UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

TRABAJO FIN DE MÁSTER

CONTROL DE UN ROBOT DE BALANCEO

Autor: Mikel Iturrioz Dorronsoro

Equipo docente: Luis de la Torre Cubillo
Dictino Chaos García

Curso Académico: 2020/2021

Convocatoria: septiembre de 2021

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

TRABAJO FIN DE MÁSTER

TIPO A: PROYECTO ESPECÍFICO PROPUESTO POR UN PROFESOR

CONTROL DE UN ROBOT DE BALANCEO

Autor: Mikel Iturrioz Dorronsoro

Equipo docente: Luis de la Torre Cubillo
Dictino Chaos García

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:

Mikel Iturrioz Dorronsoro

Resumen

En este trabajo fin de máster (TFM) se hace uso del robot GoPiGo3 con el Kit BalanceBot, convirtiendo el robot clásico en una suerte de péndulo invertido. Este robot se basa en la placa Raspberry Pi 3 que se comunica directamente con la placa de circuito GoPiGo3 Dexter Industries.

En la placa Raspberry Pi se ejecuta un servidor RIP. Este servidor, por un lado, utilizando la API proporcionada por GoPiGo3 envía órdenes a los actuadores del robot o lee datos de los sensores que tiene implementados. Para que eso sea posible la placa de circuito GoPiGo3 Dexter Industries hace de interfaz entre la placa Raspberry Pi y el robot GoPiGo3. Por otro lado, el servidor RIP permite establecer conexión entre el robot GoPiGo3 (servidor RIP) y el cliente RIP, que en este caso es una interfaz web desarrollada mediante la herramienta EjsS.

La aplicación informática escrita en JavaScript se puede visualizar utilizando cualquier navegador web, y dota de todas las funciones necesarias para realizar el control del robot GoPiGo3. De este modo, el usuario puede controlar el robot remotamente.

El robot GoPiGo3 BalanceBot es un robot de dos ruedas de equilibrio vertical. Este sistema naturalmente es inestable, por lo que se ha tenido que diseñar e implementar un controlador PID en el robot para mantener erguido sobre sus dos ruedas. Utilizando un giroscopio se mide el ángulo de inclinación del eje vertical, y la medida se retroalimenta al algoritmo del controlador PID. El controlador PID procesa, calcula y genera la salida de velocidad correspondiente para controlar los motores y mantener la posición vertical del robot GoPiGo3.

Palabras clave

GoPiGo3, BalanceBot, Dexter industries, API, Python, Raspberry Pi, Raspbian, servidor RIP, elemento RIP, EjsS, interfaz web, aplicación informática, IMU, PID.

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE CÓDIGOS

1. Introducción

Un robot de equilibrio de dos ruedas es un sistema de **péndulo invertido** (Inverted Pendulum o IP [1]) que se mantiene erguido sobre dos ruedas. El péndulo invertido es un problema de control clásico y se encuentra entre los sistemas más difíciles de controlar en el campo de la ingeniería de control. El sistema de péndulo invertido, a diferencia de muchos otros sistemas de control, es naturalmente inestable. Por lo tanto, el sistema debe controlarse para alcanzar la estabilidad en este estado inestable. Para equilibrar un robot de péndulo invertido de dos ruedas, es necesario tener información precisa del ángulo de inclinación mediante el uso de una medición en él. Además, es necesario implementar un controlador para compensar dicha inclinación. Utiliza los giroscopios y acelerómetros para detectar la inclinación del eje vertical, y el controlador genera señales de par a cada motor para evitar que el sistema caiga al suelo.

A diferencia de otros robots móviles, el robot de equilibrio tiene los beneficios de su pequeña dimensión, versatilidad y bajo costo. La investigación de los robots equilibradores de dos ruedas ha aumentado en los últimos años debido a la invención de la aplicación de transporte humano, Segway [2]. Este proyecto en particular consiste en el modelado del robot, diseñar un controlador PID (Proportional-Integral-Derivative controller o controlador Proporcional-Integral-Derivativo) e implementar el controlador en el robot de dos ruedas. De este modo ajusta su posición cuando está a punto de caer hacia adelante o hacia atrás para evitar la inestabilidad y conseguir el equilibrio.

En la *Figura 1* se muestra el modelo Segway PT (Segway Personal Transporter) inventado por Dean Kamen [2], el fundador del fabricante estadounidense Segway Inc. Este modelo es una aplicación de transporte humano de un robot de dos ruedas. Este robot es un transportador humano eléctrico y autoequilibrado con un complejo sistema de control y estabilización giroscópica controlado por computadora. El dispositivo se balancea sobre dos ruedas paralelas y se controla moviendo el peso del cuerpo. Al utilizar esta tecnología, el usuario puede atravesar pequeños pasos o bordillos y permite una fácil navegación en varios terrenos. Solo se necesitan tres giroscopios para todo el sistema, los sensores adicionales se incluyen como precaución de seguridad.

Figura 1: *Segway PT [2]*

Para la realización de este proyecto se ha utilizado el robot **GoPiGo3** con el Kit **BalanceBot** [10], convirtiendo el robot clásico en una suerte de péndulo invertido. El objetivo del proyecto es controlar dicho robot, consiguiendo mantener el equilibrio del robot sobre las dos ruedas que tiene.

El robot GoPiGo3 es un robot basado en la placa **Raspberry Pi 3** [11]. En esta placa se ejecuta el código programado en Python [12] para el control del robot. Raspberry Pi se comunica directamente con la **placa de circuito GoPiGo3 Dexter Industries**, de modo que utilizando la **API** (Application Programming Interface o Interfaz de Programación de Aplicaciones) proporcionada por GoPiGo3 [13][14][15] se envían órdenes a los actuadores del robot y/o se leen datos de los distintos sensores que tiene el robot. Por otro lado, en la placa Raspberry Pi se ejecuta un **servidor RIP** (Remote Interoperability Protocol o Protocolo de Interoperabilidad Remota) [16]. Esto permite establecer conexión entre una aplicación informática y el robot, mediante una red Wifi.

Por ende, se desarrolla una interfaz web utilizando la herramienta **EjsS** (Easy Java/Javascript Simulations) [17]. Esta interfaz dotará de todas las funciones necesarias para realizar un control completo del robot GoPiGo3. La comunicación entre el servidor RIP que se ejecuta en la placa Raspberry Pi y la interfaz web EjsS es posible porque hablan el mismo protocolo. Es decir, en ambas entidades de software se implementa el protocolo RIP.

En otros trabajos se han planteado soluciones parecidas. En el caso concreto de [18][19] se ha desarrollado un robot en forma de coche, utilizando la plataforma robótica GoPiGo2, el servidor RIP y la herramienta EjsS. En este caso se hace uso del robot GoPiGo3 con el Kit BalanceBot, desarrollando un robot de balanceo.

Poniendo la mirada en este proyecto, los objetivos concretos son:

1. Puesta a punto del robot y familiarización con su funcionamiento básico: envío de órdenes de los actuadores y lectura de los sensores entre otras tareas.
2. Diseño de un controlador que permita el correcto equilibrio del robot de balanceo, su desplazamiento sobre una superficie mientras se encuentra en posición invertida y un rechazo a pequeñas perturbaciones.
3. Diseño de una aplicación informática que pueda usarse para realizar el control remoto del robot.

2. Estado del Arte

La robótica es la rama de la ingeniería mecánica, de la ingeniería eléctrica, de la ingeniería electrónica, de la ingeniería biomédica y de las ciencias de la computación, que se ocupa del diseño, construcción, operación, estructura, manufactura y aplicación de los robots.

La robótica ha ido integrándose poco a poco tanto en la industria como en la sociedad, dando solución a problemas que hace unos años serían imposibles de solucionar. Los robots móviles se implementan cada vez más en la actualidad y se utilizan en una variedad de aplicaciones diferentes, que incluyen exploración, búsqueda y rescate, manipulación de materiales en áreas peligrosas y entretenimiento. Los robots con patas pueden pasar por encima de obstáculos, pero, su diseño y control son más complejos debido a la mayor cantidad de grados de libertad. Los robots con ruedas son más eficientes energéticamente, tienen una estructura mecánica más simple, y una dinámica más simple en comparación con la que requieren los robots con patas para hacer contacto con el suelo y proporcionar una fuerza motriz. Los robots con al menos tres ruedas pueden lograr estabilidad estática, simplificando aún más la dinámica.

Un robot de equilibrio de dos ruedas es un tipo importante de robots móviles. Ajusta su posición cuando está a punto de caer hacia adelante o hacia atrás para evitar la inestabilidad y conseguir así el equilibrio. A lo largo de la historia se han creado diferentes prototipos o modelos de vehículo de dos ruedas. En 1988 se dio uno de los primeros reportes sobre la implementación de un péndulo invertido en dos ruedas, por T. Kanamura.

En el año 2001 se revela la aplicación comercial más conocida del péndulo invertido sobre dos ruedas, el Segway (ver *Figura 1*), inventado por Dean Kamen [2].

Felix Grasser [3], un investigador del Instituto Federal Suizo de Tecnología construyó JOE en el año 2002, un prototipo de un vehículo de dos ruedas (ver *Figura 2*). Debido a su configuración con dos ruedas coaxiales, cada una de las cuales es acoplado a un motor de corriente continua, el vehículo puede hacer vueltas en U estacionarias. Un sistema de control, compuesto por dos controladores espaciales de estado lineal que utiliza información sensorial de un giroscopio, pilota los motores para mantener el sistema en equilibrio.

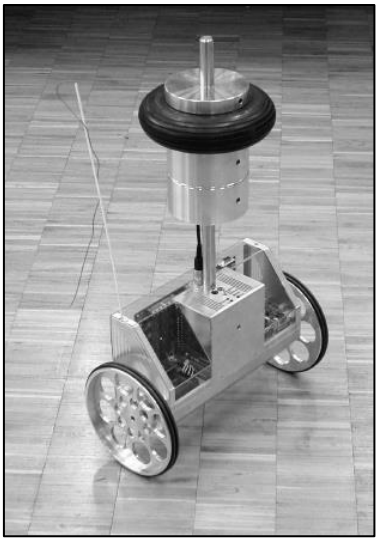


Figura 2: *JOE [3]*

Anderson [4], construyo el robot nBot en el año 2003 (ver *Figura 3*). La idea básica de nBot es bastante simple: accionar las ruedas en la dirección en la que cae la parte superior del robot. Si las ruedas pueden accionarse de tal manera que permanezcan por debajo del centro de gravedad del robot, el robot permanece equilibrado. Esto requiere dos sensores de retroalimentación: un sensor de inclinación o ángulo para medir la inclinación del robot con respecto a la gravedad y codificadores de rueda para medir la posición de la base del robot. Las medidas se retroalimentan a la plataforma como un voltaje del motor, que es proporcional al par, para equilibrar y accionar el robot.

Figura 3: *nBot [4]*

La investigación de vehículos y robots cuya esencia está basada en la teoría del péndulo invertido y autoequilibrio ha ganado gran importancia a lo largo de la última década entre investigadores y entusiastas de la robótica. El robot de equilibrio de dos ruedas es un sistema de péndulo invertido que tiene una propiedad inestable, multivariable, complicada y no lineal. Por lo tanto, es necesario investigar una técnica y un método de sistema de control adecuados para controlar el sistema. Es decir, el robot equilibrador de dos ruedas es una aplicación del péndulo invertido que requiere diseñar e implementar un controlador en el robot para mantener su posición vertical. Se pueden implementar varios tipos de controladores en un robot equilibrador de dos ruedas, como LQR (Linear Quadratic Regulator o Regulador Cuadrático Lineal), PPC (Pole Placement Controller o Controlador por Asignación de Polos), FLC (Fuzzy Logic Controller o Controlador Lógico Difuso), SMC (Sliding Mode Controller o Controlador de Modo Deslizante) y PID [33].

A continuación, se muestran varios trabajos empleando los diferentes controladores mencionados anteriormente [34]:

- **Tipo de controlador LQR** [5]: en este artículo se construyen modelos cinemáticos y dinámicos y se implementan dos métodos de control: PID y LQR. Se controlan dos subsistemas: autoequilibrio (evitando que el sistema se caiga cuando se mueve hacia adelante o hacia atrás) y la rotación (regulación del ángulo de dirección cuando gira hacia la izquierda o hacia la derecha). El controlador PID se usa para controlar los dos subsistemas, y LQR en cambio, se usa para controlar solo el subsistema de autoequilibrio. Se utiliza Matlab para la simulación y se comparan los dos métodos. El resultado muestra que LQR tiene un mejor desempeño que PID para el control del subsistema de autoequilibrio.

- **Tipo de controlador PPC** [6]: este artículo presenta una evaluación comparativa entre controladores de tipo PPC y LQR para controlar el equilibrio de un robot móvil de dos ruedas. Se aplica perturbación para probar el equilibrio del robot. Se analiza la simulación en la aplicación Matlab y se observa el rendimiento de la posición, la velocidad, el ángulo y la tasa de ángulo del robot de equilibrio para encontrar el mejor controlador para el modelo de robot de equilibrio de dos ruedas. Se deduce que la técnica PPC ofrece mejor rendimiento en comparación con la técnica LQR.
- **Tipo de controlador FLC** [7]: en este trabajo se realizan algunos experimentos de simulación en el entorno no perturbado y perturbado. Se construye la ecuación cinética utilizando la teoría de la mecánica dinámica de Newton, y se diseñan el controlador PPC y FLC. Los resultados de la simulación indican que el algoritmo FLC puede realizar el control de autoequilibrio del robot de dos ruedas con éxito y evitar que el robot se caiga, a fin de satisfacer los objetivos de control anticipados del robot y obtener el mejor rendimiento dinámico. El controlador FLC proporciona una mayor robustez en comparación con el método PPC.
- **Tipo de controlador SMC** [8]: este artículo trata sobre el modelado de péndulo invertido de dos ruedas y el diseño del control PISMC (Proportional Integral Sliding Mode Controller o Controlador de Modo Deslizante Integral Proporcional) para el sistema. Se deriva el modelo matemático del sistema de péndulo invertido de dos ruedas que es altamente no lineal. Se propone un controlador robusto basado en el control SMC para realizar la estabilización robusta y el rechazo de perturbaciones del sistema. Se realiza un estudio de simulación por computadora para acceder al desempeño de la ley de control propuesta. El resultado de la simulación muestra con éxito que el robot equilibrador de dos ruedas que utiliza SMC tiene una buena respuesta para lograr la característica deseada en comparación con PPC.

Todos esos trabajos son factibles para la simulación, pero todavía no se han implementado esas técnicas en el hardware. Hay muchos problemas para implementarlos en la práctica, especialmente en LQR, FLC y SMC, aunque se pueden usar simulaciones para mostrar la robustez a las perturbaciones y las incertidumbres del modelo. En este trabajo se ha optado por el controlador **PID** [33]. El PID ha demostrado ser popular entre la comunidad de ingenieros de control, puesto que ha demostrado tener un excelente comportamiento en el control de un sistema inestable. Este controlador es capaz de tomar la acción correcta con ciertas deficiencias en las regiones de error máximo y mínimo que podrían resolverse en el futuro mediante la programación de ganancias [9].

3. Arquitectura y herramientas

A continuación, se muestran los elementos principales utilizados para el desarrollo de este proyecto:

- **Robot GoPiGo3**: se hace uso del robot GoPiGo3 de Dexter Industries [10]. En concreto se utiliza el Kit BalanceBot. Este kit permite convertir el robot clásico en una suerte de péndulo invertido.
- **Placa Raspberry Pi**: el robot GoPiGo3 está basado en Raspberry Pi [11]. En esta placa se ejecuta el código Python [12] desarrollado para el funcionamiento del robot, así como el servidor RIP para establecer comunicación con el usuario (aplicación informática).
- **Placa de circuito Dexter Industries GoPiGo3**: esta placa sirve de interfaz entre la placa Raspberry Pi y el robot. Entre otras tareas, envía órdenes a los actuadores (motores, por ejemplo) y lee datos de los sensores (sensor IMU y receptor de infrarrojos, por ejemplo).
- **Servidor RIP**: el servidor RIP [16] es un servidor de comunicaciones con implementaciones en LabVIEW y Python. En este trabajo se ha usado la segunda, ya que se ejecuta en la placa Raspberry Pi del robot y la implementación en Python ofrece más facilidades de comunicación con las APIs nativas del GoPiGo3. Mediante este protocolo de comunicaciones se realiza la operación remota desde un navegador con el robot, es decir, el servidor RIP permite la comunicación entre la aplicación informática desarrollada y el robot.
- **Aplicación informática (interfaz EjsS)**: se ha desarrollado una interfaz web mediante la herramienta EjsS [17] para realizar el control del robot. Esta interfaz se ejecutará en el PC del usuario. De este modo el usuario podrá operar con el robot remotamente.

En la *Figura 4* se observa la interconexión de los diferentes elementos utilizados.

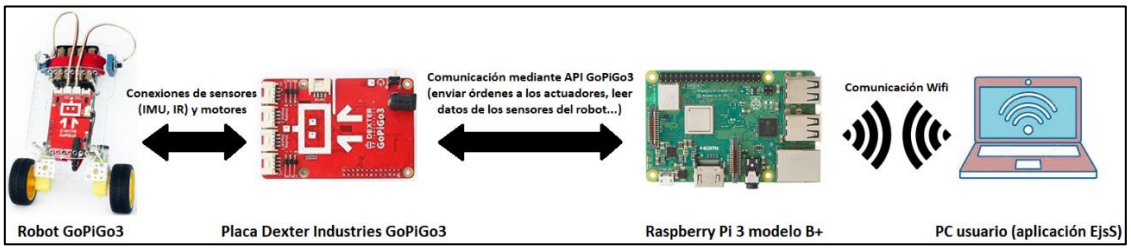


Figura 4: *Interconexión de los elementos*

Tal y como se observa en la *Figura 4*, la comunicación entre el PC del usuario (interfaz web) y el servidor RIP que se está ejecutando en la placa Raspberry Pi se realiza mediante una red Wifi. De este modo se consigue controlar el robot de forma remota.

3.1 Descripción de los elementos del hardware

A continuación, se analizarán los diferentes elementos del robot. En el apartado *9.1 Detalles de los elementos del hardware* se muestran más detalles sobre estos elementos.

3.1.1 Placa Raspberry Pi

La **Raspberry Pi** es una serie de ordenadores de placa reducida, desarrollado en el Reino Unido por la Raspberry Pi Foundation [11]. En este caso se ha utilizado el último producto de la gama Raspberry Pi 3, la **Raspberry Pi 3 Model B +** concretamente (ver *Figura 5*).

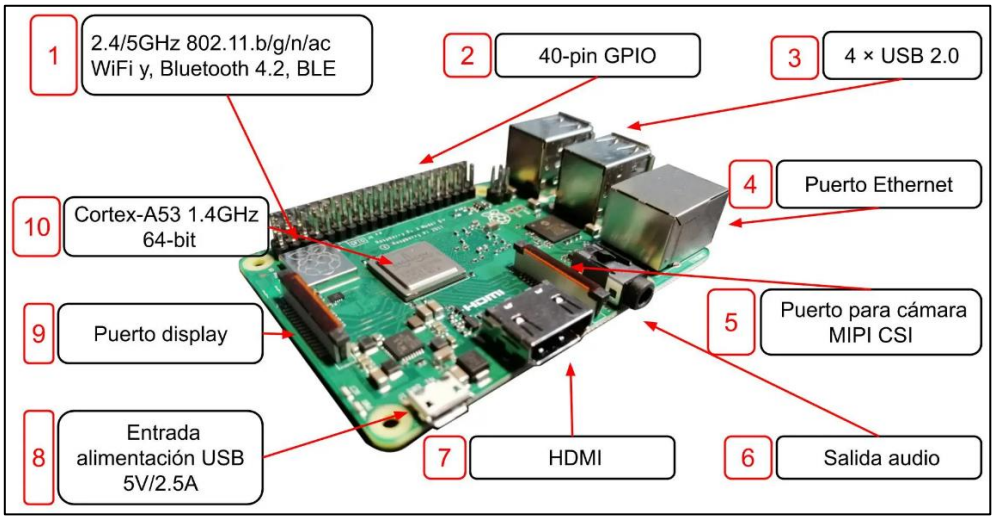


Figura 5: *Raspberry Pi 3 Model B + [20]*

El software es de código abierto, siendo su sistema operativo oficial una versión adaptada de Debian, denominada Raspberry Pi OS (Raspbian). Dexter Industries utiliza un software personalizado para sus robots, llamado **Raspbian for Robots** (ver *Figura 51* del apartado *9.3 Conexión con el robot GoPiGo3*). Raspbian for Robots tiene todo el software que necesita integrado para conectar al robot GoPiGo3. Además, todos los programas de codificación que se necesitan, como Scratch y Python, ya están integrados, listos con programas de muestra para ejecutar, probar y comenzar a aprender.

La imagen Raspbian for Robots (imagen de Raspbian adaptada para el robot GoPiGo3) se puede descargar desde la propia página web de Dexter Industries [21].

3.1.2 Robot GoPiGo3 BalanceBot

GoPiGo3 es una plataforma robótica móvil para Raspberry Pi desarrollada por **Dexter Industries** [10]. Este robot está pensado para ser utilizado como un kit educativo para el aprendizaje tanto de robótica como de programación, puesto que ofrece una amplia gama de sensores y actuadores para ensamblar en él.

En este proyecto se hace uso del kit de extensión **BalanceBot** (ver *Figura 6*). Este kit convierte el robot GoPiGo3 en un robot de equilibrio vertical. Entre otras cosas, los elementos principales del kit básico GoPiGo3 y BalanceBot son los siguientes:

- Placa de circuito Dexter Industries GoPiGo3.
- Sensor IMU (Inertial Measurement Unit o Unidad de Medición Inercial).
- Receptor de infrarrojos y mando a distancia por infrarrojos.
- Motor y codificador.

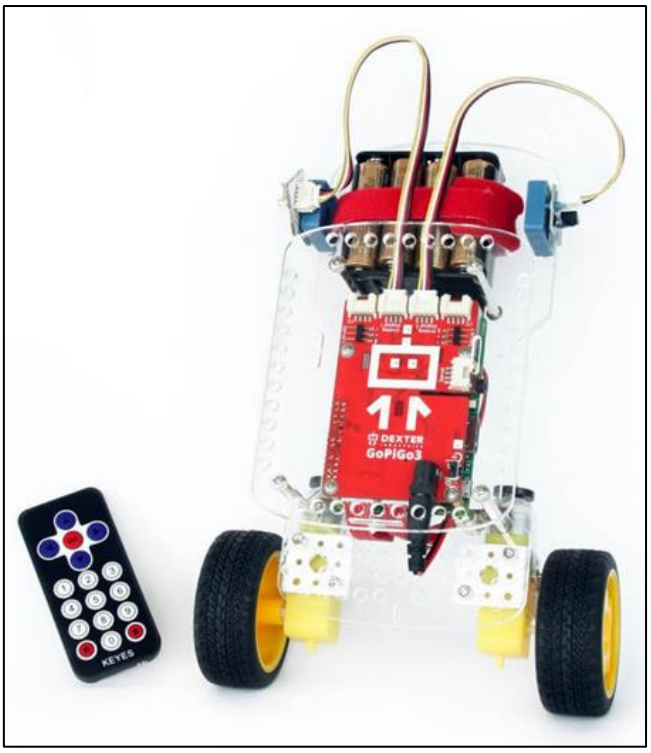


Figura 6: *Robot GoPigo3 BalanceBot [22]*

3.1.2.1 Placa Dexter Industries GoPiGo3

La **placa de circuito Dexter Industries GoPiGo3** [23] se apila sobre la Raspberry Pi sin necesidad de ninguna otra conexión. La comunicación entre los dos se produce a través de la interfaz I2C. A continuación, se muestran las especificaciones de la placa GoPiGo3 (ver *Figura 7* y *Figura 8*):

- **Conexiones de sensores**: GoPiGo3 tiene 5 puertos Grove en los que se pueden conectar diferentes sensores.
  - **Puertos I2C**: en este caso no se utilizan estos dos puertos.
  - **Puertos de sensor analógico/digital**: en estos dos puertos se pueden conectar dispositivos analógicos y digitales. En el primer puerto se conecta un sensor IMU (mirar apartado *3.1.2.2 Sensor IMU*), y en el segundo puerto un receptor de infrarrojos (mirar apartado *3.1.2.3 Receptor de infrarrojos*).
  - **Puertos seriales**: este puerto no se utiliza.
- **Conexiones de servo**: para el control de servos. En este caso no se utiliza.
- **Conexiones de motores**: tiene dos conexiones para los motores, uno para el motor izquierdo y el otro para el derecho (mirar apartado *3.1.2.4 Motor y codificador*).
- **Microcontrolador:** todo el control de los motores y las E/S de los sensores en el GoPiGo3 se realiza mediante un microcontrolador. El microcontrolador actúa como intérprete entre Raspberry Pi y GoPiGo3. Envía, recibe y ejecuta comandos enviados por Raspberry Pi.
- **Puerto de alimentación**.

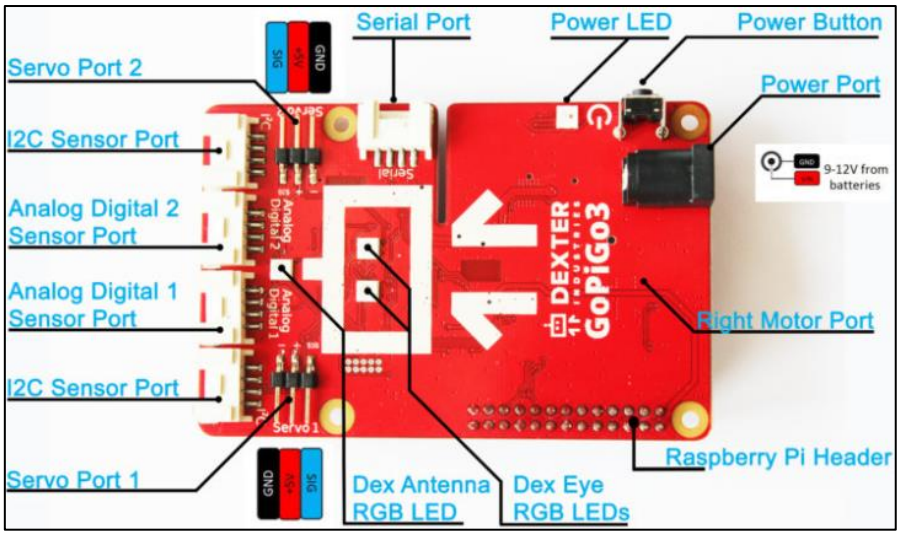


Figura 7: *GoPiGo3 lado superior [23]*

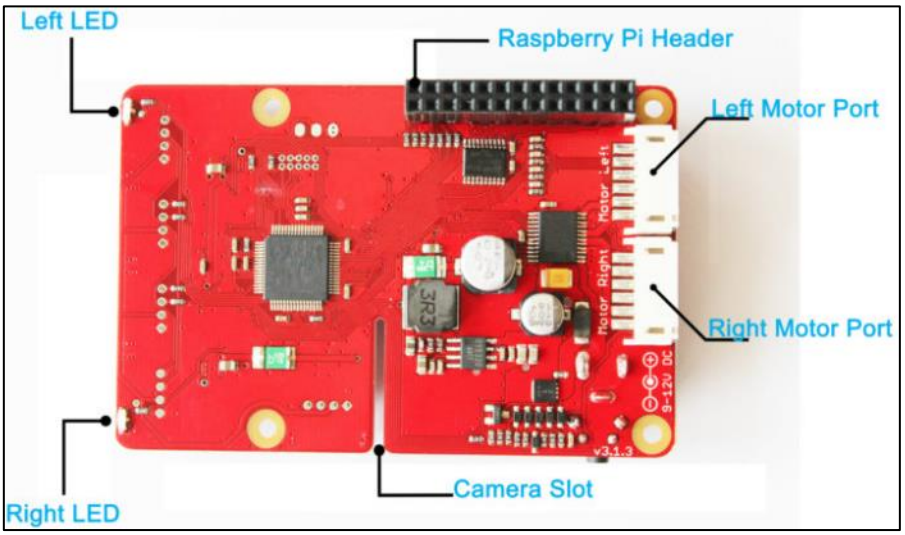


Figura 8: *GoPiGo3 lado inferior [23]*

3.1.2.2 Sensor IMU

El **sensor IMU** [24] se utiliza para detectar el movimiento, la orientación y la posición del robot (ver *Figura 9*). Para ello, el sensor tiene una brújula, un acelerómetro y un giroscopio integrados.

La conexión a GoPiGo3 se realiza mediante un cable Grove. En este caso, se ha conectado al primer puerto del sensor analógico/digital (puerto “AD1” o bus "GPG3_AD1").

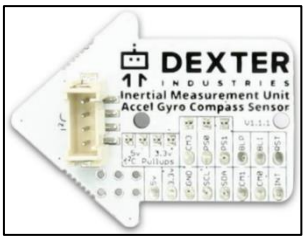


Figura 9: *Sensor IMU [24]*

3.1.2.3 Receptor de infrarrojos

El **IR receiver** (Infrared Receiver o Receptor de Infrarrojos) [25] recibe las señales emitidas por un transmisor de infrarrojos compatible a una distancia de hasta 10 metros (ver *Figura 10*).

La conexión de este dispositivo a la placa GoPiGo3 también se realiza mediante un cable Grove. Se ha conectado al segundo puerto del sensor analógico/digital (puerto "AD2" o bus "GPG3_AD2").

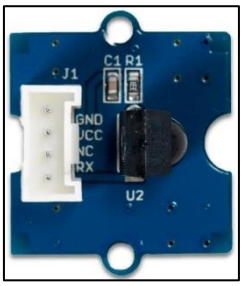


Figura 10: *Receptor de Infrarrojos (IR receiver) [25]*

El receptor de infrarrojos requiere un **IR Remote Control** (Infrared Remote Control o Control Remoto de Infrarrojos) [26]. Se ha utilizado el mando a distancia por infrarrojos de la *Figura 11*. Mediante este transmisor se podrá controlar el robot GoPiGo3. Tal y como se ha mencionado anteriormente puede enviar señales bien dentro de los 10 metros.

Figura 11: *Control Remoto de Infrarrojos (IR Remote Control) [26]*

3.1.2.4 Motor y codificador

Los **motores** de la GoPiGo3 tienen **codificadores** integrados (ver *Figura 12*), lo que le da al robot un control preciso y exacto del motor. De este modo se mejora la precisión de conducción y dirección del GoPiGo3, por lo que no hay virajes inesperados. Estos dos motores se acoplan a las ruedas del robot GoPiGo3.

Figura 12: *Motor y codificador*

3.2 API del robot GoPiGo3

La **API** es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas de programación.

El robot GoPiGo3 viene pre-instalado con una API de programación [13][14][15], mediante la cual se pueden enviar órdenes a los actuadores o leer datos del robot. Esta API se puede utilizar con diferentes lenguajes de programación, como Python y Scratch. En este caso, se ha optado por Python [12], puesto que es un lenguaje de programación bastante fácil y sencillo de aprender y permite una comunicación más inmediata con la implementación Python del servidor de comunicaciones RIP.

Dexter Industries ha desarrollado una biblioteca o API llamada EasyGoPiGo3 para que sea más fácil de programar el robot GoPiGo3 en Python [13][14]. Aunque esta biblioteca simplifique la programación de GoPiGo3, en este trabajo se ha decidido no utilizarlo, y se utiliza la **clase GoPiGo3** [15]. Esta clase GoPiGo3 dispone de varios métodos o funciones. Los métodos utilizados en este proyecto se muestran en *Tabla 1* y *Tabla 2*.

Nombre función o método	Descripción
gopigo3.GoPiGo3().set_grove_type(port, type)	Establecer tipo de Grove: - port → Los puertos de Grove. Grove1 y/o Grove2. - type → El tipo de dispositivo Grove, consultar GroveSensorType. Ejemplo: - gopigo3.GoPiGo3().set_grove_type(gopigo3.GoPiGo3().GROVE_2, gopigo3.GoPiGo3().GROVE_TYPE.IR_DI_REMOTE) → Configurar el dispositivo como un receptor IR y al puerto AD2.
gopigo3.GoPiGo3().set_motor_dps(port, dps)	Ajustar la velocidad del motor: - port → El puerto del motor a utilizar. Puede ser motor izquierdo o motor derecho. - dps → Velocidad del robot en DPS (grados por segundo de la rueda del robot). Se define entre 0-1000 DPS. La velocidad predeterminada se establece en 300 DPS. Ejemplo: - gopigo3.GoPiGo3().set_motor_dps(gopigo3.GoPiGo3().MOTOR_LEFT, 400) → Velocidad del motor izquierdo 400 DPS. - gopigo3.GoPiGo3().set_motor_dps(gopigo3.GoPiGo3().MOTOR_RIGHT, 700) → Velocidad del motor derecho 700 DPS.

gopigo3.GoPiGo3().set_motor_power(port, power)	Establecer la potencia del motor en porcentaje (%): - port → El puerto del motor a utilizar. Puede ser motor izquierdo o motor derecho. - power → La potencia del motor en porcentaje (%). Se define entre -100% a 100%, o -128% para flotación. Ejemplo: - gopigo3.GoPiGo3().set_motor_power(gopigo3.GoPiGo3().MOTOR_LEFT, -40) → Establecer potencia del motor izquierdo en -40%. - gopigo3.GoPiGo3().set_motor_power(gopigo3.GoPiGo3().MOTOR_RIGHT, 80) → Establecer potencia del motor derecho en 80%.
gopigo3.GoPiGo3().offset_motor_encoder(port, positionOffset)	Compensación de un codificador de motor: - port → El puerto del motor a utilizar. Puede ser motor izquierdo o motor derecho. - positionOffset → El desplazamiento del codificador en grados (°). Ponga a cero el codificador desplazándolo por la posición actual. Ejemplo: - gopigo3.GoPiGo3().offset_motor_encoder(gopigo3.GoPiGo3().MOTOR_LEFT, gopigo3.GoPiGo3().get_motor_encoder (gopigo3.GoPiGo3().MOTOR_LEFT)) → Reset encoder izquierdo (posición = 0°). - gopigo3.GoPiGo3().offset_motor_encoder(gopigo3.GoPiGo3().MOTOR_RIGHT, 20) → Desplazamiento de 20° para el codificador derecho.

Tabla 1: *Clase GoPiGo3 métodos Set*

Nombre función o método	Descripción
gopigo3.GoPiGo3().get_manufacturer()	Devuelve el nombre del fabricante de GoPiGo3.
gopigo3.GoPiGo3().get_board()	Leer los 20 caracteres del nombre de la placa GoPiGo3.
gopigo3.GoPiGo3().get_id()	Leer el número de serie del hardware GoPiGo3 de 128 bits.
gopigo3.GoPiGo3().get_version_firmware()	Leer la versión de firmware.
gopigo3.GoPiGo3().get_version_hardware()	Leer la versión de hardware.
gopigo3.GoPiGo3().get_voltage_battery()	Obtener el voltaje de la batería en voltios (V).
gopigo3.GoPiGo3().get_grove_value(port)	Obtener valor de un puerto Grove: - port → El puerto de Grove, Grove1 o Grove2. Ejemplo: - gopigo3.GoPiGo3().get_grove_value(gopigo3.GoPiGo3().GROVE_1) → Obtener valor del puerto Grove 1. - gopigo3.GoPiGo3().get_grove_value(gopigo3.GoPiGo3().GROVE_2) → Obtener valor del puerto Grove 2.
gopigo3.GoPiGo3().get_motor_encoder(port)	Obtener la posición de cada motor (codificador). Se define entre 0-360°: - port → El puerto del motor a utilizar. Puede ser motor izquierdo o motor derecho. Ejemplo: - gopigo3.GoPiGo3().get_motor_encoder(gopigo3.GoPiGo3().MOTOR_LEFT) → Obtener la posición del motor izquierdo. - gopigo3.GoPiGo3().get_motor_encoder(gopigo3.GoPiGo3().MOTOR_RIGHT) → Obtener la posición del motor derecho.

Tabla 2: *Clase GoPiGo3 métodos Get*

Mediante los métodos Set se enviarán órdenes a los actuadores del robot, por ejemplo, establecer la potencia del motor. En cambio, los métodos Get se utilizan para leer u obtener datos del robot.

Por otro lado, para la lectura del sensor IMU se hace uso de la clase InertialMeasurementUnit (ver *Tabla 3*).

Nombre función o método	Descripción
di_sensors.inertial_measurement_unit.InertialMeasurementUnit(bus).read_gyroscope()	Leer la velocidad angular del giroscopio. Devuelve la velocidad angular como una tupla de valores X, Y, Z en grados/seg (°/s): - bus → El bus al que está conectado el sensor IMU. Ejemplo: - di_sensors.inertial_measurement_unit.InertialMeasurementUnit(bus = "GPG3_AD1").read_gyroscope() → Leer la velocidad angular del giroscopio.

Tabla 3: *Clase InertialMeasurementUnit*

3.3 Servidor RIP

El **servidor RIP** es un servidor de comunicaciones [16]. Su objetivo es ofrecer una solución de comunicación simple pero poderosa que se pueda utilizar desde los clientes web. RIP solo utiliza protocolos estándar HTTP [27], compatibles con los principales navegadores web.

El protocolo es de código abierto, bajo la Licencia Pública General GNU. Es un protocolo de comunicación para ser utilizado en el modelo cliente-servidor, especialmente diseñado para laboratorios online (Online Laboratories o OL) en los que el cliente se ejecuta dentro de un navegador web. RIP proporciona un mecanismo simple para que los usuarios y las máquinas/programas del cliente adquieran información sobre las experiencias del laboratorio definidas en el servidor y sobre las entradas y salidas de cada experiencia (se define una experiencia como cada una de las actividades de laboratorio que se pueden realizar con una implementación OL). El protocolo también define métodos para leer y escribir los valores de estas entradas y salidas, respectivamente.

RIP se basa en dos pilares: los métodos POST y GET para el transporte de comunicaciones y JSON-RPC [29] para formatear mensajes. POST y GET son métodos HTTP que, a su vez, se basan en comunicaciones TCP. Por otro lado, JSON-RPC se basa en el formato JSON.

La comunicación entre dos entidades de software es posible cuando hablan el mismo protocolo. Por lo tanto, la implementación RIP es necesaria tanto en el cliente como en el servidor.

3.3.1 Implementación RIP. Perspectiva del servidor

En la arquitectura de un servidor RIP que implementa el protocolo RIP hay tres subsistemas funcionales:

1. El **Servidor Web** (Web Server): maneja las conexiones del cliente, sesiones de usuario… Admite tres tipos de solicitudes:
    - **Metadatos** (Meta-data): un cliente, que desea obtener información sobre el laboratorio, lanza una solicitud HTTP GET a la URL asociada al laboratorio. El servidor RIP responde con una estructura JSON-RPC que informa al cliente.
    - **Observador** (Observer): un cliente que desea recibir actualizaciones sobre el estado de la planta se suscribe a un evento de SSE [28] asociada a la experiencia de interés. Se utiliza para obtener actualizaciones de datos de servidor a cliente.
    - **Operador** (Operator): un cliente, que desea actuar sobre el OL o recibir una actualización de datos a pedido, envía una solicitud POST con el comando codificado como una estructura RIP-JSON-RPC.
2. **Intérprete de comandos** (Command Interpreter): es el que habla RIP.
3. **Ejecutor** (Executor): controla la ejecución de los programas de control del laboratorio.

