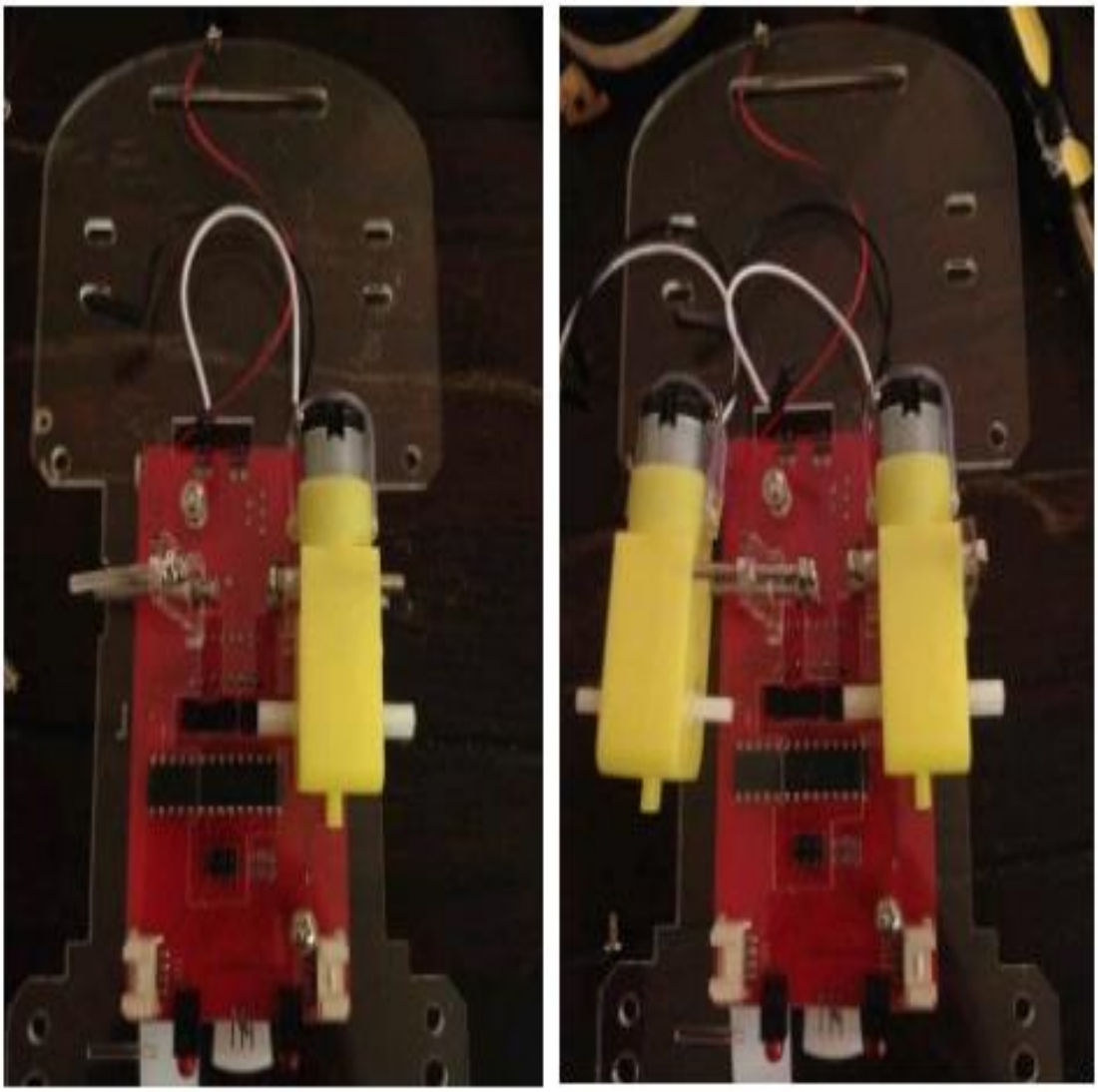


Figura 73. Montaje del cuerpo de la plataforma robótica. Vista de los motores conectados al chasis. Paso 2.

Como se puede apreciar en la figura anterior es necesario conectar los motores a los encoders de la placa GoPiGo-2. Los encoders son necesarios ya que proporcionan información sobre la dirección y velocidad a los motores. Para su correcta conexión, es necesario conectar los cables negros al conjunto de pines más cercanos al motor (en cada caso, ya sea derecho o izquierdo).

A continuación se va a proceder a fijar los motores al chasis del cuerpo del robot para posteriormente acoplar las ruedas internas de material acrílico. Este paso es muy simple tal y como se puede apreciar en la siguiente figura (PASO-3).

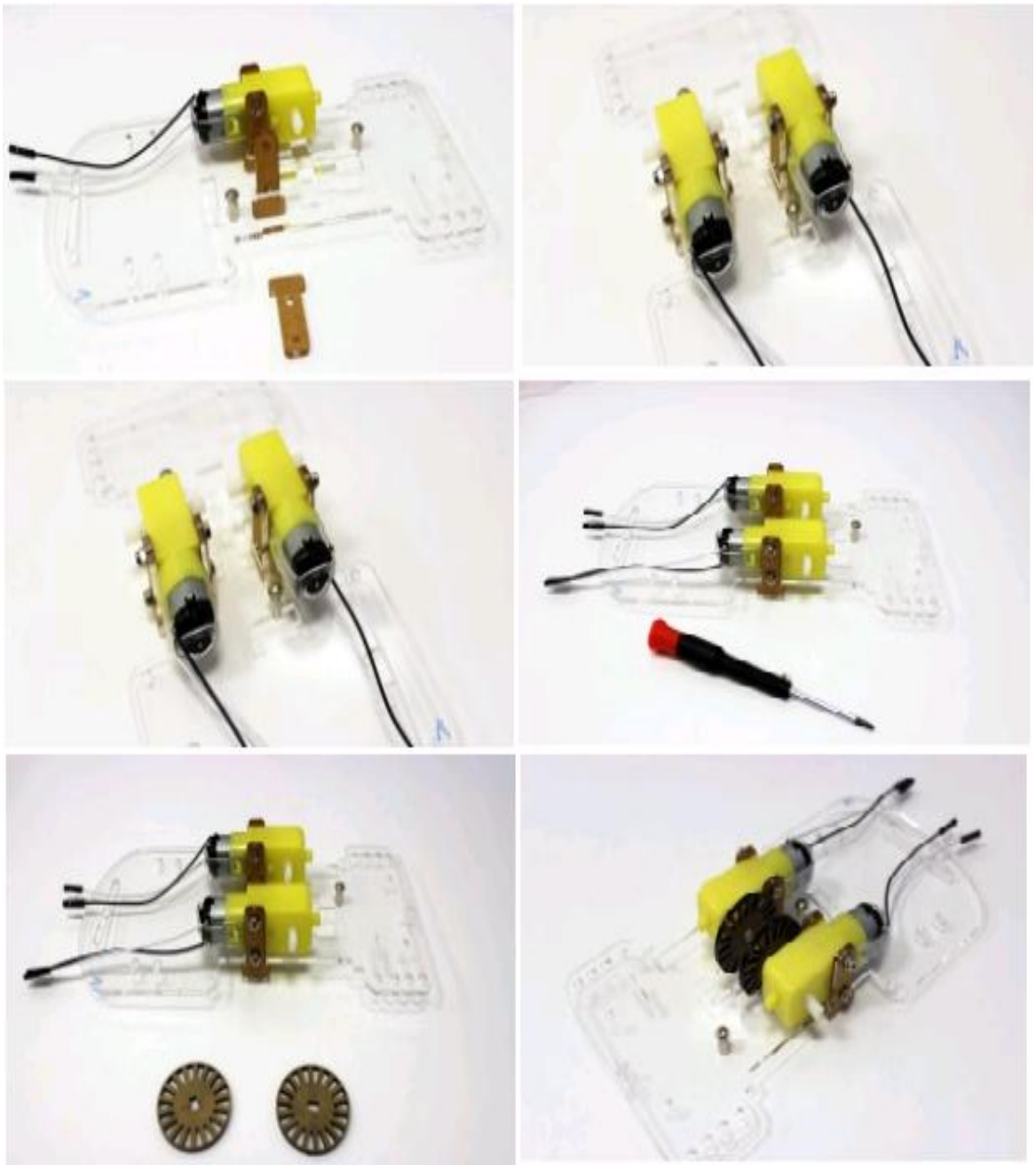


Figura 74. Montaje del cuerpo de la plataforma robótica. Conexión de las ruedas internas. Paso 3.

El robot GoPiGo-2 viene con cuatro tipos de soportes hexagonales. Ya se ha utilizado el soporte más corto para montar la placa GoPiGo-2. El más largo de los soportes es para el montaje de la rueda giratoria tal y como se puede apreciar en la siguiente figura (PASO-4).

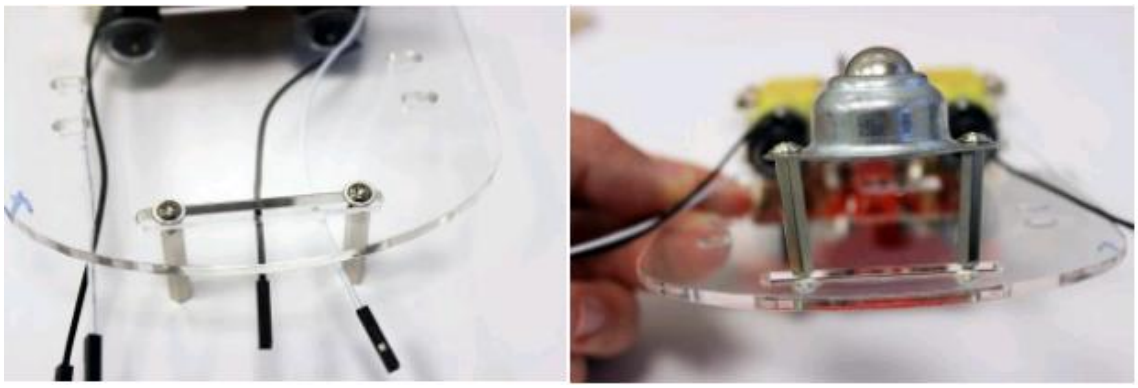


Figura 75. Montaje del cuerpo de la plataforma robótica. Montaje de la rueda giratoria. Paso 4.

A continuación, se debe acoplar el adaptador de baterías / pilas. Para ello se deben utilizar dos espaciadores hexagonales como los usados con anterioridad para acoplar el adaptador atornillándolos al chasis del robot GoPiGo-2. Primero, se conectarán estos espaciadores al GoPiGo. Finalmente, ajustar la correa al adaptador de baterías (PASO-5).

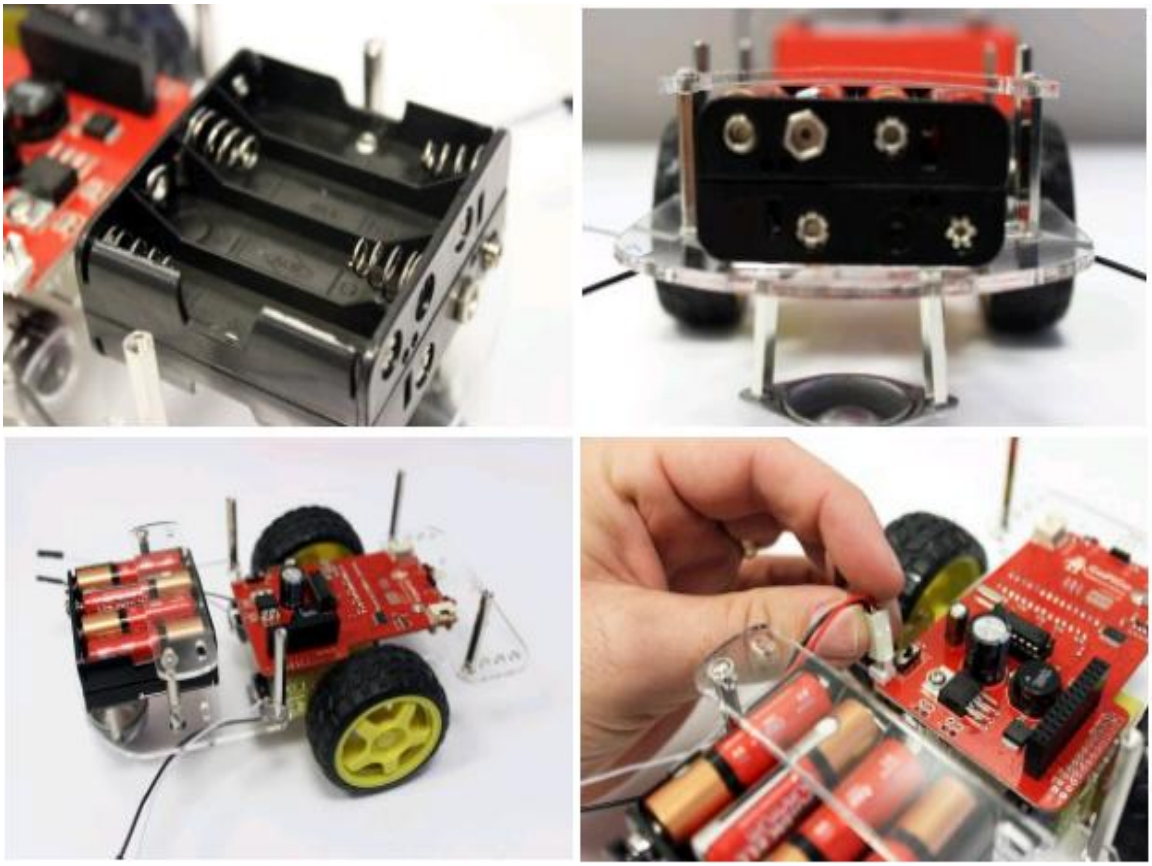


Figura 76. Montaje del cuerpo de la plataforma robótica. Montaje del adaptador de baterías. Paso 5.

A continuación, se debe proceder a colocar las ruedas. Para ellos es necesario deslizarlas en la posición correcta sin forzar la pieza para no estropearla (Paso-6).

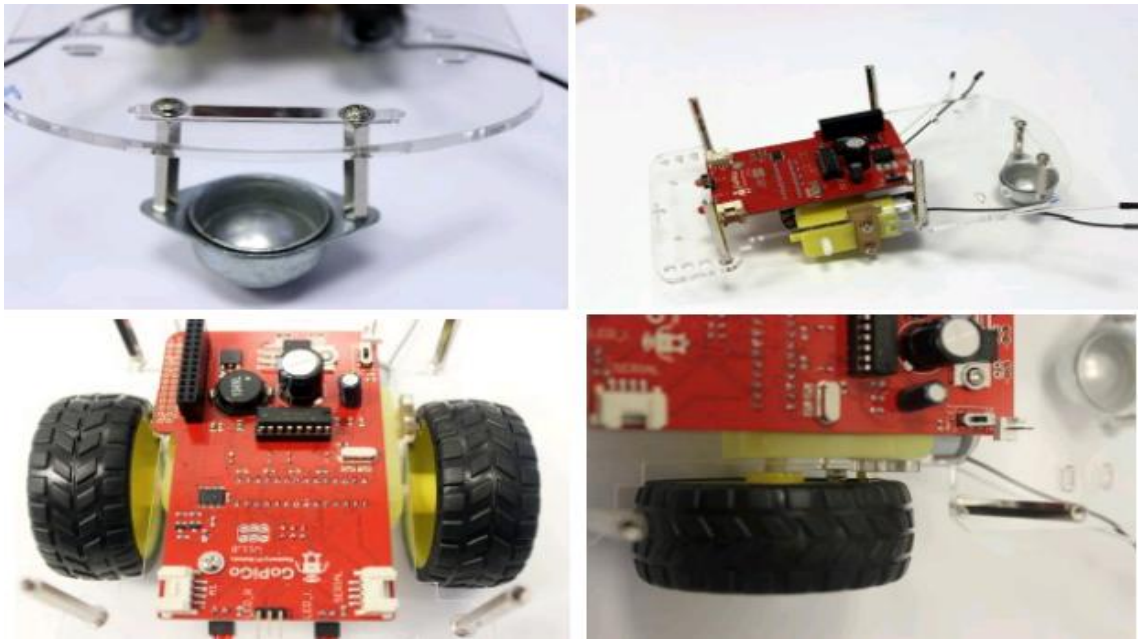


Figura 77. Montaje del cuerpo de la plataforma robótica. Montaje de las ruedas. Paso 6.

Posteriormente se procederá a conectar la placa Raspberry Pi al GoPiGo-2. Es necesario colocar la placa Raspberry Pi sobre el GoPiGo conectando los pines del bus GPIO en la palca Raspberry Pi. Posteriormente se debe colocar la placa protectora superior de plástico acrílico. La placa superior, colocar los cuatro espaciadores hexagonales proporcionados, y atornillarlos (PASO-7).

