UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

"Diseño e implementación de un controlador centralizado para un exoesqueleto de extremidades inferiores"

Tesis previa a la obtención
del título de Ingeniero de Sistemas

Autores:

Cristina Alejandra Balcázar Maldonado. CI. 0105287965
Elio Santiago Quevedo Silva. CI. 0604422220

Directores:

Ing. Miguel Ángel Zúñiga Prieto. CI. 0102498052
Ing. Darwin Fabián Astudillo Salinas. CI. 0103907036

Cuenca - Ecuador
23 de abril de 2019

Resumen

Existen diversos tipos de enfermedades o eventos que afectan la movilidad de las extremidades inferiores de una persona. Estas condiciones afectan a su calidad de vida y a su capacidad para desplazarse de manera independiente. La rehabilitación física es uno de los tratamientos que permiten que estas personas recuperen su movilidad funcional. Ante la necesidad de una terapia eficiente y consistente, los exoesqueletos han constituido un complemento al tratamiento de rehabilitación. Estos dispositivos reducen el esfuerzo físico del fisioterapeuta y proveen herramientas que le permiten preocuparse por la recuperación del paciente.

Los exoesqueletos constan de actuadores colocados en las articulaciones para generar las trayectorias de movimiento. Al mismo tiempo, los sensores adquieren información del paciente, como las señales de electromiografía y electroencefalografía, o de los componentes de *hardware*, como el voltaje de un paquete de baterías y la posición de los motores. En este caso, los sistemas de control son los encargados de coordinar el funcionamiento de los componentes del exoesqueleto y garantizar la ejecución de una terapia de rehabilitación. Para que los fisioterapeutas interactúen con el exoesqueleto, requieren de un dispositivo de mando a manera de control remoto o de una aplicación que se ejecuta en dispositivos móviles o computadoras personales.

El exoesqueleto *Autonomous Lower Limb Exoskeleton* (ALLEX-1) desarrollado en la Universidad de Cuenca requiere un sistema de control que coordine y dirija sus funcionalidades. Por este motivo, en el presente trabajo de titulación se diseña e implementa un sistema de control centralizado para un exoesqueleto de extremidades inferiores. Este objetivo se consigue mediante el desarrollo de tres subsistemas: *AllexApp*, como una aplicación móvil para el control de alto nivel que le permite al usuario enviar comandos al exoesqueleto y gestionar las sesiones de rehabilitación; *AllexControl* como el subsistema embebido que se encarga de la coordinación de sensores y actuadores; y el sistema auxiliar *AllexServer* que es un servidor web para gestionar las sesiones de rehabilitación de un paciente.

Palabras clave: Exoesqueleto. Sistema de control. Interacción humano-máquina.

Abstract

There are several diseases and events that have an impact on lower limb mobility. These medical conditions affect people's quality of life and their ability to move independently. Physical therapy is one of the treatments that allows the recovery of the patient's functional mobility. Given the need of an efficient and consistent therapy, exoskeletons have been a complement to rehabilitation treatment. These devices reduce the physical effort of the physiotherapist and provide tools that allow them to concentrate on the patient's recovery.

Exoskeletons consist of actuators placed close to the joints in order to generate movement trajectories. At the same time, sensors acquire patient information, such as electromyography and electroencephalography signals, or hardware components, such as the voltage of a battery pack and the position of the motors. In this case, the control system is the responsible for the coordination and operation of the exoskeleton components so that it guarantees the execution of a rehabilitation therapy. Physiotherapists have to interact with the exoskeleton, for that reason, they require a remote-control device or an application that runs on mobile devices or personal computers.

The Autonomous Lower Limb Exoskeleton (ALLEX-1), developed at the University of Cuenca, requires a control system that coordinates and supervises its functionalities. This undergraduate thesis presents the design and implementation of a centralized control system for a lower extremity exoskeleton. This objective is achieved through the development of three subsystems: AllexApp, as a mobile application for high-level control that allows the user to send commands to the exoskeleton and manage rehabilitation sessions; AllexControl, as the embedded subsystem that is responsible for the coordination of sensors and actuators; and AllexServer, an auxiliary web server that manages patients' rehabilitation sessions.

Keywords: Exoskeleton. Control system. Human-machine interaction.

Índice general

Índice de figuras.

Índice de tablas.

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Cristina Alejandra Balcázar Maldonado en calidad de autora y titular de los derechos morales y patrimoniales del trabajo de titulación "Diseño e implementación de un controlador centralizado para un exoesqueleto de extremidades inferiores", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 17 de Abril del 2019

Cristina Balcázar M.

Cristina Alejandra Balcázar Maldonado

C.I: 0105287965

Cláusula de Propiedad Intelectual

Cristina Alejandra Balcázar Maldonado, autor del trabajo de titulación "Diseño e implementación de un controlador centralizado para un exoesqueleto de extremidades inferiores", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 17 de Abril del 2019

Cristina Alejandra Balcázar Maldonado

C.I: 0105287965

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Elio Santiago Quevedo Silva en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Diseño e implementación de un controlador centralizado para un exoesqueleto de extremidades inferiores", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 17 de Abril del 2019

Elio Santiago Quevedo Silva

C.I: 0604422220

Cláusula de Propiedad Intelectual

Elio Santiago Quevedo Silva, autor del trabajo de titulación "Diseño e implementación de un controlador centralizado para un exoesqueleto de extremidades inferiores", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 17 de Abril del 2019

Elio Santiago Quevedo Silva

C.I: 0604422220

Dedicatoria

A mis padres Freddy y Teresa, por todo su amor, por lo valores que me impartieron, por enseñarme a no rendirme, por cada uno de los sacrificios que hicieron para que cumpla mis metas y principalmente por su apoyo incondicional que me ha permitido llegar a cumplir un sueño más.

A mi hermana Yessenia y sobrina Samantha, por estar siempre presentes, por acompañarme y por todo el apoyo que me brindaron a lo largo de esta etapa en mi vida.

A toda mi familia y amigos que me apoyaron desde el inicio, que aportaron a mi formación tanto profesional e hicieron de mi una mejor persona y de una u otra forma me acompañan en todos mis sueños y metas.

Cristina Balcázar M.

Dedicatoria

El presente trabajo de titulación constituye una muestra de agradecimiento al esfuerzo y sacrificio diario de mis padres, Marcos y Karina, para proveer todos los valores, recursos y herramientas necesarias para mi formación humana, académica y profesional. Sobre todo, agradezco sus recomendaciones, motivación, apoyo y uno que otro jalón de orejas que me permitieron seguir adelante en esta etapa que culmina, así como en mis futuras metas. A mi hermano Darío, por el apoyo brindado cuando lo necesitaba.

A mis abuelitos y tíos quienes, a pesar de la distancia, me han brindado su apoyo en cada tarea que he ejecutado a lo largo de mi carrera.

A todas aquellas personas, tanto amigos como docentes, que con sus conocimientos, experiencias y consejos han contribuido en la construcción de mi persona.

Elio Quevedo.

Agradecimientos

El presente trabajo de titulación, representa una labor conjunta de quienes formaron parte del proyecto de investigación "*Prototipo de exoesqueleto usable en las extremidades inferiores*". Agradecemos al Ing. Ismael Minchala, director del proyecto de investigación, por la confianza brindada al permitirnos contribuir en parte del desarrollo de una de sus ideas para proveer una terapia de rehabilitación a personas con dificultades en la caminata.

De la misma manera, agradecemos a nuestros directores, Ing. Miguel Ángel Zuñiga e Ing. Fabian Astudillo, por la guía brindada durante el desarrollo del sistema y del documento con el objetivo de mejorar su calidad.

Al Ing. Esteban Mora, por sus recomendaciones y apoyo en el diseño de las funcionalidades del sistema. De la misma manera, a Lic. María Ayavaca y Lic. Verónica Cárdenas por su punto de vista clínico del funcionamiento de un exoesqueleto y sus funcionalidades para la rehabilitación.

A nuestros compañeros del Exo Team, posteriormente rebautizado como Allex Team, por el conocimiento, experiencias y problemas compartidos, así como su amistad.

Finalmente, agradecemos a nuestros familiares, amigos y docentes que nos han apoyado durante este proceso.

Cristina Balcázar, Elio Quevedo

Acrónimos

ALLEX-1 Autonomous Lower Limb EXoskeleton.

API Application Programming Interfaces.

BDD Block Definition Diagram.

CONADIS Consejo Nacional para la Igualdad de Discapacidades.

CORBA Common Object Request Broker Architecture.

DDS Data Distribution Service.

DDSI DDS Interoperability Wire Protocol.

EEG Electroencefalografía.

EMG Electromiografía.

IDL Interface Definition Language.

IFR International Federation of Robotics.

OMG Object Management Group.

QoS Quality of Service.

RCL ROS Client Library.

REST Representational State TRansfer.

RFCOMM Radio Frequency Communication.

ROS Robot Operating System.

ROSIDL Robot Operating System Interface Definition Language.

RTOS Real-Time Operating System.

RTPS Real-Time Publisher-Subscriber Wire Protocol.

SBC Single-Board Computer.

SysML Systems Modeling Language.

UML Unified Modeling Language.

UUID Universally Unique IDentifier.

YAML YAML Ain't Markup Language.

Capítulo 1

Introducción

En el presente capítulo se contextualiza la problemática asociada al desarrollo de un sistema de control centralizado para un exoesqueleto de extremidades inferiores. Esto incluye la definición de los objetivos y contribuciones de la solución de *software* propuesta en este proyecto de titulación. Primero, se describen los antecedentes que motivan la ejecución del proyecto (sección 1.1). Esto sirve como base para identificar el problema que se debe resolver y las contribuciones de la solución de *software* a ser implementada (sección 1.2). A continuación, se enuncian los objetivos que se deberán cumplir para solucionar el problema identificado (sección 1.3) y se presenta la metodología de desarrollo aplicada en el proyecto (sección 1.4). El capítulo culmina con una visión general de la estructura del documento (sección 1.5).

1.1. Antecedentes y Motivación

La acción de caminar (caminata) es una actividad de gran valor para la calidad de vida de las personas [1]. Sin embargo, puede ser afectada debido a un accidente o por la aparición de diversas patologías de origen neurológico o cardiovascular. Este es el caso de enfermedades como la apoplejía o las lesiones en la columna vertebral y el cerebro. En el Ecuador, el Consejo Nacional para la Igualdad de Discapacidades (CONADIS) reportó que 458.505 personas se encontraban registradas con algún tipo de discapacidad, de las cuales el 46.72 % tenía alguna discapacidad física [2].

La rehabilitación física es un apoyo para el tratamiento de aquellas personas (pacientes) que padecen de patologías que afectan la movilidad de sus extremidades inferiores. El objetivo de este tratamiento es que los pacientes puedan recuperar

cierto nivel de independencia en la ejecución de sus actividades cotidianas. Durante una terapia, los fisioterapeutas guían y controlan una rutina de actividades físicas que permiten que el paciente recupere la fuerza muscular y el control necesario para que pueda caminar de manera segura [3]. Uno de los dispositivos que pueden ser usados como complemento al tratamiento de rehabilitación son los exoesqueletos, en el caso de la caminata, aquellos diseñados para extremidades inferiores.

Los exoesqueletos para extremidades inferiores son sistemas robóticos constituidos por diferentes tipos de actuadores y sensores que son coordinados por una unidad de control. Los actuadores son dispositivos encargados de producir el movimiento sobre las articulaciones basándose en la información cinética y/o biológica adquirida a través de los sensores. Este conjunto de interacciones asegura la reproducción de movimientos propios de la locomoción humana y el almacenamiento de datos derivados de la misma. El objetivo de estos sistemas robóticos es "*incrementar la habilidad ambulatoria de una persona que sufre de una patología en la pierna y proveer de mecanismos que aumenten la fuerza de una o más articulaciones de las extremidades inferiores*" [4]. Según [5], los exoesqueletos son adecuados como complemento para las terapias de rehabilitación porque permiten un entrenamiento motor orientado a tareas intensivas y repetitivas. Al mismo tiempo que se reduce el esfuerzo físico del fisioterapeuta, le permite también concentrarse en monitorizar la recuperación de la movilidad funcional del paciente.

Varias han sido las propuestas para la construcción de exoesqueletos para extremidades inferiores. Una de ellas es el proyecto de investigación "*Prototipo de exoesqueleto usable en las extremidades inferiores*", ganador del XIV concurso de proyectos investigación convocado por la Dirección de Investigación de la Universidad de Cuenca (DIUC). Durante este proyecto se desarrolló el exoesqueleto bautizado como ALLEX-1 (*Autonomous Lower Limb Exoskeleton 1*). ALLEX-1 fue diseñado como un exoesqueleto bilateral con seis grados de libertad que permite el desplazamiento del paciente en el plano sagital. El presente proyecto de titulación parte del desarrollo de ALLEX-1 para proveer un *software* de control centralizado que permita su funcionamiento.

1.2. Planteamiento del Problema

Según el estudio sistemático realizado por [6], hasta el año 2014 se han documentado cincuenta tipos de sistemas robóticos para la rehabilitación de extremidades inferiores. Estos sistemas han sido distribuidos de manera comercial, o en su defecto, son prototipos que todavía están en períodos de investigación. Algunos de estos trabajos presentan la arquitectura de *hardware* y *software* usada durante el desarrollo e implementación de sus sistemas de control.

Con respecto al hardware empleado, se han usado circuitos impresos que han sido diseñados de manera independiente por los laboratorios de investigación de

varios exoesqueletos. Este es el caso de los exoesqueletos *HYPER* [7] y *Hybrid Neuroprosthesis (HNP)* [8], los cuales poseen placas para la adquisición, transmisión y procesamiento de datos. Por otro lado, también existen propuestas para el uso de computadoras de placa reducida de distribución comercial. Este es el caso de los exoesqueletos H2, que usa una *Raspberry Pi* [9], y *Biomot* [10] que usa una placa denominada *BeagleBoneBlack*. En relación al *software* para la implementación del sistema, se ha usado sistemas operativos de tiempo real como *xPC* junto con *Simulink* en el exoesqueleto presentado en el trabajo [11] o el sistema operativo *QNX* en un exoesqueleto con un grado de libertad para la rodilla [12]. Una desventaja es que muchos de los trabajos no poseen información sobre el *software* usado en el desarrollo de su controlador o emplean sistemas que requieren de licencias para su implementación.

Para su uso en la rehabilitación, se requieren dispositivos que permitan la interacción entre el fisioterapeuta y el exoesqueleto. Estos pueden ser controles alámbricos con un conjunto de botones y palancas, como en el caso de los exoesqueletos *Rex* [13] y *Ekso-GT* [14]. La ventaja de este tipo de dispositivos es que la comunicación con el exoesqueleto es ejecutada en tiempo real; sin embargo, se vuelve difícil alterar o agregar nuevas funcionalidades. Otra de las alternativas corresponde a aplicaciones de escritorio para computadoras personales como la que se presenta en [15] para el exoesqueleto *C.S.One* y *ALTACRO* [16]. Finalmente, se pueden usar interfaces gráficas para dispositivos móviles como en el caso del exoesqueleto *H2* [9]. A pesar de que estos dispositivos permiten controlar el movimiento de los exoesqueletos, no gestionan la información de las terapias de rehabilitación de los pacientes.

La construcción del exoesqueleto ALLEX-1 ha sido fragmentada en diferentes trabajos de investigación [17–19], cada uno de los cuales soluciona un conjunto de problemas asociados con la estructura física del exoesqueleto, el sistema de comunicación, el manejo de actuadores, la fuente de energía y la utilización de señales biológicas para detectar la intención de movimiento. Sin embargo, cada una de estas tareas ha sido ejecutada de manera independiente. Como resultado, no existe una estructura de control que integre y coordine el funcionamiento de los componentes del exoesqueleto con el objetivo de ejecutar ejercicios de rehabilitación para las extremidades inferiores.

Otro problema a considerar es que el exoesqueleto es un producto dirigido a fisioterapeutas, quienes son los profesionales encargados de planificar y personalizar las terapias de rehabilitación. Por tanto, requieren de una interfaz hombre-máquina (Human-Machine Interface - HMI) que les permita interactuar con este dispositivo robótico. Estas herramientas deben incluir funcionalidades como la configuración de una sesión de rehabilitación u obtener retroalimentación sobre el avance de la terapia.

Por estos motivos, la contribución del presente trabajo de titulación es el desarrollo

del *software* de control centralizado para ALLEX-1. Esto incluye el *software* que permita la comunicación de datos y eventos de control entre los componentes de *hardware* y el desarrollo de una interfaz gráfica que facilite la interacción de los fisioterapeutas con el exoesqueleto. Esta interfaz gráfica debe permitir la gestión de las sesiones de rehabilitación y movimientos del exoesqueleto.

1.3. Objetivos

El objetivo del presente trabajo de titulación es diseñar e implementar una solución de interfaz hombre-máquina que facilite el control e interacción con fines terapéuticos de un exoesqueleto para extremidades inferiores.
Este objetivo se descompone en los siguientes objetivos específicos:

1. Identificar los requerimientos del proyecto "Prototipo de exoesqueleto usable en las extremidades inferiores" para el control centralizado de los componentes electro-mecánicos de un exoesqueleto y de su interacción con los usuarios.
2. Diseñar e implementar un sistema de *software* para el control de los componentes de un exoesqueleto para extremidades inferiores.
3. Diseñar e implementar una interfaz hombre-máquina que permita la interacción de los usuarios, para fines terapéuticos, con un exoesqueleto.
4. Evaluar la solución propuesta a través de pruebas de ejecución de movimientos en un exoesqueleto.

1.4. Metodología

En el presente proyecto de titulación, se utiliza un proceso de desarrollo apoyado en las etapas de la ingeniería de *software* basada en componentes. Se considera la ejecución de las siguientes actividades de trabajo:

1. **Identificación y formulación del problema:** Etapa en la cual se identifica el problema a ser resuelto para posteriormente definirlo de manera precisa.
2. **Investigación bibliográfica:** Etapa en la cual se recopila información sobre el diseño y programación de sistemas robóticos. Se extrae información sobre las funcionalidades comunes de los exoesqueletos para rehabilitación y las interfaces gráficas que proveen. Además, se identifican las soluciones actuales y los problemas que presentan estos sistemas robóticos.
3. **Investigación de herramientas de desarrollo:** Se estudian y comparan las herramientas de implementación con el objetivo de escoger aquellas que puedan ser beneficiosas para el desarrollo del proyecto.
4. **Definición de la infraestructura:** En esta etapa se definen las herramientas de *software* a ser usadas para el desarrollo e implementación del sistema de control. Esto incluye la documentación, instalación y elaboración de

aplicaciones de prueba que permitan determinar si la aplicación es ejecutable en el ambiente operativo.

5. **Requerimientos de Aplicación:** Se realizan reuniones con los participantes del proyecto con el objetivo de determinar las funcionalidades que debe cumplir el sistema.
6. **Diseño de componentes:** En esta etapa se divide al sistema en componentes y se especifica su funcionalidad e interfaces.
7. **Implementación e integración:** Se programan los componentes del sistema de control y se evalúa su integración con el resto del sistema. Esto puede derivar en el mantenimiento o corrección de los componentes existentes.
8. **Evaluación de la ejecución:** Se evalúa la funcionalidad del *software* desarrollado con los componentes de *hardware* que han sido construidos para ALLEX-1.

1.5. Estructura del Documento

En este capítulo se ha presentado la motivación, objetivos y procesos requeridos para el desarrollo de un sistema de control centralizado para el exoesqueleto ALLEX-1. El resto del proyecto de titulación se organiza en los capítulos representados en la figura 1.1 y que se detalla a continuación:

- **Capítulo 2.** Marco Tecnológico y Trabajos Relacionados.
  En este capítulo se presentan los conceptos relacionados a la robótica y su aplicación en la medicina de rehabilitación mediante el uso de exoesqueletos. A continuación, se detallan los componentes de la arquitectura de *software* y la infraestructura de las aplicaciones robóticas. Finalmente, se exponen algunas decisiones de diseño e implementación que sustentan el presente trabajo.
- **Capítulo 3:** Sistema de Control Centralizado para ALLEX-1.
  En este capítulo se presenta la especificación de requerimientos para el sistema de control centralizado. Este *software* está compuesto por tres subsistemas, estos son: una aplicación para dispositivos móviles denominada *AllexApp*, el sistema embebido para el control del exoesqueleto llamado *AllexControl* y el servidor web denominado *AllexServer*.
- **Capítulo 4.** Diseño del Sistema de Control.
  En este capítulo se presentan las decisiones de diseño estructural de los componentes de cada subsistema y la dinámica de sus interacciones mediante el lenguaje de modelado de sistemas (*SysML*).
- **Capítulo 5.** Implementación del Sistema de Control.
  En este capítulo se presentan las herramientas de *software* empleadas para la programación, configuración y despliegue de cada subsistema de ALLEX-1.
- **Capítulo 6.** Pruebas de Funcionamiento.
  En este capítulo se presenta el funcionamiento del sistema desarrollado en el presente trabajo de titulación a través de escenarios de uso.

- **Capítulo 7.** Conclusiones y Trabajos Futuros.
  En este capítulo se presentan las contribuciones del presente trabajo de titulación y las líneas de investigación o desarrollo futuro que pueden derivarse de la implementación del sistema de control centralizado.
- **Anexo A.** Casos de Uso.
  En este anexo se formalizan los requerimientos de los subsistemas a manera de tablas de definición de casos de uso.
- **Anexo B.** Definición de Servicios Web.
  En este anexo se describen los servicios web que son provistos por el servidor de aplicaciones.
- **Anexo C.** Configuración de *AllexControl*.
  En este anexo se presentan recomendaciones y comandos requeridos para la instalación, configuración y ejecución del *software AllexControl* que es ejecutado en una *Raspberry Pi 3*.
- **Anexo D.** Archivos de Configuración.
  En este anexo se presentan los archivos creados para la configuración previa al despliegue del sistema de control centralizado, así como, la definición de movimientos para los ejercicios de rehabilitación.
- **Anexo E.** Manual de Usuario.
  En este anexo se presenta una guía con la funcionalidad de cada una de las pantallas de *AllexApp*.

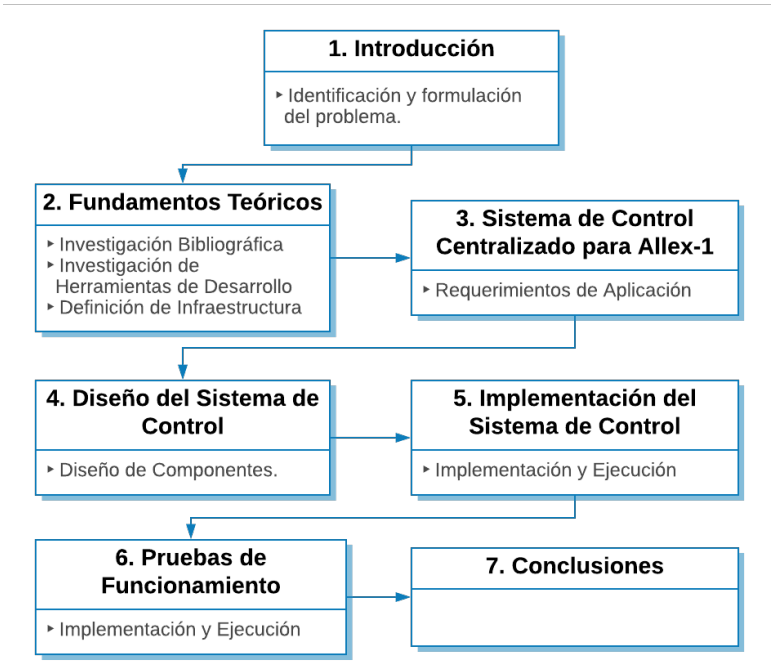


Figura 1.1: Estructura del documento.

Capítulo 2

Marco Tecnológico y Trabajos Relacionados

En este capítulo se presentan los conceptos requeridos para entender el presente trabajo de titulación, la base tecnológica que sustenta su proceso de desarrollo y los trabajos relacionados a la creación de sistemas de control y supervisión. La descripción inicia con una visión general sobre el mundo de la robótica, sus componentes y aplicaciones (sección 2.1). Como parte de esta clasificación, se presenta un tipo de robot médico para rehabilitación conocido como exoesqueleto (sección 2.2). A continuación, se enuncian los problemas del desarrollo de sistemas robóticos desde el punto de vista de *software* (sección 2.3). Con el objetivo de conocer la infraestructura de implementación de este tipo de sistemas, se detallan los componentes de un sistema embebido y se presentan algunas decisiones de diseño (sección 2.4). Particularmente se describen las características del *framework Robot Operating System 2 (ROS2)* (sección 2.4). Finalmente, se presenta la infraestructura y tipos de controles de mando usados en los exoesqueletos comerciales y en desarrollo (sección 2.5).

2.1. Robótica

Según [20], el objetivo de la robótica contemporánea es el de "*diseñar, controlar e implementar sistemas capaces de realizar una tarea específica de alto nivel*". En esta ciencia se incluye el estudio de tres funciones básicas: 1) la percepción como el mecanismo para obtener información sobre el ambiente, mediante el uso de sensores, 2) el proceso de planificación o control, en el cual los datos de los sensores y el conocimiento del problema generan directrices para ejecutar tareas, y 3) la actuación, funciones que controlan a los actuadores para cumplir con las directrices solicitadas [21]. De esta manera, se complementan o sustituyen las actividades tediosas, repetitivas o peligrosas que realizan los seres humanos [22].

La *Robotic Industries Association* (RIA) define a un *robot* como un *"manipulador reprogramable y multifuncional diseñado para mover materiales, partes, herramientas o dispositivos especializados usando trayectorias variables para ejecutar diversas tareas"* [23]. Por otro lado, La *International Federation of Robotics* (IFR) incluye el concepto de autonomía, entendiéndose como la capacidad de un sistema robótico para realizar tareas basándose en su estado actual y lo que percibe del medio, sin que existan intervenciones por parte de terceros [24]. En términos generales un robot es cualquier dispositivo compuesto por: 1) sensores que envían datos a una computadora o circuito integrado con capacidad de procesamiento. 2) la computadora que genera comandos en función de la información recibida por los sensores, y 3) los actuadores que ejecutan los comandos de la computadora.

Como conclusión, la propiedad fundamental de los sistemas robóticos es la de convertir información compleja en una acción física para realizar una tarea útil. Esta capacidad de reemplazar, complementar o transcender la actuación humana ha tenido una profunda influencia en muchos campos de la sociedad, incluyendo la producción industrial, exploración, control de calidad, actividades médicas y procesos de laboratorio.

Clasificación

Los robots pueden ser clasificados de acuerdo al ambiente en el que operan, como se detalla en la figura 2.1 o según su campo de aplicación, como se muestra en la figura 2.2.

1. **Según el ambiente operativo**

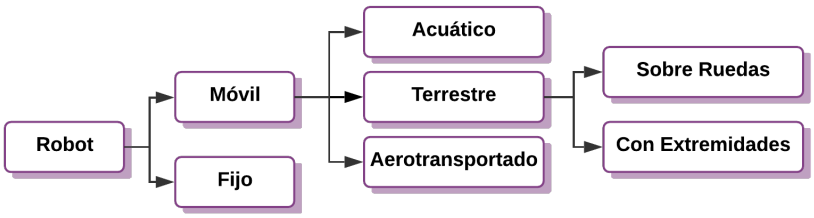


Figura 2.1: Clasificación de los robots según el ambiente de operación [25].

Los robots pueden agruparse en fijos y móviles bajo este criterio. La diferencia está en las características del entorno sobre el cual se desenvuelven y el tipo de trabajo que realizan. Los fijos son principalmente manipuladores robóticos industriales que trabajan en entornos conocidos y adaptados para facilitar su funcionamiento. Estas máquinas realizan actividades específicas y repetitivas, como soldar o pintar partes de un auto. Por otro lado, los robots móviles se desplazan y realizan tareas en entornos amplios y que presentan diversos grados de incertidumbre debido a que es un ambiente desconocido que incluye entidades impredecibles como los seres humanos, animales y otros tipos de obstáculos.

Los robots móviles basan su diseño en el ambiente sobre el que se movilizan: los acuáticos que utilizan la navegación, terrestres (como los autos) y aéreos (como los drones). Esta clasificación no es mutuamente exclusiva, ya que pueden existir máquinas que se muevan tanto sobre agua como en el suelo. Los robots móviles terrestres pueden ser divididos adicionalmente en dos clases: 1) robots con ruedas, y 2) aquellos que tienen extremidades para caminar y desplazarse. Este último tipo de robots se utilizan principalmente para el movimiento en ambientes no estructurados.

2. **Según el campo de aplicación**

Según el campo de aplicación previsto y las tareas que realizan, pueden clasificarse en industriales y de servicio como se presenta en la figura 2.2.

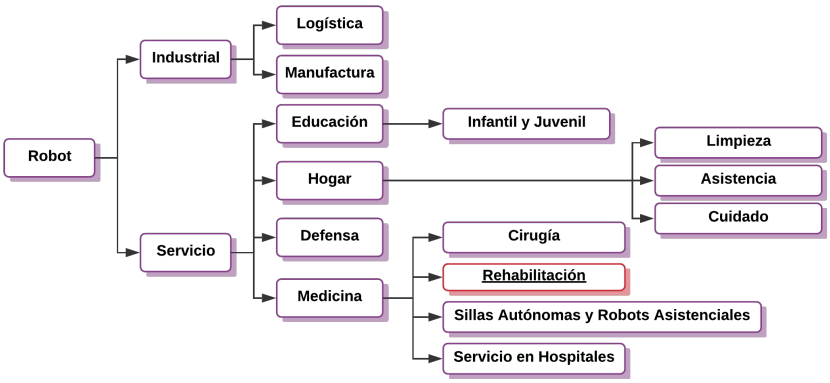


Figura 2.2: Clasificación de los robots según campo de aplicación. Basado en: [22, 25]

- **Robots de Servicio**

Corresponden a la transición en el uso de robots desde la industria hacia la entrega de servicios en casas, hoteles, hospitales, restaurantes, almacenes u otros tipos de tiendas. De esta manera los trabajadores tienen una máquina que apoye la ejecución de sus tareas de manera rápida, segura y efectiva [26]. Se debe destacar que el objetivo de estas máquinas es únicamente de apoyo, no buscan ocupar o reemplazar un puesto de trabajo.

A su vez este tipo de máquinas pueden ser de dos tipos: los robots de servicio personal y los de uso profesional. El primero es un robot para la ejecución de actividades no comerciales por parte de personas no profesionales en el área. Por otro lado, las de uso profesional son generalmente utilizadas por personal con entrenamiento previo. Los robots de servicio tienen aplicaciones en múltiples campos como la educación, defensa, actividades del hogar y la medicina. Particularmente, en este documento, es de importancia la aplicación médica de los robots.

- **Robots en la Medicina**

Los robots médicos de servicio profesional que permiten que los médicos realicen tratamientos o cirugías en pacientes de manera segura, eficiente y precisa. En los últimos años ha aumentado el uso de robots en otros campos de la medicina como la rehabilitación y el entrenamiento. Estos sistemas presentan fortalezas y limitaciones en su aplicación de acuerdo a los criterios que se enuncian en la tabla 2.1 a continuación:

	Fortalezas	**Limitaciones**
Humano	Excelente juicio. Excelente destreza. Capaz de integrase y actuar sobre múltiples fuentes de información. Versátil y capaz de improvisar.	Propenso a la fatiga y falta de atención. Control de movimiento limitado. Precisión geométrica limitada.
Robot	Excelente precisión geométrica. Incansable y estable. Puede diseñarse para operar a diferentes escalas de movimiento y carga útil. Capaz de integrar múltiples fuentes de datos numéricos y de sensores.	Falta de criterio. Difícil de adaptarse a nuevas situaciones. Destreza limitada. Capacidad limitada para integrar e interpretar información compleja.

Tabla 2.1: Fortalezas y limitaciones de los humanos y robots en la medicina [27].

Según [27], los robots médicos pueden clasificarse de muchas maneras ya sea por el diseño de sus manipuladores (cinemática, actuación), por el nivel de autonomía (preprogramado, teleoperación, control cooperativo restringido) o por su entorno operativo previsto (escáner, sala de operaciones convencionales).

Los robots en la medicina tienen aplicaciones en la rehabilitación, cirugía, servicios hospitalarios, sillas autónomas y asistencia al paciente. En el presente trabajo son de interés los robots para rehabilitación.

Los robots para rehabilitación son máquinas diseñadas para mejorar el movimiento de personas con una discapacidad física [28]. Para ello el sistema informático propone las tareas que se deben ejecutar y registra los avances del paciente. Esto permite que el fisioterapeuta pueda programar sesiones de rehabilitación para el paciente y evaluar su progreso [22, 29]. El proceso de diseño de este tipo de robots debe estar centrado en el usuario, con la participación de grupos de pacientes, fisioterapeutas y médicos a lo largo de su desarrollo.

2.2. Exoesqueletos

Un exoesqueleto es un robot de servicio profesional con aplicaciones destinadas a imitar, aumentar o mejorar los movimientos del cuerpo. Estas máquinas tienen diseños mecánicos con componentes hidráulicos o electromecánicos que proveen de mayor potencia y fuerza al movimiento [30]. Pueden ser usados en la rehabilitación, despliegue militar y aumento de fuerza. Por esta razón, su diseño integra diversas disciplinas como la medicina, electrónica, física y mecánica [31].

Según el tipo de usuario, los exoesqueletos pueden ser pasivos o activos. Los pasivos están orientados a usuarios sanos o con discapacidad parcial en donde las fuerzas del movimiento pueden ser aplicadas por el usuario. Los activos están orientados a usuarios con paraplejía, en donde las fuerzas necesarias para el movimiento son proporcionadas completamente por el exoesqueleto [30].

Existen dos aspectos que todavía requieren mejoras para reducir el esfuerzo o coste metabólico del uso de un exoesqueleto [32] : 1) mejorar la interacción entre el humano y exoesqueleto, y 2) mejorar el hardware y el sistema de control. Esto debido a que el rendimiento de un exoesqueleto aumenta con la potencia y disminuye con el peso.

2.2.1. Exoesqueletos para Rehabilitación

Los exoesqueletos son herramientas de asistencia que mejoran las cualidades físicas de un paciente con lesiones que le impiden moverse de manera autónoma. Este tipo de dispositivo tiene mecanismos poliarticulares capaces de generar fuerza y ayudar a la locomoción del paciente. El principal beneficio es el alto índice de efectividad en la recuperación de los rangos de movimiento funcional de la cadera, rodilla y tobillo [30, 33, 34].

Los exoesqueletos para extremidades inferiores son diseñados para cumplir tres funciones principales como son: aumentar, asistir o efectuar completamente el movimiento de las piernas. Particularmente las dos últimas funciones son aplicadas en la rehabilitación de personas con patologías o traumas que dificultan su independencia en la caminata. Este campo de investigación ha gozado de mayor desarrollo en los últimos años debido a los avances en la electrónica, mecánica, medicina y las ciencias de la computación. En efecto, según el estudio sistemático realizado por Calderón et al. en [6], desde el año 2004 al 2014 se han desarrollado cincuenta dispositivos robóticos para la rehabilitación de la caminata, de los cuales quince han sido evaluados en clínicas y el resto son prototipos en investigación.

En [5, 6] se realiza una distinción entre los exoesqueletos y otro grupo de robots para rehabilitación conocidos como efectores finales. Un efector final, como el sistema G-EO (figura 2.3a), es una máquina en la cual se coloca al paciente en un sistema de suspensión parcial de peso corporal mediante un arnés de cuerpo

completo. El objetivo es que la persona pueda estar de pie sobre plataformas en la base de la máquina las cuales simulan el ciclo de caminata o los movimientos de subir una escalera. El dispositivo reduce el esfuerzo requerido por los terapeutas para sujetar al paciente, pero su dificultad práctica está en el tamaño que ocupa.

Por tanto, se considerará como exoesqueleto únicamente a aquellos que causan movimiento sobre las articulaciones. Estos a su vez suelen clasificarse en estáticos y móviles. Los estáticos son similares a los efectores finales, ya que los pacientes pueden estar sujetos por arneses, pero caminan sobre cintas y el movimiento de la cadera y rodilla está controlada por rutinas de los actuadores. Un ejemplo es el exoesqueleto ReoAmbulator (figura 2.3b) con actuadores colocados en el muslo y pierna del paciente. Los exoesqueletos móviles le permiten al paciente desplazarse en el medio con mayor independencia. Consta de un conjunto de actuadores colocados a nivel de las articulaciones como cadera y rodilla, pudiendo extenderse hasta el tobillo. Por lo general requiere que el paciente se apoye en andadores o muletas. Un ejemplo de este tipo de máquina es ReWalk (figura 2.3c), uno de los primeros exoesqueletos bilaterales con autorización de la *Food and Drug Administration* (FDA) para ser usado en hogares y clínicas. Este exoesqueleto le permite al paciente caminar, girar y subir o bajar gradas. Se debe notar que todos estos tipos de exoesqueletos son complementarios, puesto que cada uno está diseñado para pacientes con diferentes patologías.

(a) Efector final: G-EO Systems [35]

(b) Exoesqueleto fijo: ReoAmbulator

(c) Exoesqueleto móvil: ReWalk

Figura 2.3: Tipos de robots para rehabilitación.

En [36], se describen los componentes comunes de los exoesqueletos como los sensores, actuadores y algoritmos de control:

Actuadores: Cerca del 72 % de los exoesqueletos usan motores eléctricos para efectuar el movimiento, siendo otras alternativas los sistemas hidráulicos y neumáticos. La preferencia por los motores eléctricos se deriva de la facilidad con la cual se realiza el control por posición.

Sensores: Los más comunes son aquellos que permiten determinar la posición de los motores, el torque o la fuerza. Estos tienen el objetivo de regular y controlar las trayectorias de movimiento. Otros incluyen sensores de fuerza generalmente en la planta del exoesqueleto para detectar el impacto con el suelo y de esa manera controlar las etapas del ciclo de marcha. Las investigaciones recientes están asociadas al uso de señales biológicas como las de electromiografía y electroencefalografía para detectar la intención de movimiento de los pacientes.

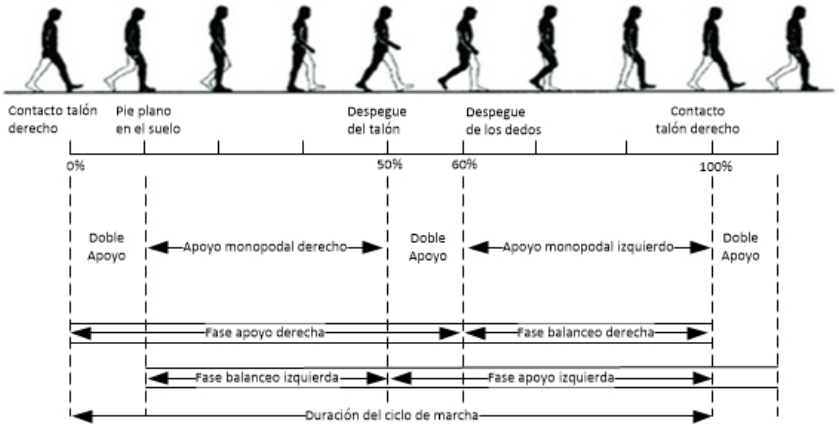
Control: Según [37], los algoritmos de control en los exoesqueletos están caracterizados por el modelo usado para describirlos, los parámetros y su estructura jerárquica. El modelo del algoritmo de control se refiere a la manera en que se representan las trayectorias de los movimientos. Se documenta el uso de modelos matemáticos de locomoción, aproximaciones matemáticas de las trayectorias y modelos basados en redes neuronales y aprendizaje automático. Otras entradas para el sistema de control dependen de los sensores disponibles como las señales de electromiografía, electroencefalografía, fuerza, etc. Se considera que el sistema de control en los exoesqueletos puede organizarse de manera jerárquica [37, 38].

2.2.2. Marcha Bípeda en Exoesqueletos

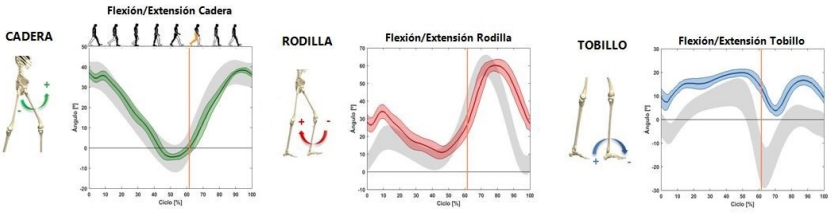
La marcha bípeda es la forma de desplazamiento que caracteriza al ser humano y lo diferencia del resto de las especies de animales [39]. Consiste en una secuencia de movimientos coordinados y alternantes que permiten que una persona pueda desplazarse [40]. La marcha normal puede ser modificada por muchos factores, sean estos externos (terreno, calzado, vestido, carga), internos (sexo, peso, altura, edad), psicológicos, físicos y patológicos. Los cambios que se dan en la marcha normal pueden ser transitorios o permanentes. En varias enfermedades que tienen origen neurológico la marcha se encuentra alterada causando problemas de equilibrio, coordinación, tono muscular o parálisis.

Fases de la Caminata: La marcha normal consta de una fase estática y otra dinámica. La fase estática o de apoyo constituye el 60 % de la misma y ocurre cuando una pierna sufre carga y está en contacto con el suelo. La fase de balanceo o dinámica constituye el 40 % y ocurre cuando avanza la otra pierna para dar el siguiente paso como se puede observar en la figura 2.4a.

Determinantes de la Marcha El movimiento de las articulaciones puede ser modelado mediante funciones de la posición angular de cada articulación respecto al tiempo en el plano sagital como se presenta en la figura 2.4b. En este caso, el objetivo del exoesqueleto es seguir dicha trayectoria de la biomecánica de la articulación para imitar el movimiento propio de la caminata o de otros movimientos propios de las extremidades inferiores.



(a) Fases de la caminata. Basado en: [41–44]



(b) Patrón de marcha en el plano sagital [45].

Figura 2.4: Dispositivos de mando conectados al exoesqueleto.

2.2.3. Autonomous Lower Limb Exoskeleton (ALLEX-1).

Autonomous Lower Limb EXoskeleton 1 (ALLEX-1) es el exoesqueleto usado durante el presente trabajo de titulación (figura 2.5). Corresponde a un robot de servicio profesional para terapias de rehabilitación que fue diseñado como un exoesqueleto móvil bilateral para extremidades inferiores, con seis grados de libertad que permite el desplazamiento del paciente en el plano sagital.

Al momento de la redacción de este documento únicamente está construida la extremidad izquierda del exoesqueleto. Esta fue desarrollada como parte del proyecto de titulación realizado por Blandín y Velasco en [17]. La estructura de esta extremidad consta de ejes de transmisión de torque que están fabricados en acero. Los actuadores corresponden a tres motores *Brushless Direct Current* (BLDC) colocados en reductores para la cadera, rodilla y tobillo. El control de posición es gestionado por el controlador EPOS4 50/8. Este controlador tiene integrado un sensor Hall para determinar la posición del rotor. El sistema de comunicación entre los controladores usa el protocolo *Control Area Network* (CAN) para la comunicación en la capa física. Finalmente, el programador de la aplicación puede interactuar con estos controladores de comunicación mediante la librería de comandos EPOS4 para C++. Esta librería gestiona un puerto de comunicación que usa el protocolo *CANOpen*.

También cuenta con una batería recargable de ion de litio colocada dentro de una mochila. Este sistema de administración de energía fue desarrollado como parte del proyecto de titulación de Mendieta y Muñoz [19], cuyo objetivo es otorgar autonomía e independencia energética a todos los componentes de hardware del exoesqueleto.

Adicionalmente, ALLEX-1 contempla el uso de señales biológicas como la electromiografía para detectar la intención de movimiento de un paciente y efectuar el ciclo de caminata. Esto es estudiado por Astudillo y Charry en su trabajo de titulación [18], en donde se detecta la intención de movimiento mediante redes neuronales. Debido a que pueden existir casos de pacientes patológicos en los cuales la intención muscular es baja o inexistente, se considera como alternativa el uso de señales electroencefalográficas.

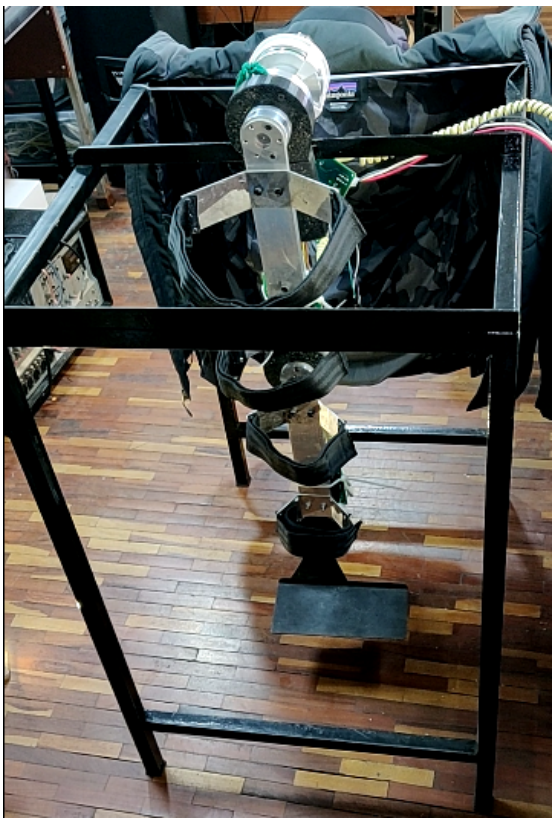


Figura 2.5: Exoesqueleto *Autonomous Lower Limb Exoskeleton* (ALLEX-1)

2.3. Software para Sistemas Robóticos

Los sistemas robóticos resultan de la integración de elementos de *hardware*, *software* y su interacción con el ambiente que los rodea (incluyendo la interacción humana). Su objetivo es ejecutar una tarea y ofrecer un servicio. Según [46], este tipo de sistemas consta de una capa de *hardware* y otra capa de *software*. La capa

de *hardware*, también denominada capa de control, es aquella que contiene los *drivers* para interactuar con los sensores, actuadores y unidades de procesamiento. Por otro lado, la capa de *software*, o de comportamiento, es aquella que utiliza las interfaces provistas por la capa de control para ejecutar una tarea compleja. Esta capa de *software* se puede separar en tres niveles de abstracción adicionales de acuerdo a las recomendaciones para sistemas robóticos de Brugali [47] y que se resumen en la figura 2.6. Estos niveles son:

- **Componentes horizontales:** Se refiere a aquellos componentes de *software* que se encargan de la interacción directa con los componentes de *hardware*, librerías de adquisición o procesamiento de datos, simulación y comunicación entre procesos. Este tipo de componentes se los llama horizontales porque pueden ser usados para resolver problemas independientemente de su dominio de aplicación. Su código es poco flexible, debido a que su implementación y generalidad están restringidas por las funcionalidades, interfaces y limitaciones propias del proveedor de los elementos de *hardware*.

- **Componentes verticales:** Los componentes verticales son generalmente algoritmos cuyos parámetros de entrada provienen de los datos adquiridos de los sensores, del estado general del sistema o de comandos de interfaces externas. La salida de estos algoritmos causa un cambio en el estado del sistema, envían comandos a los actuadores o generan variables que serán de utilidad para otros servicios. Ésta es la capa que define el comportamiento y utilidad de los componentes horizontales.

- **Componentes de aplicación:** Son aquellas interfaces externas que configuran al robot, cambian su estado y reportan los resultados de la ejecución de comandos o los datos adquiridos. En este tipo de sistemas se incluyen las aplicaciones para dispositivos móviles, control por servicios web, control remoto, mecanismos de interacción humano máquina, etc.

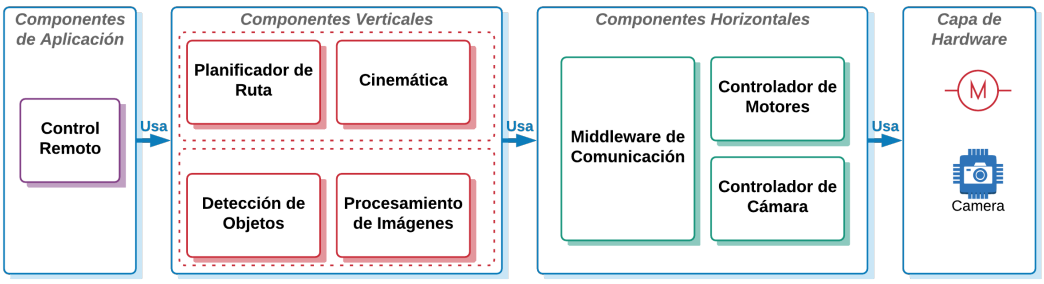


Figura 2.6: Niveles de abstracción de un sistema robótico [46, 47].

2.3.1. Desafíos del Diseño e Implementación de Software para Sistemas Robóticos

El desarrollo de *software* para robots enfrenta los mismos desafíos que son tratados por la ingeniería de *software*, es decir, la necesidad de documentar, diseñar e implementar soluciones teniendo en consideración las limitaciones de tiempo, costo y mantenimiento. Sin embargo, existen otros problemas asociados a la interacción entre componentes y la incertidumbre que se genera en los ambientes sobre los cuales actúa la máquina. Algunos de estos problemas son señalados por Brugali en dos de sus trabajos [48, 49] como son:

Variabilidad de hardware y software: La robótica es un campo en continua evolución que integra conocimientos de diferentes áreas de investigación. Por este motivo, está sujeto a cambios en la configuración y número de sensores o actuadores, o en su defecto, a la adición de nuevos comportamientos para resolver problemas más complejos. Como consecuencia, el dispositivo puede interactuar con su entorno de mejor manera. Por ejemplo, en el desarrollo de un *rover* para la NASA [50], se detalla que la mayor preocupación en la capa de *software* es cómo conseguir que las interfaces y algoritmos sean lo suficientemente generales, sin que esto tenga como consecuencia una degradación en la calidad y rendimiento del sistema. Algunas de las soluciones consisten en usar un *framework* que facilite la integración de componentes y definición de interfaces, así como descomponer la funcionalidad del sistema en pequeños módulos. En la descomposición se debe usar como criterio de división el comportamiento que deberá tener el robot en lugar de los detalles de su implementación.