3.3.2 Implementación RIP. Perspectiva del cliente

Un cliente RIP simplemente debe implementar los métodos de protocolo de comunicación requeridos (POST, GET y SSE) con el formato apropiado para leer y escribir el contenido de los mensajes (JSON-RPC) y la estructura y funciones definidas por RIP.

3.3.3 Funciones del protocolo RIP

Las funciones definidas y utilizadas en el protocolo se dividen en dos tipos: internas y externas.

3.3.3.1 Funciones internas

Las funciones internas son funciones privadas, utilizadas internamente por las implementaciones del servidor RIP para comunicarse con el programa de control utilizado en el OL. A continuación, se muestra la lista de funciones internas que debe implementar un servidor RIP:

1. **info = basicInfo ()**: devuelve la lista de experiencias definidas en el OL.
2. **info = experienceInfo (expId)**: devuelve la lista de variables definidas en el programa de control asociado a la experiencia definida por el parámetro de entrada **expId**.
3. **readablelist, writablelist = open(expId)**: devuelve la lista de variables que se pueden leer y escribir (en dos diferentes variables) definidas en la experiencia asociada al parámetro de entrada **expId**.
4. **close (expId)**: cierra el programa de control de la experiencia OL definida por el parámetro de entrada **expId**.
5. **start (expId)**: abre e inicia la ejecución del programa de control asociado a la experiencia OL definida por el parámetro de entrada **expId**.
6. **run (expId)**: ejecuta el programa de control asociado a la experiencia OL definida por el parámetro de entrada **expId**.
7. **stop (expId)**: detiene la ejecución del programa de control asociado a la experiencia OL definida por el parámetro de entrada **expId**.
8. **readVariableNames, readVariableValues = get(expId, variableNames)**: recupera los valores actuales de las variables (**readVariableValues**) especificadas por el parámetro de entrada **variableNames**. Para que esto suceda, estas variables deben existir en el programa de control. Cuando se llama al método get(), **readVariableNames** contiene solo el nombre de las variables que se leyeron correctamente, no todas las solicitadas en **variableNames**.
9. **set (expId, variableNames, variableValues)**: escribe los valores recibidos (**variableValues**) en las variables especificadas (**variableNames**) de la experiencia OL. Para que esto suceda, estas variables deben existir en el programa de control.

3.3.3.2 Funciones externas

Las funciones externas son métodos del lado del cliente. Los clientes RIP utilizan estas funciones para obtener datos sobre las experiencias definidas y para escribir y leer datos hacia y desde el OL. A continuación, se muestra la lista de funciones externas que debe implementar un cliente RIP:

1. **result = info (callback, expId = null)**: recupera la información de metadatos sobre la experiencia definida por el parámetro de entrada **expId**. Este parámetro de entrada es opcional y si no se especifica, el método entonces devuelve información de metadatos de todas las experiencias definidas.

2. **connect (expId, callback)**: se conecta a una experiencia definida por el parámetro de entrada **expId** y establece conexión con el SSE para comenzar a recibir actualizaciones de datos del servidor.
3. **set (variableNames, variableValues, callback, expId)**: escribe los valores recibidos (**variableValues**) en las variables especificadas (**variableNames**) de la experiencia OL. Para que esto suceda, estas variables deben existir en el programa de control.
4. **result = get (variableNames, callback, expId)**: recupera los valores actuales de las variables (**result**) especificadas por el parámetro de entrada **variableNames**. Para que esto suceda, estas variables deben existir en el programa de control.

3.4 Herramienta EjsS

EjsS [17], también conocido como EJS o Ejs, es una herramienta de creación gratuita escrita en Java que ayuda a los no programadores a crear simulaciones interactivas en Java o Javascript, principalmente con fines de enseñanza o aprendizaje. Este software ha sido creado por Francisco Esquembre, con la colaboración de Félix Jesús García Clemente.

Esta herramienta está diseñada para la creación de simulaciones por computadora discretas. Una simulación por computadora discreta es un programa de computadora que intenta reproducir, con fines pedagógicos o científicos, un fenómeno natural a través de la visualización de los diferentes estados que puede tener. Cada uno de estos estados se describe mediante un conjunto de variables que cambian en el tiempo debido a la iteración de un algoritmo dado. EjsS está diseñado para las personas que estén más interesadas en el contenido de la simulación, el fenómeno simulado en sí, y mucho menos en los aspectos técnicos necesarios para construir la simulación.

En particular, EjsS crea aplicaciones Java y JavaScript que son independientes de la plataforma, o simulaciones que pueden visualizarse usando cualquier navegador web (y por lo tanto distribuidas a través de Internet), que leen datos a través de la red y que pueden controlarse usando scripts desde las páginas web.

EjsS utiliza el concepto de espacio de trabajo para organizar su trabajo. Un espacio de trabajo es una carpeta o directorio donde EjsS almacena sus archivos de simulación para un proyecto determinado. Dentro de una carpeta o directorio de espacio de trabajo, se encuentran cuatro subcarpetas (ver *Figura 13*):

- **config**: esta carpeta contiene archivos de configuración creados por EjsS cuando cambia sus opciones de tiempo de ejecución.
- **export**: esta es la carpeta predeterminada que EjsS le ofrece al usuario cuando crea paquetes autoejecutables (archivos JAR) o archivos HTML para su implementación.
- **output**: esta es una carpeta que EjsS usa para almacenar archivos intermedios creados cuando ejecuta una simulación.
- **source**: esta es la carpeta donde el usuario debe guardar sus archivos de simulación (XML), junto con los archivos auxiliares, como imágenes GIF o archivos de datos.

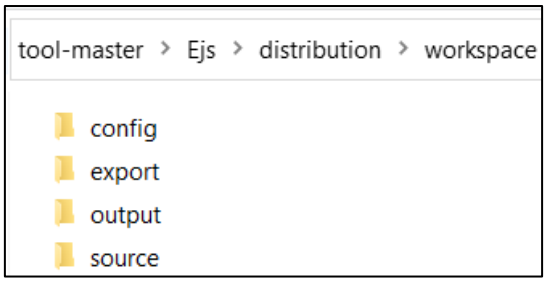


Figura 13: *Estructura del espacio de trabajo de EjsS*

3.4.1 Interfaz de usuario de EjsS

La *Figura 14* muestra el interfaz de usuario de EjsS.

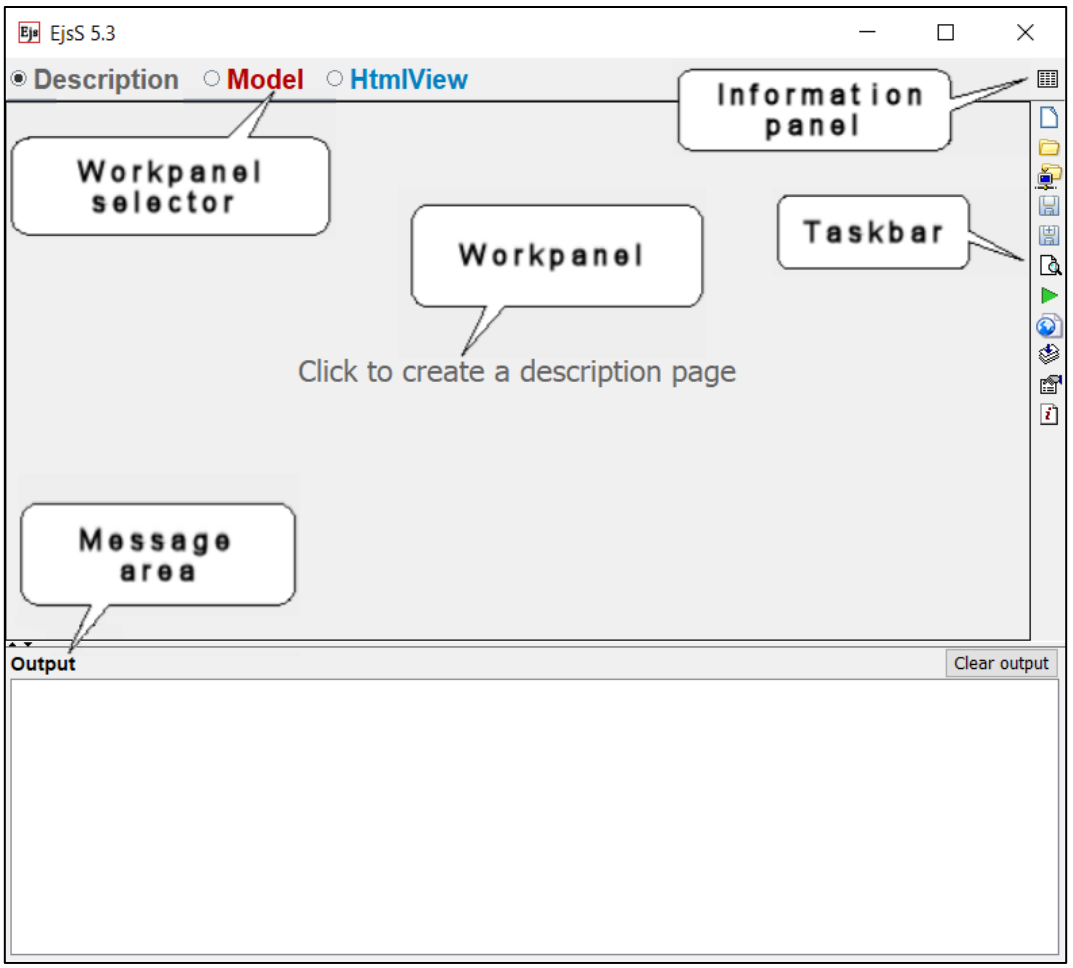


Figura 14: *Interfaz de usuario EjsS [17]*

- **Information panel**: este icono sirve para mostrar un panel de información para la simulación.
- **Taskbar**: la barra de tareas de la derecha proporciona una serie de iconos para crear, abrir y guardar un archivo, buscar el código, ejecutar una simulación, traducir la interfaz de simulación, empaquetar una o más simulaciones en un archivo JAR o exportarlas como subprogramas en páginas web, configurar EjsS y mostrar la ayuda de EjsS.
- **Message area**: la parte inferior de la interfaz contiene un área de salida donde EjsS muestra mensajes informativos.

- **Workpanel**: la parte central de la interfaz contiene los paneles de trabajo donde se construye la simulación.
- **Workpanel selector**: EjsS proporciona tres paneles de trabajo para crear la simulación:
    - **Description**: permite crear y editar una narrativa introductoria basada en HTML que describe la simulación. Esta información se mostrará al usuario al inicio cuando se ejecute la simulación como una aplicación. Para completar la descripción se puede utilizar el editor HTML simple que está integrado en EjsS o el usuario puede importar archivos HTML creados con otras herramientas.
    - **Model**: está dedicado al proceso de modelado. Dentro de este panel se encuentran seis subpaneles:
        - **Variables**: se utiliza para declarar e inicializar las variables que describen el modelo de la simulación
        - **Initialization**: permite escribir algoritmos de inicialización.
        - **Evolution**: permite escribir algoritmos que describen cómo el modelo cambia en el tiempo.
        - **Fixed relations**: establecen relaciones fijas entre las variables y los cálculos adicionales necesarios para mantener actualizado el modelo.
        - **Custom**: permiten al usuario crear métodos (funciones y subrutinas) para usar en otras partes del *Model* y *HtmlView*. Estos métodos se suman a una lista estándar de métodos predefinidos de EjsS.
        - **Elements**: ofrece objetos de modelo predefinidos y listos para usar que simplifican algunas tareas avanzadas.
    - **HtmlView**: está dedicado a la tarea de construir la interfaz gráfica de usuario de la simulación, que permite a los usuarios finales controlar la simulación y mostrar su salida. Se construye la interfaz seleccionando elementos de paletas y agregándolos al árbol de elementos de la vista.

4. Desarrollo del proyecto

Una interfaz de laboratorio remoto desarrollada con EjsS puede comunicarse fácilmente con el hardware/software del laboratorio mediante RIP. Para esto, se necesitan dos cosas: el elemento RIP de EjsS (mirar apartado *4.2 Implementación de EjsS en el proyecto*) y una implementación de servidor RIP (mirar apartado *4.1 Implementación del servidor RIP en el proyecto*).

En este apartado, por una parte, se explicará cómo se ha utilizado en el proyecto el servidor RIP implementado en Python. Por otra parte, se mostrará la interfaz web desarrollada mediante EjsS.

En los apartados *9.2 Ensamblaje del robot GoPiGo3*, *9.3 Conexión con el robot GoPiGo3* y *9.4 Instalar paquetes y librerías para el servidor RIP* se muestra información útil para el desarrollo de este proyecto.

4.1 Implementación del servidor RIP en el proyecto

Como se ha mencionado en el apartado *3.3.3 Funciones del protocolo RIP*, los servidores RIP deben implementar las funciones internas, mientras que los clientes RIP deben implementar las funciones externas. Los métodos basados en las funciones externas de un cliente RIP se comunican con el servidor web de un servidor RIP, el intérprete de comandos lo traduce en una serie de uno o más funciones internas, y finalmente son ejecutadas por el componente ejecutor (ver *Figura 15*).

En este caso, el servidor RIP se ejecuta en la placa Raspberry Pi, y es el encargado de realizar la comunicación entre la aplicación informática desarrollada en EjsS y el robot GoPiGo3. La aplicación informática (cliente RIP) envía comandos al servidor RIP, el intérprete de comandos traduce estos comandos, y utilizando el API del robot GoPiGo3 envía órdenes a los actuadores y sensores del robot. La placa Dexter Industries GoPiGo3 sirve de interfaz entre la placa Raspberry Pi y el robot GoPiGo3.

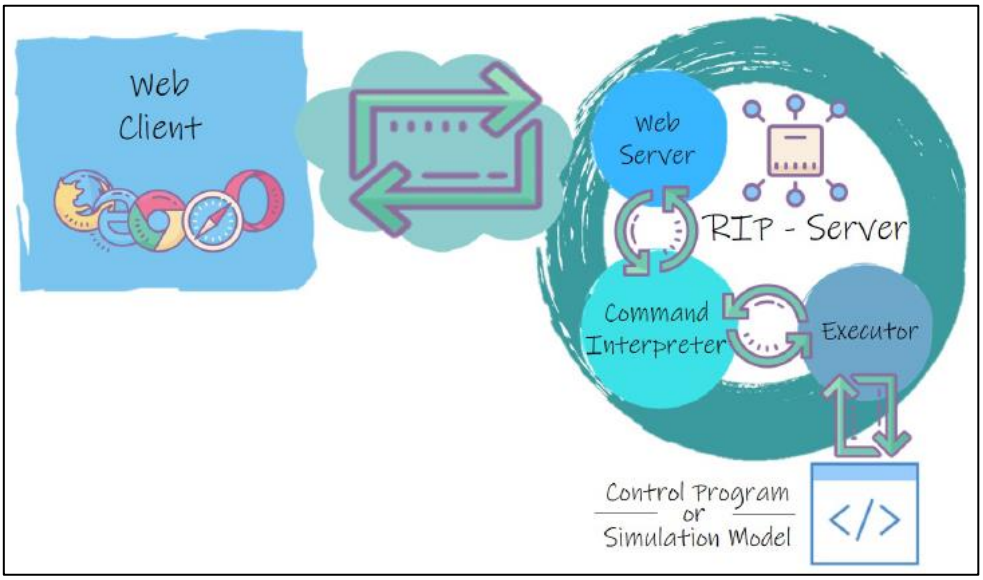


Figura 15: *Arquitectura del servidor RIP [16]*

La API del servidor RIP ha sido proporcionada por el equipo docente. Dicho software está disponible en [30]. Aun así, se han realizado modificaciones para lograr el intercambio de datos entre la aplicación informática y el proyecto desarrollado en Python para el control del robot. A continuación, se muestran los ficheros definidos para el desarrollo del proyecto (el código desarrollado de cada uno de estos ficheros se encuentra en el apartado *10 Anexo B*):

- **App.py**: arranca el servidor HTTP [27].
- **AppConfig.py**: configuración de hardware del servidor HTTP.
- **RIPGoPiGo3.py**: implementa el servidor JSON-RPC [29]. Interpreta y trata las tramas recibidas por el servidor HTTP.
- **GoPiGo3Operation.py**: controla la lógica de ejecución del robot y lee y envía datos al robot.
- **GoPiGo3Communication.py**: permite el acceso a las variables del robot sin la necesidad de tener conocimientos sobre la API GoPiGo3 [15].

4.1.1 App.py

El fichero **App.py** se encarga de arrancar el servidor HTTP [27]. Recibe las solicitudes del cliente (aplicación informática desarrollada en EjsS) y manda las respuestas al cliente.

Este fichero es el mismo que ha proporcionado el equipo docente, no se ha realizado ningún cambio en él.

4.1.2 AppConfig.py

El servidor RIP se divide aproximadamente en dos niveles: el primero implementa la funcionalidad RIP y el segundo implementa el acceso a hardware [31]. Como el nivel de comunicación generalmente no está sujeto a cambios, para configurar el servidor solo se necesita vincular la aplicación a una implementación de hardware específica en el archivo de configuración **AppConfig.py** y, opcionalmente, proporcionar información adicional sobre la experiencia alojada en el servidor. Para la realización de este proyecto se ha definido una nueva configuración o interfaz de hardware, llamado **RIPGoPiGo3** (mirar apartado *4.1.3 RIPGoPiGo3.py*).

Tal y como se observa en el *Código 1*, el nombre del módulo que implementa el acceso al hardware (RIPGoPiGo3) se especifica en el campo *config.control.impl_module*. Por otra parte, *info* contiene la definición de la experiencia, incluyendo el nombre de la experiencia, una breve descripción, autores y palabras clave y la lista de los elementos que se pueden leer y escribir.

```
# Autor: Mikel Iturrioz
# Última modificación: 04/08/2021

# This file contains the configuration of the RIP server application.
config = {
  # TO DO: The server will listen to host:port
  'server': {
    'host': '127.0.0.1',
    'port': 8080,
  },
  # The 'control' section configures the mapping between the RIP protocol
  # and the actual implementation of the functionality.
  # The 'impl' field should contain the name of the module (.py) and the
  # class that implement the control interface
  'control': {
     'impl_module': 'RIPGoPiGo3',  # Contiene el nombre del módulo que implementa el
acceso al hardware. Se ha definido un nuevo módulo de control
     # Also, if the class name is not the same as the module name:
     #'impl_name': 'RIPOctave',
     'info': { # Contiene la definición de la experiencia
       'name': 'RIP GiPiGo3',
       'description': 'An implementation of RIP to control a GoPiGo3 session',
       'authors': 'M. Iturrioz',
       'keywords': 'Raspberry Pi GopiGo3, RIP',
       # Server readable objects
       'readables': [],
       # Server writable objects
       'writables': []
     },
  }
}
```

Código 1: *AppConfig.py*

4.1.3 RIPGoPiGo3.py

Como se ha mencionado en el apartado anterior, **RIPGoPiGo3** es el nombre del módulo que implementa el acceso al hardware. El nuevo código implementado **RIPGoPiGo3.py** crea una subclase que amplia RIPGeneric. Es decir, se ha creado una clase llamada RIPGoPiGo3 que extiende RIPGeneric. En esta nueva clase se han definido los métodos set(), get() y getValuesToNotify() [31]. Estos métodos definen como leer y escribir objetos del servidor.

Entre los parámetros de entrada del método **set** se encuentran **variables** y **values**, es decir, **set([variables],[values])**. Cuando se llama a este método, los valores especificados en **values** se asignan a las variables del programa de control denominados como cadenas en **variables**. En este caso, mediante el método set se podrán escribir las variables de la *Tabla 4*.

Variable	Tipo de dato	Descripción
setModoControl	String	Selecciona el modo de funcionamiento del robot GoPiGo3. Existen las siguientes opciones: - Manual → Robot GoPiGo3 en modo de funcionamiento Manual. - Automático → Robot GoPiGo3 en modo de funcionamiento Automático.
setRobotVeloManual	Int	Establece la velocidad manual del robot GoPiGo3 en DPS (grados por segundo de la rueda del robot).
setMovimientoManual	Int	Ejecuta movimiento manual del robot GoPiGo3. Existen las siguientes opciones: - 1 → Mover el robot GoPiGo3 hacia adelante. - 2 → Mover el robot GoPiGo3 hacia atrás. - 3 → Mover el robot GoPiGo3 hacia el lado izquierdo. - 4 → Mover el robot GoPiGo3 hacia el lado derecho. - 5 → Girar el robot GoPiGo3 hacia el lado izquierdo. - 6 → Girar el robot GoPiGo3 hacia el lado derecho. - 7 → Parar robot GoPiGo3.
setResetRobotEncoders		Resetea la posición de los encoders del motor izquierdo y derecho del robot GoPiGo3 (posición = 0°).
setPotenciaMotorIzq	Int	Define la potencia del motor izquierdo del robot GoPiGo3 en porcentaje (%).
setPotenciaMotorDer	Int	Define la potencia del motor derecho del robot GoPiGo3 en porcentaje (%).

setControlRemotoIR	Bool	Activa/Desactiva el Control Remoto IR (para el programa Balance Bot). Existen las siguientes opciones: - 0 → Control Remoto IR desactivado. No utilizar mando a distancia por infrarrojos. Utilizar las flechas de la interfaz web. - 1 → Control Remoto IR activado. Utilizar mando a distancia por infrarrojos.
setMovControlWebIR	Int	Guarda el botón seleccionado mediante las flechas de la interfaz Web (para el programa Balance Bot). Existen las siguientes opciones: - 1 → Botón o flecha presionada: adelante. - 2 → Botón o flecha presionada: izquierda. - 3 → Botón o flecha presionada: OK. - 4 → Botón o flecha presionada: derecha. - 5 → Botón o flecha presionada: atrás.
setStartBalanceBot		Inicia la ejecución del programa Balance Bot.
setStopBalanceBot		Para la ejecución del programa Balance Bot.
setKGYROANGLE	Double	Define constante KGYROANGLE (para el programa Balance Bot).
setKGYROSPEED	Double	Define constante KGYROSPEED (para el programa Balance Bot).
setKPOS	Double	Define constante KPOS (para el programa Balance Bot).
setKSPEED	Double	Define constante KSPEED (para el programa Balance Bot).
setKDRIVE	Double	Define constante KDRIVE (para el programa Balance Bot).
setKSTEER	Double	Define constante KSTEER (para el programa Balance Bot).
setKVOLTAGE	Double	Define constante KVOLTAGE (para el programa Balance Bot).
setDRIVESPEED	Double	Define constante DRIVE SPEED (para el programa Balance Bot).
setSTEERSPEED	Double	Define constante STEER SPEED (para el programa Balance Bot).
setTIMEFALLLIMIT	Double	Define constante TIME FALL LIMIT (para el programa Balance Bot).
setStartScript	String	Carga e inicia el programa de control (script) desarrollado por el usuario.
setStopScript		Para la ejecución del programa de control (script) desarrollado por el usuario.

Tabla 4: *Método set. Variables del servidor RIP*

En el caso del método **get** se tiene como parámetro de entrada **variables**, es decir, **get([variables])**. Cuando se llama a este método, se leen y devuelven los valores de las variables del programa de control denominadas como cadenas en **variables**. El resultado devuelto por este método es una matriz mixta con los valores actuales de las variables del programa de control especificadas en el parámetro de entrada **variables**. Utilizando el método get se podrá leer el valor de las variables de la *Tabla 5*.

Variable	Tipo de dato	Descripción
connect	String	Establece conexión con el servidor RIP. Devuelve los siguientes valores (valores para la aplicación informática EjsS): - Conectado = True → Variable que indica conexión establecida correctamente. - Fabricante de GoPiGo3. - Referencia placa GoPiGo3. - Número de serie del hardware GoPiGo3. - Versión firmware. - Versión hardware.
disconnect	String	Finaliza conexión con el servidor RIP. Devuelve los siguientes valores (valores para la aplicación informática EjsS): - Conectado = False → Variable que indica conexión cerrada correctamente. - Fabricante de GoPiGo3 = "" → Inicializar valor. - Referencia placa GoPiGo3 = "" → Inicializar valor. - Número de serie del hardware GoPiGo3 = "" → Inicializar valor. - Versión firmware = "" → Inicializar valor. - Versión hardware = "" → Inicializar valor.

Tabla 5: *Método get. Variables del servidor RIP*

Por último, el método **getValuesToNotify** devuelve las variables que serán reportadas periódicamente al usuario (usando el canal SSE [28]). El método debe devolver una lista donde el primer elemento es una lista que contiene el nombre de las variables notificadas y el segundo elemento es otra lista con los valores correspondientes. A continuación, en la *Tabla 6*, se muestran las variables que serán reportadas periódicamente al usuario o cliente (en este caso a la aplicación informática desarrollada en EjsS).

Variable	Tipo de dato	Descripción
InfoVisorAvisos	String	Mensajes o logs generados en el servidor RIP del robot GoPiGo3.
BateriaRobot	Int	Voltaje batería del robot GoPiGo3 en voltios (V).
SensorIMUX	Int	Valor del sensor giroscópico IMU. Velocidad angular del eje X en grados/seg (°/s).
SensorIMUY	Int	Valor del sensor giroscópico IMU. Velocidad angular del eje Y en grados/seg (°/s).
SensorIMUZ	Int	Valor del sensor giroscópico IMU. Velocidad angular del eje Z en grados/seg (°/s).
BotonIRSeleccionado	String	Flecha seleccionada en el Control Remoto (para el programa Balance Bot).
EncMotorIzq	Int	Posición del encoder izquierdo en grados (°).
VueltasEncMotorIzq	Int	Número de vueltas del encoder izquierdo.
GradosEncMotorIzq	Int	Grados de la última vuelta del encoder izquierdo (°).
EncMotorDer	Int	Posición del encoder derecho en grados (°).
VueltasEncMotorDer	Int	Número de vueltas del encoder derecho.
GradosEncMotorDer	Int	Grados de la última vuelta del encoder derecho (°).
DiametroRuedaRobot	Int	Diámetro de la rueda del robot GoPiGo3 en milímetros (mm).
VelocidadActualManual	Int	Velocidad actual del robot GoPiGo3 para el modo de funcionamiento Manual en DPS (grados por segundo de la rueda del robot).

Tabla 6: *Método getValuesToNotify. Variables del servidor RIP*

4.1.4 GoPiGo3Operation.py

GoPiGo3Operation.py es el que controla la lógica de ejecución del robot. Utilizando los métodos mencionados en el apartado *3.2 API del robot GoPiGo3*, la placa Raspberry Pi enviará comandos a la placa Dexter Industries GoPiGo3. La placa Dexter Industries GoPiGo3 procesará dichos comandos, enviando órdenes a los actuadores o leyendo datos del robot.

El funcionamiento del servidor RIP es el siguiente. El servidor RIP instalado en el robot GoPiGo3 espera a una solicitud de conexión por parte del cliente RIP (interfaz web EjsS). Cuando llega una petición de conexión se establece conexión con el robot. Por una parte, se establece conexión con el SSE [28] para comenzar a recibir actualizaciones de datos del servidor, y, por otra parte, se inicia el ciclo de trabajo del robot desarrollado en GoPiGo3Operation.py. Una vez establecida la conexión, se podrán ejecutar los métodos de protocolo de comunicación (set y get) implementados en el cliente RIP.

En el ciclo de trabajo del robot existen dos modos de funcionamiento:

- **Modo manual**: mediante este modo de funcionamiento el usuario podrá realizar el control manual del robot. En la interfaz web EjsS se ha creado un panel de control manual donde el operario puede realizar todos los movimientos manuales del robot (robot GoPiGo3 hacia adelante, atrás, izquierda, derecha…). Se explicará en detalle en el apartado *4.2.2.2.1 Panel PanelPrincipalControlRobotGoPiGo3*.
- **Modo automático**: en el modo de funcionamiento automático existe la opción de ejecutar diferentes programas de control:
    - **Programa Balance Bot**: se ha creado un programa para realizar el equilibro vertical del robot GoPiGo3 BalanceBot [32]. El usuario tendrá acceso para cambiar el valor de varias constantes que influyen en el programa de equilibrio del robot (ver apartado *4.1.4.1 Programa BalanceBot*).
    - **Programa de control (script) del usuario**: además del programa BalanceBot, en la interfaz web se ha añadido un editor de código donde el usuario podrá crear su propio programa de control del robot, y, después, descargar y ejecutar en el robot GoPiGo3 (ver apartado *4.1.4.2 Programa de control del usuario*).

Cabe destacar que, en ambos modos de funcionamiento, tanto manual como automático, mediante el canal SSE se actualizarán periódicamente varias variables. Estas variables se han mencionado en la *Tabla 6* del apartado *4.1.3 RIPGoPiGo3.py*.

En la *Figura 16* se muestra el esquema de funcionamiento del servidor RIP implementado en el robot GoPiGo3.

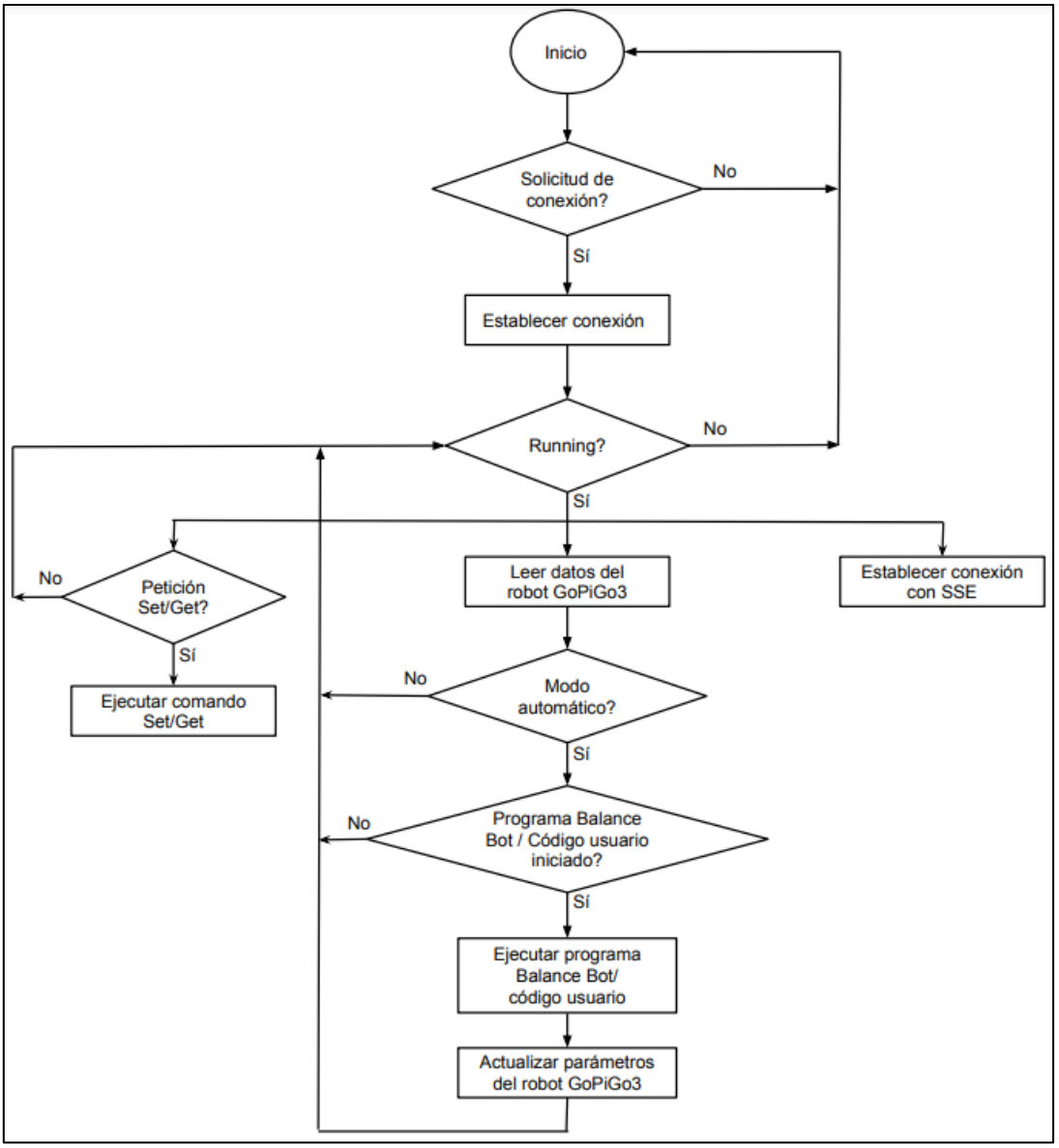


Figura 16: *Funcionamiento servidor RIP del robot GoPiGo3*

Se observa en la *Figura 16* que, una vez establecida la conexión entre el servidor RIP (robot GoPiGo3) y el cliente RIP (interfaz web EjsS), se establece conexión con el canal SSE y se ejecutan los comandos set y get implementados en la interfaz web. Además, se inicia el ciclo de trabajo del robot. En este ciclo, por un lado, se lee el valor de los parámetros del robot como, por ejemplo, el voltaje de la batería, el valor del sensor IMU, la posición de los encoders… Por otro lado, si el robot trabaja en modo de funcionamiento automático y está iniciado un programa de control (programa Balance Bot o código desarrollado por el usuario), se ejecuta dicho programa y se actualizan los parámetros del robot.

4.1.4.1 Programa BalanceBot

El robot GoPiGo3 BalanceBot se basa en un modelo de péndulo invertido [1]. Muchos investigadores e ingenieros están trabajando en el péndulo invertido y su aplicación para realizar un robot autoequilibrado debido a su naturaleza inestable.

El algoritmo de control [32] que se utiliza para mantener la posición de equilibrio del robot GoPiGo3 es el controlador **PID** [33]. El controlador proporcional, integral y derivativo (PID) es conocido como controlador de tres términos. Es un mecanismo de retroalimentación de bucle de control que se utiliza ampliamente en la industria. El controlador intenta ajustar y corregir el error entre el proceso medido y el proceso deseado y generar medidas correctivas para ajustar el proceso en consecuencia. Este controlador debe ejecutarse con suficiente frecuencia y al mismo tiempo dentro del rango controlable del sistema. En *la Figura 17 se* muestra un sistema en lazo cerrado, en el cual se utiliza un controlador PID para alcanzar el estado de salida deseado.

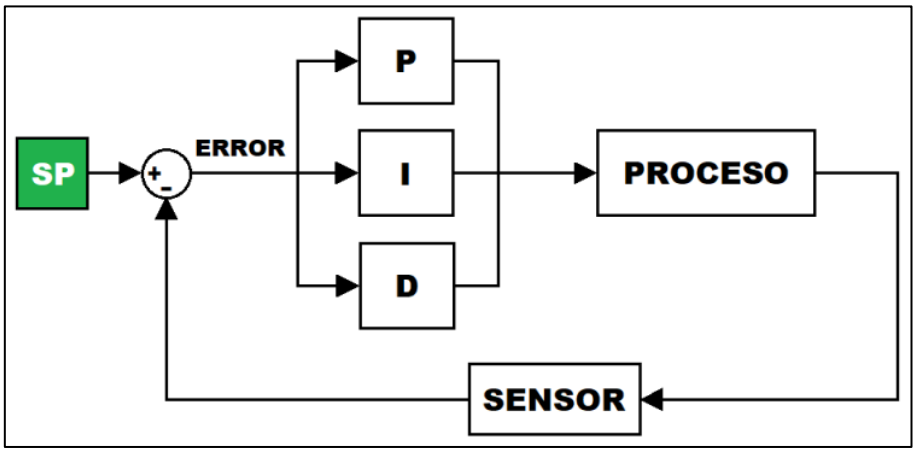


Figura 17: *Control PID [33]*

La *Figura 18* muestra el punto de ajuste y el ángulo de inclinación real del robot de dos ruedas. El error es la diferencia entre el ángulo de inclinación real y el ángulo de inclinación deseado (punto de ajuste). Como sugiere su nombre, el controlador PID contiene tres partes, que son el término proporcional, el término integral y el término derivado. Estos términos tienen un efecto diferente en la respuesta del motor. Para equilibrar el robot, el punto de ajuste del robot debe ser 0°.

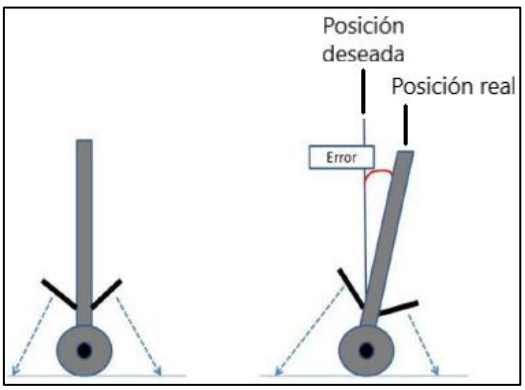


Figura 18: *Punto de ajuste y ángulo de inclinación real del robot [34]*

Los componentes principales en el circuito del robot equilibrador son la unidad de medición inercial (IMU), el controlador Raspeberry Pi, la placa Dexter Industries GoPiGo3 y los motores (ver *Figura 19*). La placa GoPiGo3 hace de interfaz entre la placa Raspberry Pi y los actuadores y sensores del robot GoPiGo3. El sensor IMU cumple el papel del elemento *SENSOR* de la *Figura 17*. Este sensor entre otras cosas se puede utilizar para medir la aceleración y la velocidad angular del robot. En este caso se hace uso del giroscopio integrado en la IMU para medir la velocidad angular a lo largo de sus tres ejes (X, Y, Z). Aquí solo es necesario el valor en el eje Z, ya que da el valor de inclinación en el eje considerado. Al comparar el ángulo de inclinación real medido por el sensor IMU con el punto de ajuste deseado, se obtiene el error, es decir, se obtiene la diferencia entre el punto de ajuste deseado y el ángulo real medido. A continuación, este error se alimenta al algoritmo del controlador PID. El controlador PID procesa, calcula y genera la salida de velocidad correspondiente para controlar los motores, con el fin de lograr el equilibrio del robot de la manera correcta. Una vez conseguido equilibrar el robot, utilizando los botones *Arriba*, *Abajo*, *Izquierda* y *Derecha* del mando a distancia por infrarrojos o las flechas habilitadas en la interfaz web se podrá conducir el robot. Es por ello por lo que se ha añadido un receptor de infrarrojos en el robot. En el apartado *4.2.2.2.1 Panel PanelPrincipalControlRobotGoPiGo3* se habla en detalle sobre la gestión del control remoto.

