UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

Framework de comunicaciones para robótica educativa, distribuida y colaborativa

Cristian Vázquez Cordero.

Mayo, 2018.

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

Framework de comunicaciones para robótica educativa, distribuida y colaborativa

Autor: Cristian Vázquez Cordero

Tutor: José Moreno del Pozo

Tribunal Calificador:

Presidente: Pilar Bachiller Burgos

Secretario: Pablo Bustos García de Castro

Vocal: Pedro Manuel Núñez Trujillo

AGRADECIMIENTOS

Después de un intenso período de diez meses, hoy es el día que finalizo este TFE mediante este documento y con ello el grado de Ingeniería Informática en Ingeniería de Computadores. Han sido meses de investigación y dedicación a un proyecto que considero con futuro, meses de duro trabajo, meses dónde he crecido tanto profesional como personalmente, meses dónde he conocido nuevas tecnologías y por supuesto, grandes personas. Por ello, y por todo el trato recibido, me gustaría agradecer a todas las personas que han estado presentes, de una forma u de otra, en este proyecto.

Quiero agradecer al Laboratorio de Robótica y Visión Artificial de la Universidad de Extremadura (Robolab), por todo el apoyo recibido y la aceptación personal que he tenido dentro del mismo. Sois más que un grupo de investigación, sois un grupo de profesionales, expertos, especialistas y, por supuesto, amigos unidos por aquello que os apasiona. Un grupo de persona que luchan diariamente para conseguir avances que puedan ayudar a la sociedad. Particularmente me gustaría nombrar a mi tutor, José Moreno del Pozo y a Francisco Marcelino Andrés Hernández, por vuestra valiosa ayuda y dedicación desinteresada hacia mi persona. Me habéis tendido la mano, brindado la oportunidad de participar en esto, ayudado en todo lo posible y más, gracias. Siempre estaré enormemente agradecido.

Para finalizar, agradecer también el apoyo a mis padres, a mi novia y mis amigos, no solo durante estos meses sino durante estos cuatro años que hoy culminan.

¡Muchas gracias a todos!

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La **robótica** es uno de los ámbitos más complejos de entender y desarrollar para la gran mayoría de la sociedad. Esta disciplina comprende campos de diferentes ingenierías, como son Ingeniería mecatrónica, Ingeniería eléctrica, Ingeniería electrónica, Ingeniería biomédica y ciencias de la computación. Abarca el diseño, la construcción, la operación y la creación de robots, así como el desarrollo del software del mismo.

Por todo ello pensamos que es de interés común acercar la robótica a la sociedad, esto es, crear un medio para acercar la robótica a cualquier persona, más allá de los expertos de las diferentes ingenierías y ciencias que participan en ello.

Para satisfacer el deseo de aquellas personas con curiosidad por la robótica existe la **robótica educativa**.

La robótica educativa es un medio de aprendizaje en el cual participan estas personas con curiosidad por la robótica, por el diseño y la construcción de robots propios. Además, también es una forma pedagógica para que estudiantes se acerquen al complejo mundo de la robótica y de otras disciplinas horizontales tan dispares como las matemáticas, la física o el aprendizaje de lenguas extranjeras.

Este trabajo nace en el ámbito del Laboratorio de Robótica y Visión Artificial de la Universidad de Extremadura (en adelante **Robolab**) y más explícitamente en el proyecto denominado "**PYRO4BOT**".

PYRO4BOT es un entorno de trabajo (*framework*) para el desarrollo de robots de forma sencilla, abierta y apto para todos los públicos, actualmente se encuentra en fase de desarrollo y está escrito en el lenguaje de programación Python ®. Este *framework* debe permitir al usuario el despliegue de un sistema robótico de una forma amigable y cómoda en un tiempo razonablemente corto. En definitiva, si se dispone de una base robótica, un conjunto de sensores y actuadores conectados conforme a sus especificaciones, el usuario, casi de forma independiente a sus conocimientos previos (excepto un nivel básico de programación) debe hacer uso del robot de forma rápida y sencilla.

En este **trabajo de fin de estudios** (TFE) se propone la ampliación del *framework* para, en primer lugar, mejorar sus capacidades y posteriormente mediante la creación de un ***Middleware* de comunicaciones** poder realizar comunicaciones con otros robots, es decir, ofrecer a PYRO4BOT la posibilidad de realizar robótica distribuida y colaborativa.

ABSTRACT

Robotics is the most complex objective to understand and develop for the majority of society. This discipline includes fields of different engineering, such as Mechanical Engineering, Electrical Engineering, Electronic Engineering, Biomedical Engineering and Computer Science. It covers the design, construction, operation and creation of robots, as well as the development of the software itself.

For all this we think it is of common interest to bring robotics to society, that is, to create a means to bring robotics closer to anyone, beyond the experts of the different engineering and science involved in it.

To satisfy the desire of those people with a curiosity for robotics, there is **educational robotics.**

Educational robotics is a learning medium in which these people participate with curiosity for robotics, for the design and construction of their own robots. In addition, it is also a pedagogical way for students to approach the complex world of robotics and other horizontal disciplines as diverse as mathematics, physics or foreign language learning.

This work was born in the field of Robotics and Artificial Vision Laboratory of the University of Extremadura (hereinafter **Robolab**) and more explicitly in the project called "**PYRO4BOT**".

PYRO4BOT is a working environment (framework) for the development of robots in a simple, open and suitable for all audiences, is currently in development phase and is written in the programming language Python ®. This framework should allow the user to deploy a robotic system in a friendly and comfortable way in a reasonably short time. In short, if you have a robotic base, a set of sensors and actuators connected according to your specifications, the user, almost independently of their previous knowledge (except a basic level of programming) should make use of the robot quickly and simple.

In this end-of-studies project (**TFE**) we propose the extension of the framework to, firstly, improve its capabilities and subsequently by creating a communications Middleware to be able to communicate with other robots, that is, offer PYRO4BOT the possibility to perform distributed and collaborative robotics.

1 INTRODUCCIÓN

En pleno siglo XXI, cuando se escuchan palabras como “robots, dron, autómatas…” causan extrañeza y lejanía (cada vez menos) a la gran mayoría de la sociedad. Una sociedad dominada cada vez más por **entidades autónomas e inteligentes**, una sociedad que usa continuamente esta “potencial” inteligencia artificial sin ser consciente de ello, no puede permitirse que al escuchar estas palabras sientan rareza por su desconocimiento. Por ello, el presente documento realiza un intento de acercar estos términos extraordinarios al seno de la sociedad mundial.

Vivimos en el siglo donde gracias a la **robótica** una persona puede ser operada en un tiempo ínfimo y con una precisión sobrehumana, el siglo dónde tareas que eran realizadas por el ser humano comienzan a ser llevadas a cabo por **máquinas autónomas o semiautónomas**. La sociedad debe poseer la capacidad de conocer qué son y cómo funcionan estas máquinas.

El objetivo del presente documento no es la realización ni de la aplicación de ninguna de las tecnologías anteriormente nombradas, sino acercar o poder acercar a **la sociedad el ámbito de la robótica.**

Otorgar a un ente de inteligencia es la primera de las tareas a desarrollar cuando se diseña un sistema robótico, pero, ¿Y si parte de su inteligencia estuviera fuera de él?

En este TFE se plantea la posibilidad de realizar **robótica distribuida**, es decir, de poder tener “parte” del robot fuera de él, ya que hay ocasiones en las cuales una tarea es

demasiado compleja para realizarla en un robot de tamaño reducido y se precisa de una máquina de alto rendimiento y de también gran envergadura, por ello, es obligado "distribuir" este cómputo entre el robot y el equipo de altas prestaciones.

Otra necesidad añadida sería el hacer uso de equipos externos que no puedan estar físicamente conectados al dispositivo, como por ejemplo sensores de temperatura, presión atmosférica o de movimiento.

Figura 1: Robot acompañante de Toyota (Wikipedia, 2015)

Además del planteamiento de uso de la **robótica a nivel individual y robótica distribuida**, se plantea la posibilidad de otorgar **inteligente colectiva** a un enjambre de robots trabajando para conseguir un objetivo común, es decir, lo que se denomina **robótica colaborativa**.

¿Y si queremos dotar a nuestro enjambre de robots de la capacidad de comunicar a todos cuando ocurre un evento? Esta comunicación será una de las tareas de estudio y desarrollo a lo largo de este TFE.

La importancia que está tomando la tecnología en nuestras vidas y la rapidez con la que ésta avanza, hace que ésta tecnología se convierta en una parte integral del proceso de formación de niños y jóvenes. Es importante desarrollar propuestas en la que se ofrezcan a este colectivo interactuar con estas nuevas tecnologías, mediante el manejo de las mismas.

Por ello, creemos fuertemente en el uso de la **robótica con carácter educativo**. El uso de la robótica en las aulas, el aprender a crear robots para el futuro y el hacer uso de robots para aprender otras asignaturas son tareas a tener en cuenta para la realización de este TFE.

2
OBJETIVOS

En base a un prototipo muy básico de software de control de robots con habilidades muy restringidas, se plantean los siguientes objetivos generales para el presente trabajo fin de estudios:

- Mejorar el entorno de trabajo básico y desarrollar un nuevo *framework* para programación de robots de forma sencilla (PYRO4BOT), haciendo hincapié en las siguientes cuestiones:
  - Multiplataforma.
  - Escalabilidad.
  - Robustez.
- Permitir el establecimiento de comunicaciones con otros dispositivos.
- Permitir la realización de robótica colaborativa.
- Crear un entorno de trabajo que permita ser usado a todos los niveles:
  - Sin conocimientos de informática ni de programación.
  - Con conocimientos básicos de programación.
  - Con conocimientos avanzados de programación.
- Creación de un conjunto de ejemplos estables para el uso de la comunidad robótica

3

ESTADO DEL ARTE

Antes de llevar a cabo el desarrollo de este TFE se ha realizado una revisión del estado actual del arte en lo que atañe a la robótica educativa o robótica pedagógica, una disciplina que tiene por objeto la concepción, creación y puesta en funcionamiento de prototipos robóticos y programas especializados con fines pedagógicos.

A continuación, se detallan:

- Productos de mercado.
- Lenguajes.
- Técnicas de comunicaciones.
- Sistemas operativos especializados.

3.1 mBlock

Es un software basado en **scratch**, un lenguaje de programación visual desarrollado por el MIT Lab Tab, que permite una determinada programación de robots basados en placas de Arduino. [1]

Este software tiene muchas limitaciones, entre ellas la imposibilidad de comunicar un robot con otro, lo que lo hace inviable para nuestra propuesta.

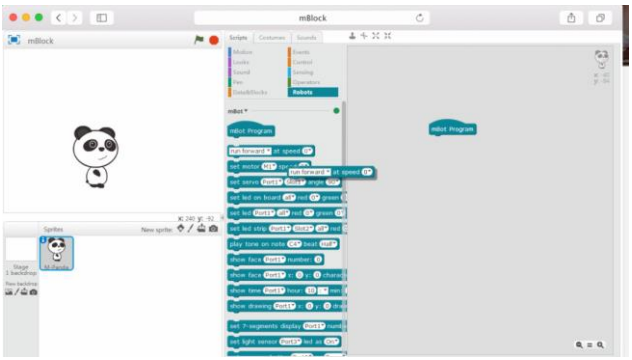

Figura 2: Mblock

Permite la programación de pequeños robots prefabricados (denominados mBot) y poder manejar en cierta manera algunos de sus sensores, pero, no permite comunicaciones entre robots, así como el uso de dispositivos en tiempo real o programación con cierta complejidad, dado su muy bajo nivel.

3.2 LEGO

LEGO Group A/S es una empresa y marca de juguetes danesa reconocida principalmente por sus bloques de plástico interconectables, que poco a poco derivó sus productos a la fabricación de robots móviles simples para el público infantil.

LEGO tiene una serie de robots en venta preparados para funcionar. Además, existen distintos kits de montaje de robots. Usan un software propio denominado *LEGO Mindstorms EV3* disponible para Windows/Mac. Además de tener apps para Android/IOs. [2]

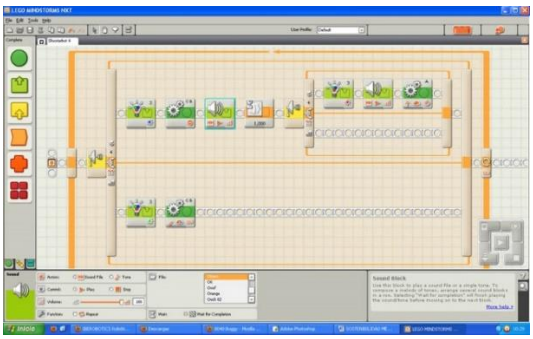

Figura 3: LEGO Software

Sin duda alguna es un proyecto muy consolidado y avanzado, pero, el gran problema es que su software es **privativo** y exclusivo a robots propios de la marca. En consecuencia,

no permite hacer crecer a un robot tanto como el usuario quiera y, además, la adquisición de los robots LEGO conlleva un precio mucho más alto que un robot convencional.