Transición de la simulación al ambiente operativo: Simular consiste en usar modelos matemáticos, algoritmos, gráficos por computadora o eventos que permiten observar el comportamiento del *software* de un robot en diferentes ambientes, sin necesidad de interactuar directamente con el *hardware*. La ventaja de la simulación está en la detección temprana de errores de programación o diseño, selección de un algoritmo o parámetros óptimos para los casos de uso planteados y simulación de la física de un ambiente. Gazebo, RVIZ, V-REP, RobotSim, Webots y OpenHRP son ejemplos de este tipo de plataformas. Los simuladores generalmente proveen herramientas para el diseño gráfico del dispositivo, definición de la física del ambiente e interfaces para interactuar con el ambiente simulado. A pesar de los beneficios que tiene la simulación, ésta añade complejidad al sistema puesto que se requiere mantener el código del ambiente de prueba y el de producción, así como las configuraciones del simulador. Como resultado cualquier cambio en los componentes reales implicaría que la validez de la simulación puede degradarse o incluso perderse [49].

2.3.2. Separación de Asuntos

De lo descrito anteriormente, se puede concluir que el principal interés en el diseño de un sistema robótico es alejarse de las arquitecturas de *software* monolíticas, mismas que no son lo suficientemente dinámicas para adaptarse a los nuevos requerimientos de los sistemas. Por este motivo se introduce el concepto de la *separación de asuntos* (del inglés *separation of concerns*), el cual es un principio de diseño que busca separar al sistema en unidades lógicas con funcionalidades o abstracciones similares. De esta manera se facilita la evolución, adaptabilidad y comprensión del sistema [51]. En los trabajos de [52, 53] se detallan las cinco características que deben ser consideradas para conseguir la separación de asuntos en un sistema robótico: el cómputo, comunicación, coordinación y configuración como los elementos que dan flexibilidad al sistema, y por otro lado, la composición que garantiza que todas las demás características sean consistentes al otorgarles un comportamiento predecible. Se debe buscar un beneficio medio entre la generalidad del sistema y los factores que determinan su cohesión.

2.3.2.1. Cómputo

Es la unidad que provee la funcionalidad al sistema mediante la transformación de datos [53]. Según [48], esta transformación está definida en función a tres elementos: la estructura de datos, operaciones y comportamiento. La estructura de datos es la información que es modificada por el algoritmo, ésta puede ser de entrada o de salida. Las operaciones son los algoritmos que se encargan de la lectura, modificación o escritura de las estructuras de datos y que como resultado causan un cambio en el estado del sistema. El último elemento es el comportamiento que se refiere a la lógica con la cual se ejecuta una secuencia de operaciones de tal manera que se provee el servicio al sistema. Como resultado, esta unidad esconde los detalles de implementación al exponer únicamente la estructura de datos de entrada y salida. Adicionalmente, no se requiere especificar el mecanismo por el cual los datos llegan a la unidad de cómputo.

2.3.2.2. Comunicación

Son todos aquellos mecanismos de intercambio de datos entre los componentes computacionales, incluyendo características de calidad de servicio como la disponibilidad, ancho de banda, difusión, retransmisiones, entre otras. Algunos de los mecanismos o patrones de comunicación incluyen los de tipo solicitud-respuesta y publicación-suscripción. Se debe agregar que la elección de una de las arquitecturas de comunicación no excluye el uso de otras.

La comunicación solicitud-respuesta es un mecanismo de intercambio de mensajes entre dos participantes en donde cada mensaje requiere una respuesta por parte del destinatario. Como se observa en la figura 2.7, este proceso consta de tres pasos:

1. El remitente (*cliente*) del mensaje solicita un servicio al destinatario (*servidor*).
2. El servidor procesa los parámetros recibidos y genera una respuesta.
3. El cliente usa dicha respuesta para continuar con su proceso.

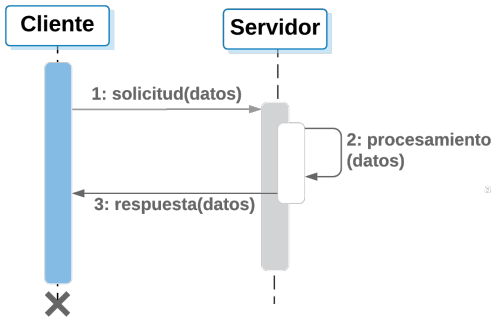


Figura 2.7: Diagrama de secuencia de la comunicación solicitud-respuesta

En el modelo publicador-suscriptor existen tres participantes: el publicador o remitente del mensaje, el *broker* o mediador y el suscriptor o destinatario del mensaje. El funcionamiento de este patrón se resume en la figura 2.8:

1. El o los suscriptores se registran ante el *broker*, indicando que están interesados en recibir aquellos datos que son publicados en un tema (o *topic*) determinado.
2. El publicador envía un mensaje al *broker* en el cual especifica una estructura de datos y el tema en el cual se publica el mensaje.
3. El *broker* notifica o envía la estructura de datos a todos aquellos suscriptores que previamente se hayan registrado en el tema.
4. Al recibir el mensaje, el o los suscriptores ejecutan una función (denominada *callback*) que inicia el procesamiento de la estructura de datos recibida.

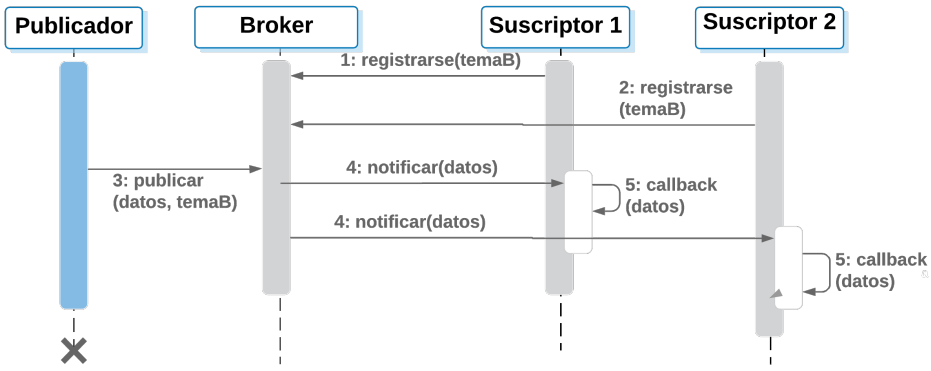


Figura 2.8: Diagrama de secuencia de la comunicación publicador-suscriptor

La ventaja de este tipo de comunicación es que el emisor del mensaje no requiere conocer a los destinatarios o gestionar la calidad del servicio. Lo mismo es cierto en

el caso de los suscriptores, no requieren conocer si existe un publicador, únicamente reaccionan a los mensajes enviados por el *broker*. Como resultado se consigue un sistema desacoplado y que facilita la extensibilidad y flexibilidad a los cambios de la topología de los componentes computacionales.

2.3.2.3. Coordinación

Cada unidad de cómputo se ejecuta de manera independiente y concurrente, como consecuencia, se requiere de un mecanismo que controle que el comportamiento de todo el sistema sea consistente. Esto se consigue al coordinar una máquina de estados finitos para cada componente [52]. Según [48] existen dos modelos de coordinación:

- **Dirigido por los datos:** El estado general del sistema depende de los datos que son intercambiados por las unidades de cómputo. En este caso, el coordinador puede ser implementado dentro de cada unidad de cómputo.
- **Dirigido por el control:** El estado general del sistema depende del estado de cada unidad de cómputo. En esta configuración, todos los componentes monitorizan sus estados entre sí. Para facilitar la implementación se requiere de un componente que controle los estados de todo el sistema.

2.3.2.4. Configuración

Es toda aquella representación de información que es usada para modificar o personalizar el comportamiento, comunicación o topología del sistema. La configuración puede ser alterada mediante archivos, llamadas a servicios, parámetros, estructuras de datos en memoria compartida, etc., mismas que permiten modificar el sistema durante la compilación y/o ejecución.

2.3.2.5. Composición

Es el elemento que modela la estructura de los componentes del sistema y sus relaciones. Por tanto, especifica las características que deben tener las unidades de configuración, comunicación, coordinación y cómputo. Su función es la de especificar y acoplar estas unidades, permitiendo que su comportamiento sea predecible.

En la figura 2.9 se detalla un ejemplo, que, de manera genérica, muestra la estructura del patrón de composición que ha sido adoptada por varios *frameworks* de robótica. Cada composición tiene una unidad de cómputo la cual recibe o envía datos a otra unidad mediante el componente de comunicación. Los componentes de comunicación conocen el tipo de dato que se está transfiriendo y han acordado su calidad de servicio. El configurador actúa como un servicio que puede ser llamado desde del módulo de comunicación o desde el de cómputo. Los coordinadores notifican de eventos o cambios de estado en el componente local. También puede existir un coordinador global que notifique o controle los cambios en todo el sistema.

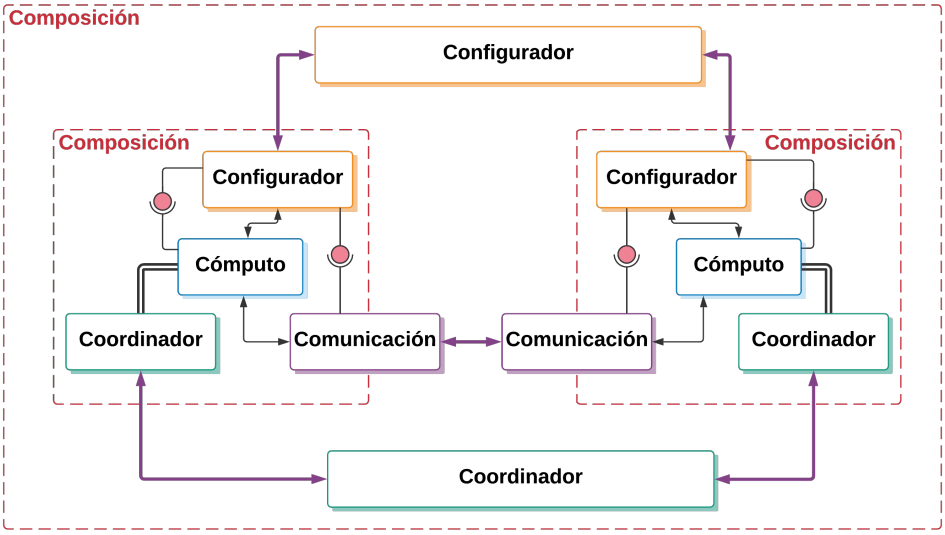


Figura 2.9: Estructura de una composición. Basado en: [53]

2.3.3. Sistemas de Control

Un sistema de control es un conjunto de componentes cuya configuración y datos de entrada generan la respuesta que se espera del sistema [54]. Para conseguir este objetivo, un sistema de control consta de componentes como los sensores, actuadores y controladores.

Aparte de los componentes de control se requiere un objetivo o una referencia como entrada para el controlador, tal y como se presenta en la figura 2.10. La referencia es un valor numérico que es usado para medir el error respecto a una variable que se requiere controlar. Sin embargo, esta referencia también puede ser un objetivo o estado que el controlador deberá cumplir al ejecutar una secuencia de procesos.

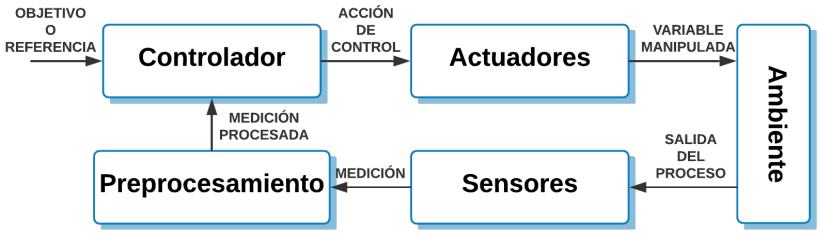


Figura 2.10: Componentes de un sistema de control [55].

Control Jerárquico: A medida que aumenta la complejidad de un sistema, se requiere de un mecanismo que permita coordinar los subsistemas de control. El primer paso es dividir el problema en componentes más pequeños e interconectar-

los de manera jerárquica. Según Findeisen en [56], las arquitecturas de control jerárquico tienen dos estructuras:

- **Multicapa:** Consiste en algoritmos separados por capas, cada una de las cuales tiene un algoritmo de control más específico y que actúa en diferentes intervalos de tiempo.
- **Multinivel:** El objetivo que debe cumplir todo el sistema está dividido en objetivos locales, los cuales deben estar coordinados.

En la figura 2.11 se representa el modelo de control jerárquico de tipo multinivel, mismo que está distribuido en n niveles con diferentes grados de abstracción. La diferencia entre cada nivel está en el tiempo de procesamiento y la representación de la información. Los niveles de control inferiores procesan datos de tipo cuantitativo a una mayor frecuencia, esto debido a que sus resultados requieren de una mayor precisión. Los niveles superiores trabajan a una menor frecuencia y obtienen información integrada desde los niveles inferiores. Esto les permite encargarse de tareas como:

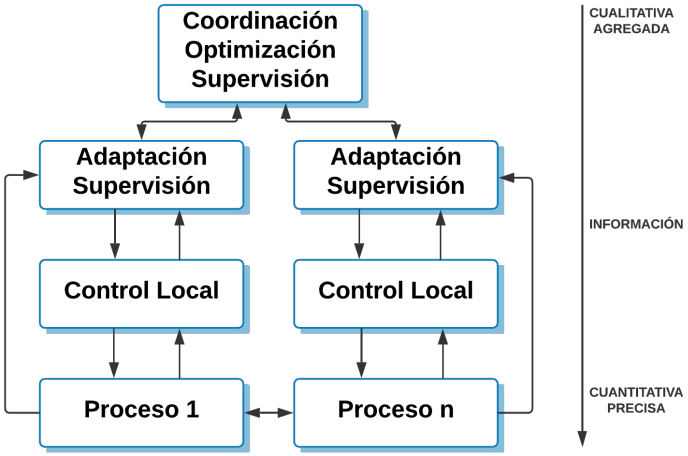


Figura 2.11: Arquitectura jerárquica de control [55].

- **Coordinación:** Componente encargado de dirigir el funcionamiento, configuración y referencias de los componentes de niveles inferiores de tal manera que en conjunto cumplan el objetivo deseado en el sistema.
- **Optimización:** Se refiere a componentes que ejecutan algoritmos que optimizan el valor de una variable. Generalmente están asociados al control adaptativo o al aprendizaje máquina.
- **Supervisión:** Componente que verifica el estado de funcionamiento de los elementos de control local, verifica si los resultados obtenidos son válidos para conseguir el objetivo del sistema y genera comandos que permiten que el sistema actúe correctamente.
- **Adaptación:** Capacidad del controlador para reaccionar correctamente ante cambios en el ambiente y mantener la ejecución del comportamiento deseado.

Es importante considerar la influencia de la HMI en la arquitectura de control. El factor cognitivo y biológico también puede ser considerado en el diseño jerárquico, como en el caso del control de vuelo para aviones [57]. Los datos y estados del sistema de control estimulan los sentidos del ser humano. Estos datos proveen la información necesaria para la toma de decisiones que afectan o regulan las tareas del sistema de control.

Finalmente, se requiere modelar de manera discreta aquellas decisiones de alto nivel de acuerdo a las funcionalidades de un robot y el ambiente en el que se desenvuelve. Esto se consigue mediante autómatas o máquinas de estados finitos (*Finite State Machine* - FSM) [58]. Las FSM están formadas por:

- **Estados:** Es el conjunto de tareas o comportamientos que puede tener el sistema.
- **Transiciones:** Son enlaces que determinan cuales son los cambios de estado válidos y las condiciones que rigen este cambio.

2.3.4. Arquitecturas de Desarrollo de Software

Una de las primeras decisiones en el diseño de *software* consiste en especificar la arquitectura general del sistema. De acuerdo al estándar IEEE 1471-2000 [59], se define a una arquitectura de *software* como: *"La organización fundamental de un sistema, mismo que está constituido por componentes, las relaciones entre los mismos y el ambiente, y los principios que guían su diseño y evolución"*. Un modelo arquitectónico es un modelo de transferencia de conocimientos que permite expresar la solución de *software* a todas las partes interesadas y en un lenguaje de alto nivel. Esta visión global del sistema facilita la selección de herramientas de desarrollo, anticipar riesgos y adaptarse a los cambios en los requerimientos.

Como se enunció en la sección 2.3.2, el desarrollo de *software* para robots está orientado a la descomposición de funcionalidades y reutilización de algoritmos. Por ese motivo los estilos arquitectónicos usados con mayor frecuencia son aquellos orientados a objetos, componentes y servicios, aunque también existe un creciente interés por el desarrollo dirigido por modelos [60].

2.3.4.1. Arquitectura Basada en Componentes

Se considera que un componente es una unidad de composición sujeta al uso de terceros. La funcionalidad del componente es expuesta mediante interfaces que tienen un contrato definido, dependencias explícitas y despliegue independiente [61].

- **Unidad de Composición:** Es la unidad mínima que encapsula la funcionalidad y que provee o utiliza interfaces de otros componentes. En los sistemas robóticos se lo puede considerar como la unidad de encapsulación de funcionalidad robótica [47].

- **Interfaces con Contratos:** Las interfaces son las funcionalidades que un componente expone a los demás de manera independiente a su implementación. Se conoce como contrato al acuerdo que existe entre los componentes para cumplir con su funcionalidad. Esto incluye características como la calidad de servicio, precondiciones, postcondiciones y tipos de datos.
- **Despliegue Independiente:** Los componentes desarrollados son desplegados o instalados por sí mismos, siempre y cuando cumplan con las dependencias en el ambiente de operación.
- **Dependencias explícitas:** Se debe declarar bajo qué condiciones del ambiente operativo podrán ser usados los componentes. Esto incluye sistemas operativos, plataformas, mecanismos de comunicación, etc.

2.3.4.2. Patrón de Arquitectura Dirigida por Eventos

Un patrón arquitectónico es un conjunto de lineamientos o recomendaciones que enuncian soluciones comunes a problemas recurrentes en el desarrollo de sistemas. Como se detalla en la sección 2.3.2.3, los sistemas robóticos requieren de un mecanismo para notificar del cambio de estado a otros componentes para mantener la consistencia de los comportamientos. La arquitectura dirigida por eventos es una de las estrategias que permite cumplir este requerimiento de distribución de la información. Este patrón puede ser incluido en la arquitectura basada en componentes como se redacta en [62], mediante dos modelos: el de publicación-suscripción descrito en la sección 2.3.2.2 y la comunicación punto a punto.

2.3.4.3. Systems Modeling Language (SysML)

SysML se define como un lenguaje de modelado de propósito general para cualquier sistema dentro de las áreas de ingeniería. Reúsa un subconjunto de características de la versión 2 de UML y propone extensiones de acuerdo a los requerimientos de la ingeniería de sistemas. En la figura 2.12 están representados aquellos diagramas que son iguales a UML mediante un bloque de línea fina, aquellos que han sido modificados mediante un bloque de línea gruesa y los creados por SysML con líneas punteadas.

En este caso no se consideran los diagramas de clase debido a que SysML provee estructuras para modelar un sistema independientemente del paradigma de programación a ser usado. Después los programadores tendrán la libertad de usar programación orientada a objetos, estructurada, basada en componentes, etc.

Los elementos clave y de composición para describir cualquier sistema en SysML son los bloques. De acuerdo con la definición del OMG en [63], un bloque puede representar tanto a unidades estructurales como a unidades de comportamiento. Éstos son usados para especificar y diseñar la composición física y lógica de un sistema, así como sus elementos humanos, de *hardware* y *software*. La forma en que cada uno de estos bloques interactúa entre sí es variada. Por ejemplo, los bloques

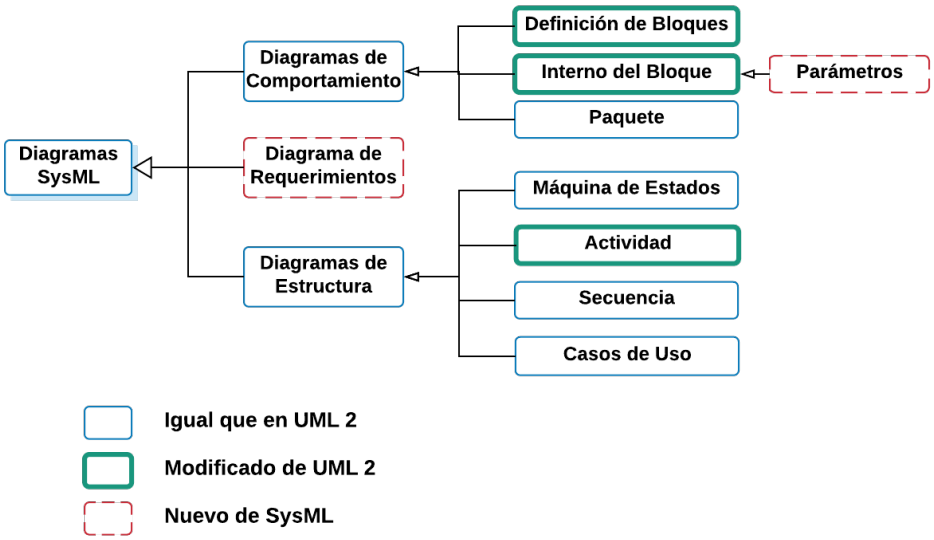


Figura 2.12: Tipos de diagramas en SysML [63].

pueden comunicarse mediante llamadas a funciones de *software*, transiciones de estado, flujos de entrada y salida de datos, etc. Esta descomposición en bloques es favorable para el desarrollo de sistemas complejos en los que no se requiere modelar únicamente la estructura del *software*.

Los cuatro pilares fundamentales para la descripción de un sistema con SysML son los diagramas estructurales, de comportamiento, de requerimientos y parámetros. En el presente proyecto se usaron los que se describen a continuación:

- **Estructural:** Se refiere a la vista estática del sistema en la cual se definen los bloques que la componen, sus atributos, operaciones y relaciones. Esto se consigue mediante los diagramas de definición de bloques (*Block Definition Diagram* - BDD), los diagramas internos de bloque (*Internal Block Diagram* - IBD) y los diagramas de paquetes. Los BDD son similares a los diagramas de clase en los cuales se especifican los atributos, las relaciones de asociación y herencia entre los componentes del sistema. Los IBD muestran las relaciones de comunicación entre los bloques de un sistema y sus interfaces.
- **Comportamiento:** Son aquellos diagramas que representan el comportamiento de todo el sistema, la interacción de un conjunto de componentes o el funcionamiento interno de un bloque. En este grupo están los diagramas de caso de uso, actividad, secuencia y de máquina de estados finitos.

2.4. Infraestructura para Sistemas de Control

Durante la implementación de un sistema, se realiza un proceso de transición desde el diseño conceptual hacia su instanciación sobre *hardware* y *software* concretos. Se conoce como infraestructura al conjunto de componentes de *hardware*

y *software*, sus relaciones, dependencias y características de calidad que aseguran el correcto funcionamiento de todo el sistema. En la figura 2.13 se presenta la infraestructura para sistemas embebidos la cual consta de tres capas. La capa de *hardware* que es la base del sistema y otras dos capas opcionales: la de *software* y la de aplicación.

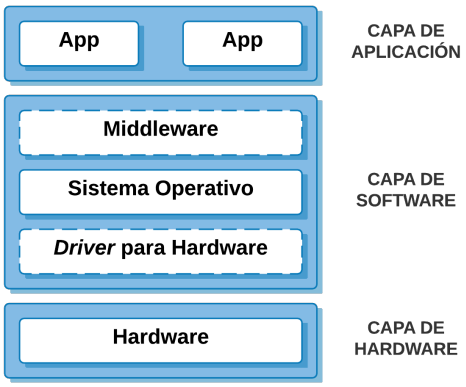


Figura 2.13: Arquitectura de la infraestructura de un sistema embebido [64].

2.4.1. Elementos de Hardware

Se entiende como *hardware* a todos aquellos recursos físicos y tangibles que forman parte de un sistema. En el caso de los sistemas embebidos, se refiere a las unidades de cómputo y a diferentes dispositivos de entrada o salida de datos. Según [65], los sistemas de computación embebidos pueden ser microcontroladores, matrices de puertas programables (FPGA) o circuitos integrados de aplicación específica (ASIC). Uno de los dispositivos de mayor escala que son usados en procesos de control son los computadores de placa reducida (Single-Board Computers - SBC). Éstas son pequeñas placas de circuito impreso que tienen todos los elementos de un modelo de la máquina de Von Neumann como son las unidades de procesamiento, control, memoria principal y sistema de entrada y salida de datos. A continuación, se describen características de la *Raspberry Pi* que es la SBC usada como parte del proyecto.

Raspberry Pi: La *Raspberry Pi* (figura 2.14), es una SBC de bajo costo y consumo energético reducido. Fue creada por la *Raspberry Pi Foundation* a partir del año 2012 con el objetivo de promover la enseñanza de las ciencias de la computación en la educación primaria y secundaria; sin embargo, también se ha estudiado su aplicación en proyectos de electrónica. El sistema operativo que usa es una distribución de GNU/Linux derivada de Debian y que se denomina Raspbian.

En el presente trabajo se usó la versión 3 Modelo B, la cual está equipada con un conjunto de chips Broadcom BCM2387 con procesadores de cuatro núcleos

ARM Cortex-A53 de la familia ARMv8 y con velocidad de reloj de 1.2 GHz. Otras ventajas a diferencia de sus predecesoras son la conectividad inalámbrica mediante el estándar IEEE 802.11 b/g/n *Wireless* LAN y *Bluetooth* 4.1, es decir, *Bluetooth* clásico y *Bluetooth Low Energy*. Posee 1GB de memoria RAM LPDDR2. Adicionalmente incluye un conector GPIO que posee 40 pines, de los cuales 27 son para entrada y salida de datos y otros para SPI, UART e I^2C.

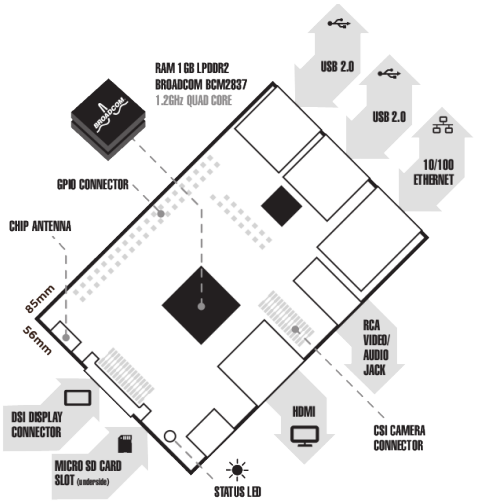


Figura 2.14: Componentes de la Raspberry Pi [64].

2.4.2. Elementos de Software

La capa de *software* debe estar como mínimo compuesta por el *kernel* o núcleo del sistema operativo. Opcionalmente puede existir una capa de *middleware* y otra de *drivers* para el control de los elementos de *hardware*.

2.4.2.1. Sistemas Operativos

El sistema operativo es aquel componente que funciona como una capa de abstracción que permite que el *middleware* y los ejecutables a nivel de aplicación puedan interactuar con los recursos de *hardware* y *software* del sistema embebido. Las funciones que implementa un sistema operativo y sus dependencias son variadas, pero como mínimo se requiere que gestione el uso de la memoria, de procesos y dispositivos de entrada y salida. La elección de un sistema operativo para un robot depende de la arquitectura de *hardware* del microcontrolador a ser usado y del soporte que exista para dicha plataforma. Un segundo criterio de elección es determinar si se requiere que el sistema operativo ofrezca garantías de tiempo real.

Un proceso tiene garantías de tiempo real si su infraestructura contribuye al determinismo temporal, es decir, el proceso es ejecutado dentro de un límite de tiempo conocido. Esto no significa que el tiempo de procesamiento del algo-

ritmo debe ser instantáneo, simplemente conocido. Basados en esta premisa, el determinismo temporal puede clasificarse en:

- **Estricto (*Hard*):** Son aquellos sistemas en los que se garantiza que todos los límites de tiempo se cumplen, caso contrario el sistema no podría funcionar y entraría en un estado de falla.
- **Firme (*Firm*):** Son sistemas en los cuales es posible tolerar cierto porcentaje de procesos que han excedido su tiempo de procesamiento; sin embargo, no entra en un estado de fallo y la respuesta generada no tiene valor.
- **Blando (*Soft*):** Son sistemas en los que se tolera el sobrepasar los límites de tiempo; sin embargo, los datos con retardo en el tiempo son aceptados con la consecuente degradación del servicio.

Sistemas Operativos de Tiempo Real: Los sistemas operativos de tiempo real (*Real-Time Operating System* - RTOS) son diseñados para ofrecer mecanismos y configuraciones que permitan garantizar el determinismo temporal. Sin embargo, el desarrollador debe afinar algunas de estas características para que el determinismo pueda cumplirse. Los sistemas operativos de propósito general como *GNU/Linux*, *Windows* y *macOS* no forman parte de este grupo debido a que tienen funcionalidades que causan latencias como es el caso de la paginación y la planificación o *scheduling* de los procesos. La paginación genera interrupciones de latencia no determinista debido al uso de la memoria virtual. Por otro lado, está la planificación en la cual se debe garantizar que un proceso de mayor prioridad no sea reemplazado por otro de menor prioridad.

RTOS en el mercado: Según el estudio de mercado realizado por la empresa *AspenCore* y reportado en [66], las razones que influyen en la decisión de un RTOS son: aspectos de infraestructura de hardware como su arquitectura y procesadores, disponibilidad del código fuente, soporte técnico, costo de las licencias, rendimiento en tiempo real y en menor medida aspectos de seguridad, uso de recursos computacionales, herramientas de *software* y soporte de *drivers*.

En la tabla 2.2 se presentan algunos RTOS que se utilizan en proyectos e investigaciones como se muestra en [66, 67]. En la elaboración del listado se usaron criterios de exclusión como: la disponibilidad de código abierto y gratuidad con lo que se descartaron sistemas operativos como *ThreadX*, *QNX*, *VxWorks*, *Windows Embedded*, *MQX* y *Micrium*, mismos que tenía un porcentaje de uso superior al 2 % según [66]. Otros sistemas como *Contiki*, *TinyOS*, *OpenWSN* y *Nut/OS* que no ofrecen un *kernel* con mecanismos de apropiación, y por tanto, no tiene garantías de tiempo real. Tampoco se consideran aquellos sistemas operativos que no tienen soporte para procesadores ARM, requeridos para la *Raspberry Pi*. Bajo este criterio no se considera a *nanoRK*. Por otro lado, están aquellos sistemas operativos que solo tienen soporte para ARM Cortex-M como *CoOS*, *RIOT*, *ChibiOS* y µ*Clinux*.

RTOS	Arquitectura	Soporte en Raspberry Pi 3 Modelo B
Linux RT Preempt	Monolítico	Se requiere compilar la distribución Raspbian con el parche RT Preempt, con soporte en el repositorio oficial.
Xenomai	Co-Kernel para GNU/Linux	Configurar el parche de Xenomai sobre GNU/Linux.
FreeRTOS	Microkernel RTOS	Sin soporte oficial, existen instrucciones de terceros para portar el código a la plataforma.
nuttX	Monolítico o Microkernel	El soporte fue descontinuado y estaba dirigido principalmente a la Raspberry Pi Zero.
eCos	RTOS monolítico	Soporte planificado pero todavía no ha sido liberado.

Tabla 2.2: Listado de sistemas operativos de tiempo real. Basado en: [67]

Basándose en la información recopilada en la tabla 2.2, se puede determinar que una *Raspberry Pi 3* modelo B solo tiene soporte oficial con el sistema operativo *Raspbian* y por tanto *Linux RT Preempt* y *Xenomai* son los mejores candidatos para ser usados como sistema operativo de tiempo real. Esto también es expuesto en [68], donde se determina que muchos de los otros RTOS ya no son compatibles en este modelo.

2.4.2.2. Middleware Robótico

Un *middleware* es una capa de abstracción de *software* que facilita la interacción entre los componentes de una aplicación, el sistema operativo y los *drivers* para el *hardware* mediante diferentes protocolos de comunicación entre procesos como *Common Object Request Broker Architecture* (CORBA), *Remote Procedure Call* (RPC) o *Data Distribution Service* (DDS). Según Noergaard en [64], los *middlewares* para sistemas embebidos ofrecen alguna de las siguientes características:

- **Adaptabilidad:** Del *software* de aplicación para acoplarse a los cambios de disponibilidad de recursos computacionales.
- **Conectividad y comunicación interna:** Permite la comunicación entre aplicaciones mediante interfaces estandarizadas.
- **Flexibilidad:** Permite configurar la funcionalidad del *middleware* o de las aplicaciones para que puedan satisfacer los requerimientos del sistema.
- **Escalabilidad:** Se puede añadir nuevo *software* de aplicación al sistema si éste respeta las convenciones del *middleware* y la definición de las interfaces de comunicación entre procesos.

- **Portabilidad:** Puede estar diseñado para ser ejecutado sobre diferentes sistemas operativos y arquitecturas de procesador, lo cual es transparente para el *software* de aplicación.
- **Seguridad:** Asegurarse que el *software* de aplicación tenga acceso a los recursos computacionales autorizados.

En el caso de la robótica se considera como *middleware* a todo aquel *software* que facilita la integración de procesos dentro de un sistema robótico, siendo lo más importante la comunicación entre procesos. Muchos *middleware* son incluidos como parte de un *framework* [69] o de ambiente de desarrollo para la robótica (*Robotic Development Environments – RDE*) [70], es decir, que proveen un conjunto de herramientas que son usadas para producir *software* de la capa de aplicación. Algunas características propias del *middleware* para robots son:

- **Ofrecer funcionalidades comunes a los sistemas robóticos:** Como navegación, manipulación, cinemática, reconocimiento de imágenes, inteligencia artificial y otros tipos de algoritmos que se pueden aplicar a diferentes tipos de robots.
- **Soporte a componentes embebidos y de bajos recursos:** Debido a que se interactúa con componentes con limitaciones en memoria, procesamiento y capacidades de comunicación.

Existen varios *middlewares* debido a que diferentes instituciones tienen diferentes requerimientos para sus sistemas robóticos. Las características de algunos de estos sistemas han sido recopilados en evaluaciones sistemáticas que analizan sus características considerando la contribución al proceso de desarrollo [70], tecnologías de comunicación entre procesos [71], arquitectura de componentes [72], y características no funcionales y de plataforma [69, 73]. Como resultado se encontraron 27 implementaciones de *middlewares* para robots, a las cuales se aplicaron los siguientes criterios de exclusión para encontrar herramientas apropiadas para el desarrollo del sistema:

- El *software* de aplicación generado debe ejecutarse sobre el sistema operativo *GNU/Linux*. La ejecución sobre el sistema operativo *Windows* es opcional pero considerada en caso de que se requieran realizar pruebas en la máquina de alguno de los desarrolladores.
- Su desarrollo puede realizarse desde una computadora con el sistema operativo *GNU/Linux*. El desarrollo usando el sistema operativo *Windows* es opcional.
- El desarrollo de la aplicación debe ser realizado en C++.
- El *middleware* debe tener un proyecto de desarrollo activo (haber tenido mantenimiento en los últimos 5 años) y una página web desde la cual descargar el código fuente, archivos ejecutables y documentación.
- El uso del *middleware* debe ser gratuito.
- El *middleware* debe ser de código abierto.
- El *middleware* debe estar orientado al control y tener garantías de tiempo real.

Se obtuvieron cuatro plataformas con capacidad de procesamiento en tiempo real como son *OpenRTM-aist*, *ROS2*, *SmartMDSD* y *OROCOS* (el cual puede considerarse parte del *framework ROCK (Robot Construction Kit)*. En la tabla 2.3 se resumen las características de los *middlewares* seleccionados. Como conclusión de la tabla anterior se determinó que todos los *middlewares* cumplen los requerimientos para un sistema de control. Sin embargo, el factor que determinó el uso de ROS2 fue la documentación, foros de discusión y ejemplos disponibles en línea. Algunas características del sistema seleccionado son detallados en la sección 2.4.3.

Característica	OpenRTM-aist	ROS2	SmartMDSD	ROCK
Modelo del Sistema	CBSD, MDE	CBSD	MDE	CBSD
Arquitectura de procesador	i386, x86_64, ppc, arm	x86_64, arm	ARM, x86_64	?
Sistema Operativo	Linux, FreeBSD, Windows, macOS	Linux, Windows, macOS, freeRTOS, nuttX	QNX, RTAI-Linux	Linux, Xenomai
Lenguaje de Programación	C++, Java, Python	C++, Python	C++	C++, Ruby
Comunicación entre procesos	CORBA	DDS/RTPS	ACE, CORBA, DDS	CORBA (ACE/TAO)
Simulador	OpenHRP-3, Gazebo (No oficial)	Gazebo, TF2	Gazebo	VizKit3D
Seguridad	No	SROS2 (Paquete)	Si	No
Conexión en tiempo de ejecución	Sí	Sí	Si	Sí
Documentación	Disponible (Japonés)	Disponible	Disponible	No Actualizada

Tabla 2.3: Características de los *middlewares* robóticos

2.4.2.3. Bases de Datos de Series de Tiempo

Una base de datos de series temporales (*Time-Series Database* - TSDB) es aquella que está optimizada para datos con marcas de tiempo. Se entiende como un dato temporal a todas aquellas mediciones o eventos que pueden ser rastreados, monitorizados, muestreados o agregados a lo largo del tiempo [74]. Estos datos generalmente se registran como una nueva entrada que llega en un orden cronológico. Cada nuevo registro permite analizar el fenómeno en el pasado, controlar los cambios actuales y predecir cómo pueden ocurrir los cambios en el futuro.

Se usan en sistemas con sensores que emiten flujos de métricas y eventos. Esto significa que estas bases de datos deben ser escalables y usables para soportar nuevas cargas de trabajo, más puntos de datos, monitorización y controles.

- **Escala:** Debido a que los datos de series temporales se acumulan rápidamente, las TSDB incluyen optimizaciones de rendimiento. Estas mejoras incluyen tasas de consumo más altas, consultas rápidas y mejor compresión de datos.
- **Usabilidad:** Las TSDB incluyen funciones y operaciones comunes al análisis de datos de series temporales como: políticas de retención de datos, consultas continuas y agregaciones de datos en el tiempo.

Las bases de datos de series de tiempo son un segmento en constante crecimiento en la industria de las bases de datos. Según el sitio web independiente *DB-Engines*, se clasifica a las bases de datos de acuerdo a su popularidad en los motores de búsqueda, las menciones en las redes sociales, las publicaciones de trabajo y el volumen de discusiones [75]. Existen un total de 27 bases de datos de series de tiempo, a las cuales se les aplicó los siguientes criterios de exclusión para encontrar la herramienta adecuada para el desarrollo:

- La base de datos debe ser de código abierto.
- El uso de la base de datos debe ser gratuito.
- Su modelo primario de base de datos debe ser asociado a las series de tiempo.
- Debe ser compatible con los sistemas operativos de *Linux* y *Windows*.
- Debe soportar los lenguajes de programación *Java* o *Python*.
- La base de datos debe tener una página web desde la cual se pueda descargar el código fuente, archivos ejecutables y documentación.
- La categoría de la base de datos debe ser *Real-time Analytics*. Entendiéndose *Real-Time Analytics* como la propiedad para realizar un análisis de datos tan pronto estos se encuentren disponibles.

Como resultado, se obtuvieron cuatro bases de datos de series de tiempo, que cumplen con los criterios planteados anteriormente. Éstas son *InfluxDB*, *Prometheus*, *TimescaleDB* y *KairosDB*. En la tabla 2.4 se recopilan las características principales de las bases de datos seleccionadas. *TimescaleDB* fue la base de datos escogida para el desarrollo de la aplicación debido a la disponibilidad de documentación, su funcionamiento simple y su integración con otras bases de datos relacionales.

Característica	InfluxDB	Prometheus	TimescaleDB	KairosDB
Categoría	*Analítica en tiempo real*	*Sistema de Monitoreo*	*Analítica en tiempo real*	*Analítica en tiempo real*
Sistema Operativo	Linux, OS X	Linux, Windows	Linux, OS X, Windows	Linux, OS X, Windows
Alta Disponibilidad	Doble escritura de servidores	Doble escritura de servidores	Sí	Clustering
Complejidad operacional	Baja (media con alta disponibilidad)	Baja	Baja (media con alta disponibilidad)	Media
Tipo de datos soportado	int64, float64, bool, y string	float64	Todos los tipos de datos de PostreSQL	string, float32, float64
Lenguaje de consulta	InfluxQL (como SQL)	PromQL	SQL	Solamente búsqueda
Modelo de datos	métricas, campos, etiquetas	etiquetas, métricas	tablas relacionales	métricas, etiquetas
Documentación	Disponible	Disponible	Disponible	Disponible

Tabla 2.4: Características de las bases de datos de series de tiempo

2.4.3. *Framework* para Desarrollo de Sistemas Robóticos Robot Operating System 2 (ROS2)

El *Robotic Operating System* (ROS) es un proyecto que inició en el año 2007 con el objetivo de proveer herramientas que solucionan problemas comunes del desarrollo de robots. En el año 2010 se publicó la primera versión estable del *middleware*, el cual fue diseñado con cinco objetivos en mente [76]:

- **Red de pares:** Permite la comunicación entre procesos incluso si los mismos están en diferentes máquinas huésped.
- **Multilenguaje:** Facilitar el soporte de diferentes lenguajes de programación.
- **Basado en herramientas:** Tiene una estructura de micronúcleo, es decir, provee un conjunto de funcionalidades básicas como parte de diferentes paquetes.

- **Ligero:** ROS debe ser usado como una capa de abstracción para integrar una librería de *software* que es independiente al *framework*.
- **Gratuito y de código abierto**

La funcionalidad de ROS ha sido extendida con paquetes desarrollados por terceros y ha sido implementada en proyectos de investigación y de la industria. Sin embargo, la principal desventaja de su diseño es que no garantiza el determinismo temporal que es requerido para proyectos críticos. El problema está en el protocolo usado para la implementación de la comunicación entre procesos. Por este motivo, a partir del año 2015 inició el desarrollo de la segunda versión de ROS, publicándose la primera versión estable a finales del año 2017.

2.4.3.1. Componentes

ROS2 sigue un patrón de composición distribuida, esto quiere decir que no se requiere un componente central o maestro que esté encargado del descubrimiento y conexión del resto de nodos del sistema. En la tabla 2.5 se presentan los conceptos básicos para el desarrollo de un componente de ROS2 en su tercera versión (denominada *Crystal Clemmys*) y los lenguajes de programación que soportan su implementación. Estos conceptos están clasificados de acuerdo al principio de separación de asuntos presentado en la sección 2.3.2.

Problema	**Elemento**	**C++**	**Python**
Comunicación	Mensajes (*Messages*)	✓	✓
	Servicios (*Services*)	✓	✓
	Acciones (*Actions*)	✓	✗
	Comunicación intra-proceso	✓	✗
Cómputo	Nodo (*Node*)	✓	✓
Composición	Ejecutor (*Executor*)	✓	✓
Configuración	Argumentos (*Node Arguments*)	✓	✓
	Parámetros (*Parameters*)	✓	✓
Coordinación	Ciclo de Vida (*Node Lifecycle*)	✓	✗
	Despliegue de componentes	✗	✓

Tabla 2.5: Elementos de los componentes de ROS2.

2.4.3.2. Comunicación

La comunicación entre procesos se realiza mediante un modelo de publicación y suscripción centrada en los datos (*Data-Centric Publish-Suscribe* - DCPS). Este modelo se define mediante tres especificaciones de la *Object Management Group* (OMG) para la transmisión efectiva y robusta de datos en tiempo real:

- **DDS (*Data Distribution Service*):** Según [77], es un estándar que asegura la interoperabilidad entre aplicaciones. El objetivo es ofrecer una definición estandarizada de las interfaces, su comportamiento y calidad del servicio.
- **RTPS (*Real-Time Publish-Subscribe Protocol*):** Es un estándar que define la estructura de los mensajes y como estos son transmitidos o recibidos mediante protocolos de la capa de transporte como TCP/UDP/IP [78]. Esto garantiza la interoperabilidad entre proveedores de tecnología DDS.
- **IDL (*Interface Definition Language*):** Es un mecanismo para representar objetos de manera independiente al lenguaje de programación y plataforma. Es usado por DDS para que la definición de los datos a ser intercambiados sea explícita [79].

ROS2 incluye una capa de abstracción adicional llamada ROSIDL que evita que el desarrollador tenga que interactuar directamente con los conceptos de DDS. Esto permite que las aplicaciones mantengan su funcionalidad en caso de que se cambie la tecnología de transporte o su proveedor. La desventaja es que existe una latencia adicional que resulta de la transformación de un objeto en ROS a su representación en DDS, como se reporta en [80].

2.4.3.3. Funcionamiento del Protocolo DDS

En la figura 2.15 se presenta el modelo de comunicación de ROS2 mediante una versión modificada del modelo *Open System Interconnection* (OSI) orientado a *middlewares*. En este caso, se desea enviar un dato desde una aplicación que los produce hacia otra que los consume. En ROS un dato es representado como un objeto o estructura de datos de un lenguaje de programación. Posteriormente, es transformado en una instancia de IDL para que sea manejado por la capa DDS. En esta capa se serializa el mensaje y se configuran las características de calidad de servicio. Al momento de formar una trama RTPS, cada uno de los mensajes serializados constituye un submensaje. Como resultado, cada trama RTPS puede contener varios submensajes que se dirigen a un mismo consumidor. Dependiendo de los requerimientos de calidad de servicio se usa una trama UDP o TCP para ser enviado a la máquina o proceso que requiere consumir los datos. El consumidor realiza la secuencia inversa de tareas para obtener la estructura de datos enviada.

Desde el punto de vista del desarrollador de aplicaciones todos los mecanismos de comunicación en ROS2 se definen mediante cinco conceptos que se presentan en la figura 2.16. Estos son:

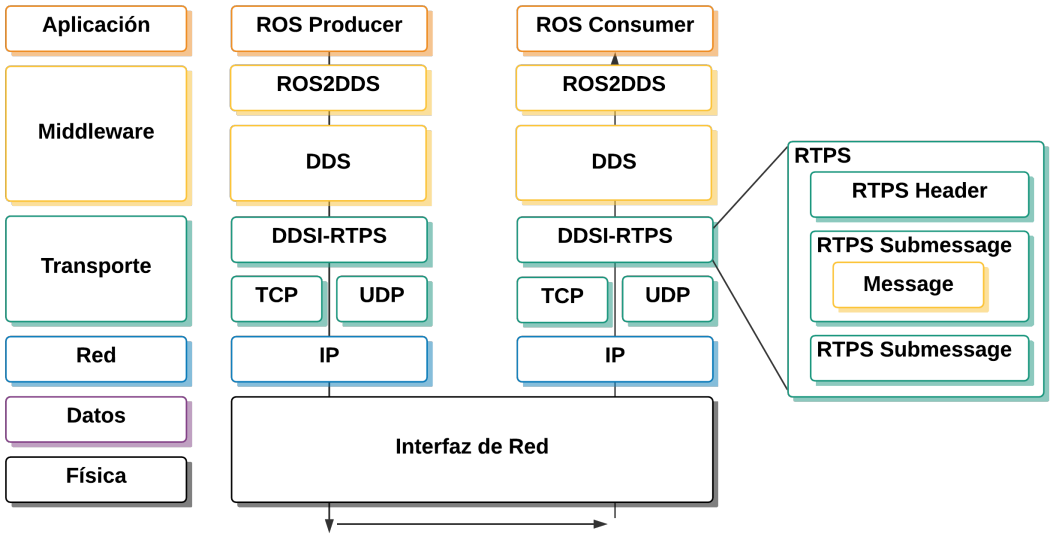


Figura 2.15: Protocolo de comunicación en ROS2.

- **Productor:** Elemento que genera y envía un objeto.
- **Consumidor:** Elemento que requiere un objeto enviado por algún productor.
- **Mensaje:** Estructura de datos definida mediante ROSIDL. Es decir, un archivo de texto en el cual se define el tipo de dato y nombre de la variable.
- **Tema (*topic*):** Identificador que asocia a un mensaje, consumidor y productor. Funciona como un canal de transmisión de mensajes. Los productores envían mensajes a un tema y los consumidores quieren obtener mensajes enviados a dicho tema.
- **Políticas de Calidad de Servicio:** Características no funcionales de la comunicación que son negociadas por los consumidores y productores. Se recomienda que ambos lados del canal tengan la misma definición de calidad de servicio, caso contrario no se puede garantizar su cumplimiento.

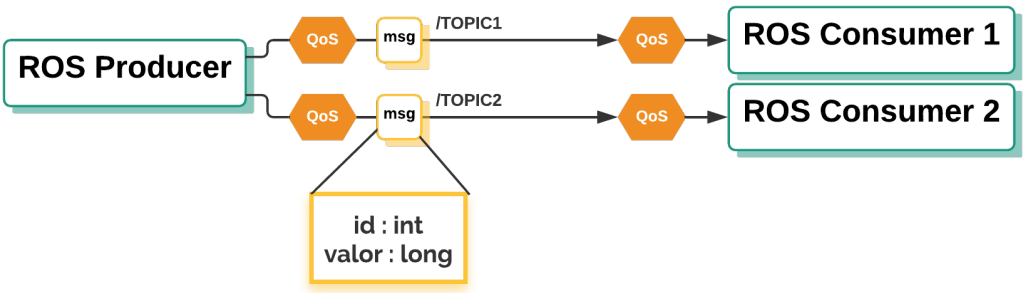


Figura 2.16: Modelo *Data-Centric Publish-Subscribe* desde el punto de vista del desarrollador.