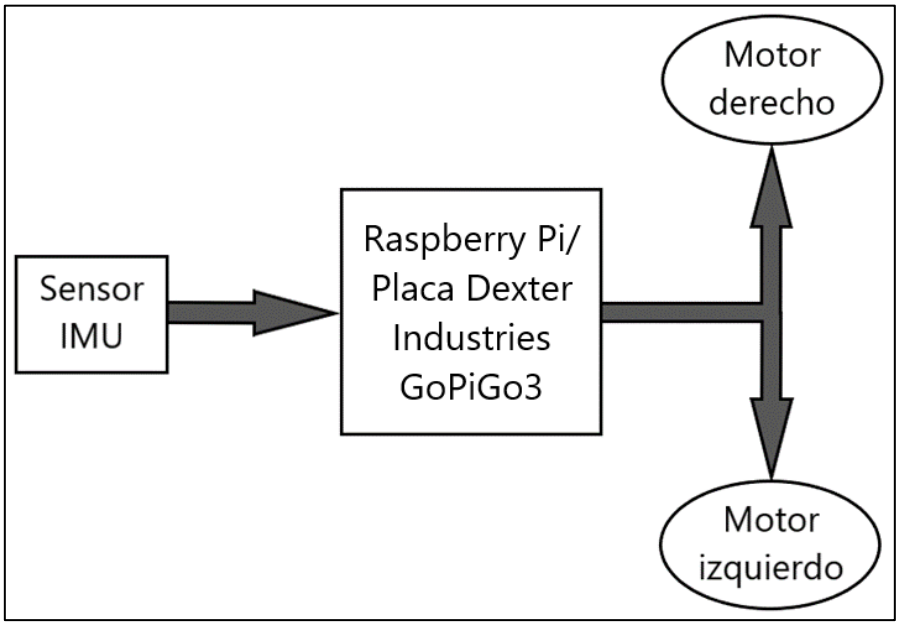


Figura 19: *Diagrama de bloques general del sistema electrónico del robot GoPiGo3*

Cuando se aplica el controlador PID, el robot de dos ruedas permanece su posición vertical solo si se eligen los valores apropiados de ganancia de Kp (término proporcional), Ki (término integral) y Kd (término derivado). Estas constantes funcionan para un entorno determinado: el tamaño de las ruedas, el peso de la batería, la superficie sobre la que está trabajando el robot GoPiGo3 BalanceBot… Para un correcto comportamiento de equilibrio del robot se han creado varías constantes [36] en el programa BalanceBot [32]. El robot de equilibrio se verá afectado de manera más significativa por las siguientes constantes PID:

- **KGYROANGLE**: cuánto responde el robot al ángulo integrado del giroscopio. Aumentar esto hará que el robot responda de manera más agresiva al ángulo del robot. Aumentar demasiado hará que el robot oscile y caiga. Disminuir demasiado hará que el robot no responda lo suficiente y caiga.

- **KGYROSPEED**: cuánto responde el robot a los cambios de ángulo (velocidad de rotación, es decir, caída). Aumentar esto hará que el robot responda de manera más agresiva a los cambios en el ángulo del robot. Aumentar demasiado hará que el robot oscile y caiga. Disminuir demasiado hará que el robot no responda lo suficiente y caiga.

Además de KGYROANGLE y KGYROSPEED, hay que tener en cuenta también las siguientes constantes:

- **KPOS**: cuánto intenta el robot permanecer en su lugar (para que no se desvíe). También la agresividad con la que intentará conducir hasta la posición especificada (al conducir con control remoto). Aumentar esto hará que el robot intente permanecer mejor en su lugar, pero si está demasiado alto, el robot no podrá mantenerse erguido, ya que se concentrará en permanecer en su lugar. Disminuir esto permitirá que el robot deambule un poco y puede hacer que el equilibrio sea más estable (ya que no estará tan enfocado en permanecer en su lugar).
- **KSPEED**: cuánto afecta la velocidad actual del motor a la potencia del motor.
- **KDRIVE**: evita que el robot se ponga en marcha o se detenga de forma agresiva. Se utiliza para conducir el robot con control remoto.
- **KSTEER**: no se usa para el algoritmo de equilibrio, pero se usa para decirle a los motores cuánto deben responder a una diferencia entre la velocidad izquierda y derecha del motor (para girar). Aumentar esto hará que los giros respondan mejor, pero podría desequilibrar al robot si se configura demasiado alto. Disminuir demasiado hará que el giro sea menos receptivo.
- **KVOLTAGE**: escala la potencia del motor de acuerdo con el voltaje de la batería para que automáticamente se reajuste para las baterías agotadas. No conviene cambiar esta constante.
- **DRIVE_SPEED**: rapidez con la que el robot conducirá con el control remoto. Establecer este valor demasiado alto hará que el robot sea inestable, ya que se concentrará demasiado en la conducción y no lo suficiente en el equilibrio.
- **STEER_SPEED**: rapidez con la que el robot dirigirá con el control remoto. Establecer este valor demasiado alto hará que el robot sea inestable, ya que se concentrará demasiado en dirigir y no lo suficiente en el equilibrio.
- **TIME_FALL_LIMIT**: establece el tiempo de espera después del cual el robot asume que se cayó y debe reiniciarse.

Si es necesario cambiar las constantes porque el robot no se está equilibrando o hay problemas, conviene empezar a ajustar las constantes KGYROANGLE y KGYROSPEED.

4.1.4.2 Programa de control del usuario

En la interfaz web desarrollada mediante EjsS se ha añadido un editor de código (mirar apartado *4.2.2.2.1 Panel PanelPrincipalControlRobotGoPiGo3*). De este modo, el usuario desde la interfaz web tiene la opción de crear su propio código Python, y después, descargar y ejecutar dicho código en el robot.

En el programa que desarrolla el usuario en el editor de la interfaz web, puede utilizar directamente la API de programación mencionada en el apartado *3.2 API del robot GoPiGo3*. De

este modo podrá enviar órdenes a los actuadores y leer datos de los sensores, realizando el control del robot GoPiGo3. Aun así, para facilitar el trabajo de programación en el editor de la interfaz web se ha creado una clase llamada **GoPiGo3Communication** (ver apartado *4.1.5 GoPiGo3Communication.py*). Utilizando esta clase el usuario podrá leer o consultar datos de los sensores (IMU, receptor de infrarrojos…) y enviar órdenes a los actuadores del robot (motores), sin la necesidad de saber cómo funciona la API del robot GoPiGo3.

4.1.5 GoPiGo3Communication.py

La clase **GoPiGo3Communication** se utiliza para intercambiar datos entre el robot y el programa Python desarrollado por el usuario en la aplicación informática. Esta clase facilita las labores de programación del usuario, puesto que el usuario podrá acceder a ciertos datos del robot sin la necesidad de tener conocimientos sobre la API GoPiGo3 mencionada en el apartado *3.2 API del robot GoPiGo3*.

En la *Tabla 7* se muestran las variables de la clase GoPiGo3Communication.

Variable	Tipo de dato	Descripción
BATERIA	Int	Indica el voltaje de la batería del robot GoPiGo3 en voltios (V).
IMUSENSOR_X	Int	Indica el valor del sensor giroscópico IMU. Velocidad angular del eje X en grados/seg (°/s).
IMUSENSOR_Y	Int	Indica el valor del sensor giroscópico IMU. Velocidad angular del eje Y en grados/seg (°/s).
IMUSENSOR_Z	Int	Indica el valor del sensor giroscópico IMU. Velocidad angular del eje Z en grados/seg (°/s).
BOTON_IR	Int	Indica el número de la flecha seleccionada en el Control Remoto: - 1 → Botón o flecha seleccionada: adelante. - 2 → Botón o flecha seleccionada: izquierda. - 3 → Botón o flecha seleccionada: OK. - 4 → Botón o flecha seleccionada: derecha. - 5 → Botón o flecha seleccionada: atrás.
ENCMOTORIZQ	Int	Indica la posición del encoder izquierdo en grados(°).
VUELTASENCMOTORIZQ	Int	Indica el número de vueltas del encoder izquierdo.
GRADOSENCMOTORIZQ	Int	Indica los grados de la última vuelta del encoder izquierdo (°).
ENCMOTORDER	Int	Indica la posición del encoder derecho en grados (°).
VUELTASENCMOTORDER	Int	Indica el número de vueltas del encoder derecho.
GRADOSENCMOTORDER	Int	Indica los grados de la última vuelta del encoder derecho (°).
DIAMETRORUEDA	Int	Indica el diámetro de la rueda del robot GoPiGo3 en milímetros (mm).
POTENCIAMOTORIZQ	Int	Define la potencia del motor izquierdo del robot GoPiGo3 en porcentaje (%).
POTENCIAMOTORDER	Int	Define la potencia del motor derecho del robot GoPiGo3 en porcentaje (%).
RESETENCODERS	Bool	Resetea la posición de los encoders del motor izquierdo y derecho del robot GoPiGo3 (posición = 0°).
msg	String	Mensaje generado en el programa de control desarrollado por el usuario para visualizar en el Visor de Avisos de la interfaz web.
Ciclo_PrimerCiclo	Bool	Indica el estado del programa de control desarrollado por el usuario: - Ciclo_PrimerCiclo = True → Primer ciclo del programa de control desarrollado por el usuario ejecutándose. - Ciclo_PrimerCiclo = False → Programa de control desarrollado por el usuario no está en el primer ciclo de ejecución.
VariableAuxiliar		Sirve para guardar cualquier variable interna del programa de control del usuario para utilizar en cada ciclo de ejecución del programa.

Tabla 7: *Variables de la clase GoPiGo3Communication*

En la clase GoPiGo3Operation para iniciar el programa de control desarrollado por el usuario se ejecuta un subproceso. Este subproceso contiene el código desarrollado por el usuario. Para interactuar con dicho subproceso se ha creado la clase GoPiGo3Communication, la cual se manda como parámetro de entrada cuando se ejecuta la llamada al subproceso. En el código desarrollado por el usuario se recupera el objeto que permite acceder a la clase GoPiGo3Communication, denominado *ComunicacionGoPiGo3*. De este modo, dentro del

subproceso se puede acceder a las variables de la clase GoPiGo3Communication, para leer o consultar su valor o para escribir un nuevo valor a las variables (en el apartado *5.2 Prueba 2. Modos de funcionamiento del ciclo de trabajo del robot* se explica detalladamente este proceso). A continuación, se muestran ejemplos de cómo leer el valor de las variables de lectura de la clase GoPiGo3Communication en el editor de código del usuario:

- **ComunicacionGoPiGo3['BATERIA']** → Obtener el voltaje de la batería del robot GoPiGo3 en voltios (V).
- **ComunicacionGoPiGo3['ENCMOTORIZQ']** → Obtener la posición actual del encoder del motor izquierdo en grados (°).
- **ComunicacionGoPiGo3['DIAMETRORUEDA']** → Obtener diámetro de la rueda del robot en milímetros (mm).
- **ComunicacionGoPiGo3['VariableAuxiliar']** → Obtener el valor de cualquier variable interna del programa de control del usuario para utilizar en cada ciclo de ejecución del programa.

Al final del subproceso o código del usuario hay que volver a salvar los datos y devolver a la clase GoPiGo3Operation, de modo que las variables modificadas tendrán efecto en el ciclo de trabajo del robot. Para modificar el valor de las variables de escritura de la clase GoPiGo3Communication, por ejemplo, se debería de implementar el siguiente comando o instrucción en el programa de control desarrollado por el usuario:

- **ComunicacionGoPiGo3['POTENCIAMOTORIZQ']** = PotenciaMotorIzq → Salvar la potencia con la que tiene que empezar a girar el motor izquierdo. *PotenciaMotorIzq* será un valor en porcentaje (%) que se ha calculado en el programa de control desarrollado por el usuario.
- **ComunicacionGoPiGo3['RESETENCODERS']** = True → Resetear la posición (posición = 0°) de los encoders del motor izquierdo y derecho.

A modo de ejemplo, en el apartado *10.6 Código usuario ejemplo* se muestra un programa de control que podría desarrollar el usuario. En este código se sigue el proceso definido anteriormente (en el apartado *5.2 Prueba 2. Modos de funcionamiento del ciclo de trabajo del robot* se explica detalladamente este proceso). Primero, se recupera el objeto (ComunicacionGoPiGo3) que permite acceder a la clase GoPiGo3Communication, y, después, se desarrolla el código. En este caso, dependiendo del botón (adelante, izquierda, derecha o atrás) seleccionado en el mando a distancia por infrarrojos (Control Remoto IR) se definirá una potencia u otra para los motores del robot. Al final del código se salva el valor de la potencia que se ha definido para cada motor y se devuelve dicho valor a la clase GoPiGo3Operation. Esta clase se encargará de enviar órdenes al robot para que los motores empiezan a girar a la potencia definida.

4.2 Implementación de EjsS en el proyecto

Para el desarrollo de la interfaz web se ha utilizado el software EjsS [37]. En este apartado se explicará la aplicación informática desarrollada para el proyecto mediante EjsS.

4.2.1 Añadir, configurar y utilizar el elemento RIP

Las aplicaciones web desarrolladas en EjsS pueden ser simulaciones o interfaces de laboratorio remoto. Cuando la aplicación es una interfaz de laboratorio remoto y la comunicación con el hardware/software del laboratorio se puede implementar fácilmente mediante RIP, se utiliza el elemento RIP.

En este apartado se describe cómo agregar, configurar y usar el elemento RIP [38] para permitir que la aplicación informática basada en EjsS se comunique con el servidor RIP (y su software de control) que se está ejecutando en la placa Raspberry Pi del robot GoPiGo3.

Para agregar el elemento RIP a la aplicación de EjsS hay que seguir los siguientes pasos:

1. Elegir la pestaña *Model → Elements*. El elemento RIP se encuentra dentro de la categoría *SoftwareLinks* (ver *Figura 20*).

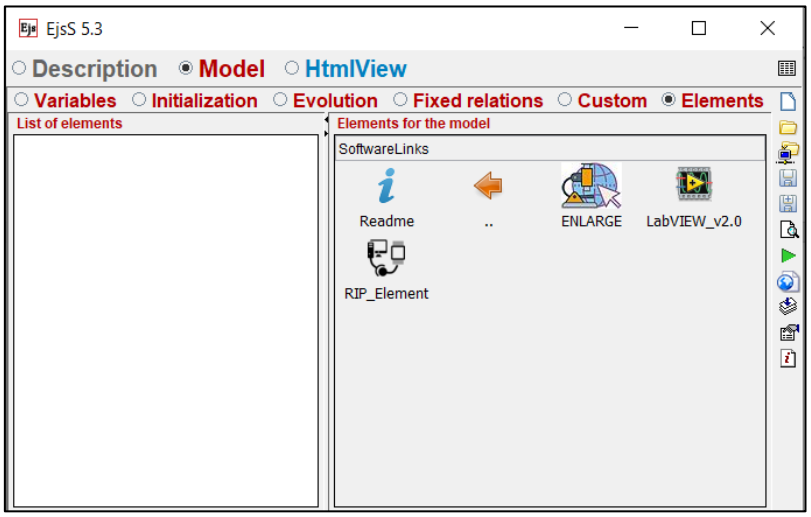


Figura 20: *Elementos de SoftwareLinks en el editor EjsS*

2. Agregar el elemento RIP a la aplicación. Para ello, arrastrar y soltar el elemento RIP de derecha a izquierda y asignar un nombre cuando se solicite (se observa en la *Figura 21* que en este caso se denomina *RIP*). Se hará uso del nombre asignado para intercambiar datos con el servidor RIP que se está ejecutando en la placa Raspberry Pi (mediante los métodos connect, set y get).

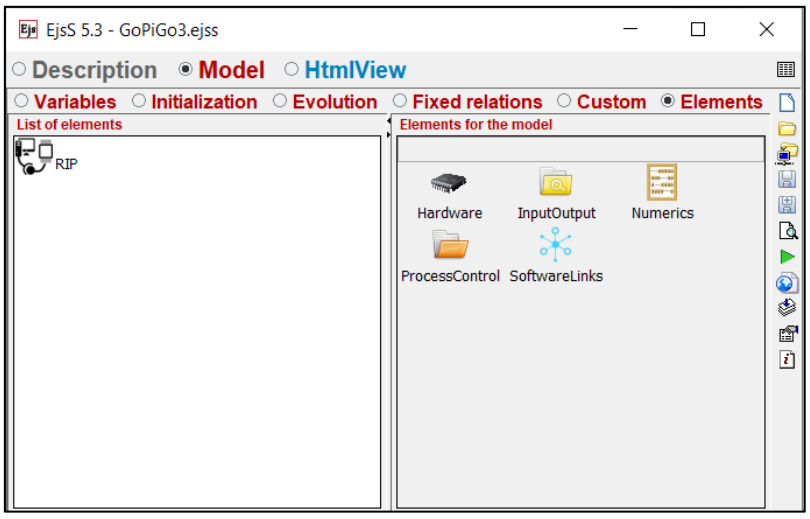


Figura 21: *Agregar el elemento RIP a la aplicación EjsS*

3. Una vez que se agrega el elemento a la aplicación, se debe configurarlo antes de comenzar a usarlo. Para ello, al hacer doble clic en el elemento agregado se abrirá una ventana de configuración. La ventana de configuración presenta tres pestañas: *Server Configuration*, *Experience* y *Auto Update*.
    - **Server Configuration**:
        - **Server url**: URL donde se aloja el servidor RIP. Se define la dirección IP del servidor RIP.
        - **Get Experiences**: botón para obtener las experiencias definidas en el servidor RIP y los métodos de servicio web disponibles para obtener información sobre cada experiencia.
        - **Table**: aquí aparece la información de los métodos de servicio web disponibles después de presionar el botón *Get Experiences*.

      En la *Figura 22* se muestra la pestaña *Server Configuration* con la URL adecuada (la dirección IP del servidor RIP coincide con la dirección de la *Figura 54* del apartado *9.3 Conexión con el robot GoPiGo3*). Tal y como se ve en la tabla, solo hay una fila, lo que significa que solo hay un método de servicio web definido en el servidor RIP que se ejecuta en la URL especificada.

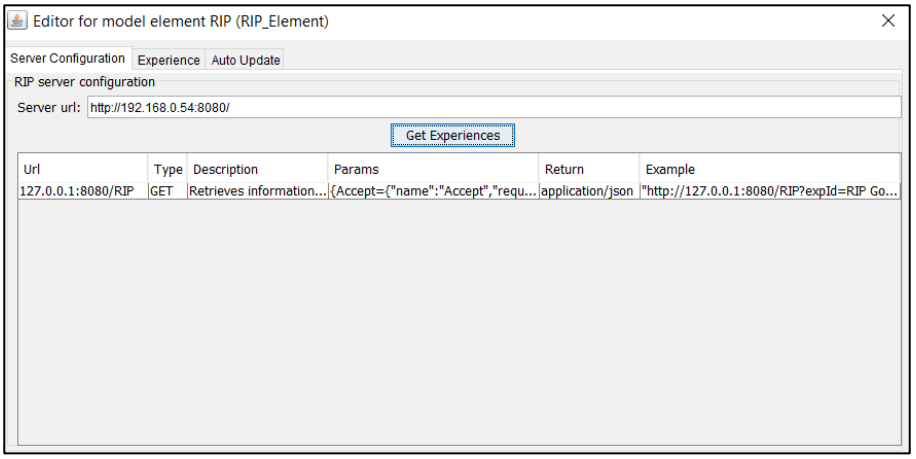


Figura 22: *Configuración elemento RIP. Pestaña Server Configuration*

    - **Experience**:
        - **Choose an Experience**: después de presionar el botón *Get Experiences*, este menú desplegable ofrece una lista con las experiencias disponibles.
        - **Lab Description**: una descripción de la experiencia seleccionada en el menú desplegable.
        - **Table**: información de los métodos de servicio web RIP disponibles para comunicarse con la experiencia seleccionada en el menú desplegable.

      La *Figura 23* muestra la pestaña *Experience* con la experiencia RIP GoPiGo3 seleccionada. La lista de experiencias disponibles se recibe del servidor RIP al que está conectado el elemento RIP. En este caso RIP habilita 3 métodos para comunicarse con la experiencia seleccionada. De arriba a abajo, estos métodos se utilizan para: escribir el valor de una variable en el servidor RIP, suscribirse para obtener un flujo de datos con los valores actualizados de una o más variables del servidor RIP (canal SSE [28]), y obtener el valor de una o más variables por solicitud del cliente.

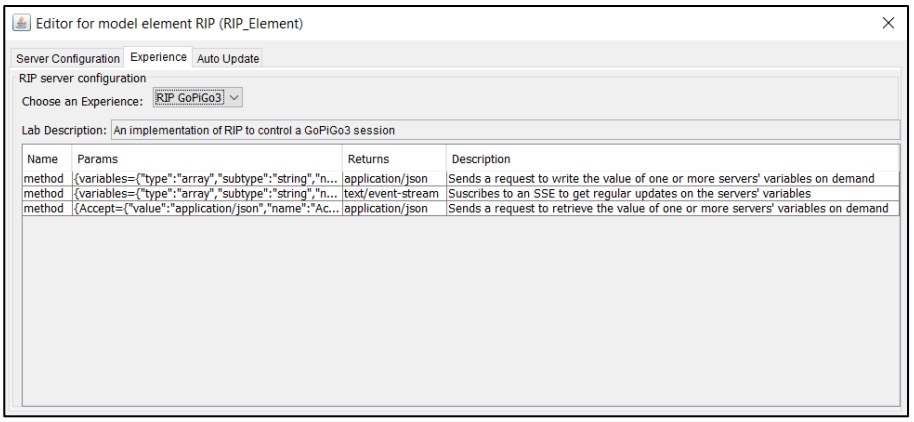


Figura 23: *Configuración elemento RIP. Pestaña Experience*

- **Auto Update**:
  - **Server Writables Variables**: cuando se ha seleccionado una experiencia en el menú desplegable (en este caso RIP GoPiGo3*)*, esta tabla muestra la lista de variables de escritura del servidor RIP.
  - **Server Readable Variables**: cuando se ha seleccionado una experiencia en el menú desplegable (en este caso RIP GoPiGo3*)*, esta tabla muestra la lista de variables de lectura del servidor RIP.
  - **Table of Linked Variables**: al hacer clic con el botón derecho en las dos tablas anteriores, los usuarios pueden vincular las variables del servidor RIP con las variables EjsS para configurar su sincronización automática. Dichos enlaces establecidos aparecen en esta tabla.

La *Figura 24* muestra la pestaña *Auto Update*, donde las variables del servidor RIP están vinculadas con las variables de EjsS. Los enlaces se establecen como get (variables de lectura) o set (variables de escritura) automáticamente, dependiendo de la naturaleza de la variable del servidor RIP. Con la configuración get, las variables EjsS recibirán automáticamente los valores de las variables del servidor RIP vinculadas (mirar la *Tabla 6* del apartado *4.1.3 RIPGoPiGo3.py*). Estas actualizaciones de valores se reciben periódicamente, usando el canal SSE [28].

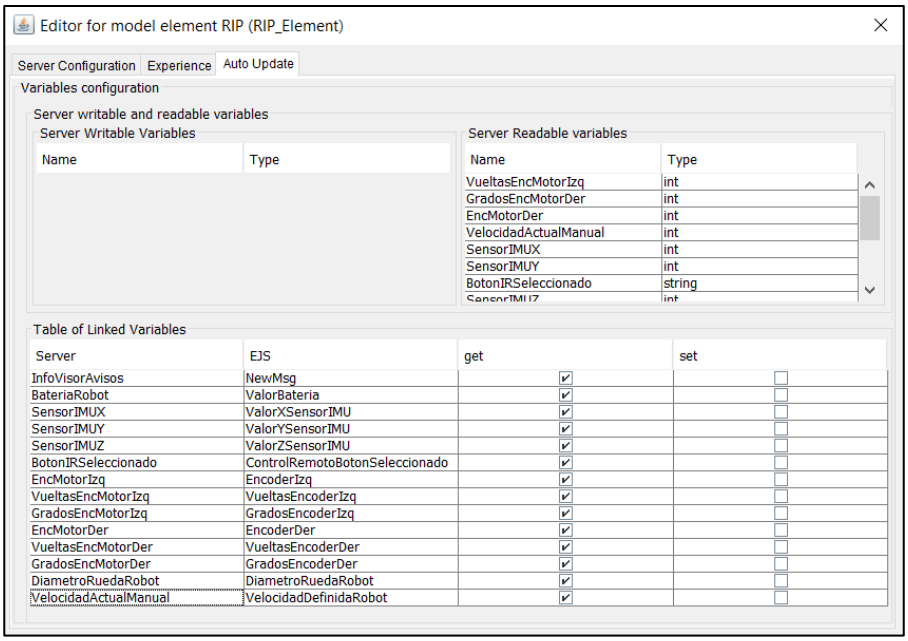


Figura 24: *Configuración elemento RIP. Pestaña Auto Update*

Para terminar, hay que mencionar que el elemento RIP proporciona tres métodos (se han mencionado anteriormente en el apartado *3.3.3.2 Funciones externas*). Se ha visto en la *Figura 21* que en este caso el elemento RIP se denomina *RIP*:

- **RIP.connect()**: se utiliza para conectarse al servidor RIP.
- **RIP.set([variableNames],[variableValues])**: se utiliza para escribir los valores (**variableValues**) en las variables especificadas (**variableNames**). Generalmente se invoca como resultado de una acción de interacción del usuario, por ejemplo, cuando el usuario presiona un botón de la aplicación informática.
- **RIP.get([variableNames])**: se utiliza para obtener los valores actuales de las variables especificadas por el parámetro **variableNames**. Este método se utiliza muy poco en la aplicación EjsS, debido a que se ha configurado el elemento RIP parar recuperar automáticamente los valores de las variables del servidor RIP (mediante el canal SSE [28]).

Los métodos set() y get() aceptan matrices como parámetros de entrada. Por lo tanto, si se desea obtener (leer) o establecer (escribir) el valor de más de una variable al mismo tiempo, hay que agregarlos a los parámetros de la matriz de entrada.

4.2.2 Aplicación informática desarrollada (EjsS)

En la aplicación informática desarrollada para el proyecto se han implementado en un principio todas las funciones necesarias para realizar el control del robot (la programación desarrollada en el software EjsS se encuentra en el apartado *11 Anexo C*). Entre otras cosas, se pueden leer datos de los diferentes sensores del robot, realizar movimientos manuales del robot, cargar y ejecutar un programa de balanceo o un código desarrollado por el usuario… Todo este control del robot se realizará remotamente, utilizando el elemento RIP añadido en la aplicación EjsS (mirar apartado *4.2.1 Añadir, configurar y utilizar el elemento RIP*).

A continuación, se muestra la interfaz gráfica EjsS desarrollada.

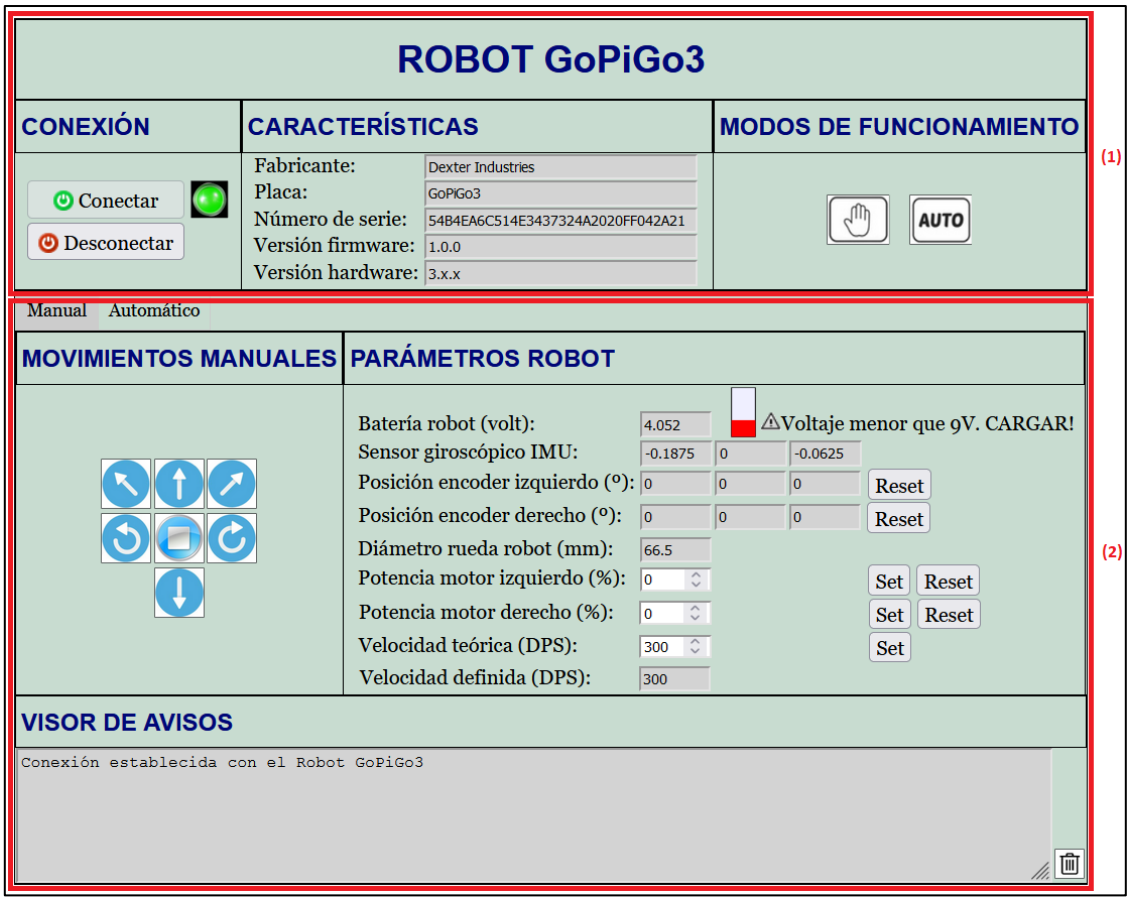


Figura 25: *Aplicación informática EjsS desarrollada. (1) PanelGeneralRobotGoPiGo3 y (2) PanelControlRobotGoPiGo3*

En la *Figura 25* se observa que la interfaz de usuario se divide en dos paneles denominados (1) *PanelGeneralRobotGoPiGo3* y (2) *PanelControlRobotGoPiGo3*.

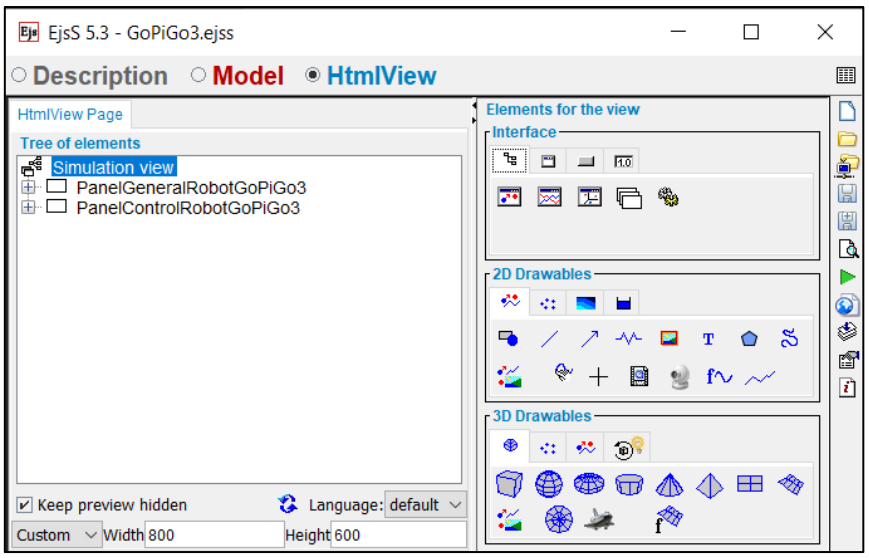


Figura 26: *Estructura general de la aplicación informática EjsS desarrollada*

4.2.2.1 Panel PanelGeneralRobotGoPiGo3

Como se ve en la *Figura 27*, este panel se divide a su vez en otros tres subpaneles: *CONEXIÓN*, *CARACTERÍSTICAS* y *MODOS DE FUNCIONAMIENTO*.

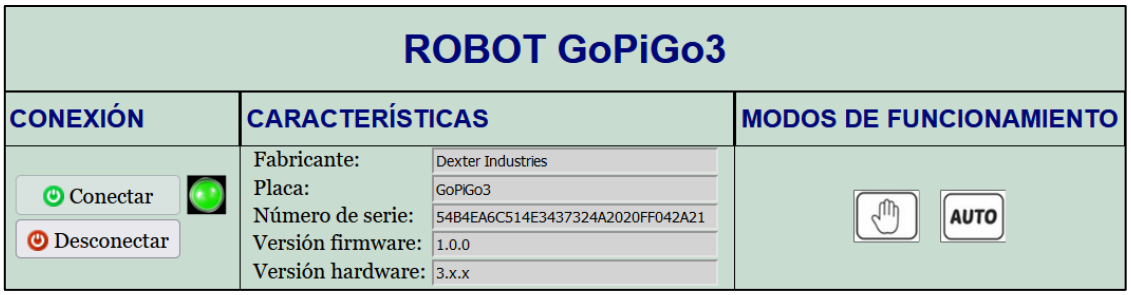
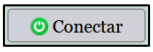


Figura 27: *Panel PanelGeneralRobotGoPiGo3*

El panel *CONEXIÓN* (ver *Figura 28*) sirve para establecer/cerrar conexión entre el cliente RIP y el servidor RIP:

- 

: al pulsar el botón *Conectar* se ejecutará el comando **RIP.connect()** proporcionado por el cliente. De este modo, se intentará establecer conexión entre la aplicación informática EjsS y el servidor RIP que se está ejecutando en la placa Raspberry Pi. En el caso de que se establezca la conexión correctamente, se habilitará el resto de la interfaz para que el usuario pueda interactuar y operar con el robot.
- 

: mediante el botón *Desconectar* se cerrará la conexión con el robot, y se pararan todas las acciones y movimientos que se estaban ejecutando en el robot GoPiGo3.
- 
: indica el estado de conexión actual. Si se visualiza este LED quiere decir que la aplicación informática está conectada con el servidor RIP.
- 
: indica el estado de conexión actual. Si se visualiza este LED quiere decir que la aplicación informática no está conectada con el servidor RIP.

Figura 28: *Panel CONEXIÓN*

En el apartado *4.2.1 Añadir, configurar y utilizar el elemento RIP* se ha mencionado que el método get no se utiliza mucho en la aplicación EjsS de este proyecto. Este método se implementa solamente en los botones *Conectar* y *Desconectar*. Esto se debe a que al establecer/cerrar la conexión con el robot hay que obtener las características del robot GoPiGo3 mencionadas a continuación (mirar la *Tabla 5* del apartado *4.1.3 RIPGoPiGo3.py*).

El panel *CARACTERÍSTICAS* es solamente de visualización (ver *Figura 29*). Se muestran algunas características y referencias del robot GoPiGo3:

- Fabricante de GoPiGo3.
- Referencia placa GoPiGo3.
- Número de serie del hardware GoPiGo3.
- Versión firmware.

- Versión hardware.

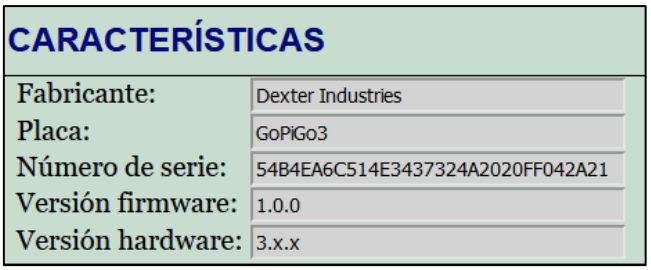


Figura 29: *Panel CARACTERÍSTICAS*

El panel *MODOS DE FUNCIONAMIENTO* gestiona los modos de funcionamiento del ciclo de trabajo del robot (ver *Figura 30*). Existen dos modos de funcionamiento:

-  : seleccionar modo de funcionamiento manual.
- 

: seleccionar modo de funcionamiento automático.

Figura 30: *Panel MODOS DE FUNCIONAMIENTO*

4.2.2.2 Panel PanelControlRobotGoPiGo3

En la *Figura 31* y *Figura 32* se muestra que el panel *PanelControlRobotGoPiGo3* se divide en dos subpaneles: (1) *PanelPrincipalControlRobotGoPiGo3* y (2) *PanelVisorDeAvisos*.

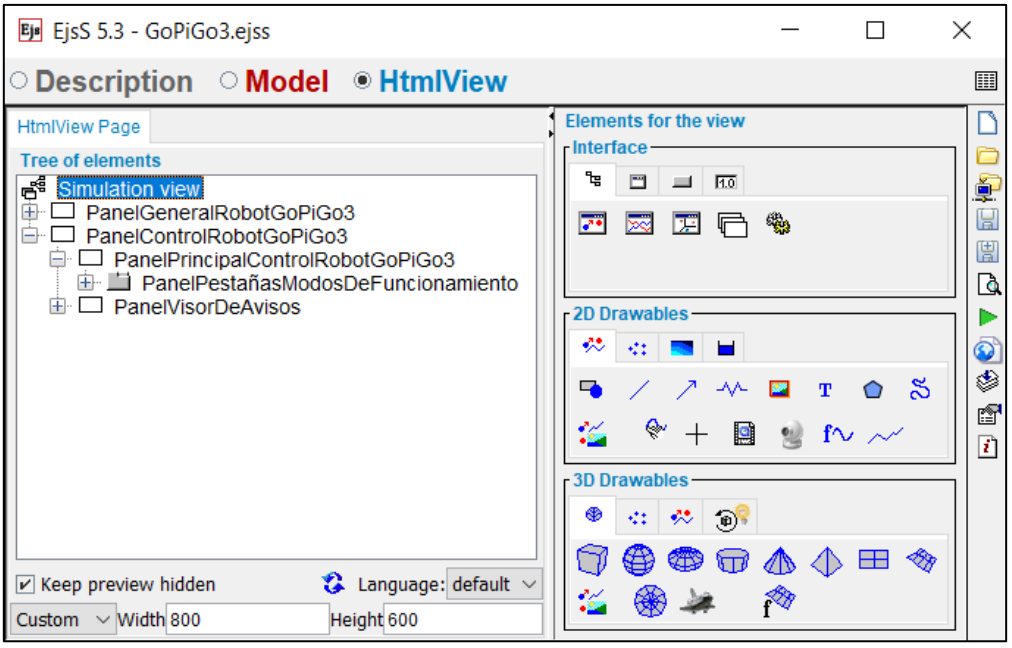


Figura 31: *Estructura del panel PanelControlRobotGoPiGo3*

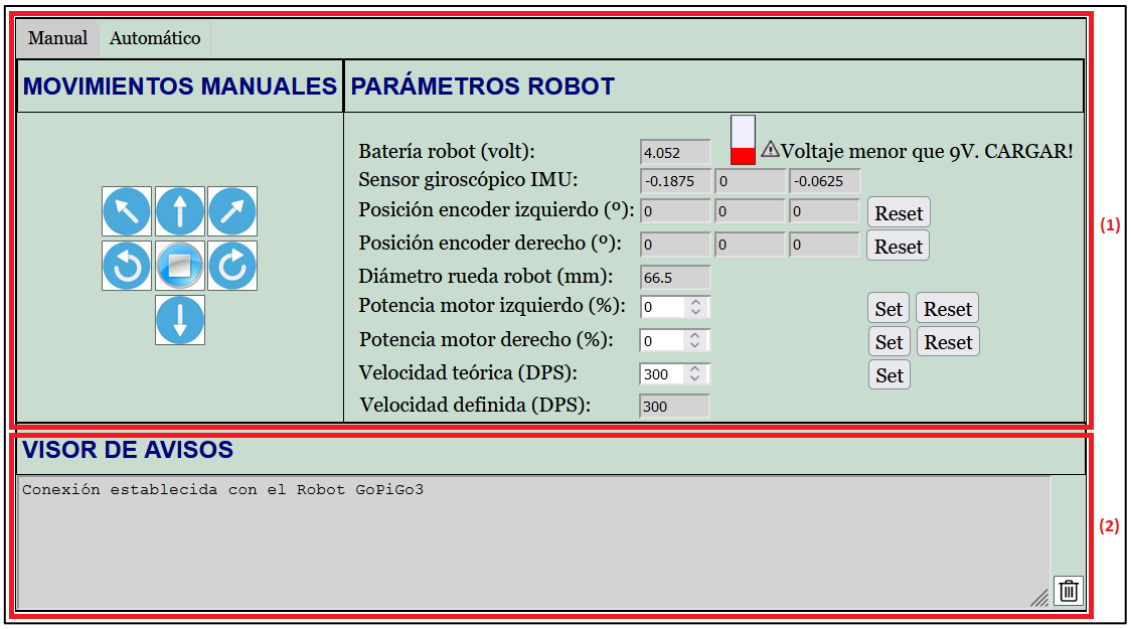


Figura 32: *Panel PanelControlRobotGoPiGo3. (1) PanelPrincipalControlRobotGoPiGo3 y (2) PanelVisorDeAvisos*

4.2.2.2.1 Panel PanelPrincipalControlRobotGoPiGo3

En la aplicación EjsS se ha añadido un elemento llamado *TabbedPanel*. Este elemento es un contenedor que muestra solo un elemento secundario cada vez, es decir, es un separador con pestañas. De este modo, dependiendo del modo de funcionamiento (manual/automático) seleccionado en el panel *MODOS DE FUNCIONAMIENTO* (*Figura 30*) se visualizará una pestaña u otra, mostrando un contenido diferente. Existen dos posiciones diferentes para las pestañas:

- **Manual**: se visualiza esta pestaña si se selecciona el modo de funcionamiento manual.
- **Automático**: se visualiza esta pestaña en el caso de que se haya seleccionado el modo de funcionamiento automático.