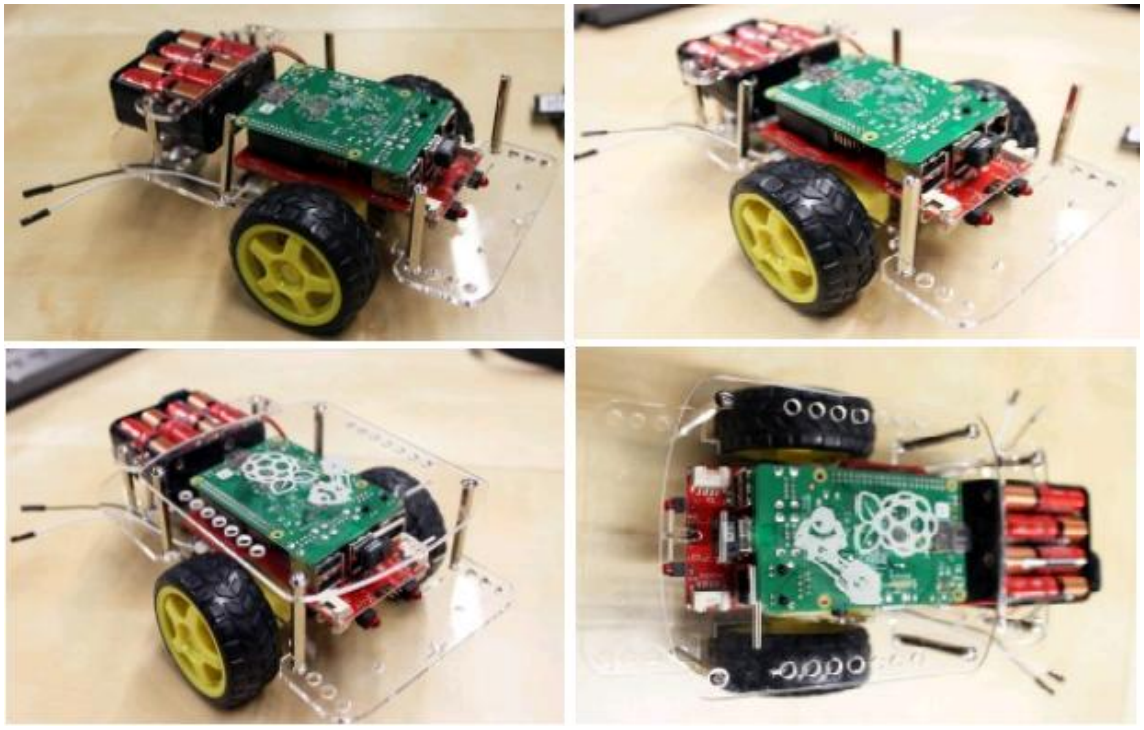


Figura 78. Montaje del cuerpo de la plataforma robótica. Conexión de la placa Raspberry Pi. Paso 7.

Montaje del cuerpo de la plataforma robótica. Acoplar el Servo.

En este sub-apartado se va a proceder a explicar cómo montar el servo en la plataforma robótica. En la siguiente figura se pueden apreciar los componentes necesarios para el montaje incluidos en el kit del robot.



Figura 79. Elementos necesarios para el montaje del servo en la plataforma robótica.

Primero colocar el tornillo largo suministrado con el paquete del servo a través de los dos orificios en el extremo de las piezas con forma de "T". Deslizar el servo en el eje. Conectar el labio del servo al servo con un tornillo pequeño. Abrir los agujeros en el labio del servo atornillándolo con uno de los tornillos grandes incluidos (Paso-8).

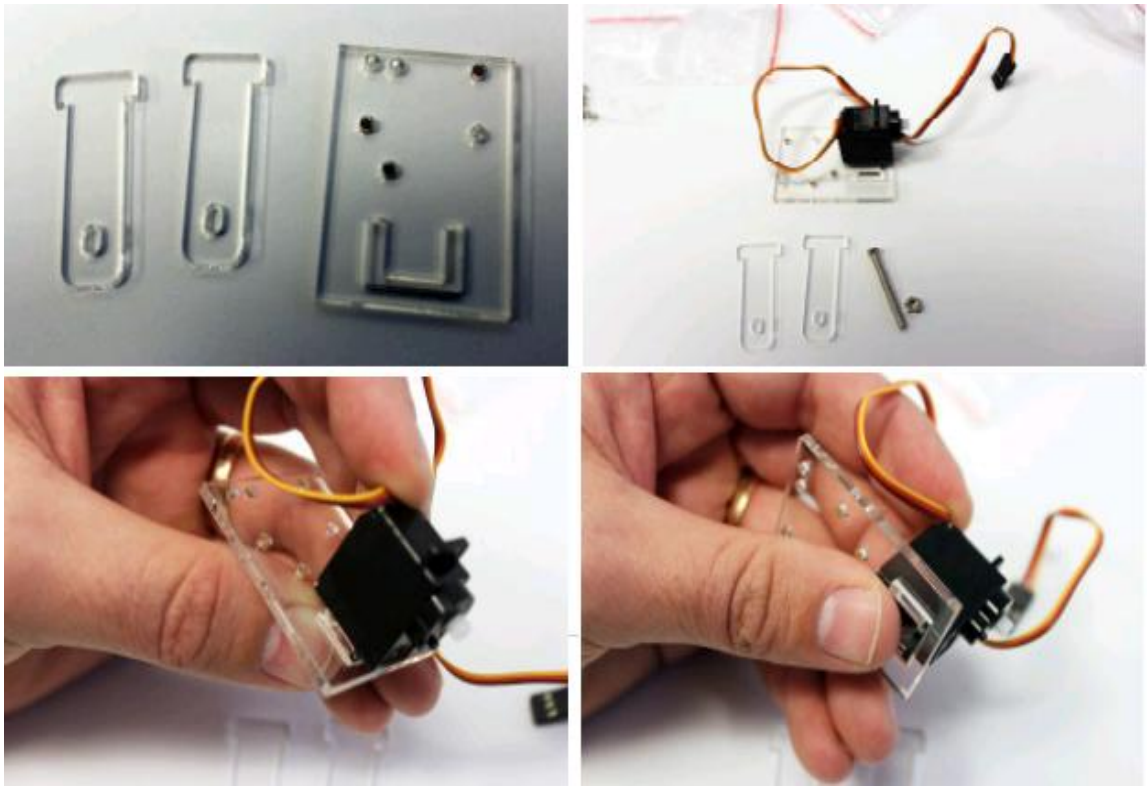


Figura 80. Montaje del servo acoplado a la plataforma robótica. Paso 8.

Finalmente conectar la conexión eléctrica del servo a la placa GoPiGo-2 tal y como se puede apreciar en la siguiente figura (Paso-9).

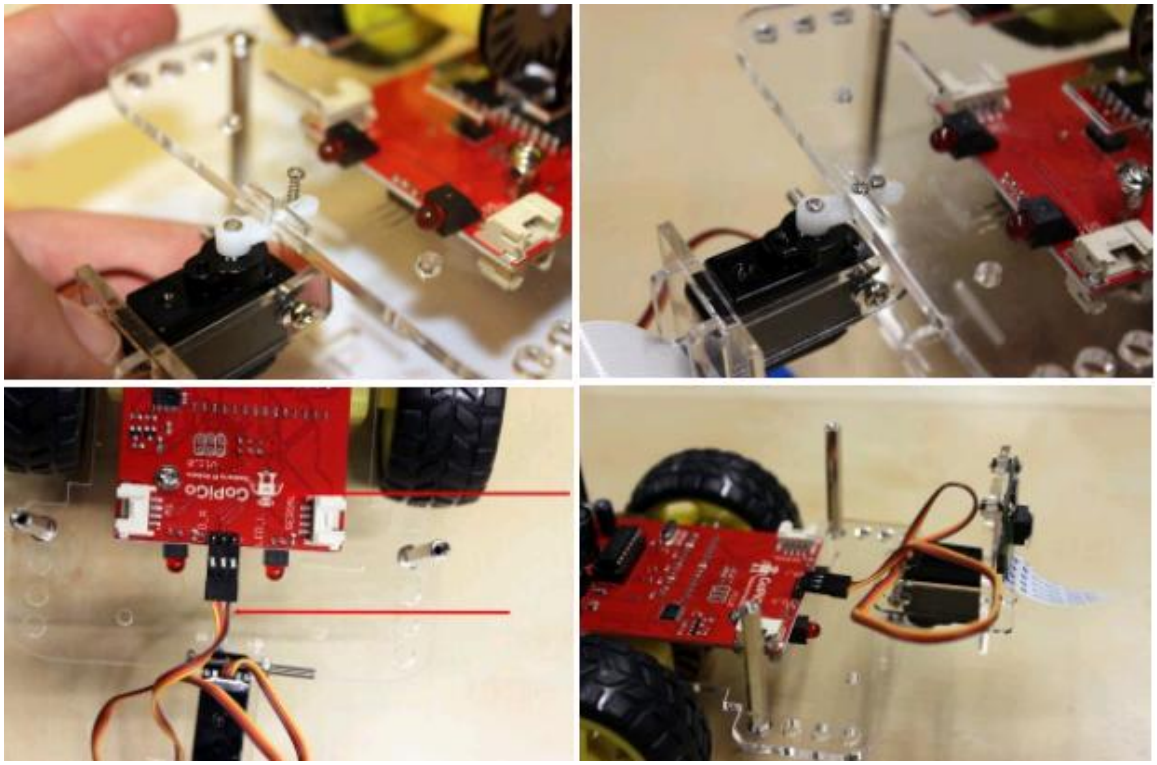


Figura 81. Montaje del servo acoplado a la plataforma robótica. Conexión eléctrica a la palca GoPiGo-2. Paso 9.

Montaje del cuerpo de la plataforma robótica. Acoplar el Sensor de Ultrasonidos.

El sensor de ultrasonidos se puede conectar al servo con dos de los tornillos que están incluidos en el paquete de piezas del servo. El sensor de ultrasonidos tiene dos orificios para tornillos para permitir su montaje (Paso-10).

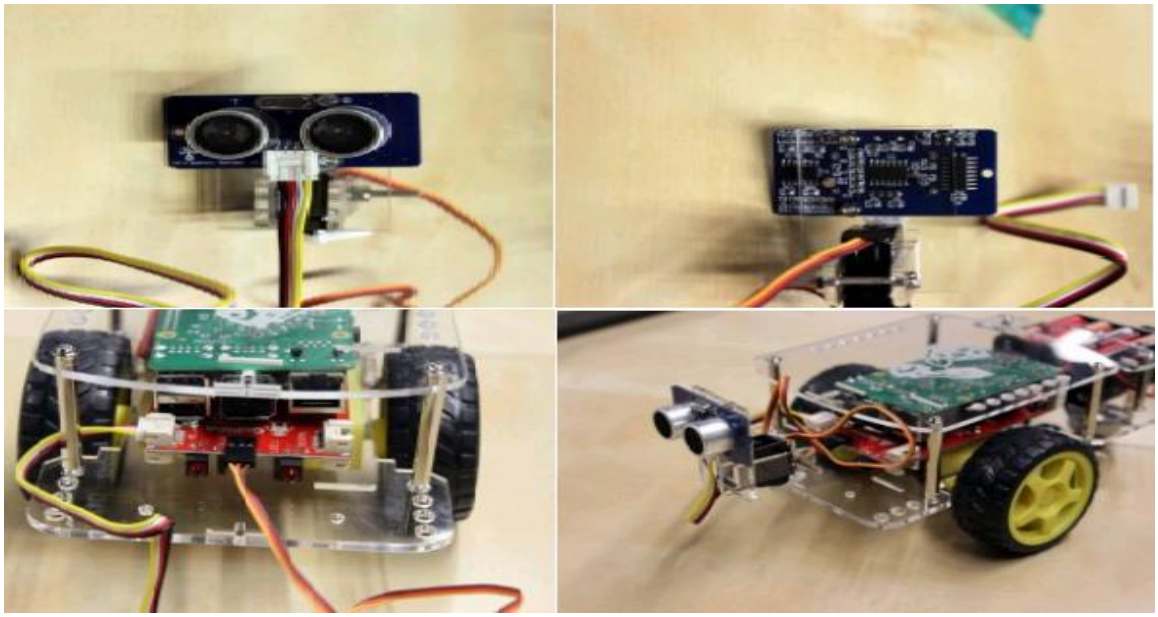


Figura 82. Montaje del sensor de ultrasonidos acoplado a la plataforma robótica. Conexión eléctrica a la palca GoPiGo-2. Paso 10.

Montaje del cuerpo de la plataforma robótica. Instalación de la cámara modelo *Raspberry Pi Camera Board*.

La instalación de la cámara en el robot GoPiGo-2 es relativamente sencilla. Para ello es necesario acoplar el extremo del cable de video al conector de video de la placa Raspberry Pi.

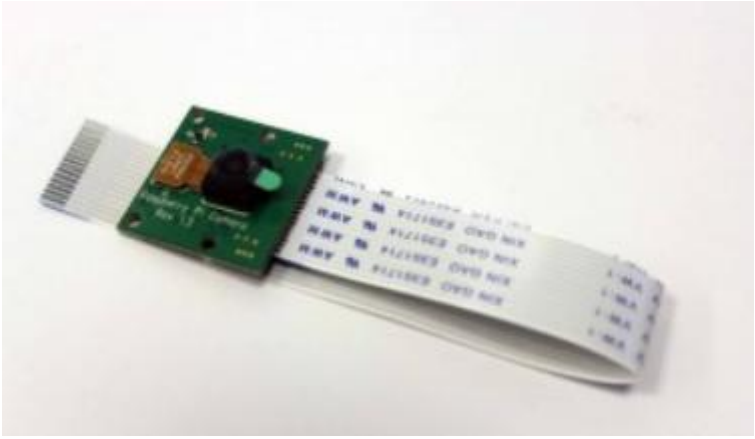


Figura 83. Cámara del robot GoPiGo-2. Modelo "Raspberry Pi Camera Board".

El conector de video de la placa Raspberry Pi se encuentra situado justo detrás del conector Ethernet. Una vez conectado el cable de video (y dado que se ha utilizado en el apartado anterior se ha utilizado el servo-motor para acoplar el sensor de ultrasonidos) se debe pasar el cable de video por la hendidura de la capa de plástico acrílico superior para terminar de acoplar la cámara en dicha superficie. Una vez realizadas todas las operaciones la cámara debe quedar acoplada en la base de plástico acrílico tal y como se puede apreciar en la siguiente figura.

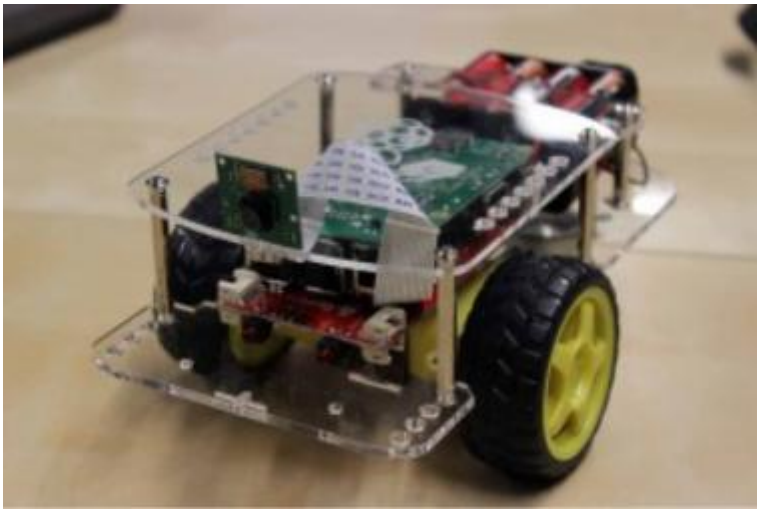


Figura 84. Cámara acoplada en la base de plástico acrílico del robot GoPiGo-2.

Instalación de la Imagen de Raspbian y actualización de paquetes y librerías.

Como se ha comentado con anterioridad la compañía Dexter Industries proporciona dos tipos de imágenes para trabajar con la placa Raspberry Pi. La imagen con la que se ha desarrollado este trabajo es *Raspbian* ya que se trata de una imagen 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.

Instalación de la Imagen de Raspbian.

La tarjeta Raspberry Pi 3 Model B incluye una ranura en la cual introducir una tarjeta micro-SD en la cual debe ir instalada la imagen a utilizar con el robot por lo que una vez montado el robot, el primer paso necesario antes de empezar a desarrollar software para la plataforma robótica es la instalación de la imagen que se utilizará como base para la tarjeta Raspberry Pi 3 Model B, en este caso la imagen de Raspbian. Para realizar esta tarea se ha utilizado el software Etcher (https://etcher.io). Etcher es una aplicación gratuita y de código abierto desarrollada para facilitar en todo lo posible la grabación de imágenes ISO, IMG y otros formatos comprimidos directamente y de forma segura a memorias USB y tarjetas de memoria.

Las principales características de esta aplicación son:

- Ofrece una interfaz sencilla y agradable.
- Cuenta con sistemas de validación para evitar la creación de memorias corruptas.
- Facilita la elección de la unidad USB de destino evitando equivocarnos y borrar por error un disco duro.
- Escrito completamente en JavaScript, HTML, Node.js y Electron.
- Permitirá grabar varias memorias de forma simultánea.

Para instalar la imagen de Raspbian en la tarjeta micro-SD primero se debe arrancar la aplicación Etcher cuya ventana principal se muestra a continuación.

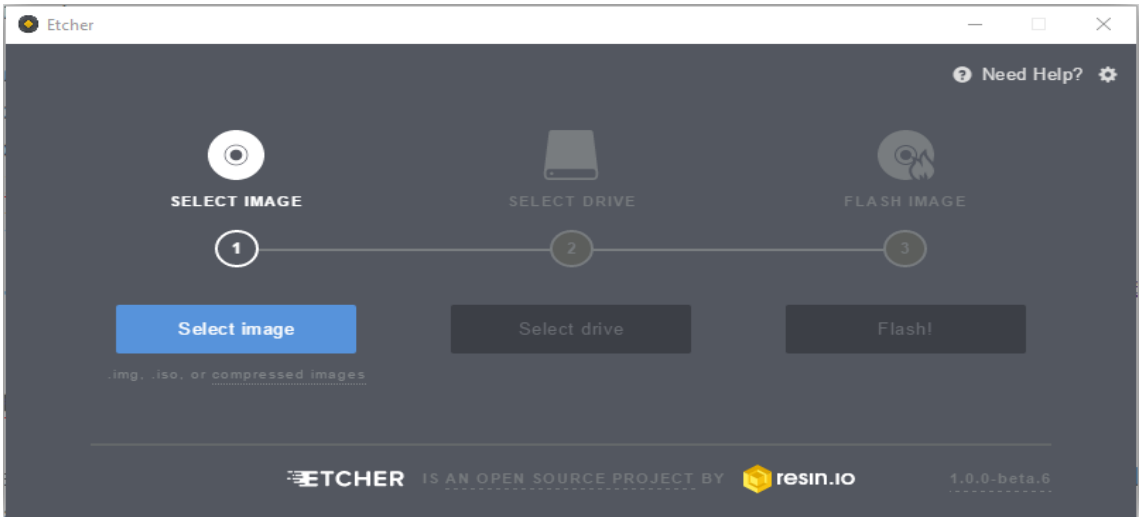


Figura 85. Ventana principal de la aplicación Etcher para el grabado de la imagen de Raspbian.