Perfiles de Calidad de Servicio Un perfil de calidad de servicio es un conjunto de políticas que son usadas en la comunicación. DDS define 22 políticas relacionadas al consumo de recursos, tolerancia a fallos, distribución de mensajes y fiabilidad de la comunicación. Sin embargo, la API de ROS2 solo permite modificar cuatro que se resumen en la tabla 2.6.

Política	Valores	Descripción
Profundidad (*DEPTH*)	Valor numérico	Número de mensajes que son almacenados en la pila. En el caso de los productores, es el número máximo de mensajes publicados que son almacenados, si el parámetro de durabilidad lo autoriza. En el caso de lo consumidores es el número de mensajes que esperan en la pila hasta que son procesados o retirados.
Historia	KEEP_LAST	Almacena un número máximo de mensajes como lo define el parámetro DEPTH.
	KEEP_ALL	Ignora el parámetro DEPTH y almacena el número de mensajes que permita la configuración del proveedor de servicio DDS.
Fiabilidad	BEST_EFFORT	Publica el mensaje pero no controla si éste se ha perdido.
	RELIABLE	El productor debe asegurarse de que el mensaje ha llegado, reintentando el envío si es necesario.
Durabilidad	TRANSIENT_LOCAL	Los productores almacenan los mensajes definidos por el parámetro DEPTH y lo retransmiten si un nuevo suscriptor se registra.
	VOLATILE	Los mensajes no son almacenados por los productores.

Tabla 2.6: Políticas de calidad de servicio.

Mensajes: Es el mecanismo de comunicación para el envío asíncrono de información con un patrón de interacción publicador-suscriptor. La llegada de un mensaje provoca que el suscriptor ejecute una función.

Servicios: Es un mecanismo de comunicación que se comporta de manera similar a una llamada a procedimiento remoto. Un servicio es una definición de dos estructuras de datos o mensajes: un mensaje para la solicitud y otro para la respuesta. Por tanto, corresponde a la interacción de tipo cliente-servidor.

2.4.3.4. Nodos

El nodo es la unidad central del *software* de aplicación producido en ROS2. Todas las aplicaciones de este *framework* están formadas por la composición y coordinación de varias de estas estructuras. Como se presenta en la figura 2.17, un nodo consta de instancias de los elementos de comunicación como publicadores, suscriptores, clientes y servidores. Incluye funciones para configurar su funcionalidad y manejar su estado interno.

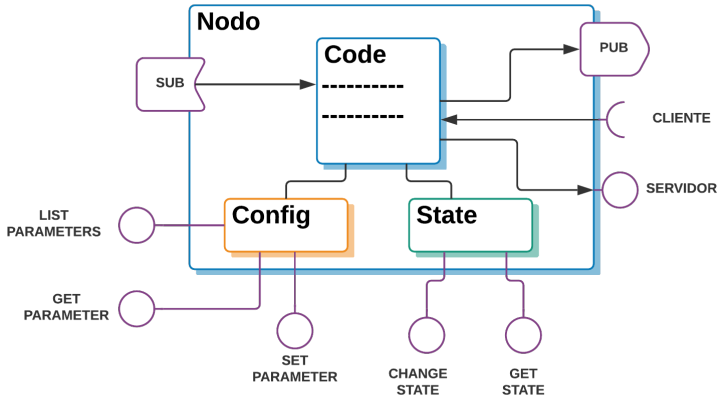


Figura 2.17: Estructura de un nodo de ROS2.

2.4.3.5. Ejecutores

Un ejecutor corresponde a un proceso de ejecución en el sistema operativo huésped. Por esa razón los nodos que deben ejecutarse en un mismo proceso son instanciados y agregados al ejecutor. Otra función de este componente es el de gestionar la ejecución de las funciones de *callback* de los suscriptores, clientes y temporizadores (función que se ejecuta con una frecuencia de tiempo configurada). Se debe considerar que cada vez que se genera un evento en un nodo, éste entra en una cola de ejecución compartida con el resto de nodos del proceso. El ejecutor puede tener uno o múltiples hilos para resolver los eventos presentes en dicha cola.

2.4.3.6. Argumentos y Parámetros

Los argumentos son un mecanismo para el paso de parámetros desde la línea de comandos, con el objetivo de reconfigurar el funcionamiento de los nodos previo a su ejecución. Por otro lado, los parámetros son un conjunto de pares clave y valor que le pertenecen a cada nodo. Los parámetros pueden ser accedidos o modificados desde otras partes del sistema mediante los servicios que exponen y en tiempo de ejecución. Estos servicios permiten listar, leer o escribir parámetros de manera atómica. Los parámetros son pasados a un nodo mediante un archivo de texto en formato YAML (*YAML Ain't Markup Language*) como los que se muestran en el anexo D. Este archivo debe tener una estructura definida: en el

primer nivel se coloca el nombre del nodo, en el segundo se escribe la palabra clave *ros___parameters* y a partir del tercer nivel los pares de clave y valor.

2.4.3.7. Ciclo de Vida

Existe un tipo especial de nodo que incluye una máquina de estados finita a la cual se la conoce como el ciclo de vida del nodo (*node lifecycle*). La idea de este modelo es facilitar la administración de los nodos al conocer si están preparados para ejecutar su funcionalidad. Como se observa en el diagrama de estados de la figura 2.18, se han definido cuatro estados y cinco transiciones con su propia semántica. Las transiciones son funciones programables por el desarrollador para preparar al nodo antes de cambiar de estado o, en su defecto, decidir si alguna condición no se ha cumplido y cancelar la transición.

- **UNCONFIGURED:** Es el estado inicial de cada nodo al momento de ser instanciado, generalmente en este punto ya se ha ejecutado el constructor y se han creado los elementos de comunicación y configuración.
- **INACTIVE:** El nodo ha reservado toda la memoria requerida para poder ejecutar su funcionalidad. Esto incluye la configuración de los elementos de hardware. Se debe agregar que en este estado no se ejecuta ningún evento del nodo.
- **ACTIVE:** El nodo puede ejecutar todos los procesos y eventos de la pila de eventos.
- **SHUTDOWN:** Estado que libera las referencias del nodo en la memoria.

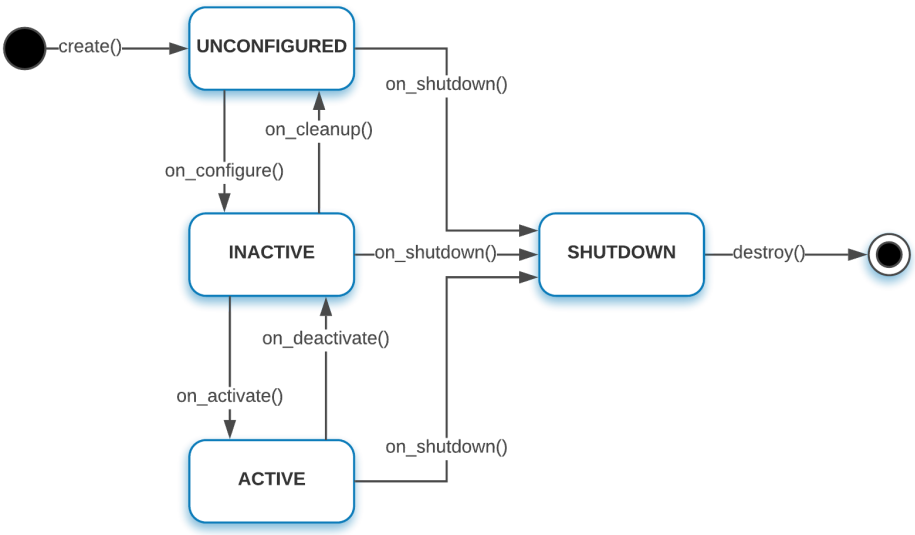


Figura 2.18: Ciclo de vida de un nodo en ROS2.

Al igual que los parámetros, el ciclo de vida expone servicios a otros nodos. Son de importancia dos de ellos *get_state* para obtener el estado actual y *change_state* para cambiarlo. Una de las ventajas es preparar al nodo para su funcionamiento en tiempo real durante los estados *UNCONFIGURED* e *INACTIVE*.

2.4.3.8. Despliegue de Nodos

Existen dos mecanismos para ejecutar un proceso que contenga nodos. El más sencillo es usar una interfaz de línea de comandos mediante el archivo binario *ros2 run*; sin embargo, esta opción no es escalable cuando se requieren ejecutar varios procesos de manera automatizada. Por ese motivo se ofrece una segunda opción conocida como *ros2launch*, la cual ofrece una API para el lenguaje de programación *python* en el cual los nodos puedan ser ejecutados y configurados.

2.5. Trabajos Relacionados

En esta sección se presentan trabajos relacionados a la elaboración de un sistema de control para un exoesqueleto desde el punto de vista de infraestructura de *software* y *hardware*, así como las interfaces existentes para que el fisioterapeuta o los usuarios puedan enviar comandos.

Infraestructura para el sistema de control: No existen estudios sistemáticos que integren las herramientas de *software* y *hardware* que son utilizadas para el control de exoesqueletos. Por tanto, se documentará la infraestructura empleada por algunos exoesqueletos de acuerdo a la infraestructura para sistemas embebidos propuesta en la sección 2.4.

En el caso de la capa de *hardware* se ha constatado el uso de SBC ya sean éstas de uso comercial o fabricadas por los mismos investigadores. Dentro de las placas producidas por los investigadores están aquellas que se elaboraron para el exoesqueleto *HYPER* [7] y *Hybrid Neuroprosthesis (HNP)* [8]. En estas máquinas se incluyen módulos *Bluetooth* para su comunicación con interfaces gráficas programadas con dispositivos móviles y un transceptor para comunicarse con el resto de componentes. El sistema de instrumentación en *HYPER* utiliza el protocolo CAN y en *HNP* se elaboró una placa de acondicionamiento de señales que es usada para adquirir e integrar las medidas de los diferentes sensores.

En el caso de exoesqueletos que usan SBC comerciales están *H2* y *BioMot*. H2 es una órtesis activa con 6 grados de libertad que está controlada por una *Raspberry Pi 2*, actualmente está siendo usada para estudios de intención de movimiento en tiempo real mediante señales electromiográficas [9]. Por otro lado, el exoesqueleto *BioMot* utiliza una *BeagleBoneBlack* como *hardware* de control de seis actuadores, incluyendo señales de electromiografía, electroencefalografía e inercia [10].

En lo referente a los sistemas operativos de tiempo real, los exoesqueletos tienden a usar *software* comercial como el módulo de *LabView* para tiempo real conocido como *xPC Target*. Este módulo permite ejecutar las simulaciones realizadas en *Simulink*, en computadoras personales con capacidad de tiempo real [11]. También está el *RTOS QNX* en el caso de una órtesis para contribuir a la mejora de la

agilidad de la fase de balanceo de una pierna [12].

Sobre el uso de *middleware* robótico en exoesqueletos para extremidades inferiores se han encontrado dos trabajos. En el primero se usa *OROCOS* para las tareas de control mecatrónico de un dispositivo para que los astronautas puedan ejercitarse en el espacio [81]. En exoesqueletos para rehabilitación la primera versión de ROS ha sido usada en el desarrollo del exoesqueleto *BioMot* [10], en el cual se considera importante el uso de sistemas operativos de tiempo real para disminuir la latencia en la ejecución de algoritmos.

Interfaces de Control: Uno de los objetivos de la rehabilitación es que los fisioterapeutas puedan controlar de manera intuitiva y configurable la ejecución de rutinas de un exoesqueleto. Lo primero que se debe determinar es el medio de comunicación entre el exoesqueleto y el profesional médico o el paciente. Esta comunicación puede realizarse mediante un dispositivo incorporado en el *hardware* del exoesqueleto o con un aparato externo como computadoras personales o celulares. Como se enuncia en [82], los exoesqueletos *Rex* y *Ekso-GT* poseen controles de mando incorporados. En el caso de *Rex* el control consta de una palanca de mando en el soporte para el brazo derecho del usuario (figura 2.19a). Este mando incluye una pantalla *Liquid Cristal Display* (LCD) con información sobre el estado de carga de la batería y un menú de selección del tipo de movimiento que desea ejecutar. Adicionalmente tiene tres botones, dos de los cuales son usados para aceptar o cancelar el movimiento. En el caso de *Ekso-GT* existe un control de mando conectado directamente a la parte posterior del exoesqueleto (figura 2.19b). Este dispositivo de mando le sirve al terapeuta para elegir y configurar las variables de los ejercicios de rehabilitación y recibir una retroalimentación auditiva o visual sobre el progreso del ejercicio, como por ejemplo, determinar cuál fue el evento que inició el movimiento.

(a) Mando del Exoesqueleto Rex. Rex Bionics, REX: Unique functions & benefits of a robotic walking device, 2016. Visitado: Feb. 27, 2019. [Streaming video]. Disponible: https://youtu.be/t2Q4UrxmwYE

(b) Mando del Exoesqueleto Ekso-GT. Ekso Bionics, SmartAssist Demonstration DE, 2017. Visitado: Feb. 27, 2019. [Streaming video]. Disponible: https://youtu.be/3W8Rsjrz-Ro

Figura 2.19: Dispositivos de mando conectados al exoesqueleto.

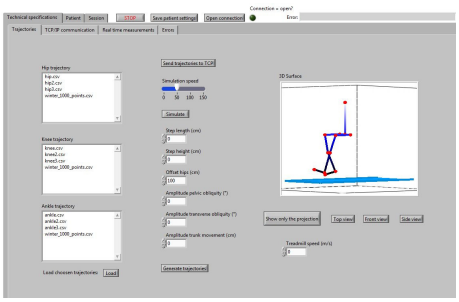
Otra forma de interacción es aquella en la que se usa una conexión cableada

desde el exoesqueleto hacia una computadora personal encargada de adquirir datos y enviar comandos. Este tipo de conexión es útil en el caso de efectores finales o exoesqueletos estáticos. En [15] se presenta la interfaz gráfica de usuario para el sistema *C.S.One* (figura 2.20a). Este es un exoesqueleto fijo con cuatro grados de libertad usado en la rehabilitación de las extremidades inferiores de un infante. En este caso la comunicación con el microcontrolador de la estructura se realiza mediante el bus serial universal (USB). En esta interfaz gráfica las señales de control del exoesqueleto son de inicio o cancelación de la caminata. Adicionalmente, posee secciones en su pantalla para la configuración de la terapia de rehabilitación. En esta pantalla se seleccionan las articulaciones que deben desplazarse, la velocidad del paso en revoluciones por minuto y el número de pasos a ser efectuados. En el centro de la pantalla presenta una sola gráfica para dibujar la posición angular de los motores respecto al tiempo.

Otros sistemas interactúan con el exoesqueleto mediante el protocolo TCP/IP. Este tipo de interacción es útil en interfaces gráficas de adquisición de datos de sensores debido a la alta frecuencia con la cual se generan. Este caso se presenta en un sistema desarrollado en *LabView* para adquirir información de la posición angular de cuatro motores y las fuerzas de interacción [83]. Otra interfaz gráfica de este tipo se presenta en [84] para la adquisición de señales de electromiografía. Un ejemplo final de interacción mediante TCP/IP es la interfaz gráfica para el exoesqueleto ALTACRO [16] creada en *LabView* (figura 2.20b), en la cual se incluyen pantallas para la administración de usuarios y la personalización de terapias de rehabilitación mediante la creación y carga de archivos en formato CSV.

Finalmente está la comunicación entre los dispositivos móviles y el controlador, usando protocolos como *Bluetooth*. Este es el caso de una aplicación para *Android* (figura 2.20c) que se comunica con el exoesqueleto *H2* [10]. Esta pantalla incluye comandos para activar o desactivar la transmisión de datos mediante *Bluetooth*, *WiFi* o en el Bus CAN. Incluye botones para iniciar o cancelar la caminata y seleccionar hasta 10 niveles de velocidad.

(a) GUI de escritorio para C.S.One [15].



(b) GUI en LabView para ALTACRO [16].

(c) GUI en Android para H2 [10].

Figura 2.20: Dispositivos de mando en *hardware* externos.

2.6. Conclusiones

El desarrollo de sistemas de *software* para robots es un proceso complejo debido a la heterogeneidad de *software*, *hardware* y la naturaleza interdisciplinaria de su investigación y aplicación. En el caso particular del desarrollo de *software* para exoesqueletos se requiere de la integración de conocimientos de áreas como la electrónica, mecánica, ciencias de la computación y medicina. Esta diversidad de áreas de conocimiento trae diferentes interesados, cada uno, con su conjunto particular de requerimientos. El desarrollo debe estar orientado al bienestar del paciente. Por otro lado, se debe facilitar la configuración de una sesión de rehabilitación para el fisioterapeuta. Como se documentó en la sección anterior (2.5), las interfaces gráficas de usuario se caracterizan por funcionalidades de control de movimientos y configuración de parámetros de caminata, con cierta tendencia a incorporar aspectos de sesiones de rehabilitación como definición de citas, registro de pacientes, etc.

La principal dificultad para la definición de una infraestructura es la falta de propuestas para la infraestructura de *software* usada en el desarrollo de otros exoesqueletos. Por tanto, se creó una pila de tecnologías de acuerdo al análisis de infraestructura de otras soluciones de *software* para sistemas embebidos. En la figura 2.21, se muestran los resultados sobre las tecnologías de implementación elegidas basado en un estudio previo de herramientas disponibles. Éstas son GNU/Linux con el parche RT PREEMPT como sistema operativo de tiempo real (sección 2.4.2.1), *Robotic Operating System 2* como tecnología de *middleware* (sección 2.4.2.2) y todo esto ejecutado sobre una *Raspberry Pi 3* modelo B (sección 2.4.1).

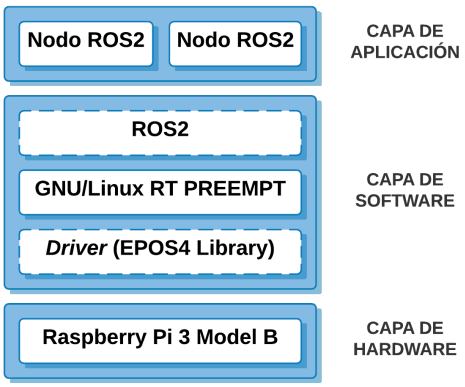


Figura 2.21: Infraestructura del sistema embebido a ser usado en el desarrollo del exoesqueleto.

La capa de aplicación será detallada en el resto de este documento usando la infraestructura descrita anteriormente. Para el desarrollo de la misma se utiliza un proceso de desarrollo basado en componentes.

CAPÍTULO 3

Sistema de Control Centralizado para ALLEX-1

En el presente capítulo se presenta el proceso de especificación de requerimientos para el sistema de control centralizado. En primer lugar, se presentan las generalidades y objetivos del proyecto (sección 3.1). A continuación, se describe el ambiente operativo del sistema y su interacción con componentes externos al exoesqueleto (sección 3.2). Finalmente, se enuncian los requerimientos funcionales (sección 3.3), no funcionales (sección 3.4) y cómo estos influyen en las características de los diferentes subsistemas (sección 3.5).

3.1. Introducción

Según [85], no existe un proceso de ingeniería de requerimientos que sea específico para el desarrollo de sistemas robóticos. Por tanto, se ejecutan las actividades tradicionales de este proceso como son: la elicitación, análisis, negociación, documentación, validación y mantenimiento. En ese estudio se demuestra que existe un mayor énfasis en las actividades de elicitación y la de documentación o especificación. La fase de elicitación incluye todos aquellos procesos ejecutados para entender el dominio de la aplicación como la revisión literaria y de infraestructura (capítulo 1), el problema y finalmente los intereses de los usuarios u organización que solicita el producto.

La información necesaria para detallar los requerimientos del sistema de control centralizado se obtuvo mediante reuniones con docentes y estudiantes de la carrera de Ingeniería Electrónica y Telecomunicaciones, así como docentes de la carrera de Terapia Física de la Universidad de Cuenca, quienes están asociados

al proyecto de investigación. En este capítulo se documenta la especificación de requerimientos de *software* basándose en la estructura del documento propuesto por la norma *IEEE 830-1998 Software Requirements Specification (SRS).*

3.1.1. Propósito

Este capítulo formaliza de manera detallada los requerimientos, funcionalidad y restricciones de un sistema de control y supervisión para un exoesqueleto de extremidades inferiores. Para el control se considera una estructura jerárquica en tres niveles. El nivel inferior se encarga de la adquisición de datos de los sensores (posición, EMG y EEG) y del control de la trayectoria mediante el movimiento de los motores. El nivel intermedio recepta la información del software de adquisición de nivel inferior para generar comandos de intención de movimiento o coordinar los diferentes componentes del sistema. Finalmente, en el nivel jerárquico superior tendrá una interfaz de usuario que permita que los fisioterapeutas puedan controlar las funcionalidades del exoesqueleto y gestionar el registro médico de los pacientes que lo usan. Por tanto, para integrar los componentes del exoesqueleto se realizan tres actividades: 1) adquirir datos de los sensores, 2) ejecutar y coordinar la rotación de los motores y 3) gestionar la transmisión de los datos desde el controlador hacia un servidor y a la aplicación móvil.

3.1.2. Audiencia

La especificación de requerimientos está dirigida a dos grupos de participantes: 1) el personal técnico que está encargado del desarrollo, mantenimiento y extensión de las funcionalidades del sistema de control centralizado para el exoesqueleto y 2) el personal médico encargado de gestionar y configurar las sesiones de rehabilitación mediante el uso del exoesqueleto.

3.2. Descripción General

En esta sección se enuncian las características funcionales del *hardware* y *software* de la primera versión del sistema de control centralizado para el exoesqueleto ALLEX-1 y el mecanismo de interacción con los usuarios. Además, se describen los componentes o módulos internos del sistema, sus dependencias y relaciones.

3.2.1. Perspectiva del Producto

A continuación, se detalla el modelo de contexto del sistema, es decir cada uno de los sistemas de *hardware* o *software* externos con los que interactúa el controlador. Estos sistemas externos están representados de color azul en el diagrama de bloques de la figura 3.1.

1. **Controladores EPOS4:** Son controladores de posición para motores *maxon* a través de los cuales se puede obtener información de los sensores de los

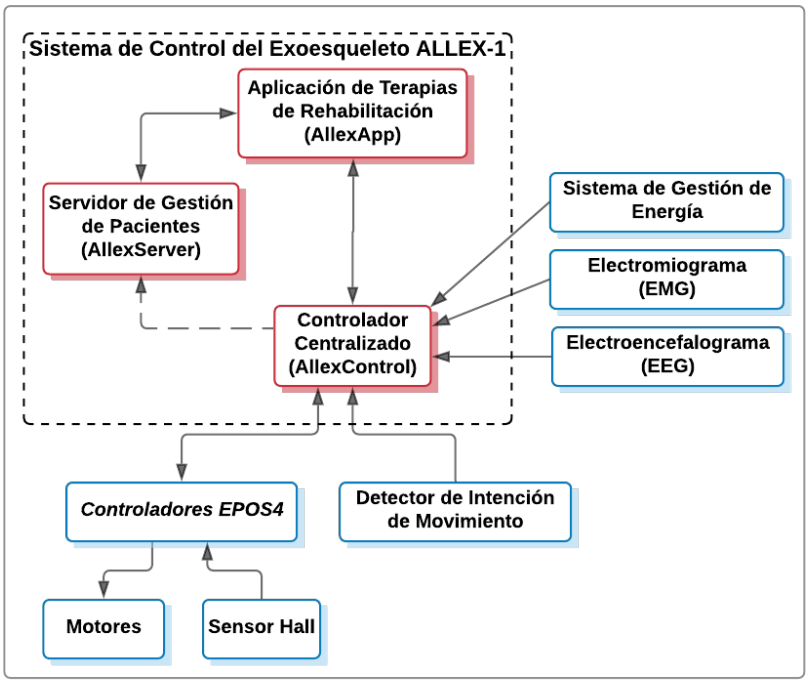


Figura 3.1: Diagrama de contexto del controlador

motores como la posición angular y velocidad mediante los *encoder*, y la corriente mediante el sensor *hall*. Adicionalmente, recibe comandos para controlar y ejecutar la trayectoria angular de los actuadores colocados en las tres articulaciones de las extremidades inferiores: cadera, rodilla y tobillo.

2. **Sistema de Gestión de Energía:** El sistema provee información sobre la temperatura mediante dos termistores colocados en los extremos de la batería, adicionalmente datos de corriente, voltaje de cada una de las celdas y el voltaje total de la batería. Estos datos ingresan a la SBC mediante su puerto serial.
3. **Señales EMG:** Consiste en la medición de una señal eléctrica que reconoce las contracciones musculares de distintos músculos de las extremidades inferiores [86] y que son recolectados desde una red CAN.
4. **Señales EEG:** Es un método de lectura, almacenamiento y análisis de los potenciales eléctricos generados por el sistema nervioso central (CNS) [87], en donde se mide la actividad cerebral al caminar mediante un dispositivo de electroencefalografía.
5. **Detección de Intención de Movimiento:** Red neuronal que tiene como datos de entrada las señales de electromiografía (EMG) y electroencefalografía (EEG) para generar una salida binaria en donde:

   - 0 : No se ha detectado intención de movimiento en el periodo considerado.
   - 1 : Existe intención de movimiento.

3.2.2. Módulos del Sistema

En términos generales el sistema de control depende de tres subsistemas y de las interacciones entre los mismos. Estos subsistemas están representados en la figura 3.1 y cumplen con las siguientes funciones:

- **Aplicación de Terapias de Rehabilitación (*AllexApp*):** Es el encargado de ejecutar las funcionalidades de la capa de control de nivel jerárquico superior. Con este subsistema los fisioterapeutas pueden controlar el funcionamiento del exoesqueleto y gestionar la información de los pacientes. Como se documentó en la sección 2.5, existen diferentes mecanismos de interacción con el exoesqueleto; sin embargo, como ALLEX-1 es un exoesqueleto móvil, también se requiere que el mando que permite su control esté en un dispositivo portátil como los celulares o las tabletas.
- **Controlador Centralizado (*AllexControl*):** Cumple las funciones de control de nivel intermedio e inferior al encargarse de: 1) adquirir información de los sensores, 2) coordinar la ejecución de movimientos y 3) transmitir los datos de los sensores hacia *AllexApp*.
- **Servidor de Gestión de Pacientes (*AllexServer*):** Es un servidor web desplegado en una red accesible para el usuario de la aplicación móvil. Este subsistema se encarga de la recepción y almacenamiento de los datos del paciente y la recuperación de datos históricos de sesiones de rehabilitación previas.

La interacción de estos tres subsistemas se realiza de la siguiente manera:

- *AllexApp* envía mensajes a *AllexControl* con el objetivo de controlar los estados del exoesqueleto. También recibe y muestra datos de la posición angular de los motores de las articulaciones en una gráfica temporal, así como las alertas de errores o advertencias que sucedan dentro del controlador (como por ejemplo la desconexión de un sensor).
- *AllexApp* se comunica con *AllexServer* para administrar la información del historial médico del paciente en relación a las sesiones en las cuales usó el exoesqueleto.
- *AllexControl* envía los datos de los sensores hacia *AllexServer* cuando el exoesqueleto se encuentre en modo de lectura o ejecutando un movimiento.

Esta última interacción entre *AllexControl* y el *AllexServer* no es implementada en el presente proyecto. Sin embargo, algunas guías o alternativas para su futuro desarrollo se presentan en la sección 7.2.2.

3.2.3. Participantes del Proyecto y Características

A continuación, se redactarán los diferentes roles y características de los usuarios que están encargados de la producción y mantenimiento del *software*, así como los usuarios finales del mismo como los fisioterapeutas que supervisan al paciente y los ingenieros que desarrollan aspectos del robot diferentes al *software*.

Actor	Rol
ACT-1 - Desarrolladores del Sistema	Se refiere tanto a aquellas personas que participan en el proceso de especificación de requerimientos, diseño e implementación de los sistemas. Así como aquellos que realizan el mantenimiento o extensión de la funcionalidad de los subsistemas de la aplicación.
ACT-2 - Fisioterapeutas	Según la definición declarada por la Organización Mundial de la Salud, en su clasificación de los trabajadores de la salud [88], los fisioterapeutas son profesionales que evalúan, planifican e implementan programas de rehabilitación que mejoran o restauran las funciones motoras humanas, las habilidades de movimiento, calman el dolor y tratan o previenen dificultades físicas como lesiones, enfermedades y otras discapacidades. Son los principales usuarios de la aplicación móvil debido a que tienen el conocimiento necesario para supervisar, planificar y revisar sesiones de terapia con el uso de un exoesqueleto que permitan una mejora en la movilidad de los pacientes. Por este motivo necesitan que la aplicación tenga funcionalidades para personalizar el movimiento.
ACT-3 - Personal técnico	Principalmente ingenieros en electrónica y telecomunicaciones o ingenieros de sistemas quienes pueden modificar las funcionalidades del controlador centralizado, acceder o extraer información del servicio de datos de tal manera que estos pueden ser procesados posteriormente.

Tabla 3.1: Participantes del proyecto.

3.2.4. Ambiente Operativo

El ambiente operativo, denominado infraestructura del sistema, ha sido definido en 2.6 para el controlador centralizado. En resumen, usa como *hardware* la SBC *Raspberry Pi* 3 Modelo B debido a su tamaño, precio y aplicaciones. El sistema operativo *Raspbian* con el parche para tiempo real PREEMPT RT. El *middleware* elegido es ROS2 que usa como mecanismo de comunicación el protocolo DDS.

Por otro lado, *AllexApp* es una aplicación móvil diseñada e implementada con el sistema operativo Android. La razón por la cual se implementa inicialmente para este sistema operativo es la accesibilidad a las herramientas de desarrollo, mismas que pueden ser usadas desde los sistemas operativos *Windows*, *GNU/Linux* y *macOS*. El soporte mínimo será para la API 19 del sistema operativo Android, también denominada KitKat (versión 4.4) y la versión máxima soportada es la API 27. Como se puede apreciar en la figura 3.2, esto significa que hasta el 26 de octubre del 2018 puede ser usado en el 96,5 % de los dispositivos con Android activo. En el caso de *AllexServer* se usa *SpringBoot* como *framework* de desarrollo para el servidor de aplicaciones puesto que integra diferentes paquetes propios del desarrollo de servidores y publicaciones de API, así como la facilidad de despliegue

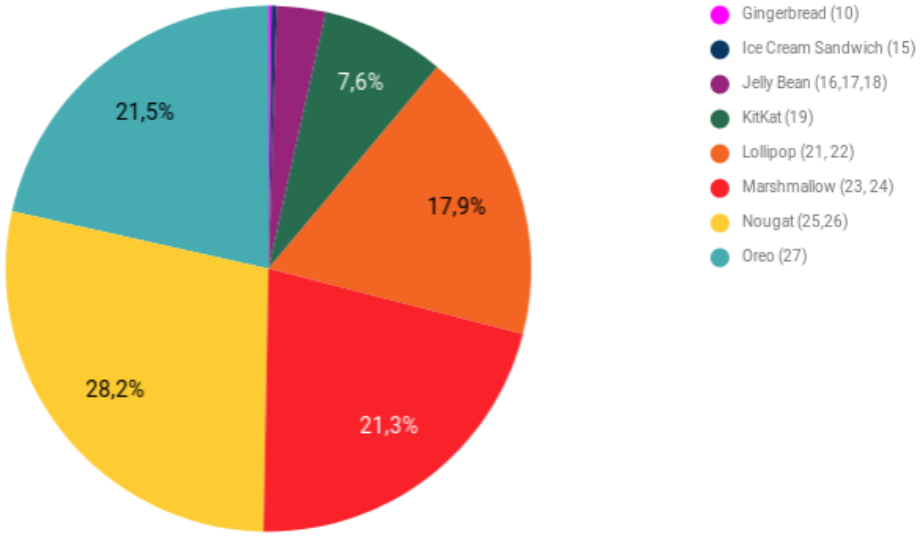


Figura 3.2: Distribución del uso de versiones de Android [89].

del *software* debido a que posee un servidor de aplicaciones embebido.

3.2.5. Restricciones de Diseño e Implementación

A continuación, se enuncian algunas restricciones asociadas al desarrollo del proyecto:

- Se requiere que el dispositivo móvil usado tenga conexión a internet para que pueda enviar los datos a *AllexServer*.
- El lenguaje de programación usado para el desarrollo debe ser C++ o *Python* para facilitar la integración de los módulos al sistema que depende de ROS2. En su defecto se puede realizar la implementación en un lenguaje de programación para el cual exista una librería con soporte de algún proveedor de tecnología DDS. Por ejemplo *OpenDDS* y *RTI Connext* tienen soporte para Java.
- De manera preferencial para los procesos que requieran ser ejecutados en tiempo real se debe usar el lenguaje de programación C++ para su desarrollo.
- La comunicación de los eventos de control debe ser confiable.

3.2.6. Suposiciones y Dependencias

A continuación, se redactan suposiciones, dependencias y limitaciones consideradas durante el desarrollo del proyecto:

- No se implementarán los algoritmos de detección de intención de movimiento debido a que no forma parte del alcance del presente proyecto. Sin embargo, se simulará la generación de dichos eventos y la consecuente ejecución del movimiento.

- Todos los proyectos asociados al exoesqueleto ALLEX-1 deberán tener sus sistemas desarrollados en C++ o *Python* para facilitar su integración. Caso contrario se deberá simular la generación de sus señales.
- El exoesqueleto actualmente tiene construida una de sus extremidades; sin embargo, el diseño del sistema debe contemplar el funcionamiento de la segunda.

3.3. Requerimientos del Sistema

En esta sección se enuncian los requerimientos funcionales del sistema de control centralizado de acuerdo a las funcionalidades deseadas por los participantes del proyecto. Los requerimientos están organizados de acuerdo al subsistema en el cual va a ser implementado.

3.3.1. Aplicación de Sesiones de Rehabilitación (*AllexApp*)

APP1: Permitir el acceso a la aplicación únicamente a aquellos usuarios que han sido registrados en *AllexServer* por un usuario administrador.

APP2: Gestionar la ficha médica para el paciente. Se define como ficha médica a un documento en donde el fisioterapeuta registra información como datos personales y las condiciones o circunstancias de los pacientes. Para el presente sistema se requiere el registro o edición de datos como: un identificador (cédula de identidad o pasaporte), nombre, fecha de nacimiento, sexo, peso, estatura y diagnóstico médico.

APP3: Recuperar la ficha médica de un paciente mediante su código de identificación o nombre.

APP4: Permitir la creación de una sesión de rehabilitación para un paciente. Se entiende como una sesión de rehabilitación a el proceso en el cual se ha controlado el exoesqueleto para ejecutar algún ejercicio de rehabilitación. Para la creación de este tipo de sesiones se debe incluir un título y descripción para la misma.

APP5: Gestionar la conexión de la aplicación con el controlador centralizado para poder controlarlo remotamente.

APP6: Solicitar al exoesqueleto que inicie el proceso de lectura de datos de los sensores.

APP7: Mostrar la información de la posición angular de los motores en la pantalla a manera de una gráfica temporal durante el proceso de lectura de datos.

APP8: Solicitar al exoesqueleto que termine el proceso de adquisición de datos.

APP9: Ofrecer la opción de configurar el movimiento previo a su ejecución. Eso incluye:

- Elegir el tipo de movimiento.
- Configurar la velocidad del ciclo de movimiento.
- Seleccionar el número de ciclos del movimiento a ser repetidos.

APP10: Solicitar al exoesqueleto para que inicie el movimiento previamente configurado.
APP11: Solicitar al exoesqueleto que cancele el movimiento que está en ejecución.
APP12: Mostrar la información del estado de carga de la batería que proviene de *AllexControl*. Esta puede estar en tres estados: alta, media y baja.
APP13: Recibir notificaciones de error que hayan sucedido en *AllexControl*, como los errores de configuración de movimiento o comando inexistente.
APP14: Mostrar una gráfica temporal con los datos recolectados al terminar un movimiento. El médico tiene la opción de cancelar y eliminar los datos adquiridos para reintentar nuevamente el ejercicio o guardar dichos datos en el servidor.
APP15: Listar el historial de sesiones de rehabilitación del paciente.
APP16: Al seleccionar una sesión de rehabilitación que ha sido completada, es decir, aquella en la que el fisioterapeuta ha guardado los datos de la sesión, el médico podrá visualizar las gráficas temporales de la posición angular de los motores situados en cada una de las articulaciones.
APP17: Permitir el registro del diagnóstico funcional del paciente. Este diagnóstico corresponde a un texto que los fisioterapeutas pueden ingresar al evaluar una sesión de rehabilitación.
APP18: Cerrar la sesión del usuario en la aplicación.

3.3.2. Controlador Centralizado (*AllexControl*)

CTR1: Adquirir a través del puerto serial los datos del Sistema de Gestión de Energía como la temperatura, corriente, voltaje de las celdas y voltaje total del exoesqueleto.
CTR2: Al detectar un evento de lectura o de ejecución de movimientos del exoesqueleto, se deberá configurar e iniciar el proceso de adquisición de los siguientes datos:

- En el caso de los motores se requiere obtener su posición angular, corriente y velocidad.
- Señales de electromiografía.
- Señales de electroencefalografía.

CTR3: Desactivar el estado de adquisición de señales cuando el usuario a enviado el evento de *"culminación de lectura"*.
CTR4: Configurar los parámetros del movimiento a ser efectuado como:

- Tipo de movimiento.
- Duración del ciclo de movimiento en segundos
- Número de repeticiones del ciclo de movimiento.

CTR5: Iniciar la ejecución de un ejercicio en caso de detectar un evento de intención de movimiento.
CTR6: En caso de detectar un evento de error mientras el exoesqueleto está ejecutando un movimiento, este deberá detenerse al terminar el ciclo de caminata en el que se encuentre. Una vez que termine el ciclo, los motores deberán

colocar al paciente en una posición final previamente definida. En el caso de la camina el paciente debe estar de pie.

CTR7: Gestionar la conexión inalámbrica entre el dispositivo que ejecuta *AllexApp* con *AllexControl*.

CTR8: Notificar a *AllexApp* sobre los cambios de estado en el nivel de carga de la batería del exoesqueleto.

3.3.3. Servidor de Gestión de Pacientes (*AllexServer*)

SRV1: Autorizar el acceso de los recursos web a los usuarios del dispositivo móvil cuyas credenciales estén registradas en el sistema.

SRV2: Gestionar el historial médico de los pacientes.

SRV3: Recuperar información personal del paciente y el listado de sus sesiones de rehabilitación.

SRV4: Gestionar los datos de posición angular de cada motor en la sesión de rehabilitación del paciente seleccionado.

3.4. Requerimientos No Funcionales

NFR1: Para asegurar la operación del sistema en tiempo real, se debe adquirir los datos con las frecuencias presentadas en la tabla 3.2:

Señal	**Frecuencia (*Hz*)**
Señales de electromiografía	1000
Señales de electroencefalografía	138
Datos sobre posición, corriente y velocidad de los motores	100
Datos del Sistema de Gestión de Energía	1

Tabla 3.2: Frecuencias de muestreo para la adquisición de datos

NFR2: Únicamente los usuarios registrados y autorizados por *AllexServer* pueden ingresar a *AllexApp*.

NFR3: Se usan archivos de configuración para que el personal técnico especializado pueda modificar parámetros del funcionamiento del exoesqueleto. Estos parámetros son la frecuencia de muestreo, tipos de datos a ser adquiridos, tipo de ambiente de ejecución (prueba o producción), estado del funcionamiento de las articulaciones.

NFR4: La aplicación debe ser fácil de extender. Es decir, nuevas funcionalidades y sensores podrán ser introducidos en la aplicación.

3.5. Descripción de los Subsistemas

En esta sección se extiende la descripción de los subsistemas en base a los requerimientos planteados anteriormente. El objetivo es tener una visión general de la funcionalidad provista por todo el sistema antes de indicar sus detalles de diseño e implementación.

3.5.1. Aplicación de Terapias de Rehabilitación (*AllexApp*)

AllexApp es una aplicación para dispositivos móviles Android. Ésta fue desarrollada para gestionar los procesos de terapias de rehabilitación mediante el uso de ALLEX-1. La secuencia normal de funcionalidades para la pantalla de la aplicación se describe a manera de un diagrama de actividades en la figura 3.3.

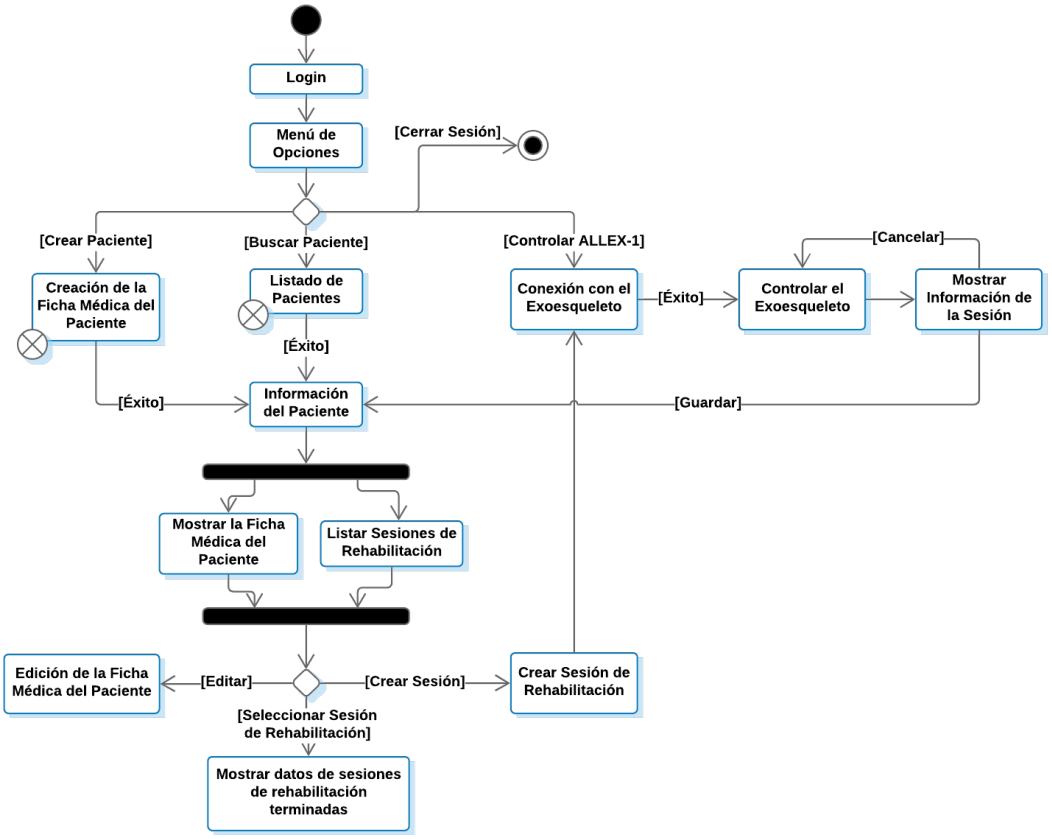


Figura 3.3: Diagrama de navegación para la interfaz móvil.

La primera funcionalidad a la cual se accede en *AllexApp* permite gestionar las sesiones de los usuarios. Esto incluye las configuraciones que realiza la aplicación para comunicarse con *AllexServer* y verificar si el usuario está autorizado para acceder a alguno de los servicios web publicados. La validación se realiza mediante

los pares de número de identificación y contraseña. Una vez que la aplicación recibe la autorización de ingreso desde el servidor, se carga el menú de opciones.

De acuerdo al flujo normal de actividades de la aplicación (figura 3.3), desde el menú principal se puede acceder a otros cuatro flujos de actividades como son crear un paciente, buscar un paciente, controlar el exoesqueleto y cerrar la sesión. En el caso de la creación del paciente, se contempla el almacenamiento de datos como: el número de cédula o pasaporte a manera de identificación única del paciente, sus nombres y apellidos, la fecha de nacimiento, sexo y el diagnóstico médico en el que se detalla el estado de salud del paciente. Adicionalmente, otros dos datos de interés son el peso y la estatura, mismos que podrían ser usados en futuros estudios para comprobar la influencia de estos factores en el desempeño del exoesqueleto. Si el paciente es exitosamente creado y almacenado en el servidor se carga la pantalla de información del paciente.

Una segunda forma de acceder a la información del paciente es mediante la opción de búsqueda de pacientes en el menú principal. Para ello el usuario debe ingresar el número de identificación del paciente o el nombre del mismo. En caso de que el paciente exista se carga la pantalla de información de paciente. En esta pantalla se muestran dos tipos de datos: 1) la información de la ficha médica que fue previamente registrada en la pantalla de creación de pacientes y 2) la información sobre las sesiones de rehabilitación que ha tenido el paciente.

Respecto a la gestión de sesiones de rehabilitación existen dos funcionalidades que pueden ser usadas desde la pantalla de información del paciente: 1) la creación de una sesión de rehabilitación y 2) el acceso a una sesión de rehabilitación completada. En el caso de la creación de una nueva sesión de rehabilitación se solicita al usuario que ingrese un título y una descripción para la sesión. La razón es que se requiere de un texto que facilite la búsqueda de una sesión de rehabilitación. Una vez que se ha creado la sesión de rehabilitación en el servidor, se carga la pantalla para el control del exoesqueleto.

El proceso de control inicia con la búsqueda y conexión de la *Raspberry Pi* que contiene el *software AllexControl*. Una vez que la conexión se establece entre los dos dispositivos, se presenta al usuario una pantalla similar al prototipo de la figura 3.4. Esta pantalla presenta a su izquierda una sección para la configuración del movimiento, dos botones para la ejecución de los comandos de lectura y ejecución del movimiento, y finalmente gráficas temporales para la visualización de las curvas de posición angular de los motores en el lado derecho de la pantalla. El botón de lectura solicita al exoesqueleto que obtenga estas métricas de posición de los motores. El botón de ejecutar permite realizar el movimiento del exoesqueleto siempre y cuando se haya llenado el formulario de configuración con el número de repeticiones del ciclo de movimiento y la velocidad del ciclo.

Figura 3.4: Prototipo de la interfaz gráfica de control.

Una vez que se ha ejecutado un movimiento, el usuario puede visualizar la gráfica temporal total de cada articulación en una nueva pantalla. Posteriormente, el usuario puede decidir si guardar los datos recolectados en *AllexServer* o eliminarlos y reintentar un nuevo movimiento. Esto es la base para la última pantalla de la aplicación en la cual se pueden visualizar los datos almacenados de sesiones de rehabilitación previas. En esta misma pantalla el médico puede escribir el diagnóstico funcional del paciente basado en el análisis de los datos recolectados.

3.5.2. Controlador Centralizado

El controlador centralizado (AllexControl) es un sistema que es ejecutado sobre la SBC. Se denomina centralizado debido a que es un solo dispositivo el que se encarga de las tareas de supervisión y coordinación del exoesqueleto. Sin embargo, la estructura interna del controlador está conformada por procesos distribuidos los cuales se comunican entre sí mediante paso de mensajes con datos o eventos. Por este motivo, la mayor parte de sistemas de control usados en robótica se basan en un modelo de comunicación de estímulo-respuesta [90] como la arquitectura basada en eventos. Esto permite que el *software* desarrollado sea flexible y extensible.

Los módulos generales de este sistema pueden observarse en el diagrama de bloques de la figura 3.5:

Adquisición de Datos: Es el módulo que se encarga de manejar el estado de los sensores del sistema. Desde el punto de vista de coordinación verifica si el módulo de sensor se encuentra en el estado activo o inactivo. En caso de que esté

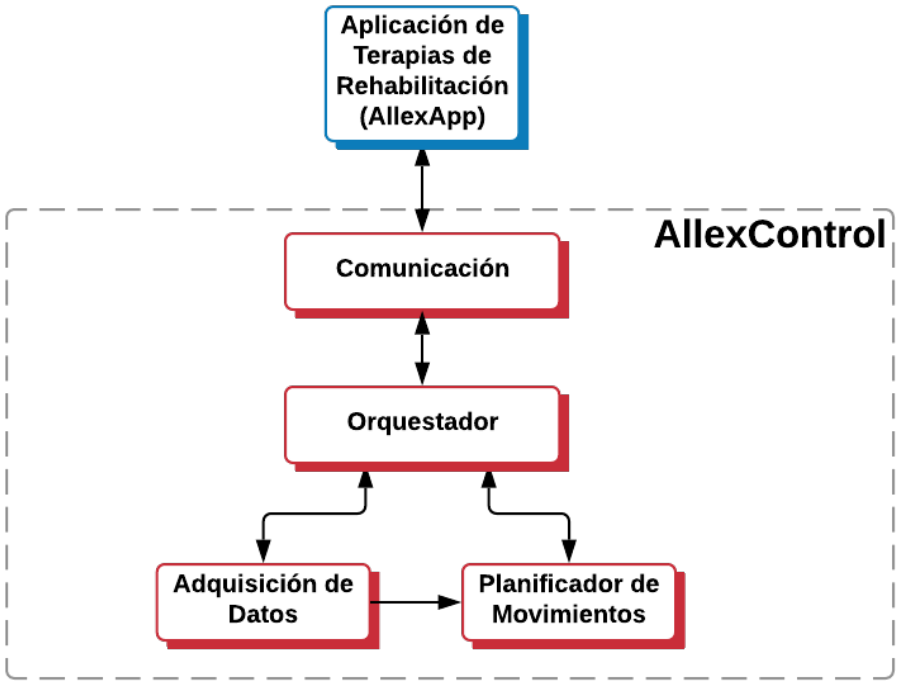


Figura 3.5: Arquitectura del Controlador Centralizado.

activo, se encarga de adquirir los datos de su entorno en la frecuencia configurada.