El panel *MANUAL* (ver *Figura 33*) se divide a su vez en otros dos subpaneles: *MOVIMIENTOS MANUALES* y *PARÁMETROS ROBOT*.

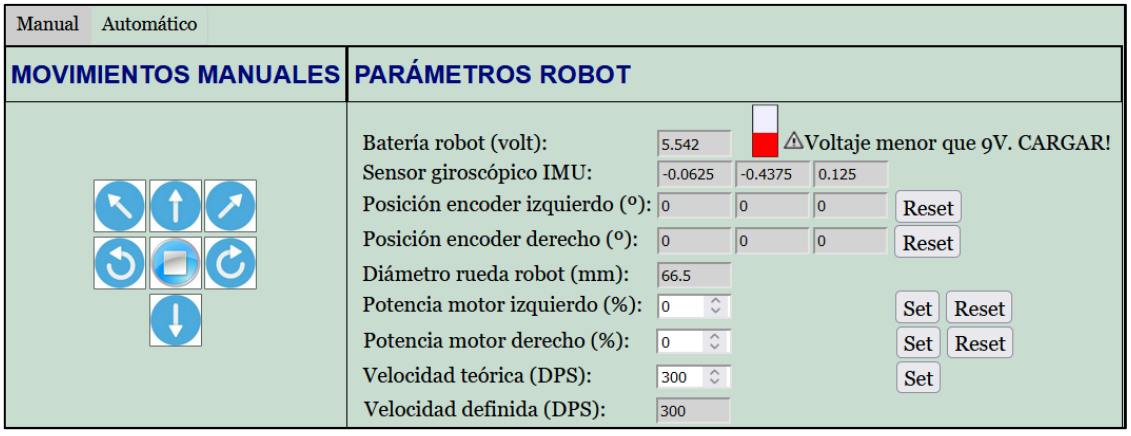


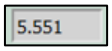
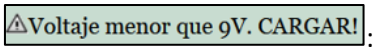
Figura 33: *Panel MANUAL*

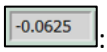
El panel *MOVIMIENTOS MANUALES* se utilizará para realizar los movimientos manuales del robot (ver *Figura 34*):

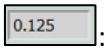
-  : mover el robot GoPiGo3 hacia el lado izquierdo.
-  : mover el robot GoPiGo3 hacia adelante.
-  : mover el robot GoPiGo3 hacia el lado derecho.
-  : girar el robot GoPiGo3 hacia el lado izquierdo.
-  : parar el robot GoPiGo3.
-  : girar el robot GoPiGo3 hacia el lado derecho.
-  : mover el robot GoPiGo3 hacia atrás.

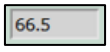
Figura 34: *Panel MOVIMIENTOS MANUALES*

El panel *PARÁMETROS ROBOT* del modo manual está relacionado principalmente con los parámetros del robot GoPiGo3 (ver *Figura 35*):

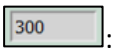
- **Batería robot**: visualiza el voltaje de la batería del paquete de baterías recargable en voltios (V). Al ser una variable de lectura no se puede modificar su valor.
  -  : indica el voltaje de la batería en voltios (indicador numérico).
  -  : indica el voltaje de la batería (indicador visual):
    -  : voltaje mayor que 9.5 V.
    -  : voltaje entre 9 V y 9.5 V.
    -  : voltaje menor que 9 V.
  - 

: mensaje o aviso para el usuario. Se visualiza este mensaje cuando el voltaje de la batería del robot es menor que 9 V, para que el usuario sepa que tiene que cargar el paquete de baterías.
- **Sensor giroscópico IMU**: indica la medición del giroscopio integrado en el sensor IMU. Al ser una variable de lectura no se puede modificar su valor.
  -  : indica la velocidad angular a lo largo del eje X en grados/seg (°/s).

-  : indica la velocidad angular a lo largo del eje Y en grados/seg (°/s).
-  : indica la velocidad angular a lo largo del eje Z en grados/seg (°/s).

- **Posición encoder izquierdo**: indica la posición del encoder del motor izquierdo. Al ser una variable de lectura no se puede modificar su valor.
  -  : indica la posición del encoder del motor izquierdo en grados (°).
  -  : indica el número de vueltas del encoder del motor izquierdo.
  -  : indica la posición de la última vuelta del encoder del motor izquierdo (°).
  -  : resetea el valor del encoder del motor izquierdo y derecho. Establece la posición de ambos encoders en 0°.
- **Posición encoder derecho**: indica la posición del encoder del motor derecho. Al ser una variable de lectura no se puede modificar su valor.
  -  : indica la posición del encoder del motor derecho en grados (°).
  -  : indica el número de vueltas del encoder del motor derecho.
  -  : indica la posición de la última vuelta del encoder del motor derecho (°).
  -  : resetea el valor del encoder del motor izquierdo y derecho. Establece la posición de ambos encoders en 0°.
- **Diámetro rueda robot**: al ser una variable de lectura no se puede modificar su valor.
  -  : indica el diámetro de la rueda del robot en milímetros (mm).
- **Potencia motor izquierdo**: define la potencia del motor izquierdo. Es una variable de escritura, por lo tanto, se puede modificar su valor.
  -  : nuevo valor de la potencia para el motor izquierdo en porcentaje (%). Hay que escribir un valor entero entre -100% y 100%.
  -  : escribe o establece el nuevo valor de la potencia del motor izquierdo.
  -  : resetea la potencia del motor izquierdo (potencia = 0%).
- **Potencia motor derecho**: define la potencia del motor derecho. Es una variable de escritura, por lo tanto, se puede modificar su valor.
  -  : nuevo valor de la potencia para el motor derecho en porcentaje (%). Hay que escribir un valor entero entre -100% y 100%.
  -  : escribe o establece el nuevo valor de la potencia del motor derecho.
  -  : resetea la potencia del motor derecho (potencia = 0%).
- **Velocidad teórica**: define la velocidad del robot para los movimientos manuales (panel *MOVIMIENTOS MANUALES*, ver *Figura 34*). Es una variable de escritura, por lo tanto, se puede modificar su valor.
  -  : nuevo valor de la velocidad en DPS (grados por segundo de la rueda del robot). Hay que escribir un valor entero entre 0 DPS y 1000 DPS.

-  : escribe o establece el nuevo valor de la velocidad para los movimientos manuales del robot.

- **Velocidad definida**: indica la velocidad definida para los movimientos manuales del robot. Está relacionado con el parámetro anterior (es la respuesta del parámetro anterior). Al ser una variable de lectura no se puede modificar su valor.
  -  : indica la velocidad definida en el robot en DPS. Tiene que ser un valor entre 0 DPS y 1000 DPS.

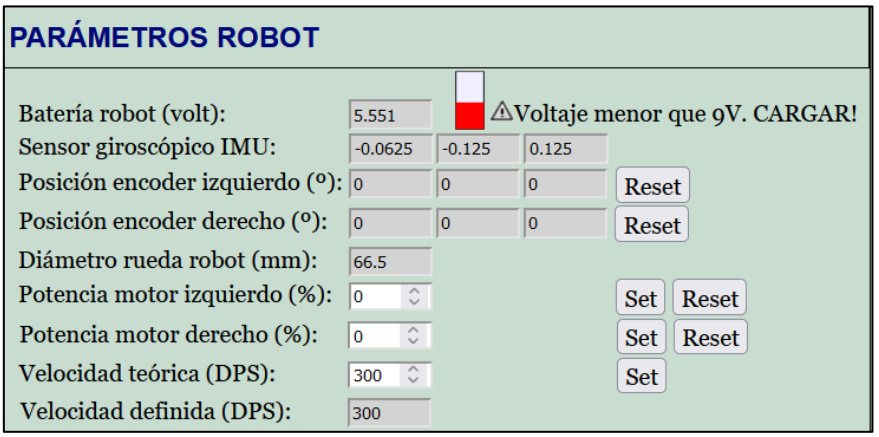


Figura 35: *Panel PÁRAMETROS ROBOT (modo manual)*

El panel *AUTOMÁTICO* (ver *Figura 36*) se divide a su vez en otros dos subpaneles: *PARÁMETROS ROBOT* y *PROGRAMAS ROBOT*.

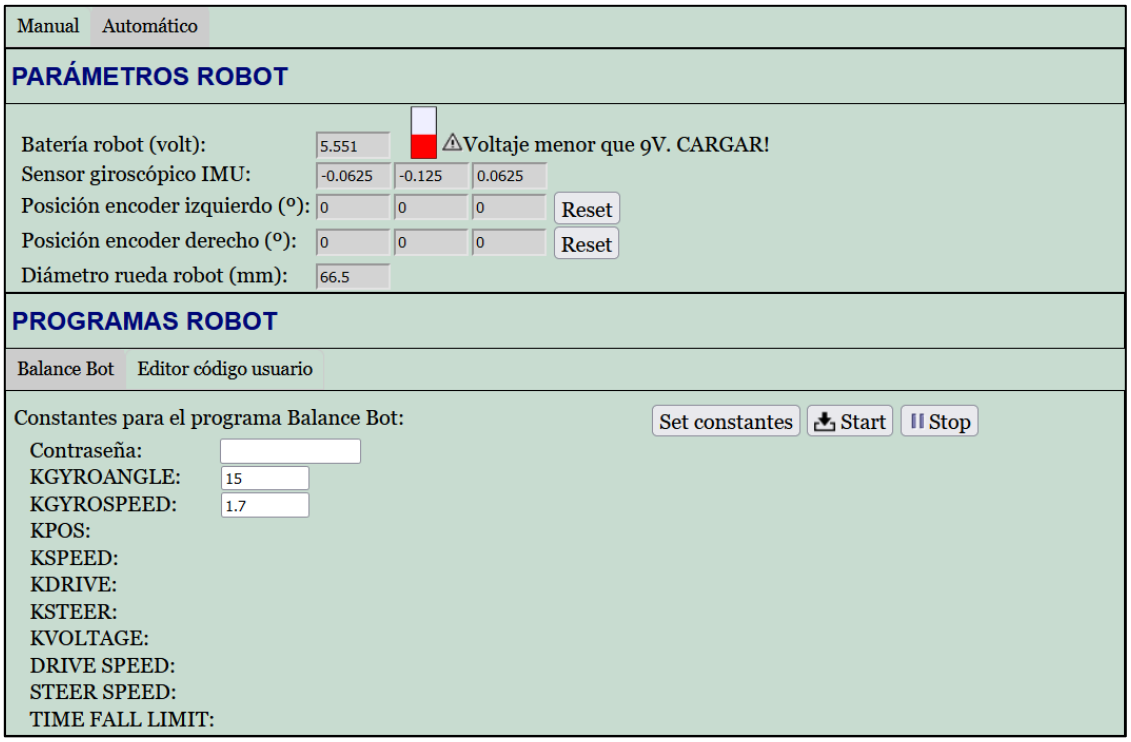


Figura 36: *Panel AUTOMÁTICO*

El panel *PARÁMETROS ROBOT* del modo automático está relacionado principalmente con los parámetros del robot GoPiGo3 (ver *Figura 37*). Las variables de lectura que se visualizan en este panel son los mismos que se han mencionado en la *Figura 35*. Comparando con ese panel, en

este caso no hay la opción de escribir ningún parámetro para el control del robot, puesto que esa opción solamente se habilita en el modo de funcionamiento manual.

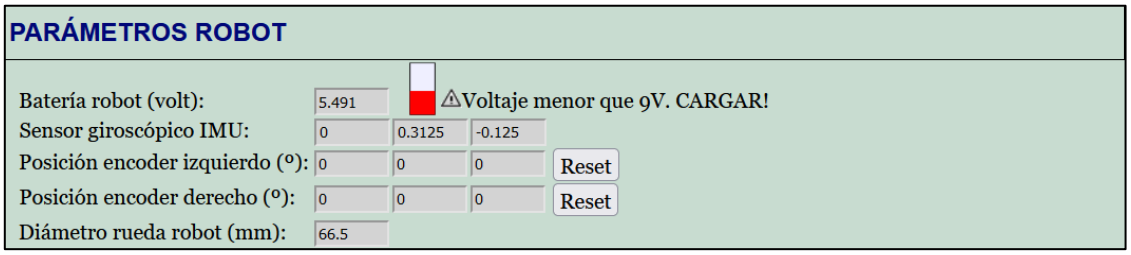


Figura 37: *Panel PÁRAMETROS ROBOT (modo automático)*

En la *Figura 38* se muestra una imagen del panel *PROGRAMAS ROBOT*.

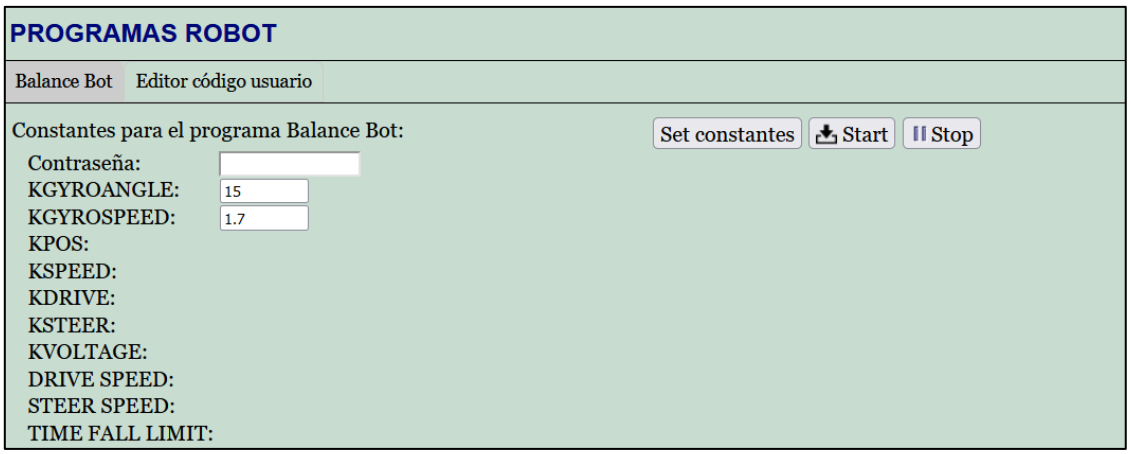


Figura 38: *Panel PROGRAMAS ROBOT*

Tal y como se ha visto en el apartado *4.1.4 GoPiGo3Operation.py*, cuando se selecciona el modo de funcionamiento automático existe la opción de ejecutar diferentes programas de control. Por un lado, existe la opción de ejecutar el programa Balance Bot para mantener el robot GoPiGo3 en equilibrio vertical. Por otro lado, en la interfaz web se ha incluido un editor de código (script) que le permite al usuario desarrollar su propio programa de control en lenguaje Python. En el panel *PROGRAMAS ROBOT* se ha añadido otro elemento de tipo *TabbedPanel* (ver *Figura 38*). De este modo, el usuario podrá elegir una de las dos pestañas siguientes:

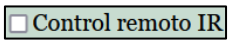
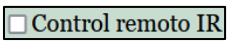
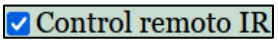
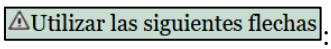
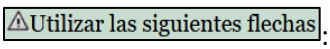
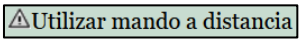
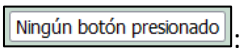
- **Balance Bot**: en esta pestaña se podrá iniciar la ejecución del programa Balance Bot que está incluido en la clase GoPiGo3Operation (mirar apartado *4.1.4.1 Programa BalanceBot*). Además, se ofrece la posibilidad de cambiar el valor de algunas constantes de dicho programa.
    - **Contraseña**: en el apartado *4.1.4.1 Programa BalanceBot* se ha explicado el significado de todas las constantes que se visualizan en esta pestaña. Si existen problemas para mantener el robot en equilibrio, conviene empezar a ajustar las constantes KGYROANGLE y KGYROSPEED. Por ello, en un principio el usuario tendrá acceso solamente para modificar el valor de estas dos constantes. Aun así, introduciendo la contraseña **UNED** en este campo se podrá modificar el valor de todas las constantes. Es una variable de escritura, por lo tanto, se puede modificar su valor.

- [ ]: campo para introducir la contraseña **UNED** y habilitar la visualización de todas las constantes.

- **KGYROANGLE**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [15]: campo para definir o introducir el valor del constante KGYROANGLE.
- **KGYROSPEED**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [1.7]: campo para definir o introducir el valor del constante KGYROSPEED.
- **KPOS**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [0.07]: campo para definir o introducir el valor del constante KPOS.
- **KSPEED**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [0.1]: campo para definir o introducir el valor del constante KSPEED.
- **KDRIVE**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [-0.02]: campo para definir o introducir el valor del constante KDRIVE.
- **KSTEER**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [0.25]: campo para definir o introducir el valor del constante KSTEER.
- **KVOLTAGE**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [0.083]: campo para definir o introducir el valor del constante KVOLTAGE.
- **DRIVE SPEED**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [175]: campo para definir o introducir el valor del constante DRIVE SPEED.
- **STEER SPEED**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [100]: campo para definir o introducir el valor del constante STEER SPEED.
- **TIME FALL LIMIT**: es una variable de escritura, por lo tanto, se puede modificar su valor.
  - [2]: campo para definir o introducir el valor del constante TIME FALL LIMIT.
- [Set constantes]: escribe o establece el valor de las constantes definidas anteriormente para el programa Balance Bot.
- [Start]: botón para iniciar la ejecución del programa Balance Bot.

-  : botón para parar la ejecución del programa Balance Bot.

Una vez conseguido equilibrar el robot GoPiGo3, en el programa Balance Bot se ha incluido la opción de conducir el robot. Por ello, cuando se inicia la ejecución del programa Balance Bot pulsando el botón *Start*, se visualizan controles para la conducción del robot GoPiGo3. Existen dos opciones para la conducción del robot: la primera sería utilizar el mando a distancia por infrarrojos, y, la segunda, utilizar las flechas (adelante, izquierda, derecha y atrás) de la *Figura 39*.

-  : activa/desactiva el control remoto IR. Es una variable de escritura, por lo tanto, se puede modificar su valor.
  -  : desactivar control remoto IR. Se visualizan las flechas en la interfaz web para la conducción del robot (*Figura 39*). No hay que utilizar el mando a distancia por infrarrojos.
  -  : activar control remoto IR. Hay que utilizar el mando a distancia por infrarrojos para la conducción del robot y no se visualizan las flechas en la interfaz web (*Figura 40*). En el mando a distancia hay que utilizar los botones: adelante, izquierda, OK, derecha y atrás.
-  : mensaje o aviso para el usuario.
  -  : mensaje que se visualiza para el usuario cuando el control remoto IR está desactivado (*Figura 39*).
  -  : mensaje que se visualiza para el usuario cuando el control remoto IR está activado (*Figura 40*).
- **Botón actual seleccionado**: indica la flecha seleccionada para el control remoto (en ambos casos, cuando se utiliza el mando a distancia por infrarrojos o las flechas de la interfaz web). Al ser una variable de lectura no se puede modificar su valor.
  -  : indica la flecha seleccionada. Visualiza las siguientes opciones: adelante, izquierda, OK, derecha, atrás, y, ningún botón seleccionado.
-  : conducir el robot GoPiGo3 hacia adelante cuando el control remoto IR está desactivado.
-  : conducir el robot GoPiGo3 hacia el lado izquierdo cuando el control remoto IR está desactivado.
-  : seleccionar botón OK cuando el control remoto IR está desactivado.
-  : conducir el robot GoPiGo3 hacia el lado derecho cuando el control remoto IR está desactivado.
-  : conducir el robot GoPiGo3 hacia atrás cuando el control remoto IR está desactivado.

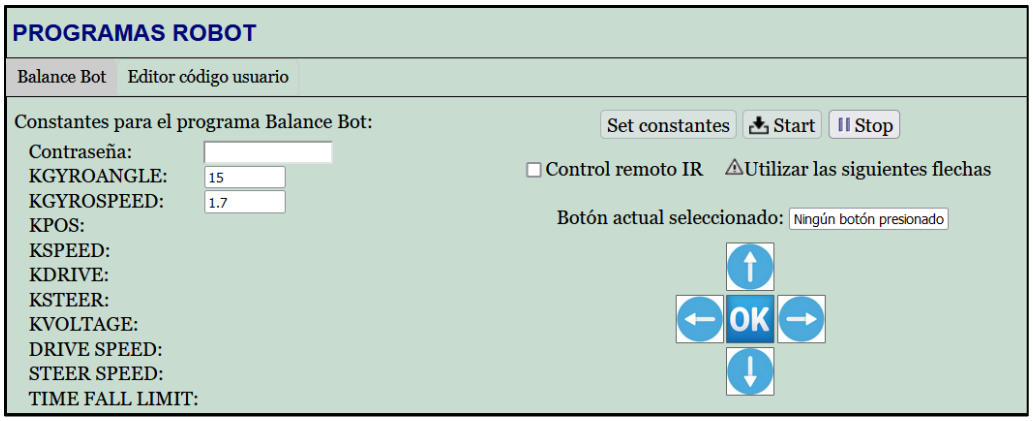


Figura 39: *Panel PROGRAMAS ROBOT BALANCE BOT, Control Remoto desactivado*

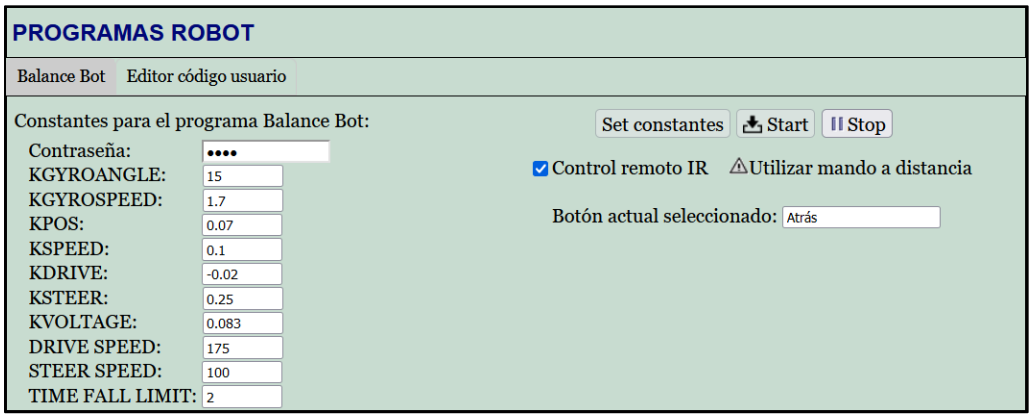


Figura 40: *Panel PROGRAMAS ROBOT BALANCE BOT, Control Remoto activado*

- **Editor código usuario**: en esta pestaña el usuario podrá desarrollar su programa de control (ver *Figura 41*). Para ello, tendrá que utilizar el lenguaje de programación Python. Esta pestaña ofrece las siguientes opciones:
  - **Nombre del archivo a guardar**: define el nombre con el que se quiere guardar el programa de control desarrollado por el usuario. Es una variable de escritura, por lo tanto, se puede modificar su valor.
    -  : campo para definir o introducir el nombre del programa de control desarrollado por el usuario.
  -  : botón para guardar el programa de control desarrollado por el usuario. Se guarda el archivo en el PC local del usuario.
  -  : botón para abrir un fichero local del PC del usuario. Al abrir el fichero se copiará el código en el área de texto donde el usuario desarrolla su programa de control.
  - **Área de texto** (): el usuario tiene la opción de escribir su programa de control en esta área. Es un campo de escritura, por lo tanto, se puede modificar su valor.
  -  : botón para descargar e iniciar el programa de control desarrollado por el usuario en el robot.
  -  : botón para parar el programa de control desarrollado por el usuario.

-  : botón para borrar el contenido del área de texto donde el usuario escribe su código.

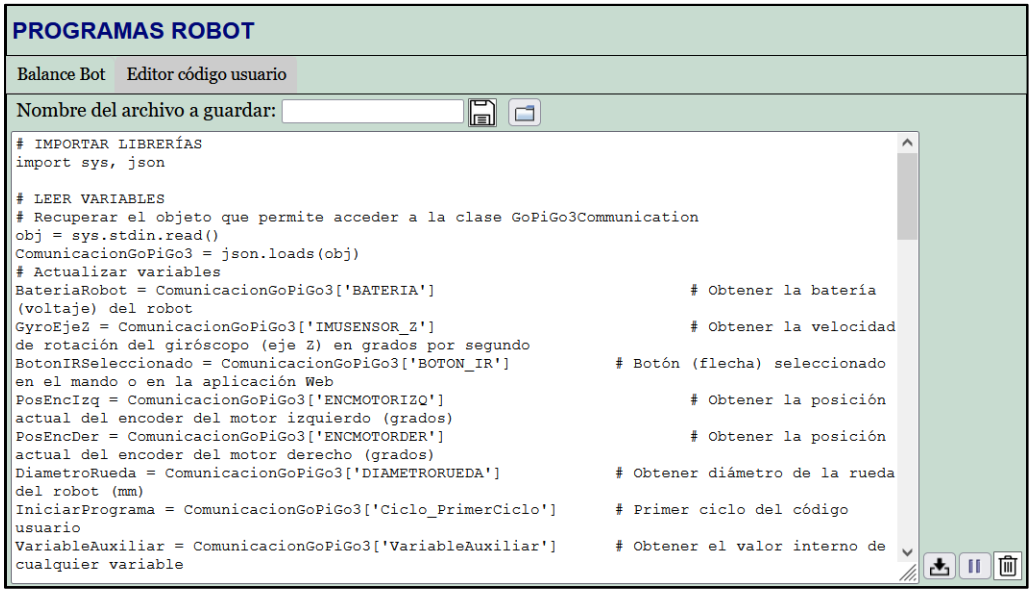


Figura 41: *Panel PROGRAMAS ROBOT EDITOR CÓDIGO USUARIO*

4.2.2.2.2 Panel PanelVisorDeAvisos

En este último panel se muestran los mensajes o logs generados en el servidor RIP del robot GoPiGo3 (ver *Figura 42*). Estos mensajes se pueden producir en la ejecución del programa de control desarrollado por el usuario o en la ejecución del ciclo de trabajo del robot. De este modo, el usuario tendrá información del robot en todo momento.

- **Área de texto**: visualiza los mensajes o logs. Al ser un campo de lectura no se puede modificar su valor.
-  : botón para borrar el contenido del área de texto donde se muestran los mensajes o logs generados en el servidor RIP del robot GoPiGo3.

Figura 42: *Panel VISOR DE AVISOS*

5. Pruebas y resultados

En este apartado se muestran las pruebas realizadas y los resultados obtenidos, con el fin de saber si se han cumplido o no los objetivos propuestos al principio del trabajo. Principalmente, se han realizado las siguientes pruebas:

- Establecer conexión entre la aplicación informática EjsS (cliente RIP) y el robot GoPiGo3 (servidor RIP que se ejecuta en la placa Raspberry Pi).
- Probar los distintos modos de funcionamiento del ciclo de trabajo del robot y las funciones que ofrece cada modo.

Se ha grabado un video donde se muestran las pruebas realizadas (este video se ha adjuntado en la entrega del trabajo).

5.1 Prueba 1. Establecimiento de conexión entre interfaz web EjsS y servidor RIP

La primera prueba que se ha hecho es establecer la conexión entre la aplicación informática EjsS y el robot GoPiGo3. En la *Figura 43* se muestran los resultados obtenidos, donde el LED de color verde muestra el estado de la conexión (conexión establecida correctamente). En el caso de que se establezca la conexión correctamente se visualiza el resto de la interfaz, entre otras cosas, las características del robot GoPiGo3, como, por ejemplo, fabricante, referencia de la placa, versión firmware …

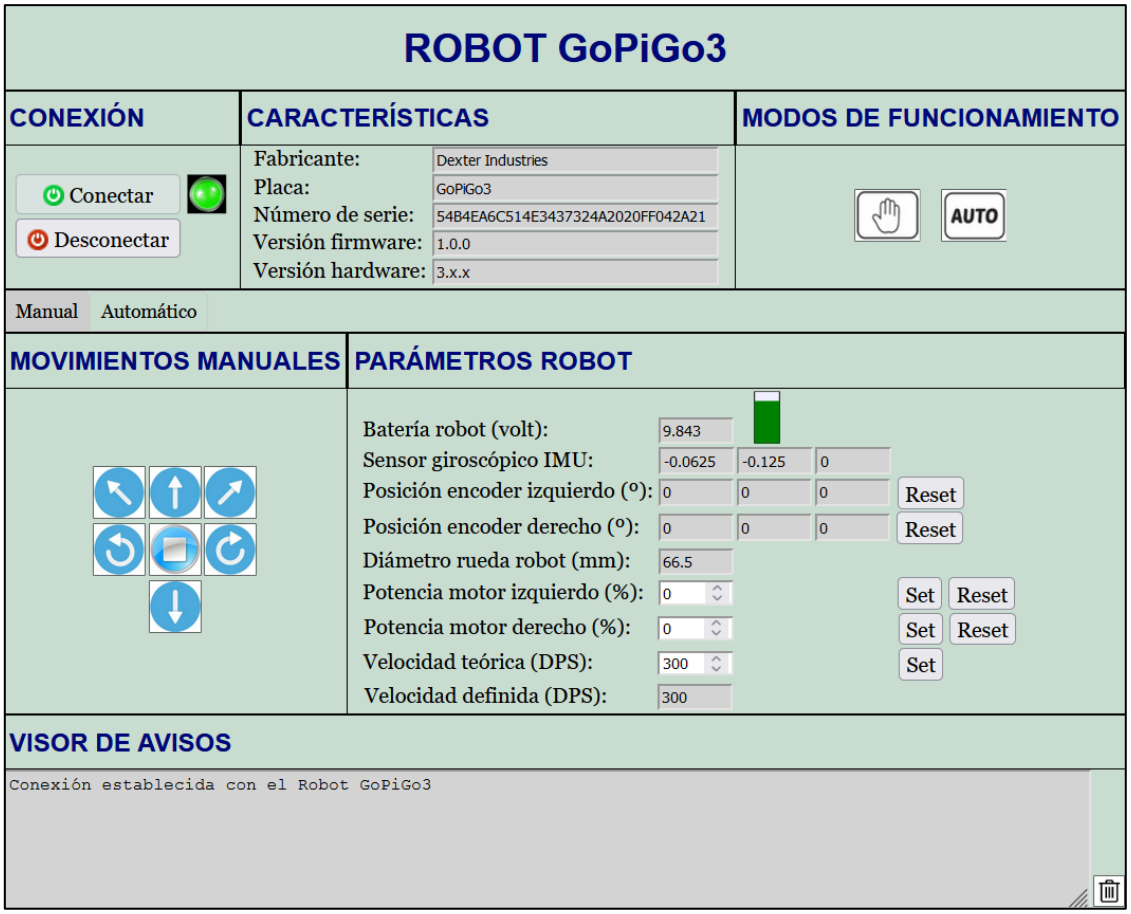


Figura 43: *Interfaz web. Establecimiento de conexión*

Al establecer conexión con el robot, por un lado, se establece conexión con el canal SSE [28] y se ejecutan los métodos de protocolo de comunicación (set y get), los cuales influyen en el ciclo de trabajo del robot. Por el otro lado, se inicia un subproceso (thread) dentro de la clase GoPiGo3Operation del código Python (ver apartado *4.1.4 GoPiGo3Operation.py*). Los subprocesos en Python son una entidad dentro de un proceso que se pueden programar para su ejecución. Es decir, es una secuencia de tales instrucciones dentro de un programa que se puede ejecutar independientemente de otros códigos. Los subprocesos proporcionan una forma de mejorar el rendimiento de las aplicaciones mediante el paralelismo. En este caso, tal y como se observa en el *Código 2*, se inicia el subproceso *Worker*.

```
# Conectar
    def connect(self):
        try:
            self.ResetVariables()                # Inicializar variables del programa
            self.GoPiGo3Communication.reset()    # Inicializar variables de la clase
GoPiGo3Communication
            self.setPotenciaMotorIzq(0) # Potencia motor izquierdo (parar robot)
            self.setPotenciaMotorDer(0) # Potencia motor derecho (parar robot)
            self.ResetRobotEncoders(0)  # Resetear posición de los encoders del motor
izquierdo y derecho
            self.Running = True          # Conectado y en ejecución
            self.thread  =  threading.Thread(target=self.Worker)    #  Crear  hilo  de
ejecución (función Worker)
            self.thread.start()                                  # Iniciar la actividad
del hilo
            Fabricante = self.GPG.get_manufacturer()             # Fabricante GoPiGo3
            ReferenciaPlaca = self.GPG.get_board()                 # Referencia placa
GoPiGo3
            NumeroSerie = self.GPG.get_id()                      # Número de serie del
hardware GoPiGo3
            Version_fw = self.GPG.get_version_firmware()         # Versión firmware
            Version_hw = self.GPG.get_version_hardware()         # Versión hardware
            Conectado = True                                     # Conectado (variable
para la aplicación Web)
            Msg = "Conexión establecida con el Robot GoPiGo3"
            self.AplicacionWeb_Msg = Msg                           # Mensaje a mostrar
en el Visor de Avisos de la aplicación Web
            return Conectado, Fabricante, ReferenciaPlaca, NumeroSerie, Version_fw,
Version_hw  # Devolver valores
        except:                                                    # Error detectado
durante la ejecución del programa
            Msg = str("Error inesperado durante el establecimiento de la conexión con
el robot GoPiGo3:"+str(sys.exc_info()[0])) # sys.exc_info()[0]: tipo de excepción
            self.AplicacionWeb_Msg = Msg                         #    Mensaje    a
mostrar en el Visor de Avisos de la aplicación Web
            print(Msg)
```

Código 2: *Clase GoPiGo3Operation. Función Connect, iniciar subproceso Worker*

Mediante el subproceso *Worker* se realiza la lectura de los parámetros del robot GoPiGo3. Para ello, se utiliza la función *LeerDatos()* del *Código 3*, que sirve para leer: voltaje de la batería, sensor giroscópico IMU, receptor IR, posición de los encoders del motor izquierdo y derecho, y, diámetro de la rueda del robot. Además, mediante la función *ControlProgramaRobot()* se ejecuta el programa de control, es decir, el programa Balance Bot (ver apartado *4.1.4.1 Programa BalanceBot*) o el código desarrollado por el usuario (ver apartado *4.1.4.2 Programa de control del usuario*).

```
# Worker. Hilo de ejecucion (de este modo se permite que el programa ejecute
múltiples operaciones simultáneamente en el mismo espacio de proceso)
    def Worker(self):
        while(self.Running):     # Conectado y en ejecución
            self.LeerDatos()     # Leer datos del robot GoPiGo3 (batería, IMU sensor,
receptor IR, encoders y diámetro de la rueda)
            if self.ModoControlRobot == "Automatico" and
(self.StartProgramaBalanceBot or self.StartScriptUsuario):  # Modo de funcionamiento
Automático y programa en ejecución
                self.ControlProgramaRobot()  # Ejecutar el programa Balance Bot o el
programa de control (script) desarrollado por el usuario
                # Actualizar parámetros del robot GoPiGo3
                self.setPotenciaMotorIzq(self.GoPiGo3Communication.POTENCIAMOTORIZQ)
# Definir potencia motor izquierdo
                self.setPotenciaMotorDer(self.GoPiGo3Communication.POTENCIAMOTORDER)
# Definir potencia motor derecho
                if self.GoPiGo3Communication.RESETENCODERS:
                    self.ResetRobotEncoders(0)  # Resetear posición de los encoders
del motor izquierdo y derecho
                time.sleep(0.02)     # Esperar 0.02 segundos
            else:
                time.sleep(0.1)      # Esperar 0.1 segundos
```

Código 3: *Clase GoPiGo3Operation. Función Worker, funciones LeerDatos() y ControlProgramaRobot()*

5.2 Prueba 2. Modos de funcionamiento del ciclo de trabajo del robot

El robot GoPiGo3 tiene dos modos de funcionamiento: modo manual y modo automático.

En el modo de trabajo manual se realiza el control manual del robot. Por un lado, se pueden realizar los movimientos manuales del robot mediante las flechas habilitadas en la interfaz web.

Por ejemplo, si se pulsa la flecha para mover el robot hacia adelante () se ponen en marcha los dos motores del robot a la misma velocidad, para que el robot avance hacia adelante en línea recta. Además, en el visor de avisos aparece un mensaje indicándole al usuario de que el robot se está moviendo hacia adelante (ver *Figura 44*).