Puedes manejar un robot de LEGO y programarlo mediante un software similar a scratch pero tiene muchas limitaciones.

3.3 OpenQbo Robotic

Partiendo de conocimientos más avanzados, OpenQbo Robotic, es una **distribución de *Linux* basada en *Ubuntu y ROS*** que actualmente se encuentra en fase de desarrollo. Está creada casi de forma exclusiva a la programación de su propio robot, "Q.bo One". [3] [4]

Es un software más avanzado que los dos anteriores, permite realizar actividades de carácter avanzando, pero de nuevo tiene limitaciones. En este caso, está destinado exclusivamente a la programación de un robot, aunque permite cualquier tipo de programación, la finalidad es la programación de su robot propio.

Además, al ser un software avanzando, conlleva una curva de aprendizaje bastante alta en comparación a lo que se pretende realizar.

Dejando a un lado la dificultad que esto conlleva, la otra gran limitación es el coste del producto. En torno a 500$ por robot son cifras que multiplican hasta en 7 veces el coste de unos de nuestros robots basados en PYRO4BOT.

3.4 Programación basada en componentes

La **programación orientada a componentes** (que también es llamada basada en componentes) es una rama de la ingeniería del software, con énfasis en la descomposición de sistemas ya conformados en componentes funcionales o lógicos con interfaces bien definidas usadas para la comunicación entre dichos componentes.

En relación con el desarrollo de software robótico basado en componentes, existen *frameworks* muy robustos y bien desarrollados.

3.4.1. RoboComp

Un gran ejemplo de ello sería ***RoboComp.*** Este *software* puede ser enunciado como:

*"*Un *framework* de programación de código abierto que proporciona herramientas para crear y modificar componentes de software que se comunican a través de interfaces públicas. Los componentes pueden requerir, suscribir, implementar o publicar interfaces de manera transparente al usuario. La construcción de nuevos componentes se realiza utilizando dos idiomas específicos de dominio, IDSL y CDSL. Con IDSL define una interfaz y con CDSL especifica cómo se comunicará el componente con el mundo.

Con esta información, un generador de código crea las fuentes en lenguaje C++ y/o Python, basadas en CMake, que se compilan y ejecutan automáticamente.

Cuando se tienen que cambiar algunas de estas características, el componente se puede regenerar y todo el código específico del usuario se preserva gracias a un mecanismo de herencia." [5]

En este software, **las comunicaciones son realizadas a través de ICE** (siglas en inglés de *Internet Communications Engine*). Ice, es un marco RPC de código abierto desarrollado por ZeroC. Proporciona SDK para C++, C#, Java, JavaScript, MATLAB, Objective-C, PHP, Python y Ruby, y se puede ejecutar en varios sistemas operativos, incluidos Linux, Windows, macOS, iOS y Android. [6]

RoboComp/ICE es un software muy robusto y sofisticado, permite realizar robótica con una complejidad muy alta y unos resultados muy precisos. Sin embargo, conlleva mucha dificultad en su aprendizaje y está destinado solo para expertos en programación.

3.5 Robot Operating System (ROS)

"Robot Operating System (**ROS**, Sistema Operativo para Robots) es un conjunto de bibliotecas de software y herramientas que ayudan a crear aplicaciones robóticas. Desde los controladores hasta los algoritmos de última generación y con potentes herramientas de desarrollo, ROS tiene lo que se necesita para un proyecto de robótica. Y es todo de código abierto." [7]

Desde su lanzamiento en 2007 por Willow Garage en el Laboratorio de Inteligencia Artificial de Stanford hasta hoy en día, es el **software de referencia** en lo que atañe a la **robótica.**

"ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para un sistema UNIX (Ubuntu (Linux)) aunque también se está adaptando a otros sistemas operativos como Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows, considerados como 'experimentales'.

ROS tiene dos partes básicas: la parte del sistema operativo, ros, como se ha descrito anteriormente y ros-pkg, una suite de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados pilas o en inglés stacks) que implementan las funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc." [8]

En lo que respecta a las comunicaciones, ROS se organiza en un grafo de computación. Una red *peer-to-peer* de procesos cuando realizan acciones juntos. Dentro del grafo existen nodos, maestros, servidor de parámetros, mensajes, temas y bolsas.

Cada **nodo es un proceso** que realiza computación dentro del grafo, este nodo se comunica con otros mediante **mensajes.** Estos mensajes es una estructura de datos simple (enteros, punto flotante, booleano, etc.). Los mensajes se **enrutan** a través de un **sistema de publicación / suscripción.** Un nodo toma ese mensaje y lo almacena en un **tema determinado.** Cada tema es identificado por un nombre. Cada nodo interesado en ese tema, se suscribirá y recibirá la información cada vez que ese tema tenga nuevas publicaciones. [9]

3.5.1. Robot Operating System 2 (ROS2)

ROS2 es una **nueva versión** de ROS, actualmente se encuentra en desarrollo. Su propósito es lanzar una versión cada 6 meses y adaptar ROS a todos los cambios que han surgido en la robótica y en la comunidad de ROS desde que éste fue lanzado. [10]

Respecto a sus comunicaciones, ROS2 usa un nuevo middleware de comunicaciones llamado **Servicio de Distribución de Datos** *(Data Distribution Service***, DDS**).

"El servicio de distribución de datos para sistemas es un estándar de grupo de administración de objetos (OMG) de máquina a máquina (**middleware de extremo a extremo**) que tiene como objetivo habilitar intercambios de datos escalables, en tiempo real, confiables, de alto rendimiento e interoperables." [11]

"DDS proporciona un **transporte de publicación-suscripción** que es muy similar al de ROS. DDS utiliza el "Lenguaje de descripción de interfaz (**IDL**)" tal como lo define el Object Management Group (OMG) para la definición y serialización de mensajes. DDS tiene un transporte de estilo de solicitud-respuesta (llamado DDS-RPC).

ROS ha tomado la decisión de usar **DDS en su nueva versión** debido a qué de esta manera tienen menos código que mantener, el comportamiento y las especificaciones exactas ya han sido definidas en la documentación y cuentan con detallados casos de uso. En consecuencia, la comunidad puede revisar, auditar e implementar el middleware en diversos grados de interoperabilidad.

Como inconveniente, al ser un middleware de extremo a extremo, ROS debe de funcionar dentro de ese diseño existente. Si el diseño no contemplase algún caso de uso de ROS estarían en un problema. [12]

3.6 Python

Este trabajo se desarrollará en el lenguaje de programación ***Python***, concretamente la versión 2.7.11 aunque se prevé continuar con el desarrollo de este proyecto posteriormente y actualizar a versiones 3.6.5 o superiores.

Python es un lenguaje de programación **interpretado**, con **tipado dinámico** y **multiplataforma**. Además, es un lenguaje **multiparadigma**, soporta **programación orientada a objetos**, **programación imperativa** y **programación funcional**. Por supuesto, Python es un lenguaje con licencia de **código abierto**. [13] [14]

La filosofía de este lenguaje es **favorecer el uso de un código legible**, ésta, junto al resto de características anteriormente mencionadas hacen de este lenguaje el idóneo para el proyecto a desarrollar.

Además, gracias a la gran aceptación que tiene por parte de la comunidad, a la gran cantidad de **recursos disponibles**, tales como librerías, libros, tutoriales… el aprendizaje de éste es mucho menor que el de otros lenguajes.

Hemos decidido **aplicar la filosofía de Python al proyecto**, es decir, cuanto más sencillo para el usuario final, mejor.

3.7 PYRO

PYRO (***Python Remote Objects)*** es una librería para el manejo de objetos remotos en Python. Permite construir aplicaciones en las cuales los objetos pueden comunicarse con otros objetos en red. Sin la necesidad de crear un complejo sistema de comunicación, es posible tener acceso a un objeto que se encuentra ejecutándose en otro sistema. [15]

PYRO se encuentra en su cuarta versión (en adelante **PYRO4**), está escrito completamente en **pure Python** y al igual que el propio lenguaje, es multiplataforma y multiversión. Se trata, sin lugar a dudas, de la **piedra angular** software de este proyecto.

La estructura de funcionamiento de la API sería la siguiente:

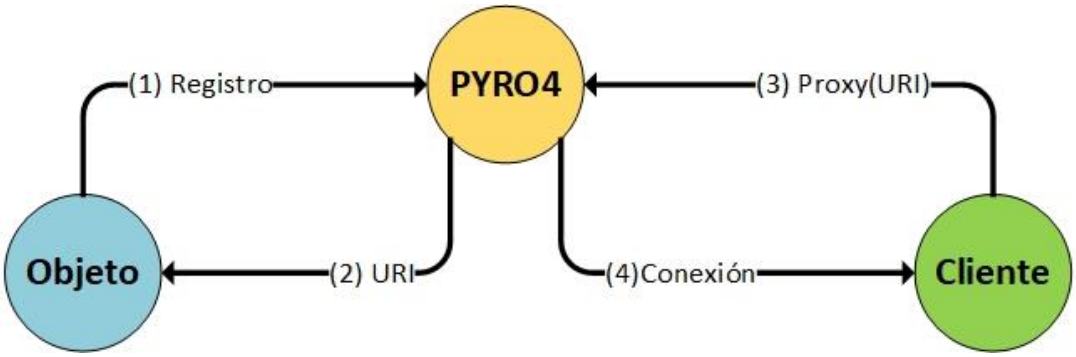

Figura 4: Estructura de funcionamiento de PYRO4

La ilustración explica la forma más básica de funcionamiento de PYRO4. Un objeto se registra en la API y este le asigna ***identificador de recursos uniforme*** (URI), este objeto queda a la escucha de peticiones externas.

Otro objeto (cliente) puede acceder a él **si conoce su URI** y si se encuentra en la misma red o en una red que pueda acceder a la del recurso.

Además, la API nos permite indicar **qué métodos o clases queremos dar a conocer al exterior.** Por defecto, no se expone ningún método ni clase, sino que debemos indicarlo mediante un **patrón de diseño** conocido como ***decorador***. Un decorador permite añadir funcionalidades extras a una función o clase.

4 MATERIALES Y MÉTODOS

Durante el siguiente punto hablaremos en profundidad sobre el hardware que se va a emplear, así como del desarrollo del software.

4.1 Raspberry Pi ®

En el desarrollo de este proyecto, se usarán como plataformas hardware distintas versiones de **Raspberry Pi**. Es la plataforma de cómputo principal del sistema.

Raspberry Pi es un ordenador de tamaño reducido de bajo coste creado por la fundación Raspberry Pi. Sus especificaciones son:

Tabla 1: Características Raspberry Pi 2 Modelo B

Raspberry Pi 2 Modelo B	
SoC	Broadcom BCM2836
CPU - GPU	900 MHz quad-core ARM Cortex A7 - Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Redes	10/100 Ethernet, 2.4GHz 802.11n wireless
Almacenamiento	microSD
GPIO	40-pin, poblados.
Puertos	HDMI, 3.5mm audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
Tamaño	85.60 mm × 56.5 mm × 17 mm

Tabla 2: Características Raspberry Pi 3 Modelo B

Raspberry Pi 3 Modelo B	
SoC	Broadcom BCM2837
CPU- GPU	1.2GHz 64-bt quad-core ARMv8 - Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Redes	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth	Bluetooth 4.1, Bajo consumo
Almacenamiento	microSD
GPIO	40-pin, poblados.
Puertos	HDMI, 3.5mm audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
Tamaño	85.60 mm × 56.5 mm × 17 mm

Figura 5: Raspberry Pi 3 Modelo B

Tabla 3: Características Raspberry Pi Zero W

Raspberry Pi Zero W	
SoC	Broadcom BCM2835
CPU - GPU	1GHz ARM11 core - Broadcom VideoCore IV
RAM	512MB LPDDR2
Redes	2.4GHz 802.11n wireless
Almacenamiento	microSD
GPIO	40-pin, despoblado.
Puertos	mini-HDMI 1080p60, 3.5mm audio-video jack, 4× Micro-USB para datos y alimentación, Camera Serial Interface (CSI), Display Serial Interface (DSI)
Tamaño	65mm x 30mm x 5mm

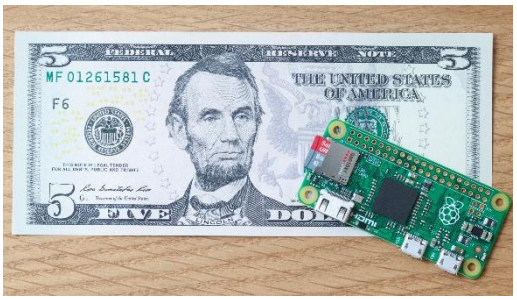

Figura 6: Comparación Raspberry Pi Zero con billete de 5$.

Como sistema operativo, ejecuta una versión adaptada de *Debian*, denominada ***Raspbian***, aunque permite usar otros sistemas operativos, incluido una versión de Windows 10.

La principal razón de uso de este hardware en este trabajo estriba en su **tamaño reducido**. Cualquier otro hardware, haciendo uso de sistema basado en UNIX y con las librerías correspondientes sería apto para ejecutar el entorno PYRO4BOT.