Planificador de Movimientos: Módulo encargado de configurar y controlar la ejecución de los ejercicios de rehabilitación como la caminata. La configuración está asociada a la duración de un ciclo de movimiento y el número de repeticiones del mismo. El control de la trayectoria de movimiento que deben seguir los motores se basa en las curvas biomecánicas de movimiento angular de las articulaciones de la cadera, rodilla y tobillo en el plano sagital.

Este módulo requiere de la información de los sensores de electromiografía y electroencefalografía para ejecutar los algoritmos de intención de movimiento. Si el sistema detecta que el usuario tiene intención de moverse durante la ejecución de un ejercicio de rehabilitación, entonces ejecuta un ciclo de movimiento.

Comunicación: Este módulo gestiona la conexión y transmisión inalámbrica de datos del controlador centralizado hacia *AllexApp*. Esto implica la transmisión de tres tipos de datos:

- La recepción de comandos de ejecución de acciones desde *AllexApp*.
- La transmisión de los datos de posición angular de los motores para que pueda ser visualizado en tiempo real.
- La transmisión de estados del exoesqueleto como el estado de batería, errores como por ejemplo un parámetro equivocado en la configuración de

movimientos o eventos como aquellos que notifican que el movimiento ha terminado.

Orquestador: *AllexControl* al ser internamente un sistema distribuido requiere de un módulo que se encargue de gestionar o coordinar la interacción entre todos los sistemas.

3.5.3. Servidor de Gestión de Pacientes

El servidor web (*AllexServer*) está diseñado para proporcionar la lógica de negocio asociada a la gestión de datos de los pacientes que han sido registrados en la aplicación móvil o la información sobre posición angular de los motores obtenidos desde *AllexControl*.

Capítulo 4

Diseño del Sistema de Control

En este capítulo se presenta una de las principales contribuciones del presente proyecto de titulación que consiste en el diseño del sistema de control centralizado para *ALLEX-1*. El diseño inicia con el modelo de la arquitectura del sistema, representado mediante diagramas de definición de bloques propuestos por SysML (sección 4.2). A continuación, se describe el comportamiento de los subsistemas de control centralizado. La descripción inicia con el comportamiento interno de *AllexControl*, representado como un conjunto de diagramas de máquinas de estado y de actividad (sección 4.3). A continuación, se documenta el mecanismo de interacción entre *AllexApp* y *AllexControl* (sección 4.4). Respecto a *AllexServer*, se describen el modelo de datos y las interfaces web que son provistas a la aplicación móvil (sección 4.5). Finalmente, se describe la interacción de *AllexApp* con el servidor de aplicaciones (sección 4.6).

4.1. Introducción

El diseño es aquella etapa de desarrollo de *software* en la cual se definen los componentes del sistema, su comportamiento e interrelaciones. A diferencia de la etapa de requerimientos en la cual se establece qué es lo que se debe realizar, el diseño define cómo va a comportarse el sistema. Generalmente, el diseño es independiente de la tecnología de programación de implementación. El nivel de granularidad del modelo depende del dominio del proyecto y del mantenimiento ante los cambios de requerimientos. El diseño no es un proceso estático, se requieren mejoras en los modelos cada vez que se proponen nuevas ideas o se encuentran mejores soluciones [90].

Se usan lenguajes de modelado como un mecanismo para explicar e ilustrar

la manera en la que está estructurado el sistema. En el dominio de *software* para sistemas robóticos se prefiere usar estándares de la OMG como el lenguaje de modelado de sistemas (*Systems Modeling Language* - SysML) o, en su defecto, MARTE (*Modeling and Analysis of Real Time and Embedded Systems*) para sistemas embebidos con características de tiempo real. Según [91], no se usa UML debido a que está centrado en procesos de *software* y es difícil expresar conceptos de los componentes de *hardware* del sistema. Por tal motivo, se usa la especificación SysML con el objetivo de reducir la brecha semántica entre los sistemas, el *software* y otras disciplinas de la ingeniería.

4.2. Arquitectura del Sistema

La primera actividad del proceso de diseño consiste en especificar los componentes que tendrá el sistema mediante un diagrama de definición de bloques (BDD). En este diagrama se presentan las relaciones de agregación, composición o herencia entre los componentes del sistema de control y la multiplicidad de dichas relaciones. Cada uno de los subsistemas corresponde a una entidad de *software* independiente como se presenta en la figura 4.1. Se debe indicar que estos diagramas no indican la manera en la cual los bloques se comunican entre sí, únicamente representan su estructura. Cada uno de estos subsistemas está consti-

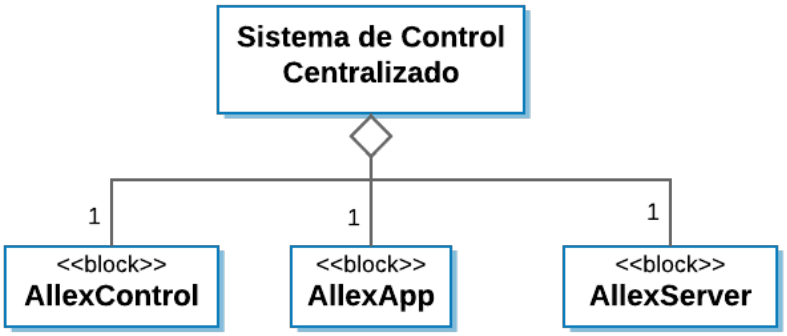


Figura 4.1: Diagrama de definición de bloques para el Sistema de Control Centralizado.

tuido internamente por un conjunto de bloques que le otorgan su funcionalidad. En el caso de *AllexControl*, su diagrama de definición de bloques se presenta en la figura 4.2a con un total de 18 partes:

- El **orquestador** coordina la actividad de los bloques del sistema.
- El *Sistema de Gestión de Energía* es un sistema independiente que adquiere y transmite datos de consumo energético de los componentes del exoesqueleto. El módulo de batería es la parte de *software* que transmite la información hacia otros componentes.
- El módulo de comunicación permite el intercambio de información con la aplicación móvil. Consta de dos bloques:

  - **Transmisión**: Para el envío de datos y estados a *AllexApp*.
  - **Recepción**: Para recibir los comandos enviados desde *AllexApp*.
- El bloque **sensor** adquiere periódicamente los datos en un intervalo de tiempo. Este comportamiento es heredado por los bloques para los datos de EMG, EEG y los sensores de los motores.
- El bloque **Intención EMG** detecta la intención de movimiento a partir de los datos de electromiografía.
- El bloque **Intención EEG** detecta la intención de movimiento a partir de los datos de electroencefalografía.
- El bloque **planificador** de movimiento coordina la ejecución de las secuencias de movimiento.
- El bloque **Extremidad** corresponde a un bloque para cada extremidad del exoesqueleto que incluye la funcionalidad de cada articulación. Cada articulación está conformada por dos bloques: un actuador y un sensor. El bloque **actuador** se encarga de la ejecución de los comandos de cambio de posición de los motores. El bloque **sensor** recupera los datos de la posición, corriente y velocidad de los motores. El bloque **EPOS4** se refiere al *hardware* para el controlador de posición que es configurado por cada extremidad.

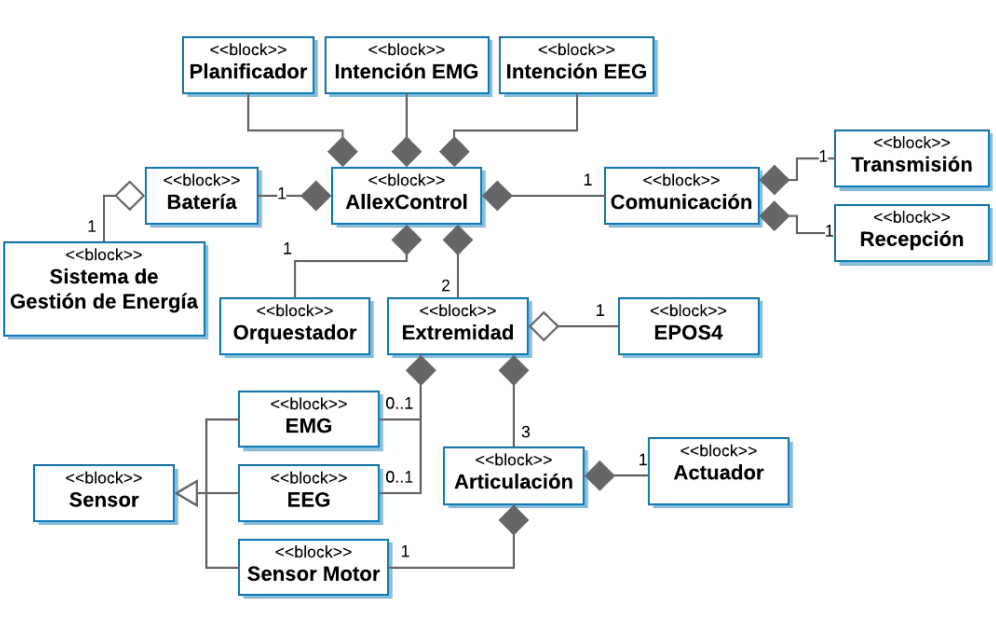
En el caso de *AllexApp* (figura 4.2b) su sistema está compuesto por cuatro bloques asociados a las interfaces gráficas y mecanismos de interacción:

- El bloque de **comunicación** para transferir información con el controlador centralizado. Consta de dos partes:
  - El de **transmisión** que envía comandos a *AllexControl*.
  - El de **recepción** que recibe datos y eventos desde *AllexControl*.
- El componente **ControlGui** corresponde a la interfaz gráfica que el usuario usa para enviar comandos al exoesqueleto y visualizar los datos.
- El bloque **Cliente** es el encargado de realizar solicitudes a los recursos web de *AllexServer*.
- El bloque **UsuarioGui** se encarga de manejar aquellas pantallas de la interfaz gráfica de usuario para gestionar la comunicación con el bloque **servidor** de *AllexServer*.

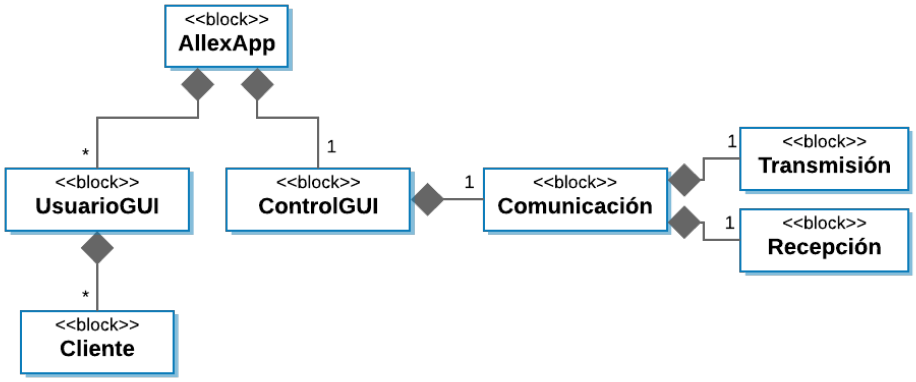
Finalmente, *AllexServer* está constituido por dos bloques como se presenta en la figura 4.2c. Uno de los bloques corresponde al **Servidor**, el cual responde a las solicitudes del **Cliente** de *AllexApp*. El otro bloque es el de la base de datos de series de tiempo la cual se encarga de la persistencia de la información.

4.3. Diseño del Subsistema AllexControl

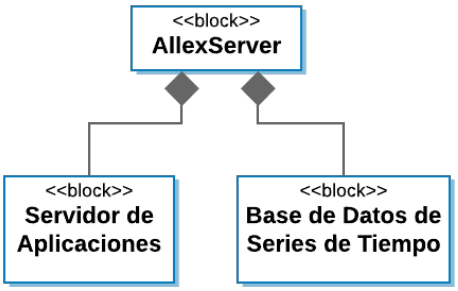
En esta sección se explica la lógica interna del controlador al describir la comunicación entre sus componentes y la administración de sus estados.



(a) AllexControl



(b) AllexApp



(c) AllexServer

Figura 4.2: Diagrama de definición de bloques para *AllexApp*, *AllexControl* y *AllexServer*

4.3.1. Coordinación de Componentes

Con el objetivo de coordinar el comportamiento entre todos los componentes del exoesqueleto es necesario definir una máquina de estados global como la que se muestra en la figura 4.3. Ésta está basada en las principales funcionalidades del exoesqueleto como son la conexión desde *AllexApp*, la lectura y la ejecución de movimientos. Adicionalmente, cada bloque de *AllexControl* tiene una máquina de estados interna similar al ciclo de vida indicado en la sección 2.4.3.7. Esta máquina de estados global está conformada por los siguientes estados y transiciones:

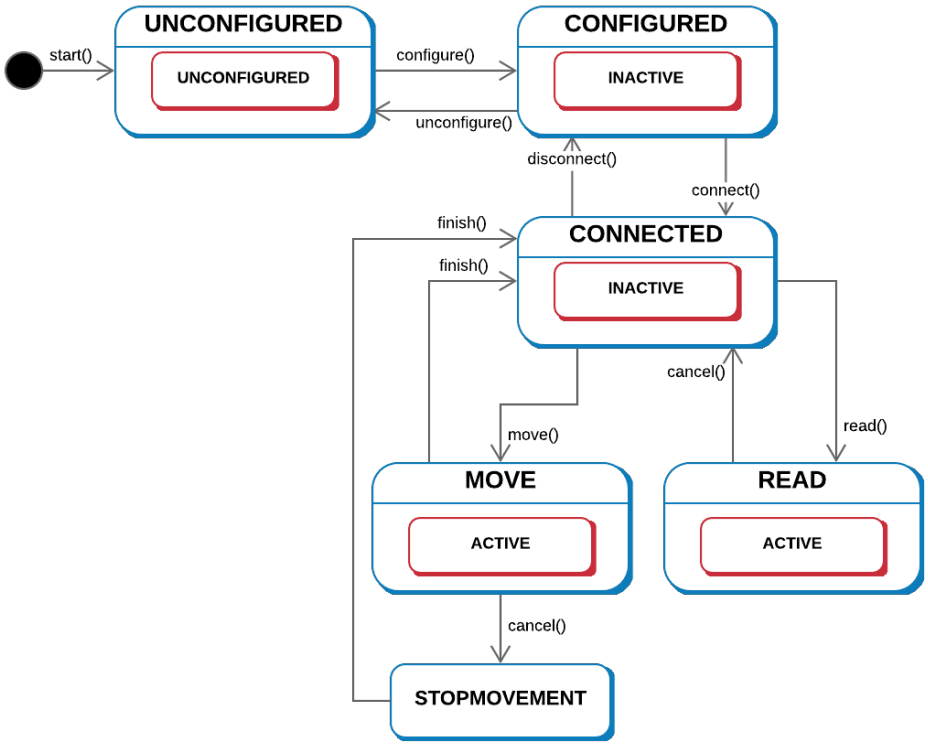


Figura 4.3: Diagrama de máquinas de estado del exoesqueleto.

- **UNCONFIGURED:** Este es el estado que tiene *AllexControl* cuando inicia su proceso de ejecución.
  - *configure:* Transición en la cual se cambia el estado del ciclo de vida de cada bloque del sistema desde *UNCONFIGURED* a *INACTIVE*. Como resultado, todos los bloques han configurado los recursos requeridos para su ejecución. Si esta condición no se cumple, el sistema sigue reintentando esta transición.
- **CONFIGURED:** Este es el estado que tiene *AllexControl* cuando todos los bloques con una máquina de estados interna de ciclo de vida se encuentran en el estado *INACTIVE*. Adicionalmente, *AllexControl* está listo para aceptar conexiones desde un dispositivo móvil.

  - *unconfigure:* Transición en la que alguno de los bloques del sistema regresa al estado *UNCONFIGURED*.
  - *connect:* Transición en la cual se ha aceptado la solicitud de conexión inalámbrica desde *AllexApp*.

- **CONNECTED:** Estado en el cual *AllexControl* está conectado con *AllexApp* y por tanto recibe comandos o envía datos.
  - *read:* Transición en la que se cambia el estado del ciclo de vida de los sensores a *ACTIVE*. En este estado inicia el proceso de adquisición y transmisión de datos.
  - *move:* Transición en la cual se cambia el estado del ciclo de vida de los sensores y actuadores a *ACTIVE*. A continuación, inicia la transmisión de datos y ejecución de movimientos. En esta etapa también se configura la información del movimiento a ser ejecutado por cada actuador.
  - *disconnect:* Transición en la cual se ha detectado una pérdida de la conexión con *AllexApp*. Como consecuencia, el exoesqueleto regresa al estado *CONFIGURED* para esperar una nueva conexión.
- **READ:** Estado en el cual los sensores están publicando datos en el sistema.
  - *cancel:* Es un evento de finalización de lectura de datos que ha llegado desde *AllexApp*. Como resultado, todos los sensores que estaban en estado *ACTIVE* en su ciclo de vida pasan al estado *INACTIVE*.
- **MOVE:** Estado en el cual el exoesqueleto está realizando la rutina de ejercicios que fue solicitada por el fisioterapeuta desde *AllexApp*.
  - *finish:* Se han ejecutado los ciclos de movimiento que fueron configurados desde *AllexApp*. Por tanto, el exoesqueleto regresa al estado *CONNECTED* para recibir nuevos comandos. Adicionalmente, los bloques de sensores y actuadores regresan al estado *INACTIVE*.
  - *cancel:* Se ha cancelado la ejecución del movimiento de rehabilitación desde *AllexApp*. Como resultado, inicia la rutina de finalización de movimiento.
- **STOPMOVEMENT:** Inicia la rutina de finalización de movimiento en la cual el exoesqueleto regresa a una postura de inicio de movimientos. En el caso de una caminata la posición final es dejar al paciente en una postura de pie.
  - *finish:* Ha terminado la ejecución del movimiento, por tanto todos los sensores y actuadores regresan al estado *INACTIVE* de su ciclo de vida.

Cada uno de los estados presentados son modelados como elementos de diferentes enumeraciones. La máquina de estados presentada tiene la enumeración *ExoState* (figura 4.4). De la misma manera los eventos que son enviados desde la aplicación móvil y que provocan el cambio de estado se representan mediante la enumeración *ExoCmd*.

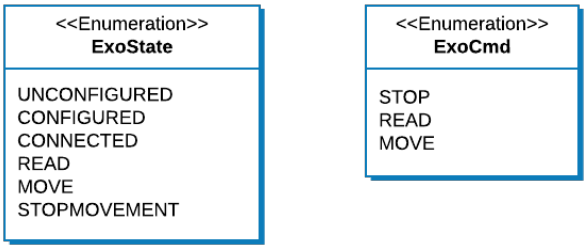


Figura 4.4: Enumeración *ExoState* y *ExoCmd* para estados del exoesqueleto.

4.3.2. Estructuras de Datos de los Sensores

Se definió la estructura de datos **SensorData** para que todos los sensores transmitan la información que han adquirido en el sistema. El formato de este mensaje está presentado en el bloque de la figura 4.5. El objetivo es facilitar el proceso de validación y transformación de datos por parte de los bloques que requieren consumir dicha información.

Adicionalmente se ha definido la estructura de datos **Time** para almacenar la información sobre el instante de tiempo en el que se adquirió el dato. En este caso las marcas temporales son representadas mediante el formato UNIX *epoch*. Este formato contabiliza el número de segundos que han transcurrido desde el primero de enero de 1970 a las cero horas del tiempo universal coordinado (UTC) hasta el momento en que se registró el dato en la variable. El parámetro *sec* de la estructura *Time* almacena este número de segundos. La variable *nanosec* registra el número de nanosegundos desde *sec* hasta el momento en que el dato fue registrado. De esta manera, aumenta la precisión con la que se obtienen los datos.

El tipo de dato **SensorData** es en dónde se almacena la información adquirida de los sensores y está conformado por seis parámetros:

- *limb:* indica el identificador de la extremidad en la cual se leyó el dato. Ésta puede tomar el valor *RIGHT* para la pierna derecha y *LEFT* para la pierna izquierda como se presenta en la enumeración **LimbType**.
- *joint:* almacena la articulación en la cual se originó el dato. Esta puede ser *HIP* para la cadera, *KNEE* para la rodilla o *ANKLE* para el tobillo, de acuerdo a la enumeración **JointType**.
- *node:* especifica el nombre del bloque del cual proviene la información. Por ejemplo, en el caso de los actuadores su identificador sería *l_knee* para los sensores de la rodilla de la pierna izquierda.
- *variable:* especifica la variable enviada por cada nodo. Esto debido a que un mismo nodo puede publicar varias variables. Por ejemplo, los actuadores envían variables de posición, corriente o velocidad.
- *data:* escribe el valor cuantitativo de la variable.
- *timestamp:* especifica la marca de tiempo en la cual se obtuvo la información

usando la estructura de datos **Time** que fue descrita anteriormente.

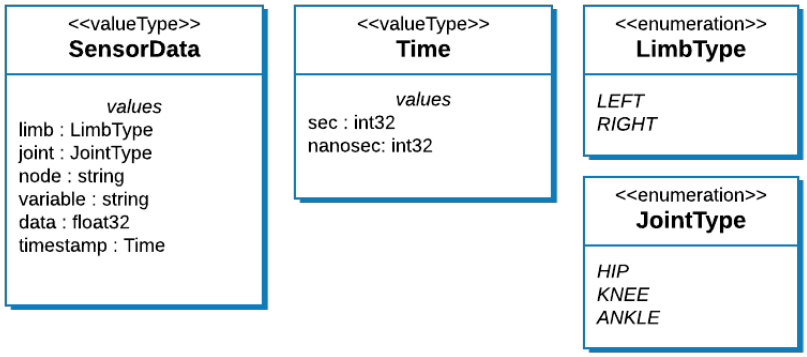


Figura 4.5: Tipo de dato SensorData para el intercambio de mensajes de sensores.

4.3.3. Bloques de Sensores

Los bloques que heredan la funcionalidad del bloque **sensor** son aquellos para los motores, EMG y EEG. La diferencia entre cada uno de ellos está en el significado de los datos que transmiten por sus puertos de salida y la frecuencia con que lo realizan.

En estos bloques es conveniente especificar valores que puedan ser configurados antes de la ejecución o durante la misma. Por ejemplo, la frecuencia para la adquisición y transmisión de datos podría cambiar dependiendo de la carga computacional sobre el sistema o dependiendo de los requerimientos de adquisición de datos de otros módulos. Otro tipo de variables son aquellas que tienen valores lógicos que permiten que el desarrollador tome decisiones. Por ejemplo, elegir si los datos de una variable deben ser adquiridos y transmitidos, o simplemente ignorados. Esto sucede en el caso de los sensores de los actuadores, en donde se puede decidir si el bloque publica sus tres variables o solamente una de ellas. La activación del ambiente de prueba se realiza mediante el valor lógico denominado *test*. Esta configuración es útil durante el proceso de depuración del sistema y en aquellos casos en los que no se tiene acceso continuo al *hardware* del exoesqueleto. Si su valor lógico es verdadero, el bloque genera datos simulados. Caso contrario intenta leerlos desde el dispositivo de *hardware*.

En la figura 4.6 se presenta el comportamiento general de los bloques de **sensores** mediante un diagrama de actividad. En los estados *UNCONFIGURED* e *INACTIVE* del ciclo de vida, los bloques de sensores han preparado toda la información requerida para su configuración. Los sensores esperan dos tipos de eventos para ejecutar las tareas de adquisición de mensajes:

- La solicitud para cambiar su estado interno a *ACTIVE*, al terminar la transición el bloque ya está preparado para ejecutar su funcionalidad.

- El siguiente mensaje a esperar es el estado general del exoesqueleto. Los valores que activan la funcionalidad son *READ*, *MOVE* o *STOPMOVEMENT*.

A partir de ese momento se activa un temporizador con un periodo de tiempo previamente configurado. Cada vez que se activa el temporizador, se leen los datos de los sensores cuando el modo de funcionamiento es de producción. Caso contrario, se generan datos que son simulados. Los datos generados son empaquetados en mensajes de tipo **SensorData** y transmitidos al resto del sistema.

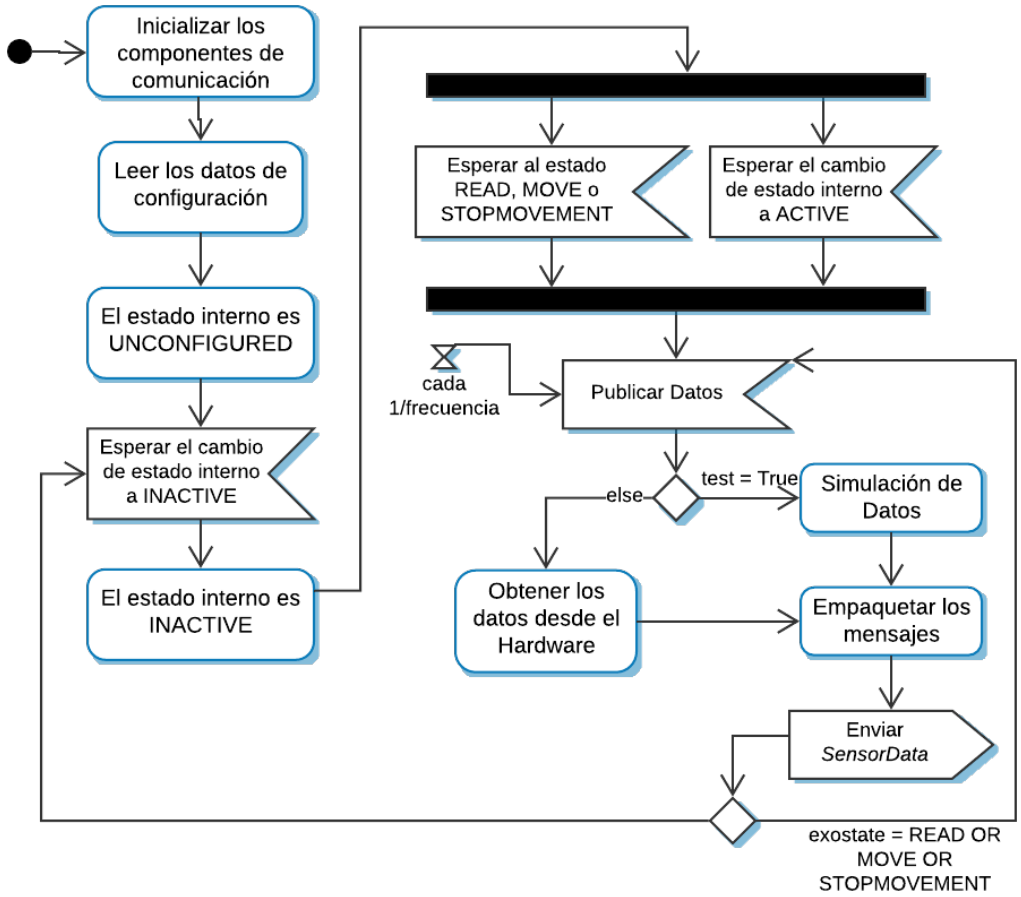


Figura 4.6: Diagrama de actividad del funcionamiento de los sensores.

4.3.4. Bloque del Sistema de Gestión de Energía

En la funcionalidad de este bloque participan dos elementos como son el bloque del *Sistema de Gestión de Energía* y el bloque de **Batería**. Estos dos componentes son diferentes, el primero se refiere al *hardware* encargado de proveer los requerimientos de energía eléctrica para el exoesqueleto y el segundo se refiere al componente de *software* que se encarga de la adquisición de datos, procesamiento

y difusión de los mismos en el sistema. En la figura 4.7 se presentan mediante diagramas internos de bloque los atributos configurables y puertos de comunicación de los componentes asociados a la gestión de energía. Los puertos se representan mediante bloques con flechas direccionales que indican si la información es de entrada o de salida de datos.

La parte de hardware se encarga del envío de los datos de sus sensores al

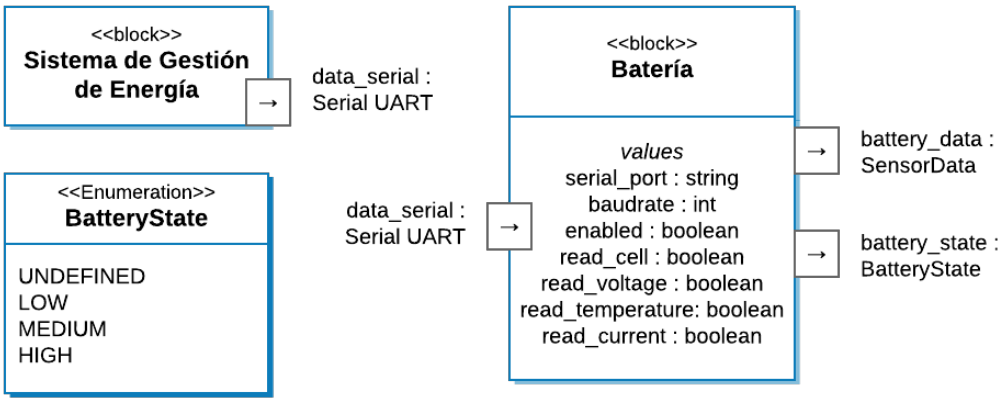


Figura 4.7: Diagrama de definición de bloques del Sistema de Gestión de Energía.

controlador centralizado mediante el puerto serial. El formato con el cual se envían los datos usa una representación separada por comas como se observa en la tabla 4.1.

Variable	Representación
Voltaje de las Celdas	C,d1,d2,d3,d4,d5,d6,d7,d8
Temperatura	T,d1,d2
Voltaje	V,d1
Corriente	I,d1

Tabla 4.1: Estructura de datos que envía el Sistema de Gestión de Energía.

El primer carácter de estos datos semiestructurados corresponde siempre a una letra que representa el tipo de variable, seguido de un conjunto de datos numéricos separados por comas. El voltaje de las celdas inicia con la letra "*C*" seguida por ocho valores numéricos que representan el voltaje individual de cada una de ellas. En el caso de la temperatura existe un valor numérico por cada uno de los dos termistores que han sido colocados cerca de la pila de baterías. De la misma forma existe información sobre el voltaje y corriente total que consume el *exoesqueleto*.

El bloque de **batería**, al recibir dichos datos, debe separarlos de acuerdo al tipo de variable y empaquetarlos en el tipo de dato **SensorData**. Por ejemplo, la estructura de datos tiene la siguiente información para el caso del voltaje de las celdas:

- *limb:* No aplica en este caso porque es independiente de la extremidad.
- *joint:* No aplica en este caso porque es independiente de la articulación.
- *node:* Toma el valor *battery* para indicar que la información proviene del bloque de **batería**.
- *variable:* Toma el valor *cell1* para indicar que la información proviene de la celda de energía número 1.
- *data:* Valor numérico correspondiente al voltaje de la celda.
- *timestamp:* Marca de tiempo en formato *epoch*.

Particularmente la información del voltaje total es de importancia para determinar el estado de carga del *Sistema de Gestión de Energía*. Su valor varía entre 28.7 V como máximo y 24 V como mínimo. Como se visualiza en la enumeración *BatteryState* de la figura 4.7. Se consideran cuatro estados:

- **UNDEFINED:** Es el estado inicial del módulo de batería cuando todavía no se ha recibido información sobre el voltaje total.
- **HIGH:** El voltaje total es superior o igual a 27 V.
- **MEDIUM:** El voltaje total es superior o igual a 25 V.
- **LOW:** El voltaje total es inferior a 25 V.

Cuando el bloque de batería ha detectado un cambio en el estado de carga, se envía un evento por el puerto de comunicación denominado *battery_state* para que el resto de bloques sean notificados de dicho cambio. Particularmente es útil para el fisioterapeuta quién es notificado a través del bloque de la interfaz gráfica **ControlGui**.

Los atributos que se especifican en el bloque de **batería** (figura 4.7) deben ser configurables. Esto significa que sus valores pueden cambiarse antes o durante su ejecución. Un parámetro de tipo lógico denominado *enabled* determina si el bloque debe ser ejecutado o no. Esta característica es útil durante las etapas de pruebas en donde no se requiere usar el módulo de batería. Otros valores lógicos importantes son *read_cell*, *read_temperature*, *read_voltage* y *read_current* que determinan si la información de esas variables debe ser publicada. Los parámetros modificables para este componente están definidos en un archivo de texto como el que se muestra en el anexo D.1.

La funcionalidad de los bloques para la gestión de la energía se define en el diagrama de actividades de la figura 4.8. El *Sistema de Gestión de Energía* una vez que ha iniciado su ejecución, publica los datos de sus sensores cada segundo. El bloque de **batería** es desplegado en el sistema y procede a crear un *socket* de comunicaciones en el puerto serial. Si el bloque de **batería** detecta que la conexión es exitosa, la primera acción que realiza es limpiar el *buffer* de datos del puerto.

De esta manera se elimina aquella información que no fue leída anteriormente y que posiblemente está desactualizada. A continuación, el bloque de **batería** inicia un proceso en el cual espera la escritura de datos en el puerto serial. Con el objetivo de evitar que el proceso de espera sea infinito, se reinicia la lectura de datos cada segundo. En el momento en que llega un dato, se debe detectar el tipo de variable que ha sido recibido. Para ello, se analiza su estructura de acuerdo al formato especificado en la tabla 4.1. Si el desarrollador autorizó la lectura y transmisión de dicho dato, se genera un mensaje **SensorData** que es enviado al resto del sistema. Particularmente es de interés la variable del voltaje porque a partir de ella se determina el estado de carga de la batería, misma que es notificada al resto del sistema.

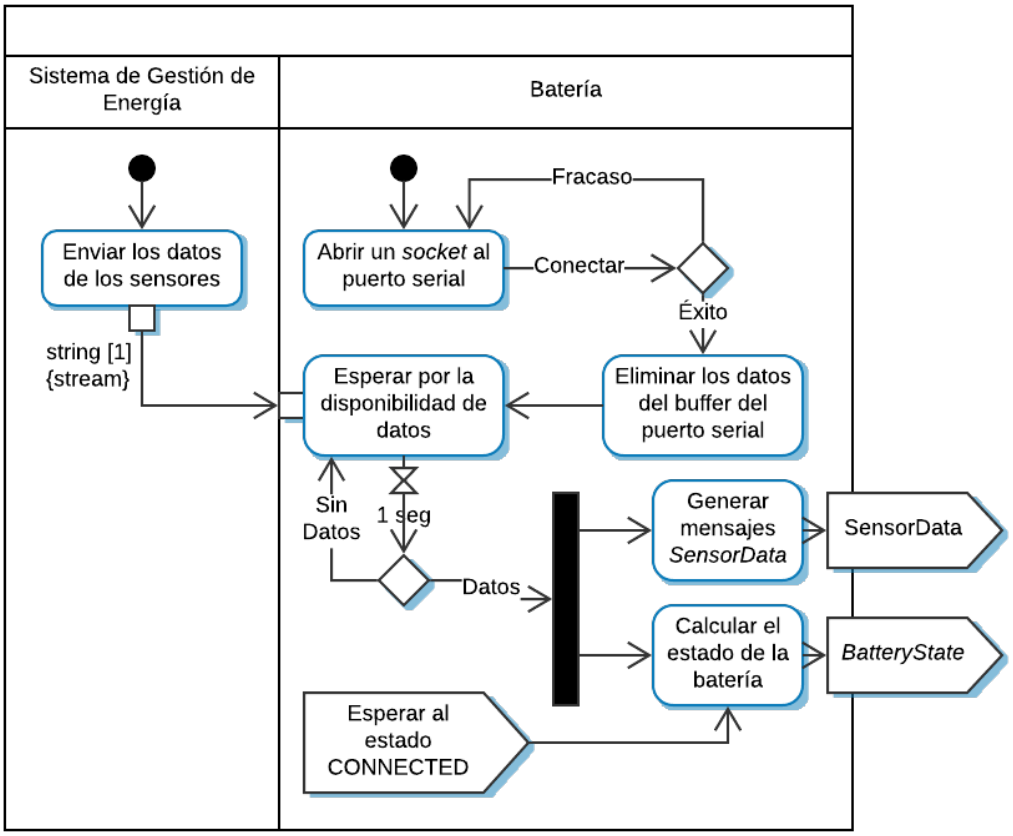


Figura 4.8: Diagrama de actividad del *Sistema de Gestión de Energía*.

4.3.5. Bloque de Comunicación

Es el bloque encargado de la interacción con *AllexApp*. Para facilitar la explicación del funcionamiento de este bloque se establece que el mecanismo de comunicación entre los dos subsistemas se realiza mediante el protocolo *bluetooth* en la versión 4.1. Este bloque está compuesto por dos partes funcionales como son:

el bloque de **transmisión** y el de **recepción** de datos. La estructura de estas partes se detalla en el diagrama de definición de bloques de la figura 4.9.

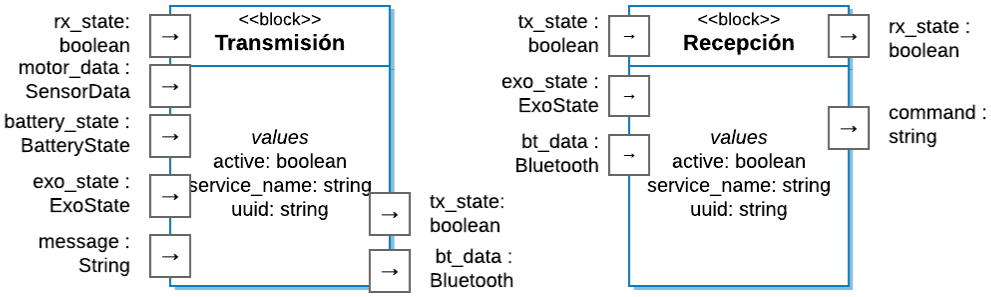


Figura 4.9: Diagrama de definición de bloques de comunicación.

4.3.5.1. Estructura del Bloque de Transmisión

El bloque de **transmisión** es el encargado de enviar datos y eventos desde *AllexControl* a *AllexApp*. Por ello en su definición de bloque tiene un puerto de salida de datos para el dispositivo *bluetooth*. Para la comunicación con otros componentes de *AllexControl* presenta puertos de entrada de datos como:

- *battery_state:* Para la información del estado de la batería.
- *motor_data:* Para la posición angular de los motores.
- *message:* Para mensajes de confirmación.

Esta información es enviada por *bluetooth* como un conjunto de *bytes* que representan a una cadena de caracteres separados por comas. Los diferentes tipos de mensajes enviados hacia *AllexApp* se resumen en la tabla 4.2.

Mensaje	Representación
Datos de motores	d,limb,joint,value,timestampsec,nanosec
Confirmación	ack,exostate
Error	e,codigo
Batería	b,batterystate

Tabla 4.2: Mensajes transmitidos hacia *AllexControl*.

En el caso de los datos de los motores, el mensaje tiene la siguiente estructura:

- El mensaje inicia con la letra *d*, que significa *"data"*.
- Los campos *limb* y *joint* corresponden a los valores de las enumeraciones

LimbType y **JointType** respectivamente y le sirven a *AllexApp* para conocer en cuál gráfica temporal dibujar el punto.
- En *value* está el valor cuantitativo de la posición angular.
- Los campos *timestampsec* y *nanosec* son la marca de tiempo en la cual se leyó el dato en formato *epoch*. Esto le permite a *AllexApp* dibujar al punto en la gráfica de posición angular respecto al tiempo.

Para la información del estado de carga de la batería únicamente se envía una cadena de texto que inicia con la letra "*b*" y está seguida por el valor del estado de la batería usando la enumeración *BatteryState*. Los mensajes de confirmación son aquellos que se envían a *AllexApp* cuando el bloque **orquestador** autoriza un comando que el usuario ha solicitado. Por ejemplo, si el comando que se solicita es *READ* (con valor numérico 1 dentro de la enumeración), la respuesta transmitida será "*ack,1*" con lo que se confirma la ejecución del comando a la aplicación.

4.3.5.2. Estructura del Bloque de Recepción

El bloque de **recepción** cumple únicamente una función que es la de recibir los comandos del tipo de dato *ExoCmd* desde *AllexApp* y transmitirlos al bloque **orquestador** para que los valide y ejecute. Estos mensajes tienen la estructura que se muestra en la tabla 4.3. Todos los mensajes recibidos tienen una estructura de datos separada por comas en donde el primer valor corresponde a la abreviación "*cmd*" relacionada a la palabra comando.

Comando	Representación
STOP	cmd,0
READ	cmd,1
MOVE	cmd,2, exercise, iterations, hiking_time, emg_active, eeg_active

Tabla 4.3: Mensajes transmitidos desde *AllexControl*.

En el caso de los comandos para leer (*READ*) o detener la actividad actual (*STOP*) se recibe solamente un texto que, después de los caracteres *cmd*, posee el identificador del comando de acuerdo a la enumeración *ExoCmd*. Por otro lado, en el caso del mensaje *MOVE* se reciben otros parámetros que son de utilidad para la configuración del movimiento. Estos se resumen en un nuevo tipo de dato denominado **ConfigRequest** definido en la figura 4.10 y consta de:

- **exercise:** Identificador del ejercicio a ser ejecutado.
- **iteration:** Número de veces que un ciclo de ejercicios tiene que ser ejecutado.
- **hiking_time:** Duración del ciclo de ejercicios.
- **emg_active:** Habilita el uso de la intención de movimiento usando señales EMG.
- **eeg_active:** Habilita el uso de la intención de movimiento usando señales EEG.

Figura 4.10: Tipo de dato para la solicitud de configuración.

4.3.5.3. Administración de la Conexión

Como se indicó en la definición de la máquina de estados general (sección 4.3.1), los bloques de conexión inicializan un servidor *bluetooth* cuando todos los bloques han sido configurados y el **orquestador** ha notificado a todo el sistema que el estado actual es *CONFIGURED*. A partir de ese momento los dos bloques de **comunicación** esperan una solicitud de conexión desde un cliente *AllexApp*. Si la conexión fue exitosa, los dos bloques envían eventos por sus puertos de flujo de datos *rx_state* o *tx_state*. De esta manera se informa al **orquestador** que la conexión se ha establecido. En ese momento, el **orquestador** determina que *AllexControl* pasa al estado *CONNECTED* e informa dicho evento a todos los bloques. Si la conexión con el bloque **transmisor** o **receptor** falla, se regresa al estado *CONFIGURED* y se cierran todas las conexiones. Este comportamiento se visualiza en el diagrama de actividad de la figura 4.11. Si todas las conexiones son exitosas, el bloque de **recepción** recibe comandos y el de **transmisión** envía datos.

4.3.6. Bloques de Detección de Intención de Movimiento

En este grupo están los bloques de detección de intención de movimiento por señales EMG y EEG. Cada uno de estos bloques requieren una de las señales biológicas para ejecutar un algoritmo cuya salida determina si se ha detectado o no la intención de movimiento del paciente. No se implementarán estos algoritmos como parte del alcance del presente proyecto (sección 3.2.6); sin embargo, se simularán los eventos de detección de intención de movimiento. La estructura y funcionalidad a ser presentada a continuación corresponde a la detección de intención de movimiento por EMG, pero la misma descripción es válida para el caso de las señales EEG. Esto debido a que los algoritmos se modelan como un sistema de caja negra a ser implementados o especificados en trabajos futuros.

El bloque **EMG** (figura 4.12) tiene una funcionalidad y estructura que es heredada del bloque **Sensor** (sección 4.3.3). Como parte de los datos que genera en su puerto de salida está *emg_data* para el envío de datos de señales EEG de tipo **SensorData** y *error* para notificar al **orquestador** que ha existido un error en

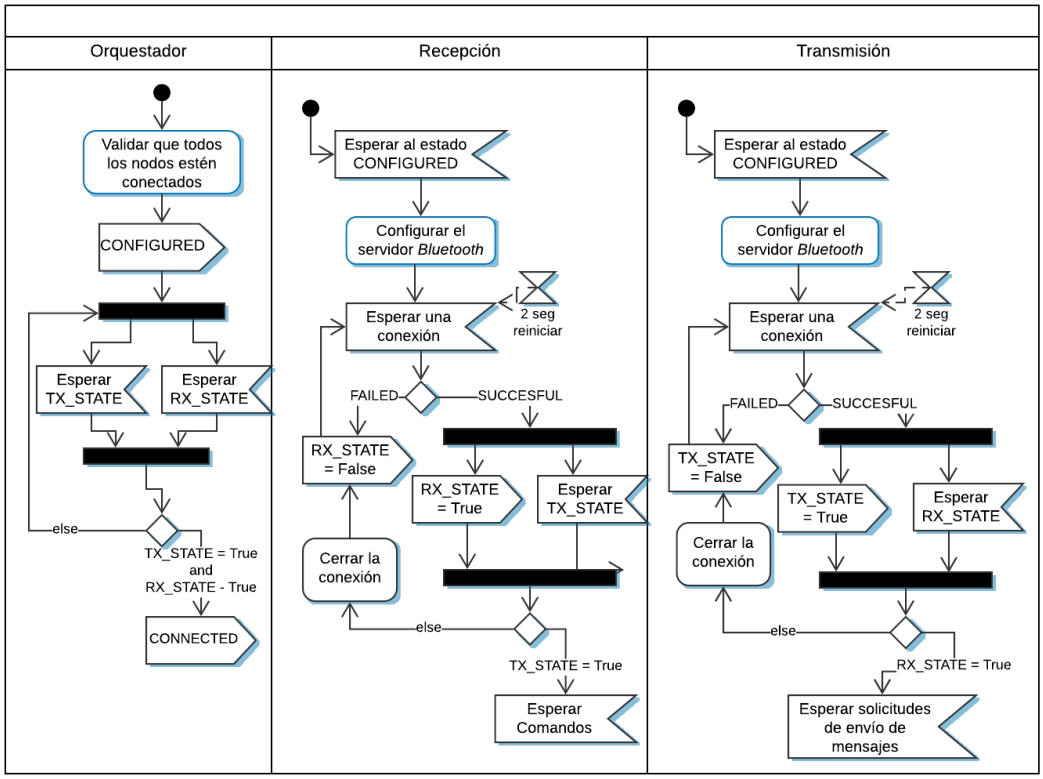


Figura 4.11: Manejo de la conexión bluetooth y cambio de estado a CONNECTED.

el bloque. Los paquetes **SensorData** son transmitidos al puerto de entrada del bloque **Intención EMG**. Esta información es usada para ejecutar un algoritmo de detección de intención de movimiento. Si el algoritmo detecta la intención, se envía un mensaje de tipo lógico mediante el puerto (*IntentionEMG*). Por otro lado, el puerto de entrada *intention_test* es usado únicamente cuando el bloque está configurado dentro del ambiente de prueba mediante la variable *test*. Cuando recibe un evento por ese puerto, automáticamente genera un mensaje de intención de movimiento que es enviado al bloque planificador.

La funcionalidad de este bloque está representada mediante el diagrama de

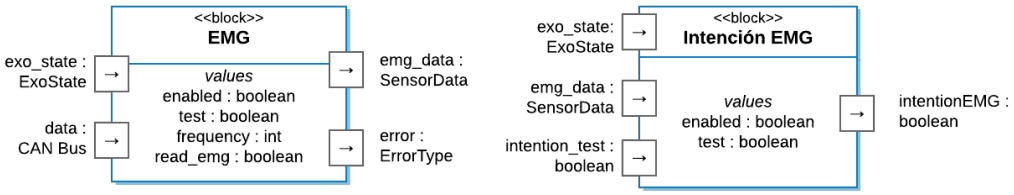


Figura 4.12: Diagrama de definición de bloques del bloque EMG y de Intención EMG.

actividades de la figura 4.13. En dicho caso de uso, el bloque **EMG** se encuentra en estado interno *ACTIVE* y publica un flujo de mensajes del tipo **SensorData**

con señales EMG. Por otro lado el bloque **Intención EMG** inicia sus actividades al configurar sus puertos de comunicación y espera que el estado global del exoesqueleto cambie a *MOVE*. Una vez que se notifica que el exoesqueleto está en dicho estado, el bloque acepta los eventos para mensajes de EMG. Si el bloque no está configurado en estado de prueba, entonces ejecuta su algoritmo de detección de intención. Si la intención ha sido detectada se genera un evento *IntencionEMG*. El caso alternativo es aquel en el cual el bloque está en estado de prueba y debe esperar un evento en el puerto *intention_test* para generar la intención de movimiento. Este evento se envía al presionar un botón en un programa externo a *AllexControl*.

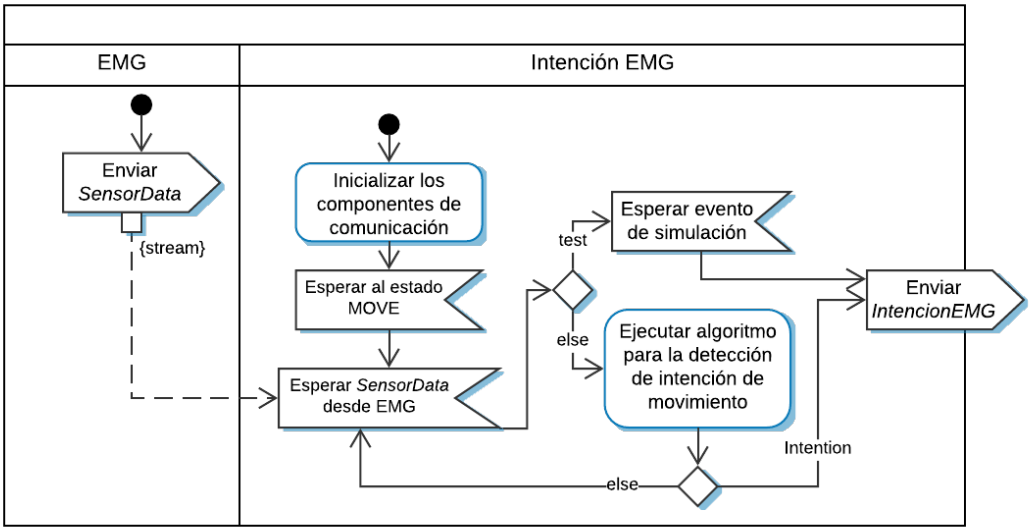


Figura 4.13: Diagrama de actividades del bloque de detección de intención de movimiento para electromiografía.