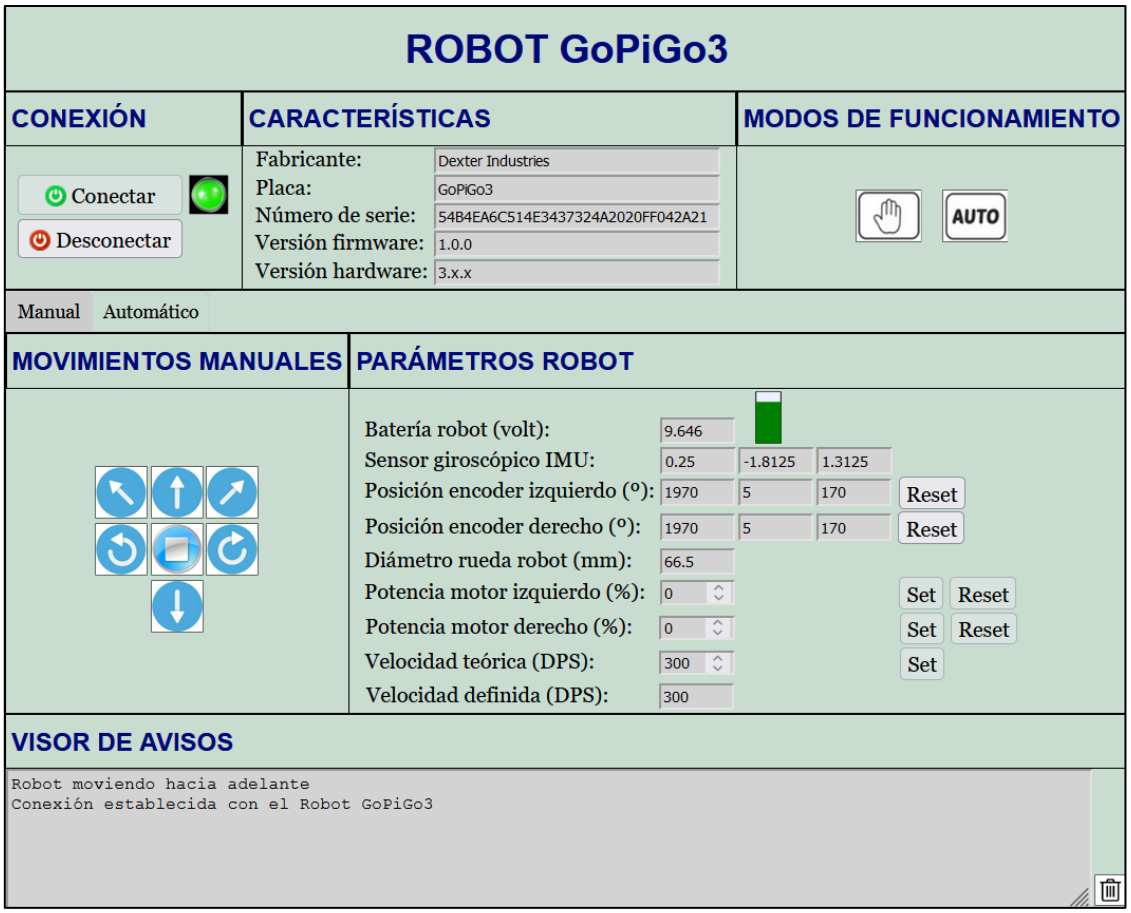


Figura 44: *Interfaz web. Modo manual robot GoPiGo3 moviendo hacia adelante*

Por otra parte, en modo manual se visualizan los parámetros de lectura de la *Figura 45* (el valor de estos parámetros se actualiza constantemente mediante el canal SSE [28]):

- Voltaje de la batería en voltios (V).
- Valor del sensor giroscópico IMU a lo largo de los ejes X, Y, Z, en grados/seg (°/s).
- Posición del encoder del motor izquierdo y derecho. Se muestra la posición en grados (°), el número de vueltas del encoder, y los grados de la última vuelta.
- Diámetro de la rueda del robot en milímetros (mm). A la hora de construir el controlador PID para mantener en equilibrio el robot GoPiGo3, el entorno es un factor determinante. Entre otras cosas el tamaño de las ruedas influye, por lo tanto, se visualiza el diámetro de la rueda para que el usuario tenga dicha información.
- Velocidad de los motores para los movimientos manuales del robot en DPS (grados por segundo de la rueda del robot).

Y también se ofrece la opción de modificar el valor de los siguientes parámetros:

- Potencia motor izquierdo y derecho en porcentaje (%).
- Velocidad de los motores para los movimientos manuales del robot en DPS (grados por segundo de la rueda del robot).

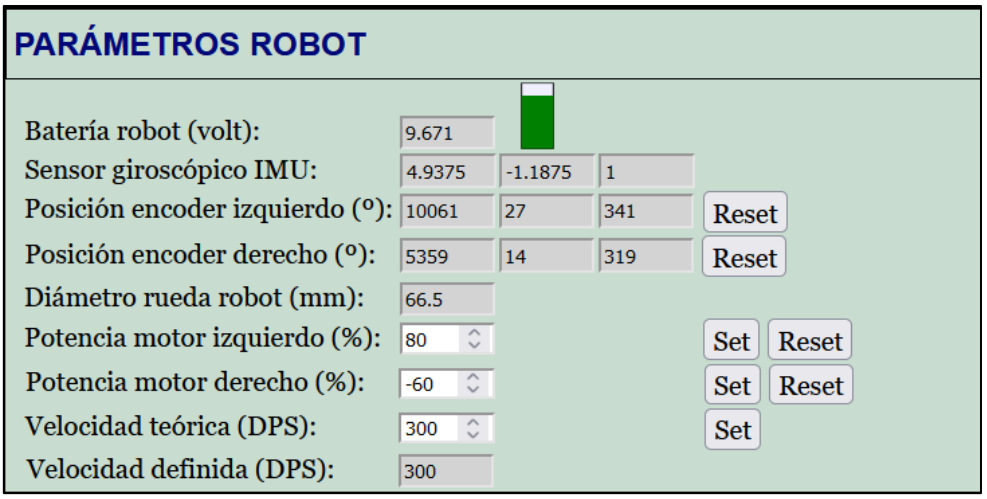


Figura 45: *Interfaz web. Modo manual parámetros del robot GoPiGo3*

En el modo de funcionamiento automático, se visualizan los mismos parámetros de lectura que en el modo manual. Además, en el ciclo automático existe la posibilidad de ejecutar programas de control. Por un lado, se ha creado el programa Balance Bot [32], que se utiliza para mantener el robot GoPiGo3 en equilibrio vertical. En el apartado *4.1.4.1 Programa BalanceBot* se ha comentado que se han creado varías constantes [36] para un correcto comportamiento de equilibrio del robot, donde el usuario tiene la opción de modificar el valor de dichas constantes mediante la aplicación informática EjsS. En la *Figura 46* se observa el valor de estas constantes para que el robot se mantenga en equilibrio.

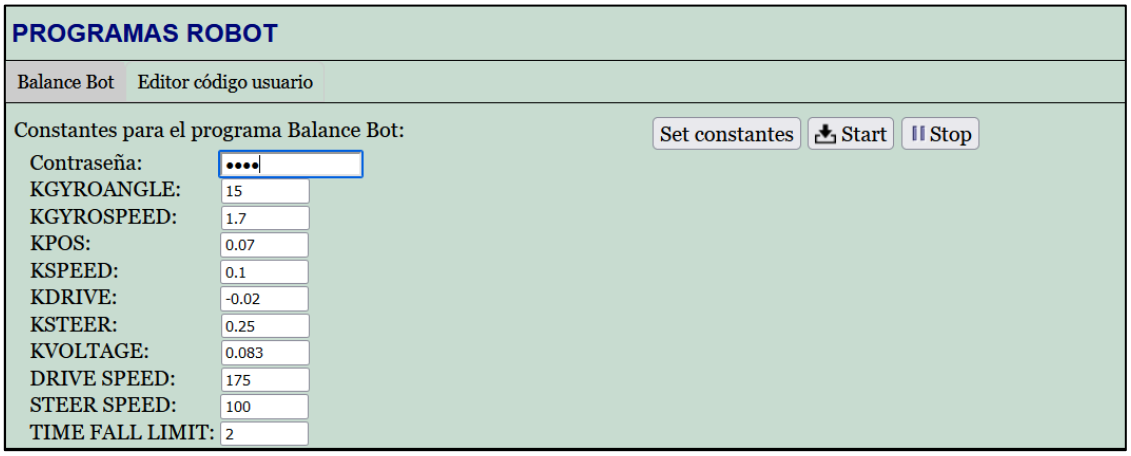


Figura 46: *Interfaz web. Constantes programa Balance Bot*

Una vez conseguido equilibrar el robot, se puede conducir el robot mediante el mando a distancia por infrarrojos o los botones de la interfaz web (mirar apartado *4.2.2.2.1 Panel PanelPrincipalControlRobotGoPiGo3*). En el video que se ha grabado también se observa que el robot responde bastante bien frente a perturbaciones externas.

Por otro lado, en modo de funcionamiento automático el usuario tiene la opción de desarrollar sus propios programas de control en la interfaz web, y después, descargar y ejecutar en el robot. Es cierto que el usuario podría desarrollar su propio programa de control para mantener el robot GoPiGo3 en equilibrio y ejecutar desde la interfaz web. En este caso, el programa Balance Bot [32] que se ha creado para el equilibrio del robot utiliza como algoritmo de control el controlador PID. Este controlador se debe ejecutar con suficiente frecuencia y dentro del rango controlable

del sistema. Tal y como se explicará más adelante, para iniciar el código desarrollado por el usuario se ejecuta un subproceso dentro de la clase GoPiGo3Operation. El tiempo de ejecución de este subproceso es demasiado alto para que el controlador PID del programa Balance Bot funcione adecuadamente, por lo que se ha decidido crear una nueva función dentro de la clase GoPiGo3Operation (llamada *ProgramaBalanceBot()*) que contiene el programa para mantener el robot GoPiGo3 en equilibrio vertical. Para más detalles consultar el apartado *10.4 Clase GoPiGo3Operation.py*.

En el apartado *4.1.5 GoPiGo3Communication.py* se ha mencionado que se ha desarrollado un código de usuario como ejemplo para descargar y ejecutar en el robot desde la interfaz web (para más detalles mirar apartado *10.6 Código usuario ejemplo*). En este ejemplo, utilizando el mando a distancia por infrarrojos se realizan los movimientos del robot (adelante, izquierda, derecha y atrás), definiendo la potencia correspondiente a los motores en cada caso. En el visor de avisos se muestran los mensajes generados en el código del usuario (ver *Figura 47*).

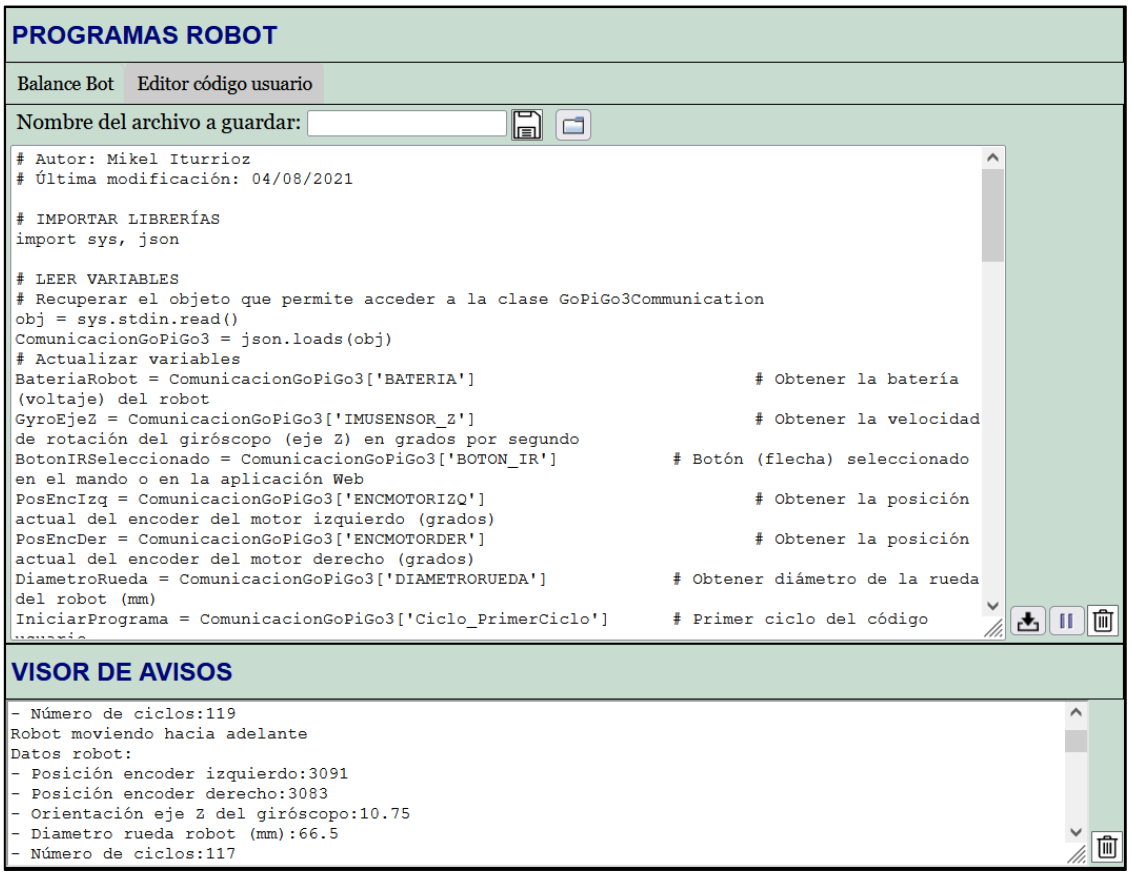


Figura 47: *Interfaz web. Código de ejemplo desarrollado por el usuario*

En el ejemplo del programa de control desarrollado por el usuario en vez de utilizar la API del robot GoPiGo3 [15] se utilizan las variables de la clase GoPiGo3Communication (ver *Tabla 7* del apartado *4.1.5 GoPiGo3Communication.py*) para enviar órdenes a los actuadores del robot o leer datos de los sensores que tiene implementados. La clase GoPiGo3Communication se ha creado para simplificar la tarea de programación del usuario, implementando de la siguiente manera.

En la clase GoPiGo3Operation se ejecuta la llamada a un subproceso con el programa de control desarrollado por el usuario (el programa de control del usuario se encuentra en el fichero *CodigoUsuario.py*). En la llamada al subproceso como parámetro de entrada (stdin) se envían las variables de la clase GoPiGo3Communication (ver *Código 4*).

```
# Ejecutar un subproceso. Programa de control (script) desarrollado por el usuario
                self.SubprocesoUsuario = subprocess.Popen(['python','-u',
'CodigoUsuario.py'],  # Abrir un subproceso
                    stdin = subprocess.PIPE,
                    stdout = subprocess.PIPE
                )
                # Interactuar con el proceso: enviar datos a stdin (clase
GoPiGo3Communication) y leer datos de stdout
                output =
self.SubprocesoUsuario.communicate(input=self.GoPiGo3Communication.toJSON().encode()
)[0]
```

Código 4: *Clase GoPiGo3Operation. Llamada al subproceso del programa de control desarrollado por el usuario*

En el programa de control desarrollado por el usuario lo primero que se hace es recuperar el objeto que permite acceder a la clase GoPiGo3Communication. En el *Código 5* se muestra el código que realiza dicha recuperación, donde el objeto recuperado se llama *ComunicacionGoPiGo3*.

```
# Recuperar el objeto que permite acceder a la clase GoPiGo3Communication
obj = sys.stdin.read()
ComunicacionGoPiGo3 = json.loads(obj)
```

Código 5: *Código del usuario. Recuperar el objeto de la clase GoPiGo3Communication*

Una vez recuperado el objeto de la clase GoPiGo3Communication, se podrán utilizar en el programa de control del usuario las variables mencionadas en la *Tabla 7* del apartado *4.1.5 GoPiGo3Communication.py*. Al final del programa de control del usuario hay que salvar y devolver los datos. Para devolver los datos a la clase GoPiGo3Operation se escribirá en el código del usuario la instrucción del *Código 6*.

```
# SALVAR DATOS EN EL CONTEXTO
ComunicacionGoPiGo3['POTENCIAMOTORIZQ'] = PotenciaMotorIzq # Potencia motor izquierdo
ComunicacionGoPiGo3['POTENCIAMOTORDER'] = PotenciaMotorDer # Potencia motor derecho
ComunicacionGoPiGo3['RESETENCODERS'] = ResetEncoders              #  Reset  encoders
(izquierdo y derecho)
ComunicacionGoPiGo3['MENSAJE'] = Mensaje                                  # Mensaje a
mostrar en el Data logger
ComunicacionGoPiGo3['Ciclo_PrimerCiclo'] = IniciarPrograma # Primer ciclo del código
usuario
VariableAuxiliar['NumCiclos'] = NumCiclosActual                           #   Guardar
el valor del número de ciclos para utilizar en la siguiente ejecución del programa

# DEVOLVER DATOS
print(json.dumps(ComunicacionGoPiGo3))
```

Código 6: *Código del usuario. Devolver los datos a la clase GoPiGo3Operation*

Por su parte, la clase GoPiGo3Operation leerá los datos devueltos (leer datos de stdout) y se actualizarán los valores de las variables o parámetros para el ciclo de trabajo del robot.

```
# Interactuar con el proceso: enviar datos a stdin (clase GoPiGo3Communication) y
leer datos de stdout
                output =
self.SubprocesoUsuario.communicate(input=self.GoPiGo3Communication.toJSON().encode()
)[0]
                # Deserializar cadenas nativas o bytes (datos de stdout) que constan
de datos JSON en Python Dictionary
                Communication = json.loads(output.decode('utf-8'))
                # Actualizar parámetros en el programa
                if self.Running and self.ModoControlRobot == "Automatico":  #
Conectado y en modo Automático
                    self.GoPiGo3Communication.POTENCIAMOTORIZQ =
Communication['POTENCIAMOTORIZQ']   # Potencia motor izquierdo
                    self.GoPiGo3Communication.POTENCIAMOTORDER =
Communication['POTENCIAMOTORDER']   # Potencia motor derecho
                    self.GoPiGo3Communication.RESETENCODERS =
Communication['RESETENCODERS']          # Reset encoders (motor izquierdo y derecho)
                    self.GoPiGo3Communication.msg = Communication['MENSAJE']
# Mensaje del programa de control (script) desarrollado por el usuario
                    self.AplicacionWeb_Msg = self.GoPiGo3Communication.msg
# Mensaje a mostrar en el Visor de Avisos de la aplicación Web
                    self.GoPiGo3Communication.Ciclo_PrimerCiclo =
Communication['Ciclo_PrimerCiclo'] # Primer ciclo del programa de control (script)
desarrollado por el usuario
                    self.GoPiGo3Communication.VariableAuxiliar =
Communication['VariableAuxiliar']   # Para almacenar cualquier valor interno en el
programa de control (script) desarrollado por el usuario (para tener accesible en
cada ejecución)
```

Código 7: *Clase GoPiGo3Operation. Leer datos devueltos del programa de control del usuario y actualizar variables*

Finalmente, dependiendo del código desarrollado en el programa de control del usuario se enviará un nuevo comando al robot. Por ejemplo, en el código de ejemplo que se ha desarrollado (mirar apartado *10.6 Código usuario ejemplo*) si se pulsa el botón adelante en el mando a distancia por infrarrojos el valor de las variables *PotenciaMotorIzq* y *PotenciaMotorDer* será: **PotenciaMotorIzq = 50%** y **PotenciaMotorDer = 50%**. Al salvar y devolver los datos en el código del usuario, en la clase GoPiGo3Operation se actualizará el valor de las variables *self.GoPiGo3Communication.POTENCIAMOTORIZQ* y *self.GoPiGo3Communication.POTENCIAMOTORDER*, siendo **self.GoPiGo3Communication.POTENCIAMOTORIZQ = 50** y **self.GoPiGo3Communication.POTENCIAMOTORDER = 50**. De este modo, mediante la instrucción del *Código 8* que se ha escrito en la lógica de ejecución del robot (clase GoPiGo3Operation), se enviará una orden a los motores del robot GoPiGo3 para que empiecen a girar con una potencia del 50%.

```
# Worker. Hilo de ejecucion (de este modo se permite que el programa ejecute
múltiples operaciones simultáneamente en el mismo espacio de proceso)
    def Worker(self):
        while(self.Running):     # Conectado y en ejecución
            self.LeerDatos()     # Leer datos del robot GoPiGo3 (batería, IMU sensor,
receptor IR, encoders y diámetro de la rueda)
            if self.ModoControlRobot == "Automatico" and
(self.StartProgramaBalanceBot or self.StartScriptUsuario):  # Modo de funcionamiento
Automático y programa en ejecución
                self.ControlProgramaRobot()  # Ejecutar el programa Balance Bot o el
programa de control (script) desarrollado por el usuario
                # Actualizar parámetros del robot GoPiGo3
                self.setPotenciaMotorIzq(self.GoPiGo3Communication.POTENCIAMOTORIZQ)
# Definir potencia motor izquierdo
                self.setPotenciaMotorDer(self.GoPiGo3Communication.POTENCIAMOTORDER)
# Definir potencia motor derecho
                if self.GoPiGo3Communication.RESETENCODERS:
                    self.ResetRobotEncoders(0)  # Resetear posición de los encoders
del motor izquierdo y derecho
                time.sleep(0.02)     # Esperar 0.02 segundos
            else:
                time.sleep(0.1)      # Esperar 0.1 segundos
```

Código 8: *Clase GoPiGo3Operation. Función Worker, actualizar potencia de los motores (modo automático)*

6. Conclusiones

Para la realización de este trabajo se ha utilizado el robot GoPiGo3 de Dexter Industries con el Kit BalanceBot [10]. Como punto de partida se ha tomado el servidor RIP [16] proporcionado por el equipo docente [30]. Para el control del robot se ha escrito un nuevo código en lenguaje de programación Python [12], el cual ha sido integrado con el servidor RIP que se ejecuta en la placa Raspberry Pi. Este nuevo código se utiliza para enviar órdenes a los actuadores o leer información de los sensores, es decir, para ejecutar el control del robot.

Mediante la herramienta EjsS [17] se ha creado una aplicación JavaScript que se puede visualizar usando cualquier navegador web. Esta aplicación informática utiliza el elemento RIP, implementando los métodos de protocolo de comunicación (POST, GET y SSE [28]) con el formato apropiado para leer y escribir el contenido de los mensajes (JSON-RPC [29]) y la estructura y funciones (connect, set y get) definidas por RIP.

De este modo, al tener dos entidades que hablan el mismo protocolo, Raspberry Pi como servidor RIP y la aplicación EjsS como cliente RIP, se consigue establecer comunicación entre ellos. Observando los resultados obtenidos se puede decir que la comunicación entre el servidor RIP y el cliente RIP se ha establecido correctamente mediante la red inalámbrica Wifi.

Haciendo uso de la interfaz web se puede controlar el robot en su totalidad. En modo de funcionamiento manual se pueden realizar todos los movimientos del robot GoPiGo3. Se pueden ajustar parámetros para el funcionamiento del robot, y, además, se visualiza información sobre los sensores implementados en el robot.

Cuando se trabaja en modo automático, existe la posibilidad de cargar distintos programas de control al robot GoPiGo3. Por un lado, se ha diseñado un controlador PID [33] que permite el correcto equilibrio del robot de balanceo. El usuario tiene que iniciar simplemente este código desarrollado [32] para conseguir mantener el robot en posición vertical, y una vez conseguido esto, podrá conducir el robot mediante el mando a distancia por infrarrojos o el panel de control (flechas) de la interfaz web. En la aplicación EjsS se han visualizado los valores de las constantes que influyen en el controlador diseñado o en el código de balanceo. De este modo, si hay problemas para equilibrar el robot el usuario podrá modificar el valor de dichas constantes en la aplicación informática EjsS y conseguir el objetivo de equilibrar el robot.

Por otra parte, en modo de funcionamiento automático el usuario puede descargar y ejecutar en el robot GoPiGo3 sus propios programas de control. Esto permite al usuario crear su propio código y realizar las pruebas con el robot. En este trabajo, como ejemplo, se ha implementado un código el cual modifica el valor de la potencia de los motores para realizar diferentes movimientos del robot. Además, se genera un mensaje con los datos obtenidos de los sensores del robot para mostrar en el visor de avisos de la interfaz web. A todo esto, hay que sumarle la ventaja de que el usuario puede modificar o leer en su programa de control parámetros como, por ejemplo, potencia de los motores, sensor IMU, receptor IR… sin necesidad de tener conocimientos sobre la API GoPiGo3 [15].

Aun así, en el desarrollo del proyecto han surgido algunos problemas:

- Si se realiza la lectura de los datos del robot a través de su API en continuo y muy rápido, surgen problemas de lectura (sobre todo con el sensor IMU). Para solucionar esto, en modo de funcionamiento manual se ha introducido un delay de 0.1 seg en cada ciclo de ejecución, y, en automático 0.02 seg. De este modo, se retrasa la lectura de los datos y se consigue un mejor funcionamiento.
- Se ha diseñado un controlador PID para el correcto equilibrio del robot de balanceo GoPiGo3. Este controlador hay que ejecutar con suficiente frecuencia y dentro del rango tolerable del sistema. En un principio, se planteó ejecutar el programa de balanceo utilizando la aplicación web EjsS, como si fuese un programa de control desarrollado por el usuario. Pero, tal y como se ha comentado en el apartado *5.2 Prueba 2. Modos de funcionamiento del ciclo de trabajo del robot*, en la clase GoPiGo3Operation para iniciar el programa de control desarrollado por el usuario se ejecuta un subproceso. Al hacer esto, el tiempo real del bucle del controlador PID aumenta mucho, y los resultados obtenidos no son nada buenos puesto que resulta muy difícil mantener el robot en equilibrio. Por ello, al final se ha planteado el programa de balanceo como una función más dentro de la clase GoPiGo3Operation, consiguiendo una mejor respuesta en cuanto al tiempo real del bucle y consiguiendo mantener el robot en equilibrio.
- En la aplicación EjsS si se utiliza el método get para obtener el valor de los parámetros del servidor RIP periódicamente, surgen problemas. En este caso, para evitar este problema el cliente RIP (aplicación informática desarrollada en EjsS) se suscribe a un evento de SSE [28] asociada al servidor RIP. De este modo se obtienen actualizaciones de datos de servidor a cliente periódicamente. El método devuelve una lista donde el primer elemento es una lista que contiene el nombre de las variables notificadas y el segundo elemento otra lista con los valores correspondientes.
- El receptor de infrarrojos utilizado en el proyecto recibe las señales a una distancia de hasta 10 metros. Si se quiere utilizar la aplicación EjsS para controlar el robot de forma remota, el uso del mando a distancia por infrarrojos no tiene mucho sentido, puesto que el receptor de infrarrojos no recibirá las señales emitidas a una distancia mayor a 10 metros. Por ello, cuando se está ejecutando el programa Balance Bot también se ha incluido la opción de conducir el robot mediante los botones o flechas de la interfaz web.

6.1 Líneas de trabajo futuro

A pesar de todo, el planteamiento realizado no está exenta de limitaciones y se pueden plantear mejoras mirando hacia el futuro:

- Sería interesante introducir una rampa de inclinación creciente por la que suba el robot hasta colocarse casi en posición vertical, y a partir de ahí mantener la posición mediante el controlador PID desarrollado en el proyecto. Como posible solución se podría colocar un apoyo de rodadura (una rueda loca) en el extremo contrario de donde están posicionadas las ruedas del robot. Además, la rampa debería de ser más estrecha que la separación entre las ruedas actuales del robot, para que el robot monte la rueda loca delantera en la rampa y vaya adquiriendo posición invertida sin que las ruedas traseras empiecen a subir también. Aunque no se haya implementado esta mejora en el proyecto, sí que se ha ejecutado un planteamiento inicial del mismo que requiere más

pruebas futuras para ponerlo en funcionamiento. A continuación, en la *Figura 48* se muestran unas imágenes de la rueda loca y la rampa de inclinación creciente.

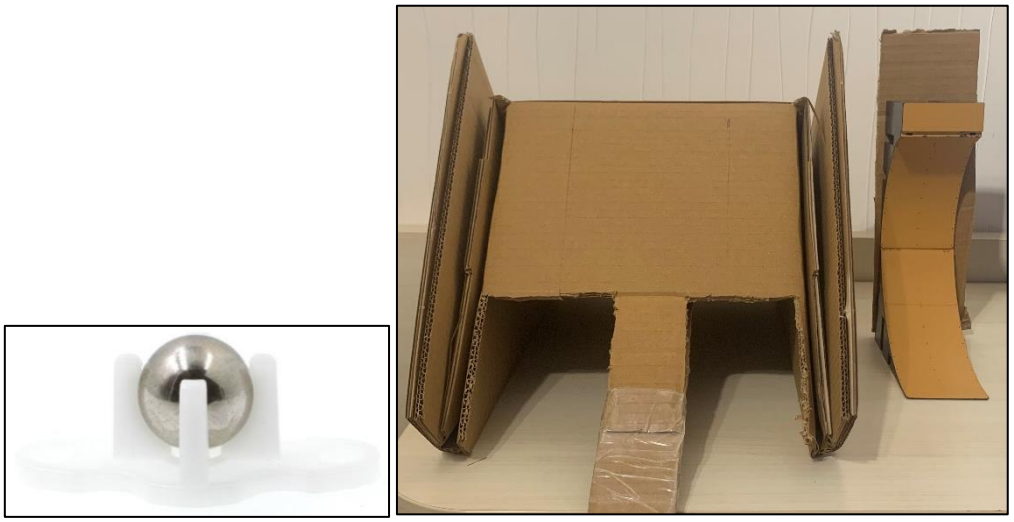


Figura 48: *Rueda loca y rampa de inclinación creciente*

7. Lista de referencias y bibliografía

- [1] Wikipedia. Inverted pendulum [En línea]. [Consulta: 02-07-2021]. Disponible en: https://en.wikipedia.org/wiki/Inverted_pendulum.
- [2] Dean Kamen. Segway [En línea]. [Consulta: 19-07-2021]. Disponible en: https://es-es.segway.com/.
- [3] Felix Grasser, Aldo D'Arrigo, Silvio Colombi, Alfred Rufer, "JOE: A mobile, inverted pendulum", *IEEE Transactions on Industrial Electronics*, pp. 107-114, Marzo 2002.
- [4] David P. Anderson (2003). Nbot, a two wheel balancing robot [En línea]. [Consulta: 07-08-2021]. Disponible en: http://www.geology.smu.edu/~dpa-www/robo/nbot/.
- [5] Wei An, Yangmin Li, "Simulation and control of a two-wheeled self-balancing robot", *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 456-461, Diciembre 2013.
- [6] N. M. Abdul Ghani, D. Ju, H. Z Othman, M. A. Ahmad, "Two wheels mobile robot using optimal regulator control", *2010 10th International Conference on Intelligent Systems Design and Applications*, pp. 1066-1070, Noviembre 2010.
- [7] Junfeng Wu, Wanying Zhang, "Design of fuzzy logic controller for two-wheeled self-balancing robot", *Proceedings of 2011 6th International Forum on Strategic Technology*, pp. 1266-1270, Agosto 2011.
- [8] S.W Nawawi, M.N Ahmad, J.H.S Osman, A.R Husain, M.F Abdollah, "Controller Design for Two-wheels Inverted Pendulum Mobile Robot Using PISMC", *2006 4th Student Conference on Research and Development*, pp. 194-199, Junio 2006.
- [9] M. Abdullah Bin Azhar, Waseem Hassan, Usman Rahim, "PID control behavior and sensor filtering for a self balancing personal vehicle", *2012 International Conference of Robotics and Artificial Intelligence*, pp. 7-10, Octubre 2012.
- [10] Dexter Industries. Dexter Industries [En línea]. [Consulta: 11-02-2021]. Disponible en: https://www.dexterindustries.com/.
- [11] Raspberry Pi. Teach, Learn, and Make with Raspberry Pi [En línea]. [Consulta: 13-07-2021]. Disponible en: https://www.raspberrypi.org/.
- [12] Python Software Foundation. The official home of the Python Programming Language [En línea]. [Consulta: 13-02-2021]. Disponible en: https://www.python.org/.
- [13] Dexter Industries. Source code for easygopigo3 [En línea]. [Consulta: 18-02-2021]. Disponible en: https://gopigo3.readthedocs.io/en/master/_modules/easygopigo3.html.
- [14] Dexter Industries. API - Library - EasyGoPiGo3 [En línea]. [Consulta: 18-02-2021]. Disponible en: https://gopigo3.readthedocs.io/en/master/api-basic/easygopigo3.html.
- [15] Microsoft. GoPiGo Class [En línea]. [Consulta: 18-02-2021]. Disponible en: https://docs.microsoft.com/en-us/dotnet/api/iot.device.gopigo3.gopigo?view=iot-dotnet-1.3.
- [16] Luis de la Torre, Jesús Chacón, Dictino Chaos (2019, Octubre 30). Rip-spec [En línea]. [Consulta: 02-03-2021]. Disponible en: https://github.com/UNEDLabs/rip-spec.
- [17] Francisco Esquembre, Félix Jesús García Clemente (2021, Febrero 25). Easy Java Simulations Wiki [En línea]. [Consulta: 02-03-2021]. Disponible en: https://www.um.es/fem/EjsWiki/.

- [18] Juan José Romero Marras, "Control de un robot basado en Raspberry", Trabajo Fin de Máster, Máster en Ingeniería de Sistemas y Control, UNED-UCM, Madrid, 2016-2017.
- [19] Jorge Durán Murillas, "Integración de un robot autónomo basado en Raspberry en el sistema de laboratorios remotos", Trabajo Fin de Máster, Máster en Ingeniería de Sistemas y Control, UNED-UCM, Madrid, 2018.
- [20] HETPRO. Raspberry PI 3 B+ Lección 1 - HETPRO/TUTORIALES [En línea]. [Consulta: 14-07-2021]. Disponible en: https://hetpro-store.com/TUTORIALES/raspberry-pi-3-b-plus/.
- [21] Dexter Industries. Install GoPiGo OS or Raspbian for Robots on an SD Card for the Raspberry Pi [En línea]. [Consulta: 15-02-2021]. Disponible en: https://www.dexterindustries.com/howto/install-raspbian-for-robots-image-on-an-sd-card/.
- [22] Génération Robots. GoPiGo3 BalanceBot Extension Kit [En línea]. [Consulta: 14-07-2021]. Disponible en: https://www.generationrobots.com/en/402775-gopigo3-balancebot-extension-kit.html.
- [23] Dexter Industries. Hardware and Port Description [En línea]. [Consulta: 11-02-2021]. Disponible en: https://www.dexterindustries.com/GoPiGo/learning/hardware-port-description/.
- [24] Dexter Industries. IMU Sensor [En línea]. [Consulta: 11-02-2021]. Disponible en: https://www.dexterindustries.com/store/imu-sensor/.
- [25] Dexter Industries. Grove Infrared Receiver GoPiGo.io [En línea]. [Consulta: 11-02-2021]. Disponible en: https://gopigo.io/grove-infrared-receiver/.
- [26] Dexter Industries. Remote Control GoPiGo.io [En línea]. [Consulta: 11-02-2021]. Disponible en: https://gopigo.io/remote-control/.
- [27] The Internet Society (1999). Http 1.1 specification. [En línea]. [Consulta: 03-08-2021]. Disponible en: https://datatracker.ietf.org/doc/html/rfc2616.html.
- [28] Ian Hickson (2009). Server-sent events specification. [En línea]. [Consulta: 03-08-2021]. Disponible en: https://www.w3.org/TR/eventsource/.
- [29] Matt Morley (2010). Json-rpc 2.0 specification. [En línea]. [Consulta: 03-08-2021]. Disponible en: https://www.jsonrpc.org/specification.
- [30] Luis de la Torre, Jesús Chacón (2020, Julio 19). Rip-python-server [En línea]. [Consulta: 02-03-2021]. Disponible en: https://github.com/UNEDLabs/rip-python-server.
- [31] Jesús Chacón, "USER GUIDE FOR THE RIP SERVER IN PYTHON", Departamento de Informática y Automática, UNED, Octubre 2019.
- [32] Matthew Richardson (2017, Octubre 16). GoPiGo3 BalanceBot.py [En línea]. [Consulta: 11-02-2021]. Disponible en: https://github.com/DexterInd/GoPiGo3/blob/master/Projects/BalanceBot/BalanceBot.py.
- [33] Lucas Alpi (2019, Diciembre 2). Control PID: rompiendo la barrera del tiempo [En línea]. [Consulta: 04-07-2021]. Disponible en: https://www.novusautomation.com/site/default.asp?Idioma=34&TroncoID=053663&SecaoID=0&SubsecaoID=0&Template=../artigosnoticias/user_exibir.asp&ID=638091.

- [34] Suhardi Azliy Bin Junoh, "TWO-WHEELED BALANCING ROBOT CONTROLLER DESIGNED USING PID", Trabajo Fin de Máster, Máster en Ingeniería Eléctrica, UTHM, Malaysia, 2015.
- [35] Shubhank Sondhia, Ranjith Pillai. R, Sharat S. Hegde, Sagar Chakole, Vatsal Vora, "DEVELOPMENT OF SELF BALANCING ROBOT WITH PID CONTROL", *International Journal of Robotics Research and Development (IJRRD)*, vol. 7, no. 1, pp. 1-6, Abril 2017.
- [36] Dexter Industries. Running the GoPiGo3 BalanceBot [En línea]. [Consulta: 11-02-2021]. Disponible en: https://www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/assemble-gopigo3-balancebot/running-gopigo3-balancebot/.
- [37] Félix Jesús García Clemente, Luis de la Torre, Jesús Chacón (2020, Agosto 26). EjsS tool [En línea]. [Consulta: 09-03-2021]. Disponible en: https://gitlab.com/ejsS/tool.
- [38] Luis de la Torre, "USER GUIDE FOR THE REMOTE INTEROPERABILITY PROTOCOL ELEMENT IN EJSS", Departamento de Informática y Automática, UNED, Junio 2019.
- [39] Dexter Industries. Assemble the GoPiGo3 BalanceBot [En línea]. [Consulta: 13-02-2021]. Disponible en: https://www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/assemble-gopigo3-balancebot/.
- [40] Dexter Industries. Connect to the GoPiGo3 Robot [En línea]. [Consulta: 11-02-2021]. Disponible en: https://www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/2-connect-to-the-gopigo-3/.
- [41] CherryPy Team (2014). Basics — CherryPy 3.2.4 documentation. [En línea]. [Consulta: 06-08-2021]. Disponible en: https://cherrypydocrework.readthedocs.io/basics.html.
- [42] Damien P. George, Paul Sokolovsky (2014). ujson – JSON encoding and decoding. [En línea]. [Consulta: 16-07-2021]. Disponible en: https://docs.micropython.org/en/latest/library/ujson.html.

8. Listado de siglas, abreviaturas y acrónimos

°	Degrees o Grados
°/s	Degrees/Seconds o Grados/Segundos
seg	Seconds o Segundos
%	Percentage o Porcentaje
mm	Millimeters o Milímetros
m	Meter o Metro
g	Gram o Gramo
°C	Degree Celsius o Grado Celsius
V	Volts o Voltios
V_{cc}	Volts of Direct Current (V_{DC}) o Voltios en Corriente Continua
CC	Direct Current (DC) o Corriente Continua
A	Ampere o Amperio
Kp	Proportional Gain o Término Proporcional
Ki	Integral Gain o Término Integral
Kd	Derivative Gain o Término Derivado
DPS	Degrees per second of the robot's wheel o Grados por segundo de la rueda del robot
KHz	KiloHertz
GHz	GigaHertz
Mbps	Megabits per second o Megabits por segundo
GB	Gigabyte
TFM	Trabajo Fin de Máster
UNED	Universidad Nacional de Educación a Distancia
IP	Inverted Pendulum o Péndulo Invertido
PT	Personal Transporter o Transporte Personal
API	Application Programming Interface o Interfaz de Programación de Aplicaciones
RIP	Remote Interoperability Protocol o Protocolo de Interoperabilidad Remota
EjsS	Easy Java/Javascript Simulations
IMU	Inertial Measurement Unit o Unidad de Medición Inercial
IR receiver	Infrared Receiver o Receptor de Infrarrojos
IR Remote Control	Infrared Remote Control o Control Remoto de Infrarrojos
LQR	Linear Quadratic Regulator o Regulador Cuadrático Lineal
PPC	Pole Placement Controller o Controlador por Asignación de Polos
FLC	Fuzzy Logic Controller o Controlador Lógico Difuso
SMC	Sliding Mode Controller o Controlador de Modo Deslizante
PISMC	Proportional Integral Sliding Mode Controller o Controlador de Modo Deslizante Integral Proporcional
PID	Proportional-Integral-Derivative controller o controlador Proporcional-Integral-Derivativo

LED	Light Emitting Diode o Diodo Emisor de Luz
DOF	Degree Of Freedom o Grados de Libertad
GNU	General Public License o Licencia Pública General
OL	Online Laboratories o Laboratorios Online
HTTP	Hypertext Transfer Protocol o Protocolo de Transferencia de Hipertexto
TCP	Transmission Control Protocol o Protocolo de Control de Transmisión
URL	Uniform Resource Locator o Localizador Uniforme de Recursos
JSON	JavaScript Object Notation o Notación de Objeto de JavaScript
RPC	Remote Procedure Call o Llamada a Procedimiento Remoto
SSE	Server Sent Events o Evento Enviado por el Servidor
HTML	HyperText Markup Language o Lenguaje de Marcado de Hipertexto
JAR	Java Archive
Wifi	Wireless Fidelity
IP	Internet Protocol o Protocolo de Internet
PC	Personal Computer o Computadora Personal
SO	Operating System o Sistema Operativo
SD	Secure Digital
VNC	Virtual Network Computing o Computación Virtual en Red
RAM	Random Access Memory o Memoria de Acceso Aleatorio
IEEE	Institute of Electrical and Electronics Engineers o Instituto de Ingenieros Eléctricos y Electrónicos
LAN	Local Area Network o Red de Área Local
BLE	Bluetooth Low Energy o Bluetooth de Baja Energía
USB	Universal Serial Bus o Bus Universal en Serie
GPIO	General Purpose Input/Output o Entrada/Salida de Propósito General
HDMI	High Definition Multimedia Interface o Interfaz Multimedia de Alta Definición
MIPI	Mobile Industry Processor Interface o Interfaz de Procesador de la Industria Móvil
DSI	Display Serial Interface o Interfaz Serie de Pantalla
CSI	Camera Serial Interface o Interfaz Serie para Cámaras
MPEG	Moving Picture Experts Group
PoE	Power over Ethernet o Energía sobre Ethernet

9. Anexo A

9.1 Detalles de los elementos del hardware

En este apartado se muestran los detalles de los elementos utilizados en el desarrollo del proyecto.