Como se puede apreciar en la figura anterior, el proceso de copiado consta de 3 pasos:

1. **Select Image**: se debe seleccionar la imagen ISO, IMG o de cualquier otro formato compatible para cargarla en el programa.
2. **Select Drive**: se debe seleccionar la unidad de destino donde se grabará la imagen seleccionada en el paso 1.
3. **Flash!**: iniciar el proceso de grabación de la imagen en la tarjeta micro-SD.

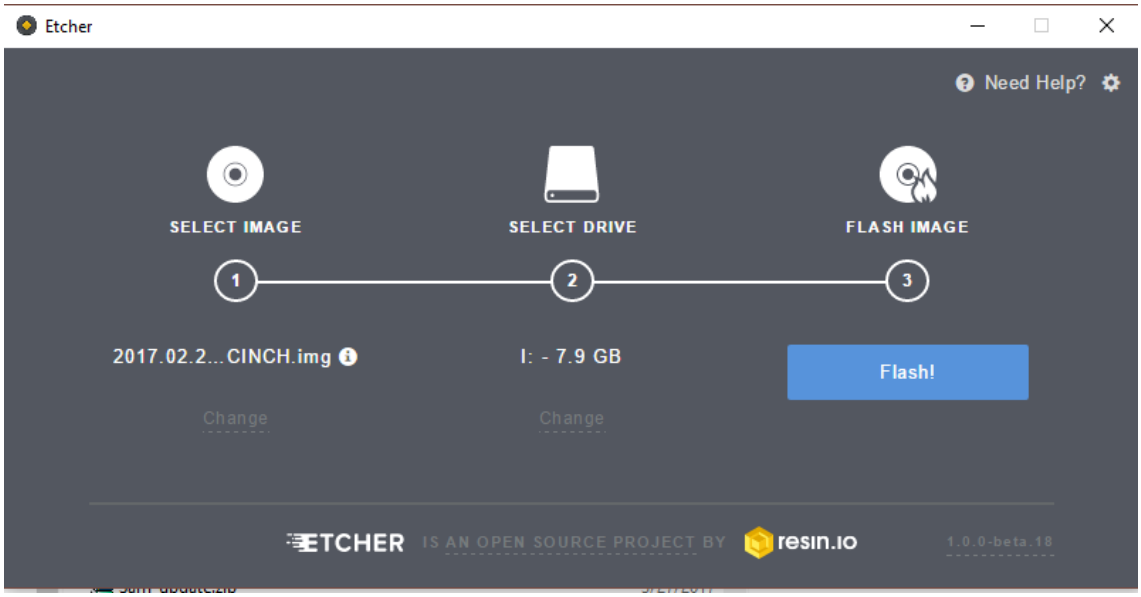


Figura 86. Ventana principal de la aplicación Etcher preparada para grabar una imagen en un una unidad de destino seleccionada.

El proceso de grabación de la imagen tardará varios minutos, según el tamaño de esta y la velocidad tanto del USB como del ordenador. Una vez finalice la copia de los datos, Etcher realizará una serie de configuraciones adicionales con el fin de verificar que todo está correcto y no se ha generado un medio corrupto.

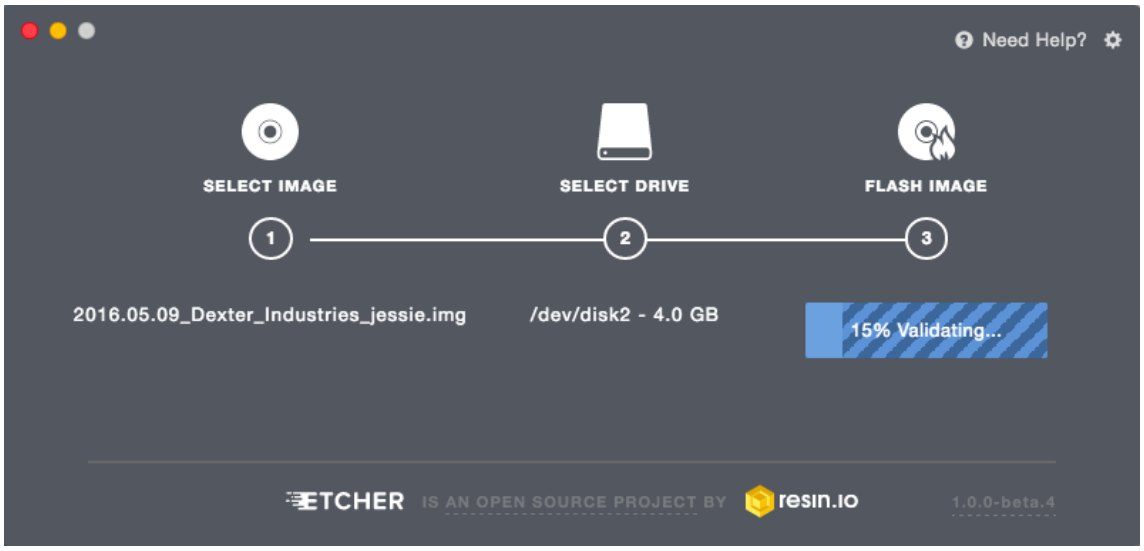


Figura 87. Ventana principal de la aplicación Etcher. Proceso de verificación / validación de la imagen grabada.

Una vez finalice dicha verificación se tendrá grabada en la tarjeta micro-SD la imagen de Raspbian seleccionada. Ahora con la imagen instalada se puede proceder a arrancar el robot usando la tarjeta micro-USB.

Conexión con el Robot y actualización de librerías.

Para poder trabajar con el robot es necesario conectarse a él. Este procedimiento puede ser algo tedioso por lo que a continuación se describen los pasos necesarios para establecer la conexión entre el PC de desarrollo y el robot GoPiGo 2.

- Conectar el cable de ethernet en el Raspberry Pi (placa verde) y el otro extremo del cable de ethernet en el PC de desarrollo. La alimentación del robot debe estar encendida, por lo que se deben ver las luces junto al puerto de Ethernet parpadeando como ocurre en la siguiente figura.

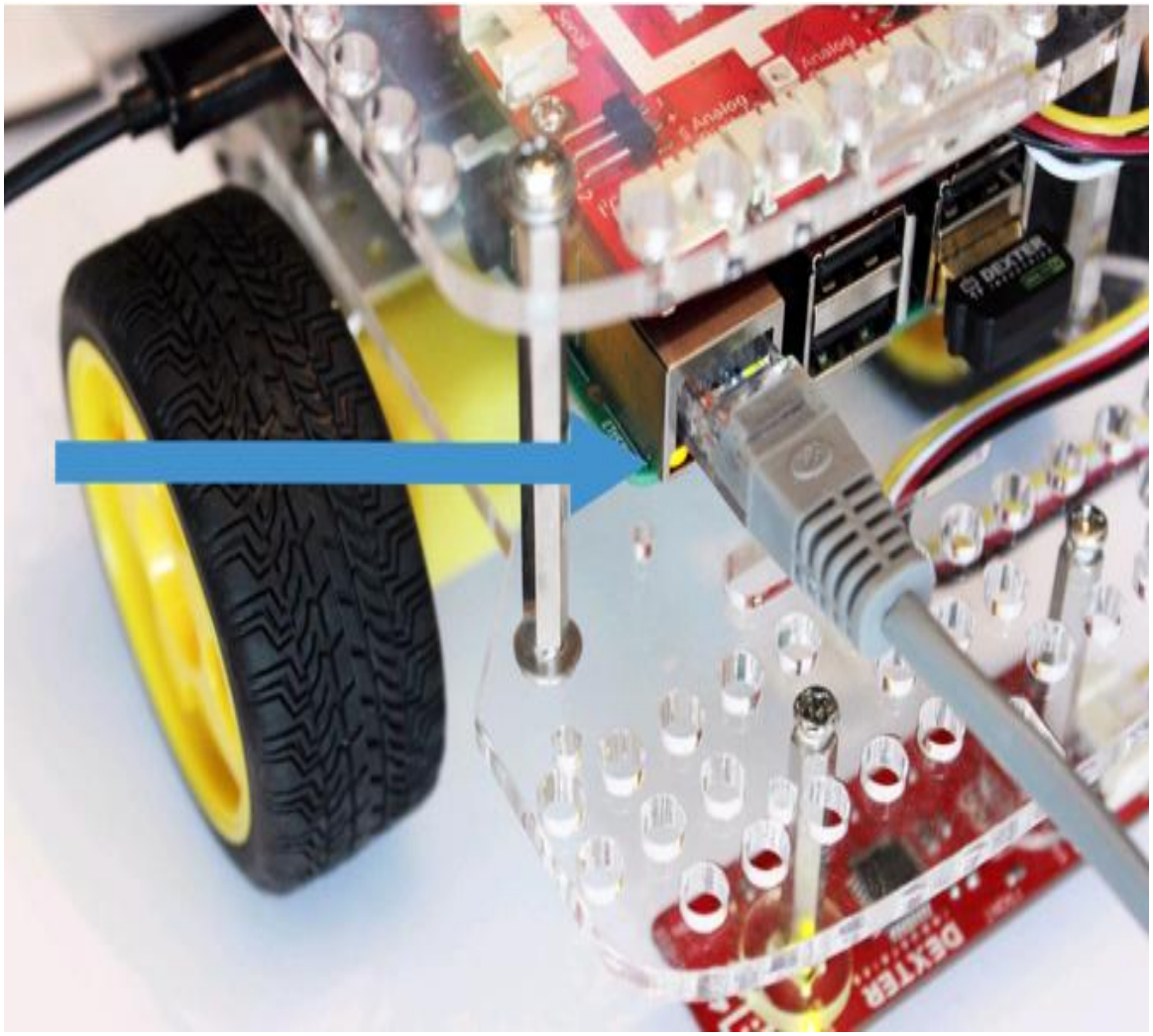


Figura 88. Conexión del robot (placa Raspberry Pi 3) con el PC de desarrollo mediante cable Ethernet.

- A continuación es necesario abrir un navegador web y acceder a la siguiente URL "http://dex.local" por medio de la barra de direcciones del navegador. (figura 27).

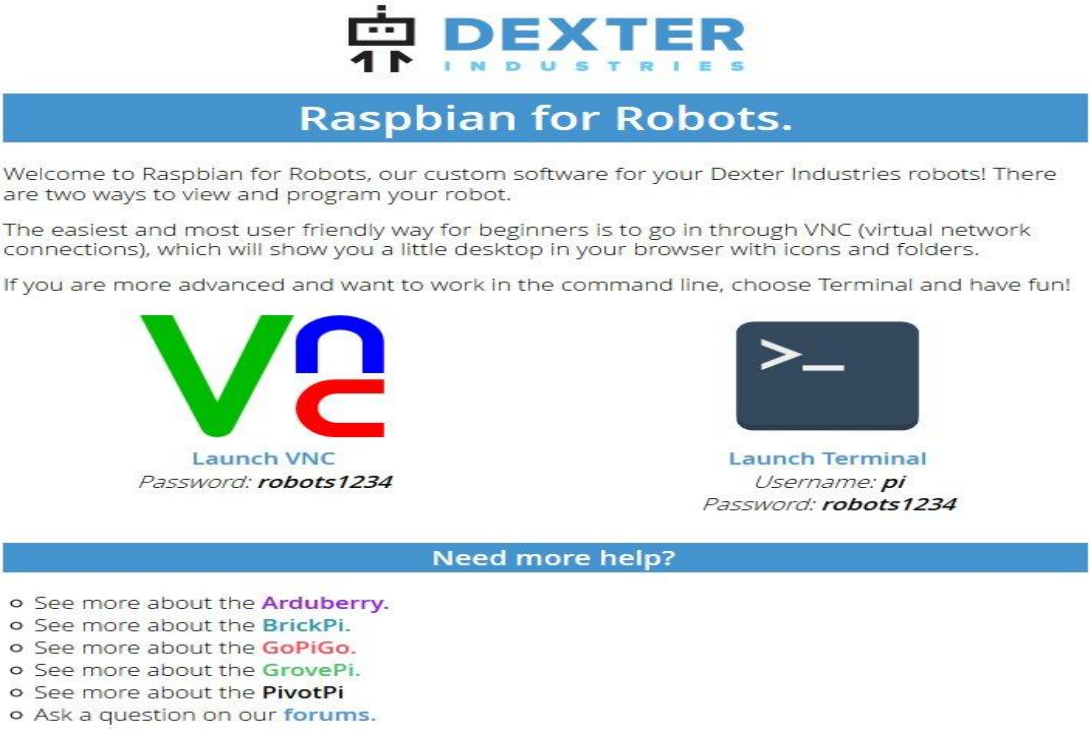


Figura 89. Vista principal de la url dex.local una vez que se ha establecido la comunicación con el robot GoPiGo 2.

- Una vez que ha conseguido el acceso a la url dex.local se debe proceder a configurar la conexión Wifi del robot de tal forma que no sea necesario volver a usar el cable Ethernet para conectar el robot al PC de desarrollo. Para ello se debe pulsar sobre la opción *"Launch VNC"*. Una pulsada esta opción aparecerá en el navegador un cuadro de diálogo que solicita el password de acceso. En este caso se debe usar el password *"robots1234"* para acceder. Una vez introducida el password se obtiene acceso al escritorio del robot GoPiGo 2 de Dexter Industries.

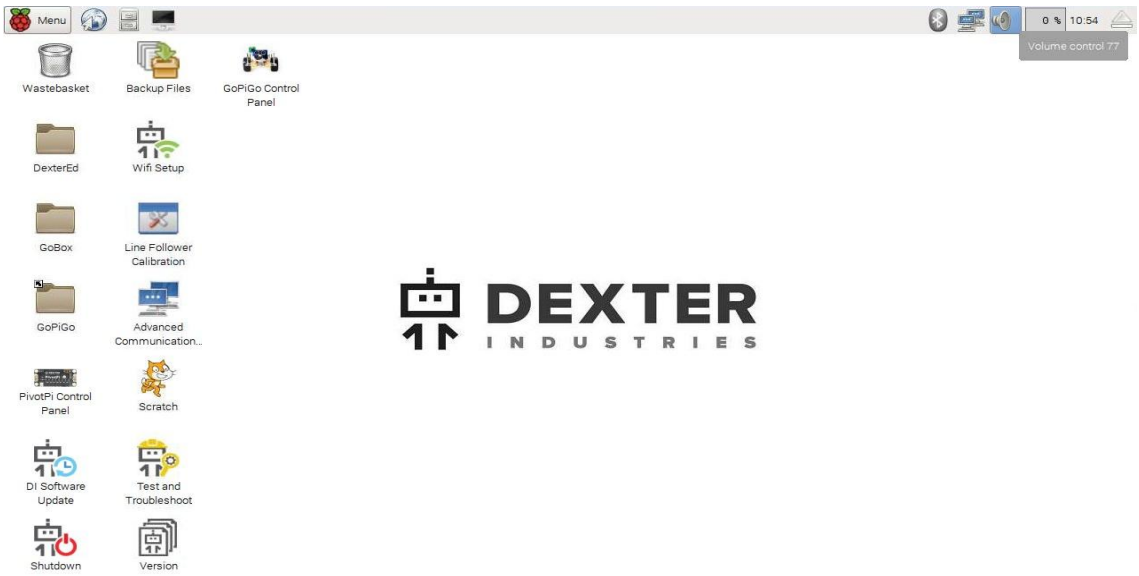


Figura 90. Escritorio Dexter Industries del robot GoPiGo 2.

- Como se puede apreciar en la figura 28 desde el escritorio del robot GoPiGo 2 se pueden llevar a cabo muchas funciones relacionadas con la configuración y mantenimiento del robot así como la programación del mismo. Además el escritorio dispone de una aplicación sencilla de control remoto del robot llamada *"GoPiGo Control Panel"* que puede ser ejecutada para probar acciones simples de control sobre el robot tales como: comprobar el estado de la batería, mover el robot hacía adelante, mover el robot hacía atrás, mover el robot hacía la derecha, mover el robot hacía la izquierda, encender los dos leds delanteros, etc...