4.2 PYRO4BOT

PYRO4BOT es un *framework* de programación de código abierto, desarrollado **íntegramente en Python**. Tiene como objetivo **brindar la oportunidad de programar un robot sin importar el nivel de conocimientos que tenga el usuario que lo desarrolle**.

PYRO4BOT plantea **tres niveles de funcionamiento en relación a la dificultad de uso**:

- Nivel básico
- Nivel intermedio
- Nivel avanzado / experto

Nivel básico

En este nivel se ubicarían aquellas personas **sin conocimientos en programación**. En la actualidad, la composición del robot se realiza mediante un fichero de texto, pero se plantea el uso de **gráficos en líneas futuras** y el uso de lenguajes como Scratch.

Nivel intermedio

Nivel planteado para personas **con conocimientos en programación,** pero sin ser expertos completamente. Con un nivel básico de programación, permite configurar y programar un robot en pocos minutos y funcionar rápidamente. No es necesario modificar el código fuente del *framework* para poder programar una base robótica.

Nivel avanzado / experto

Nivel planteado para usuarios que tienen un **nivel muy avanzado en programación** y tienen capacidad para modificar completamente el código fuente. Si se quiere modificar el *framework*, PYRO4BOT es **completamente abierto y libre**. Cualquier usuario tiene permisos para modificar cualquier aspecto que necesite, incluso mejorar el sistema.

Este *framework* se enfoca especialmente en **hardware de bajo coste**, esto es, dispositivos que cualquier persona pueda adquirir a precios asequibles, acercando así la robótica a la sociedad. En consecuencia, PYRO4BOT permite el **desarrollo de bases robóticas de diferentes topologías**, no está destinado a un hardware determinado, sino que se destina a cualquier tipo de hardware capaz de ofrecer soporte con un sistema operativo basado en Linux.

Además, se pretende componer una **base robótica de forma sencilla y rápida**. Indicar al software qué hardware se está usando, seleccionarlo y entrar en ejecución en cuestión de minutos. Esta herramienta tiene una capacidad muy alta de creación de bases robóticas en un período de tiempo muy corto, dicho de otro modo, posee una curva de aprendizaje sencilla.

De igual modo, se busca **establecer comunicaciones con otros robots de forma simple**. Se pretende que el usuario pueda comunicar un robot con uno o varios y que puedan trabajar de forma simultánea, es decir, de forma **distribuida y colaborativa.**

A diferencia de otros entornos robóticos, PYRO4BOT no busca una eficiencia máxima, **se prioriza la sencillez frente a la eficiencia**.

Dadas las características anteriores, PYRO4BOT es un medio de aprendizaje para aquellas personas que tienen motivación por el diseño y la construcción de robots. Permite llevar a cabo **robótica educativa.**

4.2.1. Estado inicial

Para conseguir los objetivos planteados, en primer lugar, se ha partido de una versión inicial de PYRO4BOT, desarrollo realizado íntegramente en Robolab.

Esta versión de PYRO4BOT se encontraba en una versión muy básica, por lo que fue un paso obligado el aumento de su robustez, seguridad y la adición de una serie de funcionalidades.

En este **estadio inicial de desarrollo**, decidimos el uso de ***LearnBot***, un prototipo de robot educativo también desarrollado íntegramente en Robolab.

El contenido del proyecto en su estado inicial se encuentra en ***ANEXO 1: Estado inicial del proyecto.***

4.2.2. LearnBot

Este robot se compone de una Raspberry Pi 2 Modelo B, una placa de Arduino (*Arduino Expansion Shield)* y una batería de 10.000mah como elementos principales.

La batería alimenta a 5V tanto a la placa de Arduino como a la Raspberry Pi.

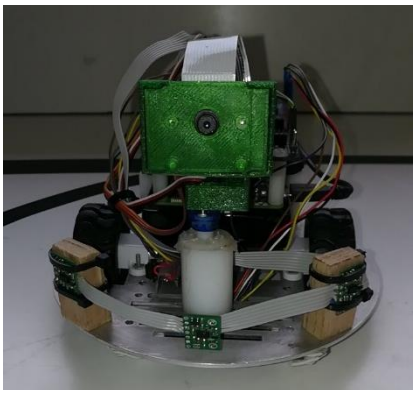

Figura 7: LearnBot

En este caso, la placa de Arduino es una **expansión de Arduino** realizada expresamente para ser acoplada a una Raspberry Pi (***Arduino Expansion Shield for Raspberry Pi).***

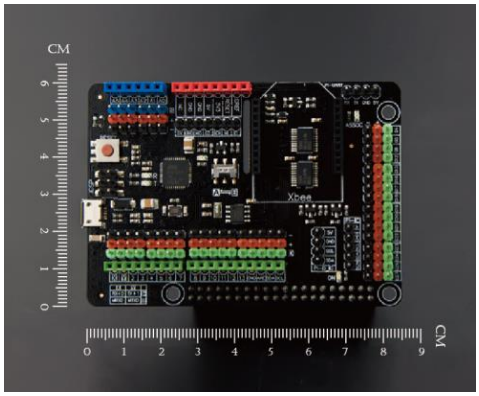

Figura 8: Arduino Expansion Shield for Raspberry Pi

El uso de Arduino **no es completamente necesario**, en este caso se usa para tener más control sobre las entradas y salidas al sistema operativo. Gracias a la placa, obtenemos mayor rendimiento de computo en las entradas y salidas ya que estas son administradas por el microcontrolador de Arduino.

La Raspberry Pi tiene conectada la placa de Arduino que su vez tiene conectados:

- Sensores **infrarrojos** (CNY70)
- Sensores **ultrasonido** (Pololu VL53L0X)
- Ruedas **motrices** (Motor a 5V)
- **Servomotor** para mover cámara en vertical (TowerPro SG-90)
- **Servomotor** para mover cámara en horizontal (TowerPro SG-50)

Además, la Raspberry Pi tiene conectada directamente una **cámara** de 5 megapíxeles que se conecta a través del conector CSI de la placa.

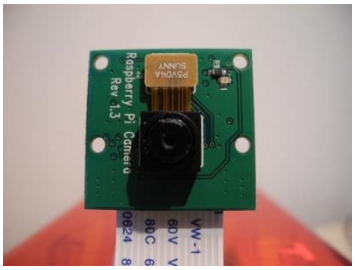

Figura 9: PiCamera

4.2.2.1. Conexiones

Para ejecutar el software, en primer lugar, es necesario realizar la conexión física de los dispositivos.

En primer lugar, conectamos el conjunto de pines de la *Raspberry Pi* a la *Arduino Expansion Shield*. La placa Arduino está preparada para conectar directamente a la Raspberry Pi.

Los **sensores de infrarrojos** son de tipo analógico. Tienen una alimentación de 5V y un pin por cada infrarrojo, 4 en total. Los 4 pines se conectan a los 4 primeros pines analógicos de la placa Arduino. La placa se encarga de realizar la conversión analógico-digital para después tratar los datos en la Raspberry Pi.

Los **sensores de ultrasonido** se conectan igualmente a 5V y al bus I2C de la placa, concretamente a los pines 11, 12, SDA y SGL.

Las **ruedas motrices** están alimentadas con 5V y conectadas a un controlador L298N. Este controlador tiene 4 pines que se conectan a la placa de Arduino a los pines 5,6,9 y 10. Estos pines se encargan del control de los motores de las ruedas.

Por último, los **servomotores SG-50** y **SG-90**, se conectan a 5V y tienen 1 pin que indica la modulación de ancho de pulsos (PWM).

Todos los datos obtenidos por la placa son enviados a la Raspberry Pi **mediante la conexión serie.**

4.3 Desarrollo, funcionamiento y mejoras

Se ha **mantenido la filosofía** que se planteaba con el desarrollo inicial del mismo, es decir, buscar la **simplicidad, la limpieza y la sencillez**. De esta forma se consigue hacer el producto más atractivo para la comunidad.

Una vez detallado el hardware que se va emplear en el comienzo de este proyecto, pasaremos a comprobar cómo funciona el software del mismo.

4.3.1. Estructura software

PYRO4BOT se basa en procesos independientes que son accesibles a través de un proxy. **PYRO4BOT** tiene un proceso principal denominado ***Nodo***, el cual es el controlador del resto de procesos. Estos procesos, denominados **componentes o servicios** trabajan de forma independiente sin formar parte del proceso ***Nodo*** que los ha lanzado, pero teniendo una relación entre ellos a través de sus proxys de comunicación.

Los proxys de acceso son creados parcialmente por el desarrollador y por la API Pyro.

4.3.2. Lanzamiento

En el lanzamiento del robot, se ejecuta un primer proceso ***lanzador*** encargado de comprobar la configuración y de ser correcta, levantar el proceso principal ***nodo*.** Una vez que éste ha sido lanzado, el proceso anteriormente llamado ***lanzador***, pasa a realizar **funciones de gestión** sobre el ***nodo*.**

El proceso denominado ***nodo*** se encarga a su vez de **poner en ejecución los componentes** que han sido predefinidos en el **fichero de configuración.**

Una vez el robot ha sido lanzado, cada uno de los componentes en él desplegados pasan a ser **procesos independientes** sin vinculación directa con el **nodo principal,** es decir, estos componentes no se encuentran incluidos en la misma zona de memoria que el nodo.

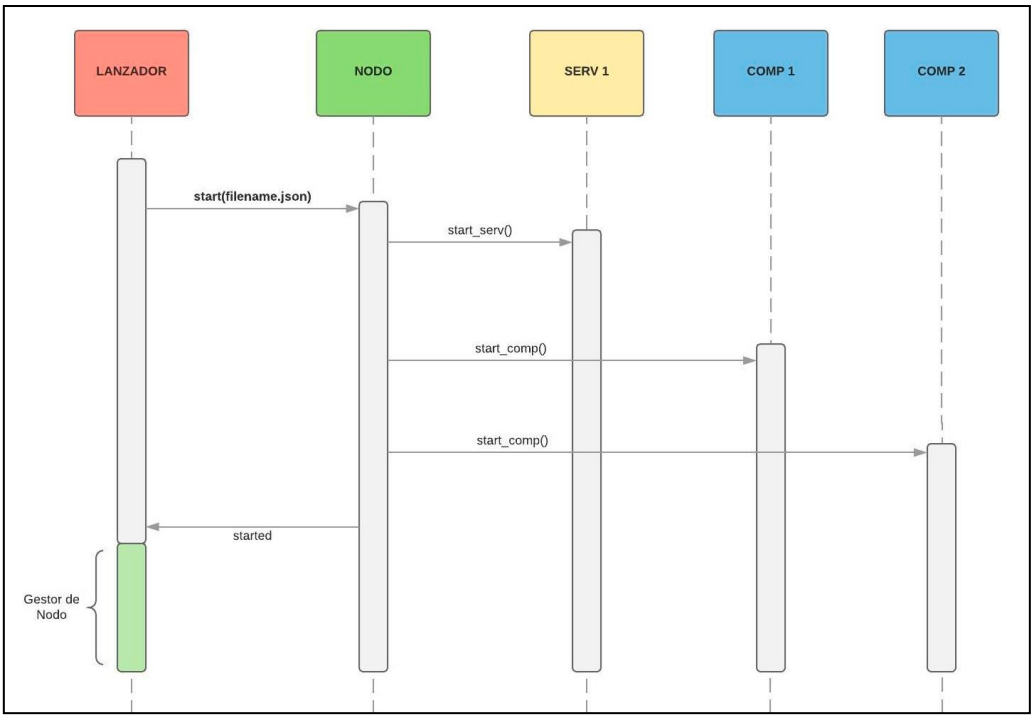

Figura 10: Secuencia de lanzamiento de un robot con PYRO4BOT.

Cada uno de los componentes son ejecutados de forma independiente, para no perder completamente las comunicaciones cada componente tiene un proxy de conexión al nodo.

4.3.3. Acceso remoto al objeto

Para poder dotar de comunicaciones al **nodo** con sus **componentes y servicios,** y, además, dotarles de **conexión hacia el mundo exterior**, hacemos uso de la **API de PYRO4.**

Tanto el nodo como cada componente, son **procesos que son accesibles desde otro proceso gracias la API de PYRO4.**

Para ello, una vez el proceso en cuestión se ejecuta, éste crea un **hilo de tipo PYRO4**. Este hilo auto-registra el objeto que va a ejecutarse en el componente otorgándole un

nombre que lo identifica. Esta acción nos devuelve un **identificador de recursos uniforme** (en adelante **URI),** que será aquella cadena de caracteres que identifique a dicho objeto para el resto del mundo.

Esta URI sigue el siguiente formato: "**PYRO:id_objeto@localización":**

- **PYRO**. Es la palabra clave que contienen todas las URIs. Invariable. Siempre va seguida de **dos puntos** (":").
- **id_objeto**. Palabra clave asignada al objeto.
- **localización**. Está compuesto por IP y puerto de la máquina dónde se está ejecutando. Sigue el formato "**IP:Puerto**"

Ejemplo de URI: *PYRO:robot@192.168.10.1:4041*

Una vez ha sido registrado, este hilo queda en *background* escuchando peticiones hacia este objeto. Las peticiones que recibe pueden ser tanto local como remotas. En este punto, cada componente hace de **servidor PYRO4.**

Cuando otro objeto, **cliente,** quiera acceder a este objeto, **necesita saber la URI** que lo identifica. Si la conoce, puede solicitar a la API **un proxy de conexión** y acceder al objeto cómo si un objeto cualquiera se tratase.