9.1.1 Detalles de la placa Raspberry Pi

Las especificaciones de la Raspberry Pi 3 Model B + (ver *Figura 5* del apartado *3.1.1 Placa Raspberry Pi*) son los siguientes:

- Procesador: Broadcom BCM2837B0, Cortex-A53 SoC de 64 bits a 1,4 GHz.
- Memoria: 1GB LPDDR2 SDRAM.
- Conectividad:
  - 2,4 GHz y 5 GHz IEEE 802.11.b/g/n/ac inalámbrico.
  - LAN.
  - Bluetooth 4.2, BLE.
  - Gigabit Ethernet sobre USB 2.0 (rendimiento máximo 300 Mbps).
- Acceso: 4 puertos USB 2.0.
- Encabezado GPIO de 40 pines.
- Video y sonido:
  - 1 × HDMI de tamaño completo.
  - Puerto de visualización MIPI DSI.
  - Puerto de cámara MIPI CSI.
- Multimedia:
  - Salida estéreo de 4 polos y puerto de video compuesto.
  - Decodificación H.264, MPEG-4 (1080p30); Codificación H.264.
  - (1080p30); OpenGL ES 1.1, 2.0 gráficos.
- Soporte de tarjeta SD: formato microSD para cargar el sistema operativo y almacenamiento de datos.
- Potencia de entrada:
  - 5V / 2.5A CC a través del conector micro USB.
  - 5V CC a través del encabezado GPIO.
  - Energía sobre Ethernet (PoE) habilitada (requiere por separado PoE HAT).
- Medio ambiente: temperatura de funcionamiento, 0-50 °C.

Raspbian, una distribución derivada de Debian que está optimizada para el hardware de Raspberry Pi, se lanzó durante julio de 2012 y es la distribución recomendada por la fundación para iniciarse. Esta distribución contiene herramientas de desarrollo para diferentes IDLE, entre otros los lenguajes de programación Python y Scratch. Aun así, permite usar otros sistemas operativos como GNU/Linux o Windows 10.

Dexter Industries utiliza el software Raspbian for Robots para su robot GoPiGo3. Este software viene en una tarjeta microSD (ver *Figura 49*), la cual se inserta en la Raspberry Pi. Esto facilita la conexión al robot, conectando el robot a la misma red Wifi que la computadora del usuario.

Figura 49: *Tarjeta microSD*

Tal y como se ha mencionado en las especificaciones, el modelo Raspberry Pi 3 dispone de Wifi integrado. Por lo tanto, para controlar el robot de forma remota no ha sido necesario, durante el desarrollo de este trabajo, utilizar el Mini Wifi Dongle. Mediante la red Wifi se consigue establecer conexión entre la interfaz web EjsS y el servidor RIP que se está ejecutando en la placa Raspberry Pi.

9.1.2 Detalles de la placa Dexter Industries GoPiGo3

A continuación, se muestran las especificaciones de la placa Dexter Industries GoPiGo3 (ver *Figura 7* y *Figura 8* del apartado *3.1.2.1 Placa Dexter Industries GoPiGo3*):

- **Conexiones de sensores**: GoPiGo3 tiene 5 puertos Grove en los que se pueden conectar diferentes sensores de una manera bastante fácil. Grove es un sistema de interconexión estandarizado y modular para sistemas de prototipado llevando un enfoque de bloques al ensamblaje electrónico.
    - **Puertos I2C**: tiene dos puertos de sensor I2C. Estos se conectan directamente a la placa Raspberry Pi a través de un chip de conversión de nivel y se conectan al mismo bus. En este caso no se utilizan estos puertos.
    - **Puertos de sensor analógico/digital**: en estos dos puertos Grove se pueden conectar dispositivos analógicos y digitales. Están conectados directamente al microcontrolador en la placa de circuito GoPiGo3. En el primer puerto se conecta un sensor IMU (mirar apartado *3.1.2.2 Sensor IMU*), mientras que en el segundo puerto se conecta un receptor de infrarrojos (mirar apartado *3.1.2.3 Receptor de infrarrojos*).
    - **Puertos seriales**: un puerto Grove conectado directamente a pines seriales en la Raspberry Pi a través de un chip de conversión de nivel. Este puerto no se utiliza.
- **Conexiones de servo**: GoPiGo3 puede controlar dos servos. Están controlados por el microcontrolador en la placa de circuito GoPiGo3. En este caso no se utilizan.
- **Conexiones de motores**: una conexión para el motor izquierdo y otra para el derecho.
- **Microcontrolador:** el microcontrolador realiza todo el control de los motores y las E/S de los sensores en el GoPiGo3. Además, GoPiGo3 le permite a la Raspberry Pi acceder directamente a algunos sensores Grove. La Raspberry Pi tiene un bus I2C mediante el cual se puede acceder directamente a los sensores conectados a los puertos I2C.
- **Puerto de alimentación**: el puerto de alimentación de la GoPiGo3 es un conector de 5,5 mm x 2,1 mm que puede recibir una potencia positiva central de 7-12 V_{CC}.

9.1.3 Detalles del sensor IMU

Las especificaciones técnicas del sensor IMU (ver *Figura 9* del apartado *3.1.2.2 Sensor IMU*) son los siguientes:

- Está equipado con 3 instrumentos de navegación: una brújula, un acelerómetro y un giroscopio. Los 3 ofrecen un rango de medición de 3 ejes.

El sensor IMU tiene 9 grados de libertad (9-DOF) y proporciona una orientación precisa y de alta velocidad del robot. Puede usarse para saber en qué dirección de la brújula gira el robot, qué tan rápido está girando, qué ángulo está con el suelo y qué tan rápido está cambiando de velocidad. Es decir, obtiene la dirección del robot, su velocidad de rotación o su aceleración.

9.1.4 Detalles del receptor de infrarrojos

A continuación, se muestran las especificaciones técnicas del receptor de infrarrojos (ver *Figura 10* del apartado *3.1.2.3 Receptor de infrarrojos*):

- Voltaje: 3.3 a 5 V.
- Frecuencia de recepción: 38 kHz.
- Alcance: máximo 10 m (por encima de esta distancia, la señal no se recibirá correctamente).
- Dimensiones: 20 × 20 × 10 mm.
- Peso bruto: 6 g.

Y las especificaciones técnicas del control remoto de infrarrojos (ver *Figura 11* del apartado *3.1.2.3 Receptor de infrarrojos*):

- Incluye un teclado numérico de 10 dígitos, teclas * y #, y teclas de dirección (izquierda, derecha, arriba, abajo, ok).
- Necesita un receptor de infrarrojos para usar el control remoto.
- Alcance: máximo 10 m.
- Dimensiones: 86,3 × 60,9 × 10,1 mm.
- Peso bruto: 453 g.

9.2 Ensamblaje del robot GoPiGo3

En la página web de Dexter Industries se pueden encontrar las instrucciones para ensamblar el robot GoPiGo3 BalanceBot [39].

9.3 Conexión con el robot GoPiGo3

En este apartado se explicará como configurar la conexión con el robot GoPiGo3 [40].

Para empezar, se conectará la placa Raspberry Pi directamente a la computadora (PC) del usuario con un cable Ethernet. Al asegurar que la tarjeta microSD está colocada correctamente, se conectará el paquete de baterías y se encenderá la placa Raspberry Pi. Se utilizará el programa VNC® Viewer para acceder a Raspberry Pi. VNC (Virtual Network Computing o Computación Virtual en Red) es un programa de software libre basado en una estructura cliente-servidor que permite observar las acciones del ordenador servidor remotamente a través de un ordenador

cliente. Para acceder, se escribirá el usuario *dex.local:1* y la contraseña *robots1234* (ver *Figura 50*). El puerto de ethernet de la placa Raspberry Pi está configurado con la dirección IP 192.168.20.2. Conviene configurar el PC del usuario en el mismo rango, para no tener problemas en la conexión.

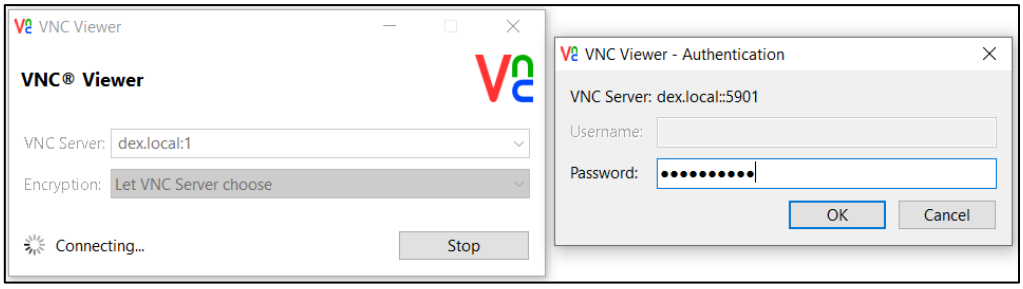


Figura 50: *Acceso al robot mediante VNC® Viewer*

Raspberry Pi ejecuta Raspberry Pi OS, una versión de un sistema operativo (SO) llamado Linux. Después de que se inicie el sistema operativo Raspberry Pi, aparecerá el escritorio (ver *Figura 51*). El modelo Raspberry Pi 3 se puede conectar a una red inalámbrica Wifi para conectar a Internet. De este modo, se podrá programar el robot GoPiGo3 de forma remota. Además, hay que estar conectado a Internet a través de Wifi para ejecutar cualquier actualización de software.

Figura 51: *Escritorio Raspberry Pi*

Para conectar Raspberry Pi a una red Wifi hay que seguir los siguientes pasos:

1. Hacer clic en el icono de red inalámbrica ( ) en la esquina superior derecha de la pantalla y seleccionar la red en el menú desplegable. Se conectará el robot a la misma red Wifi que se esté usando en la computadora del usuario (en este caso *Euskaltel-CDDY*), para programar el robot GoPiGo3 remotamente (ver *Figura 52*). Al seleccionar la red Wifi aparecerá un cuadro de diálogo donde hay que introducir la contraseña de la red inalámbrica (ver *Figura 53*).

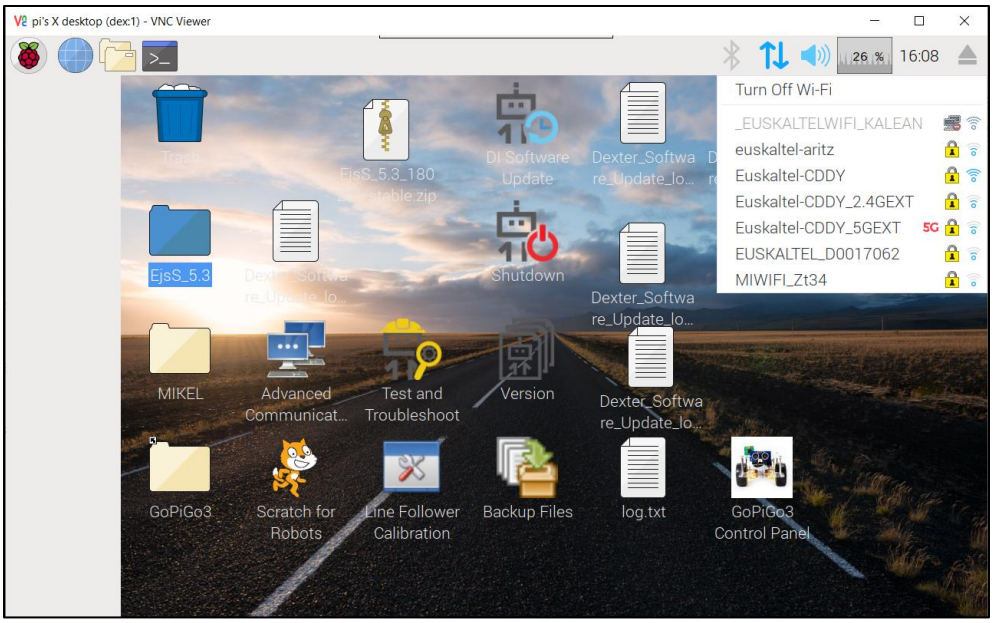


Figura 52: *Raspberry Pi, lista de redes disponibles*

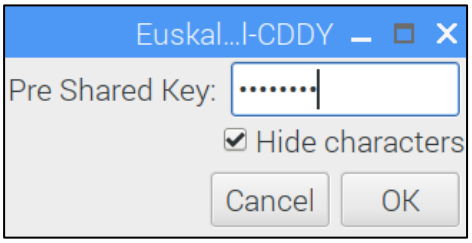


Figura 53: *Raspberry Pi, configuración con red Euskaltel-CDDY*

2. Una vez que Raspberry Pi esté conectada a Internet, es importante observar la dirección IP donde se aloja el servidor RIP (ver *Figura 54*). Esta dirección se utiliza en el apartado *4.2.1 Añadir, configurar y utilizar el elemento RIP* para realizar la configuración del cliente RIP.

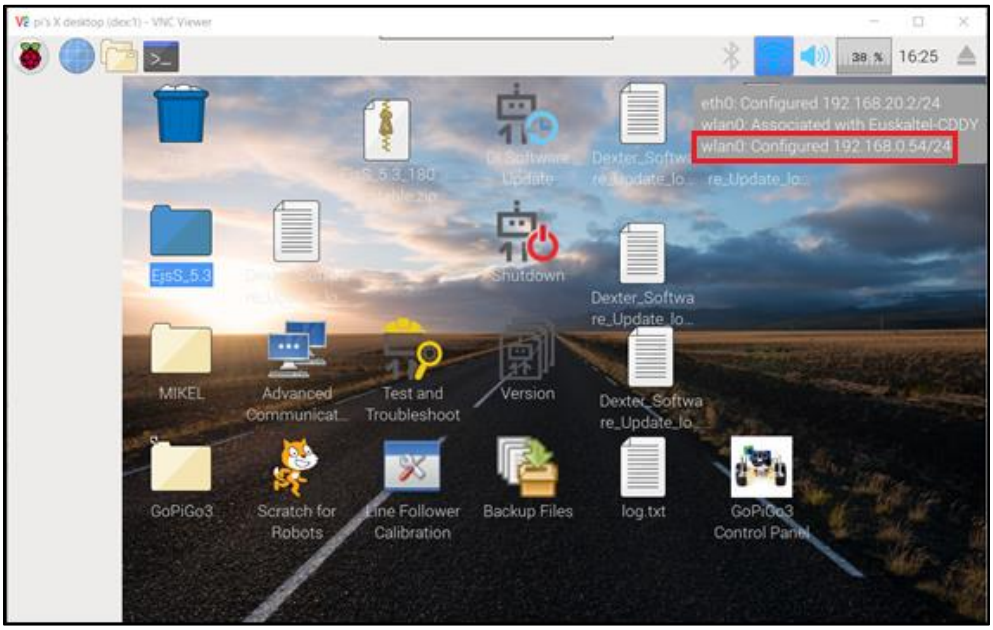


Figura 54: *Raspberry Pi, dirección IP del servidor RIP*

3. Después de conectarse a la red Wifi, se verifica que el robot está realmente conectado a Internet. Abrir el terminal de comandos en el robot y hacer un ping a la dirección *dexterindustries.com*. Si está conectado a Internet, se debería de ver algo parecido a la *Figura 55*.

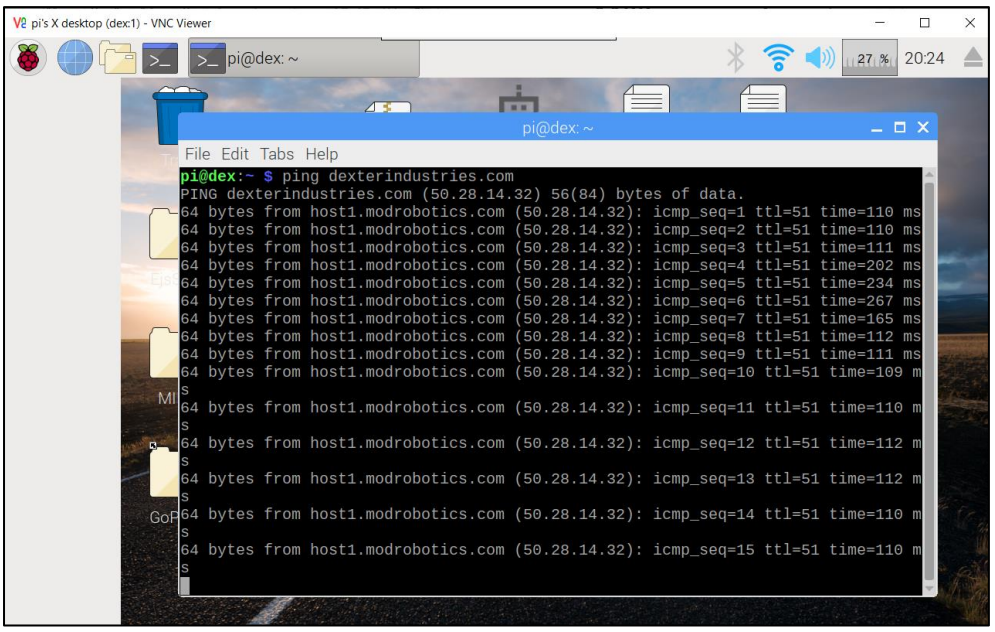


Figura 55: *Raspberry Pi, verificación de conexión del robot a Internet*

Finalmente, se verifica que el PC del usuario y la placa Raspberry Pi están conectados a la misma red (*Euskaltel-CDDY* en este caso). Para ello, apagar, retirar el cable de Ethernet y volver a encender el robot. Abrir el terminal de comandos en el PC del usuario y hacer un ping a la dirección *dex.local*. Si la conexión está funcionando correctamente, se debería de recibir una respuesta parecida a la *Figura 56*. De aquí en adelante, cada vez que se inicie el robot se conectará automáticamente a la red Wifi que se ha seleccionado en la configuración (*Euskaltel-CDDY*).

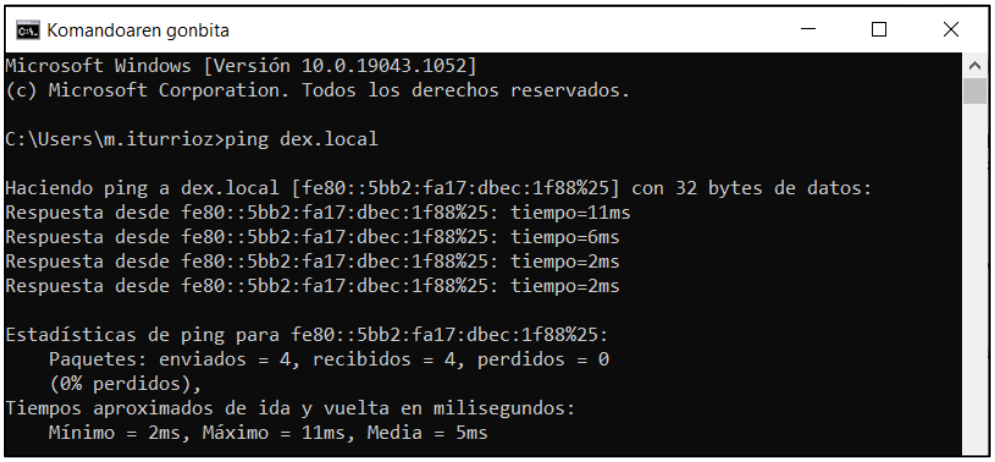


Figura 56: *Verificación de conexión entre PC usuario y Raspberry Pi (robot)*

9.4 Instalar paquetes y librerías para el servidor RIP

Para utilizar el servidor RIP en el robot GoPiGo3 es necesario instalar algunas librerías de Python [31], puesto que el software Raspbian for Robots de Dexter Industries no tiene instaladas por

defecto. El servidor RIP depende de las librerías de terceros cherrypy y ujson. Para instalar estas dependencias se utilizará el terminal de comandos de Raspberry Pi OS, escribiendo el siguiente comando (se considera ventajoso utilizar un entorno virtual a la hora de instalar librerías en la placa Raspberry Pi):

sudo pip install cherrypy ujson

Cherrypy [41] es un framework Python que está orientado a objetos y diseñado para un rápido desarrollo de aplicaciones web, envolviendo el protocolo HTTP [27]. Por otro lado, el módulo ujson [42] permite convertir entre objetos Python y el formato de datos JSON. Todas estas librerías se necesitan para utilizar el protocolo JSON-RPC [29] que se basa el protocolo RIP.

Además, hay que asegurarse de que se está utilizando la versión correcta de Python [12], es decir, la versión 3 de Python. En caso contrario, se instalará dicha versión mediante el comando:

sudo apt-get install Python

Una vez instalado todo lo necesario en Raspberry Pi, se podrá iniciar el servidor RIP. El terminal de comandos permite navegar por directorios de archivos y controlar la Raspberry Pi usando comandos escritos. Por defecto, el directorio de archivos al que accede el terminal cuando se abre es el llamado pi. Escribir el comando para cambiar el directorio a la ruta donde se encuentra el servidor RIP, y ejecutar el archivo App.py.

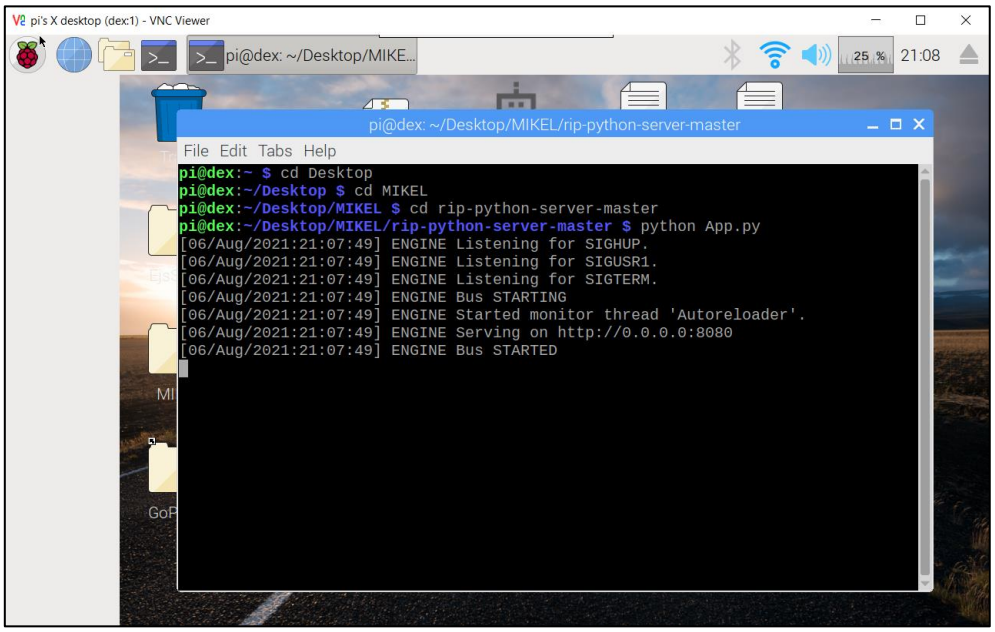


Figura 57: *Raspberry Pi, iniciar servidor RIP*

10. Anexo B

La API del servidor RIP ha sido proporcionada por el equipo docente [30]. En este anexo se muestra el código realizado como extensión del servidor RIP, que sirve para establecer comunicación con la interfaz web EjsS y ejecutar el ciclo de trabajo del robot GoPiGo3.

10.1 Clase App.py

```
import importlib
from HttpServer import HttpServer
from AppConfig import config

def load_control(control):
  module_name = 'rip.%s' % control['impl_module']
  module = importlib.import_module(module_name)
  control_name = control.get('impl_name', control['impl_module'])
  RIPControl = getattr(module, control_name)

  info = config['control']['info']
  return RIPControl(info)

if __name__ == "__main__":
  control = load_control(config['control'])

  HttpServer(
    host=config['server']['host'],
    port=config['server']['port'],
    control=control
  ).start(enable_ssl=False)
```

10.2 Clase AppConfig.py

```
# Autor: Mikel Iturrioz
# Última modificación: 04/08/2021

# This file contains the configuration of the RIP server application.
config = {
  # TO DO: The server will listen to host:port
  'server': {
    'host': '127.0.0.1',
    'port': 8080,
  },
  # The 'control' section configures the mapping between the RIP protocol
  # and the actual implementation of the functionality.
  # The 'impl' field should contain the name of the module (.py) and the
  # class that implement the control interface
  'control': {
    'impl_module': 'RIPGoPiGo3',  # Contiene el nombre del módulo que implementa
el acceso al hardware. Se ha definido un nuevo módulo de control
    # Also, if the class name is not the same as the module name:
    #'impl_name': 'RIPOctave',
    'info': { # Contiene la definición de la experiencia
      'name': 'RIP GiPiGo3',
      'description': 'An implementation of RIP to control a GoPiGo3 session',
      'authors': 'M. Iturrioz',
      'keywords': 'Raspberry Pi GopiGo3, RIP',
      # Server readable objects
      'readables': [],
      # Server writable objects
      'writables': []
    },
  }
}
```

10.3 Clase RIPGoPiGo3.py

```
# Autor: Mikel Iturrioz
# Última modificación: 04/08/2021

from rip.RIPGeneric import RIPGeneric
from GoPiGo3Operation import GoPiGo3Operation

class RIPGoPiGo3(RIPGeneric):
    '''
    RIP Server - Reference Implementation
    '''

    def __init__(self, info={}):
        '''
        Constructor
        '''
        super().__init__()
        self.GoPiGo3Operation=GoPiGo3Operation()

    def default_info(self):
        return {
            'name':'RIP GoPiGo3',
            'description': 'An implementation of RIP to control a GoPiGo3
session',
            'authors': 'M. Iturrioz',
            'keywords': 'Raspberry PI GopiGo 3, RIP',
            # Server readable objects
            'readables': [{
                'name':'InfoVisorAvisos',
                'description':'Mensajes o logs generados en el servidor RIP del
robot GoPiGo3',
                'type':'string',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'BateriaRobot',
                'description':'Voltaje batería del robot GoPiGo3',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'SensorIMUX',
                'description':'Valor del sensor giroscópico IMU. Velocidad
angular del eje X (grados/s)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'SensorIMUY',
                'description':'Valor del sensor giroscópico IMU. Velocidad
angular del eje Y (grados/s)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'SensorIMUZ',
                'description':'Valor del sensor giroscópico IMU. Velocidad
angular del eje Z (grados/s)',
```

```
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'BotonIRSeleccionado',
                'description':'Flecha seleccionada en el Control Remoto (para
el programa Balance Bot)',
                'type':'string',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'EncMotorIzq',
                'description':'Posición del encoder izquierdo (grados)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'VueltasEncMotorIzq',
                'description':'Número de vueltas del encoder izquierdo',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'GradosEncMotorIzq',
                'description':'Grados de la última vuelta del encoder izquierdo
(grados)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'EncMotorDer',
                'description':'Posición del encoder derecho (grados)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'VueltasEncMotorDer',
                'description':'Número de vueltas del encoder derecho',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'GradosEncMotorDer',
                'description':'Grados de la última vuelta del encoder derecho
(grados)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'DiametroRuedaRobot',
                'description':'Diámetro de la rueda del robot GoPiGo3 (mm)',
                'type':'int',
```

```
                'min':'0',
                'max':'Inf',
                'precision':'0'
                },
                {
                'name':'VelocidadActualManual',
                'description':'Velocidad actual del robot GoPiGo3 para el modo
de funcionamiento Manual en DPS (grados por segundo de la rueda del robot)',
                'type':'int',
                'min':'0',
                'max':'Inf',
                'precision':'0'
                }],
            # Server writable objects
            'writables': [],
        }

    # REST API
    def set(self, expid, variables, values):
        '''
        Sends one or more variables to the GoPiGo3 workspace
        '''
        return self.GoPiGo3Operation.set(variables,values)

    def get(self, expid, variables):
        '''
        Retrieve one or more variables from the GoPiGo3 workspace
        '''
        return self.GoPiGo3Operation.get(variables)

    def getValuesToNotify(self):
        '''
        Which values will be notified
        '''
        return [
            [ 'InfoVisorAvisos', 'BateriaRobot', 'SensorIMUX', 'SensorIMUY',
'SensorIMUZ', 'BotonIRSeleccionado', 'EncMotorIzq', 'VueltasEncMotorIzq',
'GradosEncMotorIzq', 'EncMotorDer', 'VueltasEncMotorDer', 'GradosEncMotorDer',
'DiametroRuedaRobot', 'VelocidadActualManual' ],
            [                          self.GoPiGo3Operation.AplicacionWeb_Msg,
self.GoPiGo3Operation.AplicacionWeb_BATERIA,
self.GoPiGo3Operation.AplicacionWeb_IMUSENSOR_X,
self.GoPiGo3Operation.AplicacionWeb_IMUSENSOR_Y,
self.GoPiGo3Operation.AplicacionWeb_IMUSENSOR_Z,
self.GoPiGo3Operation.AplicacionWeb_BOTON_IR_SELECCIONADO,
self.GoPiGo3Operation.AplicacionWeb_ENCMOTORIZQ,
self.GoPiGo3Operation.AplicacionWeb_VUELTASENCMOTORIZQ,
self.GoPiGo3Operation.AplicacionWeb_GRADOSENCMOTORIZQ,
self.GoPiGo3Operation.AplicacionWeb_ENCMOTORDER,
self.GoPiGo3Operation.AplicacionWeb_VUELTASENCMOTORDER,
self.GoPiGo3Operation.AplicacionWeb_GRADOSENCMOTORDER,
self.GoPiGo3Operation.AplicacionWeb_DIAMETRORUEDA,
self.GoPiGo3Operation.AplicacionWeb_VELOCIDADACTUALMANUAL ]
        ]
```