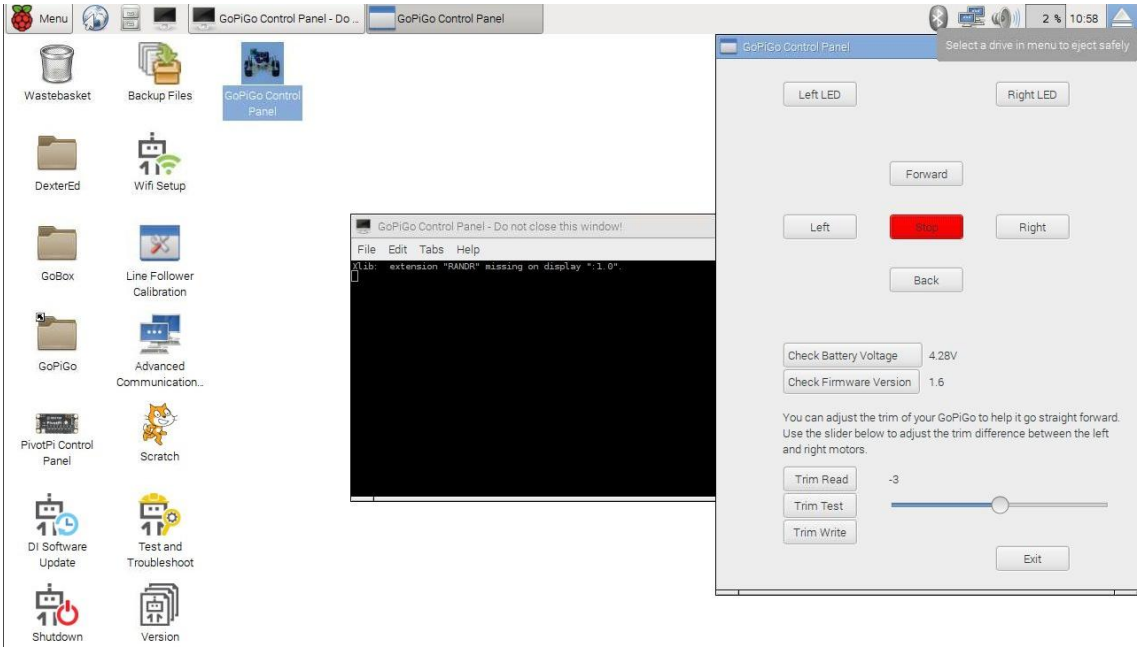


Figura 91. Vista de la aplicación de control del robot GoPiGo 2 – "GoPiGo Control Panel".

- Para configurar la WiFi del robot es necesario hacer doble click sobre el icono *"WiFi Setup"* de la parte superior izquierda del escritorio del robot GoPiGo 2.

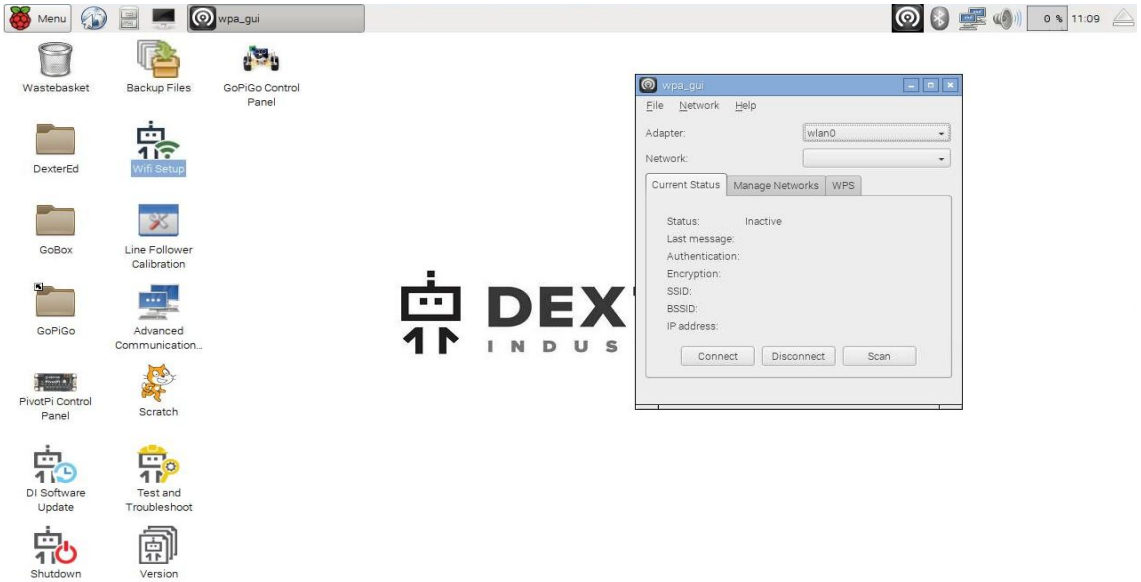


Figura 92. Ventana principal del panel de control "WiFi Setup" para la configuración WiFi del robot.

- A continuación se debe realizar un escaneo de las redes WiFi disponibles en el área a las que el robot puede conectarse. Para ello se debe pulsar el botón *"Scan"* en el panel de control *"WiFi Setup"*. Una vez encontrada la red WiFi deseada se debe pulsar sobre ella para abrir el cuadro de diálogo de configuración de dicha red WiFi. En este cuadro de diálogo se debe introducir la clave de la red WiFi para que quede grabada en el sistema y cada vez que el robot arranque se conecta automáticamente a esta red. Para que la configuración sea efectiva es necesario confirmar durante el proceso que el adaptador de red seleccionado sea *"wlan0"*.

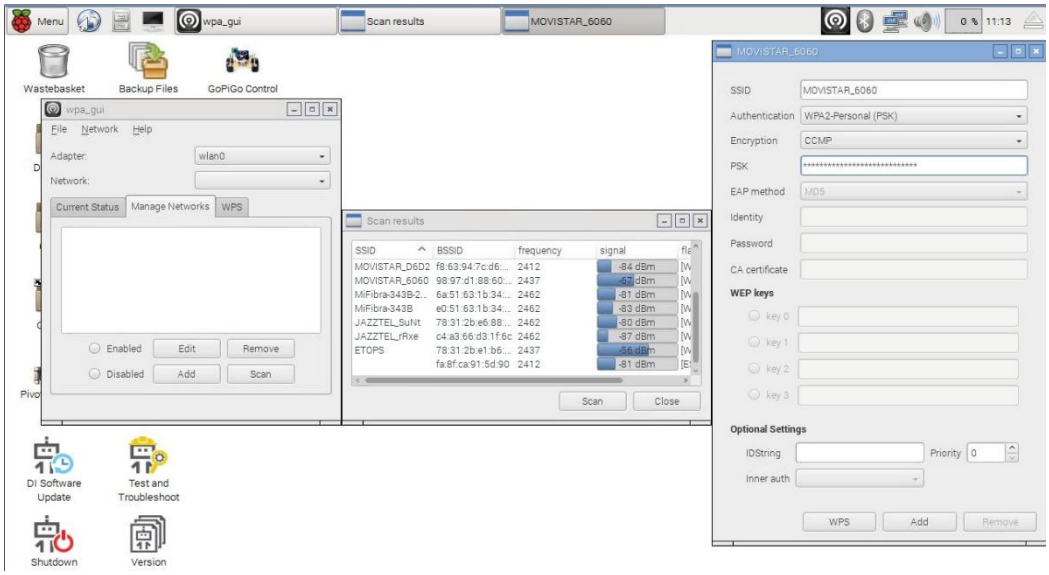


Figura 93. Escaneo y configuración de la red WiFi a la que se conectará el robot en cada arranque.

- Una vez cumplidos los requisitos anteriores es necesario establecer la conexión con la red. Para ello es necesario ir a la pestaña *"Current Status"* y pulsar sobre el botón *"Connect"*.

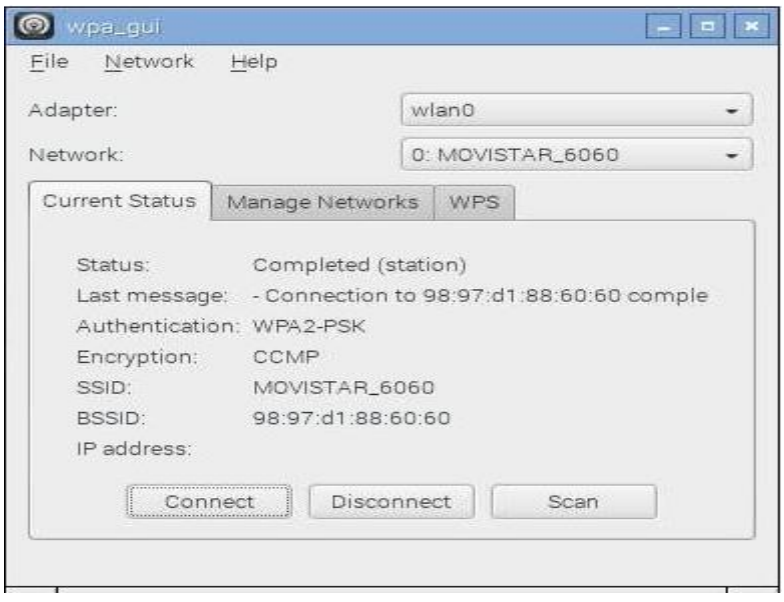


Figura 94. Establecimiento de la conexión del robot con la red WiFi (Status: Completed).

- Una vez finalizados los pasos anteriores se debe comprobar que todo es correcto y para ello se va a comprobar la conectividad con el exterior. Para ello se debe abrir la consola de comando usando el icono *"LX-Terminal"* y empleando el comando ***ping dexterindustries.com***:

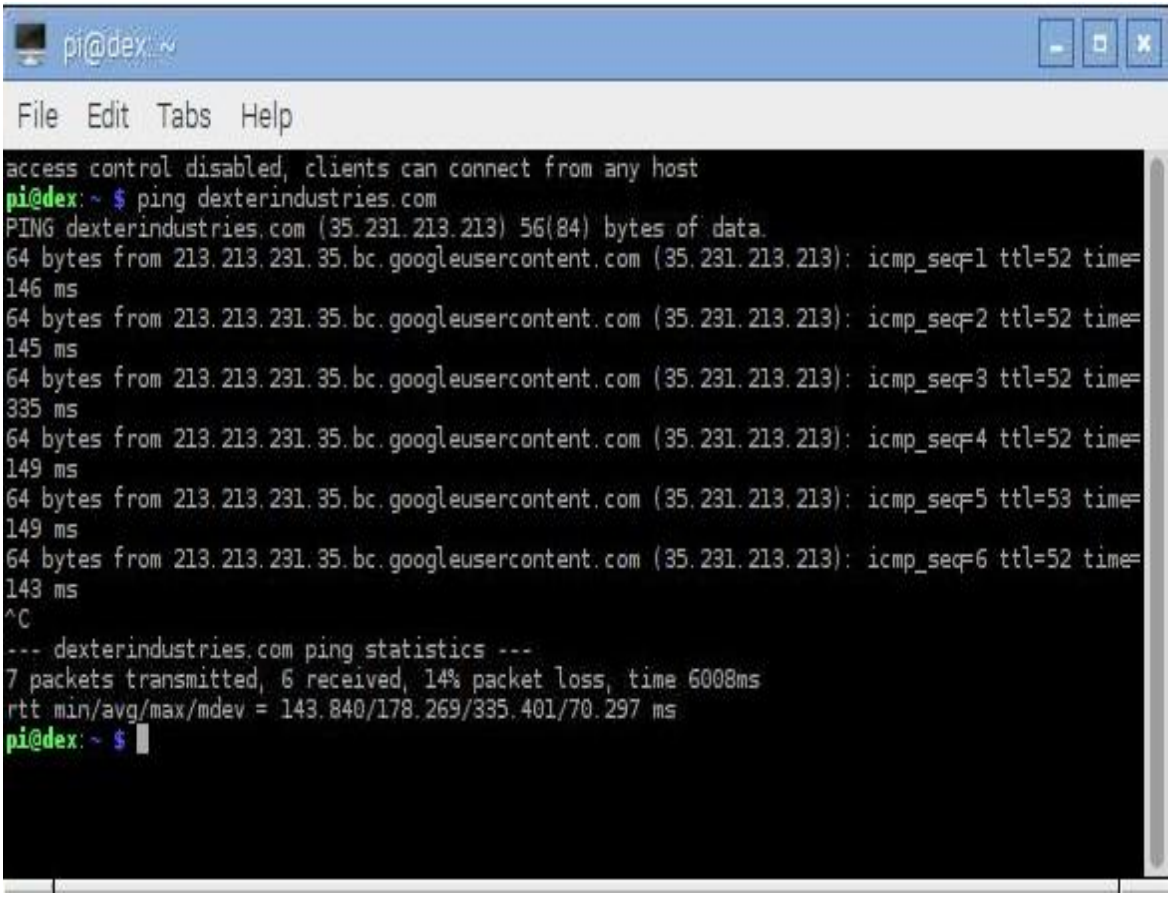


Figura 95. Prueba de conectividad del robot con una URL externa usando el comando "ping".

Con los pasos anteriores la configuración de la conexión WiFi del robot queda terminada. Una vez establecida la conexión con el robot se deben de llevar a cabo una serie de actualizaciones sobre un conjunto de librerías para poder utilizar como base del desarrollo del servidor RIP el código suministrado por los profesores. Las actualizaciones comentadas suponen por un lado instalar o actualizar la versión de Python instalada en la imagen Raspbian y por otro la instalación de una serie de librerías que por defecto no están instaladas en el robot. Para poder actualizar/instalar la versión 3 de Python en la imagen de Raspbian se debe establecer una conexión con el robot por medio de consola de comandos y aplicar el siguiente comando:

sudo apt-get install Python

Además es necesario instalar las siguientes tres librerías en el robot: **jsonrpc**, **ujson** y **cherrypy**.

Todas las librerías anteriores deben de ser instaladas ya que son necesarias para poder utilizar el protocolo JSON-RPC en el que está basada la implementación básica del servidor RIP suministrado por los profesores y que permite establecer la comunicación y realizar el intercambio de mensajes entre el robot y la aplicación EjsS desarrollada cuya descripción y funcionamiento se describirá más adelante en el capítulo 4. Para actualizar librerías se deben realizar los pasos siguientes:

- Primero se debe abrir la consola de comandos del escritorio del robot. Para ello se debe pulsar sobre el icono *"LX-Terminal"*. Una vez abierta la consola de comandos primero se procederá a instalar / actualizar Python. Para ello como se definió con anterioridad se debe utilizar el comando **sudo apt-get install Python**:

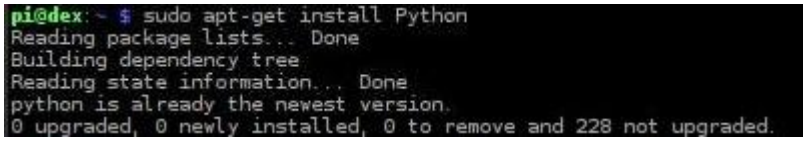


Figura 96. Actualización / instalación de la última versión de Python en el robot GoPiGo 2.

- A continuación se deben instalar las librerías presentadas en la tabla 1 y en el orden descrito en dicha tabla. Para ello primero se debe empezar por instalar la librería *"jsonrpc"* usando el comando **sudo pip install jsonrpc**:



Figura 97. Instalación de la librería jsonrpc en el robot GoPiGo 2.

- A continuación se debe instalar la librería ujson usando para ellos el comando **sudo pip install ujson**:

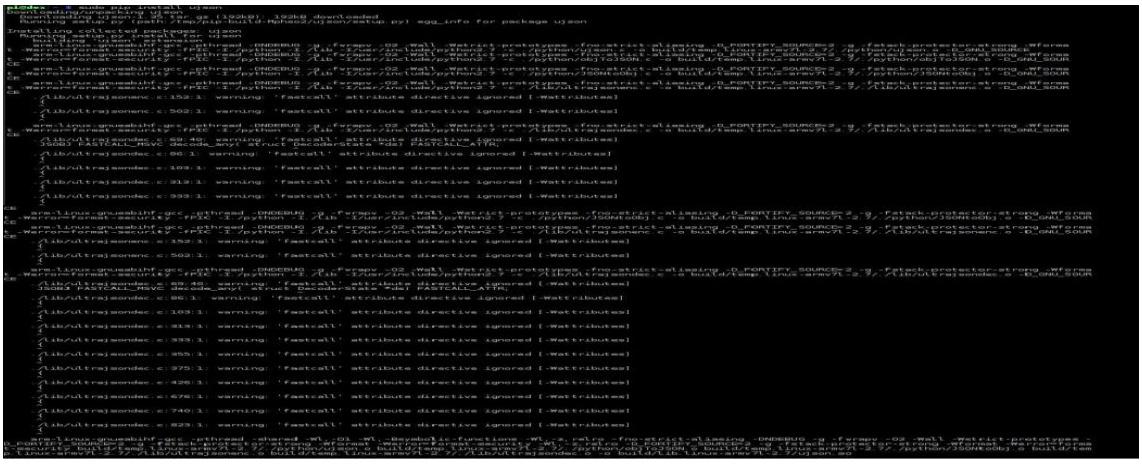


Figura 98. Instalación de la librería ujson en el robot GoPiGo 2.

- Finalmente, se debe instalar la librería *"cherrypy"* empleando para ello el siguiente comando **sudo pip install cherrypy**:

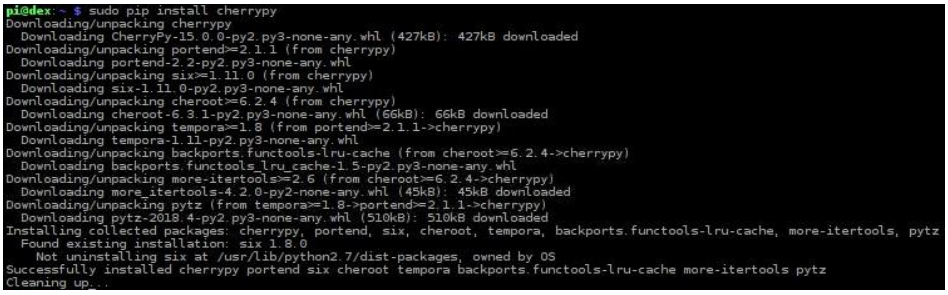


Figura 99. Instalación de la librería cherrypy en el robot GoPiGo 2.