4.3.7. Bloques de Extremidades

El bloque de **extremidad** es el encargado de enviar solicitudes de modificación al estado interno de los sensores y actuadores asociados a los motores, señales EMG y EEG. Adicionalmente debe administrar un puerto de *software* que le permite enviar comandos y recibir respuestas del *hardware EPOS4 50/8* para el control por posición. Al igual que el resto de bloques, las extremidades también tienen un conjunto de parámetros configurables como se presenta en el diagrama de definición del bloque de la figura 4.14. Existen cuatro parámetros que están asociados a la configuración de la comunicación con **EPOS4** como son:

- *device_name:* nombre de los controladores conectados, en este caso *EPOS4*.
- *protocol_stack_name:* especifica el protocolo de comunicación usado, en este caso está configurado con *CANOpen* para la interacción mediante el bus CAN.

- *interface_name:* Nombre de la interfaz de *hardware* a través de la cual se realiza la conexión.
- *port_name:* Nombre del puerto de entrada/salida a ser usado en este caso CAN0.

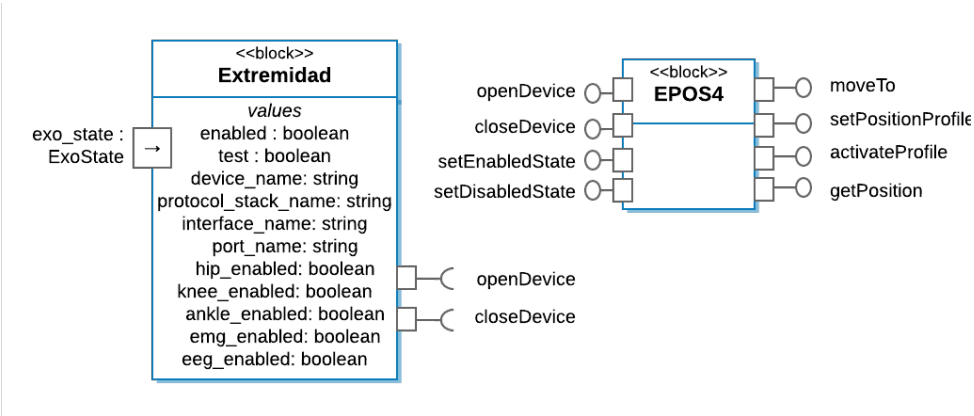


Figura 4.14: Diagrama de definición de bloques de Extremidades y EPOS4.

Otros parámetros de configuración, como aquellos que tienen el sufijo "*_enabled*", son usados para determinar si los bloques que son parte de la extremidad (como por ejemplo los actuadores) deben ser ejecutados o no. Esto es útil en las etapas de desarrollo o cuando el *hardware* desactivado está en mantenimiento. El bloque de **extremidad** también puede ser desactivado mediante el parámetro *enabled*. Esta propiedad se usa en la pierna derecha, la cual estará inactiva hasta que el *hardware* sea implementado.

El bloque **EPOS4** se modela como un conjunto de servicios con interfaces provistas y definidas por su librería de comandos [92]. Este bloque es usado por la **Extremidad** para crear los puertos de comunicación y enviar comandos. Para conseguir este objetivo son de interés las siguientes interfaces:

- *openDevice:* Interfaz mediante la cual se crea un puerto de comunicación con los controladores **EPOS4**. Para ello se requiere especificar los parámetros *device_name, protocol_stack_name, interface_name, port_name*. Si no existieron errores se obtiene un manejador para la conexión.
- *closeDevice:* Se libera la memoria del manejador para la conexión.
- *setEnabledState:* Cambia el estado del dispositivo **EPOS4** a *ENABLE*, en el cual recibe comandos de movimiento o adquiere los parámetros de posición de los motores. Se requiere que cada articulación tenga un identificador para que actúe el dispositivo adecuado.
- *setDisabledState:* Cambia el estado del dispositivo **EPOS4** a *DISABLE*. También requiere del identificador de la articulación.

El proceso de configuración de la conexión con **EPOS4** se realiza como se representa en el diagrama de actividades de la figura 4.15. Se considera el caso de una sola extremidad, aunque esta funcionalidad es aplicable también para la segunda.

En primer, lugar el bloque **orquestador** decide que el estado del exoesqueleto debe pasar de *UNCONFIGURED* a *CONFIGURED*. Para conseguir este objetivo, se envía una solicitud al bloque **Extremidad** para que cambie su estado interno a INACTIVE. Esto significa que este bloque deberá inicializar la funcionalidad de los sensores y actuadores que han sido habilitados en el archivo de configuración (descrito en el anexo D.4) y abrir un puerto de conexión con **EPOS4**. Para ello, el bloque de **extremidad** usa el servicio *openDevice* para crear el puerto. En este momento pueden suceder uno de dos eventos:

- La conexión ha fallado y se notifica al nodo orquestador acerca del fallo. En este caso se cancela el cambio de estado y el orquestador solicita que el bloque de **extremidad** reintente la transición.
- Se ha creado la conexión, por tanto, es seguro solicitar el estado *INACTIVE* de los otros actuadores y sensores de la extremidad. Si este cambio falla, también se notifica al orquestador. Caso contrario el cambio de estado es exitoso y el **orquestador** continúa con su rutina de solicitudes de activación a otros bloques.

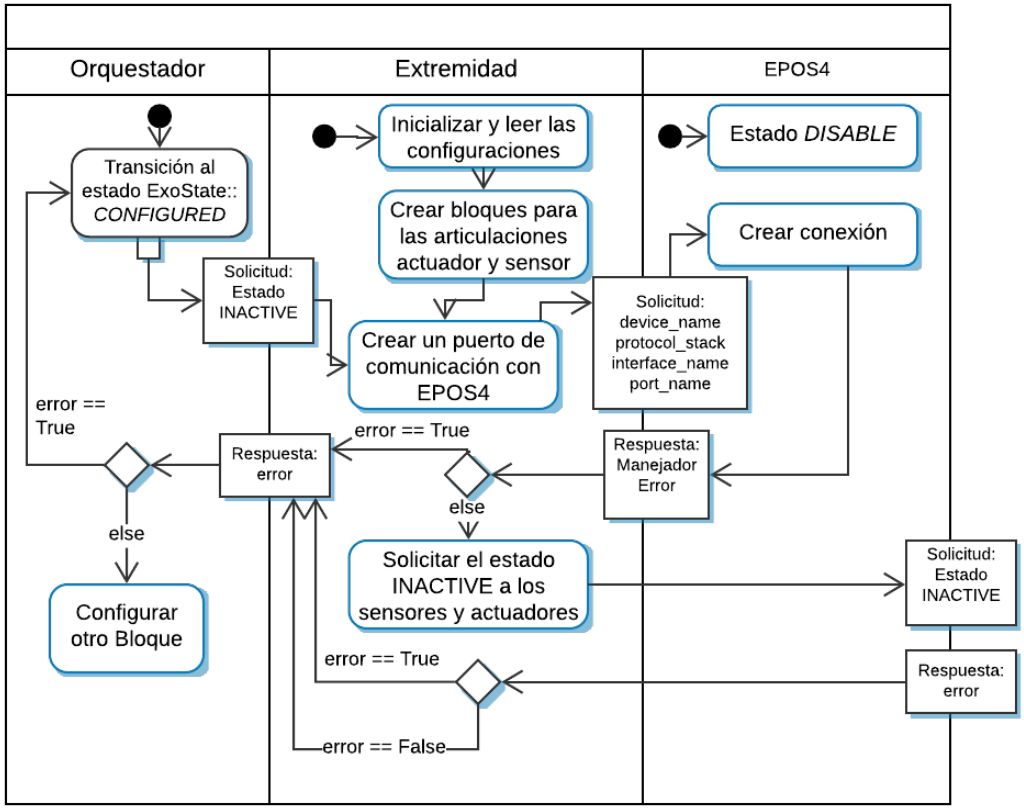


Figura 4.15: Diagrama de actividades de la creación del puerto de conexión con EPOS4.

Un segundo caso de uso para este bloque es cuando el exoesqueleto cambia al estado general *MOVE* para ejecutar un ejercicio. En esta situación el bloque de

extremidad usa el servicio *setEnabledState* para activar los controladores de cada articulación y que de esa manera puedan cambiar la posición de los motores. De forma similar, cuando termina el movimiento se desactivan mediante el servicio *setDisabledState*.

4.3.8. Bloques de los Actuadores

Son tres bloques que corresponden a los motores de la cadera, tobillo y rodilla. Los tres poseen una estructura similar, pero difieren en la trayectoria que cada uno ejecuta.

Configuración de la Trayectoria

La definición de la trayectoria de cada articulación es administrada mediante un archivo de configuración en formato *yaml* como el que se presenta en el anexo D.5. En el primer nivel de dicho archivo está el nombre del movimiento, por ejemplo *gait* en el caso de la caminata. El nombre del movimiento tiene como propiedad un identificador que es usado por *AllexApp* para solicitar la trayectoria a ser ejecutada. En el siguiente nivel están seis propiedades, una por cada articulación de ambas extremidades. Por ejemplo, *l_knee*, para la rodilla izquierda. Dentro de cada una de estas propiedades están definidos tres tipos de trayectorias denominadas:

- *initial_movement:* Es el movimiento que realiza el exoesqueleto desde una posición inicial hasta una postura que le permita iniciar con los movimientos cíclicos de rehabilitación.
- *cyclic_movement:* Son movimientos repetitivos que realizan las articulaciones del exoesqueleto. Estos movimientos se realizan n veces dependiendo de la configuración del fisioterapeuta.
- *stop_movement:* Es el movimiento que realiza el exoesqueleto desde la última posición del movimiento cíclico hasta una posición de reposo.

Dentro de cada tipo de movimiento se especifican dos listas, una denominada *target_position* y otra *cycle_percent*. Para explicar su uso, se toma por ejemplo la lista de posiciones para el movimiento cíclico de la rodilla izquierda cuyos valores son:

$$\texttt{target\_position} = [30, -15, 35, 30] \tag{4.1}$$

$$\texttt{cycle\_position} = [0, 0.52, 0.87, 1] \tag{4.2}$$

Los valores de estas listas constituyen un conjunto de puntos de la forma P(*cycle_percent, target_position*) que están representados en la gráfica de la trayectoria para la rodilla izquierda de la figura 4.16. De acuerdo a esta gráfica, los valores de la lista 4.1 representan la posición angular máxima o mínima que tiene la articulación al iniciar o al finalizar los intervalos de tiempo que están declarados en la lista 4.2. Los intervalos de tiempo se representan como un porcentaje de la totalidad del ciclo de movimiento. Esta representación es de importancia debido a

que permite ajustar la duración de la trayectoria en función del tiempo configurado por el usuario.

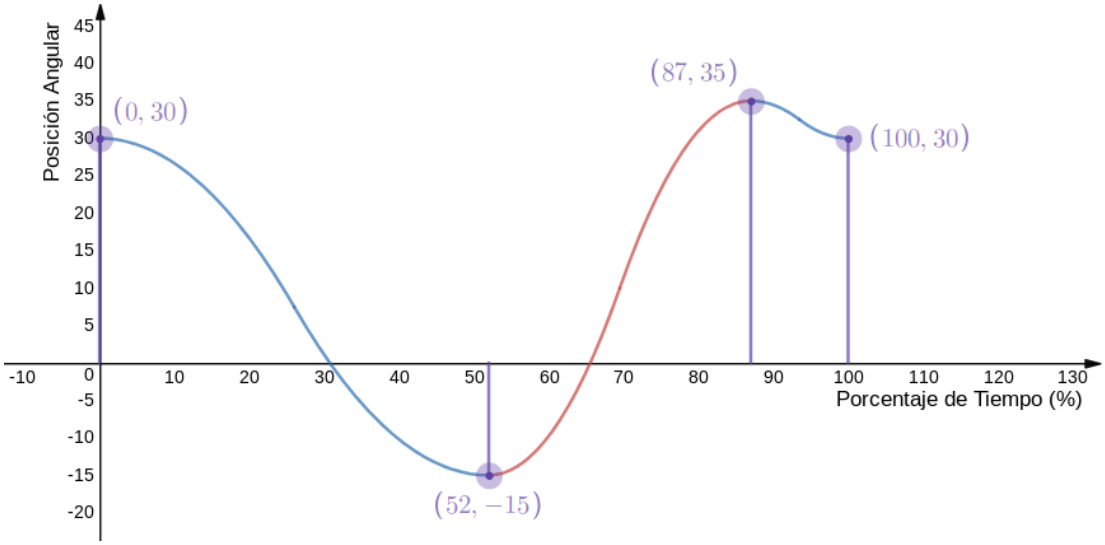


Figura 4.16: Trayectoria angular de la rodilla durante un ciclo de caminata.

En el trabajo realizado por Blandín y Velasco en [17], determinaron que si se realiza una interpolación parabólica entre cada intervalo de puntos era posible ejecutar una trayectoria continua sin transiciones bruscas, lo cual es ideal para movimientos que serán efectuados sobre seres humanos. Para ello plantean calcular el tiempo de ejecución, posiciones, velocidad y aceleración de cada intervalo mediante las funciones que se presentan a continuación:

$$P_n = \texttt{target\_position}[n] \cdot \texttt{joint\_increase} \tag{4.3}$$

$$T_n = \texttt{hiking\_time} \cdot (\texttt{cycle\_position}[n] - \texttt{cycle\_position}[n-1]) \tag{4.4}$$

$$v_n = \frac{2 \cdot 0.0145 \cdot (P_n - P_{n-1})}{T_n} \tag{4.5}$$

$$a_n = \frac{2 \cdot v_n}{T_n} \tag{4.6}$$

En donde:

P_n	=	Ángulo deseado en la articulación con el respectivo incremento debido a los reductores colocados en los motores.
$joint\_increase$	=	Incremento requerido en cada articulación. Su valor es 487 en el caso de la rodilla y el tobillo y 2230 para la cadera.
$hiking\_time$	=	Tiempo total del ciclo de movimiento.
T_n	=	Tiempo requerido para ejecutar la trayectoria del intervalo n.
v_n	=	Velocidad del motor en el intervalo n.
a_n	=	Aceleración del motor en el intervalo n.

El proceso de configuración del movimiento inicia cuando el orquestador envía una solicitud de configuración al actuador mediante el tipo de dato **ConfigRequest** (figura 4.10). Los actuadores únicamente requieren los campos *exercise* y *hiking_time* de dicho mensaje. El primer valor corresponde al identificador del movimiento que se debe ejecutar, el cual está asociado al campo *id* de los archivos de configuración de movimiento. El segundo valor indica el tiempo en el cual cada ciclo de movimiento debe completarse como se describe en la función 4.4. El cálculo de velocidad y aceleración de cada intervalo se realiza antes de ejecutar toda la rutina de ejercicios. Por tanto, el orquestador debe esperar una respuesta de cada bloque de **articulación** para conocer si la configuración fue exitosa antes de cambiar el estado del exoesqueleto a *MOVE*. Primero, la articulación valida que el ejercicio exista en el archivo de configuración y que el tiempo de cada ciclo no sea inferior a cuatro segundos para evitar que el motor de la cadera exceda el número máximo de revoluciones permitidas. Sin embargo, estos parámetros también son validados desde *AllexApp* que es originalmente el sistema que envía la solicitud.

Si no se han presentado errores, el motor ejecuta sus movimientos de rotación bajo la coordinación del bloque de **Planificación** que será presentado en la sección 4.3.9. Por el momento se resumen aquellas interfaces del bloque **EPOS4** que son usadas por los *Actuadores* como son:

- *activateProfile:* Activa el perfil por posición, esto significa que la rotación se realiza al desplazar el motor entre dos posiciones angulares absolutas.
- *setPositionProfile:* Configura el modo de operación mediante la velocidad, aceleración y desaceleración, dando como resultado el control de la trayectoria. Al ser una transición simétrica, el valor de la aceleración es igual al de la desaceleración usando la función 4.6.
- *moveTo:* Especifica la posición angular a la cual se requiere que el motor se desplace.

4.3.9. Bloques de Planificación

Es el encargado de coordinar el comportamiento de los motores y de usar las señales de intención de movimiento para ejecutar un ejercicio de rehabilitación. Su funcionamiento se basa en la máquina de estados representada en la figura 4.17. Esta funcionalidad depende de los tres tipos de trayectorias que fueron especificados para cada articulación en el archivo de configuración D.5, es decir, el movimiento inicial, cíclico y final. El proceso inicia cuando el bloque **orquestador** envía al bloque de **planificación** el mensaje **ConfigRequest**. De esta estructura de datos se obtiene el parámetro *iteration* que indica el número de veces que el movimiento cíclico debe ser repetido. Por otro lado, los parámetros *emg_enabled* o *eeg_enabled* determinan si el planificador debe esperar por los eventos de EMG o EEG respectivamente. Existen dos escenarios para el funcionamiento de la máquina de estados, uno en el que no se requiere esperar por los mensajes de intención de movimiento y otro en el cual son estos eventos los que inician cada ciclo de movimiento.

En el primer caso cuando las variables *emg_active* y *eeg_active* tienen un valor

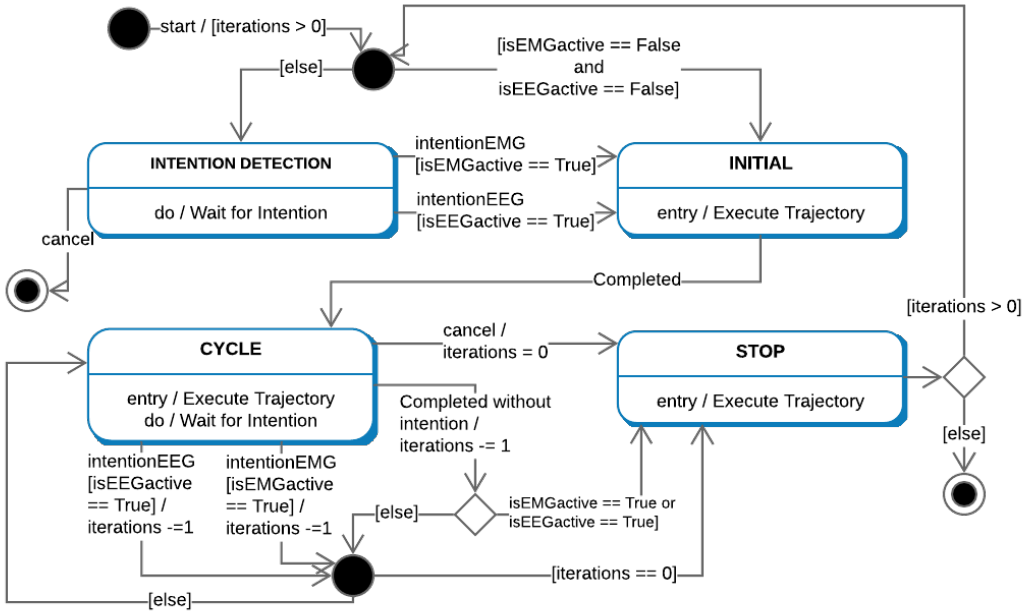


Figura 4.17: Máquina de estados para la planificación de movimientos.

lógico falso, la máquina de estados ignora las señales de intención de movimiento e ingresa directamente al estado *INITIAL*. En ese momento se envía un mensaje a todos los bloques **Actuadores** para que ejecuten la trayectoria *initial_movement* previamente configurada. Cada vez que el actuador termina de ejecutar la trayectoria, envía un evento denominado *movement_finished* al bloque de planificación. Cuando el planificador ha recibido la notificación *movement_finished* de todos los actuadores se considera que el movimiento total ha finalizado. En ese momento inicia la transición al siguiente estado, denominado *CYCLE*, en el cual se ejecutan las trayectorias cíclicas (*cyclic_movement*). El mecanismo de interacción usado es el mismo que en el movimiento inicial, se envía un comando a todos los actuadores para que ejecuten el movimiento cíclico y los actuadores notifican de la finalización mediante el evento *move_finished*. El bloque reduce el contador de iteraciones en uno y posteriormente evalúa si el número de iteraciones restantes es mayor a cero. En caso afirmativo, ejecuta nuevamente la funcionalidad del estado *CYCLE*. Si ya se han completado todas las iteraciones configuradas, inicia la transición hacia el estado *STOP*. En este último estado, todos los actuadores ejecutan el movimiento de finalización *stop_movement*. Cuando el movimiento ha sido completado finaliza la máquina de estados y se notifica al bloque **orquestador** que el movimiento ha terminado. De esta manera el exoesqueleto retorna al estado global *CONNECTED* para esperar nuevos comandos. Una alternativa de este caso de uso es cuando el usuario cancela la ejecución del movimiento antes de que termine todas las iteraciones configuradas. En esta situación todos los actuadores

deben terminar la trayectoria del estado *CYCLE* que están ejecutando e inician la transición directamente hacia el estado *STOP*. El número de iteraciones también se reduce a cero. Como resultado el exoesqueleto regresa al estado inicial de reposo.

El segundo escenario se genera cuando el usuario ha configurado el uso de la intención de movimiento para ejecutar los ejercicios de rehabilitación desde *AllexApp*. En ese caso el primer estado al que ingresa el exoesqueleto es *INTENTION DETECTION*. Este es un estado de espera para la llegada de un evento de intención de movimiento. Si el evento ha sido detectado, el exoesqueleto ejecuta el estado *INITIAL*. Una alternativa dentro del estado *INTENTION DETECTION* es que el usuario cancele la ejecución del movimiento, en cuyo caso, termina directamente la funcionalidad de la máquina de estados debido a que no se ha ejecutado ningún movimiento. Si se ha ejecutado el estado *INITIAL*, inicia una transición hacia el estado *CYCLE*. Este estado, aparte de ejecutar la rutina de movimiento cíclico, también espera la llegada de señales de intención de movimiento. Si al terminar la ejecución de la trayectoria ha existido alguna señal de intención de movimiento se evalúa si se deben ejecutar más iteraciones y se repite nuevamente el estado *CYCLE*. Caso contrario, inicia la transición hacia el estado *STOP* para que el exoesqueleto regrese a una posición de reposo. Si todavía no se han ejecutado todas las iteraciones configuradas y no se ha cancelado el movimiento, la máquina de estados regresa a *INTENTION DETECTION* para repetir todo el proceso hasta que el número de iteraciones sea cero.

4.3.10. Bloque de Orquestación

El bloque de **orquestación** es el encargado de responder ante los comandos de tipo *ExoCmd* enviados desde *AllexApp*. Por tanto, decide y notifica sobre los cambios en el estado global del exoesqueleto (*ExoState*). Su estructura de bloques está conformada por los siguientes puertos de entrada:

- *tx_state:* Para recibir mensajes de conexión o desconexión del bloque de **transmisión** de datos.
- *rx_state:* Para recibir mensajes de conexión o de desconexión del bloque de **recepción** de datos.
- *error:* Para recibir mensajes de error de algún bloque y responder mediante un cambio de estado.
- *control:* Es usado para recibir mensajes con comandos que provienen desde *AllexApp* y que usan al bloque de **recepción** como un punto de conexión intermedio.

Por otro lado, los puertos de salida están asociados al envío de eventos como:

- *exo_state:* Para notificar los cambios del estado general del exoesqueleto al resto de los bloques del sistema.
- *message:* Para enviar los mensajes de confirmación y error descritos en la tabla 4.2.

Adicionalmente tiene tres tipos de interfaces requeridas para comunicación de tipo cliente - servidor. Una para solicitar el estado interno de un bloque al que se lo conoce como *get_state*, otro para solicitar el cambio del estado interno de un bloque *change_state* y finalmente una interfaz para configurar los bloques de **actuadores** y de planificación antes de la ejecución de un movimiento.

Como se indicó en la sección 4.3.1, el primer estado del exoesqueleto es *UNCONFIGURED*. La primera tarea del orquestador es leer los parámetros de configuración *enabled* de los bloques con el objetivo de conocer cuáles son los que deben estar activos y de esa manera únicamente interactuar con ellos. Esta funcionalidad es importante al momento de realizar pruebas con un conjunto previamente determinado de bloques. Después, se crean los puertos de comunicación y servicios antes de iniciar la transición del exoesqueleto al estado *CONFIGURED*.

En la transición de configuración, el bloque **orquestador** solicita a las articulaciones de las extremidades que cambien su estado interno a *INACTIVE* y que se conecten con el dispositivo **EPOS4** siguiendo la secuencia presentada en la figura 4.15. Si la configuración falla, se cancela la transición actual y se reintenta un segundo después. Si las extremidades han sido configuradas, se genera un evento para informar a todos los bloques sobre el cambio de estado. Este evento provoca que los bloques de **trasmisión** y **recepción** de datos activen sus servicios *bluetooth* para receptar conexiones. Si la conexión ha sido exitosa, los bloques de **trasmisión** y **recepción** envían un mensaje de confirmación al orquestador. De esta manera se notifica que el estado del exoesqueleto es *CONNECTED* a todos los bloques. En este estado el orquestador acepta los comandos de configuración descritos en la tabla 4.3 y que son enviados desde *AllexApp*.

El escenario más sencillo es aquel en el cual llega un mensaje con un comando *READ* para iniciar el proceso de lectura desde el bloque de **recepción** de la comunicación. Lo primero que realiza el orquestador es verificar que el comando sea válido dentro de la máquina de estados, es decir, que el estado actual sea *CONNECTED*. Como parte de la transición, el orquestador solicita a los bloques de **Extremidad** que pasen de estado *INACTIVE* a *ACTIVE*. Cada extremidad se asegura del cambio de estado de los bloques de su composición como los sensores de los motores, actuadores, **EMG** y **EEG**. En ese estado los bloques están preparados para ejecutar su funcionalidad, pero se encuentran en espera hasta que la transición termine. Caso contrario, algunos bloques iniciarían su actividad antes que otros y no existiría una garantía de que la transición de todos los bloques ha sido exitosa. Cuando el orquestador verifica que todos los bloques están en estado *ACTIVE*, emite un evento de cambio del estado del exoesqueleto a *READ* con lo cual inicia la lectura y transmisión de los datos. Adicionalmente se envía a *AllexApp* una confirmación de que el comando ha sido aceptado.

La única transición a partir del estado *READ* consiste en regresar al estado *CONNECTED*. Esto sucede cuando el usuario de *AllexApp* envía un comando

de tipo *STOP*. En este caso el orquestador cambia el estado del exoesqueleto inmediatamente a *CONNECTED* y después solicita la transición de los estados internos de cada bloque de *ACTIVE* a *INACTIVE*. Sin embargo, existe un caso alternativo en el que se cancela el estado *READ* cuando se produce una desconexión. El orquestador ha sido notificado de que *AllexApp* se ha desconectado ya sea de manera manual o por un error (por ejemplo, cuando la aplicación ha sido cerrada). Ante esa situación, se ejecuta la misma transición declarada anteriormente pero el estado final es *CONFIGURED*. Como resultado, los bloques de **transmisión** y **recepción** de datos por *bluetooth* configuran nuevamente sus servicios y esperan otro intento de conexión.

El segundo escenario corresponde a la ejecución del movimiento de rehabilitación cuando se ha recibido el comando *MOVE* con las correspondientes variables de configuración del movimiento. Después de validar la transición, se realiza una llamada al servicio para configurar el bloque de **planificación** con lo cual se valida que el número de iteraciones sea mayor a cuatro. A continuación, se realizan las llamadas a los servicios de configuración de cada *Actuador*, ahí se valida el tipo de ejercicio. Si todos los servicios fueron configurados de manera exitosa, el orquestador solicita a los bloques que cambien el estado interno *INACTIVE* a *ACTIVE*. Si todas estas transiciones no generan errores, el orquestador notifica a todos los bloques que el estado general del exoesqueleto es *MOVE* e inicia la rutina de ejecución del movimiento.

El estado *MOVE* puede terminar por tres motivos:

- El primero ocurre cuando se han completado todas las iteraciones configuradas para el movimiento y el exoesqueleto se encuentra en una postura final de reposo. El orquestador recibe el evento de finalización de movimiento, cambia el estado general del exoesqueleto a *CONNECTED* y el estado interno de cada bloque a *INACTIVE*.
- El usuario de *AllexApp* envía un comando *STOP* durante la ejecución del movimiento. El orquestador valida el evento y cambia el estado general del exoesqueleto a *STOPMOVEMENT*. El bloque de **planificación** al recibir el evento ejecuta la secuencia para cancelar el movimiento como fue descrito en la sección 4.3.9. Al final el orquestador recibe el informe de finalización del movimiento y regresa al estado *CONNECTED*.
- Los bloques de **comunicación** han detectado que la conexión ha terminado. En ese caso se asume que el movimiento ha sido cancelado para evitar errores que puedan producirse por falta de control. En ese caso se repite el proceso enunciado en el escenario anterior, pero el exoesqueleto termina en el estado *CONFIGURED* y en espera de nuevas conexiones.

4.4. Interacción entre *AllexApp* y *AllexControl*

En esta sección se describen los bloques de *AllexApp* que están encargados de la comunicación con *AllexControl*. En esta interacción participan los bloques de **Comunicación** para la transmisión de comandos y la recepción de datos, así como las interfaces gráficas asociadas al control del exoesqueleto.

4.4.1. Bloque de Comunicación

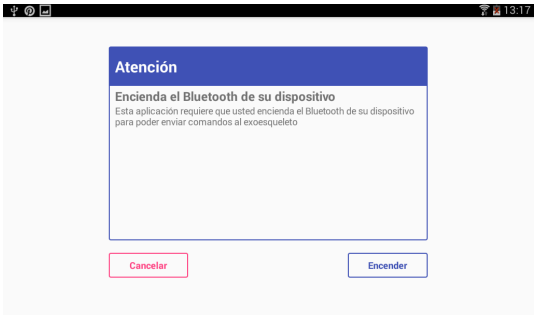
Es el componente que se encarga de administrar las cuatro tareas básicas de la pila de red de *bluetooth* como son: la configuración del adaptador local, la búsqueda de dispositivos que están sincronizados, la conexión con dichos dispositivos y el intercambio de datos. Esto lo consigue mediante los componentes de **transmisión** y **recepción** de datos, mismos que actúan como clientes que inician la conexión con *AllexControl*. Para explicar su comportamiento se usarán conceptos para la administración de un adaptador de *bluetooth* en *software*.

El primer paso consiste en verificar si la comunicación por *bluetooth* está disponible y puede ser usada en el dispositivo móvil. Para cumplir con este requerimiento se debe crear un adaptador para administrar las funciones de comunicación. Si la creación del adaptador ha fallado, eso significa que no es posible usar las funcionalidades de la aplicación que están asociadas al control del exoesqueleto con el celular o tableta que ha sido seleccionada.

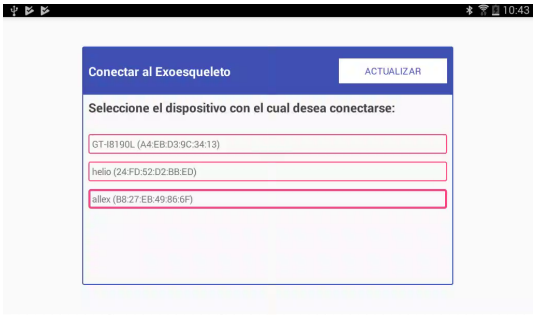
Si el adaptador ha sido creado exitosamente, se debe verificar si la funcionalidad del *bluetooth* está encendida antes de iniciar el proceso de conexión. Si el adaptador está inactivo, se carga la pantalla para activar el *bluetooth* como se muestra en la figura 4.18a. En esta pantalla se solicita al usuario que le otorgue a *AllexApp* los permisos para administrar y encender el *bluetooth*. Cuando el adaptador pasa a estado activo, se carga la pantalla de conexión que se presenta en la figura 4.18b.

Existe un caso alternativo en el cual el usuario ha modificado el estado de actividad del adaptador *bluetooth* desde las opciones del sistema operativo o desde otra aplicación. Por tanto, *AllexApp* está registrado a eventos del sistema operativo que notifican sobre los cambios en el estado de los dispositivos de *hardware* a todas las aplicaciones. Por ejemplo, si el usuario está en la pantalla para encender el *bluetooth* y recibe un evento *STATE_ON* desde el sistema operativo, deberá cargar la pantalla para iniciar la conexión. En otro escenario, si el usuario está en la pantalla de conexión y recibe un evento *STATE_OFF* porque el dispositivo *bluetooth* ha sido apagado, la aplicación carga la pantalla para activar el dispositivo *bluetooth* (figure 4.18a).

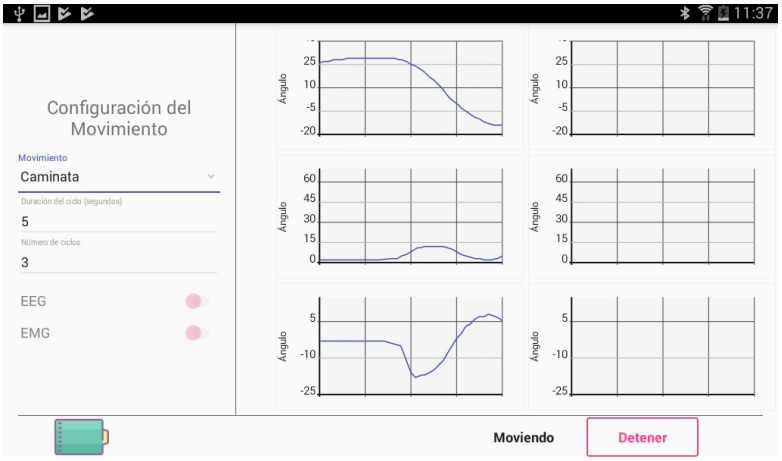
En la pantalla de conexión se listan los dispositivos que han sido previamente sincronizados con *AllexApp*. Estos dispositivos están identificados mediante



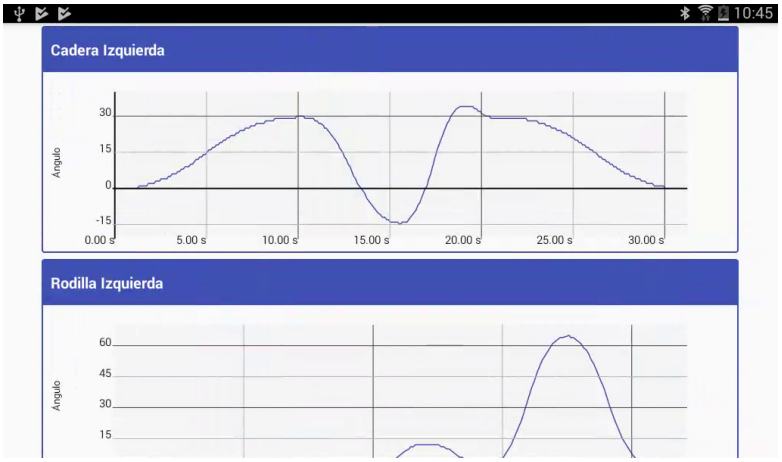
(a) Pantalla para configurar el *bluetooth*



(b) Pantalla de conexión con *AllexControl*



(c) Pantalla de control.



(d) Pantalla de Previsualización.

Figura 4.18: Pantallas de control de *AllexApp*.

su nombre y dirección MAC. Para el usuario, es de interés el dispositivo que se encuentra en dicha lista con el nombre *Allex*. Al presionar en esa opción, inicia un proceso para manejar la creación de los clientes de transmisión y recepción de datos. La aplicación debe crear dos *sockets* para el protocolo de comunicación por radio frecuencia (RFCOMM) en los cuales se especifica el UUID del servicio de transmisión y de recepción de *AllexControl*. Si el servicio existe y el dispositivo servidor acepta la conexión, se establece un canal de comunicación entre los dos sistemas. En caso de que la conexión sea negada o no se resuelva después de 12 segundos, se considera que la conexión ha fallado y se solicita al usuario que intente nuevamente la conexión.

Cuando la conexión ha sido establecida, se crea un hilo con operaciones que administran la entrada y salida de datos. La salida de datos por el bloque **transmisor** tiene funciones para convertir las cadenas de texto en *bytes*. El bloque **receptor** convierte los *bytes* recibidos en una cadena de caracteres y los publica en el sistema para que puedan ser usados por otras pantallas de la aplicación. Después de esta configuración se informa a *AllexApp* que la conexión ha sido establecida y que por tanto puede cargar la pantalla de control asociada al bloque **ControlGUI**.

4.4.2. Bloque para la Interfaz Gráfica de Control del Exoesqueleto

Corresponde a la interfaz gráfica de la figura 4.18c que usa los bloques de **transmisión** para enviar los comandos a *AllexControl* y los de **recepción** para mostrar la información sobre la posición angular de los motores, batería, cambios de estado y eventos que hayan ocurrido durante la ejecución del controlador. El funcionamiento de este bloque se basa en los tipos de comandos enviados a *AllexControl* y que se detallaron en la tabla 4.3. El primer comando es *READ* el cual es enviado cuando el usuario presiona el botón *"Lectura"* de la interfaz de control. La aplicación no permite la ejecución de nuevos comandos hasta que reciba un mensaje de confirmación *"ack,1"* o un mensaje de error desde *AllexControl*. caso de error, la pantalla regresa a su estado inicial para enviar nuevos comandos.

Cuando se recibe la confirmación del comando de lectura, se activa el hilo para el adaptador *bluetooth* de tal manera que se reciben los datos de posición angular de los motores con una sintaxis como la que se mostró en la tabla 4.2. Con esta información, el bloque conoce desde qué extremidad y articulación proviene el dato y puede dibujarlo en una de las seis gráficas de posición angular. Estas gráficas muestran los puntos como pares de posición angular respecto al tiempo. Se considera que el primer punto de posición angular está en la posición $(0, position)$, esto quiere decir que el tiempo de todos los puntos debe ser calculado de manera relativa al primero. Para evitar que las gráficas almacenen todos los puntos, únicamente se retienen aquellos generados en los últimos 4 segundos.

En el estado de lectura el único botón que puede ser presionado es el comando

para cancelar la lectura de los datos. Cuando se presiona este botón, se envía un comando *STOP* a *AllexControl*. No se puede enviar ningún otro comando hasta que se reciba el mensaje de confirmación *"ack,0"*.

El siguiente comando que se envía es *MOVE*. Para ello, el usuario previamente deberá configurar el movimiento mediante el formulario que se encuentra en la zona izquierda de la pantalla (figura 4.18c) bajo el título *"Configuración del Movimiento"*. Si el formulario no ha sido completado, aparecerá un cuadro de diálogo que valida si todos los campos han sido llenados. Este formulario incluye los siguientes elementos:

- *Movimiento:* Lista desplegable en la que se muestran los ejercicios que pueden ser ejecutados. Actualmente solo puede ejecutarse el movimiento de caminata.
- *Duración del ciclo (segundos):* Número entero de segundos para la duración del ciclo de ejercicios.
- *Número de ciclos:* Número de veces que se debe repetir el ejercicio.
- *EEG:* Activa la detección de intención de movimiento por electroencefalografía.
- *EMG:* Activa la detección de intención de movimiento por electromiografía.

Después de validar la configuración del movimiento se envía el comando *MOVE* a *AllexControl* con lo cual inicia la rutina de ejercicios. No se ejecuta ningún otro comando hasta que se reciba la confirmación en el formato *"ack,2"*. Durante la ejecución del ejercicio, el usuario puede presionar únicamente el botón *"cancelar"* para detener la ejecución del ejercicio de rehabilitación. En este periodo de tiempo se recibe también la información sobre la posición angular de los motores que es dibujada en cada gráfica temporal, pero a diferencia del comando *READ* esta información es almacenada temporalmente en una base de datos interna del dispositivo móvil.

Si el ejercicio de rehabilitación termina con el ciclo normal de ejecución, *AllexApp* pasa a la pantalla que se muestra en la figura 4.18d en la cual se previsualizan los datos de posición angular almacenados en el dispositivo móvil. Estas gráficas pueden ser escaladas y desplazadas en el dominio temporal. En esta pantalla el usuario tiene dos opciones o guarda la información en *AllexServer* o regresa a la pantalla de control para ejecutar otro ejercicio.

4.5. Diseño del Subsistema AllexServer

En esta sección se explica la lógica interna del servidor, las interfaces web expuestas y la manera en que este componente responde a las solicitudes del bloque **Cliente** de *AllexApp*. De la misma manera, se presenta el modelo de datos que se almacenará en un sistema de gestión para series de tiempo.

4.5.1. Bloque de Base de Datos de Series de Tiempo

Como se indicó en la sección 4.4.2, los datos adquiridos desde los sensores, particularmente la posición de los actuadores de las articulaciones, debe ser almacenada en una base de datos para su posterior visualización y análisis. La fiabilidad y alto rendimiento de este tipo de bases de datos, permite almacenar información que es generada de manera periódica y cuya medición varía respecto al tiempo. Por tanto, es útil en la adquisición de datos de sensores.

4.5.1.1. Definición del Modelo de Datos Gestionados por AllexServer

Se define como modelo de datos a *"un conjunto de conceptos, reglas y convenciones que permiten describir al problema que se desea resolver"* [93]. Por ende, un modelo de datos debe captar cada una de las propiedades estáticas y dinámicas de un objeto. Se debe asegurar que el modelo esté correctamente diseñado de tal manera que sea eficiente y garantice el mantenimiento de su estructura con un número mínimo de cambios. El modelo de datos planteado para *AllexServer* se presenta en la figura 4.19 mediante un diagrama entidad-relación.

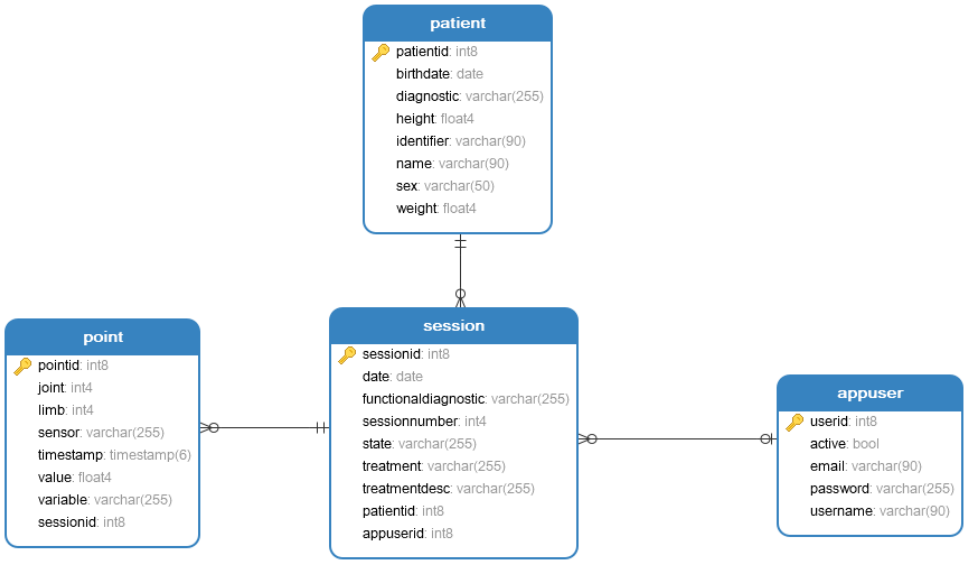


Figura 4.19: Diagrama entidad-relación del modelo de datos de *AllexServer*.

Entidades: Una entidad es una persona, cosa, concepto o suceso, real o abstracto, de interés en el dominio del problema. Es considerado como la estructura de datos que se desea almacenar. *AllexServer* cuenta con cuatro entidades principales que se describen a continuación:

appUser: Entidad que representa la información del usuario que utiliza la aplicación. Contiene atributos que permiten autenticarlo como el correo y la

contraseña.

- **active:** Atributo de tipo lógico que determina si un usuario tiene los permisos necesarios para acceder al sistema.
- **email:** Cadena de caracteres que representa el correo de registro del usuario.
- **username:** Cadena de caracteres que representa el nombre del usuario.
- **password:** Cadena de caracteres que almacena la contraseña cifrada del usuario.

Patient: Entidad que representa la información del paciente que asiste a las terapias de rehabilitación.

- **birthDate:** Atributo para marcas temporales que representa la fecha de nacimiento del paciente. La fecha tiene el formato *"yyyy-MM-dd"*.
- **diagnostic:** Cadena de caracteres que identifica la condición médica del paciente.
- **height:** Atributo de tipo flotante que representa la altura en centímetros del paciente.
- **identifier:** Cadena de caracteres que representa la cédula de identidad o pasaporte del paciente. El valor de este atributo debe ser único.
- **name:** Cadena de caracteres que almacena el nombre y apellido del paciente.
- **gender:** Cadena de caracteres que identifica el género del paciente. El valor que puede tomar el atributo es "Femenino" o "Masculino".
- **weight:** Atributo de tipo flotante que representa el peso en kilogramos del paciente.

Session: Entidad que representa la información de la sesión de rehabilitación de un paciente.

- **state:** cadena de caracteres que especifica el estado actual en el que se encuentra una sesión. El primer estado es **"No realizado"** y se da cuando la sesión ha sido creada pero aún no ha sido ejecutada. El segundo estado es **"Finalizado"** y ocurre cuando la sesión ya ha sido ejecutada.
- **date:** Atributo para marcas temporales que cumple con el formato *"yyyy-MM-dd"* y representa el momento en el que se creó la sesión.
- **funcionaldiagnostic:** Cadena de caracteres asociada a la conclusión médica de la sesión de rehabilitación.
- **sessionnumber:** Número entero que especifica el número de sesión actual del tratamiento.
- **treatment:** Cadena de caracteres que almacena información sobre el título del tratamiento para la sesión del paciente.
- **treatmentdesc:** Cadena de caracteres que almacena la descripción larga del tratamiento aplicado al paciente.

Point: Entidad que almacena los datos de las mediciones de los sensores del exoesqueleto.

- **joint:** atributo de tipo entero que representa la articulación desde la cual se adquirió el dato. El valor puede estar en el rango de 1 a 3, donde cada uno representa a una articulación, tobillo, rodilla y cadera respectivamente.
- **limb:** Atributo entero que toma dos valores: 1) 0 para representar la pierna izquierda y 2) 1 para representar la pierna derecha.
- **sensor:** Cadena de caracteres para el nombre del sensor del cual se están obteniendo los datos.
- **timestamp:** Marca de tiempo en la cual se adquieren los datos con precisión de nanosegundos.
- **value:** Atributo de tipo flotante que almacena la medición adquirida por el sensor.
- **variable:** Cadena de caracteres que almacena el tipo de dato que se está almacenando.

4.5.2. Bloque de Servidor de Aplicaciones

Un servidor de aplicaciones proporciona un entorno para el desarrollo y ejecución de aplicaciones web. Los servidores proveen su funcionalidad mediante una interfaz para programación de aplicaciones (API). Así los clientes no deben preocuparse por los detalles de implementación o de características de la infraestructura de la máquina sobre la cual se ejecutan.

Por esta razón, el bloque **Servidor** es el encargado de proveer una API con servicios web REST al bloque **Cliente** de *AllexApp*. De esta manera actúa como un intermediario entre los recursos o entidades de la base de datos y las aplicaciones móviles.

4.5.2.1. Arquitectura del Servidor

Una de las arquitecturas usadas en el desarrollo de servidores web consiste en dividir las funcionalidades y responsabilidades del sistema en una jerarquía de capas, en donde, cada capa se comunica únicamente con la inferior. Comúnmente, esta división se realiza en tres capas las cuales desde el nivel jerárquico superior son:

- *Capa de presentación:* Expone una interfaz al cliente a través de la cual es posible interactuar con la funcionalidad del servidor. Gestiona los datos, eventos y solicitudes que realiza el cliente, y delega su lógica y procesamiento a la capa de negocio.
- *Capa de negocio:* Gestiona la lógica de la aplicación al recibir los datos o eventos desde la capa de presentación. Por otro lado, se comunica con la capa de persistencia para almacenar o recuperar datos.
- *Capa de persistencia:* Gestiona la interacción con la base de datos con el objetivo de almacenar o recuperar información sobre los objetos del dominio y el estado de la aplicación.

El principal beneficio de esta arquitectura es la separación clara que existe entre la interfaz para el cliente y la lógica de la aplicación. Esto permite exponer al cliente diferentes presentaciones que utilicen la misma lógica. Por otro lado, si se redefine el modelo o tecnología de almacenamiento de datos solo se requiere modificar la capa de presentación.