10.4 Clase GoPiGo3Operation.py

```
# Autor: Mikel Iturrioz
# Última modificación: 04/08/2021

# Importar librerias
import sys              # Proporciona acceso a algunas variables utilizadas o
mantenidas por el intérprete y a funciones que interactúan con el intérprete
import time             # Proporciona varias funciones relacionadas con el
tiempoimport subprocess
import json         # Es un formato ligero de intercambio de datos inspirado por
la sintaxis literal de objetos en JavaScript
```

```
import threading    # Construye interfaces de subprocesamiento de nivel superior sobre el módulo thread de nivel inferior.
import subprocess   # Permite lanzar nuevos procesos, conectarse a sus pipes de entrada/salida/error y obtener sus códigos de resultado
import gopigo3      # Importar los controladores GoPiGo3
from di_sensors.inertial_measurement_unit import InertialMeasurementUnit     # Importar los controladores IMU
from GoPiGo3Communication import GoPiGo3Communication                        # Importar la clase GoPiGo3Communication

# Clase GoPiGo3Operation
class GoPiGo3Operation(object):
    def __init__(self):
        # Inicialización
        self.GPG = gopigo3.GoPiGo3()                        # Crear una instancia de la clase GoPiGo3. self.GPG será el objeto de GoPiGo3
        self.GoPiGo3Communication = GoPiGo3Communication()      # Crear una instancia de la clase GoPiGo3Communication
        self.Imu = InertialMeasurementUnit(bus = "GPG3_AD1") # Definir a qué bus I2C está conectada la IMU
        self.Port_Sensor_IR = self.GPG.GROVE_2                  # Definir a qué puerto está conectado el receptor IR
        self.GPG.set_grove_type(self.Port_Sensor_IR, self.GPG.GROVE_TYPE.IR_DI_REMOTE)      # Configurar el receptor IR
        self.Running = False                    # No conectado (no en ejecución)
        self.ModoControlRobot = "Manual"            # Modo de funcionamiento predeterminado: Manual
        self.ManualSpeed = 300                  # Velocidad predeterminada para modo de funcionamiento Manual: 300 DPS
        self.AplicacionWeb_VELOCIDADACTUALMANUAL = self.ManualSpeed # Velocidad predeterminada para la aplicación Web
        self.ControlRemotoIR = False              # Control Remoto receptor IR desactivado
        self.BotonControlWebIR = 0              # Ningún botón seleccionado para el Control Web IR
        self.StartProgramaBalanceBot = False    # Parar la ejecución del programa Balance Bot
        self.GoPiGo3Communication.Ciclo_PrimerCiclo = False # Primer ciclo del programa Balance Bot o programa de control (script) desarrollado por el usuario
        self.StartScriptUsuario = False         # Parar la ejecución del programa de control (script) desarrollado por el usuario
        self.SubprocesoUsuario = -1             # Programa de control (script) desarrollado por el usuario no en ejecución
        # Constantes utilizadas en el programa Balance Bot:
        self.KGYROANGLE = 15    # Ángulo general
        self.KGYROSPEED = 1.7   # Cambios repentinos de ángulo
        self.KPOS = 0.07        # Desviación de la posición objetivo
        self.KSPEED = 0.1       # Velocidad actual (real) del motor
        self.KDRIVE = -0.02     # Objetivo de velocidad de conducción (ayuda al robot a iniciar/detener la conducción)
        self.KSTEER = 0.25      # Error en la dirección del robot
        self.KVOLTAGE = 0.083   # Cambios de voltaje. Configurada para 12V. Esta constante ayuda al robot a adaptarse a las baterías con voltaje descendente
        self.DRIVESPEED = 175    # Velocidad de conducción cuando se controla mediante el Control Remoto IR (grados por segundo)
        self.STEERSPEED = 100    # Velocidad de dirección cuando se controla mediante el Control Remoto IR (grados por segundo)
        self.TIMEFALLLIMIT = 2  # Si los motores han estado funcionando a plena potencia durante 2 segundos, suponga que el robot se cayó
        # Métodos
        self.Mapping={
        # Connection Methods (Get methods)
            'connect':self.connect,         # Conectar
            'disconnect':self.disconnect,   # Desconectar
        # Set Methods
            'setModoControl':self.ModoControl,             # Selecciona el modo de funcionamiento (manual/automático) del robot GoPiGo3
```

```
            'setRobotVeloManual':self.setRobotVeloManual,      # Establecer la
velocidad manual del robot GoPiGo3
            'setMovimientoManual':self.MovimientoManual,           #   Ejecutar
movimiento manual del robot GoPiGo3
            'setResetRobotEncoders':self.ResetRobotEncoders,# Resetear posición
de los encoders del motor izquierdo y derecho
            'setPotenciaMotorIzq':self.setPotenciaMotorIzq,    #    Definir    la
potencia del motor izquierdo del robot GoPiGo3
            'setPotenciaMotorDer':self.setPotenciaMotorDer,    #    Definir    la
potencia del motor derecho del robot GoPiGo3
            'setControlRemotoIR':self.setControlRemotoIR,                      #
Activar/Desactivar Control Remoto IR
            'setMovControlWebIR':self.setMovControlWebIR,    # Guarda el botón
seleccionado en la interfaz Web (para el programa Balance Bot)
            'setStartBalanceBot':self.StartBalanceBot,       # Iniciar programa
Balance Bot
            'setStopBalanceBot':self.StopBalanceBot,          # Parar ejecución
del programa Balance Bot
            'setKGYROANGLE':self.setKGYROANGLE,              # Definir constante
KGYROANGLE
            'setKGYROSPEED':self.setKGYROSPEED,              # Definir constante
KGYROSPEED
            'setKPOS':self.setKPOS,                          # Definir constante
KPOS
            'setKSPEED':self.setKSPEED,                      # Definir constante
KSPEED
            'setKDRIVE':self.setKDRIVE,                      # Definir constante
KDRIVE
            'setKSTEER':self.setKSTEER,                      # Definir constante
KSTEER
            'setKVOLTAGE':self.setKVOLTAGE,                  # Definir constante
KVOLTAGE
            'setDRIVESPEED':self.setDRIVESPEED,              # Definir constante
DRIVESPEED
            'setSTEERSPEED':self.setSTEERSPEED,              # Definir constante
STEERSPEED
            'setTIMEFALLLIMIT':self.setTIMEFALLLIMIT,        # Definir constante
TIMEFALLLIMIT
            'setStartScript':self.StartScript,               # Cargar e iniciar
el programa de control (script) desarrollado por el usuario
            'setStopScript':self.StopScript,                  # Parar ejecución
del programa de control (script) desarrollado por el usuario
        }

    # REST API. Set
    def set(self, variables, valores):
        Tamaño = len(variables)                          # N° de variables
        for i in range(Tamaño):
            try:
                Nombre_Variable = variables[i]           # Nombre de la variable
                Valor_Variable = valores[i]              # Valor de la variable
                Action = self.Mapping[Nombre_Variable]  # Guardar el método set
correspondiente
                Action(Valor_Variable)                    # Llamar al método set
correspondiente con el valor correspondiente
            except TypeError as error:   # Error que se genera cuando una
operación o función se aplica a un objeto de tipo inapropiado
                Msg  =  str("Error método Set. Error en el tipo de dato
(TypeError):"+str(error))
                self.AplicacionWeb_Msg = Msg             # Mensaje a mostrar en
el Visor de Avisos de la aplicación Web
                print(Msg)
            except:                       # Error detectado durante la ejecución
del programa
                Msg        =        str("Error        inesperado        método
Set:"+str(sys.exc_info()[0]))  # sys.exc_info()[0]: tipo de excepción
                self.AplicacionWeb_Msg = Msg             # Mensaje a mostrar en
el Visor de Avisos de la aplicación Web
```

```
                print(Msg)

    # REST API. Get
    def get(self, variables):
        Resultado = []                                      # Inicialización
        for Nombre_Variable in variables:
            try:
                Action = self.Mapping[Nombre_Variable]  # Guardar el método get
correspondiente
                Resultado.append(Action())                  # Llamar al método get
correspondiente
            except TypeError as error:   # Error que se genera cuando una
operación o función se aplica a un objeto de tipo inapropiado
                Msg = str("Error método Get. Error en el tipo de dato
(TypeError):"+str(error))
                self.AplicacionWeb_Msg = Msg                # Mensaje a mostrar en
el Visor de Avisos de la aplicación Web
                print(Msg)
            except:                         # Error detectado durante la ejecución
del programa
                Msg         =         str("Error         inesperado         método
Get:"+str(sys.exc_info()[0]))  # sys.exc_info()[0]: tipo de excepción
                self.AplicacionWeb_Msg = Msg                # Mensaje a mostrar en
el Visor de Avisos de la aplicación Web
                print(Msg)
        return Resultado                                    # Devolver el resultado
de la lectura

    # Conectar
    def connect(self):
        try:
            self.ResetVariables()                   # Inicializar variables del
programa
            self.GoPiGo3Communication.reset()    # Inicializar variables de la
clase GoPiGo3Communication
            self.setPotenciaMotorIzq(0) # Potencia motor izquierdo (parar robot)
            self.setPotenciaMotorDer(0) # Potencia motor derecho (parar robot)
            self.ResetRobotEncoders(0)  # Resetear posición de los encoders del
motor izquierdo y derecho
            self.Running = True         # Conectado y en ejecución
            self.thread = threading.Thread(target=self.Worker)  # Crear hilo de
ejecución (función Worker)
            self.thread.start()                                 # Iniciar la
actividad del hilo
            Fabricante = self.GPG.get_manufacturer()            # Fabricante
GoPiGo3
            ReferenciaPlaca = self.GPG.get_board()              # Referencia
placa GoPiGo3
            NumeroSerie = self.GPG.get_id()                      # Número de
serie del hardware GoPiGo3
            Version_fw = self.GPG.get_version_firmware()          # Versión
firmware
            Version_hw = self.GPG.get_version_hardware()          # Versión
hardware
            Conectado = True                                    # Conectado
(variable para la aplicación Web)
            Msg = "Conexión establecida con el Robot GoPiGo3"
            self.AplicacionWeb_Msg = Msg                        # Mensaje a
mostrar en el Visor de Avisos de la aplicación Web
            return   Conectado,   Fabricante,   ReferenciaPlaca,   NumeroSerie,
Version_fw, Version_hw  # Devolver valores
        except:                                             # Error detectado
durante la ejecución del programa
            Msg = str("Error inesperado durante el establecimiento de la
conexión con el robot GoPiGo3:"+str(sys.exc_info()[0])) # sys.exc_info()[0]:
tipo de excepción
            self.AplicacionWeb_Msg = Msg                        # Mensaje a
mostrar en el Visor de Avisos de la aplicación Web
```

```
            print(Msg)

    # Desconectar
    def disconnect(self):
        self.ModoControlRobot = "Manual"            # Modo de funcionamiento
predeterminado: Manual
        self.Running = False                       # No conectado (no en ejecución)
        self.StartProgramaBalanceBot = False    # Parar la ejecución del programa
Balance Bot
        self.StartScriptUsuario = False        # Parar la ejecución del programa
de control (script) desarrollado por el usuario
        self.setPotenciaMotorIzq(0)            # Potencia motor izquierdo (parar
robot)
        self.setPotenciaMotorDer(0)             # Potencia motor derecho (parar
robot)
        self.ResetRobotEncoders(0)                  # Resetear posición de los
encoders del motor izquierdo y derecho
        self.KillScript()                          # Matar (kill) el programa de
control (script) desarrollado por el usuario
        Fabricante = ""                            # Fabricante GoPiGo3
        ReferenciaPlaca = ""                       # Referencia placa GoPiGo3
        NumeroSerie = ""                           # Número de serie del hardware
GoPiGo3
        Version_fw = ""                            # Versión firmware
        Version_hw = ""                            # Versión hardware
        Conectado = False                          # Conectado (variable para la
aplicación Web)
        Msg = "Conexión cerrada con el robot GoPiGo3"
        self.AplicacionWeb_Msg = Msg            # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
        return Conectado, Fabricante, ReferenciaPlaca, NumeroSerie, Version_fw,
Version_hw  # Devolver valores

    # Inicializar variables del programa
    def ResetVariables(self):
        self.ModoControlRobot = "Manual"            # Modo de funcionamiento
predeterminado: Manual
        self.ManualSpeed = 300                    # Velocidad predeterminada para
modo de funcionamiento Manual: 300 DPS
        self.AplicacionWeb_VELOCIDADACTUALMANUAL = self.ManualSpeed # Velocidad
predeterminada para la aplicación Web
        self.ControlRemotoIR = False              # Control Remoto receptor IR
desactivado
        self.BotonControlWebIR = 0              # Ningún botón seleccionado para
el Control Web IR
        self.StartProgramaBalanceBot = False    # Parar la ejecución del programa
Balance Bot
        self.GoPiGo3Communication.Ciclo_PrimerCiclo = False # Primer ciclo del
programa Balance Bot o programa de control (script) desarrollado por el usuario
        self.StartScriptUsuario = False        # Parar la ejecución del programa
de control (script) desarrollado por el usuario
        self.SubprocesoUsuario = -1             # Programa de control (script)
desarrollado por el usuario no en ejecución
        # Constantes utilizadas en el programa Balance Bot:
        self.KGYROANGLE = 15     # Ángulo general
        self.KGYROSPEED = 1.7    # Cambios repentinos de ángulo
        self.KPOS = 0.07         # Desviación de la posición objetivo
        self.KSPEED = 0.1        # Velocidad actual (real) del motor
        self.KDRIVE = -0.02      # Objetivo de velocidad de conducción (ayuda al
robot a iniciar/detener la conducción)
        self.KSTEER = 0.25       # Error en la dirección del robot
        self.KVOLTAGE = 0.083   # Cambios de voltaje. Configurada para 12V. Esta
constante ayuda al robot a adaptarse a las baterías con voltaje descendente
        self.DRIVESPEED = 175    # Velocidad de conducción cuando se controla
mediante el Control Remoto IR (grados por segundo)
        self.STEERSPEED = 100    # Velocidad de dirección cuando se controla
mediante el Control Remoto IR (grados por segundo)
```

```
        self.TIMEFALLLIMIT = 2  # Si los motores han estado funcionando a plena
potencia durante 2 segundos, suponga que el robot se cayó

    # Worker. Hilo de ejecucion (de este modo se permite que el programa ejecute
múltiples operaciones simultáneamente en el mismo espacio de proceso)
    def Worker(self):
        while(self.Running):    # Conectado y en ejecución
            self.LeerDatos()     # Leer datos del robot GoPiGo3 (batería, IMU
sensor, receptor IR, encoders y diámetro de la rueda)
            if      self.ModoControlRobot      ==      "Automatico"      and
(self.StartProgramaBalanceBot   or   self.StartScriptUsuario):     #   Modo   de
funcionamiento Automático y programa en ejecución
                self.ControlProgramaRobot()  # Ejecutar el programa Balance Bot
o el programa de control (script) desarrollado por el usuario
                # Actualizar parámetros del robot GoPiGo3

self.setPotenciaMotorIzq(self.GoPiGo3Communication.POTENCIAMOTORIZQ)          #
Definir potencia motor izquierdo

self.setPotenciaMotorDer(self.GoPiGo3Communication.POTENCIAMOTORDER)          #
Definir potencia motor derecho
                if self.GoPiGo3Communication.RESETENCODERS:
                    self.ResetRobotEncoders(0)    #  Resetear  posición  de  los
encoders del motor izquierdo y derecho
                time.sleep(0.02)     # Esperar 0.02 segundos
            else:
                time.sleep(0.1)      # Esperar 0.1 segundos

    # Leer datos del robot GoPiGo3 (batería, IMU sensor, receptor IR, encoders
y diámetro de la rueda)
    def LeerDatos(self):
        # Estado de la batería (voltaje)
        try:
            self.GoPiGo3Communication.BATERIA = self.GPG.get_voltage_battery()
# Leer voltaje de la batería
            self.AplicacionWeb_BATERIA    =    self.GoPiGo3Communication.BATERIA
# Actualizar valor para devolver a la aplicación Web
        except OSError as error:      # Esta excepción se produce cuando una
función del sistema retorna un error relacionado con el sistema
            Msg    =    str("Error    al    leer    voltaje    de    la    batería.
OSError:"+str(error.strerror)) # strerror: el mensaje de error correspondiente,
tal como lo proporciona el sistema operativo
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        except:                      # Error detectado durante la ejecución del
programa
            Msg = str("Ha sucedido un error inesperado al leer voltaje de la
batería:"+str(sys.exc_info()[0])) # sys.exc_info()[0]: tipo de excepción
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        # Lectura del sensor giroscópico IMU (obtener la velocidad angular del
giróscopo en grados por segundo)
        try:
            ValorIMUSENSOR = self.Imu.read_gyroscope()   # Lectura del sensor
giroscópico IMU
            self.GoPiGo3Communication.IMUSENSOR_X = ValorIMUSENSOR[0] # Eje X
            self.GoPiGo3Communication.IMUSENSOR_Y = ValorIMUSENSOR[1] # Eje Y
            self.GoPiGo3Communication.IMUSENSOR_Z = ValorIMUSENSOR[2] # Eje Z
            # Actualizar valores para devolver a la aplicación Web
            self.AplicacionWeb_IMUSENSOR_X                                    =
self.GoPiGo3Communication.IMUSENSOR_X
            self.AplicacionWeb_IMUSENSOR_Y                                    =
self.GoPiGo3Communication.IMUSENSOR_Y
            self.AplicacionWeb_IMUSENSOR_Z                                    =
self.GoPiGo3Communication.IMUSENSOR_Z
        except OSError as error:      # Esta excepción se produce cuando una
función del sistema retorna un error relacionado con el sistema
```

```
            Msg = str("Error al leer sensor IMU. OSError:"+str(error.strerror))
# strerror: el mensaje de error correspondiente, tal como lo proporciona el
sistema operativo
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        except:                         # Error detectado durante la ejecución del
programa
            Msg  =  str("Ha  sucedido  un  error  inesperado  al  leer  sensor
IMU:"+str(sys.exc_info()[0])) # sys.exc_info()[0]: tipo de excepción
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        # Lectura del receptor IR (Control Remoto)
        try:
            if self.StartProgramaBalanceBot:    # Programa Balance Bot
                if  self.ControlRemotoIR:       #  Control  Remoto  IR  activado
(utilizar mando)
                    self.GoPiGo3Communication.BOTON_IR                          =
self.GPG.get_grove_value(self.Port_Sensor_IR)  # Leer botón seleccionado en el
mando
                else:                           # Control Remoto IR desactivado
(utilizar botones de la aplicación Web)
                    self.GoPiGo3Communication.BOTON_IR = self.BotonControlWebIR
# Leer botón seleccionado en la aplicación Web
            elif self.StartScriptUsuario:       # Programa de control (script)
desarrollado por el usuario
                self.GoPiGo3Communication.BOTON_IR                              =
self.GPG.get_grove_value(self.Port_Sensor_IR)  # Leer botón seleccionado en el
mando
            # Descifrar la flecha presionada
            if self.GoPiGo3Communication.BOTON_IR == 1:     # Botón o flecha
presionada: adelante
                Msg = "Adelante"
            elif self.GoPiGo3Communication.BOTON_IR == 2:   # Botón o flecha
presionada: izquierda
                Msg = "Izquierda"
            elif self.GoPiGo3Communication.BOTON_IR == 3:   # Botón o flecha
presionada: OK
                Msg = "OK"
            elif self.GoPiGo3Communication.BOTON_IR == 4:   # Botón o flecha
presionada: derecha
                Msg = "Derecha"
            elif self.GoPiGo3Communication.BOTON_IR == 5:   # Botón o flecha
presionada: atrás
                Msg = "Atrás"
            else:                                           # Ningún botón o
flecha presionado
                Msg = "Ningún botón presionado"
            # Actualizar valor para devolver a la aplicación Web
            self.AplicacionWeb_BOTON_IR_SELECCIONADO = Msg
        except OSError as error:    # Esta excepción se produce cuando una
función del sistema retorna un error relacionado con el sistema
            Msg = str("Error al leer IR receiver. OSError:"+str(error.strerror))
# strerror: el mensaje de error correspondiente, tal como lo proporciona el
sistema operativo
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        except:                         # Error detectado durante la ejecución del
programa
            Msg  =  str("Ha  sucedido  un  error  inesperado  al  leer  IR
receiver:"+str(sys.exc_info()[0])) # sys.exc_info()[0]: tipo de excepción
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        # Lectura de la posición de los encoders (en grados) izquierdo y derecho
        try:
            # Leer posición de los encoders
            EncoderMotorIzq  =  self.GPG.get_motor_encoder(self.GPG.MOTOR_LEFT)
# Leer posición del encoder izquierdo
```

```
            EncoderMotorDer = self.GPG.get_motor_encoder(self.GPG.MOTOR_RIGHT)
# Leer posición del encoder derecho
            # Valor del encoder izquierdo
            if EncoderMotorIzq<0:
                SignEncMotorIzq = -1
            else:
                SignEncMotorIzq = 1
            VueltasEncIzq,                    GradosEncIzq                    =
divmod((SignEncMotorIzq*EncoderMotorIzq),360)
            self.GoPiGo3Communication.ENCMOTORIZQ       =       EncoderMotorIzq
# Posición del encoder izquierdo (grados)
            self.GoPiGo3Communication.VUELTASENCMOTORIZQ                        =
SignEncMotorIzq*VueltasEncIzq    # Número de vueltas del encoder izquierdo
            self.GoPiGo3Communication.GRADOSENCMOTORIZQ                         =
SignEncMotorIzq*GradosEncIzq        # Grados de la última vuelta del encoder
izquierdo (grados)
            # Valor del encoder izquierdo. Actualizar valores para devolver a
la aplicación Web
            self.AplicacionWeb_ENCMOTORIZQ                                      =
self.GoPiGo3Communication.ENCMOTORIZQ
            self.AplicacionWeb_VUELTASENCMOTORIZQ                               =
self.GoPiGo3Communication.VUELTASENCMOTORIZQ
            self.AplicacionWeb_GRADOSENCMOTORIZQ                                =
self.GoPiGo3Communication.GRADOSENCMOTORIZQ
            # Valor del encoder derecho
            if EncoderMotorDer<0:
                SignEncMotorDer = -1
            else:
                SignEncMotorDer = 1
            VueltasEncDer,                    GradosEncDer                    =
divmod((SignEncMotorDer*EncoderMotorDer),360)
            self.GoPiGo3Communication.ENCMOTORDER       =       EncoderMotorDer
# Posición del encoder derecho (grados)
            self.GoPiGo3Communication.VUELTASENCMOTORDER                        =
SignEncMotorDer*VueltasEncDer    # Número de vueltas del encoder derecho
            self.GoPiGo3Communication.GRADOSENCMOTORDER                         =
SignEncMotorDer*GradosEncDer        # Grados de la última vuelta del encoder
derecho (grados)
            # Valor del encoder derecho. Actualizar valores para devolver a la
aplicación Web
            self.AplicacionWeb_ENCMOTORDER                                      =
self.GoPiGo3Communication.ENCMOTORDER
            self.AplicacionWeb_VUELTASENCMOTORDER                               =
self.GoPiGo3Communication.VUELTASENCMOTORDER
            self.AplicacionWeb_GRADOSENCMOTORDER                                =
self.GoPiGo3Communication.GRADOSENCMOTORDER
        except OSError as error:      # Esta excepción se produce cuando una
función del sistema retorna un error relacionado con el sistema
            Msg = str("Error al leer posición encoder motor izquierdo/derecho.
OSerror:"+str(error.strerror)) # strerror: el mensaje de error correspondiente,
tal como lo proporciona el sistema operativo
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        except:                       # Error detectado durante la ejecución del
programa
            Msg = str("Ha sucedido un error inesperado al leer posición encoder
motor izquierdo/derecho:"+str(sys.exc_info()[0])) # sys.exc_info()[0]: tipo de
excepción
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        # Diámetro de la rueda del robot GoPiGo3
        try:
            self.GoPiGo3Communication.DIAMETRORUEDA  =  self.GPG.WHEEL_DIAMETER
# Leer valor del diámetro de la rueda del Robot GoPiGo3
            self.AplicacionWeb_DIAMETRORUEDA                                    =
self.GoPiGo3Communication.DIAMETRORUEDA  # Actualizar valor para devolver a la
aplicación Web
```

```
        except OSError as error:      # Esta excepción se produce cuando una función del sistema retorna un error relacionado con el sistema
            Msg = str("Error al leer diámetro rueda robot GoPiGo3. OSError:"+str(error.strerror)) # strerror: el mensaje de error correspondiente, tal como lo proporciona el sistema operativo
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de Avisos de la aplicación Web
        except:                       # Error detectado durante la ejecución del programa
            Msg = str("Ha sucedido un error inesperado al leer diámetro rueda robot GoPiGo3:"+str(sys.exc_info()[0])) # sys.exc_info()[0]: tipo de excepción
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de Avisos de la aplicación Web

    # Selecciona el modo de funcionamiento (manual/automático) del robot GoPiGo3
    def ModoControl(self, modo):
        self.ModoControlRobot = modo
        if self.ModoControlRobot == "Manual":       # Modo Manual
            # Al seleccionar el modo de funcionamiento Manual, parar programa Balance Bot y programa de control (script) desarrollado por el usuario
            self.StopBalanceBot(0)  # Parar ejecución del programa Balance Bot
            self.StopScript(0)       # Parar ejecución del programa de control (script) desarrollado por el usuario
            # Inicializar constantes utilizadas en el programa Balance Bot:
            self.KGYROANGLE = 15     # Ángulo general
            self.KGYROSPEED = 1.7    # Cambios repentinos de ángulo
            self.KPOS = 0.07         # Desviación de la posición objetivo
            self.KSPEED = 0.1        # Velocidad actual (real) del motor
            self.KDRIVE = -0.02     # Objetivo de velocidad de conducción (ayuda al robot a iniciar/detener la conducción)
            self.KSTEER = 0.25       # Error en la dirección del robot
            self.KVOLTAGE = 0.083    # Cambios de voltaje. Configurada para 12V. Esta constante ayuda al robot a adaptarse a las baterías con voltaje descendente
            self.DRIVESPEED = 175    # Velocidad de conducción cuando se controla mediante el Control Remoto IR (grados por segundo)
            self.STEERSPEED = 100    # Velocidad de dirección cuando se controla mediante el Control Remoto IR (grados por segundo)
            self.TIMEFALLLIMIT = 2   # Si los motores han estado funcionando a plena potencia durante 2 segundos, suponga que el robo
            # Mostrar mensaje en el Visor de Avisos
            Msg = "Robot GoPiGo3 en modo Manual"
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de Avisos de la aplicación Web
        elif self.ModoControlRobot == "Automatico":    # Modo Automático
            # Al seleccionar el modo de funcionamiento Automático, parar motores
            self.setPotenciaMotorIzq(0)              # Potencia motor izquierdo (parar robot)
            self.setPotenciaMotorDer(0)               # Potencia motor derecho (parar robot)
            self.ManualSpeed = 300                   # Velocidad predeterminada para modo de funcionamiento Manual: 300 DPS
            self.AplicacionWeb_VELOCIDADACTUALMANUAL   =   self.ManualSpeed   # Velocidad predeterminada para la aplicación Web
            # Mostrar mensaje en el Visor de Avisos
            Msg = "Robot GoPiGo3 en modo Automático"
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de Avisos de la aplicación Web

    # Establecer la velocidad manual del robot GoPiGo3 especificado por el argumento value
    def setRobotVeloManual(self, value):
        # La velocidad se mide en DPS = grados por segundo de la rueda del robot (s). Velocidad entre 0-1000 DPS
        self.ManualSpeed = int(value)
        self.AplicacionWeb_VELOCIDADACTUALMANUAL = self.ManualSpeed
        Msg    =    str("Velocidad    seleccionada    del    robot    GoPiGo3:"+ str(self.ManualSpeed))
```

```
        self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de Avisos
de la aplicación Web

    # Ejecutar movimiento manual del robot GoPiGo3
    def MovimientoManual(self,type):
        if self.ModoControlRobot == "Manual":
            if type==1:     # Adelante
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT                    +
self.GPG.MOTOR_RIGHT, self.ManualSpeed)     # Mover hacia adelante en velocidad
"self.ManualSpeed"
                Msg = "Robot moviendo hacia adelante"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            elif type==2:   # Atrás
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT                    +
self.GPG.MOTOR_RIGHT, (-1*self.ManualSpeed))   # Mover hacia atrás en velocidad
"self.ManualSpeed"
                Msg = "Robot moviendo hacia atrás"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            elif type==3:   # Izquierda
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT, 0)
                self.GPG.set_motor_dps(self.GPG.MOTOR_RIGHT,  self.ManualSpeed)
# Mover hacia el lado izquierdo en velocidad "self.ManualSpeed"
                Msg = "Robot moviendo hacia el lado izquierdo"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            elif type==4:   # Derecha
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT,   self.ManualSpeed)
# Mover hacia el lado derecho en velocidad "self.ManualSpeed"
                self.GPG.set_motor_dps(self.GPG.MOTOR_RIGHT, 0)
                Msg = "Robot moviendo hacia el lado derecho"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            elif type==5:   # Giro izquierda
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT,                   (-
1*self.ManualSpeed))     #  Girar  hacia  el  lado  izquierdo  en  velocidad
"self.ManualSpeed"
                self.GPG.set_motor_dps(self.GPG.MOTOR_RIGHT, self.ManualSpeed)
                Msg = "Robot girando hacia el lado izquierdo"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            elif type==6:   # Giro derecha
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT,   self.ManualSpeed)
# Girar hacia el lado derecho en velocidad "self.ManualSpeed"
                self.GPG.set_motor_dps(self.GPG.MOTOR_RIGHT,                  (-
1*self.ManualSpeed))
                Msg = "Robot girando hacia el lado derecho"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            elif type==7:   # Paro
                self.GPG.set_motor_dps(self.GPG.MOTOR_LEFT                    +
self.GPG.MOTOR_RIGHT, 0)   # Parar robot
                Msg = "Robot parado"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web

    # Resetear posición de los encoders del motor izquierdo y derecho
    def ResetRobotEncoders(self, value):
        self.GPG.set_motor_power(self.GPG.MOTOR_LEFT + self.GPG.MOTOR_RIGHT, 0)
# Parar robot (potencia = 0)
        self.GPG.offset_motor_encoder(self.GPG.MOTOR_LEFT,
self.GPG.get_motor_encoder(self.GPG.MOTOR_LEFT))   # Reset encoder izquierdo
        self.GPG.offset_motor_encoder(self.GPG.MOTOR_RIGHT,
self.GPG.get_motor_encoder(self.GPG.MOTOR_RIGHT)) # Reset encoder derecho
        if self.ModoControlRobot == "Manual":
            Msg = "Encoders del robot GoPiGo3 reseteados"
```

```
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web

    # Definir la potencia del motor izquierdo del robot GoPiGo3
    def setPotenciaMotorIzq(self, value):
        if value == "MOTOR_FLOAT":
            Potencia = self.GPG.MOTOR_FLOAT
        else:
            Potencia = int(value)
        self.GPG.set_motor_power(self.GPG.MOTOR_LEFT , Potencia)     # Definir
potencia motor izquierdo
        if self.ModoControlRobot == "Manual":
            Msg   =   str("Potencia   seleccionada   del   motor   izquierdo:"+
str(Potencia))
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web

    # Definir la potencia del motor derecho del robot GoPiGo3
    def setPotenciaMotorDer(self, value):
        if value == "MOTOR_FLOAT":
            Potencia = self.GPG.MOTOR_FLOAT
        else:
            Potencia = int(value)
        self.GPG.set_motor_power(self.GPG.MOTOR_RIGHT, Potencia)     # Definir
potencia motor derecho
        if self.ModoControlRobot == "Manual":
            Msg = str("Potencia seleccionada del motor derecho:"+ str(Potencia))
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web

    # Activar/Desactivar Control Remoto IR
    def setControlRemotoIR(self,value):
        self.ControlRemotoIR = value
        if self.ControlRemotoIR == 0:   # Control Remoto IR desactivado
            Msg = "Control Remoto IR desactivado"
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        elif self.ControlRemotoIR == 1: # Control Remoto IR activado
            Msg = "Control Remoto IR activado. Utilizar el mando"
            self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web

    # Guarda el botón seleccionado en la interfaz Web (para el programa Balance
Bot)
    def setMovControlWebIR(self,value):
        self.BotonControlWebIR = value

    # Ejecutar el programa Balance Bot o el programa de control (script)
desarrollado por el usuario
    def ControlProgramaRobot(self):
        try:
            if self.StartProgramaBalanceBot:
                self.ProgramaBalanceBot()   # Ejecutar el programa Balance Bot
            elif self.StartScriptUsuario:
                # Ejecutar un subproceso. Programa de control (script)
desarrollado por el usuario
                self.SubprocesoUsuario    =    subprocess.Popen(['python','-u',
'CodigoUsuario.py'],  # Abrir un subproceso
                    stdin = subprocess.PIPE,
                    stdout = subprocess.PIPE
                )
                # Interactuar con el proceso: enviar datos a stdin (clase
GoPiGo3Communication) y leer datos de stdout
                output                                                  =
self.SubprocesoUsuario.communicate(input=self.GoPiGo3Communication.toJSON().en
code())[0]
                # Deserializar cadenas nativas o bytes (datos de stdout) que
constan de datos JSON en Python Dictionary
```

```
                Communication = json.loads(output.decode('utf-8'))
                # Actualizar parámetros en el programa
                if self.Running and self.ModoControlRobot == "Automatico":  # Conectado y en modo Automático
                    self.GoPiGo3Communication.POTENCIAMOTORIZQ = Communication['POTENCIAMOTORIZQ']  # Potencia motor izquierdo
                    self.GoPiGo3Communication.POTENCIAMOTORDER = Communication['POTENCIAMOTORDER']  # Potencia motor derecho
                    self.GoPiGo3Communication.RESETENCODERS = Communication['RESETENCODERS']  # Reset encoders (motor izquierdo y derecho)
                    self.GoPiGo3Communication.msg= Communication['MENSAJE']  # Mensaje del programa de control (script) desarrollado por el usuario
                    self.AplicacionWeb_Msg = self.GoPiGo3Communication.msg  # Mensaje a mostrar en el Visor de Avisos de la aplicación Web
                    self.GoPiGo3Communication.Ciclo_PrimerCiclo = Communication['Ciclo_PrimerCiclo']  # Primer ciclo del programa de control (script) desarrollado por el usuario
                    self.GoPiGo3Communication.VariableAuxiliar = Communication['VariableAuxiliar']  # Para almacenar cualquier valor interno en el programa de control (script) desarrollado por el usuario (para tener accesible en cada ejecución)
                else:  # Inicializar variables
                    self.GoPiGo3Communication.POTENCIAMOTORIZQ = 0  # Potencia motor izquierdo
                    self.GoPiGo3Communication.POTENCIAMOTORDER = 0  # Potencia motor derecho
                    self.GoPiGo3Communication.RESETENCODERS = False  # Reset encoders (motor izquierdo y derecho)
                    self.GoPiGo3Communication.msg= ""  # Mensaje del programa de control (script) desarrollado por el usuario
                    self.GoPiGo3Communication.VariableAuxiliar = {}  # Para almacenar cualquier valor interno en el programa de control (script) desarrollado por el usuario (para tener accesible en cada ejecución)
            except ValueError as error:  # Esta excepción se genera cuando una operación o función recibe un argumento que tiene el tipo correcto pero un valor inapropiado
                Msg = str("Error en la ejecución del programa de control. Valor inapropiado (ValueError):"+str(error))
                self.AplicacionWeb_Msg = Msg  # Mensaje a mostrar en el Visor de Avisos de la aplicación Web
            except:  # Error detectado durante la ejecución del programa
                Msg = str("Ha sucedido un error inesperado en la ejecución del programa de control:"+str(sys.exc_info()[0]))  # sys.exc_info()[0]: tipo de excepción
                self.AplicacionWeb_Msg = Msg  # Mensaje a mostrar en el Visor de Avisos de la aplicación Web

        # Iniciar programa Balance Bot
        def StartBalanceBot(self, value):
            self.StartProgramaBalanceBot = True  # Iniciar el programa Balance Bot
            self.GoPiGo3Communication.Ciclo_PrimerCiclo = True  # Primer ciclo del programa Balance Bot
            Msg = "Programa Balance Bot iniciado"
            self.AplicacionWeb_Msg = Msg  # Mensaje a mostrar en el Visor de Avisos de la aplicación Web

        # Parar ejecución del programa Balance Bot
        def StopBalanceBot(self, value):
            self.StartProgramaBalanceBot = False  # Parar la ejecución del programa Balance Bot
            self.ControlRemotoIR = False  # Control Remoto receptor IR desactivado
            self.BotonControlWebIR = 0  # Ningún botón seleccionado para el Control Web IR
```

```
        self.GoPiGo3Communication.reset()         # Inicializar variables de la
clase GoPiGo3Communication
        self.setPotenciaMotorIzq(0)            # Potencia motor izquierdo (parar
robot)
        self.setPotenciaMotorDer(0)              # Potencia motor derecho (parar
robot)
        Msg = "Programa Balance Bot parado"
        self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de Avisos
de la aplicación Web

    # Programa Balance Bot
    def ProgramaBalanceBot(self):
        # DEFINIR CONSTANTES
        # Constantes utilizadas para definir qué tan agresivamente debe
responder el robot a:
        CTE_KGYROANGLE = float(self.KGYROANGLE)     # Ángulo general
        CTE_KGYROSPEED = float(self.KGYROSPEED)    # Cambios repentinos de ángulo
        CTE_KPOS       = float(self.KPOS)           # Desviación de la posición
objetivo
        CTE_KSPEED     = float(self.KSPEED)        # Velocidad actual (real) del
motor
        CTE_KDRIVE      = float(self.KDRIVE)         # Objetivo de velocidad de
conducción (ayuda al robot a iniciar/detener la conducción)
        CTE_KSTEER      = float(self.KSTEER)        # Error en la dirección del
robot
        CTE_KVOLTAGE     = float(self.KVOLTAGE)         # Cambios de voltaje.
Configurada para 12V. Esta constante ayuda al robot a adaptarse a las baterías
con voltaje descendente
        # Velocidad de conducción cuando se controla mediante el Control Remoto
IR (grados por segundo)
        CTE_DRIVE_SPEED = float(self.DRIVESPEED)
        # Velocidad de dirección cuando se controla mediante el Control Remoto
IR (grados por segundo)
        CTE_STEER_SPEED = float(self.STEERSPEED)
        # Si los motores han estado funcionando a plena potencia durante x
segundos, suponga que el robot se cayó
        CTE_TIME_FALL_LIMIT = float(self.TIMEFALLLIMIT)
        # Qué tan rápido se ejecuta el bucle de balance en Hz. Más lento consume
menos CPU, pero más rápido ayuda al bot de equilibrio a funcionar mejor
        # Calquier valor superior a 100 no hará ninguna diferencia, ya que el
sensor giroscópico IMU no se actualizará más rápido y los motores tardan un
tiempo en responder físicamente
        CTE_LOOP_SPEED = 100
        # Cuántos segundos debe tomar cada bucle (a 100 Hz, esto debe ser 0.01)
        CTE_LOOP_TIME = (1 / CTE_LOOP_SPEED)
        # Una constante utilizada para corregir el retraso para tratar de
mantener una velocidad de bucle perfecta (definida por LOOP_SPEED)
        CTE_KCORRECTTIME = 0.001
        # Una constante utilizada para corregir/centrar el ángulo acumulado del
giroscopio de modo que la deriva integral del giroscopio se resuelva más rápido
de lo que puede acumularse
        CTE_KGYROANGLECORRECT = 0.1
        # Diámetro de la rueda en mm
        CTE_WHEEL_DIAMETER = self.GoPiGo3Communication.DIAMETRORUEDA
        # Afinado para ruedas de 56 mm
        CTE_WHEEL_RATIO = (CTE_WHEEL_DIAMETER / 56)

        # INICIALIZAR VARIABLES
        self.GoPiGo3Communication.RESETENCODERS = False

        # CÓDIGO PRIMER CICLO
        if self.GoPiGo3Communication.Ciclo_PrimerCiclo:
            # Flotar los motores
            self.GoPiGo3Communication.POTENCIAMOTORIZQ = "MOTOR_FLOAT"
            self.GoPiGo3Communication.POTENCIAMOTORDER = "MOTOR_FLOAT"
            # Esperar hasta que se presione el botón 'OK'
            if self.GoPiGo3Communication.BOTON_IR == 3:    #     Botón     'OK'
presionado
```

```
                # Resetear encoders
                self.GoPiGo3Communication.RESETENCODERS = True
                # Inicializar variables
                self.AnguloGyro = 0
                self.SumaEncActual = 0
                self.PosicionRobot = 0
                self.DeltaEncoder = 0
                self.GiroManualARecorrer = 0
                self.OffsetTiempo = 0
                self.TiempoRealBucle = CTE_LOOP_TIME
                self.UltimoTiempo = time.time() - CTE_LOOP_TIME
                self.GoPiGo3Communication.Ciclo_PrimerCiclo = False    # Primer
ciclo del programa Balance Bot
                self.AplicacionWeb_Msg = "Robot equilibrándose. Soltar el robot"
# Mensaje a mostrar en el Visor de Avisos de la aplicación Web
            else:
                self.AplicacionWeb_Msg = "Robot en el suelo. Levante el robot y
luego presione botón 'OK' en el control remoto"  # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
        # CÓDIGO BALANCE BOT
        else:
            # Realizar bucle exactamente a la velocidad especificada por
CTE_LOOP_SPEED, y establecer TiempoRealBucle al tiempo real del bucle
            TiempoActual = time.time()      # Tiempo actual
            self.OffsetTiempo += ((self.TiempoRealBucle - CTE_LOOP_TIME) *
CTE_KCORRECTTIME)           # Para ajustar los gastos generales, de modo que
intenta realizar un bucle exactamente a la velocidad especificada por
CTE_LOOP_SPEED
            TiempoRetardo = (CTE_LOOP_TIME - (TiempoActual - self.UltimoTiempo))
- self.OffsetTiempo  # Tiempo de retardo
            if TiempoRetardo > 0:
                time.sleep(TiempoRetardo)    # Esperar "Tiempo de retardo"
            TiempoActual = time.time()      # Tiempo actual
            self.TiempoRealBucle = (TiempoActual - self.UltimoTiempo)  # Tiempo
real del bucle
            self.UltimoTiempo = TiempoActual    # Último tiempo

            # Mirar el botón presionado en el Control Remoto. Si se presiona una
flecha, conduzca o gire en consecuencia
            if self.GoPiGo3Communication.BOTON_IR == 1:     # Botón presionado:
adelante
                MotorControlManualMovimientoLineal = CTE_DRIVE_SPEED       #
Movimiento lineal del motor
                MotorControlManualGiro = 0                                 #
Movimiento rotatorio del motor
            elif self.GoPiGo3Communication.BOTON_IR == 5:   # Botón presionado:
atrás
                MotorControlManualMovimientoLineal = -CTE_DRIVE_SPEED
                MotorControlManualGiro = 0
            elif self.GoPiGo3Communication.BOTON_IR == 2:   # Botón presionado:
izquierda
                MotorControlManualMovimientoLineal = 0
                MotorControlManualGiro = -CTE_STEER_SPEED
            elif self.GoPiGo3Communication.BOTON_IR == 4:   # Botón presionado:
derecha
                MotorControlManualMovimientoLineal = 0
                MotorControlManualGiro = CTE_STEER_SPEED
            else:                                           # Ningún botón
presionado
                MotorControlManualMovimientoLineal = 0
                MotorControlManualGiro = 0
            # Integrar el giroscopio para obtener el ángulo del robot GoPiGo3
            self.AnguloGyro    +=    self.GoPiGo3Communication.IMUSENSOR_Z    *
self.TiempoRealBucle
            # Corregir los errores de integración del giróscopo (trabajar
lentamente hacia el centro)
            self.AnguloGyro  -=  (self.AnguloGyro  *  CTE_KGYROANGLECORRECT  *
self.TiempoRealBucle)
```

```
            # Calcular la velocidad del robot (motor)
            SumaEncPrevio = self.SumaEncActual      # Suma previa de la posición
de los encoders
            self.SumaEncActual    =    self.GoPiGo3Communication.ENCMOTORIZQ    +
self.GoPiGo3Communication.ENCMOTORDER   # Suma actual de la posición de los
encoders
            self.DeltaEncoder  =  self.SumaEncActual  -  SumaEncPrevio          #
Distancia recorrida del robot (diferencia entre "PosiciónEncoderActual -
PosiciónEncoderPrevio")
            VelocidadRobot = self.DeltaEncoder / self.TiempoRealBucle   #
Velocidad del robot (motor)
            # Ajustar la posición/objetivo del motor
            DistManualARecorrer    =    MotorControlManualMovimientoLineal    *
self.TiempoRealBucle # Distancia a recorrer del robot (en caso de utilizar
pulsador Control Remoto)
            self.PosicionRobot  +=  (self.DeltaEncoder  -  DistManualARecorrer)
      # Posición objetivo del robot (motor)
            # Calcular la potencia del motor
            PotenciaMotor           =           (((CTE_KGYROSPEED           *
self.GoPiGo3Communication.IMUSENSOR_Z + # ((Grados/seg del sensor giroscópico +
                        CTE_KGYROANGLE * self.AnguloGyro) / CTE_WHEEL_RATIO
+            # Grados de la integral del giróscopo) / relación de rueda +
                        CTE_KPOS                *   self.PosicionRobot   +
             # Posición de ambos motores +
                        CTE_KDRIVE     * MotorControlManualMovimientoLineal
+          # Para mejorar el rendimiento de inicio/parada +
                        CTE_KSPEED     * VelocidadRobot) /
          # Velocidad del motor en grados/seg) /
                        (CTE_KVOLTAGE * self.GoPiGo3Communication.BATERIA))
# Para mantener un rendimiento similar en un rango de voltaje
            # Si la potencia no está al máximo, registrar el tiempo
            if abs(PotenciaMotor) < 100:
                self.TiempoMotorPosOK = TiempoActual
            # Calcular la velocidad del motor izquierdo y derecho, teniendo en
cuenta el giro deseado
            self.GiroManualARecorrer      +=      MotorControlManualGiro      *
self.TiempoRealBucle    # Giro deseado del robot (en caso de utilizar pulsador
Control Remoto)
            DiferenciaEncoder    =    self.GoPiGo3Communication.ENCMOTORIZQ    -
self.GoPiGo3Communication.ENCMOTORDER     # Diferencia entre la posición del
encoder del motor izquierdo y derecho
            PotenciaGiro    =    CTE_KSTEER    *    (self.GiroManualARecorrer    -
DiferenciaEncoder)  # Potencia para realizar el giro deseado del robot (en caso
de utilizar pulsador Control Remoto)
            self.GoPiGo3Communication.POTENCIAMOTORIZQ    =    PotenciaMotor    +
PotenciaGiro  # Potencia motor izquierdo (teniendo en cuenta mantener equilibrio
+ giro deseado)
            self.GoPiGo3Communication.POTENCIAMOTORDER    =    PotenciaMotor    -
PotenciaGiro   # Potencia motor derecho (teniendo en cuenta mantener equilibrio
+ giro deseado)
            # Limitar la potencia del motor a +-100
            if  self.GoPiGo3Communication.POTENCIAMOTORIZQ  >  100:
                self.GoPiGo3Communication.POTENCIAMOTORIZQ  =  100
            if  self.GoPiGo3Communication.POTENCIAMOTORIZQ  < -100:
                self.GoPiGo3Communication.POTENCIAMOTORIZQ  = -100
            if  self.GoPiGo3Communication.POTENCIAMOTORDER >  100:
                self.GoPiGo3Communication.POTENCIAMOTORDER =  100
            if  self.GoPiGo3Communication.POTENCIAMOTORDER < -100:
                self.GoPiGo3Communication.POTENCIAMOTORDER = -100
            # Si los motores han estado funcionando a plena potencia durante al
menos CTE_TIME_FALL_LIMIT, se supone que el robot se cayó
            if (TiempoActual - self.TiempoMotorPosOK) > CTE_TIME_FALL_LIMIT:
                self.GoPiGo3Communication.Ciclo_PrimerCiclo = True      #
Primer ciclo del programa Balance Bot

    # Definir constante KGYROANGLE
    def setKGYROANGLE(self, value):
        self.KGYROANGLE = value
```

```
        self.Msg1 = "Constantes para el programa Balance Bot:"
        self.Msg2 = str("- Constante KGYROANGLE:"+str(self.KGYROANGLE))

    # Definir constante KGYROSPEED
    def setKGYROSPEED(self, value):
        self.KGYROSPEED = value
        self.Msg3 = str("- Constante KGYROSPEED:"+str(self.KGYROSPEED))

    # Definir constante KPOS
    def setKPOS(self, value):
        self.KPOS = value
        self.Msg4 = str("- Constante KPOS:"+str(self.KPOS))

    # Definir constante KSPEED
    def setKSPEED(self, value):
        self.KSPEED = value
        self.Msg5 = str("- Constante KSPEED:"+str(self.KSPEED))

    # Definir constante KDRIVE
    def setKDRIVE(self, value):
        self.KDRIVE = value
        self.Msg6 = str("- Constante KDRIVE:"+str(self.KDRIVE))

    # Definir constante KSTEER
    def setKSTEER(self, value):
        self.KSTEER = value
        self.Msg7 = str("- Constante KSTEER:"+str(self.KSTEER))

    # Definir constante KVOLTAGE
    def setKVOLTAGE(self, value):
        self.KVOLTAGE = value
        self.Msg8 = str("- Constante KVOLTAGE:"+str(self.KVOLTAGE))

    # Definir constante DRIVESPEED
    def setDRIVESPEED(self, value):
        self.DRIVESPEED = value
        self.Msg9 = str("- Constante DRIVE SPEED:"+str(self.DRIVESPEED))

    # Definir constante STEERSPEED
    def setSTEERSPEED(self, value):
        self.STEERSPEED = value
        self.Msg10 = str("- Constante STEER SPEED:"+str(self.STEERSPEED))

    # Definir constante TIMEFALLLIMIT
    def setTIMEFALLLIMIT(self, value):
        self.TIMEFALLLIMIT = value
        self.Msg11 = str("- Constante TIME FALL LIMIT:"+str(self.TIMEFALLLIMIT))
        Msg                                                                    =
self.Msg1+'\n'+self.Msg2+'\n'+self.Msg3+'\n'+self.Msg4+'\n'+self.Msg5+'\n'+sel
f.Msg6+'\n'+self.Msg7+'\n'+self.Msg8+'\n'+self.Msg9+'\n'+self.Msg10+'\n'+self.
Msg11
        self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de Avisos
de la aplicación Web

    # Cargar y iniciar el programa de control (script) desarrollado por el
usuario
    def StartScript(self,text):
        try:
            Code  =  open(".//CodigoUsuario.py","w")     #  Abrir  el  archivo
"CodigoUsuario.py (y si no existe crear)"
            Code.write(text)      # Escribir el programa de control (script)
desarrollado por el usuario en el archivo "CodigoUsuario.py"
            Code.close()        # Cerrar el archivo "CodigoUsuario.py"
            self.StartScriptUsuario = True   # Iniciar el programa de control
(script) desarrollado por el usuario
            self.GoPiGo3Communication.Ciclo_PrimerCiclo = True   # Primer ciclo
del programa de control (script) desarrollado por el usuario
```

```
            Msg = "Programa de control (script) desarrollado por el usuario
cargado y iniciado"
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        except TypeError as error:  # Error que se genera cuando una operación
o función se aplica a un objeto de tipo inapropiado
            Msg = str("Error al iniciar el programa de control desarrollado por
el usuario. Error en el tipo de dato (TypeError):"+str(error))
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web
        except:                         # Error detectado durante la ejecución del
programa
            Msg = str("Ha sucedido un error inesperado al iniciar el programa
de control desarrollado por el usuario:"+str(sys.exc_info()[0]))    #
sys.exc_info()[0]: tipo de excepción
            self.AplicacionWeb_Msg = Msg     # Mensaje a mostrar en el Visor de
Avisos de la aplicación Web

    # Parar ejecución del programa de control (script) desarrollado por el
usuario
    def StopScript(self, value):
        self.StartScriptUsuario = False      # Parar la ejecución del programa
de control (script) desarrollado por el usuario
        self.GoPiGo3Communication.reset()   # Inicializar variables de la clase
GoPiGo3Communication
        self.setPotenciaMotorIzq(0)          # Potencia motor izquierdo (parar
robot)
        self.setPotenciaMotorDer(0)            # Potencia motor derecho (parar
robot)
        Msg = "Programa de control (script) desarrollado por el usuario parado"
        self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor de Avisos
de la aplicación Web

    # Matar (kill) el programa de control (script) desarrollado por el usuario
    def KillScript(self):
        if self.SubprocesoUsuario != -1:        # Programa de control (script)
desarrollado por el usuario en ejecución
            try:
                self.SubprocesoUsuario.kill()  # Matar (kill) el programa de
control (script) desarrollado por el usuario
                self.SubprocesoUsuario = -1    # Programa de control (script)
desarrollado por el usuario no en ejecución
                Msg = "Matar (kill) programa de control (script) desarrollado
por el usuario"
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
            except:                                 # Error detectado durante la
ejecución del programa
                Msg = str("Ha sucedido un error inesperado al matar (kill) el
programa de control desarrollado por el usuario:"+str(sys.exc_info()[0]))  #
sys.exc_info()[0]: tipo de excepción
                self.AplicacionWeb_Msg = Msg    # Mensaje a mostrar en el Visor
de Avisos de la aplicación Web
```