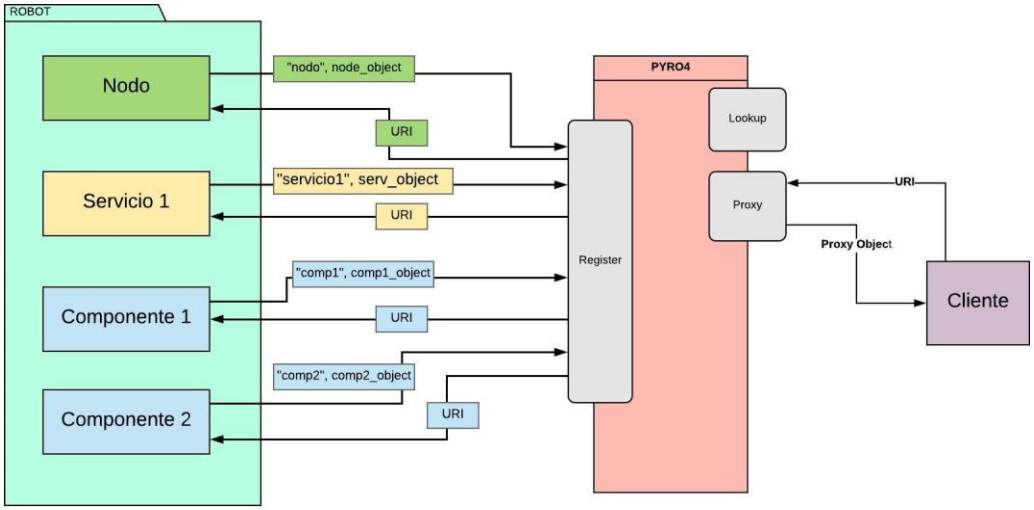

Figura 11: Esquema de funcionamiento Cliente/Servidor usando URIs.

4.3.4. Servidor de nombres

No es posible conocer las URIs de todos los objetos a los que queremos conectar mediante proxy, al igual que no es posible conocer todas las IPs de todas las páginas webs que accedemos día a día. Para ello, en la web existen un servidor DNS que nos resuelve un dominio a una IP real.

En este caso, el sistema funciona de la misma manera. Mediante **servidor de nombres.**

Antes de lanzar un robot, **se registra en el servidor de nombres para poder ser localizado fácilmente**.

Una vez dado de alta, otros procesos pueden pedir acceso al nodo y siempre que conozcan la contraseña asociada podrán acceder a él. Una vez tenga acceso, podrá consultar que componentes tiene ejecutando ese robot.

Tanto la contraseña del robot, como otros parámetros pueden ser configurados en el fichero de configuración como veremos posteriormente.

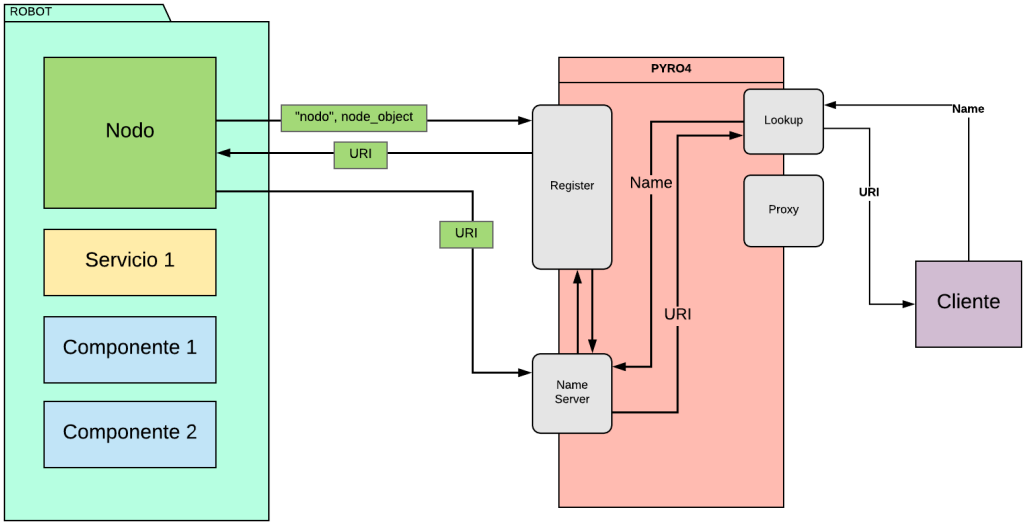

Figura 12: Esquema de funcionamiento C/S con Servidor de Nombres.

4.3.4.1. Tipos de servidores de nombres

El sistema puede funcionar de dos formas:

- Servidor de nombres genérico PYRO4
- Servidor de nombres gestionado, denominado ***Big Brother***.

En el proceso de lanzamiento del robot, se comprueba de la existencia en la red de un servidor de tipo ***Big Brother***, si existe, indica que la red está siendo **gestionada** por este proceso manejador de servidor de nombres.

En caso de no existir, se **comprueba la existencia de un servidor de nombres genérico,** si existiera, es usará este como servidor de nombres.

En caso de no existir ningún servidor de nombres de ningún tipo en la red, se **despliega automáticamente un servidor de nombres genérico.**

Dicho de otro modo, el sistema otorga preferencia a un servidor de nombres gestionado antes que un servidor de nombres genérico.

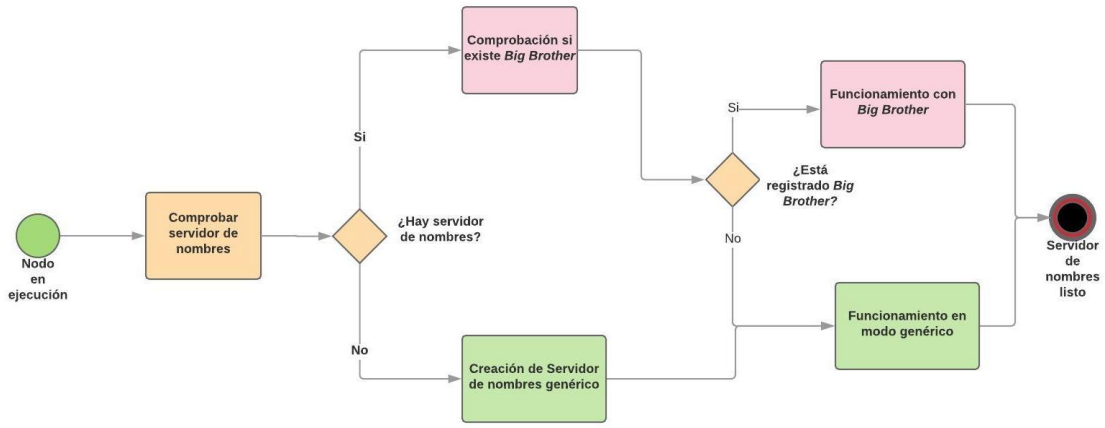

Figura 13: Esquema de funcionamiento: Obtener servidor de nombres.

4.3.4.2. Big Brother

Para otorgar de mayor funcionalidad y control al sistema, se ha desarrollado ***Big Brother***, un gestor para el servidor de nombres de PYRO. Cuando el robot trabaja con *BigBrother* tiene mayor capacidad de comunicaciones que si trabajase con un servidor genérico.

Big Brother no es más que una **capa de comunicaciones intermedia**, contiene la lógica de control de comunicaciones y es necesaria para conectar robots con unas necesidades de comunicación complejas.

Estructura

En primer lugar, se despliega en la red un **servidor de nombres público** en el cuál solo está registrado el propio *Big Brother*, a continuación, se despliega un **servidor de nombres privado** únicamente accesible para *BigBrother*.

El primero de ellos, es usado por robots para poder localizar y comprobar qué tipo de servidor de nombres se está ejecutando. Al estar registrado únicamente *Big Brother*, el robot, cuando comprueba la existencia de un servidor de nombres, si es que existiera, comprueba que estuviera ya registrado un objeto denominado *Big Brother*, si lo estuviera, obtiene el proxy de acceso a él y éste será su manejador de servidor de nombres.

El segundo es el **servidor de nombres real** que se va a usar para toda la red. Este servidor es local y por lo tanto, privado. Esto conlleva que solo *BigBrother* tiene acceso a él. De esta manera, otorgamos al servidor de nombres de una mayor seguridad siendo únicamente posible acceder a él a través de *Big Brother*.

Con el fin de otorgar mayor **funcionalidad futura**, se tomó la decisión de desplegar *Big Brother* como un proceso a parte y no incluírlo dentro del lanzamiento de un primer robot.

Funcionamiento

Big Brother tiene implementados los mismos métodos que existen en el servidor de nombres genérico, pero otorgando de mayor capacidad lógica y seguridad.

Cuando a Big Brother le llega alguna petición a su interfaz, comprueba quien realiza la petición y si satisface la directiva de seguridad (si la tuviera), le permite el acceso al servidor de nombres.

Además, esta llamada puede ser **asincrona**. Un robot puede solicitar un proxy para un objeto que aún no está presente en la red, *Big Brother* almacena esta petición, cuando el objeto solicitado entra en la red, Big Brother lo comunicará al solicitante.

Hay métodos en los cuales no es necesario añadir comprobaciones.

Metodos accesibles

Big Brother tiene un conjunto de métodos accesibles que son:

- **list**. Ejecuta el método list de la API. Retorna el conjunto de robots registrados en el servidor de nombres privado. No tiene restricciones de uso aunque si se busca seguridad se recomienda añadir una comprobación por contraseña.
- **request**. Realiza una petición para tener acceso a un proxy de un robot o componente. Este método crea un hilo por cada petición y este hilo no muere hasta que la petición no ha sido resuelta. Al invocar *request*, el objeto que lo llama envia el método dónde mandar posteriormente la respuesta. Método **asincrono**.
- **lookup.** Invoca al método lookup de la API pero no de forma directa. Este método comprueba el nombre y según sea este devuelve la URI pedida o el conjunto de ellas si se trata de un robot completo.
- **ping.** Realiza ping sobre el servidor de nombres. Devuelve true si hay conexión.
- **register.** Registra un nuevo robot en el servidor de nombres privado.
- **set_metadata.** Invoca al método set_metadata de la API.
- **proxy.** Devuelve un proxy o conjunto de proxy en función de la solicitud.
- **ready.** Devuelve true si el servidor de nombres local está desplegado.

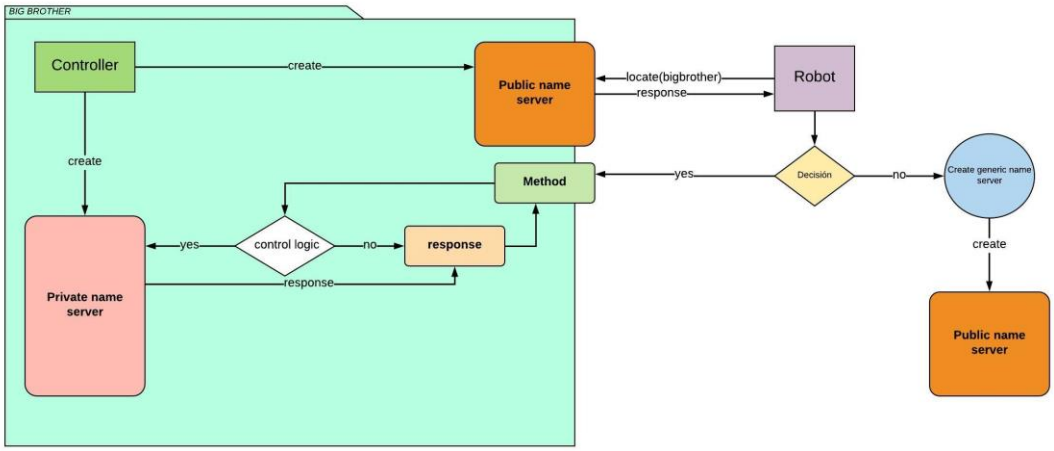

Figura 14: Servidor de nombres gestionado: Big Brother

BigBrother también posee un pequeño sistema de monitorización para controlar la carga de trabajo del mismo.

*La guía para poder monitorizar el dispositivo se encuentra en **ANEXO 2: Monitorización Raspberry Pi.***

Despliegue

Dado que la **capacidad de computo necesaria** para poder ejecutar *Big Brother* es mínima, se ha desplegado en una **Raspberry Pi Zero**.

Para ejecutar *Big Brother*, nos situamos en la carpeta ***bigbrother*** y ejecutamos:

```
python start_pyro4bot_BB.py
```

Al finalizar la ejecución, tendremos disponible una pequeña consola de administración que nos permite realizar ciertas acciones sobre el servidor de nombres.

Big Brother tiene un pequeño fichero de configuración en la carpeta ***config*** que indica ciertos parámetros como nombre, interfaz de red a usar, puertos, contrasñas, etc.