La forma en la que las diferentes capas se encuentran distribuidas en *Allex-*

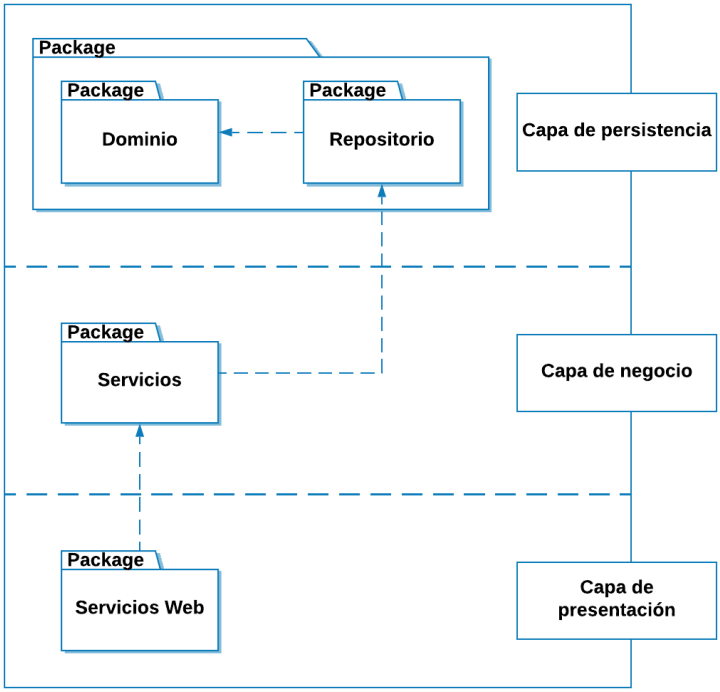


Figura 4.20: Arquitectura de capas de *AllexServer*

Server se muestra como un diagrama de paquetes en la figura 4.20. La primera capa a la cual el cliente accede es la de presentación. En el caso del servidor web desarrollado, la capa de presentación no corresponde a una interfaz gráfica. En su defecto son servicios web expuestos mediante una API REST al bloque **Cliente** de *AllexApp*. El acceso a esta capa está restringido mediante las políticas de seguridad del protocolo de autorización OAuth 2. Esta capa se encarga de validar el acceso a los recursos y obtener la información que es enviada por el cliente en los parámetros y cuerpo de la solicitud HTTP. Posteriormente deriva dichos datos hacia las funciones correspondientes de la capa de negocio. El paquete de la capa de servicios describe todos los procesos necesarios para realizar las operaciones y enviar una respuesta a la solicitud del cliente. Finalmente, está la capa de persistencia formada por dos paquetes. El primero está relacionado al dominio de la aplicación y corresponde a todas las entidades del modelo de dominio que se expuso en la sección 4.5.1.1. El segundo paquete es el repositorio, el cual se encarga de almacenar y recuperar los datos que se encuentran en el gestor de base de datos.

Los servicios web de la capa de presentación están agrupados según la entidad o recurso que modifican. La ruta del recurso y el método HTTP para acceder al recurso se resumen en el diagrama de en la figura 4.21 y la definición de todos sus parámetros y funcionalidad en el anexo B.

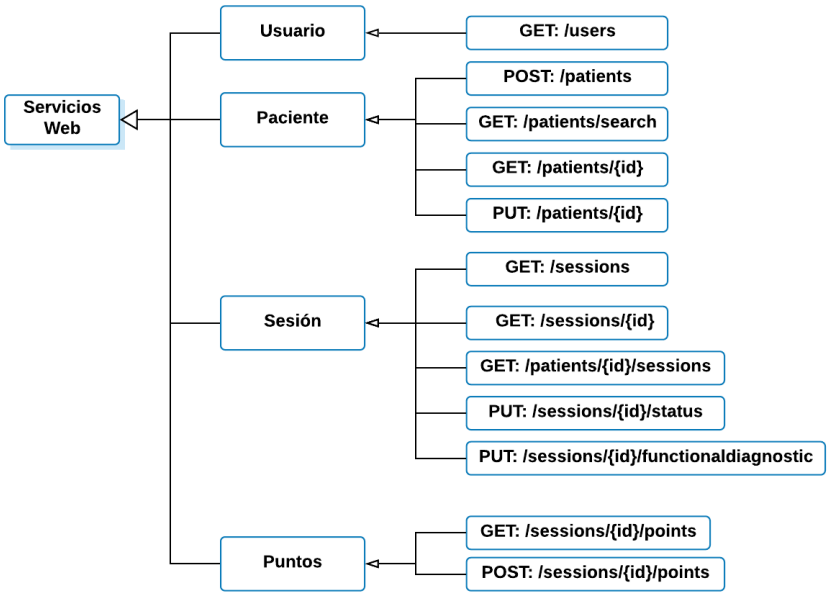


Figura 4.21: Servicios web provistos por *AllexServer*

4.6. Interacción entre *AllexApp* y *AllexServer*

En esta sección se describen los bloques de *AllexApp* que están encargados de la comunicación con *AllexServer*. La interacción entre estos bloques se realiza mediante una solicitud a los servicios web expuestos por la capa de presentación de *AllexServer*. Los bloques que se describen en esta sección son el bloque del **Cliente** que es el encargado de realizar las peticiones al servidor y el bloque de **UsuarioGUI** que maneja la lógica de las interfaces gráficas.

4.6.1. Bloque del Cliente

Es el componente encargado de realizar las solicitudes a *AllexServer*. Para la gestión de esta interacción requiere de tres componentes como son: la ruta del servicio, la autorización y el manejo de la respuesta obtenida desde el servidor. Las rutas de los servicios web se representan mediante una URI con parámetros, mismos que están definidos en el anexo B.

El componente de autorización gestiona el acceso a los recursos mediante el protocolo OAuth 2 con modo de funcionamiento mediante contraseña (*Password Grant Type*). Este modo de autorización le permite a la aplicación obtener un

token cuando el servidor valida el identificador del usuario y su contraseña. El *token* es una cadena de caracteres que representa el permiso que tiene la aplicación para acceder a los recursos del servidor sin que sea necesario enviar nuevamente las credenciales del usuario. Este *token* debe ser enviado en la cabecera de todas las solicitudes HTTP para que la petición al servidor sea autorizada.

El último componente está asociado al mecanismo para procesar las respuestas de los servicios web. Para facilitar esta actividad se creó la estructura de datos *ServerResponse* de tal manera que todos los servicios web definidos responden con la siguiente estructura de datos:

- *state:* Representa el estado de éxito del procesamiento de la solicitud. Si la solicitud terminó con el caso de uso normal, tiene un valor lógico *True*. Caso contrario, su valor es *False*.
- *message:* Cadena de caracteres que representa un mensaje que se muestra en un cuadro de diálogo de *AllexApp* después de la ejecución de un servicio. El mensaje puede ser de éxito o de error dependiendo del valor del atributo *state*.
- *response:* Representa alguna de las estructura de datos que se obtiene desde el servidor cuando se completa una solicitud HTTP de tipo GET. Generalmente corresponde a una entidad del dominio de datos o a un listado de entidades. Esta estructura de datos es enviada como un archivo JSON; sin embargo, este componente se encarga de transformarlo en un objeto que puede ser usado por la aplicación.

4.6.2. Bloque para la Interfaz Gráfica de Usuario

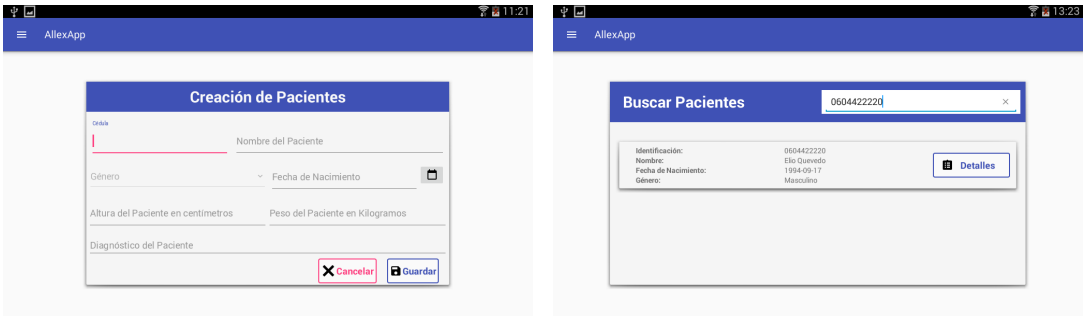
Es el encargado de gestionar la lógica de interacción de las interfaces gráficas de usuario. Esto incluye exponer información a manera de pantallas, procesar eventos que se generan al presionar elementos gráficos como los botones y gestionar las solicitudes a los servicios web. Por tanto, el comportamiento de este bloque será explicado mediante el funcionamiento de las interfaces gráficas de usuario de manera breve. En el manual de usuario, localizado en el anexo E, se explican características y comportamientos adicionales.

La primera pantalla de la aplicación es la de ingreso al sistema que se presenta en la figura 4.22. Esta pantalla tiene dos campos obligatorios para la autorización del usuario como son el correo y la contraseña. Al presionar en el botón *Ingresar*, el módulo de **Cliente** ejecuta el protocolo OAuth 2 para obtener el *token* de autorización de acceso al servidor.

Figura 4.22: Pantalla de inicio de sesión

Para registrar a un nuevo paciente en el sistema se usa el formulario que se presenta en la figura 4.23a, con campos similares a los especificados por la entidad de datos *Patient* presentada en la sección 4.5.1.1. Al presionar en el botón guardar, el bloque **Cliente** realiza una petición al servicio web */patients* para registrar a un nuevo paciente.

La pantalla de búsqueda de paciente presentado en la figura 4.23b, permite que el usuario encuentre un paciente previamente registrado. La búsqueda se realiza por nombre o número de identificación mediante el servicio web */patients/search*. Como resultado, se presenta al usuario una lista con los pacientes encontrados.



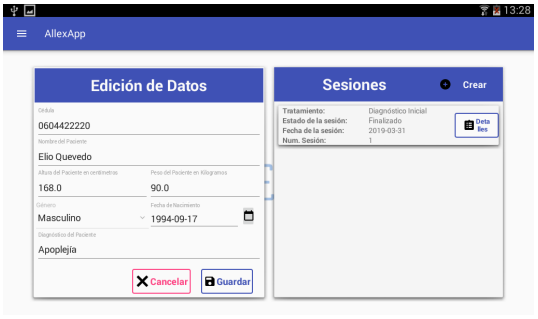
(a) Creación de paciente (b) Búsqueda de paciente

Figura 4.23: Pantallas de creación y búsqueda de pacientes.

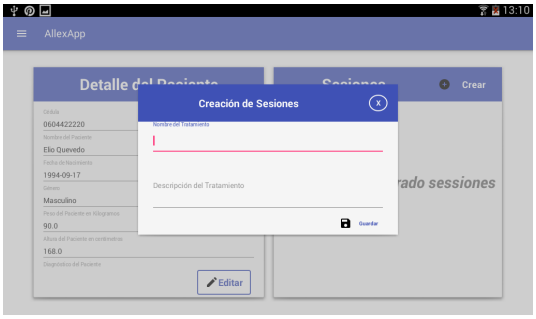
El ingreso a la pantalla de detalle de paciente (figura 4.24) ocurre de dos maneras diferentes. Una de ellas ocurre cuando el usuario crea un nuevo paciente. La otra forma se da cuando se presiona en el botón *Detalle* de los resultados de la búsqueda del paciente. La pantalla de detalles presenta la información del paciente y su historial de sesiones de rehabilitación. Los atributos del paciente se obtienen mediante el servicio *patients/{id}*. Por otro lado, el listado de sesiones de rehabilitación del paciente requiere el servicio */patients/{id}/sessions*.

Figura 4.24: Pantalla de detalle de paciente

Para cargar la pantalla de edición de detalles del paciente, el usuario debe presionar el botón *Editar* de la pantalla de detalles del paciente. Al realizar esta acción se carga una pantalla como la que se presenta en la figura 4.25a con campos editables para la información del paciente. Al presionar en el botón *Guardar*, se realiza una solicitud al servicio web */patients/{id}* para alterar la información de la ficha médica. Para crear una nueva sesión de rehabilitación se presiona en el botón crear de la pantalla de detalles del paciente. De esa manera, se muestra un cuadro de diálogo como el de la figura 4.25b la cual permite agregar una descripción para la sesión. Al presionar en el botón *Guardar*, se realiza una llamada al servicio web */sessions* que crea una nueva sesión y le asigna un número de identificación.



(a) Formulario de edición del paciente



(b) Formulario de creación de sesión de rehabilitación

Figura 4.25: Pantallas de edición de información de pacientes y creación de sesiones de rehabilitación.

Para acceder a la información de una sesión de rehabilitación finalizada, el usuario debe estar en la pantalla de detalle de pacientes. Al presionar sobre el botón *Detalles* de alguna de las sesiones de rehabilitación que se encuentran listadas, *AllexApp* realiza una petición al servicio web */sessions/{id}*. *AllexServer* devuelve información sobre el detalle de la sesión como la fecha en la que fue realizada. Conjuntamente, se utiliza el servicio web */sessions/{id}/points* para obtener los datos de posición angular de las articulaciones que se adquirieron en dicha sesión. Estos dos grupos de información se presentan en la figura 4.26a.

La pantalla de detalle de la sesión presenta una función adicional. Esta permite que el usuario agregue un diagnóstico funcional. Al pulsar sobre el botón *Diagnostico Funcional*, se abre una ventana como la que se muestra en la figura 4.26b. En esta ventana hay un cuadro de texto para el ingreso del diagnóstico funcional obtenido luego de la sesión de rehabilitación. Esta información es enviada a *AllexServer* utilizando el servicio */sessions/{id}/functionaldiagnostic*.

(a) Detalle de la sesión de rehabilitación (b) Ingreso del diagnóstico funcional

Figura 4.26: Pantallas del detalle de una sesión de rehabilitación.

CAPÍTULO 5

Implementación del Sistema de Control

En este capítulo se presentan las herramientas que fueron usadas para el desarrollo del sistema de control centralizado que fue diseñado en el capítulo anterior. Esto incluye la especificación de los entornos de desarrollo, lenguajes de programación, librerías, herramientas de compilación, depuración y despliegue. Adicionalmente, se redactan algunas recomendaciones para la implementación del sistema. Esto se realiza tanto para la aplicación móvil *AllexApp* (sección 5.2), para el sistema de control embebido *AllexControl* (sección 5.1) y el servidor de aplicaciones *AllexServer* (sección 5.3). Finalmente, se resumen algunas experiencias del desarrollo (sección 5.4).

5.1. Implementación del Subsistema *AllexControl*

El proceso de instalación para las dependencias de las capas de la infraestructura del subsistema de control embebido se presenta en el anexo C. Este subsistema se desarrolló con las librerías base que provee ROS2 para la programación en *C++* y *Python*, estas son:

- *rclcpp:* Librería para la creación de aplicaciones de ROS2 en *C++*.
- *rclpy:* Librería para la creación de aplicaciones de ROS2 en *Python*.
- *std_msgs:* Conjunto de mensajes en formato ROSIDL en el que se definen los tipos de datos primitivos de ROS2 como **bool**, **byte**, **char**, **float32**, **float64**, **int8**, **uint8**, **int16**, **uint16**, **int32**, **uint32**, **int64**, **uint64**, **string**.
- *builtin_interfaces:* Posee las definiciones de mensajes en formato ROSIDL para aquellas estructuras de datos asociadas al manejo de datos temporales

como **Time** cuya estructura fue descrita en la figura 4.5 y **Duration**.

- *rclcpp_lifecycle:* Permite crear nodos en *C++* que incluyan una máquina de estados para el manejo del ciclo de vida.
- *lifecycle_msgs:* Posee mensajes para el manejo de los estados y transiciones del ciclo de vida de los nodos. De la misma manera expone servicios que permiten conocer el estado actual de un nodo y solicitar cambios sobre dicho estado.

No se usó ningún entorno integrado de desarrollo debido a que el único IDE con soporte directo para ROS2 es *ROS Development Studio* [1]. Esta herramienta, a pesar de proveer un servicio de uso gratuito, tiene restricciones en el tiempo mensual de uso. Ante esta situación, se desarrolló la aplicación en la versión 1.34.0 del editor de texto *Atom*, misma que provee coloreado de sintaxis para *C++* y *Python*. Por este motivo se requiere administrar manualmente la estructura de directorios del proyecto (figura 5.1) y usar herramientas para la compilación y depuración de código.

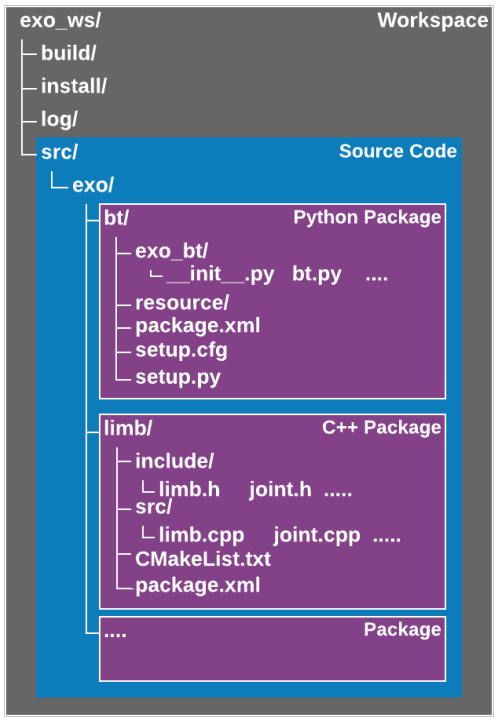


Figura 5.1: Estructura del proyecto *AllexControl*

5.1.1. Estructura de Directorios del Proyecto

El directorio raíz de ROS2 se denomina espacio de trabajo (*workspace*), el cual en este proyecto se ha denominado *exo_ws*. Dentro del espacio de trabajo debe

[1] **ROS Development Studio:** Un IDE para el desarrollo de ROS2. http://www.theconstructsim.com/rds-ros-development-studio

existir un directorio designado con el nombre *src* para el código fuente que es creado por el desarrollador. Adicionalmente, existen otros tres directorios que se crean como resultado de la compilación:

- *build:* Posee los archivos que resultan de la compilación o configuración de cada paquete.
- *install:* Son *scripts* encargados de la instalación de los archivos ejecutables del directorio *build*. De esta manera pueden ser utilizados por las herramientas de línea de comandos de ROS2.
- *log:* Contiene información sobre el proceso de compilación del proyecto. Esto incluye los comandos invocados y los registros de las respuestas del sistema.

En ROS, el código fuente del usuario en el directorio *src* debe estar organizado en paquetes. Un paquete es un conjunto de archivos que pueden incluir código fuente, conjuntos de datos, archivos de configuración o cualquier otro recurso que constituya una pieza que le otorgue funcionalidad a todo el sistema. Cada paquete debe tener un nombre único y diferente a los paquetes existentes en ROS2. Por otro lado, durante el proceso de compilación se asocia cada archivo ejecutable al paquete en el cual fueron definidos. Por tanto, estos archivos deben tener un nombre único dentro del paquete al cual pertenecen.

En la tabla 5.1 se presenta las relaciones existentes entre los paquetes y archivos ejecutables que fueron desarrollados para *AllexControl* y los bloques de la arquitectura del sistema a los cuales instancian.

Paquete	**Ejecutable**	**Bloque**
exo_battery	exo_battery	Batería
	exo_battery_test	Sistema de Gestión de Energía
exo_bt	exo_btTx	Transmisión
	exo_btRx	Recepción
exo_intention	exo_intention	Planificador
	exo_intention_emg	Intención EMG
	exo_intention_eeg	Intención EEG
exo_limb	exo_limb	Extremidad, Sensor Motor, EEG, EMG, Actuador
exo_orchestrator	exo_orchestrator	Orquestador

Tabla 5.1: Relación entre el diseño de la arquitectura del sistema y ejecutables.

Cada paquete posee un archivo de configuración denominado *package.xml* que

tiene una estructura similar a la que se presenta en el código 5.1. Este archivo posee toda la información necesaria para que el administrador de compilación de ROS2 pueda conocer el nombre del paquete (representado mediante la etiqueta *<name>*) y sus dependencias de compilación (*<build_depend>*) y ejecución (*<exec_depend>*).

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format2.xsd" schematypens
    ="http://www.w3.org/2001/XMLSchema"?>
<package format="2">
  <name>exo_limb</name>
  <version>0.5.1</version>
  <description>Módulo de control de extremidades</description>
  <license>Apache License 2.0</license>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <build_depend>exo_utils</build_depend>
  <build_depend>exo_interfaces</build_depend>
  <build_depend>limb</build_depend>
  <build_depend>std_msgs</build_depend>
  <exec_depend>motores</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Código 5.1: Archivo package.xml para la definición de paquetes

También existen paquetes que no crean archivos ejecutables, sino que corresponden únicamente a librerías de funciones, estructuras de datos comunes o mensajes y servicios descritos mediante el lenguaje ROSIDL como se presenta en la tabla 5.2.

Paquete	**Descripción**
exo_interfaces	Definición de los tipos de dato *SensorData* para la transmisión de datos de los sensores y *Config* para la rutina de configuración del movimiento
exo_utils	Definición de las enumeraciones *ExoState*, *BatteryState*, *ExoCmd*, *JointType*, *LimbType* para que puedan ser reutilizadas por los paquetes implementados en *C++*
exo_utils_py	Definición de las enumeraciones mencionadas para el paquete *exo_utils* pero implementadas como una librería para paquetes implementados en *python*

Tabla 5.2: Paquetes del sistema con definición de librerías.

Finalmente se implementaron tres paquetes adicionales, estos no están directamente relacionados a los bloques de diseño del subsistema que se plantearon en el capítulo 4. Este conjunto de paquetes que se presentan en la tabla 5.3 resuel-

ven problemas propios de la implementación como el despliegue y configuración concurrente de múltiples archivos ejecutables, así como la ejecución de pruebas.

Paquete	Descripción
exo_bringup	Se base en la funcionalidad *ros2 launch* definida en la sección 2.4.3.8 en el cual se configura e inicializa los procesos de ejecución de diferentes archivos ejecutables.
exo_test_ui	Interfaz gráfica implementada con el *framework Qt* versión 5 mediante la cual es posible simular eventos de intención de movimiento y de cambios de información de la batería. De la misma manera recibe la información sobre el estado *ExoState* del exoesqueleto, el estado de conexión y los datos de posición de los motores.
exo_performance	Usado para evaluar el número de paquetes enviados por *AllexControl* y de esa manera determinar la cantidad de paquetes de datos perdidos y la frecuencia de transmisión de los mismos.

Tabla 5.3: Paquetes para la ejecución y evaluación del sistema.

5.1.2. Construcción del Proyecto

ROS2 provee una herramienta de administración de paquetes denominada *colcon* [94]. Si bien es cierto, no es una herramienta de compilación, pero utiliza los programas *CMake* y *Python setuptools* para generar archivos ejecutables de *C++* y *Python* respectivamente.

Colcon busca los paquetes de *software* dentro del directorio *src* y obtiene la información sobre los nombres y dependencias de los mismos a partir de los archivos de configuración *package.xml*. Esta información le permite a la herramienta determinar el orden en el que los paquetes deben ser compilados e incluso realizar compilación de manera concurrente. El comando para la compilación es usado en el código 5.2 a continuación:

```
1 $ colcon build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

Código 5.2: Compilación en *AllexControl*

Como resultado de este proceso se generan los directorios *build*, *install* y log detallados en la sección 5.1.1. En ocasiones se requiere eliminar las carpetas *build* e *install* cuando se realizan cambios en el paquete de estructuras de datos *exo_interfaces* debido a que los directorios de compilación todavía preservan la información de las versiones anteriores de los mensajes y puede ocasionar comportamientos no deseados durante la ejecución del sistema.

5.1.3. Ejecución y Depuración

Una vez que el proyecto ha sido compilado e instalado es posible ejecutar los archivos binarios de cada paquete. Para ello se usa el comando *ros2 run* en el cual se indica el nombre del paquete seguido del nombre del ejecutable. Esto es útil para evaluar el funcionamiento de cada uno de los componentes del sistema. Por ejemplo para ejecutar el componente de transmisión de datos (*exo_btTx*) del paquete *exo_bt* se debe ejecutar el código 5.3.

```
$ ros2 run exo_bt exo_btTx
```

Código 5.3: Inicio del proceso de transmisión en ROS2

Generalmente cuando existen errores en tiempo de ejecución para procesos desarrollados en *Python*, las excepciones son desplegadas en la interfaz de línea de comandos, con lo que es posible determinar la causa del error y la línea de código asociada a la misma. Sin embargo, en el caso de *C++* los errores no son explícitos, por el contrario requieren una herramienta de depuración como *GNU Project Debugger (GDB)* para poder identificarlos. Para ejecutar un proceso con *GDB* se usa el comando definido en el código 5.4.

```
$ ros2 run --prefix 'gdb -ex run --args' exo_limb exo_limb
```

Código 5.4: Ejecución de paquetes con la herramienta de depuración GDB.

5.1.4. Estructuras de Datos para los Mensajes y Servicios

Las estructuras de datos están definidas dentro del paquete *exo_interfaces* con el formato ROSIDL. Los mensajes son representados mediante archivos con extensión *.msg* en el cual cada línea representa una variable, siendo el texto de la primera columna el tipo de la variable y el de la segunda columna el nombre de la misma. El tipo de la variable puede ser otro mensaje definido en algún paquete previamente compilado. En este paquete el único mensaje que se ha definido corresponde a la información para los datos de los sensores (**SensorData**) descrito en la sección 4.3.2 y representado por el archivo de mensajes expuesto a continuación:

`interfaces/msg/SensorData.msg`

```
int8 limb
int8 joint
string node
string variable
float32 data
builtin_interfaces/Time timestamp
```

Por otro lado, los servicios se definen mediante un archivo con formato *.srv*. En *AllexControl* el archivo *Config.srv* corresponde a la interfaz para el servicio de

configuración de la trayectoria de movimiento que fue definida en la figura 4.10 bajo la estructura de datos **ConfigReq**.

interfaces/srv/Config.srv

```
uint8 iteration
uint8 hiking_time
uint8 exercise
bool emg_active
bool eeg_active
---
bool resultado
```

El archivo de servicios está constituido por dos zonas de texto separadas por una línea con tres guiones. En la parte superior se coloca la estructura de datos para el mensaje de la solicitud del servicio. En la parte inferior otro conjunto de variables correspondientes a la respuesta que el cliente espera por parte del servidor. En este caso, la respuesta consiste en un valor lógico que informa sobre el éxito o fracaso de la solicitud.

No se crearon estructuras de mensajes para las enumeraciones debido a que:

- El tiempo de compilación de cada mensaje ROSIDL es alto debido a que se generan archivos para su manejo en *C++* y *Python*, así como representaciones de otros proveedores de tecnología DDS como *OpenSplice* y *e-prosima FastRTPS*.
- El funcionamiento de la comunicación cliente-servidor tiene un error de implementación en la versión *Crystal Clemmys* de ROS2. Ocasionalmente el cliente no encontraba el servicio, incluso cuando el servidor estaba activo y listo para recibir peticiones. Este error fue detectado y solucionado, pero todavía no ha sido integrado en la línea principal de desarrollo debido a que requiere más pruebas de compatibilidad con los diferentes proveedores de servicios DDS.

5.1.5. Calidad de Servicio

No todos los mensajes que son transmitidos a través de DDS requieren las mismas garantías de servicio durante el envío y recepción de datos. Por ejemplo, los mensajes de control siempre deben ser recibidos, pero es posible que una lectura de un sensor pueda ser ignorada en favor de los eventos de control. Para configurar las características de comunicación se usaron tres tipos de perfiles de calidad de servicio que se presentan en la tabla 5.4. La definición de los perfiles está basada en las cuatro políticas de calidad de servicio con las que se puede interactuar en ROS2 (sección 2.4.3.3). Los perfiles de calidad de servicio también se definen en el paquete *exo_utils* y *exo_utils_py* y son:

- *late_joined:* Perfil que garantiza la transmisión, recepción y sincronización de todos los datos. La sincronización se consigue mediante la política de durabilidad *TRANSIENT_LOCAL*. Con esta configuración los publicadores almacenan un conjunto de los últimos mensajes transmitidos. El publicador envía estos datos almacenados cuando detecta que un nodo registra un nuevo suscriptor en el sistema. De esta forma se obtiene un estado consistente para todos los nodos distribuidos en el sistema. Esta configuración es útil para datos de eventos como es el caso del estado global del exoesqueleto.
- *sensor:* Este perfil está configurado para la publicación de mensajes volátiles y con fiabilidad *BEST_EFFORT*. Como su nombre lo indica, es usado para publicar flujos de datos para los sensores en los que se puede tolerar la pérdida de mensajes en favor de eventos de control. Una ventaja de esta configuración es que disminuye la latencia de la red debido a que la alta frecuencia de adquisición de los datos de sensores podría causar múltiples retransmisiones. En ese caso puede causar que muchos de los datos que llegan a su destinatario estén desactualizados, y por tanto, ya no sean de utilidad para el procesamiento de sus algoritmos.
- *default:* Es el perfil de transmisión por defecto de ROS2, a diferencia del perfil *last_joined*, este perfil no se encarga de la sincronización de los datos debido a que la información ya no tiene valor por estar desactualizada. Sin embargo, se asegura de que el receptor reciba todos los mensajes.

Política	**late_joined**	**sensor**	**default**
Historia	KEEP_LAST	KEEP_LAST	KEEP_LAST
Fiabilidad	RELIABLE	BEST_EFFORT	RELIABLE
Durabilidad	TRANSIENT_ LOCAL	VOLATILE	VOLATILE
Profundidad	5	5	10

Tabla 5.4: Perfiles de calidad de servicio para *AllexControl*.

5.1.6. Representación del Sistema como Nodos

En la tabla 5.1 se especificó la correspondencia entre paquetes, archivos ejecutables y bloques del sistema. Sin embargo, la mínima unidad de computación de *ROS2* son los nodos. En efecto, cada uno de los archivos ejecutables debe estar compuesto internamente por uno o más nodos agrupados dentro de un solo ejecutor (sección 5.1). La única restricción está asociada al nombre de cada nodo, el cual en tiempo de ejecución debe ser único en todo el sistema.

En la tabla 5.1, muchos de los archivos ejecutables están asociados únicamente a un bloque del sistema, y por tanto, la correspondencia entre un nodo y un archivo ejecutable también es de uno a uno. Por ejemplo, el archivo ejecutable *exo_battery* posee un solo nodo con el mismo nombre. Sin embargo, el ejecutable *exo_limb* para las extremidades puede llegar a tener un máximo de nueve nodos internos para cada extremidad. Para que los nombres sean únicos para cada extremidad se definieron los prefijos *"l"* para representar a los nodos de la pierna izquierda y *"r"* para los de la pierna derecha. A continuación, se presentan los nodos de los ejecutables de extremidad, usando para ello la nomenclatura de la pierna izquierda:

- Un nodo para el bloque de *Extremidad* denominado *l_limb*.
- Tres nodos para recuperar la información de los sensores de cada articulación. Estos se denominan *l_hip_sensor*, *l_knee_sensor* y *l_ankle_sensor*.
- Tres nodos para manejar los comandos de movimiento de los actuadores de cada articulación. Estos se denominan *l_hip_actuator*, *l_knee_actuator* y *l_ankle_actuator*.
- Un nodo para la recuperación de datos de EMG denominado *exo_emg* y otro para los datos de EEG denominado *exo_eeg*.

Todos estos nodos se agrupan dentro de un solo proceso ejecutable puesto que deben usar el mismo *socket* de comunicación con los dispositivos *EPOS4* que es administrado por el nodo *l_limb*.

5.1.7. Configuración de la Comunicación

De acuerdo a la arquitectura del sistema presentada en el diagrama de bloques del capítulo de diseño (capítulo 4), cada uno de los componentes de *AllexControl* está constituido por puertos para el flujo de entrada y/o salida de datos o servicios que son provistos y requeridos. En ROS2, todos los puertos de flujo de entrada de datos se representan como un suscriptor, éste es el componente encargado de recibir mensajes y, de la misma manera, todos los puertos de salida de datos se representan como un publicador que envía mensajes a sus suscriptores. Las interfaces provistas corresponden a servicios que requieren un conjunto de parámetros para su ejecución y las interfaces requeridas corresponden a clientes que solicitan dichos servicios y esperan una respuesta de su parte.

Las interacciones entre un publicador y un suscriptor requieren de la definición de un canal de comunicación con tres componentes:

- *Tema:* Identificador único del canal a manera de una cadena de caracteres.
- *Tipo:* Estructura de datos a ser transmitida.
- *Calidad de Servicio*

Los componentes que definen los canales de comunicación de *AllexControl* se presentan en la tabla 5.5.

Tema	Tipo	QoS	Descripción
state	Int16	late_joined	Estado global de *ALLEX-1*.
rx_state	Bool	late_joined	Conexión o desconexión del nodo de recepción de datos.
tx_state	Bool	late_joined	Conexión o desconexión del nodo de transmisión de datos.
message	String	default	Cadena de caracteres para desplegar una notificación en *AllexApp*.
error	String	default	Cadena de caracteres para informar sobre un error en algún nodo.
orchestrator	String	default	Comando que es enviado al orquestador desde *AllexApp*.
motor/position	SensorData	sensor	Posición de los motores.
motor/current	SensorData	sensor	Corriente de los motores.
motor/velocity	SensorData	sensor	Velocidad de los motores.
emg_data	SensorData	sensor	Datos de electromiografía.
eeg_data	SensorData	sensor	Datos de electroencefalografía.
battery/sensor	SensorData	sensor	Información de los sensores de batería como temperatura, voltaje y corriente.
battery/state	Int8	late_joined	Estado de carga de la batería.
emg_intention	Bool	default	Detección de intención de movimiento por electromiografía.
eeg_intention	Bool	default	Detección de intención de movimiento por electroencefalografía.
intention_test	Bool	default	Simulación de los mensajes de intención de movimiento.
movefinished	Bool	default	Mensaje que informa que el movimiento de uno de los motores ha terminado.

Tema	Tipo	QoS	Descripción
movecmd	Int8	default	Mensaje que notifica a los motores si estos deben realizar el movimiento *INITIAL*, *CYCLIC* o *STOP*.

Tabla 5.5: Canales de transmisión de datos entre nodos.

Para poder transmitir o recibir datos mediante estos canales de comunicación, cada nodo debe definir un conjunto de suscriptores o publicadores dependiendo de los datos o eventos en los cuales está interesado. En la tabla 5.6, se presenta la topología de comunicaciones entre nodos. En ésta se identifica cuáles son los nodos que producen el mensaje, y cuales son aquellos que están interesados en recibirlos por cada tema que ha sido definido. De esta manera es posible determinar las dependencias entre cada uno de ellos.

Tema	Publicador	Suscriptor
state	exo_orchestrator	Todos los nodos
rx_state	exo_btRx	exo_orchestrator, exo_btTx
tx_state	exo_btTx	exo_orchestrator, exo_btRx
message	Todos los nodos	exo_orchestrator, exo_btTx
error	Todos los nodos	exo_orchestrator, exo_btTx
orchestrator	exo_btTx	exo_orchestrator, exo_test_ui
motor/position	Nodos sensores de las articulaciones como l_hip_sensor, l_knee_sensor y l_ankle_sensor	exo_btTx, exo_test_ui, exo_performance
motor/current	Nodos sensores de las articulaciones	exo_performance
motor/velocity	Nodos sensores de las articulaciones	exo_performance
emg_data	exo_emg	exo_intention_emg
eeg_data	exo_emg	exo_intention_eeg
battery/sensor	exo_battery	exo_performance

Tema	Publicador	Suscriptor
battery/state	exo_battery	exo_btTx
emg_intention	exo_intention_emg	exo_intention
eeg_intention	exo_intention_eeg	exo_intention
intention_test	exo_test_ui	exo_intention
movefinished	Actuadores como l_hip_actuator, l_knee_actuator y l_ankle_actuator	exo_intention
movecmd	exo_intention	Actuadores como l_hip_actuator, l_knee_actuator y l_ankle_actuator

Tabla 5.6: Comunicación entre publicadores y suscriptores.

Por otro lado, la comunicación cliente-servidor requiere de los mismos componentes de un canal de comunicación. La diferencia está en el tipo de dato, el cual debe ser de tipo servicio y un identificador único para el tema. El perfil de calidad para todos los servicios tiene políticas que son iguales a las del perfil *default*. En el caso de *AllexControl* se ha implementado este tipo de interacción únicamente en dos casos.

El primer caso corresponde al comportamiento de los bloques de actuadores y de planificación que fueron definidos en las secciones 4.3.8 y 4.3.9 respectivamente. El primero corresponde al servicio *Config* que fue definido en la sección 5.1.4. El identificador o tema de estos servicios está definido por el nombre del nodo seguido del sufijo *"_config"*, de esta manera se obtienen los temas *"exo_intention_config"*, *"l_hip_actuator_config"*, *"l_knee_actuator_config"* y *"l_ankle_actuator_config"*. El cliente de estos servicios es el nodo *exo_orchestrator*. Con la respuesta de cada servicio se conoce si la ejecución de la configuración fue o no exitosa en cada nodo.

El segundo caso corresponde a los servicios que modifican la máquina de estados de ciclo de vida de un nodo. Esta máquina de estados y sus servicios están implementados en la dependencia *rclcpp_lifecycle* y estos son:

- */change_state:* Es el sufijo para el identificador del servicio y que está precedido por el nombre del nodo. Por ejemplo, el identificador l_ankle_actuator/change_state representa el servicio para cambio de estado del actuador del tobillo izquierdo. Este servicio requiere que el parámetro de la solicitud sea el tipo de transición que se desea que ejecute la máquina de estados entre las definidas en la sección 2.4.3.7. Como respuesta se obtiene

si el resultado de la transacción ha sido exitoso o no.
- */get_state:* Es el sufijo para el identificador del servicio que permite obtener información sobre el estado actual de un nodo. El prefijo también corresponde al nombre del nodo. Por ejemplo, l_ankle_actuator/get_state para el estado del nodo para la rodilla izquierda. El servicio no requiere ningún parámetro en la solicitud del mensaje. Como resultado de la solicitud el cliente obtiene el estado actual del nodo.

Todos los clientes de los servicios descritos anteriormente son creados desde el nodo *exo_orchestrator*. Debido a que es el encargado de gestionar los cambios de estado en el sistema.

5.1.8. Archivos de Configuración de Funcionalidad

El último componente que se requiere en el diseño e implementación de un sistema robótico es la configuración. Los parámetros de configuración deben definirse en una estructura de datos a manera de un mapa con pares clave y valor o un archivo de texto con formato *yaml* (sección 2.4.3.6). En *AllexControl* se definieron cuatro archivos de configuración que se presentan en el anexo D como parte del paquete *exo_bringup*. Las ventajas de especificarlos como parte de un archivo *yaml* son:

- Se puede modificar el comportamiento del exoesqueleto antes de la ejecución del sistema. Por ejemplo, al ejecutar las configuraciones del ambiente de prueba o para habilitar su interacción con el *hardware* del exoesqueleto.
- No se requiere recompilar el código de cada nodo, debido a que es un proceso computacionalmente costoso para los sistemas embebidos.

No se implementó la modificación o lectura de parámetros de otros nodos en tiempo de ejecución. Esto debido a un error de funcionamiento de *Fast-RTPS* en la cual el cliente falla en la identificación del servidor correspondiente.

5.1.9. Despliegue de los Nodos del Sistema

Desplegar cada ejecutable del sistema de manera manual como se indicó en la sección 5.1.3 es un proceso repetitivo. Este proceso puede ser automatizado mediante la librería *ros2launch* para *python* debido a que administra y supervisa la ejecución de los archivos binarios instalados en el proyecto. Para ello se desarrolló el archivo *local.init.launch* en el paquete *exo_bringup*. El cual cumple dos funciones:

- Configurar todos los ejecutables de manera previa a su despliegue. Esto incluye la identificación y carga del archivo de configuración de cada nodo. Así como el envío de parámetros entre nodos debido a las limitaciones expuestas en la sección anterior.
- Inicializar la ejecución de cada binario como un proceso independiente del sistema operativo.

En la línea de código 5.5 se presenta la función usada para el despliegue de procesos. Esta función requiere como parámetros el nombre del paquete, el nombre del ejecutable y los archivos de configuración. En este caso en particular representa el proceso de la pierna izquierda del exoesqueleto.

```
action = launch_ros.actions.Node(package='exo_limb', node_executable='exo_limb' , output='screen', arguments=[str(LimbType.RIGHT_LEG.value)], parameters=[limb_config_file])
```

Código 5.5: Despliegue de un ejecutable.

El comando usado para inicializar el proceso de despliegue del sistema se muestra en el código 5.6:

```
$ ros2 launch exo_bringup local.launch.py
```

Código 5.6: Despliegue de un ejecutable.

El resultado de la ejecución se muestra en la figura 5.2. Esta incluye las confirmaciones de la ejecución de los procesos mediante el identificador de proceso (*Process IDentifier - PID*) y el nombre de cada archivo ejecutable.

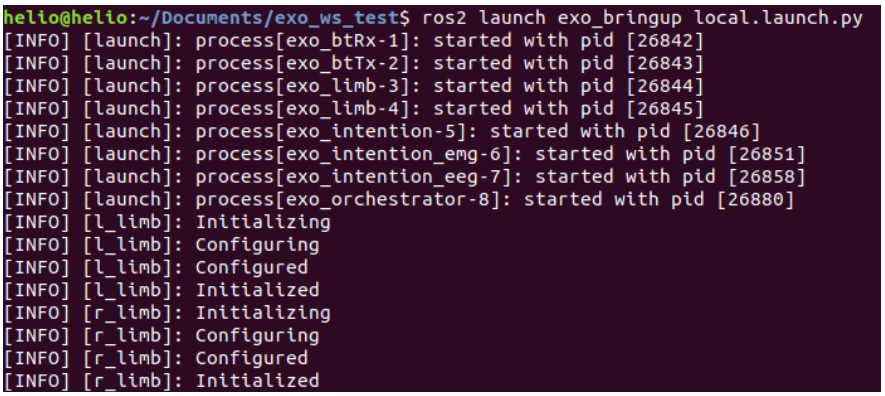


Figura 5.2: Despliegue de *AllexControl*.

5.1.10. Interfaz Gráfica de Pruebas

La interfaz gráfica de pruebas está definida dentro del paquete *exo_test_ui*. Inicialmente esta interfaz no estaba considerada dentro de los requerimientos del sistema. Sin embargo, era necesaria durante la etapa de desarrollo para facilitar la visualización del comportamiento del sistema. Esta aplicación de escritorio, cuya pantalla se presenta en la figura 5.3, fue desarrollada en el lenguaje de programación *Python* y usando la librería gráfica *Qt5*. Para agilizar el proceso de creación de los componentes gráficos se usó el programa *Qt5 Designer* el cual permite colocar los elementos gráficos en la pantalla mediante una interacción de arrastrar y soltar (*drag & drop*). Esta interfaz gráfica funciona con dos hilos:

- Un hilo para la interfaz gráfica que se encarga de procesar los eventos con los elementos gráficos como los botones, despliegue de datos, etc.

- Hilo con publicadores y suscriptores que le permite leer y enviar datos por los canales de comunicación de acuerdo a la tabla 5.6 presentada anteriormente.

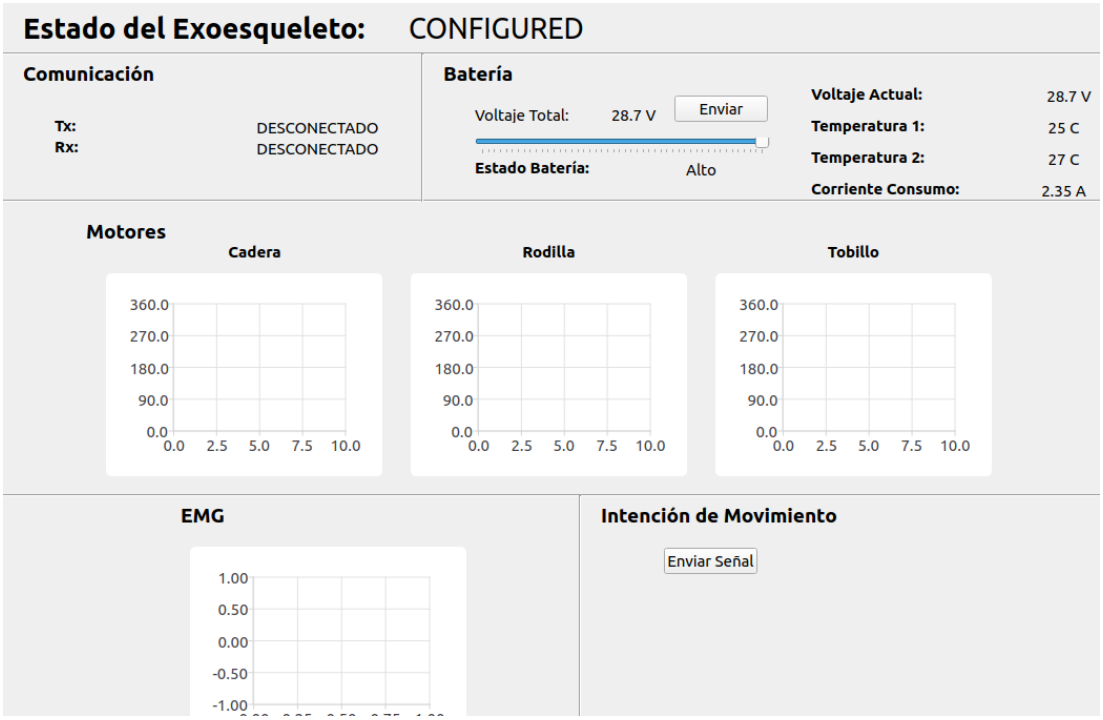


Figura 5.3: Interfaz gráfica para pruebas de *AllexControl*.

La interfaz gráfica provee al desarrollador de la aplicación las siguientes funcionalidades:

- *Estado del exoesqueleto:* Suscripción al tema *state* para conocer el estado general del exoesqueleto.
- *Comunicación:* Permite conocer el estado de conexión de los nodos de transmisión y recepción de datos mediante *bluetooth*.
- *Batería:* En el lado derecho de la pantalla muestra la información que es recolectada por los sensores de la batería como el voltaje, temperatura y corriente. Cuando el sistema funciona en ambiente de prueba se puede modificar el valor del voltaje para simular un evento de cambio del estado de la batería.
- *Motor:* Muestra gráficas de la posición angular de los motores de las articulaciones respecto al tiempo, en este caso solo de la pierna izquierda.
- *Intención de Movimiento:* Posee el botón *Enviar señal* que publica un evento *test_intention* para simular la llegada de un mensaje de intención de movimiento.

5.2. Implementación del Subsistema *AllexApp*

AllexApp fue desarrollado para dispositivos con sistema operativo *Android* usando las siguientes herramientas:

- **Entorno de Desarrollo:** *Android Studio* versión 3.3.1.
- **Lenguaje de Programación:** *Java* versión 1.8.
- **Compilación:** *Gradle* en la versión 4.1.0. Se encarga de construir el proyecto y manejar las dependencias. Como resultado de la ejecución de *gradle* se genera un archivo con extensión *apk* (*Android Package*). Los *apk* son usados para distribuir e instalar el *software* en un dispositivo con el sistema operativo *Android*.

5.2.1. Estructura de Directorios del Proyecto

Al momento de instanciar un nuevo proyecto en *Android Studio* se genera un sistema de archivos con una estructura similar a la que se muestra en la figura 5.4. Al primer nivel del directorio se lo conoce como el nivel de proyecto en el cual existen dos archivos creados por defecto y que se denominan *build.gradle* y *settings.gradle*. Particularmente el primer archivo es el único que se modifica en caso de que se requiera agregar nuevos repositorios para la descarga de las dependencias de la aplicación. Por ejemplo, en este proyecto se añadió el repositorio `https://jitpack.io` debido a que contiene una dependencia para librerías gráficas usadas en el proyecto.

A nivel de módulo se encuentra otro archivo de configuración para la cons-

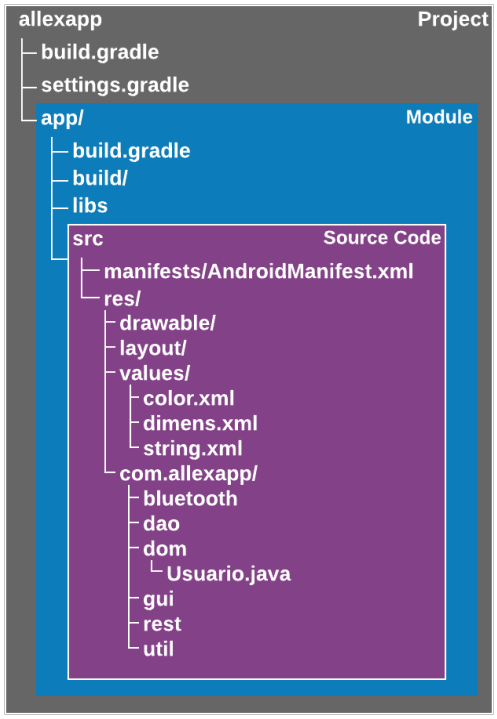


Figura 5.4: Estructura del proyecto *AllexApp*

trucción de la aplicación que también se denomina *build.gradle*. Este archivo se

diferencia del anterior en que permite definir información sobre el lenguaje de programación, versiones soportadas del sistema operativo y las dependencias a librerías externas. De acuerdo al ambiente operativo definido para el sistema (sección 3.2.4), la versión soportada de *Android* varía en el rango de la API 19 como mínimo y 28 como máximo. Las otras carpetas del mismo nivel son *build* y *libs* que almacenan los resultados de la construcción del proyecto y sus dependencias.

Finalmente, en el nivel de código fuente de la aplicación se encuentra el archivo *AndroidManifest.xml*. Este archivo permite configurar información que es requerida por el sistema antes de iniciar su ejecución como por ejemplo los permisos, características del sistema, características de las pantallas, etc. Parte de su estructura se presenta en el código 5.7 y contiene la siguiente información:

- Las etiquetas *user-permission* que se encuentran desde la línea 4 a la 9 en el código 5.7, corresponden a permisos que el usuario debe otorgar a la aplicación para que esta pueda acceder a los recursos de *hardware* o a sus datos. Los permisos *INTERNET* Y *ACCESS_NETWORK_STATE* le permiten a la aplicación crear *sockets* de comunicación por red o conocer el estado de la conexión a una red inalámbrica. Mientras que el resto de los permisos habilitan la búsqueda, emparejamiento y conexión con dispositivos *bluetooth*.
- En el elemento *application* se especifican los componentes de la aplicación. Uno de los atributos más importantes es el nombre de la aplicación que es definido en el atributo *android:appComponentFactory*.
- En el interior de la etiqueta *application* se definen los componentes de actividad (*activity*). Cada componente *activity* representa una interfaz gráfica de interacción con el usuario. La primera interfaz gráfica que se carga al abrir la aplicación es aquella que posee el atributo *android.intent.action.MAIN*.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools" package="com.allexapp">
4      <uses-permission android:name="android.permission.INTERNET"/>
5      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
6      <uses-permission android:name="android.permission.BLUETOOTH"/>
7      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
8      <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/
         >
9      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
10     <application android:name=".App" android:theme="@style/AppTheme"
11         android:appComponentFactory="@string/app_name">
12         <activity android:name=".gui.MainActivity" android:label="@string/app_name"
13             android:screenOrientation="landscape" android:theme="@style/AppTheme.
       NoActionBar">
14             <intent-filter>
15                 <action android:name="android.intent.action.MAIN"/>
16                 <category android:name="android.intent.category.LAUNCHER"/>
17             </intent-filter>
18         </activity>
```

```
19        <activity android:name=".gui.ControlActivity" android:screenOrientation="landscape"
20        android:theme="@style/AppTheme.NoActionBar"/>
21        <activity android:name=".gui.MenuActivity" android:screenOrientation="landscape"
22        android:theme="@style/AppTheme"/>
23    </application>
24 </manifest>
```

Código 5.7: Archivo *AndroidManifest.xml* de *AllexApp*.