Por otro lado, en este proyecto se ha trabajado tomando como base la implementación de un servidor RIP el cual está basado en una adaptación del servidor desarrollado por la UNED, cuyo código se puede encontrar en [6]. El servidor utiliza el protocolo JSON-RPC [14] para el intercambio de información. Esta implementación requiere también de un proceso de configuración y de instalación de librerías que se va a proceder a describir a continuación.

- El primer paso consiste en descargar la implementación del servidor RIP [6] utilizado como base en este trabajo. Para ello se puede utilizar el comando *"git"* para clonar el repositorio del código del servidor en la ubicación deseada dentro de la tarjeta de memoria micro-USB que hace las veces de espacio de almacenamiento dentro de la tarjeta Raspberry Pi 3 del robot GoPiGo-2.
- A continuación es necesario instalar las librerías propias de *ptyhon3* y *virtualenv*. Para ello se debe emplear el siguiente comando:

  sudo apt install python3 virtualenv

  Es necesario instalar la librería *virtualenv* pues se va a utilizar para instalar todas las librerías necesarias para el proyecto (junto con el intérprete de Python / Python3) en una localización virtual dentro de la carpeta del servidor RIP de tal forma que el servidor pueda ser exportable por ejemplo a otro robot sin tener que realizar de nuevo todo el proceso de instalación de librerías y de configuración.
- El siguiente paso consiste en crear el directorio virtual al cual se referenciaran las librerías y el entorno / interprete de Python instalados. Para ello se debe ejecutar el siguiente comando dentro de la carpeta donde se haya clonado el repositorio que contiene la implementación base del servidor RIP:

  virtualenv -p /usr/bin/python3 venv

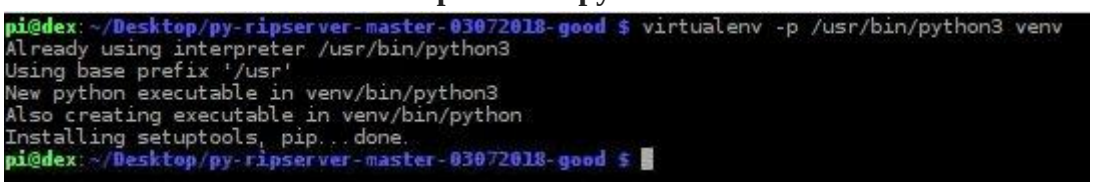


Figura 100. Crear el directorio virtual "venv" dentro del directorio del servidor RIP.

Con esto se consigue crear el directorio virtual ***"venv"*** que se utilizará como base para la consecución de los pasos siguientes.

- A continuación se proceder la librería *"GoPiGo"* en el directorio virtual ***"venv"***. Esta librería es fundamental pues define el API básico de programación del robot GoPiGo-2. Para realizar la instalación es necesario que el usuario se sitúe dentro de la ruta ***"Desktop/GoPiGo/Software/Python"*** y ejecutar el script *setup.py* de la siguiente forma: ***sudo <ruta_servidor_rip>/venv/bin/python setup.py install***

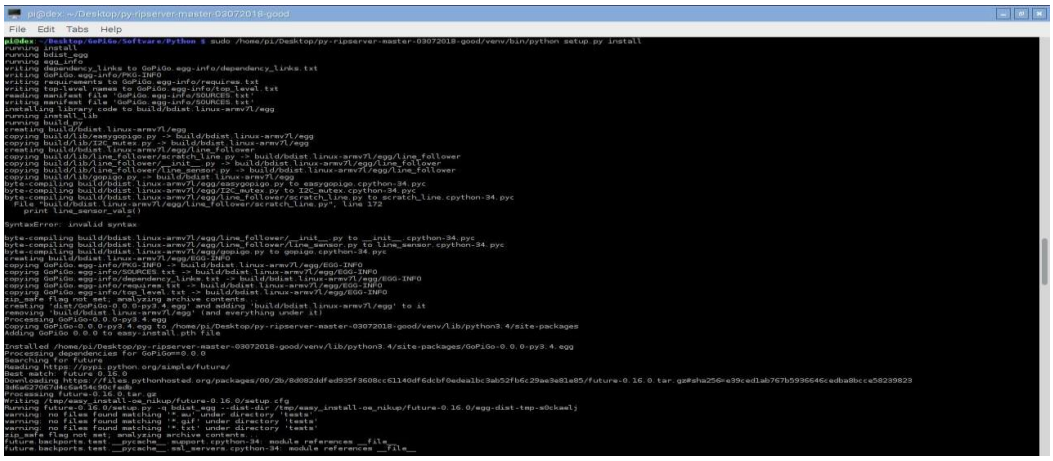


Figura 101. Instalación de la librería del robot "GoPiGo" en el directorio virtual (I).

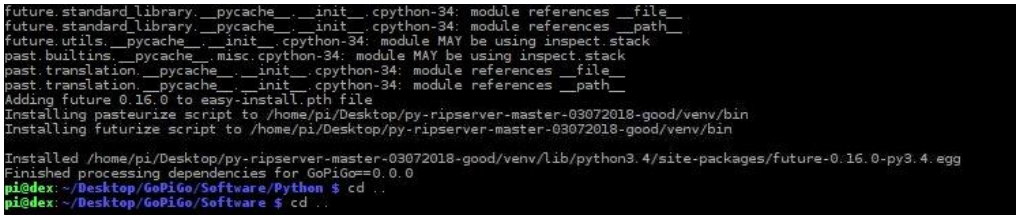


Figura 102. Instalación de la librería del robot "GoPiGo" en el directorio virtual (II).

El resultado del proceso de instalación anterior se puede apreciar en la siguiente figura.

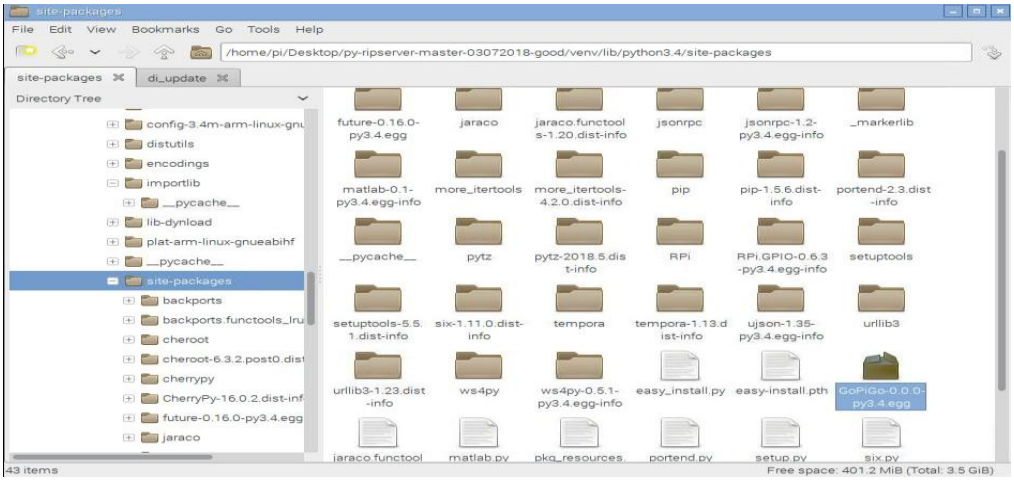


Figura 103. Resultado de la instalación de la librería "GoPiGo" en el directorio virtual "venv".

- El siguiente paso consiste en instalar dos librerías adicionales que son necesarias para la librería *"GoPiGo"* ya que se trata de dos dependencias de esta última.
    - La primera de ellas es la librería ***RPi.GPIO*** la cual permite tener acceso y control al bus GPIO de la tarjeta Raspberry Pi 3 del robot GoPiGo-2. Para realizar su instalación es necesario ejecutar el siguiente comando desde la consola: ***venv/bin/pip install RPi.GPIO***



Figura 104. Instalación de la librería RPi.GPIO en el robot GoPiGo-2.

    - La segunda librería es ***smbus***. Esta librería implementa en Python un subconjunto del protocolo I^2C. Para realizar su instalación es necesario ejecutar el siguiente comando desde la consola: ***venv/bin/pip install pysmbus***.

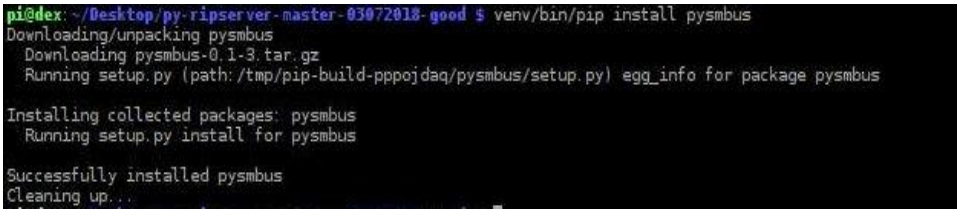


Figura 105. Instalación de la librería smbus en el robot GoPiGo-2.

- El siguiente paso consiste en reinstalar la librería *ujson* en el directorio virtual ***"venv"*** creado en los pasos anteriores. Para ello se debe ejecutar el siguiente comando desde la consola: ***venv/bin/pip install ujson***

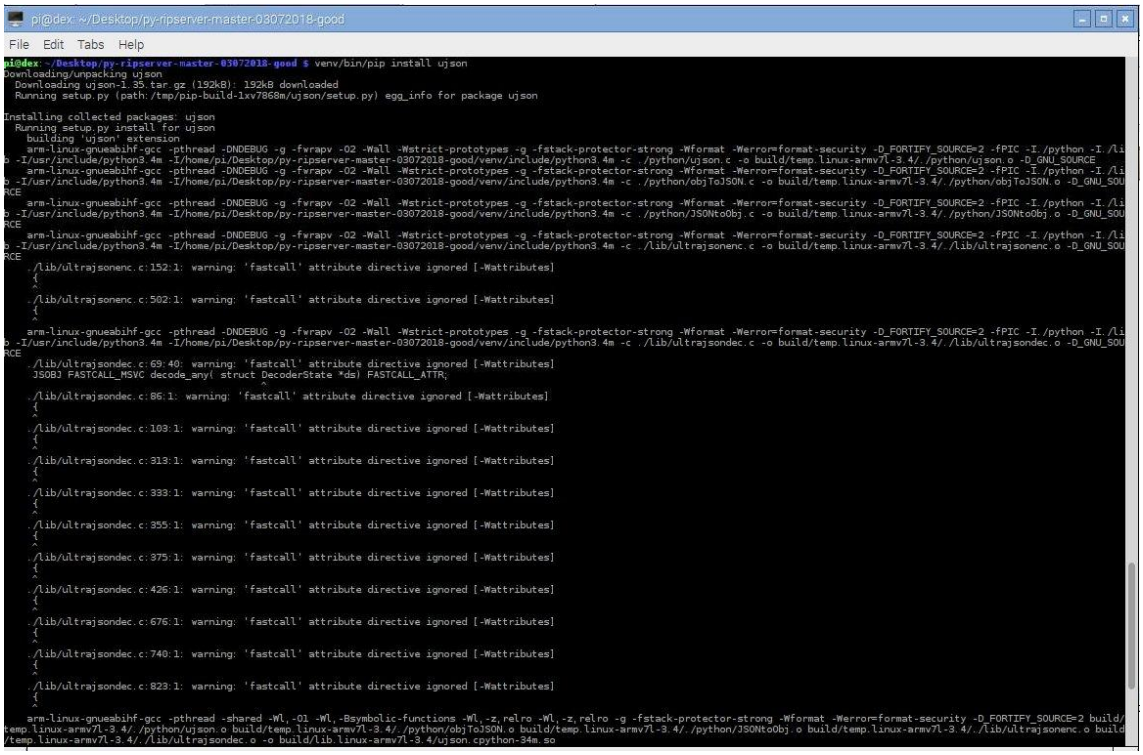


Figura 106. Instalación de la librería ujson en el directorio virtual "venv" de la implementación base del servidor RIP.

- A continuación es necesario reinstalar la librería *cherrypy* en el directorio virtual ***"venv"*** creado en los pasos anteriores. Para ello se debe ejecutar el siguiente comando desde la consola: ***venv/bin/pip install cherrypy***



Figura 107. Instalación de la librería cherrypy en el directorio virtual "venv" de la implementación base del servidor RIP.

- El siguiente paso consiste en resintalar la librería *jsonrpc* en el directorio virtual ***"venv"*** creado en los pasos anteriores. Para ello se debe ejecutar el siguiente comando desde la consola: ***venv/bin/pip install jsonrpc***

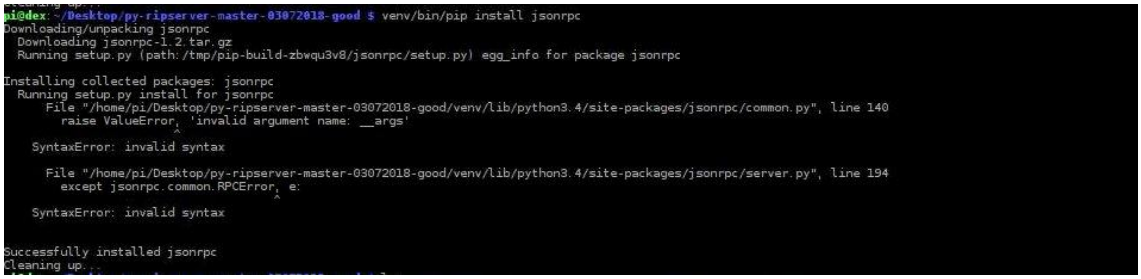


Figura 108. Instalación de la librería jsonrpc en el directorio virtual "venv" de la implementación base del servidor RIP.

- A continuación, se debe realizar la instalación de la librería *"WS4PY"*, la cual implementa el protocolo *WebSocket* definido en la *RFC6455*, en el directorio virtual ***"venv"*** creado en los pasos anteriores. Para ello se debe ejecutar el siguiente comando desde la consola: ***venv/bin/pip install ws4py***

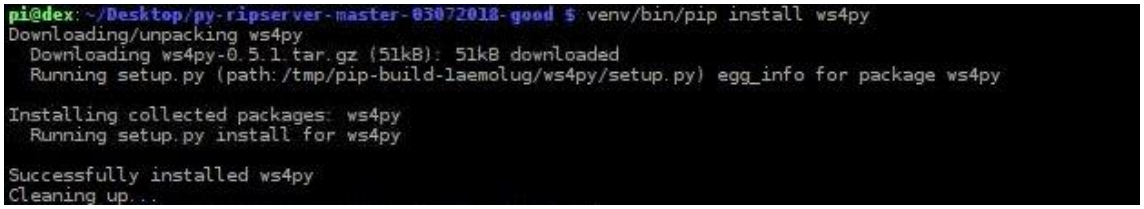


Figura 109. Instalación de la librería ws4py en el directorio virtual "venv" de la implementación base del servidor RIP.

- Finalmente, es necesario emplear el comando ***"sudo apt-get install python3-cherrypy3"***. Este paso necesario para instalar el núcleo de la librería *cherrypy* en el interprete del lenguaje de programación Python correspondiente a la versión 3.x en adelante. Este paso es estrictamente necesario pues no basta con actualizar la librería *cherrypy* (cómo se explicó en uno de los pasos previos empleando en el comando ***pip install cherrypy***). Una vez completado este paso se puede lanzar la versión de pruebas de la implementación de la clase ***"HttpServer.py"*** que acompaña a la distribución del servidor o bien lanzar una propia. Para ello se puede ejecutar el siguiente comando: ***"sudo python3 HttpServer.py"***.

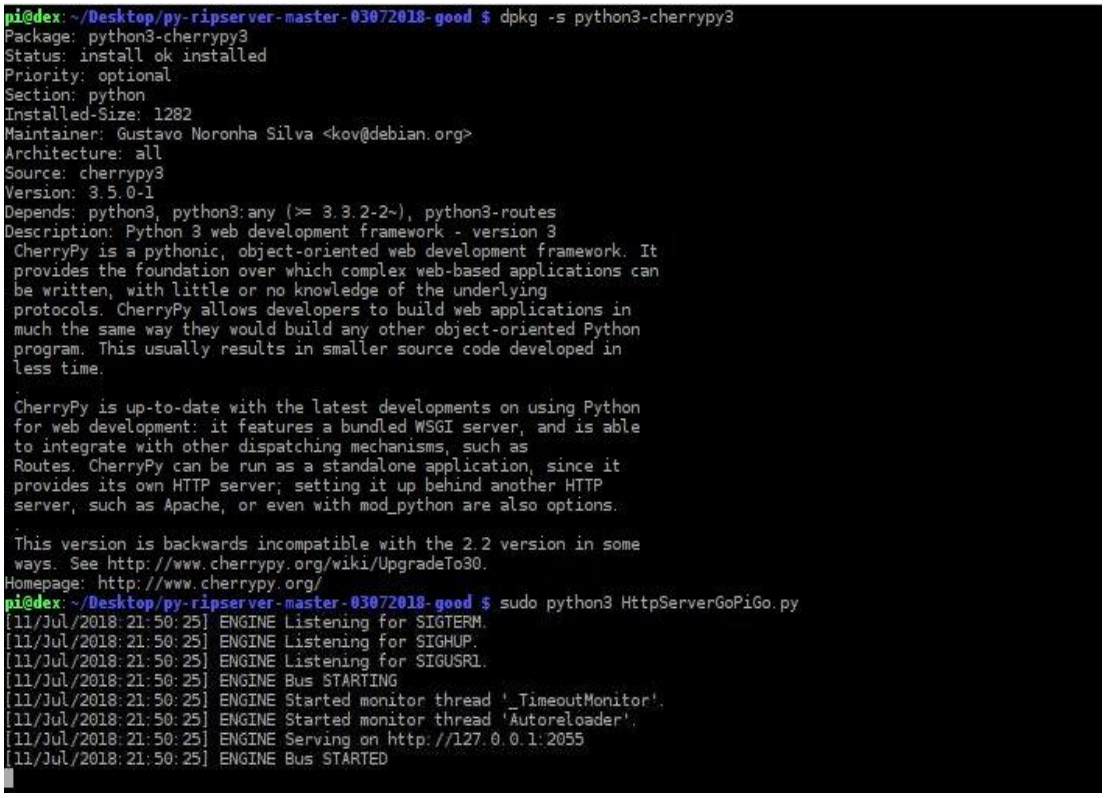


Figura 110. Comprobación de la correcta instalación de la librería python3-cherrypy3 y prueba de ejecución del servidor RIP de prueba.