La configuración de Raspberry Pi Zero para se encuentra en ***ANEXO 3: Configuración Raspberry Pi Zero para Big Brother.***

Debilidades

Existen otros aspectos que aunque pueden suponer una vulnerabilidad, no se tienen en cuenta debido a la naturaleza del sistema. Por ejemplo, sería posible conocer toda la red si otro proceso se hace pasar por BigBrother, conociendo el código fuente y sabiendo como funciona esto es posible.

Además, *Big Brother*, si está activo, es el nexo de unión de todos los robots, si dejara de estár disponible toda la comunicación dejaría de estar activa. De cara al futuro, está planteado continuar con este proyecto para otorgar mayor funcionalidad al sistema y prevenir este tipo de situaciones.

4.3.5. Servicios y Componentes

Son **procesos independientes,** cada uno de ellos se ejecuta en un ámbito distinto y ninguno de ellos tiene relación directa entre sí.

La filosofía de **servicios y componentes** sigue siendo la misma que la planteada en la versión inicial.

- **Servicios**. Aquellos procesos que ofrecen funcionalidades locales a los distintos componentes.
    - **Ejemplo**: Una servicio de Arduino que obtenga datos a través del puerto serie para uno o varios componentes.
- **Componentes**. Son aquellos entes que tienen comportamiento autónomo, que pueden ofrecer o no una serie de interfaces (a través de la API de PYRO4) y que realizan una tarea concreta.
    - **Ejemplo**: Leer información de un sensor de infrarrojos. Ejecutar otros componentes.

Todos los **servicios declarados estarán siempre presentes** en todos los componentes declarados para su uso, los servicios prestan funcionalidades del sistema a los componentes.

Los componentes suelen ir asociados al uso de un **hardware concreto** pero también pueden tener estar destinados a usar otros componentes locales.

Para otorgar al sistema de **funcionamiento en tiempo real**, es necesario la ejecución en procesos del sistema independientes y, además, para poder otorgarles **capacidad de trabajo conjunta** es necesario un sistema de comunicaciones, es aquí cuando hacemos uso de la API de Pyro4.

4.3.5.1. Creación

Para crear un nuevo componente tan solo es necesaro realizar una copia o bien de otro fichero o bien del fichero de ejemplo denominado ***Template.py.***

Con el único objetivo de minimizar y hacer lo más **sencillo** posible a cada uno de ellos, solo es necesario escribir aquello que se pretenda hacer en un método dentro del componente o servicio. Este componente gracias al uso de *herencia* y *decoradores* del lenguaje, contiene un conjunto añadido de métodos y atributos que son transparentes al usuario, de tal forma que, el usuario no tiene constancia ni necesita saber que existe, simplemente, necesita escribir **que necesita hacer**.

Gracias a la API podemos decir que métodos y/o clases de cada componente queremos **exponer** al exterior, o, dicho de otro modo, qué queremos tener disponible en la interfaz por la parte del servidor. Para ello, cada clase y/o método que queramos exponer deberá llevar el decorador ***@Pyro4.expose***

Ejemplo de componente:

```
import time
from node.libs import control
import Pyro4

class componenteA(control.Control):
    __REQUIRED = [“miatributo1”, “miatributo2”]

    def __init__(self):
        self.value = self.miatributo1 + self.miatributo2;
        self.start_worker(self.worker)


    def worker(self):
        while self.worker_run:
            # Your code here
            time.sleep(self.frec)

    @Pyro4.expose
    def get_something(self):
        return self.value
```

Supongamos un componente llamado **mi_componente** que instancia la clase **componenteA,** ejemplificada en el código anterior.

Podemos observar un atributo de clase “**__REQUIRED”.** Este atributo indica el *framework* la necesidad de tener esos dos atributos predefinidos en su configuración. Si este componente no los tuviera definidos, el sistema devolvería un error y no se desplegaría. Este atributo es una lista de atributos imprescindibles por las razones que fuera, de esta forma indicamos que se necesita indicar en el fichero de configuración.

En el método ***__init__*** escribimos los valores de inicialización del objeto. En este caso, otorgamos a ***value*** el valor de dos atributos predefinidos en la configuración. Además, hay que indicar al componente cuál o cuáles serán los métodos que serán hilos en ejecución en el componente, para ello, debemos pasarle cada uno de los métodos como parámetros al método ***start_worker.***

El código principal del componente debe estar escrito en el método ***worker*** en este caso, aunque puede ser el método que el desarrollador quiera.

Pueden existir un conjunto de métodos que no se están ejecutando de forma continua. Estos métodos tienen como objetivo realizar acciones sobre el componente u obtener algún tipo de información.

El método ***get_something*** tiene un decorador **@Pyro4.expose,** gracias a ese decorador, ese método está público para llamadas a su interfaz. Si otro componente o un cliente acceden al proxy, solo podrían ejecutar el método ***get_something***.

Los **servicios y componentes** pueden estar ya desarrollados o no. PYRO4BOT sigue actualmente en desarrollo, continuamente se van añadiendo componentes que trabajan con un hardware determinado, pero, es posible que se necesite usar un hardware que no está contemplado en ningún componente ya creado. En este caso, la persona que esté usando el *framework* deberá generar su propio componente.

4.3.5.2. Métodos, atributos y decoradores inherentes.

Existen una serie de métodos, atributos y decoradores que son inherentes a cada uno de los servicios y componentes que se despliegan.

La mayoría de ellos son heredados y otros son inyectados mediante el uso de métodos mágicos del lenguaje Python.

Atributos inherentes más importantes.

- **self.botname**. Indica el nombre del robot.
- **self**.**pyro4id**. Indica la URI del componente.
- **self.node**. Proxy de conexión hacia el nodo principal del robot.
- **self**.**cls**. Indica el nombre de la clase que se está ejecutando.
- **self._services**. Lista de proxys hacia servicios.
- **self.exposed**. Atributo que indica los métodos que expone el componente.
- **self**.**docstring**. Atributo que indica la documentación de los métodos que tiene el componente.

Métodos inherentes más importantes.

- **self.start_worker(func).** Método encargado de ejecutar como hilo del componente a la función pasada por parámetro. Este método está destinado a ser

llamado cuando se necesite que un componente tenga un hilo trabajando en background.

  - **func:** Función a ejecutar

- **self.start_publisher(publication, [frec]).** El sistema puede funcionar como publicación / suscripción. Para ello, debemos crear un objeto de tipo *Publication* e ir actualizando estos datos. Este método se encargará de publicar cada cierto tiempo la información actualizada.
  - **publication**. Objeto *Publication* con la información a publicar
  - **frec.** Indica los segundos entre cada publicación. Por defecto 0.01s. (Opcional)
- **self.start_subscripcion(object, target, target_attr, [subscripter_attr], [subscripter_password]).** Manda una subscripción al *target* pasado como parámetro y concretamente al atributo indicado en *target_attr*.
  - **target.** Objeto al cual desea suscribirse.
  - **target_attr.** Atributo del objeto al que desea suscribirse.
  - **subscripter_attr.** Indica el atributo dónde sea recibir la información. (Opcional)
  - **subscripter_password.** Indica la contraseña del suscriptor que solicita la conexión. Si no se indica nada, se usará la contraseña por defecto. (Opcional)
- **self.__exposed__().** Devuelve una lista de métodos que expone el componente.
- **self.__docstring__().** Devuelve una lista de la documentación de cada método del componente.

Decoradores inherentes más importantes.

- **@Pyro4.expose. Es el decorador por excelencia del *framework*. Se usa para indicar que ese método o clase se expone y es accesible remotamente.**
- **@flask(type, args).** Este decorador se usa para otorgar información sobre un método a uno de los servicios en desarrollo de PYRO4BOT, *Flask para Pyro4bot.* El estudio de este servicio no es objeto de este TFE.
  - **type:** Cadena de caracteres que indica el tipo de método en el cual nos encontramos. Suelen ser “sensor” o “actuator”.
  - **Args:** Usado para especificar cuantos argumentos acepta (si es un “actuator”) o devuelve (si es un “sensor”).

4.3.6. Fichero de configuración

Siguiendo la filosofía de funcionamiento y buscando la sencillez máxima para el usuario, el fichero de configuración se ha simplificado lo máximo posible. Asimismo, se da la opción de configurar el lanzamiento del robot con parámetros opcionales.

```
{
"node":
        {
        "name": “<nombre del robot>”,
        },
"services":
        {
        "<servicio 1>": {"cls": "<clase servicio1>", “atributo 1”: <valor>, "atributo 2": <valor>, …},
        "<servicio 2>": {"cls": "<clase servicio 2>", “atributo 1”: <valor> , …}
        },
"components":
        {
        "<comp 1>": {"cls": “<clase comp 1>”, “atributo 1”: <valor> , …}},
        "<comp 2>": {"cls": “<clase comp 2>”, “atributo 1”: <valor> , “atributo 2”: <valor> , …},
        "<comp 3>": {"cls": “<clase comp 3>”,  “atributo 1”: <valor> , …},
        }
}
```

En el cuadro anterior se expone el **único requisito** que se necesita para que se despliegue el robot, bastaría con indicar el nombre del mismo y todos aquellos servicios y/o componentes que sean necesarios.

Por otro lado, existen una serie de atributos opcionales que si son incluidos se tomarán en cuenta y si no se tomará su valor por defecto.

Atributos opcionales:

- **def_frec**. Indica al robot el periodo a usar en los hilos.
    - **Valor por defecto**: 0.05.
- **ethernet.** Nombre de la interfaz de red a usar.
    - **Valor por defecto:** Se usará la interfaz dónde en cuya red se esté ejecutando un servidor de nombres de PYRO4 si es que lo hubiera, si lo no lo hubiera se usará la primera interfaz disponible.

- **ip.** Dirección IP a usar.
    - **Valor por defecto:** Se tomará como valor por defecto la dirección IP de la interfaz indicada en el campo ethernet.
- **port_ns**. Puerto para despliegue de servidor de nombres de PYRO4 si es que fuera necesario su despliegue. En caso de estar usado, se usará el siguiente libre a partir de éste.
    - **Valor por defecto:** 9090.
- **port_node.** Puerto para despliegue del demonio en el nodo para aceptar peticiones de PYRO4. En caso de estar usado, se usará el siguiente libre a partir de éste.
    - **Valor por defecto:** 4040.
- **start_port**. Igual que *port_node*, pero para servicios y componentes.
    - **Valor por defecto:** 5050.
- **def_worker.** Valor booleano que indica si desea funcionar de inicio o no.
    - **Valor por defecto:** true.
- **password**. Indica la contraseña de acceso al robot para la API de Pyro4.
    - **Valor por defecto**: Nombre del robot.
- **bigbrother-password**. Indica la contraseña de acceso para usar *Big Brother*.
    - **Valor por defecto**: 'PyRobot'.

4.3.6.1. Dependencias

Uno de los objetivos planteados es poder **otorgar de comunicación con otros robots**. Para ello, cuando se necesite hacer uso de otro componente en otro robot, se indicará mediante un atributo en el componente en cuestión.

```
{
"node":
        {
        "name": "robotA",
        },
"services":
        {       },
"components":
        {
        "mi_componente": {"cls": "componenteA", "-->": "robotB.sensor_gps",
"miatributo1": 1, "miatributo2": 2 }
        }
```

Tomando como referencia el fichero de configuración anterior, tendríamos un ***robotA*** ejecutándose, con 1 único componente definido y que, además, éste tiene una dependencia con ***robotB***, concretamente con el componente llamado ***sensor_gps.***

Haciendo uso del atributo **"-->",** estamos indicando que el componente llamado ***mi_componente*** necesita para funcionar al componente ***sensor_gps*** contenido en **robotB.**

El sistema evaluará la dependencia y comprobará si puede ser resuelta o no. Además, podemos estar ante dos situaciones:

1) Se está usando servidor de nombres del tipo ***Big Brother.***
2) Se está usando servidor de nombres genérico.

Si nos encontramos ante el **primer caso**, el componente arrancará en modo **ASYN (Asíncrono),** esto quiere decir que el componente está a la espera de tener conectividad con el componente del que depende. Una vez se resuelva esta conectividad, el estado del componente pasará a modo de funcionamiento normal y comenzarán sus hilos de ejecución.

Si nos encontramos en el **segundo caso**, servidor de nombres genérico, si en el momento de ser ejecutado, la dependencia puede ser resuelta, el componente arrancará correctamente, si, por el contrario, no puede ser resuelta, ese componente no podrá arrancar ni ahora ni posteriormente. Dicho de otro modo, para poder hacer uso de esa dependencia, debe de existir en el momento de ser lanzado el robot.

El primer caso, usando ***Big Brother,*** siempre será una mejor solución debido a la lógica de control que contiene.

Tipos de dependencias

En el ejemplo anterior, se muestra un tipo de dependencia directa a un robot determinado y un componente determinado, pero existen más tipos.

- **Dependencia directa.** Dependencia a un robot determinado y componente concreto.
    - **Sintaxis:** Robot.componente
- **Dependencia completa.** Dependencia de un robot completo. Acceso a todos sus componentes. Rara vez se puede llegar a usar, pero el sistema lo contempla para usos futuros.