10.5 Clase GoPiGo3Communication.py

```
# Autor: Mikel Iturrioz
# Última modificación: 04/08/2021

# Importar librerias
import json     # Es un formato ligero de intercambio de datos inspirado por la
sintaxis literal de objetos en JavaScript

# Clase GoPiGo3Communication
class GoPiGo3Communication(object):
    def __init__(self):
        # Inicialización
```

```
        self.BATERIA = 0
        self.IMUSENSOR_X = 0
        self.IMUSENSOR_Y = 0
        self.IMUSENSOR_Z = 0
        self.BOTON_IR = 0
        self.ENCMOTORIZQ = 0
        self.VUELTASENCMOTORIZQ = 0
        self.GRADOSENCMOTORIZQ = 0
        self.ENCMOTORDER = 0
        self.VUELTASENCMOTORDER = 0
        self.GRADOSENCMOTORDER = 0
        self.DIAMETRORUEDA = 0
        self.POTENCIAMOTORIZQ = 0
        self.POTENCIAMOTORDER = 0
        self.RESETENCODERS = False
        self.msg = ""
        self.Ciclo_PrimerCiclo = False
        self.VariableAuxiliar = {}

    # Inicializar variables
    def reset(self):
        self.POTENCIAMOTORIZQ = 0
        self.POTENCIAMOTORDER = 0
        self.RESETENCODERS = False
        self.Ciclo_PrimerCiclo = False
        self.msg = ""

    # Convertir un objeto Python en una cadena json
    def toJSON(self):
        return json.dumps(self, default = lambda o: o.__dict__, sort_keys =
True, indent = 4)
```

10.6 Código usuario ejemplo

```
# Autor: Mikel Iturrioz
# Última modificación: 04/08/2021

# IMPORTAR LIBRERÍAS
import sys, json

# LEER VARIABLES
# Recuperar el objeto que permite acceder a la clase GoPiGo3Communication
obj = sys.stdin.read()
ComunicacionGoPiGo3 = json.loads(obj)
# Actualizar variables
BateriaRobot = ComunicacionGoPiGo3['BATERIA']                    # Obtener la
batería (voltaje) del robot
GyroEjeZ = ComunicacionGoPiGo3['IMUSENSOR_Z']                    # Obtener la
velocidad de rotación del giróscopo (eje Z) en grados por segundo
BotonIRSeleccionado = ComunicacionGoPiGo3['BOTON_IR']            #       Botón
(flecha) seleccionado en el mando o en la aplicación Web
PosEncIzq = ComunicacionGoPiGo3['ENCMOTORIZQ']                                #
Obtener la posición actual del encoder del motor izquierdo (grados)
PosEncDer = ComunicacionGoPiGo3['ENCMOTORDER']                                #
Obtener la posición actual del encoder del motor derecho (grados)
DiametroRueda = ComunicacionGoPiGo3['DIAMETRORUEDA']       # Obtener diámetro
de la rueda del robot (mm)
IniciarPrograma = ComunicacionGoPiGo3['Ciclo_PrimerCiclo'] # Primer ciclo del
código usuario
VariableAuxiliar = ComunicacionGoPiGo3['VariableAuxiliar'] # Obtener el valor
interno de cualquier variable

# INICIALIZAR VARIABLES
PotenciaMotorIzq = 0                    # Potencia motor izquierdo
PotenciaMotorDer = 0                    # Potencia motor derecho
ResetEncoders = False                   # Reset encoders (izquierdo y derecho)
Mensaje = ""
```

```
NumCiclosActual = 0                     # Indica el número de ciclos en ejecución

# CÓDIGO PRIMER CICLO
if IniciarPrograma:
        ResetEncoders = True            # Reset encoders (izquierdo y derecho)
        IniciarPrograma = False         # Primer ciclo del programa Balance Bot
        Mensaje = "Primer ciclo del código usuario realizado. Presionar flecha en
el Control Remoto (mando) para mover robot"
# CÓDIGO USUARIO
else:
        if BateriaRobot > 9:
                # Mirar el botón presionado en el Control Remoto. Si se presiona
una flecha, conduzca o gire en consecuencia
                if BotonIRSeleccionado == 1:     # Botón presionado: adelante
                        PotenciaMotorIzq = 50           # Potencia motor izquierdo
                        PotenciaMotorDer = 50           # Potencia motor derecho
                        Mensaje1 = "Robot moviendo hacia adelante"
                elif BotonIRSeleccionado == 5:   # Botón presionado: atrás
                        PotenciaMotorIzq = -50          # Potencia motor izquierdo
                        PotenciaMotorDer = -50          # Potencia motor derecho
                        Mensaje1 = "Robot moviendo hacia atrás"
                elif BotonIRSeleccionado == 2:   # Botón presionado: izquierda
                        PotenciaMotorIzq = 0            # Potencia motor izquierdo
                        PotenciaMotorDer = 50           # Potencia motor derecho
                        Mensaje1 = "Robot moviendo hacia el lado izquierdo"
                elif BotonIRSeleccionado == 4:   # Botón presionado: derecha
                        PotenciaMotorIzq = 50           # Potencia motor izquierdo
                        PotenciaMotorDer = 0            # Potencia motor derecho
                        Mensaje1 = "Robot moviendo hacia el lado derecho"
                else:                                           #    Ningún    botón
presionado
                        PotenciaMotorIzq = 0            # Potencia motor izquierdo
                        PotenciaMotorDer = 0            # Potencia motor derecho
                        Mensaje1 = "Robot parado"
                Mensaje2 = "Datos robot:"
                Mensaje3 = str("- Posición encoder izquierdo:"+str(PosEncIzq))
        # Posición actual del encoder del motor izquierdo
                Mensaje4 = str("- Posición encoder derecho:"+str(PosEncDer))
        # Posición actual del encoder del motor derecho
                Mensaje5 = str("- Orientación eje Z del giróscopo:"+str(GyroEjeZ))
        # Orientación del giróscopo (eje Z)
                Mensaje6 = str("- Diametro rueda robot (mm):"+str(DiametroRueda))
        # Diámetro de la rueda del robot
                if 'NumCiclos' in VariableAuxiliar:
                                # Checkear que existe la variable NumCiclos
                        NumCiclosActual = VariableAuxiliar['NumCiclos']+1
                # Aumentar número de ciclos 1 unidad
                Mensaje7 = str("- Número de ciclos:"+str(NumCiclosActual))
        # Indica el número de ciclos en ejecución
                Mensaje                                                         =
Mensaje1+'\n'+Mensaje2+'\n'+Mensaje3+'\n'+Mensaje4+'\n'+Mensaje5+'\n'+Mensaje6
+'\n'+Mensaje7
        else:
                Mensaje = "Batería baja. CARGAR!"

# SALVAR DATOS EN EL CONTEXTO
ComunicacionGoPiGo3['POTENCIAMOTORIZQ'] = PotenciaMotorIzq #   Potencia   motor
izquierdo
ComunicacionGoPiGo3['POTENCIAMOTORDER'] = PotenciaMotorDer #   Potencia   motor
derecho
ComunicacionGoPiGo3['RESETENCODERS'] = ResetEncoders       #   Reset   encoders
(izquierdo y derecho)
ComunicacionGoPiGo3['MENSAJE'] = Mensaje                                    #
Mensaje a mostrar en el Data logger
ComunicacionGoPiGo3['Ciclo_PrimerCiclo'] = IniciarPrograma #  Primer  ciclo  del
código usuario
```

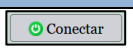
```
VariableAuxiliar['NumCiclos'] = NumCiclosActual                          #
Guardar el valor del número de ciclos para utilizar en la siguiente ejecución
del programa

# DEVOLVER DATOS
print(json.dumps(ComunicacionGoPiGo3))
```

11. Anexo C

El elemento RIP proporciona tres métodos: connect, set y get. En la *Tabla 8* se muestra el código implementado en los elementos de la interfaz web EjsS que utilizan el elemento RIP (en este caso el elemento RIP se llama *RIP*).

Nombre del elemento	Nombre del panel	Imagen	Tipo de acción	Código implementado

BotonConectar	CONEXIÓN		OnClick	//Establecer conexión
				RIP.connect();
				//Iniciar simulación
				_play();
				//Obtener características del Robot GoPiGo3
				RIP.get(["connect"], function(result) {
				[Conectado,Fabricante,ReferenciaPlaca,NumeroSerie,Version_fw,Version_hw] = result[0];
				});
				//Inicializar variables
				ResetVariables();

BotonDesconectar	CONEXIÓN		OnClick	//Parar robot
				RIP.set(["setPotenciaMotorIzq","setPotenciaMotorDer"],[0,0]);
				//Inicializar variables
				ResetVariables();
				//Terminar ejecución del Robot GoPiGo3
				RIP.get(["disconnect"], function(result) {
				[Conectado,Fabricante,ReferenciaPlaca,NumeroSerie,Version_fw,Version_hw] = result[0];
				});
				//Parar simulación
				_pause(); //Parar la simulación
				_reset(); //Resetear la simulación

PanelModoManual	MODOS DE FUNCIONAMIENTO		OnClick	if (ModoManual == false){
				//Robot en modo Manual
				RIP.set(["setModoControl"],["Manual"]);
				ModoManual = true
				ModoAutomatico = false
				//Inicializar variables
				//Resetear contraseña para constantes del programa Balance Bot
				VisualizarCteBalanceBot = ""
				_model.getView().ValorPanelContraseñaCte.setValue("");
				//Inicializar constantes del programa Balance Bot
				CTE_KGYROANGLE = 15
				CTE_KGYROSPEED = 1.7
				CTE_KPOS = 0.07
				CTE_KSPEED = 0.1
				CTE_KDRIVE = -0.02
				CTE_KSTEER = 0.25
				CTE_KVOLTAGE = 0.083
				CTE_DRIVESPEED = 175
				CTE_STEERSPEED = 100
				CTE_TIMEFALLLIMIT = 2
				//Parar ejecución de programas
				ProgramaBalanceBotIniciado = false
				ProgramaUsuarioIniciado = false
				ControlRemotoIR = 0
				}

PanelModoAutomatico	MODOS DE FUNCIONAMIENTO		OnClick	if (ModoAutomatico == false){
				//Robot en modo Automático
				RIP.set(["setModoControl"],["Automatico"]);
				ModoAutomatico = true
				ModoManual = false
				//Inicializar variables
				//Parar movimientos manuales
				RobotEnMovimiento = false
				//Potencia motor izquierdo = 0
				PotenciaMotorIzq = 0
				_model.getView().NuevoValorPanelParametrosManualPotenciaMotorIzquierdo.setValue(PotenciaMotorIzq);
				//Potencia motor derecho = 0
				PotenciaMotorDer = 0
				_model.getView().NuevoValorPanelParametrosManualPotenciaMotorDerecho.setValue(PotenciaMotorDer);
				//Velocidad manual robot = 300
				NuevaVelocidadRobot = 300
				_model.getView().NuevoValorPanelParametrosManualVelocidadTeorica.setValue(NuevaVelocidadRobot);
				}

PanelBotonManualIzquierda	MOVIMIENTOS MANUALES		OnClick	//Mover robot hacia el lado izquierdo
				RIP.set(["setMovimientoManual"],[3]);
				RobotEnMovimiento = true

PanelBotonManualAdelante	MOVIMIENTOS MANUALES		OnClick	//Mover robot hacia adelante
				RIP.set(["setMovimientoManual"],[1]);
				RobotEnMovimiento = true

PanelBotonManualDerecha	MOVIMIENTOS MANUALES		OnClick	//Mover robot hacia el lado derecho
				RIP.set(["setMovimientoManual"],[4]);
				RobotEnMovimiento = true

PanelBotonManualGiroIzquierda	MOVIMIENTOS MANUALES		OnClick	//Girar robot hacia el lado izquierdo
				RIP.set(["setMovimientoManual"],[5]);
				RobotEnMovimiento = true

PanelBotonManualParar	MOVIMIENTOS MANUALES		OnClick	//Parar robot
				RIP.set(["setMovimientoManual"],[7]);
				RobotEnMovimiento = false

PanelBotonManualGiroDerecha	MOVIMIENTOS MANUALES		OnClick	//Girar robot hacia el lado derecho
				RIP.set(["setMovimientoManual"],[6]);
				RobotEnMovimiento = true

PanelBotonManualAtras	MOVIMIENTOS MANUALES		OnClick	//Mover robot hacia atrás
				RIP.set(["setMovimientoManual"],[2]);
				RobotEnMovimiento = true

Boton1ManualResetearEncoders	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Resetear encoders del robot
				RIP.set(["setResetRobotEncoders"],[0]);

Boton2ManualResetearEncoders	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Resetear encoders del robot
				RIP.set(["setResetRobotEncoders"],[0]);

BotonManualSetPotenciaMotorIzquierdo	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Set potencia motor izquierdo
				RIP.set(["setPotenciaMotorIzq"],[PotenciaMotorIzq]);

BotonManualResetPotenciaMotorIzquierdo	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Reset potencia motor izquierdo
				PotenciaMotorIzq = 0
				_model.getView().NuevoValorPanelParametrosManualPotenciaMotorIzquierdo.setValue(PotenciaMotorIzq);
				RIP.set(["setPotenciaMotorIzq"],[PotenciaMotorIzq]);

BotonManualSetPotenciaMotorDerecho	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Set potencia motor derecho
				RIP.set(["setPotenciaMotorDer"],[PotenciaMotorDer]);

BotonManualResetPotenciaMotorDerecho	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Reset potencia motor derecho
				PotenciaMotorDer = 0
				_model.getView().NuevoValorPanelParametrosManualPotenciaMotorDerecho.setValue(PotenciaMotorDer);

				RIP.set(["setPotenciaMotorDer"],[PotenciaMotorDer]);

BotonManualSetVelocidadTeorica	PARÁMETROS ROBOT (MODO MANUAL)		OnClick	//Set velocidad teórica manual del robot
				RIP.set(["setRobotVeloManual"],[NuevaVelocidadRobot]);

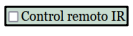
Boton1AutoReseteaerEncoders	PARÁMETROS ROBOT (MODO AUTOMÁTICO)		OnClick	//Resetear encoders del robot
				RIP.set(["setResetRobotEncoders"],[0]);

Boton2AutoReseteaerEncoders	PARÁMETROS ROBOT (MODO AUTOMÁTICO)		OnClick	//Resetear encoders del robot
				RIP.set(["setResetRobotEncoders"],[0]);

BotonAutoSetConstantesBalanceBot	PROGRAMAS ROBOT BALANCE BOT		OnClick	//Definir las constantes para el programa Balance Bot
				RIP.set(["setKGYROANGLE","setKGYROSPEED","setKPOS","setKSPEED","setKDRIVE","setKSTEER","setKVOLTAGE","setDRIVESPEED","setSTEERSPEED","setTIMEFALLLIMIT"],[CTE_KGYROANGLE,CTE_KGYROSPEED,CTE_KPOS,CTE_KSPEED,CTE_KDRIVE,CTE_KSTEER,CTE_KVOLTAGE,CTE_DRIVESPEED,CTE_STEERSPEED,CTE_TIMEFALLLIMIT]);

BotonStartBalanceBot	PROGRAMAS ROBOT BALANCE BOT		OnClick	//Iniciar programa Balance Bot
				RIP.set(["setStartBalanceBot"],[0]);
				ProgramaBalanceBotIniciado = true

BotonStopBalanceBot	PROGRAMAS ROBOT BALANCE BOT		OnClick	//Parar programa Balance Bot
				RIP.set(["setStopBalanceBot"],[0]);
				ProgramaBalanceBotIniciado = false
				//Desactivar Control Remoto IR
				ControlRemotoIR = 0
				RIP.set(["setControlRemotoIR"],[ControlRemotoIR]);

CheckBoxControlRemoto	PROGRAMAS ROBOT BALANCE BOT		OnChange	//Activar/Desactivar Control Remoto IR
				RIP.set(["setControlRemotoIR"],[ControlRemotoIR]);

WrappedPanelBotonAdelante	PROGRAMAS ROBOT BALANCE BOT		OnPress	//Mover robot hacia adelante
				RIP.set(["setMovControlWebIR"],[1]);

			OnRelease	//Parar robot
				RIP.set(["setMovControlWebIR"],[0]);

WrappedPanelBotonIzquierda	PROGRAMAS ROBOT BALANCE BOT		OnPress	//Mover robot hacia el lado izquierdo
				RIP.set(["setMovControlWebIR"],[2]);

			OnRelease	//Parar robot
				RIP.set(["setMovControlWebIR"],[0]);

WrappedPanelBotonOK	PROGRAMAS ROBOT BALANCE BOT		OnPress	// Botón OK seleccionado
				RIP.set(["setMovControlWebIR"],[3]);

			OnRelease	//Parar robot
				RIP.set(["setMovControlWebIR"],[0]);

WrappedPanelBotonDerecha	PROGRAMAS ROBOT BALANCE BOT		OnPress	// Mover robot hacia el lado derecho
				RIP.set(["setMovControlWebIR"],[4]);

			OnRelease	//Parar robot
				RIP.set(["setMovControlWebIR"],[0]);

WrappedPanelBotonAtras	PROGRAMAS ROBOT BALANCE BOT		OnPress	// Mover robot hacia atrás
				RIP.set(["setMovControlWebIR"],[5]);

			OnRelease	//Parar robot
				RIP.set(["setMovControlWebIR"],[0]);

PanelBotonGuardarArchivo	PROGRAMAS ROBOT EDITOR CÓDIGO USUARIO		OnClick	//Nombre del archivo (default name)
				if (NombreArchivoUsuario == ""){
				NombreArchivoUsuario = "CodigoUsuario.py"
				}
				//Guardar archivo
				_tools.downloadText(NombreArchivoUsuario,CodigoUsuario)
				//Visualizar mensaje en el Visor de Avisos
				ManualMsg = "Guardar archivo del usuario"
				Msg = ManualMsg+'\n'+Msg

BotonAbrirArchivo	PROGRAMAS ROBOT EDITOR CÓDIGO USUARIO		OnClick	//Abrir archivo
				readFile();
				//Visualizar mensaje en el Visor de Avisos
				ManualMsg = "Abrir archivo del usuario"
				Msg = ManualMsg+'\n'+Msg

BotonCargarArchivo	PROGRAMAS ROBOT EDITOR		OnClick	//Cargar y ejecutar el código creado por el usuario
				RIP.set(["setStartScript"],[CodigoUsuario]);

	CÓDIGO USUARIO			ProgramaUsuarioIniciado = true

BotonPararScript	PROGRAMAS ROBOT EDITOR CÓDIGO USUARIO		OnClick	//Parar de ejecutar el código creado por el usuario
				RIP.set(["setStopScript"],[0]);
				ProgramaUsuarioIniciado = false

Tabla 8: *Elementos de la interfaz web EjsS que utilizan el elemento RIP*

En los elementos de la *Tabla 9* aunque no se utilice el elemento RIP se implementa un código.

Nombre del elemento	Nombre del panel	Imagen	Tipo de acción	Código implementado

ValorPanelContraseñaCte	PROGRAMAS ROBOT BALANCE BOT		OnChange	//Leer la contraseña introducida por el usuario y guardar en la variable "ContraseñaCteBalanceBot"
				ContraseñaCteBalanceBot = _model.getView().ValorPanelContraseñaCte.getValue();
				//Visualización de las constantes para el programa Balance Bot
				if (ContraseñaCteBalanceBot == "UNED") {
				VisualizarCteBalanceBot = true
				}
				else {
				VisualizarCteBalanceBot = false
				}

ScriptCodigoUsuario	PROGRAMAS ROBOT EDITOR CÓDIGO USUARIO		On Change	//Leer el código introducido por el usuario y guardar en la variable "CodigoUsuario"
				CodigoUsuario = _model.getView().ScriptCodigoUsuario.getValue();

PanelBotonAutoBorrarCodigoUsuario	PROGRAMAS ROBOT EDITOR CÓDIGO USUARIO		OnClick	//Borrar código creado por el usuario
				CodigoUsuario = ""
				_model.getView().ScriptCodigoUsuario.setValue(CodigoUsuario);
				//Visualizar mensaje en el Visor de Avisos
				ManualMsg = "Borrar código creado por el usuario"
				Msg = ManualMsg+'\n'+Msg

PanelBotonBorrarMensajesVisorDeAvisos	VISOR DE AVISOS		OnClick	//Borrar mensajes del Visor de Avisos
				Msg = ""

Tabla 9: *Elementos de la interfaz web EjsS con código implementado*

Por otra parte, tal y como se ha visto en el apartado *3.4.1 Interfaz de usuario de EjsS*, el panel de trabajo *Model* se dedica al proceso de modelado. En la pestaña *Evolution* se ha definido la llamada a los algoritmos que describen el modelo (ver *Figura 58*). Estos algoritmos (que se desarrollan en la pestaña *Custom*) se ejecutarán cíclicamente y sirven para actualizar el valor de algunas variables definidas en el programa de EjsS.

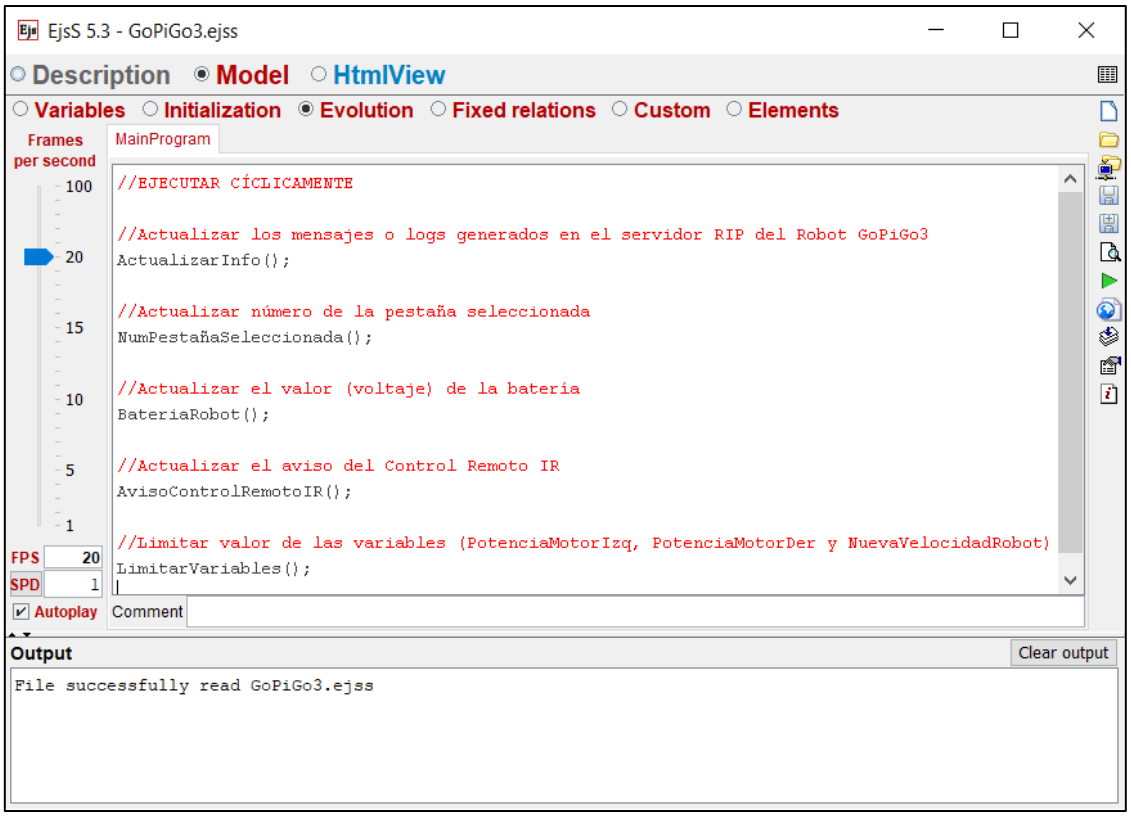


Figura 58: *Llamada a los algoritmos que describen el modelo (pestaña Evolution)*

A continuación, se describe brevemente la tarea de cada algoritmo:

- **ActualizarInfo()**: este algoritmo actualiza los mensajes o logs generados en el servidor RIP del robot GoPiGo3 (ver *Código 9*). Los mensajes se visualizan en el visor de avisos de la aplicación EjsS. Esto permite al usuario obtener información del robot, facilitando el control del mismo.

```
//Función para actualizar los mensajes o logs generados en el servidor RIP del Robot GoPiGo3
function ActualizarInfo() {
 if (Conectado) {
  if (NewMsg != OldMsg) {
   Msg = NewMsg+'\n'+Msg  //Actualizar mensaje a visualizar en el Visor de Avisos
   OldMsg = NewMsg
  }
 }
}
```

Código 9: *Función ActualizarInfo()*

- **NumPestañaSeleccionada()**: en la aplicación EjsS se ha añadido un separador de pestañas (*TabbedPanel)*, de modo que cuando se elige el modo de funcionamiento manual () se visualiza un contenido y cuando se selecciona el modo de funcionamiento automático (

) otro contenido diferente. Mediante esta función (ver *Código 10*), se gestiona la visualización de una pestaña u otra dependiendo del modo de funcionamiento seleccionado por el usuario en la interfaz web.

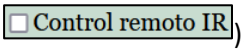
```
//Función para gestionar pestaña de funcionamiento (manual/automático)
function NumPestañaSeleccionada() {
 if (ModoManual == true) {
  PestañaSeleccionada = 0  //Pestaña seleccionada: manual
 }
 else {
  if (ModoAutomatico == true) {
   PestañaSeleccionada = 1  //Pestaña seleccionada: automático
  }
 }
}
```

Código 10: *Función NumPestañaSeleccionada()*

- **BateriaRobot()**: dependiendo del valor del voltaje de la batería, el indicador visual de batería puede tener el color verde (), amarillo () o rojo (). Este algoritmo gestiona el color de dicho visualizador (ver *Código 11*). Además, activa/desactiva la variable que se utiliza para visualizar el mensaje "Voltaje menor que 9V. CARGAR!".

```
//Función para gestionar el valor (voltaje) de la batería
function BateriaRobot() {
 if (ValorBateria < 9) {
  BateriaBaja = true    //Aviso batería baja (mostrar mensaje batería robot NOK)
  ColorBateria = "Red"  //Visualización batería (barra de color)
 }
 else {
  if (ValorBateria >= 9 && ValorBateria <= 9.5) {
   BateriaBaja = false
   ColorBateria = "Yellow"
  }
  else{
   if (ValorBateria > 9.5) {
    BateriaBaja = false
    ColorBateria = "Green"
   }
  }
 }
}
```

Código 11: *Función BateriaRobot()*

- **AvisoControlRemotoIR()**: cuando se está ejecutando el programa Balance Bot, el usuario tiene la opción de conducir el robot remotamente mediante el mando a distancia por infrarrojos o utilizando los botones (flechas) habilitados en la interfaz web. Existe una opción para activar/desactivar el control remoto IR (

). En caso de que se active esta opción, se utilizará el mando a distancia por infrarrojos para conducir el robot. Y cuando se desactive, se usan los botones o flechas de la aplicación informática. Mediante esta función (ver *Código 12*) se muestra un aviso que le indica al usuario que tipo de control tiene que utilizar para la conducción del robot, o el mando a distancia por infrarrojos, o los botones (flechas).

```
//Función para gestionar el aviso del Control Remoto IR
function AvisoControlRemotoIR() {
 if (ControlRemotoIR == true) {
  AvisoControlRemoto = "Utilizar mando a distancia"
 }
 else {
  if (ControlRemotoIR == false) {
   AvisoControlRemoto = "Utilizar las siguientes flechas"
  }
 }
}
```

Código 12: *Función AvisoControlRemotoIR()*

- **LimitarVariables()**: esta función limita el valor de las variables *PotenciaMotorIzq*, *PotenciaMotorDer* y *NuevaVelocidadRobot* (ver *Código 13*). Estas variables se utilizan para el control manual del robot:
  - **PotenciaMotorIzq**: define la potencia del motor izquierdo para el modo de funcionamiento manual. Este algoritmo limita el valor de la variable entre -100% y 100%. Por lo tanto, el usuario no podrá definir un valor que no esté dentro del rango mencionado.
  - **PotenciaMotorDer**: define la potencia del motor derecho para el modo de funcionamiento manual. Este algoritmo limita el valor de la variable entre -100% y 100%. Por lo tanto, el usuario no podrá definir un valor que no esté dentro del rango mencionado.
  - **NuevaVelocidadRobot**: define la velocidad de los motores cuando se realizan los movimientos manuales del robot mediante el panel de control manual (flechas) de la interfaz web. Este algoritmo limita el valor de la variable entre 0 DPS y 1000 DPS. Por lo tanto, el usuario no podrá definir un valor que no esté dentro del rango mencionado.

```
//Función para limitar valor de las variables (PotenciaMotorIzq, PotenciaMotorDer y NuevaVelocidadRobot)
function LimitarVariables() {
 //Limitar potencia motor izquierdo entre -100 y 100
 PotenciaMotorIzq = _model.getView().NuevoValorPanelParametrosManualPotenciaMotorIzquierdo.getValue();
 if (PotenciaMotorIzq < -100) {
  PotenciaMotorIzq = -100
_model.getView().NuevoValorPanelParametrosManualPotenciaMotorIzquierdo.setValue(PotenciaMotorIzq);
 }
 if (PotenciaMotorIzq > 100) {
  PotenciaMotorIzq = 100
  _model.getView().NuevoValorPanelParametrosManualPotenciaMotorIzquierdo.setValue(PotenciaMotorIzq);
 }
 //Limitar potencia motor derecho entre -100 y 100
 PotenciaMotorDer = _model.getView().NuevoValorPanelParametrosManualPotenciaMotorDerecho.getValue();
 if (PotenciaMotorDer < -100) {
  PotenciaMotorDer = -100
  _model.getView().NuevoValorPanelParametrosManualPotenciaMotorDerecho.setValue(PotenciaMotorDer);
 }
 if (PotenciaMotorDer > 100) {
  PotenciaMotorDer = 100
  _model.getView().NuevoValorPanelParametrosManualPotenciaMotorDerecho.setValue(PotenciaMotorDer);
 }
 //Limitar velocidad manual del robot entre 0-1000 DPS
 NuevaVelocidadRobot = _model.getView().NuevoValorPanelParametrosManualVelocidadTeorica.getValue();
 if (NuevaVelocidadRobot < 0) {
  NuevaVelocidadRobot = 0
  _model.getView().NuevoValorPanelParametrosManualVelocidadTeorica.setValue(NuevaVelocidadRobot);
 }
 if (NuevaVelocidadRobot > 1000) {
  NuevaVelocidadRobot = 1000
  _model.getView().NuevoValorPanelParametrosManualVelocidadTeorica.setValue(NuevaVelocidadRobot);
 }
}
```

Código 13: *Función LimitarVariables()*

Además de los algoritmos mencionados anteriormente, en la pestaña *Custom* se han desarrollado otros dos algoritmos:

- **readFile()**: esta función (ver *Código 14*) sirve para abrir archivos locales del PC del usuario. Al abrir el archivo se cargará automáticamente el contenido de este en el editor de código del usuario de la interfaz web que sirve para desarrollar los programas de control del usuario.

```
//Función para leer (abrir) archivos locales
function readFile() {
 _tools.uploadText(function(text) {
  CodigoUsuario = text  //Copiar en la variable "CodigoUsuario" el archivo local abierto
  _model.getView().ScriptCodigoUsuario.setValue(CodigoUsuario);
 })
}
```

Código 14: *Función readFile()*

- **ResetVariables()**: esta función (ver *Código 15*) sirve para inicializar el valor de algunas variables definidas en el programa de EjsS.

```
//Función para resetear variables
function ResetVariables() {
 ModoManual = true
 ModoAutomatico = false
 RobotEnMovimiento = false
 PotenciaMotorIzq = 0
 _model.getView().NuevoValorPanelParametrosManualPotenciaMotorIzquierdo.setValue(PotenciaMotorIzq);
 PotenciaMotorDer = 0
 _model.getView().NuevoValorPanelParametrosManualPotenciaMotorDerecho.setValue(PotenciaMotorDer);
 NuevaVelocidadRobot = 300
 _model.getView().NuevoValorPanelParametrosManualVelocidadTeorica.setValue(NuevaVelocidadRobot);
 VisualizarCteBalanceBot = ""
 _model.getView().ValorPanelContraseñaCte.setValue("");
 CTE_KGYROANGLE = 15
 CTE_KGYROSPEED = 1.7
 CTE_KPOS = 0.07
 CTE_KSPEED = 0.1
 CTE_KDRIVE = -0.02
 CTE_KSTEER = 0.25
 CTE_KVOLTAGE = 0.083
 CTE_DRIVESPEED = 175
 CTE_STEERSPEED = 100
 CTE_TIMEFALLLIMIT = 2
 ProgramaBalanceBotIniciado = false
 ProgramaUsuarioIniciado = false
 ControlRemotoIR = 0
 NombreArchivoUsuario = ""
 CodigoUsuario = ""
 _model.getView().ScriptCodigoUsuario.setValue(CodigoUsuario);
 Msg = ""
}
```

Código 15: *Función ResetVariables()*