Otros directorios en el nivel de código fuente son:

- En el directorio *com.allexapp* está el código de la aplicación definido por el usuario. El código ha sido definido en seis paquetes:
  - *bluetooth:* Define las clases básicas para la administración de la conexión, envío y recepción de datos mediante el protocolo *bluetooth*.
  - *dao:* Define las clases para el almacenamiento interno de la información en una base de datos. Esta base de datos es usada para almacenar temporalmente la posición angular de los motores.
  - *dom:* Posee la estructura de las estructuras de datos que se pueden obtener como respuesta del servidor y que fueron definidas en la sección 4.5.1.1.
  - *gui:* Directorio para la definición de clases para las interfaces gráficas. Esto incluye la definición de actividades y fragmentos.
  - *rest:* Posee funciones para el envío de peticiones al servidor de aplicaciones y procesamiento de su respuesta.
  - *util:* Directorio para las enumeraciones y funciones comunes a todos los paquetes como la navegación del sistema.
- El directorio *res* correspondiente a los archivos de recursos para la interfaz gráfica como colores, imágenes, textos, pantallas, etc. En la siguiente sección se presenta más información sobre los contenidos de este directorio.

5.2.2. Diseño de la Interfaz Gráfica

En el directorio *res* se definen elementos estáticos de la interfaz gráfica. En la carpeta *drawable* se guardan los iconos e imágenes de la aplicación. En el directorio *values* se encuentran archivos con pares clave y valor para definir colores, dimensiones de los componentes de la pantalla y cadenas de texto. Todos estos elementos pueden ser utilizados y referenciados desde la lógica de la interfaz gráfica.

El directorio *layout* es en donde se definen los archivos en formato *xml* para el diseño y estructura de la interfaz gráfica. Esto incluye elementos como botones, cuadros de texto, cuadros de notificación, etc. que son representados como etiquetas dentro de dicho archivo. *Android Studio* facilita la creación de estos archivos *xml* mediante un editor gráfico en el cual las pantallas se elaboran al arrastrar y soltar los componentes gráficos en la pantalla que se está diseñando. Posteriormente se pueden modificar las propiedades de cada uno, como por ejemplo las dimensiones, color, texto, sombra, entre otras.

Por defecto cada uno de estos componentes gráficos tienen dimensiones que son definidas de manera absoluta. Sin embargo, este comportamiento no es el ideal para adaptarlo a la resolución de la pantalla de diferentes dispositivos móviles. Por tanto, se recomienda el uso de los elementos *ConstraintLayout* y *Guideline* desde el inicio del diseño para especificar los límites de ancho y alto de los componentes visuales de la interfaz gráfica. De esta manera se diseña la interfaz mediante proporciones relativas en lugar de valores absolutos.

La figura 5.5 representa la estructura de la pantalla de control definida mediante *Guidelines*. Esta estructura corresponde a las líneas punteadas para los límites superior, inferior, izquierdo y derecho de los elementos gráficos. La desventaja de usar este estilo de diseño es que aumenta la complejidad de la estructura de la interfaz. Esto se debe a que los límites ocupan espacio dentro del código y todos los elementos gráficos deben tener referencias a los límites que los rodean. Como resultado aumenta la cantidad de código y se vuelve más difícil realizar cambios. Una solución a este problema es dividir a la interfaz en componentes individuales y después integrarlos en una sola pantalla. Los archivos de diseño no son ejecutables

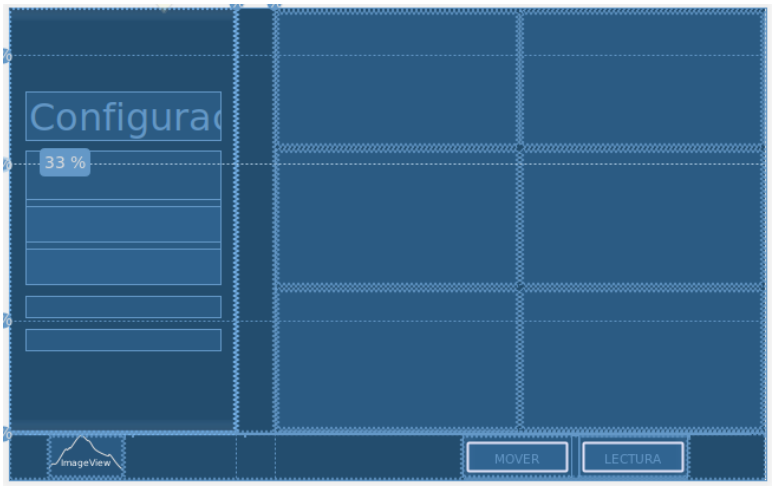


Figura 5.5: Vista *blueprint* de la pantalla de control.

por sí mismos. Deben ser referenciados por algún objeto de *java* que especifique la lógica de interacción con el usuario. Por este motivo se puede considerar que el desarrollo de aplicaciones en este sistema operativo sigue un patrón de diseño MVC (Modelo-Vista-Controlador). Los controladores de la interfaz gráfica pueden ser de dos tipos: la actividad (*Activity*) si es un elemento gráfico que ocupa toda la pantalla o fragmentos (*Fragment*) si corresponde a un conjunto de elementos gráficos dentro de un *Activity*. En este proyecto las actividades y fragmentos están definidos en el directorio *gui*.

5.2.3. Problema de la Navegación

Durante el flujo de ejecución del sistema se pueden cargar múltiples actividades y fragmentos dependiendo de las acciones ejecutadas por cada usuario. Cada actividad o fragmento requiere de un conjunto de parámetros para poder configurar la vista inicial de la pantalla y ejecutar su funcionalidad. Sin embargo, estos componentes pueden ser invocados desde diferentes partes de la aplicación, teniendo como consecuencia una redundancia en la definición de funciones de carga de pantallas. Por tanto, se creó la clase *Navigator* en el directorio *utils* en la que se crearon operaciones para administrar las transiciones entre las pantallas de manera centralizada. Como resultado, la implementación de la transición es transparente para las diferentes actividades o fragmentos que las invocan.

5.2.4. Almacenamiento de Datos

Como se indicó en la sección 4.6.1, *AllexApp* recibe un *token* o identificador único que está asociado a una sesión y a un usuario. Este identificador debe ser almacenado en una localización segura debido a que es usado para autorizar al cliente ante el servidor *AllexServer*. Para ello se usa el archivo de configuración de preferencias compartidas o *SharedPreferences* el cual permite almacenar pares de datos en una estructura clave valor y que únicamente es visible para la aplicación que los declara. Si se requiere usar el *token* en una llamada a servicio, se recupera esta clave desde el archivo de configuración. Esta información es eliminada únicamente cuando el usuario cierra su sesión. Se debe considerar que la aplicación no tiene garantías de preservar la seguridad del *token* si el dispositivo móvil tiene el sistema operativo con permisos de administrador (lo que generalmente se conoce como *rooting*).

Por otro lado, se requiere de almacenamiento persistente para la información de los datos de sensores de posición angular. *Android* provee diferentes mecanismos como el uso de archivos o pares de datos de tipo clave-valor. Sin embargo, el único mecanismo de almacenamiento estructurado que es provisto por defecto es *SQLite*. Las desventajas de interactuar con esta base de datos en bajo nivel es que se requiere código repetitivo para el acceso de datos, mantenimiento de las consultas, ausencia de validación de las consultas en tiempo de compilación y dificultad para realizar las transformaciones entre el modelo de datos de *SQLite* y los objetos de *Java*. Por tal motivo *Android* provee una capa de abstracción denominada *Room Persistence Library*. Esta librería permite un mapeo objeto-relacional que facilita el proceso de desarrollo de un componente de acceso a datos. Al mismo tiempo esta herramienta optimiza y provee seguridad al uso de la base de datos relacional. Por estos motivos se usó dicha implementación para los datos almacenados en la pantalla de control.

5.2.5. Ejecución y Depuración

La ejecución del sistema se realizó de dos maneras:

- Mediante un dispositivo virtual de *Android* (*AVD - Android Virtual Device*). Las pruebas en este dispositivo se realizaron para interactuar directamente con *AllexServer* en la misma máquina de desarrollo. De esta manera es más sencillo realizar correcciones al diseño de las interfaces e identificar errores.
- Mediante un dispositivo móvil en modo de desarrollador. El dispositivo físico se ha usado para las pruebas finales y la interacción con *AllexControl* debido a que no es posible simular el funcionamiento del *Bluetooth* desde el emulador.

5.3. Implementación del Subsistema *AllexServer*

Este subsistema fue desarrollado con las siguientes herramientas:

- **Lenguaje de Programación:** *Java* versión 1.8. Por tanto, puede ser ejecutado en cualquier sistema operativo que incluya la máquina virtual de *Java* (JVM).
- **Entorno Integrado de Desarrollo:** *Netbeans* en la versión 9.0.
- **Construcción:** Para la administración de las dependencias y construcción del proyecto se usó la versión 4.0.0 de *Maven*. Las dependencias se especifican en un archivo *xml* denominado *pom.xml*.

Para facilitar el desarrollo de la aplicación se usó el *framework Spring*. *Spring* es una plataforma de código abierto para el desarrollo y configuración de aplicaciones que se ejecutan en *Java*[2]. Está compuesta por un conjunto de módulos que proveen funcionalidades asociadas al manejo de datos, seguridad, servicios web, comunicación entre procesos y despliegue de aplicaciones. Para facilitar la configuración e integración de estos componentes se usa el módulo *Spring Boot* en la versión 2.0.0. Este módulo está incluido en *Netbeans* como un *plug-in* que se encarga de la configuración y ejecución del proyecto. Otros módulos de este *framework* que son usados en el proyecto son:

- *Spring Data JPA:* Facilita la conexión con la base de datos y el mapeo objeto relacional para el almacenamiento y recuperación de datos.
- *Spring Boot Security:* Permite autenticar a los usuarios del sistema y después autorizar su acceso a los recursos del servidor mediante el protocolo *OAuth 2*.
- *Spring Boot Web:* Permite exponer la funcionalidad del sistema mediante servicios *REST*.

[2]**Spring Framework:** https://spring.io/projects/spring-framework

Para el almacenamiento de los datos se usó la versión 11.1 de la base de datos relacional *PostgreSQL* a la cual se le agregó la extensión *Timescale* (versión 1.2) para la administración de datos de series de tiempo.

5.3.1. Estructura de Directorios del Proyecto

El *plugin NB SpringBoot*[3] es el encargado de crear la estructura inicial del proyecto mediante el servicio *Spring Initializr*. Este servicio provee un asistente (o *wizard*) para la creación del proyecto en el que se consultan cuáles son los módulos de *Spring* que se desean agregar en el proyecto. Como resultado se obtiene una estructura de directorios similar a la mostrada en la figura 5.6:

Por defecto a nivel de proyecto se crea el archivo *pom.xml* que incluye las

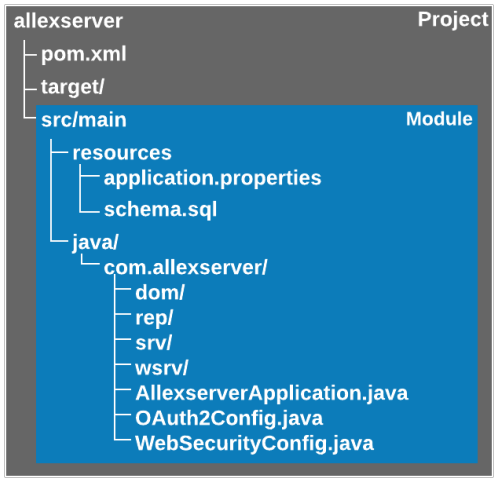


Figura 5.6: Estructura del proyecto *AllexServer*.

dependencias definidas en la herramienta *Spring Initializr*. En el directorio hermano *target* se genera el archivo *jar* para la ejecución del servidor de aplicaciones. El directorio *src/main* corresponde al nivel de módulo de la aplicación para el espacio de programación.

En el nivel de módulo existen dos directorios: *resources* y *java*. En *resources* se encuentran los archivos de configuración como:

- *application.properties:* Es un archivo como el que se presenta en el código 5.8 en el cual se definen parámetros de la aplicación como el nombre, puerto por el cual se proveen los servicios y la conexión con la base de datos.
- *schema.sql:* Un *script* en *Data Definition Language* (DDL) para la creación de las tablas requeridas para administrar la autorización y autenticación

[3]**NB SpringBoot:** *Plug-in* de *Spring Boot* para *Netbeans* encargado de la creación y configuración inicial de un proyecto. http://plugins.netbeans.org/plugin/67888/nb-springboot

del usuario como los *token* de autorización y contraseñas. Este archivo es leído cuando inicia la ejecución de la aplicación y crea las tablas si éstas no existen.

```
spring.application.name = allexserver
server.port=9091
spring.datasource.url= jdbc:postgresql://127.0.0.1:5432/allexdb
spring.datasource.platform=postgres
spring.datasource.username=****
spring.datasource.password=****
spring.jpa.properties.hibernate.default_schema = public
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.
    PhysicalNamingStrategyStandardImpl
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.use-new-id-generator-mappings=true
server.max-http-header-size=10000000
```

Código 5.8: Propiedades de la aplicación *AllexServer*.

En el directorio *java* se encuentran las clases que se definieron para la lógica de programación entre los servicios *REST* expuestos y su almacenamiento e interacción con la base de datos. El archivo *AllexserverApplication.java* es el encargado de iniciar la ejecución del proyecto y configurar todos los módulos a ser usados. Por otro lado, los archivos *OAuth2Config.java* y *WebSecurityConfig.java* configuran el módulo *Spring Security* para autenticar y autorizar a un usuario mediante el protocolo *OAuth 2*.

Como se indicó en la sección 4.5.2.1, la arquitectura del servidor es monolítica y separada por capas. Por este motivo la funcionalidad está dividida en tres paquetes:

- *rep:* El paquete de repositorios en el cual se definen las consultas para acceder a la información de la base de datos usando el módulo *Spring Data JPA*.
- *srv:* Paquete de servicios que está encargado de comunicarse con la capa de acceso a datos para recuperar o almacenar información de acuerdo a la lógica del servicio web.
- *wsrv:* Paquete que expone la estructura de los servicios web de *Allexserver* con su ruta, parámetros y cuerpo de la solicitud. En esta sección se verifica si el usuario está autorizado para acceder al servicio mediante su *token* o identificador en el protocolo *OAuth 2*.

5.3.2. Ejecución y Despliegue

Una aplicación desarrollada con *Spring Boot* puede ser configurada para tener un servidor de aplicaciones embebido. De esta manera no es necesario instalar el servidor en la máquina huésped. En el presente proyecto se usó el servidor *Apache Tomcat* en la versión 8.5.28.

Para generar el proyecto se usa el comando de *maven* presentado en el código 5.9. Durante su ejecución se descargan las dependencias definidas en el archivo *pom.xml* y genera un archivo comprimido en formato *.jar*.

```
$ mvn clean package
```

Código 5.9: Generación del proyecto *AllexServer*.

Finalmente se ejecuta *Allexserver* como una aplicación de java mediante el comando descrito en el código 5.10:

```
$ java -jar target/allexserver-0.0.1-SNAPSHOT.jar
```

Código 5.10: Comando de ejecución del proyecto *AllexServer*.

Como resultado, inicia la ejecución de la aplicación en el puerto 9091 del servidor como se observa en la figura 5.7. El sistema generará todas las tablas requeridas para su ejecución, el único requerimiento es que la máquina huésped incluya un gestor de bases de datos con la base de datos *allexdb* creada previamente.

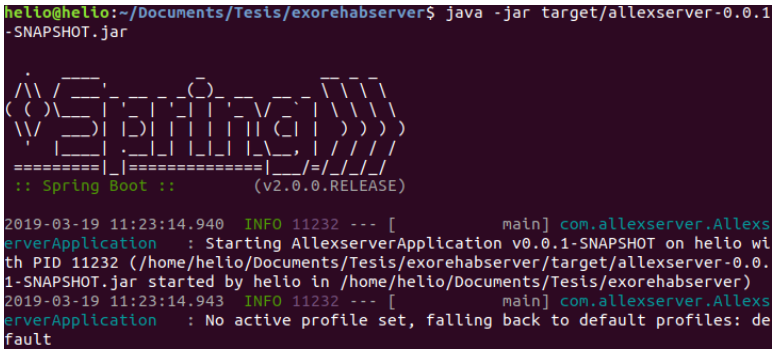


Figura 5.7: Despliegue de *AllexServer*.

5.4. Experiencias de Desarrollo

En esta sección se detallan diferentes problemas o retos suscitados durante el desarrollo del proyecto. Así como soluciones y recomendaciones que pueden ser de utilidad en la implementación de futuras versiones del sistema o del exoesqueleto.

- El tiempo de compilación de los componentes de *software* escritos en *C++* es alto debido a que se enlazan múltiples librerías asociadas a ROS2 y DDS. Particularmente el proceso de generación de archivos IDL para la comunicación por DDS es el más costoso. Esto es cierto sobre todo cuando se compila directamente en la *Raspberry Pi 3* debido a sus limitaciones de procesamiento. Por ejemplo, en una computadora portátil con un procesador Intel Core i7-3630M 2.4GHz y 12 GB de memoria RAM el proceso de compilación de todo el proyecto tarda un minuto, mientras que en la *Raspberry Pi 3* demora cerca de quince minutos. Si bien es cierto la solución inicial a este problema es la compilación cruzada, ésta no pudo ser aplicada debido a la existencia de dependencias de *python*. Adicionalmente las instrucciones para

la compilación cruzada en un contenedor de *Docker* estaban desactualizadas y no funcionaba con versiones recientes de ROS2. Por tanto, se recomienda realizar pruebas mediante simulaciones en la computadora de desarrollo y después realizarlo en la SBC.

- Una de las mayores desventajas de ROS2 en su versión *Crystal Clemmys* es que únicamente expone 4 de las 22 políticas DDS existentes. Esta situación no permite afinar algunas características de la comunicación en tiempo real sin que el usuario tenga que interactuar directamente con la implementación específica de un proveedor DDS, lo cual reduce la generalidad del sistema. Un ejemplo es el tiempo de vida (*lifespan*) de un mensaje, es decir, el tiempo durante el cual el suscriptor considera que el mensaje es válido, esto es útil en casos en los que se tolera la pérdida de datos y no se requiere tener información desactualizada. Otra política de interés es el tiempo límite (*deadline*), una característica que permite notificar a las aplicaciones cuando un mensaje no ha cumplido con su tiempo de llegada. Afortunadamente en [95] se diseña la funcionalidad para incorporar las políticas *lifespan*, *deadline* y *liveliness* en la cuarta versión de ROS2 *Dashing Diademata* a ser liberada el 31 de Mayo del 2019. Por tanto, se recomienda aplicar estas políticas cuando estén disponibles.

- El Dispositivo Virtual de Android (*Android Virtual Device* - AVD) no puede simular la funcionalidad del *Bluetooth*, por tanto toda evaluación de la comunicación y corrección de errores debe ser realizada desde un dispositivo físico en modo de desarrollo.

CAPÍTULO 6

Pruebas de Funcionamiento

En el presente capítulo se presenta la ejecución del sistema usando escenarios para los casos de uso que se presentan en la sección A. La descripción de cada uno de estos escenarios sigue la estructura que se presenta a continuación:

- *Objetivo:* Resumen del escenario a ser ejecutado.
- *Caso de uso relacionado:* Identificador de el o los casos de uso que son cubiertos por el escenario planteado. Estos se encuentran definidos en el anexo A.
- *Tareas:* Conjunto de actividades que deben ser ejecutadas para cumplir con el escenario planteado.
- *Resultado Esperado:* Detalla el estado o funcionalidad que deberá cumplir el sistema cuando se ejecutan las tareas del escenario.
- *Resultado Obtenido:* Detalla lo que sucedió en el sistema cuando un usuario ejecuta las tareas planteadas.

Para estas pruebas se considera que *AllexServer* se está ejecutando en una máquina en la misma red local que *AllexApp*. El módulo de autorización está activado, por tanto, se requieren permisos para acceder a los recursos de los servicios web. Por otro lado, el *Raspberry Pi 3* se encuentra conectado a la red *Control Area Network* (CAN) del exoesqueleto mediante el módulo PiCAN. El *Raspberry Pi 3* ejecuta el *software AllexControl* que fue desplegado con la configuración que se presenta en el anexo D.

6.1. Escenario de Creación de Pacientes

El usuario debe ingresar al sistema para crear un nuevo paciente.
Casos de uso relacionados: UC-1 (Autenticación de Usuarios - Tabla A.1), UC-2 (Gestión de la ficha médica del paciente - Tabla A.2).
Tareas:

1. Abrir la aplicación desde la pantalla de aplicaciones del dispositivo móvil. El nombre de la aplicación es *AllexApp*.
2. En la pantalla de ingreso, escribir la dirección de correo registrado y la contraseña asignada.
3. Presionar el botón *Ingresar* de la pantalla.
    a **Caso de Error 1:** La aplicación informa que las credenciales ingresadas son incorrectas. Revisar nuevamente si las credenciales están correctamente escritas y reintentar.
    b **Caso de Error 2:** No todos los campos han sido completados. Debe asegurarse de que los campos para el correo y contraseña tengan texto y reintente la acción.
4. En la pantalla del menú de la aplicación que se carga al completar el ingreso, seleccione la opción *Crear Paciente*.
5. En la pantalla *Creación de Pacientes*, ingrese su información personal para crear un nuevo paciente. En el campo *Diagnóstico del Paciente* puede escribir el texto *Apoplejía*.
6. Presionar el botón *Guardar* para terminar el proceso de creación del paciente.
    a **Caso de Error 1:** No todos los campos han sido completados. Debe asegurarse de que todos los campos marcados de color rojo tengan la información requerida y reintente la acción.
    b **Caso de Error 2:** El paciente ya está registrado en el sistema. Se considera que ya no es necesario continuar el escenario puesto que el objetivo ha sido cumplido previamente.

Resultado Esperado: Se presentan los datos ingresados en la ventana de información del paciente y un cuadro de diálogo que notifica que el paciente ha sido creado.
Resultado Obtenido: El usuario ha ingresado a la aplicación con las credenciales asignadas y ha creado un paciente con los datos del formulario como se muestra en la figura 6.1. De esta manera es posible crear sesiones de rehabilitación para dicho paciente.

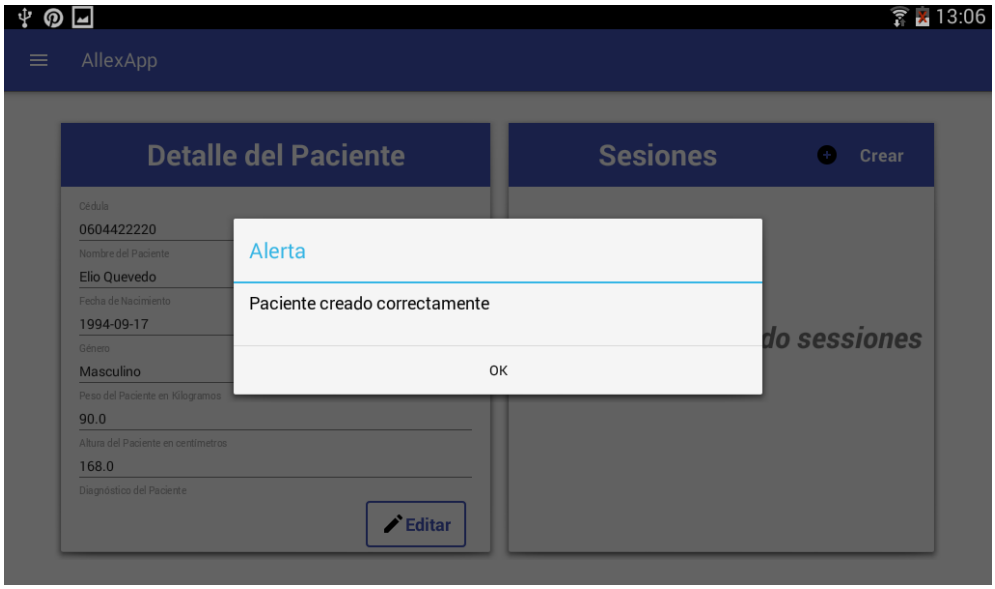


Figura 6.1: Notificación de que el paciente ha sido exitosamente registrado en el sistema.

6.2. Escenario de Creación de una Terapia de Rehabilitación

El usuario planifica una sesión de rehabilitación para el paciente creado en el escenario de la sección 6.1. Primero debe buscar la ficha médica del paciente, solicitar la creación de una nueva sesión de rehabilitación y gestionar la conexión con el exoesqueleto.
Casos de uso relacionados: UC-3 (Búsqueda de Pacientes - Tabla A.3), UC-5 (Crear una Sesión de Rehabilitación - Tabla A.5), UC-6 (Conexión con el exoesqueleto - Tabla A.6)
Tareas: Se debe considerar que el usuario inicia sus actividades desde el menú principal de la aplicación.

1. Seleccione la opción *Buscar Paciente* para iniciar el proceso de búsqueda.
2. En el cuadro de búsqueda, ingrese el número de identificación del paciente creado en el escenario 6.1.
3. Se carga una lista de pacientes que tienen un identificador igual o similar al buscado.
4. Presione el botón *Detalles* del paciente requerido, es decir, aquel que tiene un identificador igual al buscado.
5. Se carga la pantalla de información del paciente. En esta pantalla presione el botón *Crear* que se encuentra en el listado de sesiones.
6. Se despliega un cuadro de diálogo con un formulario para la creación de pacientes. En el cuadro de texto con título *nombre del tratamiento* ingrese la frase *Diagnóstico Inicial*. Por otro lado, en el cuadro de texto *descripción* ingrese la frase *Primera sesión de rehabilitación para el paciente*. Presione en el botón *Guardar* del cuadro de diálogo para iniciar el proceso de creación

de la sesión.

a **Caso de Error 1:** No todos los campos han sido completados. Debe asegurarse de que los campos para el título y la descripción tengan texto. Posteriormente reintente la acción.

7. En la pantalla de conexión por *bluetooth* que se carga a continuación presione en el botón *Encender*. De esta manera le otorga permisos a la aplicación para encender el módulo *bluetooth* del dispositivo móvil usado.
8. A continuación se presentan los dispositivos con los cuales es posible establecer la conexión. Presione en la opción *allex* para conectarse con el exoesqueleto.

a **Caso de Error 1:** La conexión ha fallado. En ese caso debe reintentar la acción o comprobar si el dispositivo de control está funcionando.

Resultado Esperado: El usuario ingresa en la pantalla de control del exoesqueleto y está asociado a una sesión de rehabilitación.
Resultado Obtenido: El usuario ha creado la sesión de rehabilitación con el título y descripción solicitada como se presenta en la figura 6.2. En este estado la sesión ha sido creada pero todavía no ha sido finalizada.

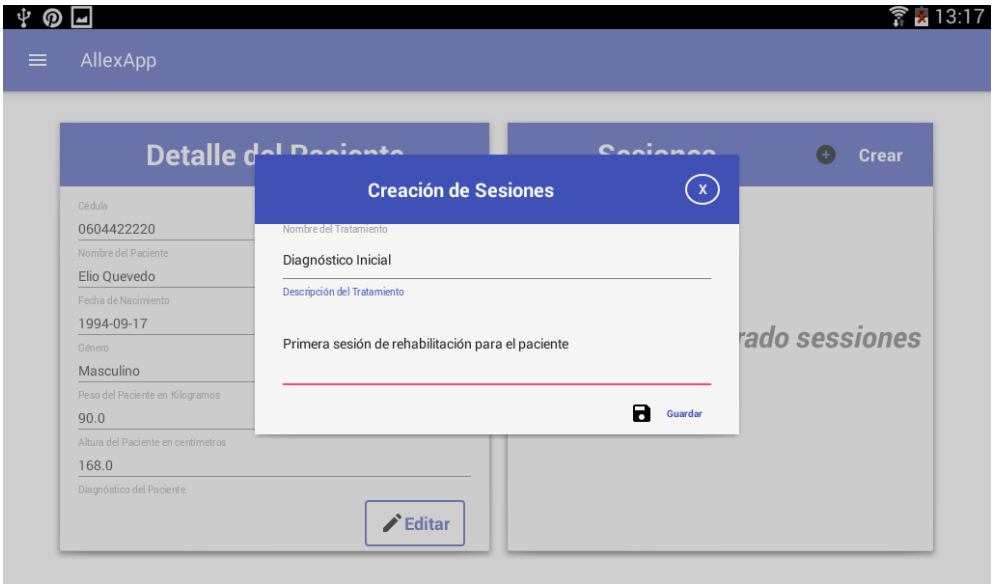


Figura 6.2: Creación de sesiones de rehabilitación.

6.3. Escenario de Lectura de Datos del Exoesqueleto

El usuario se encuentra en la pantalla de control del exoesqueleto y envía el comando para iniciar con el proceso de lectura de datos. El escenario termina cuando el usuario cancela el comando de lectura.

Casos de uso relacionados: UC-7 (Lectura de datos desde el exoesqueleto - Tabla A.7) y UC-8 (Cancelar el proceso de lectura de datos del exoesqueleto - Tabla A.8).
Tareas: Se debe considerar que el usuario inicia sus actividades desde el menú principal de la aplicación.

1. El usuario presiona el botón *Lectura* que está ubicado en la esquina inferior derecha de la pantalla de control.
2. Inicia el proceso de adquisición de datos.
3. A continuación, presionar el botón *Cancelar* para detener el proceso de adquisición de datos.

Resultado Esperado: El usuario puede visualizar en las gráficas temporales de la pantalla de control la posición angular de los motores. Cuando cancela la lectura de datos regresa al estado inicial de la pantalla de control.
Resultado Obtenido: El usuario puede observar la posición angular de los motores como se presenta en la figura 6.3. Sin embargo, la lectura siempre es cero para las articulaciones de la extremidad izquierda debido a que el mecanismo de control por posición de *EPOS4* no permite el desplazamiento de los motores cuando están activos como parte de su algoritmo de control.

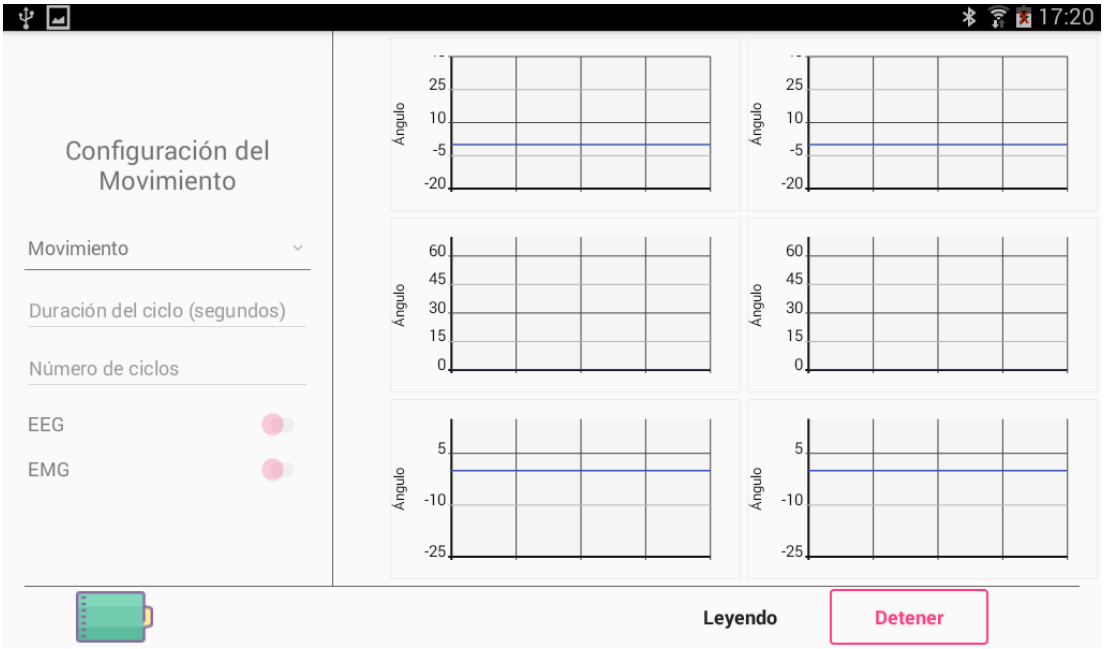


Figura 6.3: Ejecución del comando de lectura de datos en la pantalla de control.

Por otro lado, se usa el *software Wireshark* para filtrar los paquetes que son transmitidos dentro del controlador. Como se explicó en la sección 2.4.3.3 el protocolo de comunicación entre procesos que se usó es RTPS, particularmente la implementación *Fast-RTPS* que es provista por la empresa *eProsima*. Por tanto, se usa el siguiente filtro *rtps* para obtener únicamente dichos mensajes desde la interfaz *localhost* (*lo*) del dispositivo. Con la herramienta *Graph I/O* de Wireshark

se genera una gráfica en la que se representa el número de paquetes transmitidos por segundo como se presenta en la figura 6.4.

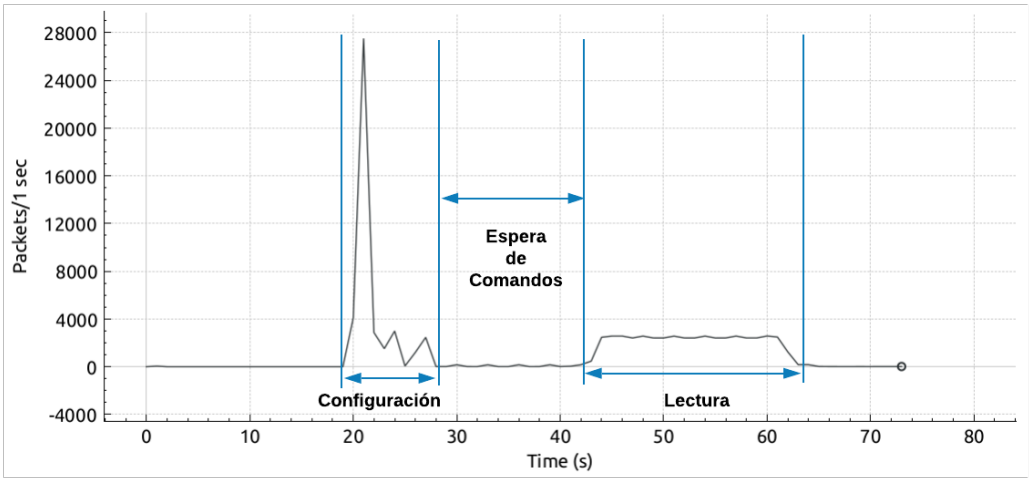


Figura 6.4: Paquetes RTPS transmitidos en el sistema. La información fue adquirida mediante el *software Wireshark*.

En esta gráfica se identifican tres etapas del funcionamiento de *AllexControl*:

- La primera sección corresponde a los mensajes de configuración de ROS2. Se puede observar que existe un pico máximo de 31755 paquetes transmitidos en un segundo. En esta sección todos los nodos acuerdan sobre los tipos de datos a ser enviados, el canal de comunicación, los publicadores y suscriptores, clientes, servicios y parámetros de configuración del sistema.
- La sección de espera corresponde a aquel tiempo en que la aplicación no realiza ninguna acción hasta recibir la conexión de un cliente *AllexApp* o un comando de un dispositivo conectado. En este tiempo se transmiten cerca de 43234 paquetes cada dos segundos. Estos son mensajes de tipo *Heartbeat* para informar a los suscriptores sobre los mensajes que tiene actualmente.
- La última sección representa la publicación de mensajes en el sistema. Aproximadamente se generan 71000 mensajes por segundo. Hasta que el usuario cancela el comando y regresa al estado de espera de comandos.

De esta manera se muestra que el comando enviado por la aplicación activa el proceso de adquisición de datos. Durante esta etapa se generan aproximadamente 0,35 *Megabytes* de paquetes por segundo y cada paquete tiene un tamaño entre 138 a 438 *bytes*. Por este motivo, el número de mensajes en la red durante el proceso de lectura de datos no es alto. Según [80], las latencias son aceptables y constantes puesto que el peso de cada estructura de datos es menor a 64 Kb.

6.4. Escenario de Ejecución de un Movimiento sin la Señal de Intención de Movimiento

El usuario se encuentra en la pantalla de control del exoesqueleto y envía el comando para iniciar el proceso de ejecución de un movimiento sin que la máquina de estados espere la señal de intención de movimiento. El escenario termina con la ejecución normal del movimiento.
Casos de uso relacionados: UC-9 (Ejecutar un movimiento en el exoesqueleto - Tabla A.9), UC-11 (Pantalla de previsualización de movimiento - Tabla A.11)
Tareas: Se debe considerar que el usuario inicia sus actividades en la pantalla de control del exoesqueleto.

1. El usuario coloca los siguientes parámetros de configuración para el movimiento:
   a **Tipo de movimiento:** Caminata.
   b **Número de iteraciones:** 3 iteraciones.
   c **Duración del ciclo:** 5 segundos.
2. Inicia el proceso de ejecución de movimiento al presionar en el botón *Movimiento*.
3. Esperar hasta que terminen los ciclos de marcha configurados y la pantalla de control permita el ingreso de otro comando.
4. Se visualiza la información de la sesión de rehabilitación en la pantalla de previsualización.

Resultado Esperado: Se ejecutan los ciclos de marcha y se visualizan sus resultados en las pantallas de previsualización.
Resultado Obtenido: El exoesqueleto inicia la ejecución de la secuencia de marcha como se observa en la figura 6.5 en la cual se presenta el exoesqueleto en movimiento en el lado izquierdo y una sección de las gráficas de la pantalla de control que marcan la posición angular de las articulaciones de la pierna izquierda.

Por otro lado, en la figura 6.6 se puede observar el comportamiento de la máquina de estados finitos para este escenario. El primer movimiento es *INITIAL* para llevar al exoesqueleto a una posición de inicio del movimiento cíclico. A continuación se ejecutan los tres ciclos de movimiento configurados en el movimiento *CYCLIC* y termina con el movimiento *STOP* para que el exoesqueleto regrese a la posición inicial de reposo.

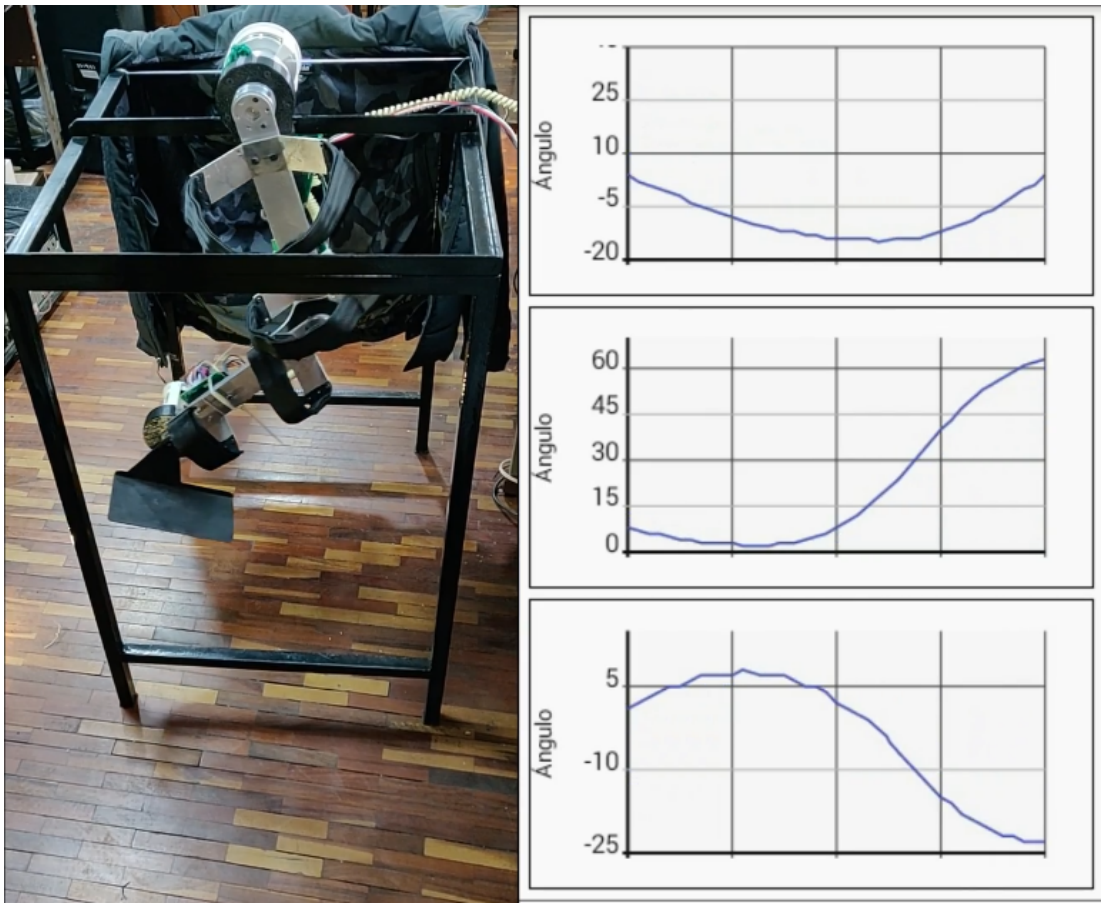


Figura 6.5: Ejecución de la trayectoria de movimiento en el exoesqueleto.

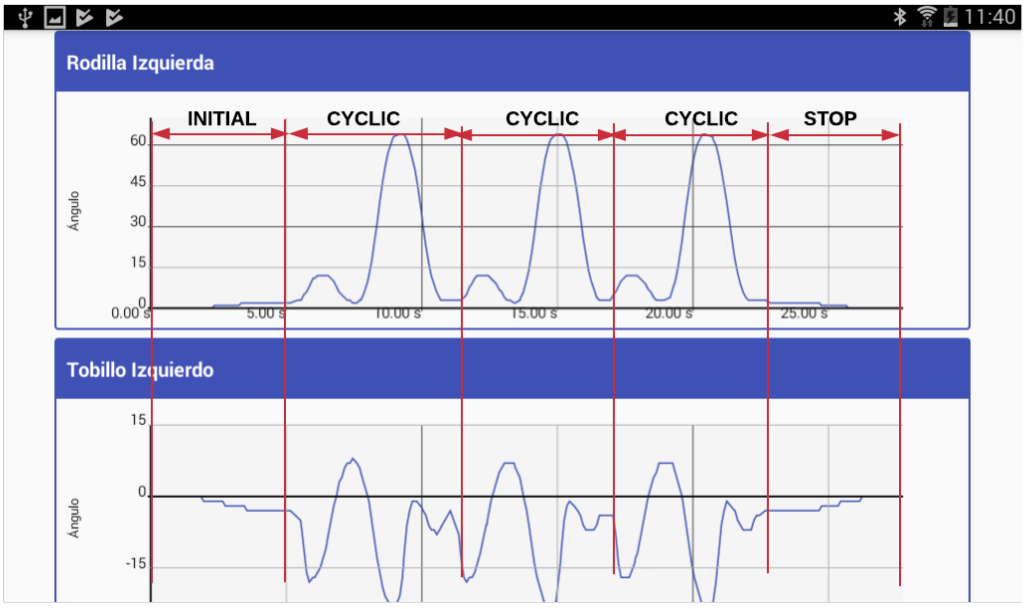


Figura 6.6: Trayectoria sin intención de movimiento.

6.5. Escenario de Ejecución de un Movimiento con la Señal de Intención de Movimiento

El usuario se encuentra en la pantalla de control del exoesqueleto y envía el comando para iniciar el proceso de ejecución de un movimiento con señales de intención de movimiento. El escenario termina con la cancelación del movimiento durante el tiempo de espera de una nueva señal.
Casos de uso relacionados: UC-9 (Ejecutar un movimiento en el exoesqueleto - Tabla A.9), UC-10 (Cancelar la ejecución del movimiento - Tabla A.10), UC-11 (Pantalla de previsualización de movimiento - Tabla A.11)
Tareas: Se debe considerar que el usuario inicia sus actividades en la pantalla de control del exoesqueleto.

1. El usuario coloca los siguientes parámetros de configuración para el movimiento:
    a **Tipo de movimiento:** Caminata.
    b **Número de iteraciones:** 3 iteraciones.
    c **Duración del ciclo:** 10 segundos.
    d **EEG:** Presionar el botón para activar la detección de intención por electromiografía.
2. Inicia el proceso de ejecución de movimiento al presionar en el botón *Movimiento*.
3. Usar la interfaz de pruebas presentada en la sección 5.1.10 para simular una señal de intención de movimiento y que inicie la ejecución del primer y segundo ciclo de marcha.
4. Presione el botón cancelar para que termine la ejecución del movimiento y no se realice el último ciclo de marcha.
5. Se visualiza la información de la sesión de rehabilitación en la pantalla de previsualización.

Resultado Esperado: Se ejecutan los ciclos de marcha y se visualizan sus resultados en las pantallas de previsualización.
Resultado Obtenido: En la figura 6.7 se presenta la trayectoria ejecutada por los motores de la rodilla izquierda al usar señales de intención de movimiento. Este comportamiento corresponde a la máquina de estados finitos del bloque de *Planificación* que se presentó en la sección 4.3.9. En este caso, el sistema espera una señal de intención de movimiento mediante el evento *INTENTION DETECTION*, período durante el cual el exoesqueleto no ejecuta ninguna acción. Una vez que se detecta la intención de movimiento, el exoesqueleto ejecuta las secuencias de movimiento *INITIAL* y *CYCLIC*. Si durante este tiempo no se reciben señales de intención de movimiento, el exoesqueleto ejecuta la trayectoria *STOP*. El proceso anteriormente descrito se repite una vez más para el segundo paso. Sin embargo, de acuerdo a las tareas planteadas en este escenario, el tercer paso no es ejecutado debido a que el usuario presiona el botón *Cancelar*.

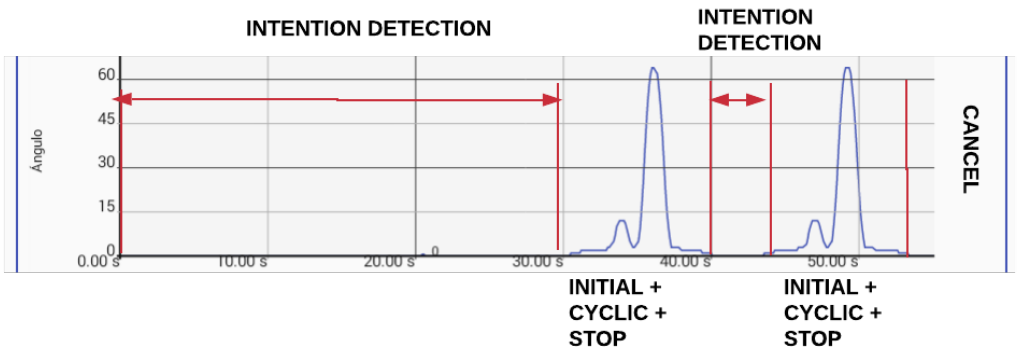


Figura 6.7: Trayectoria con intención de movimiento.

Capítulo 7

Conclusiones y Trabajos Futuros

En el presente capítulo se presentan las principales contribuciones que derivan del desarrollo de este trabajo de titulación (sección 7.1). Finalmente, se enuncian recomendaciones y trabajos futuros que se proponen para nuevas iteraciones de aquellos proyectos que estarán asociados al exoesqueleto y su sistema de control (sección 7.2).

7.1. Conclusiones

El desarrollo de *software* para sistemas robóticos presenta retos asociados a la variabilidad de *hardware* y *software*, manejo de sistemas en tiempo real, comunicación entre procesos, configuración y coordinación de componentes distribuidos. Particularmente, la dificultad de los sistemas robóticos de servicio está en la integración de componentes humanos, *hardware* y *software*. Estos retos no son ajenos al desarrollo de un sistema de control para un exoesqueleto de extremidades inferiores. En este tipo de sistemas se requiere que los diferentes sensores y actuadores puedan interactuar entre sí. Al mismo tiempo el sistema debe permitir el control de sus funcionalidades desde una interfaz para fisioterapeutas. Por tal motivo, en el presente proyecto de titulación se ha desarrollado un sistema de control centralizado para el exoesqueleto ALLEX-1 que permita coordinar el funcionamiento de los componentes de *hardware* y *software* y la interacción con el usuario desde una aplicación móvil. A continuación se presenta el grado de cumplimiento de los objetivos específicos que se plantearon en la sección 1.3.