    - **Sintaxis:** robot. (o también robot.*)
- **Dependencia indirecta.** Dependencia a un robot cualquiera y componente concreto.
    - **Sintaxis:** ?.componente
- **Dependencia indirecta a todos.** Dependencia de todos los robots que tengan un componente determinado.
    - **Sintaxis:** *.componente
- **Dependencia de nodo**. Dependencia del nodo de un robot. Rara vez se puede llegar a usar, pero el sistema lo contempla para usos futuros.
    - **Sintaxis**: robot

La dependencia directa y de nodo pueden ser resueltas por un servidor de nombres genérico, el resto solo pueden ser resueltas haciendo uso de *Big Brother*.

Dependencia correcta

Una vez la depedencia es correcta, es decir, es accesible, pasará a formar parte del atributo ***deps.***

Este atributo forma parte de una serie de atributos que todos los componentes y servicios en ejecución tienen. ***Deps*** contiene una lista de proxys con todas las depedencias que tiene un robot.

Para hacer uso de estas dependencias, accedemos al atributo **deps** y usamos el proxy como si se tratase de un objeto cualquiera. Dentro de este proxy tendremos acceso a todos los métodos y/o clases que han sido **expuestos** mediante el decorador **@Pyro4.expose** en el objeto remoto.

Debemos de tener en cuenta que si estamos tratando de enviar una instancia de una clase creada por el desarrollador, la API necesita *serializar* este objeto y para ello debemos de indicarlo nosotros mismos. Esto no ocurre si el tipo de dato a enviar está predefinido en el propio lenguaje, como pueden ser *int, float, dict, list, array…*

En la documentación de la API se expone como enviar una clase propia a través de una interfaz de Pyro4.

4.3.7. Tipologías diferentes

PYRO4BOT tiene una base de funcionamiento para funcionar con cualquier tipología de robot. Sin importar si usa o no un hardware específico, el *framework* puede ser usado bajo prácticamente cualquier tipo de hardware siempre que se use un sistema operativo basado en Linux.

4.3.7.1. AlphaBot

AlphaBot es una plataforma de desarrollo de robots de bajo coste, compatible con Raspberry Pi / Arduino.

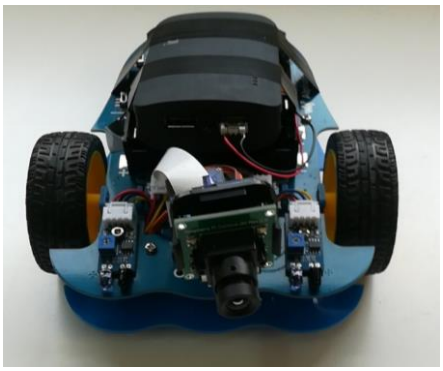

Figura 15: AlphaBot

En nuestro caso, el kit usado viene con Raspberry Pi 3 Modelo B, además incluye:

- Sensores infrarrojos
- Sensores ultrasonidos
- Sensores de medición de velocidad
- Control remoto por infrarrojos
- Cámara acoplada a dos servos.
- Motores para ruedas y controlador de ellas.

Permite realizar una gran cantidad de tareas con robots, como, por ejemplo:

- Seguir una línea.
- Evitar obstáculos.
- Comprobar velocidad.
- Recibir órdenes por infrarrojos.
- Obtener imagen de la cámara y mover la cámara.
- Mover la base robótica.

Son solo algunas de las utilidades que se pueden hacer, a partir de aquí, el máximo de tareas complejas que se pueden realizar está en la imaginación del programador.

Alphabot es una base robótica de aproximadamente 80€ y cuenta con una gran cantidad de complementos. Esta base de bajo coste puede ser desplegada por

PYRO4BOT en cuestión de minutos si los componentes se encuentran ya desarrollados. Esta base robótica es completamente ajena al desarrollo de PYRO4BOT, en cambio, es 100% compatible con el *framework* debido a la naturaleza del mismo.

4.4 Comunidad

Una de las características más importantes de este proyecto es ser de carácter abierto. Consideramos que este proyecto debe de ser completamente libre y accesible para todo el mundo y que cualquier persona pueda aportar su contenido y hacer crecer el proyecto.

Este *framework* **nace con el objetivo de plantar una semilla** sobre programación robótica fácil, rápida y sencilla.

Una vez creada esta semilla, este software tiene una base sólida de funcionamiento y permite que otros usuarios añadan nuevos componentes preparados para funcionar con otro tipo de hardware distinto. Durante el desarrollo de este TFE se han desarrollado multitud de componentes que trabajan con diferentes sensores, actuadores, cámaras, etc. En líneas futuras se pretende **conseguir una comunidad** de personas que aporten sus componentes, aporten sus ideas y pasen a formar parte del *framework*.

Así, conseguimos una mayor portabilidad y aceleramos el proceso de composición de robots. Una persona no necesita programar un componente si alguien ya lo ha hecho, puede tomar su ejemplo y directamente usarlo.

5
RESULTADOS Y DISCUSIÓN

5.1.Pruebas

Para constatar la estabilidad y robustez del sistema se han realizado distintas pruebas de funcionamiento con los equipos robóticos.

Se han realizado tanto pruebas individuales como colectivas para asegurar el correcto funcionamiento del sistema. A continuación, se exponen y detallan brevemente algunas de ellas.

Se han realizado pruebas reales con los robots **LearnBot** y **AlphaBot** con la idea de constatar el correcto funcionamiento del sistema y comprobar si se han cumplido o no los objetivos planteados.

No debemos olvidar que en el caso de **LearnBot**, nos encontramos la dificultad añadida de usar una placa de Arduino que se comunica con la Raspberry a través del puerto serie. Al estar conectado otro microcontrolador hardware como Arduino, debemos de obtener la información a través de él y no directamente desde los GPIO de Raspberry Pi.

5.1.1. Individuales

Se ha desplegado los robots con componentes para el manejo de sensores ultrasonido, sensores infrarrojos, motores de ruedas y servomotores para el movimiento de la cámara.

En el caso de **LearnBot**, al estar conectado a través de una placa de Arduino, se ha creado un servicio para la gestión del puerto serie. Además, para ambos, se ha creado un servicio para la gestión de conexiones remotas a la cámara.

Una vez instanciados, y en funcionamiento, se han realizado las siguientes pruebas:

- Movimiento sencillo del robot a través de sus motores en las ruedas.
- Seguimiento de línea en el suelo mediante sensores infrarrojos.
- Detección y seguimiento de un color determinado mediante la cámara y los motores de las ruedas.
- Evitar obstáculos mediante sensores ultrasonido y mediante sensores infrarrojos.
- Servidor de cámara para otros clientes (varios clientes pueden conectarse remotamente a la cámara del robot y observar dónde está en ese momento)
- Mostrar información en pantalla de tinta electrónica.
- Detección de marcas *AprilTag.* [16]

Los resultados de las pruebas han sido **óptimos**, el robot es capaz de llevar a cabo con una suficiente fluidez todas las tareas propuestas y el sistema es robusto. También debemos destacar el poco tiempo necesario para añadir nuevos componentes o crear componentes sencillos.

5.1.2. Distribuidas

Se han realizado pruebas de carácter distribuido, es decir, almacenando parte del cómputo en otro equipo o haciendo uso de alguna manera de otro dispositivo.

Para ello, se han realizado pruebas entre ambos robots anteriormente citados, como pueden ser:

- **Modo espejo.**

  En esta prueba, se realizaba una serie de acciones en un robot y este se encargaba de replicar esa orden en el otro.

Disponemos de los dos robots. Se despliega **LearnBot** con dos componentes, **mirror** y **ruedas.** Además, tiene desplegado el servicio de control de Arduino. Se despliega **AlphaBot** con un componente para gestionar sus motores de las ruedas y un servicio para gestionar de **GPIO.**

El componente **ruedas** es el encargado de mover los motores de las ruedas a través del servicio de Arduino, este componente se ha declarado **local** y solo puede ser usado por componentes locales.

El componente **mirror** es el encargado de recibir peticiones. Este componente tiene dependencia directa con el componente **ruedas de Learnbot** y con el componente **ruedas de AlphaBot.** Este componente recibe peticiones y las envía tanto a **LearnBot** como a **AlphaBot.** Por lo tanto, todas las peticiones realizadas se replican de igual forma a dos robots, haciendo que ambos hagan el mismo movimiento.

Lo importante de este ejemplo está destinado a comprobar la comunicación entre dos robots y la robustez del sistema. El cliente que manda la petición solo realiza una a un robot, y este es el encargado de replicarlo.

- **Detección de marcas en remoto.**

En esta prueba, se ha desplegado el robot **AlphaBot** con:

- **Servicios**:
  - GPIO. Servicio de gestión de GPIO.
- **Componentes**:
  - Ruedas. Componente encargado de manejar motores de las ruedas.
  - Alphapantilt. Componente encargado de mover los servomotores de movimiento de la cámara.
  - Obstaculos. Componente encargado de obtener información del sensor infrarrojos para detección de obstáculos.
  - Alphaultrasound. Componente encargado de obtener información de la distancia hasta el próximo objeto con el sensor de ultrasonidos.
  - Apriltag. Componente encargado de obtener imágenes de la cámara y enviarlas a otro robot para que procese la imagen y devuelva si hay o no detecciones en la imagen.

Además, se ha desplegado en un equipo de altas prestaciones un **único componente encargado de recibir imágenes y detectar marcas de tipo *AprilTags***.

El robot **AlphaBot** comienza a mover sus motores de ruedas enviando su flujo de imágenes al equipo que tiene el detector de marcas, cuando el robot le indica que ha habido una detección, se lo comunica y lo almacena.

Los resultados de estas pruebas también han sido **satisfactorios.**

En el primer ejemplo, debemos de tener en cuenta que el hardware no es el mismo en ambos robots, con lo cual, un movimiento de un robot no equivale exactamente al movimiento del otro. Los servomotores que tienen los robots son diferentes, al igual que el diámetro de las ruedas, por lo tanto, la velocidad que pueden adquirir los robots son diferentes.

5.1.3. Colaborativas

Por último, también se han realizado pruebas de carácter colaborativo en la cual un robot se comunica con el otro y comparten información y, por ende, conseguir un objetivo común.

- **Búsqueda de etiquetas *AprilTags* de forma conjunta.**

En esta prueba se combina la robótica distribuida y colaborativa, es una ampliación del ejemplo anterior, pero siendo colaborativo.

Al robot anterior **AlphaBot** se la ha añadido un nuevo componente denominado **apriltag_controller,** el cual está suscrito al componente **apriltag** desplegado en el propio robot y además al componente **apriltag desplegado en otro robot** (como veremos posteriormente). Además, es el encargado de mostrar la información y parar la ejecución cuando se alcance el objetivo.

Por otro lado, contamos con el robot **LearnBot** que está compuesto por:

- **Servicios**:
  - USBSerial. Servicio de gestión de Arduino.
- **Componentes**:
  - Ruedas. Componente encargado de manejar motores de las ruedas.
  - Pantilt. Componente encargado de mover los servomotores de movimiento de la cámara.
  - Obstaculos. Componente encargado de obtener información del sensor ultrasonidos para detección de obstáculos.

- Apriltag. Componente encargado de obtener imágenes de la cámara y enviarlas a otro robot para que procese la imagen y devuelva si hay o no detecciones en la imagen.
- Apriltag_controller. Mismo componente que el definido para AlphaBot.

Para ello, haciendo uso de los dos robots anteriormente mencionados y del equipo explicado en el ejemplo distribuido, encargado de procesar imágenes en busca de etiquetas *AprilTags*, se ha realizado un ejemplo de búsqueda de etiquetas en un entorno cerrado.

Disponemos de un entorno cerrado dónde hay colocadas un número determinado de etiquetas, el objetivo de la prueba **es encontrar todas las etiquetas en el menor tiempo posible.**

Los dos robots comienzan su movimiento de búsqueda, al mismo tiempo que se mueven, ambos mandan la imagen al equipo encargado de detectar etiquetas en la imagen. Cuando un robot ha encontrado una etiqueta, la almacena y publica de forma periódica esta información a sus suscriptores. Como ambos robots están **suscribiendo y publicando de forma mutua** (entre los diferentes componentes apriltag y apriltag_controller), conocen en todo momento que información han obtenido entre ambos, es decir, cuantas marcas han visto entre los dos.

Si esta prueba no fuera colaborativa, cada robot estaría buscando todas las marcas sin tener en cuenta que ha encontrado el otro, en cambio, se tiene en cuenta cuando una marca ya ha sido vista.

Los resultados han sido **satisfactorios** y la prueba ha funcionado correctamente. Debemos tener en cuenta **para líneas futuras** poder realizar publicación y suscripción de forma mutua entre dos componentes, en este ejemplo se ha necesitado un componente auxiliar para suscripciones.

Actualmente, si tenemos un componente en cada robot realizando publicación y a la vez suscripción entre ellos, estamos ante un problema ya que no comenzaría el componente hasta que el otro esté disponible y el otro no comenzará hasta que el primero esté disponible, con lo cual, se precisa de un **segundo componente adicional** que haga de suscriptor y reciba la información.