Un comando útil que puede ser utilizado para comprobar la correcta instalación de la librería ***"python3-cherrypy3"*** es el siguiente: ***"dpkg –s python3-cherrypy3"***. Como se puede apreciar en la figura 104 la librería instalada debe depender de ***python3*** o de cualquier otra versión superior. En caso de que existiesen conflictos con otro paquete similar este comando también lo indicaría.

Una vez finalizadas las actualizaciones de librerías sólo resta reiniciar el robot para comprobar que los cambios en la configuración de la conexión con la red WiFi han sido efectuados de forma satisfactoria. Para ello se deben seguir los pasos siguientes:

- El primer paso consiste en ejecutar el apagado ordenado del sistema. Para ello se debe pulsar sobre el icono *"Shutdown"*. Una vez se haya pulsado y se haya confirmado en el cuadro de diálogo el apagado del robot se debe comprobar que ya no se tiene acceso a la url local *"dex.local"* y posteriormente se debe confirmar que las luces de actividad del robot (led de WiFi y led de alimentación) se han apagado.



Figura 111. Apagado ordenado del robot GoPiGo 2.

- El siguiente paso consiste en desconectar el cable Ethernet de la tarjeta Raspberry Pi 3 del robot GoPiGo 2 tal y como se muestra en la siguiente figura.

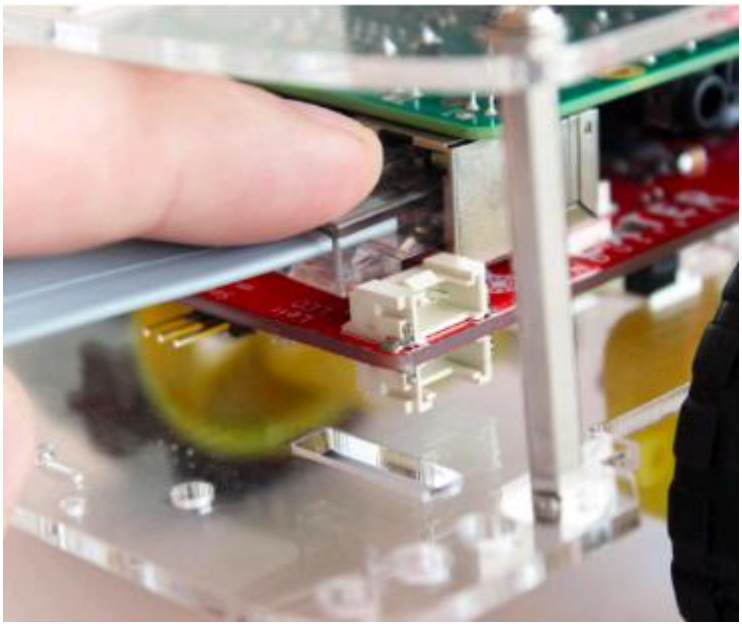


Figura 112. Desconexión del cable de red Ethernet de la tarjeta Raspberry Pi 3 del robot GoPiGo 2.

- Finalmente se debe volver a encender el robot (ya con el cable Ethernet desconectado del robot) y esperar a que termine el procedimiento de inicio, que finaliza cuando el led que indica que el adaptador WiFi es operativo ya sea de forma fija o parpadeando de forma intermitente, para volver a acceder a la url local *"dex.local"*. Si todo ha ido correctamente, el robot (ya configurado para conectarse a la red WiFi definida) dará acceso a la url local. Por último, para comprobar la conectividad entre el PC de desarrollo y el robot GoPiGo 2 se va a proceder a utilizar el comando ping de la siguiente forma: **ping dex.local**

```
C:\Users\Jorge Duran Murillas>ping dex.local

Haciendo ping a dex [192.168.1.39] con 32 bytes de datos:
Respuesta desde 192.168.1.39: bytes=32 tiempo=55ms TTL=64
Respuesta desde 192.168.1.39: bytes=32 tiempo=6ms TTL=64
Respuesta desde 192.168.1.39: bytes=32 tiempo=6ms TTL=64
Respuesta desde 192.168.1.39: bytes=32 tiempo=5ms TTL=64

Estadísticas de ping para 192.168.1.39:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 5ms, Máximo = 55ms, Media = 18ms

C:\Users\Jorge Duran Murillas>ping dex.local
La solicitud de ping no pudo encontrar el host dex.local. Compruebe el nombre y
vuelva a intentarlo.

C:\Users\Jorge Duran Murillas>ping dex.local_
```

Figura 113. Comprobación de la conectividad entre el PC de desarrollo y el robot GoPiGo 2 mediante conexión WiFi.

Como se ha visto a lo largo del presente trabajo, en el robot GoPiGo-2 se ha instalado una webcam que permite capturar flujos de video. Por defecto, el robot no dispone de ninguna funcionalidad para activar las funciones de streaming de video, sin embargo el robot, si que incluye una carpeta de ejemplos en la que entre otros se encuentra la implementación de un servidor de streaming que es la que se ha utilizado en este trabajo. La implementación de este servidor de streaming de video se encuentra en la ruta *"/Desktop/GoPiGo/Software/Python/Examples/Browser/Streaming_Robot"*.

Para instalar el servidor es necesario seguir los siguientes pasos:

- Ejecutar el comando: ***sudo browser_stream_setup.sh***. Este primer paso es esencial pues al ejecutar este comando se accederá a los repositorios remotos para poder descargar el núcleo del servidor.
- Una vez instaladas todas las dependencias principales en el paso anterior es necesario instalar algunas librerías auxiliares:
    - Instalar la librería sockjs-tornado mediante la ejecución del comando: ***sudo pip3 install sockjs-tornado –upgrade***.
    - Ejecutar el comando ***sudo apt-get update*** para actualizar las dependencias.
    - Ejecutar el comando ***sudo apt-get install python-tornado*** para instalar la última versión de la librería "*tornado" para Python.*
- Una vez instaladas las librerías anteriores el servidor de streaming se puede arrancar empleando para ello el siguiente comando: ***sudo python robot_web_server.py***.

Una vez que el servidor de streaming ha sido arrancado se puede hacer una pequeña prueba para comprobar que funciona correctamente. Para ello, abrir una ventana de navegador y escribir en la barra de direcciones: http://<dir_ip_robot>:98. Si todo ha ido correctamente se presenta al usuario un panel en el que se visualiza el flujo de video capturado por la webcam.

ANEXO B.

En este anexo se presenta el código fuente de la implementación realizada como parte de la extensión del servidor RIP que se ha utilizado como base para el desarrollo de la pieza software que permite la comunicación con el robot GoPiGo-2. Como ya se ha comentado con anterioridad el desarrollo realizado en este trabajo toma como base una implementación previa de un *Servidor RIP* [6].

Clase HttpServer (HttpServer.py).

```
# - *- coding: utf- 8 - *-

'''

Modified on 23/07/2018

@author: jcsombria

'''

import cherrypy

import json

import os



from app.RIPMeta import *

from app.GoPiGoRestController import GoPiGoRestController



class HttpServer(object):

 '''

 RIP Server implementation

 '''

 exposed = True



 def __init__(self, control=GoPiGoRestController(), host='127.0.0.1', port=2055):

  self.control = control

  self.host = host

  self.port = port

  self.experiences = [{ 'id': control.name }]

 @cherrypy.expose

 def index(self, expId=None):
```

```
    '''
    GET - Retrieve the list of experiences or information about a specific experience
    '''
    if expId is not None:
      if expId in [e['id'] for e in self.experiences]:
        response = self.control.info(self.getAddr())
      else:
        response = '{}'
    else:
      response = self.info()
    return response.encode("utf-8")


  def getAddr(self):
    return '%s:%s' % (self.host, self.port)
  @cherrypy.expose
  def SSE(self, expId=None):
    '''
    SSE - Connect to an experience's SSE channel to receive periodic updates
    '''
    cherrypy.response.headers['Content-Type'] = 'text/event-stream'
    cherrypy.response.headers['Cache-Control'] = 'no-cache'
    cherrypy.response.headers['Connection'] = 'keep-alive'
    return self.control.nextSample()
  SSE._cp_config = {'response.stream': True}
```

```
@cherrypy.expose
  @cherrypy.tools.accept(media='application/json')
  def POST(self):
    '''
    POST - JSON-RPC control commands (get/set)
    '''
    socket = cherrypy.request.body.fp
    message = socket.read()
    response = self.control.parse(message)
    response = json.dumps(response)
    return response.encode("utf-8")
  def info(self):
    '''
    Build server info string
    '''
    meta = RIPExperienceList(self.experiences, [self.buildGetInfo()])
    return str(meta)
def buildGetInfo(self):
    return RIPMethod(
      url='%s/RIP' % self.getAddr(),
      description='Retrieves information (variables and methods) of the experiences in the server',
      type_='GET',
      params=[
        RIPParam(name='Accept',required='no',location='header',value='application/json'),
        RIPParam(name='expId',required='no',location='query',type_='string'),
      ],
      returns='application/json',
      example='http://%s/RIP?expId=%s' % (self.getAddr(), self.control.name)
    )
```

```
def start(self):
    base_dir = os.path.dirname(os.path.realpath(__file__))
    log_dir = os.path.join(base_dir, 'log')
    access_log_file = os.path.join(log_dir, 'access.log')
    error_log_file = os.path.join(log_dir, 'error.log')
    cherrypy.config.update({
     'server.socket_port': self.port,
     'log.access_file' : access_log_file,
     'log.errors_file' : error_log_file
    })
    config = {
     '/': {
       '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(self, '/RIP', config)


# ----------------------------------------------------------------------------------------------
if __name__ == '__main__':
  cherrypy.config.update({'server.socket_host':'0.0.0.0'})
  HttpServer().start()
```

Clase HttpServerGoPiGo (HttpServerGoPiGo.py).

```
'''
@author: Jorge Durán Murillas <jduranmaster@gmail.com>
'''
from HttpServer import HttpServer
from app.GoPiGoRestController import GoPiGoRestController


if __name__ == '__main__':
  HttpServer(
    host='0.0.0.0',
    port=2055,
    control=GoPiGoRestController(
      name='GoPiGo-2',
      description='GoPiGo-2 RIP Server Implementation.',
      authors='J. Durán',
      keywords='Octave, Raspberry PI, Robot',
    ),
  ).start(enable_ssl=False
```

Clase HttpServerGoPiGo (HttpServerGoPiGo.py).

```
# - *- coding: utf- 8 - *-

'''

Created on 01/07/2018

@author: Jorge Durán Murillas <jduranmaster@gmail.com>

'''

from rip.RIPGeneric import RIPGeneric

from app.GoPiGoService import GoPiGoService

class GoPiGoRestController(RIPGeneric):

 '''

 GoPiGoRestController Adapter

 '''

 def __init__(self, name='gopigo2ripserver', description='GoPiGo-2 RIP Server Implementation.', authors='Jorge Durán', keywords=''):

'''

  RIPGoPiGo Constructor

  To register readable or writable variables:

    self.readables.append({

      'name':'x',

      'description':'Testing readable variable',

      'type':'float',

      'min':'-Inf',

      'max':'Inf',

      'precision':'0'

      })

    self.writables.append({

      'name':'x',

      'description':'Testing writable variable',

      'type':'float',

      'min':'-Inf',

      'max':'Inf',

      'precision':'0'

      })
```

```
super().__init__(name, description, authors, keywords)


  self.gopigoservice = GoPiGoService()


 def connect(self, expid):
  '''
  Initializes the GoPiGo Service
  '''
  return self.gopigoservice.connect()


 def set(self, expid, variables, values):
  '''
  Writes one or more variables to the workspace of the current GoPiGo session
  '''
  return self.gopigoservice.set(variables, values)


 def get(self, expid, variables):
  '''
  Retrieve one or more variables from the workspace of the current GoPiGo session
  '''
  return self.gopigoservice.get(variables)


 def getValuesToNotify(self):
  return [
   ['time'],
   [self.sampler.lastTime()]
  ]
```

Clase RIPGeneric (RIPGeneric.py).

```
# - *- coding: utf- 8 - *-

'''

Created on 01/07/2018



@author: Jorge Durán Murillas <jduranmaster@gmail.com>

'''

import os.path

import time

from jsonrpc.JsonRpcServer import JsonRpcServer

from jsonrpc.JsonRpcBuilder import JsonRpcBuilder

from app.GoPiGoService import GoPiGoService

from app.RIPMeta import *

builder = JsonRpcBuilder()

class GoPiGoRestController(JsonRpcServer):

  ''' RIP Server - Reference Implementation'''

  def __init__(self, name='gopigo2ripserver', description='GoPiGo-2 RIP Server Implementation.', authors='Jorge Durán', keywords=''):

    super().__init__(name, description)

    self.authors = authors

    self.keywords = keywords

    self.ssePeriod = 0.5

    self.sseRunning = False

    self.gopigoservice = GoPiGoService()

    self.readables = [{

        "name":"time",

        "description":"Server time in seconds",

        "type":"float",

        "min":"0",

        "max":"Inf",

        "precision":"0"

      },

    ]
```

```
self.writables = []
    self.addMethods({
     'connect': { 'description': 'Connect to Server',
      'params': { 'expId': 'string' },
      'implementation': self.connect,
     },
     'get': { 'description': 'To read server variables',
      'params': { 'expId': 'string', 'variables': '[string]' },
      'implementation': self.get,
     },
     'set': { 'description': 'To write server variables',
      'params': { 'expId': 'string', 'variables': '[string]', 'values':'[]' },
      'implementation': self.set,
     },
    })
def info(self, address='127.0.0.1:2055'):
    '''
    Retrieve the experience's info
    '''
    try:
     info = self.info_string
    except:
     info = self.build_info(address)
     self.info_string = info
    return info
def build_info(self, address):
    info = RIPServerInfo(
     self.name,
     self.description,
     authors=self.authors,
     keywords=self.keywords
    )
```

```
readables = RIPVariablesList(
    list_=self.readables,
    methods=[self.buildSSEGetInfo(address), self.buildPOSTGetInfo(address)],
    read_notwrite=True
  )
  writables = RIPVariablesList(
    list_=self.writables,
    methods=[self.buildPOSTSetInfo(address)],
    read_notwrite=False
  )
  meta = RIPMetadata(info, readables, writables)
  return str(meta)
 # CONNECT method.
 def connect(self, expid):
  return self.gopigoservice.connect()
 # GET method.
 def get(self, expid, variables):
  return self.gopigoservice.get(variables)
 # SET method.
 def set(self, expid, variables, values):
  return self.gopigoservice.set(variables, values)
def buildSSEGetInfo(self, address):
  return RIPMethod(
    url='%s/RIP/SSE' % address,
    description='Suscribes to an SSE to get regular updates on the servers\' variables',
    type_='GET',
    params=[
     RIPParam(name='Accept',required='no',location='header',value='application/json'),
     RIPParam(name='expId',required='yes',location='query',type_='string'),
     RIPParam(name='variables',required='no',location='query',type_='array',subtype='string'),
    ],
```

```
def buildPOSTGetInfo(self, address):
  elements = [{'description': 'Experience id','type': 'string'},
  {'description': 'Name of variables to be retrieved','type': 'array','subtype': 'string'}]
  return RIPMethod(
   url='%s/RIP/POST' % address,
   description='Sends a request to retrieve the value of one or more servers\' variables on demand',
   type_='POST',
   params=[
    RIPParam(name='Accept',required='no',location='header',value='application/json'),
    RIPParam(name='Content-Type',required='yes',location='header',type_='application/json'),
    RIPParam(name='jsonrpc',required='yes',type_='string',location='body',value='2.0'),
    RIPParam(name='method',required='yes',type_='string',location='body',value='get'),
    RIPParam(name='params',required='yes',type_='array',location='body',elements=elements),
    RIPParam(name='id',required='yes',type_='int',location='body'),
   ],
   returns='application/json',
   example={ '%s/RIP/POST' % address: {
    'headers': {'Accept': 'application/json','Content-Type': 'application/json'},
    'body': {'jsonrpc':'2.0', 'method':'get', 'params':['%s' % self.name, [r['name'] for r in self.readables]], 'id':'1'}
   }}
  )
def buildPOSTSetInfo(self, address):
  elements = [{'description': 'Experience id','type': 'string'},
  {'description': 'Name of variables to write','type': 'array','subtype': 'string'},
  {'description': 'Value for variables','type': 'array','subtype': 'mixed'}]
  params_post_set = [
   RIPParam(name='Accept',required='no',location='header',value='application/json'),
   RIPParam(name='Content-Type',required='yes',location='header',type_='application/json'),
   RIPParam(name='jsonrpc',required='yes',type_='string',location='body',value='2.0'),
   RIPParam(name='method',required='yes',type_='string',location='body',value='set'),
```

```
RIPParam(name='params',required='yes',type_='array',location='body',elements=elements),
    RIPParam(name='id',required='yes',type_='int',location='body'),
    RIPParam(name='variables',required='no',location='query',type_='array',subtype='string'),
  ]   example_post_set = {
    '%s/RIP/POST' % address: {
      'headers': {'Accept': 'application/json','Content-Type': 'application/json'},
      'body': {'jsonrpc':'2.0','method':'set','params':['%s' % self.name,[w['name'] for w in self.writables],['val' for w in self.writables]],'id':'1'}
    }
  }
  return RIPMethod(
    url='%s/RIP/POST' % address,
    description='Sends a request to retrieve the value of one or more servers\' variables on demand',
    type_='POST',
    params=params_post_set,
    returns='application/json',
    example=example_post_set
  )
def nextSample(self, wrap='data: %s\n\n'):
  if not self.sseRunning:
    self.sseRunning = True
    self.sampler = Sampler(self.ssePeriod)

  while self.sseRunning:
    self.sampler.wait()
    try:
      toReturn = self.getValuesToNotify()
    except:
      toReturn = 'ERROR'
    response = builder.response(result=toReturn, request_id='1')
    yield 'data: %s\n\n' % ujson.dumps(response)
```

```
def getValuesToNotify(self):
    return [
      [ 'time' ],
      [ self.sampler.lastTime() ]
    ]


class Sampler(object):


  def __init__(self, period):
    self.Ts = period
    self.reset()


  def wait(self):
    self.last = self.time
    self.time = time.time() - self.t0
    self.next = self.time / self.Ts + self.Ts
    interval = self.Ts - self.time % self.Ts
    time.sleep(interval)


  def reset(self):
    self.t0 = time.time()
    self.time = 0
    self.last = 0
    self.next = self.Ts


  def delta(self):
    return self.time - self.last


  def lastTime(self):
    return self.last
```