5.2. Resultados generales

Tras meses de trabajo en este *framework,* se ha conseguido obtener una base estable y robusta con la que componer equipos robóticos. **PYRO4BOT** trabaja con diferentes tipologías robóticas, con un funcionamiento adecuado y un desarrollo sencillo.

Después de estudiar otros entornos de desarrollo y comprobar sus carencias: complejidad elevada, desarrollo exclusivo para un robot determinado, coste elevado el hardware, coste elevado del software, entorno privativo, no existencia de una comunidad de desarrollo libre, podemos considerar el *framework* desarrollado como una gran alternativa debido esencialmente a su simplicidad.

La creación de robots es cuestión de minutos siempre y cuando los componentes que se necesiten para manejar el hardware estén desarrollados. Durante este TFE se han desarrollado una gran cantidad de componentes encargados de manejar hardware, pero como es obvio, no hay realizado un componente por cada dispositivo hardware que existe en el mundo. Si un usuario tiene la necesidad de usar un hardware que no está contemplado, deberá de crear un componente exclusivo para el manejo del mismo y añadirlo al conjunto de componentes.

La conectividad con otros dispositivos ha resultado un éxito, las comunicaciones son estables gracias al uso de la API PYRO. El uso de **objetos remotos en Python** consideramos que puede llegar a ser un nuevo paradigma de programación debido esencialmente a su sencillez. El potencial que obtenemos al poder usar un objeto remoto como si de un objeto local se tratase es notorio. Además, la API abstrae al desarrollador de aspectos relacionados con las comunicaciones, como pueden ser puertos, direcciones IP, localizaciones, etc. Un desarrollador tan sólo necesita indicar hacia dónde necesita conexión y si existe en la red, tendrá conexión mediante un proxy como si un objeto más se tratase. Esta simplicidad se consigue también gracias también al uso del lenguaje *Python.*

Las pruebas realizadas son un pequeño ápice de todo lo que el sistema podría llegar a hacer, es una primera versión que confirman que el sistema funciona y que tiene potencial para poder ser viable en el mundo de la **robótica educativa.**

5.3.Líneas futuras de desarrollo

En líneas futuras de desarrollo, proponemos:

A. La creación de una **interfaz gráfica** que permita diseñar un robot educativo de forma sencilla. Permitiendo establecer la configuración de componentes y/o servicios para finalmente **desplegar el robot.**
B. Mejorar el sistema de **publicación / suscripción** remota. Actualmente hay que indicar una serie de configuraciones para realizar publicación y suscripción, pero, consideramos que debería de existir una vía de hacerlo más sencillo, sobre todo enfocado para niveles más básicos. Además, no es posible realizar publicación/suscripción mutua entre dos componentes ya que uno espera por el otro y el otro por el uno, con lo cual, ninguno de ellos inicia.
C. En aras de conseguir el uso de PYRO4BOT a niveles básicos proponemos el uso de lenguajes de programación de tipo gráfico como ***Scratch***, aislando completamente al usuario de código de programación Python. De esta forma se consigue generar más interés en el usuario final.
D. En lo que respecta a *Big Brother*, se le podría otorgar mayor funcionalidad y mayor importancia en el marco de desarrollo. Pudiendo incluso a almacenar todos los aspectos relacionados con comunicaciones entre robots.
E. Por último, se debe intentar la consecución de una **comunidad desarrollo** y pensamos que el camino a seguir es la realización de una gran **biblioteca de componentes** que permita usar casi cualquier hardware y que, además, anime a otros usuarios a añadir los suyos propios. Idealmente, se podría establecer un **canal de comunicación con la comunidad para añadir a los componentes ya creados.**

6
CONCLUSIONES

Tras el desarrollo de este proyecto, podemos observar el gran potencial que tiene el uso de **objetos remotos en lenguaje Python aplicado a la composición de robots**. Estos robots son de bajo coste y asequibles para casi cualquier persona, gracias a PYRO4BOT pueden satisfacer su curiosidad y **ser un punto de inicio para un sistema más avanzado y complejo.**

Actualmente, el desarrollo de equipos robóticos está creciendo a pasos agigantados pero todos ellos con una complejidad muy alta, complejidad que lleva meses o incluso años de trabajo conseguir entender por parte de expertos en la materia. PYRO4BOT no es ni será un sistema para equipos robóticos especializados ni avanzados sino, un nexo de unión entre una sociedad que no conoce qué es un robot ni mucho menos cree ser capaz de programar uno. PYRO4BOT no solo ofrece un marco de desarrollo para bases robóticas actuales, sino que, debido a la naturaleza del lenguaje, debido a su carácter abierto y debido a la filosofía con la que fue creado, permitirá desarrollar bases robóticas futuras.

PYRO4BOT ofrece comunicaciones a través de la API de PYRO4, a través de ella se consigue comunicar objetos remotos como si de objetos comunes se tratase. En consecuencia, permite tratar un componente robótico remoto como propio, sin saber dónde se encuentra ni qué robot lo está ejecutando. De la misma forma ofrece otras formas de comunicaciones tradicionales como publicación / suscripción.

Gracias a estas comunicaciones, cualquier persona puede tener un enjambre de robots sin complejidad ni dificultad y sin la necesidad de tener conocimientos de bajo nivel. No es necesario saber qué es una dirección IP, o un puerto de comunicaciones ni el nombre del robot que contiene el componente, simplemente necesita conocer el componente que necesita comunicar. Un usuario puede conectar un robot con otro u otros muchos solamente indicando el nombre del componente, con este nombre, el sistema internamente realiza unas complejas comprobaciones y configuraciones que son completamente transparentes al desarrollador.

Aun así, la sencillez del sistema permite conocer que entender que está ocurriendo a bajo nivel. Si el usuario tuviera la necesidad de modificar el código puede hacerlo a su antojo, es completamente libre de ello.

7

REFERENCIAS BIBLIOGRÁFICAS

[1] mBlock, «mBlock - Program Robots,» 2017. [En línea]. Available: http://www.mblock.cc.

[2] LEGO, «LEGO Mindstorm EV3,» 2017. [En línea]. Available: https://www.lego.com/es-es/mindstorms.

[3] QBO, 2017. [En línea]. Available: http://wiki.ros.org/Robots/Qbo.

[4] The Corpora, «The Corpora Website,» 2017. [En línea]. Available: http://thecorpora.com.

[5] RoboLab, «RoboComp,» 2018. [En línea]. Available: https://github.com/robocomp/robocomp.

[6] Wikipedia, «Ice - Wikipedia,» 18 05 2018. [En línea]. Available: https://en.wikipedia.org/wiki/Internet_Communications_Engine.

[7] ROS, «Robot Operating System,» 2018. [En línea]. Available: http://www.ros.org/.

[8] Wikipedia, «ROS - Wikipedia,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico.

[9] ROS - Wiki, «ROS - WikiRos,» 2018. [En línea]. Available: http://wiki.ros.org/ROS/Concepts#ROS_Computation_Graph_Level.

[10] ROS, «ROS2 - Wiki Website,» 2018. [En línea]. Available: https://github.com/ros2/ros2/wiki.

[11] Wikipedia, «Data Distribution Service - Wikipedia,» 2018. [En línea]. Available: https://en.wikipedia.org/wiki/Data_Distribution_Service.

[12] ROS, «Ros on DDS,» 2018. [En línea]. Available: http://design.ros2.org/articles/ros_on_dds.html.

[13] Python, «Python Website,» 2018. [En línea]. Available: https://www.python.org/.

[14] Wikipedia, «Python - Wikipedia,» 27 04 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Python.

[15] Python, «Pyro - Python,» 2018. [En línea]. Available: https://pythonhosted.org/Pyro4/.

[16] AprilTags Visual Fiducial System, «AprilTags,» 18 03 2015. [En línea]. Available: https://april.eecs.umich.edu/software/apriltag/.

[17] Wikipedia, «Toyota Partner Robot,» 6 06 2015. [En línea]. Available: https://es.wikipedia.org/wiki/Toyota_Partner_Robot.

8
ANEXOS

Anexo 1: Estado inicial del proyecto

1. Funcionamiento inicial

Inicialmente partimos de un producto base (hardware y software) burdo, de difícil manejo, sin comunicaciones externas y sin posibilidad de **autonomía de cada uno de sus componentes.**

Dada una configuración, y mediante una serie de directrices, esta versión conseguía desplegar una serie de procesos para ejecutar los componentes que se quisieran.

El robot quedaba desplegado, pero no era robusto, era propenso a fallos y la curva de aprendizaje era alta. Carecía además de capacidad de comunicaciones externas. Se necesitaba un conocimiento elevado del sistema y los fallos eran frecuentes.

Para simplificar y unificar el uso del mismo, se había creado un fichero en formato JSON el cual contenía toda la configuración que necesita saber el robot para poder arrancar.

Un fichero genérico JSON, seguía la siguiente estructura:

```
{
"NODE":
        {
        "name": “<nombre del robot>”,
        "path": “<ruta donde se encuentra PYRO4BOT”,
        "etc": "<path>/etc",
        "path_cls": ["<carpeta1 clases en ficheros>", "["<carpeta2 clases en
        ficheros>"],
        "ethernet": "<interfaz de red a usar>",
        "name_server": true,
        "port_ns": 9090,
        "port_node": 4040,
        "start_port": 5050,
        "def_frec": 0.03,
        "def_worker": true,
        },
"services":
        {
        "<servicio 1>": {"cls": "<clase servicio1>", “atributo 1”: <valor>, "atributo 2":
<valor>, …},
        "<servicio 2>": {"cls": "<clase servicio 2>", “atributo 1”: <valor> , …}
        },
"components":
        {
        "<comp 1>": {"cls": “<clase comp 1>”, “atributo 1”: <valor> , …}},
        "<comp 2>": {"cls": “<clase comp 2>”, “atributo 1”: <valor> , “atributo 2”:
<valor> , …},
        "<comp 3>": {"cls": “<clase comp 3>”,  “atributo 1”: <valor> , …},
         }
}
```

NODE:

Se trata de la configuración referida al nodo. Son atributos de configuración para iniciar el nodo y toda su configuración.

- **Name**. Es el nombre que recibirá nuestro robot y, por tanto, será el nombre que será visible hacia toda la red.
- **Path.** Indica al lanzador la ruta dónde se encuentra el proyecto.
- **Etc.** Indica al lanzador parámetros de configuración.
- **Path_cls.** Indica al lanzador el nombre de las subcarpetas dónde se encuentra los componentes y servicios.
- **Ethernet.** Indica al lanzador la interfaz de red a usar

- **Name_server.** Acepta valores “true” o “false” e indica si se requiere el uso de un servidor de nombres.
- **Port_ns**. Indica el puerto para el servidor de nombres si este fuera “true”.
- **Port_node.** Indica el puerto para el nodo.
- **Start_port.** Indica el puerto de inicio de los componentes.
- **Def_frec.** Indica la frecuencia de trabajo.
- **def_worker.** Acepta valores “true” o “false” e indica si se comienza a funcionar de inicio.

services:

Dentro de servicios, se incluían aquellos servicios locales que está ejecutando el robot.

Entendemos como **servicios** a aquellos ficheros que son necesarios de forma interna, un ejemplo de ello sería el servicio de la placa de **arduino**.

components:

En este apartado se incluían aquellos componentes (sensores, actuadores…) que tiene nuestro robot. Cada **componente tiene un nombre y además tiene una serie de atributos** escritos en el fichero JSON.

Tanto servicios como componentes, siguen la misma estructura a la hora de ser definidos.

En primer lugar, el **nombre** que queramos otorgarle al componente. A continuación, recibe el atributo “**cls**” cuyo valor será la **clase** que queramos ejecutar. El resto de atributos son opcionales.

Cada atributo escrito en un componente pasará a ser atributo de la instancia de clase creada.

Estos **atributos opcionales** se usan cuando, por ejemplo, tenemos configurado un componente para usar una cámara. Podemos hacer que este componente sea configurable sin la necesidad de usar *hard code*. Podemos escribir estos atributos en el fichero JSON de configuración (por ejemplo: anchura, altura, tasa de *frames* por segundo…) y tomar estos atributos dentro del código.

Además, existe un atributo especial que indica **dependencia** del componente con otro componente o servicio. Para ello, usamos la expresión **“--->”** que **indica que ese**

componente depende de otro y que solo puede funcionar si tiene conexión con el otro.

El funcionamiento del JSON en profundidad se explicará más adelante, ya que, este mismo JSON sufrirá cambios a medida que avance este proyecto.

Fichero de configuración de ejemplo:

```
{
"NODE":
        {
        "name": “learnbot”,
        "path": “/home/crivac/PYRO4BOT/pyro4bot”,
        "etc": "<path>/etc",
        "path_cls": ["services","components"],
        "ethernet": "wlan0",
        "name_server": true,
        "port_ns": 9090,
        "port_node": 4040,
        "start_port": 5050,
        "def_frec": 0.03,
        "def_worker": true,
        },
"services":
        {
        "picam": {"cls":"picam","ethernet":"<ethernet>","width":640,"height":480},
        "usbserial": {"cls": "usbserial", "comPort": "/dev/ttyS0",
                        "comPortBaud":11520, "frec": 0.01}
        },
"components":
        {
        "laser": {"cls":"laser", "LASER": [0,0,0], "-->":[" usbserial "], "frec":0.01},
        "infrared": {"cls":"infrared", "IR": [0,0,0,0], "-->":[" usbserial "],  "frec":0.02},
        "basemotion": {"cls":"basemotion", "BASE": [0,0], "-->":[" usbserial "],
"frec":0.03},
        "pantilt": {"cls":"pantilt", "PT": [90,90], "-->":[" usbserial "], "frec":0.03}
        }
}
```

En este caso, se trata de un robot llamado ***learnbot*** que se encuentra en un ***path*** determinado, que contiene sus servicios y componentes en ***services*** y ***components,*** que usa la tarjeta de red ***wlan0*** y que va a usar **servidor de nombres** con unos puertos determinados.

Además, consta de **dos servicios**:

- **picam**. Usa el fichero con la clase ***picam***, usa un atributo ethernet que es el mismo que se ha definido previamente y tiene ancho y alto configurable.
- **usbserial**. Usa el fichero con la clase ***usbserial,*** usa el atributo ***comPort*** cuyo valor es ***/dev/ttySO*** y usa el atributo ***comPortBaud*** cuyo valor es ***11520,*** finalmente tiene otro atributo más denominado ***frec*** que tiene el valor ***0.01.***

Los atributos definidos en estos servicios estarán disponibles con el valor indicado dentro del objeto indicado. Dicho de otro modo, en la clase ***picam*** existen los atributos ***ethernet, width y height*** ya que el sistema se encarga de guardar éstos **como atributos de la instancia de clase.**

Lo mismo ocurre en el caso de ***usbserial***.

Por otro lado, consta de **cuatro componentes:**

- **laser**. Usa la clase **laser.** Tiene dos atributos que estarán declarados posteriormente en la instancia de la clase.
- **infrared**. Usa la clase **infrared.** Tiene dos atributos que estarán declarados posteriormente en la instancia de la clase.
- **basemotion**. Usa la clase **basemotion.** Tiene dos atributos que estarán declarados posteriormente en la instancia de la clase.
- **pantilt**. Usa la clase **pantilt.** Tiene dos atributos que estarán declarados posteriormente en la instancia de la clase.

2. Ejecución

Una vez entendido cómo debe de ser configurado el robot, procederemos a una **primera ejecución del mismo**, haciendo uso de la configuración anteriormente explicada.

Para poder ejecutar el robot necesitábamos tener conexión a la Raspberry Pi, para ello realizamos una conexión mediante **SSH y ejecutamos el fichero de lanzamiento con el JSON anteriormente explicado.**

El robot funciona y está listo para recibir peticiones.

3. Cliente

Para realizar una conexión inicial, es necesario crear un fichero en lenguaje Python. En dicho fichero debemos importar la clase **clientNODERB**. Esta clase, **sabiendo la URI del robot**, establecía una conexión con el robot y te permitía acceder a sus componentes en ejecución.

Anexo 2: Monitorización Raspberry

Aunque no es estrictamente necesario, se ha instalado un pequeño monitor para conocer el estado en el que se encuentra el hardware. En este anexo se explicará de forma breve como ha sido instalado.

En primer lugar, instalamos los paquetes **nginx, php-fpm y php-xml.** Una vez instalados, modificamos el fichero **/etc/nginx/sites-available/default**

Quedando la parte del server como se indica a continuación:

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        # SSL configuration
        # …
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.php index.html index.htm;

        server_name pyro4;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
        }

        location ~ /(conf)/ {
         deny all;
        }

        # pass PHP scripts to FastCGI server

        location ~ \.php$ {
                include snippets/fastcgi-php.conf;
                include fastcgi_params;
                fastcgi_pass unix:/run/php/php7.0-fpm.sock;

        }
}
```

A continuación, modificamos los permisos:

```
sudo chown -R www-data:pi /var/www/html/
sudo chmod -R 770 /var/www/html/
```

La parte del servidor ya se encuentra instalada, para acabar, instalamos el monitor.

```
cd /var/www/html/
sudo wget --content-disposition http://www.ezservermonitor.com/esm-
web/downloads/version/2.5
unzip ezservermonitor-web_v2.5.zip
sudo rm ezservermonitor-web_v2.5.zip
mv eZServerMonitor-2.5/ monitoring/
nano monitoring/conf/esm.config.json
```

Finalmente reiniciamos los servicios:

```
sudo /etc/init.d/php7.0-fpm restart
sudo /etc/init.d/nginx restart
```

Y accedemos a la IP del dispositivo para poder visualizar la información.

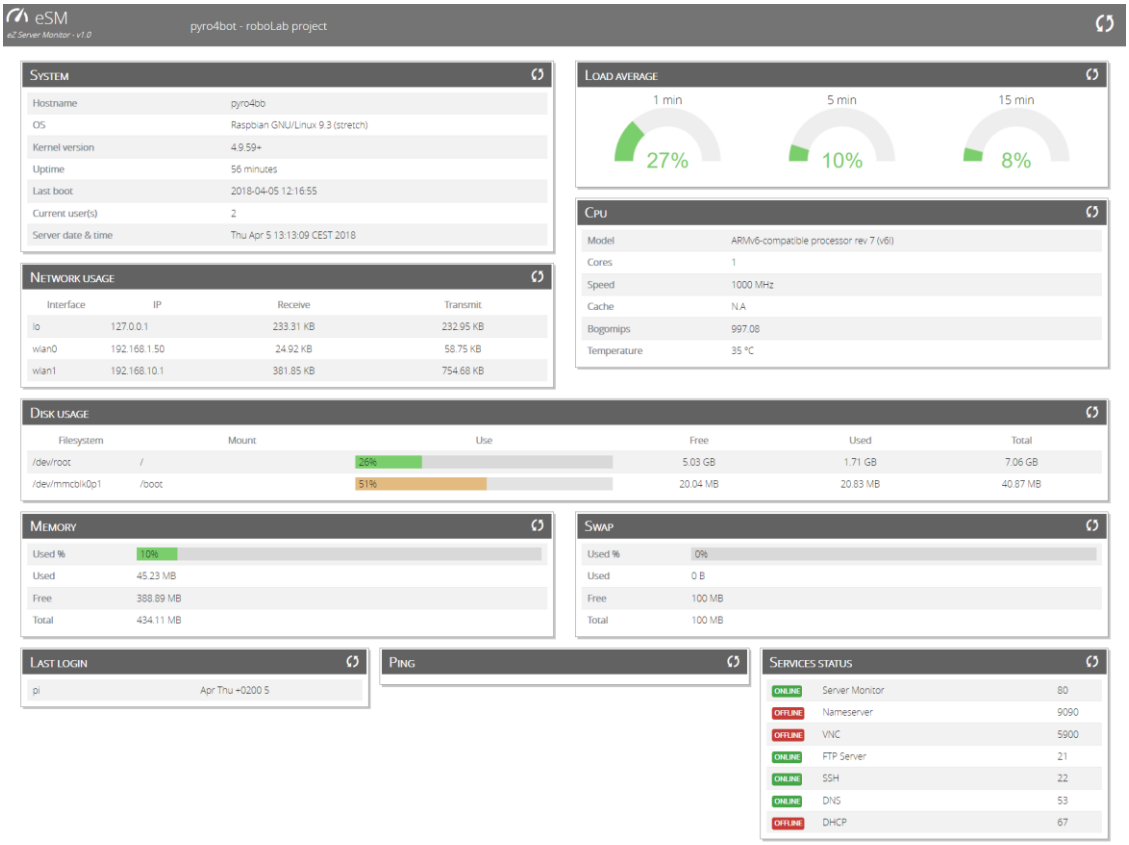

Figura 16: Monitorización de hardware

Anexo 3: Configuración Raspberry Pi Zero para *Big Brother*

Para evitar problemas de conectividad en distintas redes, se ha decidido usar esta plataforma para la **creación de una red WiFi**.

Raspberry Pi Zero tiene una **tarjeta de red WiFi interna**, además, se ha añadido una **segunda tarjeta de red WiFi por USB**. Se usará la tarjeta de red interna para conectarse a una red WiFi (si existe) y la tarjeta de red externa para la creación de la red WiFi para los robots. De esta manera, si tuviéramos una red WiFi conectada a la tarjeta de red interna automáticamente se puentearía la conexión hacia la WiFi creada, otorgándole así de una **salida a Internet.**

En primer lugar, se modifica el fichero **/etc/dhcpcd.conf** añadiendo las siguientes líneas:

```
denyinterfaces wlan1
denyinterfaces wlan0

# Static IP for wlan1
interface wlan1
static ip_address=192.168.10.1/24
static routers=192.168.10.1
static domain_name_servers=8.8.8.8 8.8.4.4
```

En las líneas anteriores no obtenemos IP por DHCP en esas dos interfaces, y además indicamos al cliente DHCP una IP estática para esa interfaz. La red a usar será la **192.168.10.0**

Además, modificamos el fichero **/etc/network/interfaces.** En este caso estamos indicando que nuestra tarjeta de red interna se conectará a una red WiFi conocida cuya dirección de red es **192.168.1.0,** además, se conectará usando la IP **192.168.1.50.** La configuración de contraseñas para redes WiFi se encuentra en el fichero **/etc/wpa_supplicant/wpa_supplicant.conf**

```
auto wlan0
auto lo
iface lo inet loopback
iface eth0 inet manual
allow-hotplug eth0

# wlan0 share internet. main nic.
allow-hotplug wlan0
#iface wlan0 inet dhcp
```

```
iface wlan0 inet static
address 192.168.1.50
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

# wlan1 create ap. secondary usb nic.
allow-hotplug wlan1
iface wlan1 inet static
address 192.168.10.1
netmask 255.255.255.0
network 192.168.10.0
gateway 192.168.10.1
```

El fichero **/etc/wpa_supplicant/wpa_supplicant.conf** también necesita seguir configurado para que la interfaz de red interna pueda conectarse a una red WiFi con acceso internet.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=ES
network={
        ssid="<nombre-red-wifi>"
        psk="<password>"
        key_mgmt=WPA-PSK
}
```

Dónde **<nombre-red-wifi>** será el SSID de nuestra red wifi y **<password>,** la contraseña asociada.

Una vez configuradas las interfaces, pasaremos a configurar el modo **router.**

Para ello, instalamos el paquete **hostapd.** Una vez instalado, modificamos el fichero **/etc/hostapd/hostapd.conf** y escribimos la siguiente configuración:

```
interface=wlan1
driver=nl80211
ssid=pyro4router
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
```

```
wpa=2
wpa_passphrase=90racano90
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

En el fichero anterior, indicamos que la interfaz a usar será **wlan1,** que se usará el nombre indicado en el parámetro **ssid** y la contraseña indicada en **wpa_passphrase.**

Una vez escrita la configuración, indicamos **hostapd** dónde se encuentra nuestro fichero de configuración. Modificamos el fichero **/etc/default/hostapd.** Reemplazamos **#DAEMON_CONF** por **DAEMON_CONF="/etc/hostapd/hostapd.conf"**

La red WiFi ya está creada, pero ahora es necesario dar **servicios DNS y DHCP a dicha red.** Para ello, instalamos el paquete **dnsmasq.** Una vez instalado, modificamos el fichero **/etc/dnsmasq.conf** otorgándole la siguiente configuración:

```
interface = wlan1
listen-address=192.168.10.1
bind-interfaces
server=8.8.8.8
domain-needed
bogus-priv
dhcp-range=192.168.10.50,192.168.10.99,255.255.255.0,24h
```

Finalmente, configuramos **las tablas de enrutamiento** (*IPTables*) y el **reenvío de IP** (*IP forwarding*).

Modificamos el fichero **/etc/sysctl.conf** y descomentamos la línea referida a *IP forwarding*, quedando dicha línea así:

```
net.ipv4.ip_forward=1
```

Modificamos las tablas de enrutamiento, para ello debemos ejecutar los siguientes comandos por consola:

```
sudo iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
sudo iptables -A FORWARD -i wlan0 -o wlan1 -m state --state RELATED,ESTABLISHED
-j ACCEPT
sudo iptables -A FORWARD -i wlan1 -o wlan0 -j ACCEPT
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Para acabar, modificamos el fichero **/etc/rc.local** y añadimos al final las siguientes líneas:

```
printf "I am Big Brother"
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
iptables-restore < /etc/iptables.ipv4.nat
exit 0
```

Por último, iniciamos los servicios y reiniciamos la Raspberry Pi Zero.

```
sudo service hostapd start
sudo service dnsmasq start
sudo reboot
```

Al reiniciar, la Raspberry Pi Zero ya tendría funcionando las dos tarjetas de red, creando una red WiFi denominada "**pyro4router"** cuya contraseña sería la indicada en **wpa_passphrase** en el fichero **/etc/hostapd/hostapd.conf**