Clase GoPiGoContext (GoPiGoContext.py).

```
'''
Created on 02/07/2018
@author: Jorge Durán Murillas <jduranmaster@gmail.com>
'''
import json
class GoPiGoContext(object):
    def __init__(self, xmax, ymax):
        self.xmax = xmax
        self.ymax = ymax
        self.VELMOTORIZQ = 0
        self.VELMOTORDER = 0
        self.STATUS= 0
        self.BATTERY= 0
        self.ULTRASONIC= 0
        self.ENCMOTORIZQ= 0
        self.ENCMOTORDER= 0
        self.ENCSTATUS= 0
        self.wheel_base_lenght=12
        self.reset()

    def reset(self):
        self.robotPosX = self.xmax/2
        self.robotPosY = self.ymax/2
        self.obstaclePosX = []
        self.obstaclePosY = []
        self.orientacion = 1.570796
        self.DIRROBOT = 0
        self.ULTRASONICHEAD = 0
        self.msg = ""
                        self.auxiliar = {}
    def toJSON(self):
        return json.dumps(self, default=lambda o: o.__dict__,   sort_keys=True, indent=4)
```

Clase GoPiGoService (GoPiGoService.py).

```
'''
Created on 02/07/2018
@author: Jorge Durán Murillas <jduranmaster@gmail.com>
'''
import os.path
import subprocess, json
import threading
from gopigo import *
from app.GoPiGoContext import *
class GoPiGoService(object):
def __init__(self):
    '''
    Constructor
    '''
    self.modoControl = "manual"
    self.running = True
    self.m1=1
    self.m2=1
    self.gopigocontext = GoPiGoContext(100,100)
    self.velmotorder = 200
    self.velmotorizq = 200
    self.portultra=15
    self.procpid = -1
    self.script = False
self.mapping = {
     # Connection Methods
      'disconnect':self.disconnect,
      'connect':self.connect,
      # 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,
        'setledon':self.ledon,
        'setledoff':self.ledoff,
    # 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,
        'getvelmotorizq':self.getvelmotorizq,
        'getvelmotorder':self.getvelmotorder,
        'getrobotposx':self.getrobotposx,
        'getrobotposy':self.getrobotposy,
        'getobstacleposx':self.getobstacleposx,
        'getobstacleposy':self.getobstacleposy,
        'getorientacion':self.getorientacion,
        'getinfo':self.getinfo
        }
      self.readData()
```

```
# Comportamiento del 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":
          # actualizar parámetros del robot.
          self.setleftspeed(self.gopigocontext.VELMOTORIZQ)
          self.setrightspeed(self.gopigocontext.VELMOTORDER)
          self.mov(self.gopigocontext.DIRROBOT)
          self.servo(self.gopigocontext.ULTRASONICHEAD)
        time.sleep(0.1)
def connect(self):
    try:
      enable_com_timeout(2000)
      self.mov(99)
      self.gopigocontext.reset()
      self.setleftspeed(self.velmotorizq);
      self.setrightspeed(self.velmotorder);
      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.gopigocontext.msg = str("Conexion establecida con el robot GoPiGo: "+ str(version))
      return version
```

```
except:
      msg = str("Error inesperado durante el establecimiento de la conexion con el robot GoPiGo:"+ str(sys.exc_info()[0]))
      self.gopigocontext.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.gopigocontext.STATUS = read_status()
    except OSError as e:
      msg = str("Error al leer la variable - STATUS OSERROR:"+ str(e.strerror))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except:
      msg = str("Error inesperado al leer la variable - STATUS:"+ str(sys.exc_info()[0]))
      self.gopigocontext.msg = msg
      print(msg)
      pass
try:
      self.gopigocontext.BATTERY = volt()
    except OSError as e:
      msg = str("Error al leer - BATTERY OSERROR error:"+ str(e.strerror))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except:
      msg = str("Ha sucedido un error inesperado al leer - BATTERY :"+ str(sys.exc_info()[0]))
      self.gopigocontext.msg = msg
      print(msg)
```

```
try:
      self.gopigocontext.ULTRASONIC = us_dist(self.portultra)
    except OSError as e:
      msg = str("Error al ler ULTRASONIC OSERROR error:"+ str(e.strerror))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except:
      msg = str("Ha sucedido un error inesperado al leer - ULTRASONIC :"+ str(sys.exc_info()[0]))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    try:
      self.gopigocontext.ENCMOTORDER = enc_read(1)
    except OSError as e:
      msg = str("Error al leer la variable - ENCMOTORDER OSERROR error:"+ str(e.strerror))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except:
      msg = str("Error inesperado al leer la variable -  ENCMOTORDER :"+ str(sys.exc_info()[0]))
      self.gopigocontext.msg = msg
      print(msg)
      pass
try:
      self.gopigocontext.ENCMOTORIZQ = enc_read(0)
    except OSError as e:
      msg = str("Error al leer la variable - ENCMOTORIZQ OSERROR error:"+ str(e.strerror))
      self.gopigocontext.msg = msg
      print(msg)
      pass
except:
      msg = str("Error inesperado al leer la variable - ENCMOTORIZQ :"+ str(sys.exc_info()[0]))
```

```
      self.gopigocontext.msg = msg
      print(msg)
      pass
    try:
      self.gopigocontext.ENCSTATUS = read_enc_status()
    except OSError as e:
      msg = str("Error al leer la variable - ENCSTATUS OSERROR error:"+ str(e.strerror))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except:
      msg = str("Error inesperado al leer la variable - ENCSTATUS :"+ str(sys.exc_info()[0]))
      self.gopigocontext.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.gopigocontext.toJSON().encode())[0]
        env = json.loads(output.decode('utf-8'))
        self.gopigocontext.robotPosX=env['robotPosX']
        self.gopigocontext.robotPosY=env['robotPosY']
        self.gopigocontext.obstaclePosX=env['obstaclePosX']
        self.gopigocontext.obstaclePosY=env['obstaclePosY']
        self.gopigocontext.VELMOTORIZQ=env['VELMOTORIZQ']
        self.gopigocontext.VELMOTORDER=env['VELMOTORDER']
        self.gopigocontext.DIRROBOT=env['DIRROBOT']
        self.gopigocontext.orientacion=env['orientacion']
        self.gopigocontext.auxiliar=env['auxiliar']
```

```
self.gopigocontext.ULTRASONICHEAD=env['ULTRASONICHEAD']
        self.gopigocontext.msg=env['msg']


    except ValueError as e:
      msg = str("Control ValueError error:"+ str(e))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except KeyError as e:
      msg = str("Control KeyError error:"+ str(e))
      self.gopigocontext.msg = msg
      print(msg)
      pass
    except:
      msg = str("Control Unexpected error:"+ str(sys.exc_info()[0]))
      self.gopigocontext.msg = msg
      print(msg)
      pass
  def stopcontrol(self):
    if self.procpid != -1:
      try:
        self.procpid.kill()
        self.procpid = -1
      except OSError:
        self.gopigocontext.msg = sys.exc_info()[0]
        pass
  def getinfo(self):
    return self.gopigocontext.msg
  def clearinfo(self, value):
    self.gopigocontext.msg = ''
  def resetmap(self, value):
    self.enctgt(0);
    self.gopigocontext.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.gopigocontext.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.gopigocontext.msg = msg
        print(msg)
      except:
        msg = str("Get Unexpected error:"+ str(sys.exc_info()[0]))
        self.gopigocontext.msg = msg
        print(msg)
    return result
# CONTROL DEL ROBOT
  def mov(self, type):
    # reset encoders
      self.gopigocontext.DIRROBOT=1
    if type==1:
      return fwd()
```

```
elif type == 2:
      self.gopigocontext.DIRROBOT=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.gopigocontext.DIRROBOT=0
      return stop()
# Funciones de control del Servo-Motor.
  def servo(self, range):
    self.gopigocontext.ULTRASONICHEAD=range
    servo(range)
  def servoenabled(self, enabled):
    if enabled==1:
      enable_servo()
    else:
      disable_servo()
  # Funciones asociadas a los encoders.
  def setm1(self, value):
    self.m1=value
  def setm2(self, value):
    self.m2=value
  def enctgt(self, target):
    if target==0:
      self.gopigocontext.reset()
    return enc_tgt(self.m1,self.m2,target)
```

```
def getencread1(self):
    return self.gopigocontext.ENCMOTORDER
  def getencread2(self):
    return self.gopigocontext.ENCMOTORIZQ
# Funciones asociadas al control de velocidad de motores.
  def increasespeed(self, value):
    self.gopigocontext.VELMOTORDER = self.gopigocontext.VELMOTORDER + 10
    self.gopigocontext.VELMOTORIZQ = self.gopigocontext.VELMOTORIZQ + 10
    return increase_speed()
  def decreasespeed(self, value):
    self.gopigocontext.VELMOTORDER = self.gopigocontext.VELMOTORDER - 10
    self.gopigocontext.VELMOTORIZQ = self.gopigocontext.VELMOTORIZQ - 10
    return decrease_speed()
  def setleftspeed(self,value):
    self.gopigocontext.VELMOTORIZQ = value
    set_left_speed(value)
  def setrightspeed(self,value):
    self.gopigocontext.VELMOTORDER = value
    set_right_speed(value)
def setspeed(self,value):
    self.gopigocontext.VELMOTORIZQ = self.gopigocontext.VELMOTORDER = value
    set_speed(value)
  def getmotorspeed(self):
    return (self.gopigocontext.VELMOTORIZQ, self.gopigocontext.VELMOTORDER)
    # Funciones de control del puerto de ultrasonidos.
  def setportultra(self, port):
    self.portultra=port
   def getultra(self):
    return self.gopigocontext.ULTRASONIC
  # Funciones de control de estado del ROBOT.
  def getvolt(self):
    return self.gopigocontext.BATTERY
```

```
  def getstatus(self):
    return self.gopigocontext.STATUS
  def getstatusenc(self):
    return self.gopigocontext.ENCSTATUS
  def getvelmotorizq(self):
    return self.gopigocontext.VELMOTORIZQ
  def getvelmotorder(self):
    return self.gopigocontext.VELMOTORDER
def getrobotposx(self):
    return self.gopigocontext.robotPosX
  def getrobotposy(self):
    return self.gopigocontext.robotPosY
  def getorientacion(self):
    return self.gopigocontext.orientacion
  def getobstacleposx(self):
    return self.gopigocontext.obstaclePosX
  def getobstacleposy(self):
    return self.gopigocontext.obstaclePosY
    def getfwver(self):
    return fw_ver()
  #LED control
  def ledon(self, led):
    return led_on(led)
  def ledoff(self, led):
    return led_off(led)
```

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

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

Clase testHttpServerGoPiGo-ripinfoserver (testHttpServerGoPiGo-ripinfoserver.py).

```
import urllib3

import sys



if __name__ == '__main__':

 pool = urllib3.PoolManager()

 serverUrl = 'http://localhost:2055/RIP'

 headers = {'Content-Type':'application/json'}

 methodCall = [

  # Test "info" method.

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver"], "id":"1" }'



 ]

 for method in methodCall:

  response = pool.urlopen('POST', serverUrl, headers = headers, body = method)

  print(response.data)

  response.close()
```

Clase testHttpServerGoPiGo-connectmethods (testHttpServerGoPiGo-connectmethods.py).

```
import urllib3

import sys



if __name__ == '__main__':

  pool = urllib3.PoolManager()

  serverUrl = 'http://localhost:2055/RIP/POST'

  headers = {'Content-Type':'application/json'}

  methodCall = [

    # Test "connect" method.

    '{ "jsonrpc":"2.0", "method":"connect", "params":["gopigo2ripserver"], "id":"1" }',

    # '{ "jsonrpc":"2.0", "method":"disconnect", "params":["gopigo2ripserver"], "id":"1" }',

  ]

  for method in methodCall:

    response = pool.urlopen('POST', serverUrl, headers = headers, body = method)

    print(response.data)

    response.close()
```

Clase testHttpServiceGoPiGo-getmethods (testHttpServerGoPiGo-getmethods.py).

```
import urllib3

import sys

if __name__ == '__main__':

 pool = urllib3.PoolManager()

 serverUrl = 'http://localhost:2055/RIP/POST'

 headers = {'Content-Type':'application/json'}

 methodCall = [

  # Test "get" methods.

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getvolt"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getstatus"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getultra"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getstatusenc"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getencread1"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getencread2"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getreadmotorspeed"]], "id":"1" }',

  # '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getmodocontrol"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getvelmotorizq"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getvelmotorder"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getrobotposx"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getrobotposy"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getobstacleposx"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getobstacleposy"]], "id":"1" }',

  '{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getorientacion"]], "id":"1" }',

  #'{ "jsonrpc":"2.0", "method":"get", "params":["gopigo2ripserver", ["getinfo"]], "id":"1" }',

 ]

 for method in methodCall:

  response = pool.urlopen('POST', serverUrl, headers = headers, body = method)

  print(response.data)

  response.close()
```

Clase GoPiGoService (GoPiGoService.py).

```
import urllib3

import sys

import time

if __name__ == '__main__':

 pool = urllib3.PoolManager()

 serverUrl = 'http://localhost:2055/RIP/POST'

 headers = {'Content-Type':'application/json'}

 methodCall = [

  # Test "set" methods. - expid:gopigo2ripserver

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setledon"], [0]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setledoff"], [0]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setledon"], [1]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setledoff"], [1]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setspeed"], [200]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setleftspeed"], [100]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setrightspeed"], [100]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setincreasespeed"], [10]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setdecreasespeed"], [10]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setservoEnabled"], [0]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setservoEnabled"], [1]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setservo"], [90]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setservo"], [110]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setservo"], [70]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setservo"], [90]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setm1"], [true]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setm2"], [true]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setctgt"], [0]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setportultra"], [15]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setresetmap"], [0]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setscript"], [" "]], "id":"1" }',

            '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setclearscript"], [0]], "id":"1" }',
```

```
'{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [1]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [99]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [2]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [99]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [3]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [99]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [4]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [99]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [5]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [99]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [6]], "id":"1" }',
  '{ "jsonrpc":"2.0", "method":"set", "params":["gopigo2ripserver", ["setmov"], [99]], "id":"1" }',
 ]
 for method in methodCall:
  response = pool.urlopen('POST', serverUrl, headers = headers, body = method)
  time.sleep(2)
  print(response.data)
  response.close()
```

ANEXO C.

En este anexo se presenta la relación completa de paneles (componentes gráficos) con respecto al código *JavaScript* incluido en los manejadores que hacen uso del elemento *"RIPClient"*.

Panel	Acción	Comando

Sub-Panel de *"Estado" – "pnRobot"*	"Conectar"	RIPClient.get(['connect'], function(result) {
		_view.txtversion.setValue(result[0]);
		ClearObstaclesMap();
		})

Sub-Panel de *"Estado" – "pnRobot"*	"Desconectar"	RIPClient.get(['disconnect'], function(result) {
		connect=false;
		_view.txtversion.setValue("0.000");
		_stop()
		});

Sub-Panel de control de Motores – *"pnmotors"*	Vel. Motores → "set"	RIPClient.set(["setspeed"],[parseInt(_view.txtmotorspeed.getValue())]); _view.txtmotorspeed.setValue(parseInt(_view.txtmotorspeed.getValue()));
Sub-Panel de control de Motores – *"pnmotors"*	Vel. Motores → "reset"	RIPClient.set(["setspeed"],[200]);
Sub-Panel de control de Motores – *"pnmotors"*	Vel. Izquierdo → "set"	RIPClient.set(["setleftspeed"],[parseInt(_view.txtspeedleft.getValue())]); _view.txtspeedleft.setValue(parseInt(_view.txtspeedleft.getValue()))
Sub-Panel de control de Motores – *"pnmotors"*	Vel. Derecho → "set"	RIPClient.set(["setrightspeed"],[parseInt(_view.txtspeedright.getValue())]); _view.txtspeedright.setValue(parseInt(_view.txtspeedright.getValue()))
Sub-Panel de control de Motores – *"pnmotors"*	Incrementar → "+10"	RIPClient.set(["increasespeed"],[10]);
Sub-Panel de control de Motores – *"pnmotors"*	Decrementar → "-10"	RIPClient.set(["decreasespeed"],[10]);

Tabla 48. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – I.

Panel	Acción	Comando
Sub-Panel de control de Servo-Motor – *"pnservo"*	"Activar"	RIPClient.set(["servoEnabled","servo"],[1,90]);
Sub-Panel de control de Servo-Motor – *"pnservo"*	"Desactivar"	RIPClient.set(["servoEnabled"],[0]);

Sub-Panel de control de Servo-Motor – *"pnservo"*	Movimiento → "←"	rangeServo=rangeServo+10;
		if (rangeServo<0)
		rangeServo=0;
		RIPClient.set(["servo"],[rangeServo]);

Sub-Panel de control de Servo-Motor – *"pnservo"*	Movimiento → "→"	rangeServo=rangeServo-10;
		if (rangeServo<0)
		rangeServo=0;
		RIPClient.set(["servo"],[rangeServo])

Sub-Panel de control de Servo-Motor – *"pnservo"*	Movimiento → "reset"	rangeServo=90; RIPClient.set(["servo"],[rangeServo])

Sub-Panel de control de Servo-Motor – *"pnservo"*	Movimiento → "scan"	rangeServo=0;
		do
		{
		RIPClient.set(["servo"],[rangeServo]);
		RIPClient.get(['ultra'], function(result) {
		_view.txtultrasensor.setValue(result[0]);
		});
		rangeServo=rangeServo+10;
		
		}while(rangeServo<=180);

Sub-Panel de control de Encoders – *"pnencoder"*	"set"	RIPClient.set(["setm1"],[encoder1]); RIPClient.set(["setm2"],[encoder2]); RIPClient.set(["enctgt"],[parseInt(_view.txtencoder.getValue())]); _view.txtencoder.setValue(parseInt(_view.txtencoder.getValue()));
Sub-Panel de control de Encoders – *"pnencoder"*	"reset"	RIPClient.set(["setm1"],[encoder1]); RIPClient.set(["setm2"],[encoder2]); RIPClient.set(["enctgt"],[0]);
Sub-Panel de control de Sensor de Ultrasonidos – *"pnultrasonic"*	"set"	RIPClient.set(["setportultra"],[parseInt(_view.txtultrasonic.getValue())]); _view.txtultrasonic.setValue(parseInt(_view.txtultrasonic.getValue()))

Tabla 49. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – II.

Panel	Acción	Comando
Sub-Panel de control de Sensores – *"pnsensores"*	"Actualizar Sensor"	RIPClient.get(['ultra','status','volt','encread1','encread2','readmotorspeed'])
Sub-Panel de control de Movimiento – *"pnopciones moves"*	"←"	RIPClient.set(["mov"],[movLeft]);
Sub-Panel de control de Movimiento – *"pnopciones moves"*	"→"	RIPClient.set(["mov"],[movRight]);
Sub-Panel de control de Movimiento – *"pnopciones moves"*	" ↑ "	RIPClient.set(["mov"],[movForward]);
Sub-Panel de control de Movimiento – *"pnopciones moves"*	" ↓ "	RIPClient.set(["mov"],[movBack]);
Sub-Panel de control de Movimiento – *"pnopciones moves"*	"Stop"	RIPClient.set(["mov"],[99]);

Sub-Panel de control de Movimiento – *"pnopciones moves"*	"Actualizar"	RIPClient.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); });

Sub-Panel de Actualización de código – *"pnbotones"*	"Cargar Script"	RIPClient.set(["script"],[_view.content.getValue()]); ;
Sub-Panel de Actualización de código – *"pnbotones"*	"Borrar Script"	RIPClient.set(["clearscript"],[0]);

Sub-Panel de Actualización de Logs – *"pnlogger"*	"Actualizar Log"	RIPClient.get(['getinfo',], function(result) {
		trace = _view.log.getValue();
		msg = result[0];
		if (msg!=''){
		trace = trace +'\n'+msg;
		_view.log.setValue(trace);
		}
		});

Tabla 50. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – III.

Panel	Acción	Comando

Sub-Panel de control de Opciones de Mapa – *"pnopcionesmapa"*	"Reiniciar Mapa"	index=0;
		while(index<nPoints){
		x[index]=0;
		y[index]=0;
		index=index+1;
		}
		nPoints=0;
		RIPClient.set(["resetmap"],[0]);

Sub-Panel de control de Opciones de Mapa – *"pnopcionesmapa"*	"Actualizar Automáticamente Mapa"	RIPClient.get(['getrobotx','getroboty','gethead','getobstaclex','getobstacley'])
Sub-Panel de control de Opciones de Mapa – *"pnopcionesmapa"*	"Modo Manual"	RIPClient.set(["modocontrol"],["manual"]);
Sub-Panel de control de Opciones de Mapa – *"pnopcionesmapa"*	"Modo Automático"	RIPClient.set(["modocontrol"],["automatico"]);

Tabla 51. Relación de componentes gráficos de la aplicación de control EjsS respecto al elemento RIPClient – IV.

A continuación se presente el código fuente de las funciones que se ejecutan de forma automática en la aplicación EjsS de control del robot y cuya misión es actualizar el valor de ciertas variables de estado del robot así como mantener actualizado el mapa de ocupación de la aplicación. Estas funciones estás definidas en el panel *"Propio"* de la pestaña *"Modelo"* en la interfaz de desarrollo del framework EjsS.

```
function UpdateScreen() {
  if(!connect && _view.txtversion.getValue()=='1.7' ){
    connect = true;
    }
  }
function UpdateSensor() {
  if ( connect && updateautomatic) {
    ActualizarSensores();
  }
}
```

```
function UpdateMapa(){
  if ( connect && mapaautomatic) {
    UpdatePositionRobot();
  }
 }
function UpdateInfo(){
  if ( connect && logautomatic) {
    GetInfo();
  }
}
function ActualizarSensores () {
  RIPServer.get(['getultra','getstatus','getvoltage','getencread1','getencread2','getreadmotorspeed'],
function(result) {
  _view.txtultrasensor.setValue(result[0]);
  _view.txtstatus.setValue(result[1]);
  _view.txtbateria.setValue(result[2]);
  _view.txtdistancemotorleft.setValue(result[4]);
  _view.txtdistancemotorright.setValue(result[3]);
  _view.txtreadmotorspeed.setValue(result[5]);
  });
}
function ClearObstaclesMap(){
   index=0;
   while(index<nPoints){
     x[index]=0;
     y[index]=0;
     index=index+1;
}
  nPoints=0;
```

```
function GetInfo(){
  RIPServer.get(['getinfo',], function(result) {
    trace = _view.log.getValue();
    msg = result[0];
    if (msg!=''){
      trace = trace +'\n'+msg;
      _view.log.setValue(trace);
    }
    });
}
function UpdatePositionRobot(){
  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 = obstaclesPosX.length;
  ClearObstaclesMap();
  index = 0;
  while(index<num && index<maxPoints){
     x[index]=obstaclesPosX[index];
     y[index]=obstaclesPosY[index];
     index+=1;
  }
  nPoints = num;
  _view.log.setValue(nPoints);
});
}
```

ANEXO D.

Código Fuente. Script 1. Control Básico de Evitación de Obstáculos.

```
import sys, json

import math

from random import randint

while True:

    obj = sys.stdin.read()

    if not obj:

        break;

    gopigocontext =json.loads(obj)


    # constantes

    velocidad_izquierda = 80

    velocidad_derecha = 81

    velocidad_giro_high = 100

    velocidad_giro_slow = 20

    obstaculo_min = 100

    error_encoder = 100

    # recuperar parámetros del contexto.

    d_left_wheel = gopigocontext['ENCMOTORIZQ']

    d_right_wheel = gopigocontext['ENCMOTORDER']

    dis_ultrasonic = gopigocontext['ULTRASONIC']

    orientacion_ultrasonic = gopigocontext['ULTRASONICHEAD']

    dir_motor = gopigocontext['DIRMOTOR']

    prev_x = gopigocontext['robotPosX']

    prev_y = gopigocontext['robotPosY']

    xmax = gopigocontext['xmax']

    ymax = gopigocontext['ymax']

    prev_theta = gopigocontext['orientacion']

    auxiliar = gopigocontext['auxiliar']

    wheel_base_lenght = gopigocontext['wheel_base_lenght']
```

```
# calcular los movimientos de las ruedas.
    prev_left_wheel=0
    prev_right_wheel=0
    if 'd_left_wheel' in auxiliar:
        prev_left_wheel = auxiliar['d_left_wheel']
        prev_right_wheel = auxiliar['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:
        # calcular la nueva posición del robot.
        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
```

```
# Calcular la posición del obstáculo
        xobs = -1
        yobs = -1
        if dis_ultrasonic < obstaculo_min:
            xobs = new_x + ((dis_ultrasonic+12)*math.cos(new_theta))/10
            yobs = new_y + ((dis_ultrasonic+12)*math.sin(new_theta))/10


        # Calcular la velocidad de los motores
        rand = -1
        if 'rand' in auxiliar:
            rand = auxiliar['rand']
        if rand == -1:
            rand = randint(0,9)
        if dir_motor == 0:
            dir_motor = 99


        if dir_motor != 2 :
            dir_motor = 99
        else:
            dir_motor = 99
       # Salvar datos en el contexto.
    gopigocontext['robotPosX']=new_x
   gopigocontext['robotPosY']=new_y
    gopigocontext['orientacion']=new_theta
    gopigocontext['obstacleX']=xobs
    gopigocontext['obstacleY']=yobs
    gopigocontext['DIRMOTOR']=dir_motor
    gopigocontext['ULTRASONICHEAD']=90
    auxiliar['d_left_wheel']=d_left_wheel
    auxiliar['d_right_wheel']=d_right_wheel
    auxiliar['rand']=rand
       # return data
    print(json.dumps(gopigocontext))
```

Código Fuente. Script 2. Navegación con Evitación de Obstáculos.

```
import sys, json
import math
from random import randint
while True:
    obj = sys.stdin.read()
    if not obj:
        break;
    gopigocontext =json.loads(obj)

    # constantes
    velocidad_izquierda = 80
    velocidad_derecha = 81
    velocidad_giro_high = 100
    velocidad_giro_slow = 20
    obstaculo_min = 100
    error_encoder = 100
    # recuperar parámetros del contexto.
    d_left_wheel = gopigocontext['ENCMOTORIZQ']
    d_right_wheel = gopigocontext['ENCMOTORDER']
    dis_ultrasonic = gopigocontext['ULTRASONIC']
    orientacion_ultrasonic = gopigocontext['ULTRASONICHEAD']
    dir_motor = gopigocontext['DIRMOTOR']
    prev_x = gopigocontext['robotPosX']
    prev_y = gopigocontext['robotPosY']
    xmax = gopigocontext['xmax']
    ymax = gopigocontext['ymax']
    prev_theta = gopigocontext['orientacion']
    auxiliar = gopigocontext['auxiliar']
    wheel_base_lenght = gopigocontext['wheel_base_lenght']
```

```
# calcular los movimientos de las ruedas.
    prev_left_wheel=0
    prev_right_wheel=0
    if 'd_left_wheel' in auxiliar:
        prev_left_wheel = auxiliar['d_left_wheel']
        prev_right_wheel = auxiliar['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:
        # calcular la nueva posición del robot.
        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
```

```
# Calcular la posición del obstáculo
    xobs = -1
    yobs = -1
    if dis_ultrasonic < obstaculo_min:
        xobs = new_x + ((dis_ultrasonic+12)*math.cos(new_theta))/10
        yobs = new_y + ((dis_ultrasonic+12)*math.sin(new_theta))/10

    # Calcular la velocidad de los motores
    rand = -1
    if 'rand' in auxiliar:
        rand = auxiliar['rand']
    if rand == -1:
        rand = randint(0,9)
    if dir_motor == 0:
        dir_motor = 1
     if dir_motor != 2 :
        # El robot se mueve hacía adelante
        if dis_ultrasonic < 10:
            # girar a la derecha
            rand = -1
            gopigocontext['DIRMOTOR']=2
            gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
            gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
        elif dis_ultrasonic < 20:
            gopigocontext['DIRMOTOR']=1
            if rand%2==0:
                gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
                gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
            else:
                gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
                gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
```

```
elif dis_ultrasonic < 50:
            # retroceder
             gopigocontext['DIRMOTOR']=1
             if rand%2==0:
                  gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
                  gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
             else:
                  gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
                  gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
        else:
            rand = -1
            gopigocontext['DIRMOTOR']=1
            gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
            gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
    else:
        if dis_ultrasonic<20:
            # girar a la izquierda
            gopigocontext['DIRMOTOR']=2
            if rand%2==0:
                gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_high
                gopigocontext['VELLEFTMOTOR'] = velocidad_giro_slow
            else:
                gopigocontext['VELRIGHTMOTOR'] = velocidad_giro_slow
                gopigocontext['VELLEFTMOTOR'] = velocidad_giro_high
        elif dis_ultrasonic < 50:
            gopigocontext['DIRMOTOR']=2
            gopigocontext['VELRIGHTMOTOR'] = velocidad_izquierda
            gopigocontext['VELLEFTMOTOR'] = velocidad_derecha
        else:
            gopigocontext['DIRMOTOR']=1
            gopigocontext['VELLEFTMOTOR'] = velocidad_izquierda
            gopigocontext['VELRIGHTMOTOR'] = velocidad_derecha
```

```
# Salvar datos en el contexto.
    gopigocontext['robotPosX']=new_x
    gopigocontext['robotPosY']=new_y
    gopigocontext['orientacion']=new_theta
    gopigocontext['obstacleX']=xobs
    gopigocontext['obstacleY']=yobs
    gopigocontext['DIRMOTOR']=dir_motor
    gopigocontext['ULTRASONICHEAD']=90
    auxiliar['d_left_wheel']=d_left_wheel
    auxiliar['d_right_wheel']=d_right_wheel
    auxiliar['rand']=rand


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

Código Fuente. Programa 1. Evitación de Obstáculos Simple.

```
from gopigo import *
import time


#Distancia al obstáculo a la cual se debe detener el robot.
distance_to_stop = 20
#orden de avance del robot.
fwd()


while True:
    # Obtener una lectura del sensor de ultrasonidos
    dist = us_dist(15)
    # Si el objeto detectado se encuentra por debajo del umbral
    # entonces se debe detener el robot GoPiGo-2.
    if dist < distance_to_stop:
        # orden de parada del robot.
        stop()
        break
    time.sleep(.1)
```

Código Fuente. Programa 2. Mapeo de Entorno con Evitación de Obstáculos.

```
from gopigo import *
import sys
from collections import Counter
import math
def us_map():
        delay=.02
        debug =0
        num_of_readings=45
        incr=90/num_of_readings
        ang_l=[0]*(num_of_readings+1)
        dist_l=[0]*(num_of_readings+1)
        x=[0]*(num_of_readings+1)
        y=[0]*(num_of_readings+1)
        buf=[0]*40
        ang=0
        lim=250         # limite de distancia en las lecturas del sensor.
        index=0
        sample=2  # número de muestras.
        print "Getting the data"
        while True:
                # Tomar las medidas obtenidas por el sensor de ultrasonidos.
                for i in range(sample):
                        dist=us_dist(15)
                        if dist<lim and dist>=0:
                                buf[i]=dist
                        else:
                                buf[i]=lim
                # Encontrar la muestra que más se repite dentro del ángulo de escaneo.
                max=Counter(buf).most_common()
                rm=-1
```

```
rm=lim


                    if debug==1:
                            print index,ang,rm
                    ang_l[index]=ang
                    dist_l[index]=rm
                    index+=1


                    # Mover el servo al siguiente ángulo.
                    servo(ang)
                    time.sleep(delay)
                    ang+=incr
                    if ang>180:
                            break


            grid_size=51
            # convertir la distancia y el ángulo a coordenadas X e Y.
            for i in range(num_of_readings+1):
                    x[i]=(int(dist_l[i]*math.cos(math.pi*(ang_l[i])/180))/10)
                    y[i]=int(dist_l[i]*math.sin(math.pi*ang_l[i]/180))/10


            for i in range(num_of_readings+1):
                    x[i]=(grid_size/2)-x[i]
                    y[i]=(grid_size/2)-y[i]


            # Crear el grid donde se mostrara la información (consola de comandos).
            grid = [[0 for a in xrange(grid_size)] for a in xrange(grid_size)]
            for i in range (num_of_readings+1):
                    if dist_l[i]<>lim:
                            grid[y[i]][x[i]]=1
            fence='-'*(grid_size+1)
```

```
# Pintar el mapa de ocupación.
        print "Map:"
        print fence*2
        for i in range(grid_size/2):
                print "|",
                for j in range (grid_size):
                        if (j==grid_size/2)and i==(grid_size/2)-1:
                                print 'x',
                        elif grid[i][j]==0:
                                print ' ',
                        else:
                                print 'o',
                print "|"
        print fence*2
        return min(dist_l) # devolver la distancia más corta.
stop()
while True:
        # Mover el robto hacía adelante, parar crear el mapa y volver a mover el robot
        # hacía adelante.
        enable_com_timeout(2000)
        enc_tgt(1,1,18)         #SFijar el encoder. Parar después de 4 rotacfiones de las ruedas.
        fwd()
        time.sleep(.2)
        while True:
                enc=read_enc_status()
                ts=read_timeout_status()
                time.sleep(.05)
                print enc,ts
                if enc == 0:
                        break
                if  ts==0:
                        break
```

```
time.sleep(2)
        enable_servo()
        # Detener el robot si algún objeto se ecneuntra a una distancia inferior a 20 cm.
        if us_map() <20:
                break
        disable_servo()
```