7.1.1. Identificar los Requerimientos del Proyecto

Para cumplir este objetivo se realizaron reuniones con personas asociadas al proyecto *"Prototipo de exoesqueleto usable en las extremidades inferiores"*. En las cuales participaron docentes, investigadores y estudiantes de la carrera de Ingeniería Electrónica y Telecomunicaciones, así como, docentes de la carrera de Terapia Física de la Universidad de Cuenca. De esta manera fue posible identificar los requerimientos y funcionalidad que debe cumplir el sistema y, al mismo tiempo, limitar el alcance de los comportamientos que serían implementados.

Como resultado se propuso el desarrollo de tres subsistemas: 1) Un *software* embebido para la coordinación del *hardware* y *software* del exoesqueleto que se conoce como *AllexControl*, 2) una aplicación para dispositivos móviles que permite gestionar las sesiones de rehabilitación de un paciente que se denominó *AllexApp*, y 3) un servidor de aplicaciones para gestionar el acceso a los recursos de la aplicación, cuyo nombre es *AllexServer*. En esta etapa se documentaron los requerimientos de cada subsistema y se formalizó su descripción mediante casos de uso como los que se presentan en el Anexo A. Estos documentos permiten:

- Facilitar la transferencia de conocimientos sobre las funcionalidades de los subsistemas del sistema de control centralizado para ALLEX-1 usando un lenguaje natural.
- Facilitar el entendimiento de las funcionalidades durante el proceso de diseño de los subsistemas.
- Definir un mecanismo para evaluar el cumplimiento de las funcionalidades solicitadas por los interesados del proyecto.

7.1.2. Diseñar e Implementar un Sistema de Software Para el Control de los Componentes de un Exoesqueleto.

El *software* para el control de los componentes corresponde al subsistema *AllexControl*. El primer paso para su diseño consistió en identificar los componentes con los que interactúa. Estos corresponden a sensores como los de electromiografía, electroencefalografía, sistema de gestión de energía, sensor *hall* y *encoders* para los motores. Estos elementos son usados para diseñar un proceso que facilite la adquisición de datos o ejecución de comandos. A continuación, se realizó una investigación bibliográfica y de herramientas de desarrollo que permitan definir una infraestructura tecnológica para todo el sistema. De esta manera se identificó que los componentes del *software* para un sistema robótico requieren de un sistema operativo de tiempo real y un *middleware* o *framework* para gestionar la comunicación entre procesos y el desarrollo de componentes de *software* de aplicación. La infraestructura planteada provee los siguientes beneficios:

- *Extensibilidad:* Al usar un *middleware* es más fácil integrar nuevos componentes de *software* al sistema. Para ello solo es necesario definir las interfaces

requeridas y propuestas.
- *Portabilidad:* Es posible ejecutar el sistema en diferentes SBC que usen GNU/Linux como sistema operativo y que cumplan con las dependencias para el *middleware* y los controladores de *hardware*.
- *Reutilización:* Los componentes podrían ser usados en el desarrollo de otros sistemas robóticos.
- *Compatibilidad:* Los componentes del sistema se pueden comunicar entre sí mediante la definición de interfaces para el protocolo DDS. Esto es válido incluso entre proveedores diferentes de dicha tecnología.
- *Características de tiempo real:* Como asignación de prioridades a procesos y reserva de memoria RAM para evitar las latencias generadas por la paginación de datos que son enviados a la memoria virtual.

La arquitectura usada para el desarrollo se basa en componentes y sigue un patrón de arquitectura basada en eventos. Esta arquitectura permite que *AllexControl* sea un sistema con procesos distribuidos que mantiene su consistencia mediante el paso de eventos y mensajes. La ventaja es permitir la generación de componentes independientes pero que pueden ser integrados al sistema si se respetan las definiciones de las interfaces.

Las contribuciones y ventajas de este subsistema en el Sistema de Control Centralizado son:

- Integración de los componentes de *hardware* como los motores y por el momento las simulaciones para los módulos de Gestión de Energía, EMG y EEG.
- Facilidad de configuración, puesto que muchos de los parámetros de conexión y funcionamiento pueden ser especificados antes del despliegue del sistema y sin que sea necesario recompilar el código. Esto es útil para determinar cuáles nodos deben ser ejecutados o modificar la frecuencia de publicación de un sensor.
- Automatización del despliegue de procesos distribuidos a partir de un solo *script*.
- Definición de un ambiente de prueba para aquellos casos en los que no es posible interactuar con el *hardware*. Por ejemplo, para probar una nueva funcionalidad si parte del exoesqueleto está en mantenimiento o construcción.
- Coordinación centralizada de los componentes del sistema mediante un nodo orquestador. Este componente garantiza la consistencia de los elementos del sistema distribuido.

7.1.3. Diseñar e Implementar una Interfaz Hombre-Máquina que Permita la Interacción de los Usuarios, para Fines Terapéuticos, con un Exoesqueleto.

Este objetivo se cumple mediante el proceso de diseño e implementación de la funcionalidad de la aplicación móvil *AllexApp*. El primer paso para conseguir

este objetivo fue diseñar los prototipos para las interfaces gráficas y el flujo de navegación de las mismas. Una vez que el diseño ha sido aprobado se procedió a la implementación y pruebas de integración con *AllexControl* y *AllexServer*. Las contribuciones de este componente son:

- *Movilidad:* Considerando que *ALLEX-1* es un exoesqueleto móvil, se requiere que el dispositivo de supervisión pueda ser desplazado junto con el fisioterapeuta.
- *Facilidad de cambio:* Considerando que es un controlador de *software* es más sencillo realizar cambios a las funcionalidades y agregar nuevos comandos.

7.1.4. Evaluar la Solución Propuesta a través de Pruebas de Ejecución de Movimientos en un Exoesqueleto.

Este objetivo se cumplió al ejecutar los tres subsistemas en conjunto para cumplir con la ejecución de una sesión de rehabilitación. De esta manera se demuestra el cumplimiento de los casos de uso definidos en el presente proyecto.

7.2. Recomendaciones y Trabajos Futuros

La solución de *software* propuesta es el primer paso para un mecanismo de integración de componentes de *hardware* e interacción humano máquina. La funcionalidad de este sistema todavía puede ser extendida y optimizada en futuras versiones del exoesqueleto ALLEX-1. Por tanto, se redactan algunas recomendaciones para la arquitectura del exoesqueleto y nuevas funcionalidades del sistema:

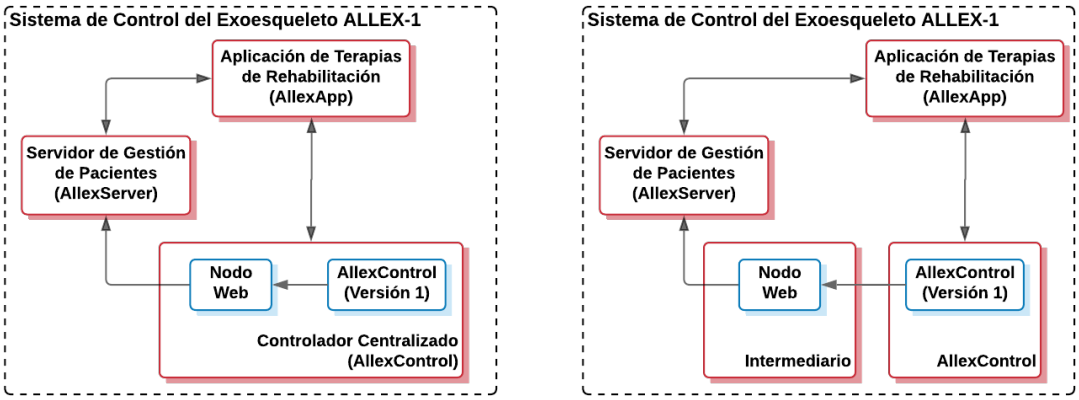
7.2.1. Establecer una Infraestructura Común para los Proyectos de Investigación

Es necesario definir una plataforma de desarrollo común para todos los proyectos de tal manera que en un futuro sea más fácil realizar su integración. Esto puede incluir decisiones sobre el lenguaje de programación o mecanismos de comunicación. Por ejemplo, se puede solicitar que la implementación de los algoritmos sea en *python* o *C++*, y si es posible, implementarlo como nodos de ROS2 que ofrecen interfaces para la comunicación de eventos y datos.

7.2.2. Almacenamiento de datos y eventos

No se recomienda realizar el almacenamiento de datos sobre el dispositivo que debe realizar el control en tiempo real, debido a que los procesos de escritura son procesos computacionalmente costosos y no deterministas. Tomando en cuenta la solución presentada en este documento se proponen dos mecanismos para extenderla y permitir el almacenamiento de datos. La primera opción se presenta en la figura 7.1a en donde se genera un nodo de ROS2 que está suscrito a los

temas que publican los datos que se deben almacenar. Posteriormente este nodo envía los datos hacia el servidor haciendo una llamada al servicio REST de almacenamiento de datos. Una segunda alternativa se presenta en 7.1b en el cual se usa un intermediario que también tiene un nodo de ROS2 pero que se encuentra en una máquina diferente, para después retransmitirlo al servidor. El primer escenario evita el uso de un elemento de *hardware* adicional en la estructura, sin embargo, aumenta la carga computacional en el nodo controlador. La segunda opción puede manejar de mejor manera los recursos a ser enviados.



(a) Comunicación directa con el servidor. (b) Uso de un intermediario.

Figura 7.1: Mecanismos de transmisión de datos a ser almacenados.

7.2.3. Evaluación del exoesqueleto al ser usado por personas sanas y pacientes patológicos

Consiste en realizar una evaluación de todo el exoesqueleto en potenciales usuarios para conocer si está de acuerdo con la propuesta. Esto se puede realizar mediante la evaluación *Quebec* de satisfacción de usuarios en relación a las tecnologías de asistencia.

ANEXO A

Casos de Uso

En el presente anexo se formalizan los requerimientos del sistema de control centralizado para extremidades inferiores de *ALLEX-1* mediante una especificación de casos de uso. Estos casos de uso están definidos desde el punto de vista de los usuarios de la aplicación móvil e integra el funcionamiento de los tres subsistemas como son *AllexApp*, *AllexControl* y *AllexServer*. Los casos están definidos mediante la estructura detallada a continuación:

- Un identificador de caso de uso con el prefijo "*UC*" (Use Case), seguido del título del caso de uso.
- Los requerimientos que son cubiertos por el caso de uso. Estos corresponden a los requerimientos funcionales descritos en la sección 3.3.
- Una descripción corta del objetivo del caso de uso.
- Actores que participan en el caso de uso de acuerdo a la lista de participantes del proyecto que se presenta en la sección 3.2.3.
- Las precondiciones se refieren al estado del sistema, pantalla o información registrada en el sistema y que es necesaria para el cumplimiento exitoso del caso de uso.
- El curso normal de eventos se refiere a la interacción entre el actor y el sistema de control centralizado que permite terminar exitosamente el caso de uso.
- Los cursos alternativos se refieren a otras formas de completar exitosamente el caso de uso.
- Las postcondiciones se refiere al estado del sistema después de la ejecución exitosa del caso de uso.
- Los casos de error se refieren a aquellas excepciones del sistema o los errores de usuario que no permiten que el caso de uso satisfaga las postcondiciones planteadas.

UC-1	**Autenticación de usuarios.**
Requerimientos	Corresponde a los requerimientos SRV-1 y APP-1
Descripción	El sistema permitirá que el usuario ingrese a la aplicación después de ingresar sus credenciales.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ El usuario debe encontrarse previamente registrado en el sistema. ▪ El usuario debe contar con las credenciales entregadas en el momento del registro.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	Inicia la ejecución de la aplicación.	El dispositivo muestra la pantalla de ingreso en la cual solicita las credenciales del usuario (correo y contraseña).
2	El usuario ingresa su correo electrónico y contraseña en el formulario. A continuación envía la solicitud de ingreso.	La aplicación valida las credenciales con *AllexServer*. En caso favorable almacena un *token* para realizar futuras solicitudes al servidor.

Postcondiciones	▪ El usuario recibe la autorización de entrar al sistema y a todos sus recursos. ▪ El usuario ingresa al menú principal de la aplicación.

Caso de error: Datos incompletos

No.	**Actor**	**Sistema**
2	Todos los datos del formulario de credenciales no han sido llenados.	El sistema notifica cuales son los campos faltantes y no realiza la petición al servidor.

Caso de error: Credenciales Incorrectas

No.	**Actor**	**Sistema**
2	El usuario ingresa un correo electrónico o contraseña incorrectos en el formulario. A continuación envía la solicitud de ingreso.	La aplicación valida las credenciales con *AllexServer*. La solicitud no se procesa y se notifica al usuario.

Tabla A.1: UC-1.- Autenticación de usuarios.

UC-2	**Gestión de la ficha médica del paciente.**
Requerimientos	Corresponde a los requerimientos SRV-2 y APP-2
Descripción	El sistema permite crear o editar la información personal de un paciente.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ El usuario está autorizado para acceder a los recursos del sistema.
Postcondiciones	▪ Se registra un nuevo paciente en el sistema. ▪ Se modifica la información de un paciente previamente registrado en el sistema.

Curso normal de eventos: Creación de la ficha médica

No.	**Actor**	**Sistema**
1	En el menú principal el usuario selecciona la opción para crear la ficha médica del paciente.	La aplicación carga una pantalla con el formulario de creación. Este formulario incluye ingresar el identificador del paciente, nombre, género, fecha de nacimiento, estatura, peso y diagnóstico médico.
2	El usuario ingresa los datos en el formulario y solicita la creación.	La aplicación valida si no existe un usuario con el mismo identificador en *AllexServer*. En caso favorable notifica al usuario y carga la pantalla de información del paciente.

Curso normal de eventos: Modificación de la ficha médica del paciente.

No.	Actor	Sistema
1	El usuario realiza el proceso de búsqueda de pacientes registrados en el sistema.	La aplicación muestra una lista de usuarios que concuerdan con el criterio de búsqueda.
2	El usuario selecciona de la lista, el paciente que desea modificar.	La aplicación solicita a *AllexServer* la información sobre el paciente seleccionado. En caso favorable carga la pantalla de información del paciente.
3	El usuario selecciona la opción de editar la información del paciente.	La aplicación crea un formulario editable con la excepción del identificador del paciente.
4	El usuario realiza las modificaciones en el formulario. Al terminar presiona el botón guardar.	La aplicación solicita a *AllexServer* que modifique la información del paciente.

Postcondiciones	▪ La ficha médica del paciente es creada en *AllexServer*. ▪ El usuario ingresa a la pantalla de la ficha médica del paciente.

Caso de error: Datos incompletos.

No.	Actor	Sistema
2	Todos los datos del formulario de de creación de la ficha médica no han sido llenados.	El sistema notifica cuales son los campos faltantes y no realiza la petición al servidor hasta que el usuario complete el formulario.

Caso de error: La ficha médica del paciente ya ha sido creada.

No.	Actor	Sistema
2	El usuario ingresa un identificador de paciente que ya ha sido creado previamente.	La aplicación al validar la información con *AllexServer*, recibe como respuesta que el usuario ya ha sido creado previamente. La solicitud no se procesa y se notifica al usuario.

Caso de error: El usuario cancela la creación del paciente.

No.	Actor	Sistema
2	El usuario cancela el proceso de creación del paciente.	La aplicación regresa al menú principal.

Caso de error: El usuario cancela la edición del paciente.

No.	Actor	Sistema
3	El usuario cancela el proceso de creación del paciente.	La aplicación no realiza la solicitud y elimina en la pantalla los cambios realizados por el usuario.

Tabla A.2: UC-2.- Gestión de la ficha médica del paciente.

UC-3	**Búsqueda de pacientes.**
Requerimientos	Corresponde a los requerimientos SRV-2 y APP-3
Descripción	Cargar la información de un paciente después de un proceso de búsqueda y listado usando como criterio el número de identificación o nombre.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ El usuario está autorizado para acceder a los recursos del sistema. ▪ El usuario está en la pantalla del menú principal.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario ingresa a la pantalla de búsqueda de pacientes.	El sistema carga el formulario de búsqueda con un campo para realizar la consulta.
2	El usuario escribe el identificador del paciente del cual requiere crear sesiones.	El sistema envía la solicitud a *AllexServer* para que responda con una lista de pacientes que cumplan con dicho criterio de búsqueda. Si la petición es exitosa, se muestra un listado de alternativas que cumplen con el criterio. La información del paciente en este listado incluye su identificación, nombre, fecha de nacimiento y género.
3	El usuario busca en la lista y encuentra la información del paciente requerido. A continuación presiona en el botón "*Detalles*" para obtener información sobre dicho paciente.	El sistema envía la solicitud a *AllexServer* para obtener la información del paciente seleccionado. Si la petición es exitosa se carga la pantalla de información del paciente.

Postcondiciones	▪ El usuario visualiza la pantalla de información del usuario.

Caso de error: No se encuentran coincidencias.

No.	Actor	Sistema
2	El usuario escribe el identificador del paciente (cédula o pasaporte) del cual requiere crear sesiones.	Se realiza la consulta al servidor pero no se han encontrado coincidencias. Se notifica al usuario que no hay información de pacientes que estén relacionados con el criterio de búsqueda.

Tabla A.3: UC-3.- Búsqueda de pacientes.

UC-4	**Listado de sesiones de rehabilitación.**
Requerimientos	Corresponde a los requerimientos SRV-3 y APP-15.
Descripción	Listar las sesiones de rehabilitación que están asociadas a un paciente.
Actores	ACT-2 (Fisioterapeuta)
Precondiciones	▪ El usuario está autorizado para acceder a los recursos del sistema. ▪ El usuario ha realizado el proceso de búsqueda de un paciente de acuerdo al caso de uso CU-3.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario ha realizado el proceso de búsqueda y ha seleccionado un paciente del cual quiere obtener la información.	La aplicación realiza una solicitud para *AllexServer* a partir de la cual obtiene la información de la ficha médica del paciente.
2		El sistema envía una solicitud a *AllexServer* para obtener las sesiones de rehabilitación asociadas a dicho paciente. Esto incluye una descripción del tratamiento, la fecha de ejecución de la sesión y el estado, es decir, si fue completada o cancelada.

Postcondiciones	▪ El usuario visualiza la pantalla de información del usuario con la información de las sesiones de rehabilitación realizadas.

Caso de error: No se encuentran coincidencias.

No.	**Actor**	**Sistema**
2		El paciente no tiene sesiones de rehabilitación registradas, por tanto no se muestra ningún objeto en el listado de sesiones de rehabilitación.

Tabla A.4: UC-4.- Listado de sesiones de rehabilitación.

UC-5	**Crear una sesión de rehabilitación.**
Requerimientos	Corresponde a los requerimientos SRV-3 y APP-4.
Descripción	Registra una nueva sesión de rehabilitación en la que se puedan almacenar los datos de posición angular del ejercicio de rehabilitación.
Actores	ACT-2 (Fisioterapeuta)
Precondiciones	▪ El usuario está autorizado para acceder a los recursos del sistema. ▪ El usuario ha realizado el proceso de búsqueda de un paciente de acuerdo al caso de uso CU-4.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario se encuentra en la pantalla de información de pacientes y presiona el botón para "*Crear*" una nueva sesión.	El sistema muestra un formulario en con dos campos: 1) para otorgar un título al tratamiento y otro para que pueda escribir una descripción más amplia.
2	El usuario llena el formulario y presiona el botón "*Crear*" para confirmar la creación de una sesión con dichos datos.	El sistema envía una solicitud a *AllexServer* para crear la sesión de rehabilitación. En caso exitoso crea en el servidor un identificador para dicha sesión. Como resultado la aplicación móvil carga la pantalla de conexión con el exoesqueleto.

Postcondiciones	▪ La aplicación obtiene el identificador de la sesión y carga la pantalla de conexión con el exoesqueleto.

Tabla A.5: UC-5.- Crear una sesión de rehabilitación.

UC-6	**Conexión con el exoesqueleto.**
Requerimientos	Corresponde a los requerimientos APP-5, APP-12, APP-13, CTR-1, CTR-7 y CTR-8.
Descripción	Establece una conexión de comunicación mediante *bluetooth* entre *AllexServer* y *AllexApp*.
Actores	ACT-2 (Fisioterapeuta)
Precondiciones	▪ El dispositivo móvil cuenta con *hardware* funcional para la comunicación mediante *bluetooth*.

Curso normal de eventos

No.	**Actor**	**Sistema**
1		Se detecta que la funcionalidad *bluetooth* no está activada, por tanto se carga la pantalla de conexión en la cual se solicita al usuario el permiso para activar la conexión.
2	El usuario aprueba los permisos para activar la funcionalidad *bluetooth*.	La aplicación solicita al sistema operativo que active la funcionalidad de *bluetooth* en el dispositivo. A continuación, se carga la pantalla de conexión en la cual se listan los dispositivos con los cuales ha sido previamente emparejado.
3	El usuario selecciona de la lista el dispositivo cuyo nombre es "*allex*" para iniciar la conexión	*AllexApp* inicia el protocolo para activar un canal de comunicación con *AllexControl*.

Caso alternativo: La funcionalidad *bluetooth* ya está activada.

No.	**Actor**	**Sistema**
1	El usuario activa la funcionalidad *bluetooth* desde las opciones del sistema operativo.	Se detecta que la funcionalidad *bluetooth* ya ha sido activada por el usuario. Por tanto, se carga la pantalla de conexión en la cual se listan los dispositivos con los cuales ha sido previamente emparejado.
2	El usuario selecciona de la lista el dispositivo cuyo nombre es "*allex*" para iniciar la conexión	*AllexApp* inicia el protocolo para activar un canal de comunicación con *AllexControl*.

Postcondiciones	▪ La aplicación crea un canal de comunicación *bluetooth* y carga la pantalla de control del exoesqueleto.

Caso de error: Desconexión intencional de la funcionalidad *bluetooth*.

No.	Actor	Sistema
2	El usuario desactiva la funcionalidad *bluetooth* desde las opciones del sistema operativo.	*AllexApp* regresa a la pantalla para solicitar al usuario los permisos de conexión.

Caso de error: Falla la creación del canal de comunicación.

No.	Actor	Sistema
3	El usuario selecciona de la lista el dispositivo cuyo nombre es "*allex*" para iniciar la conexión.	Después de doce segundos la conexión no se establece y por tanto se detienen los intentos. Se solicita al usuario que reintente la conexión.

Caso de error: Desconexión después de cargar la pantalla de control.

No.	Actor	Sistema
3		El sistema detecta que la funcionalidad *bluetooth* ha sido desactivada después de que la conexión haya sido exitosa o que la desconexión proviene desde *AllexControl*. En ese caso retorna a la pantalla de conexión por *bluetooth*.

Tabla A.6: UC-6.- Conexión con el exoesqueleto.

UC-7	**Lectura de datos desde el exoesqueleto.**
Requerimientos	Corresponde a los requerimientos APP-6, APP-7, APP-13 y CTR-2
Descripción	El usuario solicita a *AllexControl* que modifique su estado interno e inicie el proceso de lectura de datos.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ *AllexApp* tiene un canal de comunicación *bluetooth* con *AllexControl*. ▪ El usuario está en la pantalla de control. ▪ Ningún comando del exoesqueleto está en ejecución.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario solicita el inicio del proceso de lectura de datos desde el exoesqueleto.	*AllexApp* envía a *AllexControl* una solicitud para iniciar el proceso de lectura de datos.
2	El usuario no puede realizar ninguna acción.	*AllexApp* espera la confirmación de la ejecución del comando enviado al exoesqueleto.
3		*AllexApp* recibe la confirmación de *AllexControl* e inicia la recepción de datos a través de *bluetooth*.

Postcondiciones	▪ El exoesqueleto se encuentra en el estado de lectura de datos. ▪ Se recibe la información de batería. ▪ Se reciben y dibujan los datos de posición angular de los motores.

Caso de error: Fallo del comando.

No.	**Actor**	**Sistema**
2		*AllexControl* notifica que la configuración en el sistema ha fallado. Se cancela la petición y el usuario puede solicitar un nuevo comando.

Tabla A.7: UC-7.- Lectura de datos desde el exoesqueleto.

UC-8	**Cancelar el proceso de lectura de datos del exoesqueleto.**
Requerimientos	Corresponde a los requerimientos APP-8, APP-13 y CTR-3
Descripción	El usuario solicita a *AllexControl* que detenga el proceso de lectura de datos.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ *AllexApp* tiene un canal de comunicación *bluetooth* con *AllexControl*. ▪ El usuario está en la pantalla de control. ▪ Está en ejecución el comando de lectura del exoesqueleto.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario solicita la cancelación del comando de lectura.	*AllexApp* envía a *AllexControl* una solicitud para terminar el proceso de lectura de datos.
2	El usuario no puede realizar ninguna acción hasta recibir una confirmación.	*AllexApp* espera la confirmación de cancelación del comando enviado al exoesqueleto.
3		*AllexApp* recibe la confirmación de *AllexControl* y regresa a la pantalla de control para recibir nuevos comandos.

Postcondiciones	▪ El exoesqueleto no realiza ninguna actividad y el usuario puede enviar un nuevo comando.

Caso de error: Fallo del comando.

No.	**Actor**	**Sistema**
2		*AllexControl* notifica que la configuración en el sistema ha fallado. Se cancela la petición y el usuario puede solicitar un nuevo comando.

Tabla A.8: UC-8.- Cancelar el proceso de lectura de datos del exoesqueleto.

UC-9	**Ejecutar un movimiento en el exoesqueleto.**
Requerimientos	Corresponde a los requerimientos APP-10, APP-11, APP-13, CTR-2, CTR-4 y CTR-6
Descripción	El usuario configura y solicita a *AllexControl* que ejecute la secuencia de movimientos de rehabilitación.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ *AllexApp* tiene un canal de comunicación *bluetooth* con *AllexControl*. ▪ El usuario está en la pantalla de control. ▪ Ningún comando del exoesqueleto está en ejecución.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario utiliza el formulario de configuración de movimiento para elegir el tipo de movimiento a ser ejecutado (actualmente caminata), la duración del ciclo de movimiento y el número de veces que debe repetirse el ciclo de caminata. De manera opcional el usuario puede activar la detección de intención de movimiento ya sea mediante señales de electromiografía o electroencefalografía.	*AllexApp* valida la configuración y la envía a *AllexControl*. No se puede ejecutar ningún comando hasta recibir una confirmación de la configuración.
2	El usuario no puede realizar ninguna acción.	*AllexApp* espera la confirmación de la ejecución del comando enviado al exoesqueleto.
3		*AllexApp* recibe la confirmación de *AllexControl* e inicia la recepción de datos y eventos a través de *bluetooth*. Durante la ejecución del movimiento el único comando que puede enviar es el de cancelar el movimiento.
4	El usuario espera hasta que termine la rutina de movimiento.	*AllexApp* recibe la confirmación de que el movimiento ha terminado. Se carga la pantalla de previsualización de la terapia de rehabilitación.

Postcondiciones	▪ El exoesqueleto inicia la ejecución del movimiento configurado hasta que termina toda la secuencia. ▪ Se carga la pantalla de previsualización de la terapia de rehabilitación.

Caso de error: Configuración incorrecta.

No.	Actor	Sistema
2	El usuario no ha llenado todos los campos del formulario de configuración.	*AllexApp* muestra una notificación de error en la que alerta que deben completarse todos los campos para enviar la configuración al exoesqueleto.

Caso de error: Fallo del comando.

No.	Actor	Sistema
2		*AllexControl* notifica que la configuración en el sistema ha fallado. Se cancela la petición y el usuario puede solicitar un nuevo comando.

Caso de error: Fallo de la conexión.

No.	Actor	Sistema
2		*AllexApp* regresa a la pantalla de conexión y se cancela el movimiento en *AllexControl*.

Tabla A.9: UC-9.- Ejecutar un movimiento en el exoesqueleto.

UC-10	**Cancelar la ejecución del movimiento.**
Requerimientos	Corresponde a los requerimientos APP-11, APP-13 y CTR-6
Descripción	El usuario configura y solicita a *AllexControl* que cancele el movimiento que está siendo ejecutado.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ *AllexApp* tiene un canal de comunicación *bluetooth* con *AllexControl*. ▪ El usuario está en la pantalla de control. ▪ El exoesqueleto está ejecutando un ejercicio de rehabilitación.

Curso normal de eventos

No.	Actor	Sistema
1	El usuario presiona el botón para cancelar el movimiento.	*AllexApp* envía la solicitud a *AllexControl*. No se puede ejecutar ningún comando hasta recibir una confirmación del cambio de estado.
2	El usuario no puede realizar ninguna acción.	*AllexApp* espera la confirmación de la cancelación de la ejecución del movimiento.
3		*AllexApp* recibe la confirmación de *AllexControl* y espera a que llegue la notificación de culminación del movimiento. Mientras tanto, el usuario no puede ejecutar ninguna acción.

Postcondiciones	▪ El exoesqueleto termina la ejecución del movimiento configurado. ▪ Se carga la pantalla de previsualización de la terapia de rehabilitación.

Tabla A.10: UC-10.- Cancelar la ejecución del movimiento.

UC-11	**Pantalla de previsualización de movimiento.**
Requerimientos	Corresponde a los requerimientos APP-14 y SRV-4
Descripción	El usuario solicita a *AllexControl* que cancele el movimiento que está siendo ejecutado.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ *AllexApp* tiene un canal de comunicación *bluetooth* con *AllexControl*. ▪ *AllexApp* cuenta con conexión a internet o a una red local. ▪ El usuario está en la pantalla de control. ▪ El exoesqueleto terminó la ejecución de un ejercicio de rehabilitación.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario visualiza las gráficas de posición angular respecto al tiempo de cada articulación. Estas imágenes pueden ser ampliadas, reducidas o desplazadas en el dominio de la gráfica.	*AllexApp* modifica las gráficas de acuerdo a los gestos del usuario sobre la pantalla de las gráficas.
2	El usuario presiona el botón *guardar* para iniciar el proceso de almacenamiento de datos.	*AllexApp* envía las posiciones angulares de los motores a *AllexServer* para su almacenamiento.

Postcondiciones	▪ La posición angular de los motores es almacenada en *AllexServer* ▪ El usuario regresa a la pantalla de información del paciente.

Caso de error: Cancelar el almacenamiento de datos.

No.	**Actor**	**Sistema**
1	El usuario presiona el botón *cancelar* en la pantalla de previsualización.	*AllexApp* no almacena los datos y regresa a la pantalla de control.

Tabla A.11: UC-11.- Pantalla de previsualización de movimiento.

UC-12	**Pantalla de datos históricos de la sesión de rehabilitación.**
Requerimientos	Corresponde a los requerimientos APP-16, APP-17, SRV-3 y SRV-4
Descripción	El usuario solicita la información relacionada a una sesión de rehabilitación terminada y cuyos datos hayan sido almacenados.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ El usuario está autorizado para acceder a los recursos del sistema. ▪ El usuario se encuentra en la pantalla de información del paciente.

Curso normal de eventos

No.	Actor	Sistema
1	El usuario se encuentra en la pantalla de información del paciente.	*AllexApp* solicita a *AllexServer* el listado de sesiones de rehabilitación asociadas al paciente.
2	El usuario presiona un botón para solicitar los detalles de una de las sesiones de rehabilitación listadas.	*AllexApp* realiza una solicitud a *AllexServer* para obtener la información de la sesión de rehabilitación solicitada.
3	El usuario visualiza las gráficas de posición angular respecto al tiempo de cada articulación. Estas imágenes pueden ser ampliadas, reducidas o desplazadas en el dominio de la gráfica.	*AllexApp* modifica las gráficas de acuerdo a los gestos del usuario sobre la pantalla de las gráficas.
4	El usuario puede agregar un texto para el diagnóstico funcional del paciente.	*AllexApp* envía el texto del diagnóstico funcional a *AllexServer* para su almacenamiento.

Postcondiciones	▪ El usuario visualiza las gráficas de posición angular respecto al tiempo de cada articulación de una sesión previamente finalizada. ▪ El usuario agrega un texto para el diagnóstico funcional del paciente.

Tabla A.12: UC-12.- Pantalla de datos históricos de la sesión de rehabilitación.

UC-13	**Cerrar la sesión de usuario.**
Requerimientos	Corresponde a los requerimientos APP-18 y SRV-1
Descripción	El usuario cierra su sesión en la aplicación.
Actores	ACT-1 (Desarrolladores), ACT-2 (Fisioterapeuta) y ACT-3 (Personal técnico)
Precondiciones	▪ El usuario está autorizado para acceder a los recursos del sistema.

Curso normal de eventos

No.	**Actor**	**Sistema**
1	El usuario regresa al menú principal del sistema.	*AllexApp* carga la pantalla para el menú principal de la aplicación.
2	El usuario presiona la opción para cerrar la sesión de usuario.	*AllexApp* informa al servidor y elimina las credenciales de acceso.

Postcondiciones	▪ El usuario ya no tiene autorización para acceder a los recursos del sistema.

Tabla A.13: UC-13.- Cerrar la sesión de usuario.

ANEXO B

Definición de Servicios Web

En el presente anexo se describen los servicios web que son provistos por *AllexServer* para su interacción con un cliente que ejecuta *AllexApp*. Las tablas para la definición de servicios detallan los campos requeridos para realizar la solicitud HTTP así como la respuesta obtenida. Estas tablas tienen la estructura que se presenta a continuación:

- El campo *ruta del recurso* (*Resource Path*) corresponde a una plantilla de aquella sección de una URI que se encuentra entre entre el nombre del dominio y los parámetros de consulta. Su semántica permite identificar con qué recurso interactúa el servicio web.
- El *método* es un verbo del protocolo HTTP cuya semántica indica que tipo de acción se va a realiza sobre el recurso. Ejemplos de estos métodos son GET, POST, PUT y DELETE.
- El campo *descripción* resume la funcionalidad que es provista por el servicio web.
- El campo *autorización* determina si el cliente puede acceder al recurso solicitado al analizar el *token* presente en la cabecera de la petición HTTP.
- El campo *parámetro de la ruta* permite generar rutas dinámicas al colocar un valor en los parámetros que se encuentran entre llaves en la plantilla de la ruta del recurso.
- Los *parámetros de consulta* son usados para filtrar los recursos solicitados.
- Los *parámetros del cuerpo* corresponde a aquel conjunto de datos que son enviados en el cuerpo del mensaje HTTP. En este caso se usa el formato application/x-www-form-urlencoded, es decir que los datos son codificados como tuplas de clave y valor.
- El campo *respuesta* indica los datos que son enviados como resultado del procesamiento de la solicitud HTTP. En este caso la respuesta es enviada en formato JSON.

B.1. Gestión del Usuario

Ruta	**/users**
Método	POST
Descripción	Permite la creación de un usuario que puede ingresar desde *AllexApp*.
Autorización	✗
Parámetros de la Ruta	✗
Parámetros de Consulta	✗
Parámetros del Cuerpo	*username:* String que representa el nombre de usuario. *email:* String que representa el correo electrónico del usuario. *password:* String que representa la contraseña del usuario.
Respuesta	Una estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si el usuario fue creado. Caso contrario, el valor es *False*.

Tabla B.1: Servicio web para creación de usuarios

B.2. Gestión de Pacientes

Ruta	**/patients**
Método	POST
Descripción	Permite la creación de un paciente en la aplicación.
Autorización	✓
Parámetros de la Ruta	✗
Parámetros de Consulta	✗
Parámetros del Cuerpo	*id:* String que representa el número de identificación del paciente. *name:* String que representa el nombre del paciente. *sex:* String que representa el género de paciente. *diagnostic:* String que representa el diagnostico que tiene el momento de creación el paciente. *weight:* Dato entero que representa el valor del peso del paciente en kilogramos. *height:* Dato entero que representa la altura del paciente en centímetros.
Respuesta	Estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si el paciente fue creado. Caso contrario, el valor es *False*.

Tabla B.2: Servicio web para la creación de pacientes.

Ruta	**/patients/search**
Método	GET
Descripción	Permite la búsqueda de un paciente por el número de identificación o su nombre.
Autorización	✓
Parámetros de la Ruta	✗
Parámetros de Consulta	*query:* String que representa el número de identificación o el nombre de un paciente.
Parámetros del Cuerpo	✗
Respuesta	Estructura de datos *ServerResponse* en donde el campo *response* posee una lista de pacientes que cumplen con la consulta.

Tabla B.3: Servicio web para búsqueda de pacientes.

Ruta	**/patients/{id}**
Método	GET
Descripción	Obtener información del paciente mediante su identificador.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador del paciente.
Parámetros de Consulta	✗
Parámetros del Cuerpo	✗
Respuesta	Estructura de datos *ServerResponse* en donde el campo *response* tiene la información del paciente que ha sido buscado.

Tabla B.4: Servicios web para obtener información del paciente.

Ruta	**/patients/{id}**
Método	PUT
Descripción	Permite actualizar la información de un paciente basado en su identificador.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el número de identificación del paciente.
Parámetros de Consulta	✗
Parámetros del Cuerpo	*name:* String que representa el nombre del paciente. *sex:* String que representa el género de paciente. *diagnostic:* String que representa el diagnóstico que tiene el momento de creación el paciente. *weight:* Dato entero que representa el valor del peso del paciente en kilogramos. *height:* Dato entero que representa la altura del paciente en centímetros.
Respuesta	Estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si el paciente fue modificado. Caso contrario, el valor es *False*.

Tabla B.5: Servicios web para la edición de pacientes.

B.3. Gestión de Sesiones

Ruta	**/sessions**
Método	POST
Descripción	Permite la creación de una sesión de rehabilitación de un paciente.
Autorización	✓
Parámetros de la Ruta	✗
Parámetros de Consulta	✗
Parámetros del Cuerpo	*id:* Dato entero que representa el identificador del paciente para el cual se crea la sesión. *date:* Fecha actual en la que la sesión va a ser creada. *treatment:* String que especifica una descripción corta del tratamiento que se está aplicando a un paciente. *treatmentdesc:* String que representa una descripción más detallada del tratamiento aplicado a un paciente.
Respuesta	Estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si la sesión fue creada. Caso contrario, el valor es *False*.

Tabla B.6: Servicio web para la creación de sesiones.

Ruta	**/sessions/{id}**
Método	GET
Descripción	Permite obtener la información de una sesión según su identificador.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador de la sesión buscada.
Parámetros de Consulta	✗
Parámetros del Cuerpo	✗
Respuesta	Estructura de datos *ServerResponse* en donde el campo *response* tiene la información de la sesión.

Tabla B.7: Servicio web para recuperar una sesión.

Ruta	**/patients/{id}/sessions**
Método	GET
Descripción	Permite recuperar todas las sesiones de un paciente determinado.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador del paciente.
Parámetros de Consulta	✗
Parámetros del Cuerpo	✗
Respuesta	Estructura de datos *ServerResponse* en donde el campo *response* posee una lista de sesiones asociadas al paciente buscado.

Tabla B.8: Servicio web para recuperar todas las sesiones de un paciente.

Ruta	**/sessions/{id}/status**
Método	PUT
Descripción	Permite actualizar el estado de una sesión previamente creada.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador de la sesión.
Parámetros de Consulta	✗
Parámetros del Cuerpo	*status:* String que define el estado actual de una sesión de rehabilitación.
Respuesta	Estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si la sesión fue modificada. Caso contrario, el valor es *False*.

Tabla B.9: Servicio web para modificar el estado de una sesión.

url	**/sessions/{id}/functionaldiagnostic**
Método	PUT
Descripción	Permite actualizar el diagnóstico funcional de una sesión terminada.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador de la sesión que se va a actualizar.
Parámetros de Consulta	✗
Parámetros del Cuerpo	*functionaldiagnostic:* String que representa el diagnóstico funcional de una sesión.
Respuesta	Estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si la sesión fue modificada. Caso contrario, el valor es *False*.

Tabla B.10: Servicios web para modificar el diagnóstico funcional de una sesión.

B.4. Gestión de Datos de los Sensores

Ruta	**/sessions/{id}/points**
Método	GET
Descripción	Retorna todos los datos de posición angular de los motores de una sesión de rehabilitación específica.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador de la sesión de rehabilitación de la cual se debe obtener la información de los sensores.
Parámetros de Consulta	✗
Parámetros del Cuerpo	✗
Respuesta	Estructura de datos *ServerResponse* en donde el campo *response* posee una lista de datos de los sensores que cumplen con la consulta.

Tabla B.11: Servicio web para obtener un conjunto de datos de sensores.

Ruta	**/sessions/{id}/points**
Método	POST
Descripción	Permite almacenar en una base de datos la información de la posición angular de los motores.
Autorización	✓
Parámetros de la Ruta	*id:* String que representa el identificador de la sesión en la cual se almacenará la información.
Parámetros de Consulta	*limb:* Dato entero que representa una de las extremidades inferiores. *joint:* Dato entero que representa una de las articulaciones.
Parámetros del Cuerpo	*sensor:* String que representa el nombre del sensor de cual se obtuvo la información. *variable:* String que representa el tipo de dato que se está almacenando. Por ejemplo, la posición angular de los motores. *points:* String que representa un listado de pares posición y marca temporal que deben ser almacenados en una base de datos.
Respuesta	Estructura de datos *ServerResponse* en donde el campo *state* tiene valor *True* si los datos fueron almacenados. Caso contrario, el valor es *False*.

Tabla B.12: Servicio web para almacenar un conjunto de datos de sensores.

ANEXO C

Configuración de *AllexControl*

En este anexo se detalla el proceso de instalación y configuración del *software* requerido para ejecutar el subsistema *AllexControl* en una *Raspberry Pi* modelo B. El primer paso consiste en instalar el sistema operativo *GNU/Linux* RT PREEMPT (sección C.1). A continuación se instalan el *middleware* ROS2 así como sus dependencias (sección C.2). Finalmente, se instalan dependencias de comunicación como el módulo *bluetooth*, comunicación por el puerto serial y las librerías para la comunicación con *EPOS4* mediante el protocolo CAN (sección C.3).

C.1. Sistema Operativo

Se instaló el sistema operativo *Raspbian Stretch*, el cual es una distribución de GNU/Linux con herramientas, mantenimiento y soporte de la fundación *Raspberry Pi*. Dicha distribución incluye la versión 4.14 del kernel y tiene soporte para la arquitectura *armv7* de 32 *bits*. Se instaló la versión *lite* debido a que no incluye interfaz gráfica ni dependencias para *software* de usuario. La imagen de este sistema operativo se descarga mediante el siguiente enlace: `https://downloads.raspberrypi.org/raspbian_lite/images/raspbian_lite-2018-11-15/2018-11-13-raspbian-stretch-lite.zip`.

La imagen del sistema operativo debe ser cargada en una tarjeta *microSD* debido a que la *Raspberry Pi* lo utiliza como memoria de almacenamiento no volátil para datos. Se recomienda que la tarjeta *microSD* tenga como mínimo 8GB de almacenamiento debido a que se requieren 2GB para el espacio de intercambio (memoria *swap*) que permita compilar ROS2, 1.6 GB para los archivos binarios o de configuración de ROS2 y el resto para librerías, dependencias y almacenamiento de datos. Se debe ejecutar el comando *dd* del código C.1 para copiar la imagen del sistema operativo desde una computadora personal. En este caso, el parámetro *of*

del comando **dd** representa el nombre del dispositivo en el cual está montado la tarjeta SD. Este nombre se obtiene mediante el comando **fdisk -l**.

```
$ dd bs=4M if=2018-11-13-raspbian-stretch-lite.img of=/dev/sdb conv=fsync
```

Código C.1: Escritura de la imagen de Raspbian en una tarjeta microSD.

Otra funcionalidad del comando *dd* consiste en crear una imagen con la información contenida en la tarjeta *microSD*. Como resultado del proyecto de titulación se entrega una copia configurada de *Raspbian* y que fue obtenida mediante el código C.2 a continuación:

```
$ sudo dd bs=4M if=/dev/sdb | gzip > exo-crystal-Jan92019.img.gz
```

Código C.2: Crear una imagen de respaldo

Finalmente se debe agregar el *kernel* con el parche de tiempo real *PREEMPT RT* para GNU/Linux. En este caso se descargó un *kernel* precompilado desde la dirección https://github.com/lemariva/RT-Tools-RPi/tree/master/preempt-rt/kernel_4_14_74-rt44-v7%2B y se lo coloca en el directorio *boot*. En el archivo *config.txt* se modifica la siguiente línea para que el sistema operativo inicializado sea al *kernel* de tiempo real:

/boot/config.txt

```
...
kernel=vmlinuz-4.14.52-rt34+
```

C.1.1. Configuraciones de Red

El mecanismo inicial de interacción con el sistema operativo *Raspbian* requiere de un teclado para el ingreso de comandos y un monitor como periférico de salida. Para reducir la cantidad de dispositivos periféricos conectados y facilitar el proceso de desarrollo y transferencia de archivos, se sugiere activar el protocolo *Secure SHell* (SSH). De esta manera es posible administrar el dispositivo de manera remota y cifrada desde una computadora personal. Para ello se asigna la dirección IP estática *10.0.0.2* a la interfaz de red para *ethernet* (eth0), lo cual se consigue al modificar el archivo /etc/dhcpcd.conf.

/etc/dhcpcd.conf

```
...
interface eth0
metric 10000
static ip_address=10.0.0.2/24
static routers=10.0.0.1
static domain_name_servers=10.0.0.1

interface wlan0
metric 200
```

Se debe notar que en las líneas 3 y 9 el parámetro *metric* de la interfaz de red inalámbrica *wlan0* tiene mayor precedencia que la interfaz *eth0*. Esta configuración permite que las dos interfaces sean usadas al mismo tiempo.

A continuación, se debe activar el servicio SSH y habilitar su inicialización durante el arranque del sistema operativo para cual se ejecutan los comandos enunciados en el código C.3.

```
1  $ sudo systemctl enable sshd
2  $ sudo systemctl start sshd
3  $ sudo systemctl status sshd
4  ssh.service - OpenBSD Secure Shell server
5  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enab
6  Active: active (running) since Tue 2019-01-08 06:12:06 UTC; 1h 9min ago
7  Process: 398 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
8  Main PID: 423 (sshd)
9  CGroup: /system.slice/ssh.service
10         423 /usr/sbin/sshd -D
```

Código C.3: Habilitar el servicio SSH.

La configuración es exitosa si el resultado del comando *systemctl status sshd* es *active (running)*. En este caso es posible ejecutar el comando del código C.4 desde una máquina remota conectada por ethernet. Como parámetros, se ingresa el nombre de usuario pi (usuario por defecto en una *Raspberry Pi*) y la contraseña correspondiente.

```
1  $ ssh pi@10.0.0.2
2  pi@10.0.0.2's password:
```

Código C.4: Iniciar el cliente SSH.

Finalmente, para descargar las dependencias del sistema de control centralizado se requiere de conexión a una red inalámbrica. Para ello se edita el archivo wpa_supplicant.conf para autorizar la conexión a una red cuyo SSID (Service Set Identifier) se denomina *pi* como se presenta a continuación:

/etc/wpa_supplicant/wpa_supplicant.conf

```
1  ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
2  update_config=1
3
4  network={
5  ssid="pi"
6  psk=----
7  scan_ssid=1
8  key_mgmt=WPA-PSK
9  }
```

El punto de acceso *pi* es creado desde una computadora portátil. Para crearlo en el sistema operativo GNU/Linux se usa el *software create_ap*, el cual es instalado y ejecutado mediante los comandos enunciados en el código C.5:

```
$ git clone https://github.com/oblique/create_ap
$ cd create_ap
$ make install
$ sudo create_ap wlp2s0 wlp2s0 pi raspberry
```

Código C.5: Crear el punto de acceso create_ap.

C.1.2. Memoria de Intercambio

El proceso de compilación de ROS2 que se describe en la sección C.2 requiere de un espacio de memoria RAM superior a la disponible en el *Raspberry Pi* (1 GB). Para solucionar este problema se crea un archivo de intercambio en la ruta /mnt/swapfile con 2GB de capacidad como se detalla en el código C.6.

```
$ dd if=/dev/zero of=/mnt/swapfile bs=1024 count=2097152
$ chmod 600 /mnt/swapfile
$ mkswap /mnt/swapfile
$ swapon /mnt/swapfile
```

Código C.6: Creación de un archivo de intercambio.

C.2. Instalación de ROS2

Como parte del desarrollo de *AllexControl* se instaló y configuró la tercera versión estable de ROS2, denominada *ROS 2 Crystal Clemmys*. ROS2 tiene paquetes con binarios precompilados. Sin embargo, no existe ninguno que sea compatible con la arquitectura armv7 de 32 bits para la *Raspberry Pi*. Por ese motivo se debe compilar ROS2 a partir del código fuente. El primer paso es incluir el repositorio de librerías, paquetes y archivos binarios requeridos para la instalación de ROS2 como se muestra en el conjunto de instrucciones del código C.7.

```
$ sudo apt install curl
$ curl http://repo.ros2.org/repos.key | sudo apt-key add -
$ sudo sh -c 'echo "deb [arch=amd64,arm64] http://repo.ros2.org/ubuntu/main `
    lsb_release -cs` main" > /etc/apt/sources.list.d/ros2-latest.list'
$ sudo apt update
$ sudo apt upgrade
```

Código C.7: Repositorio de dependencias de ROS2 para Debian.

A continuación, se descargan las dependencias para la administración de los archivos configurables y la comunicación entre procesos. Para ello se ejecuta el conjunto de comandos que se presenta en el código C.8.

```
# Paquetes para administrar dependencias y compilación.
$ sudo apt install -y build-essential cmake git python3-colcon-common-extensions
    python3-pip python-rosdep python3-vcstool wget
```

```
# Dependencias de FastRTPS
$ sudo apt install --no-install-recommends -y libasio-dev libtinyxml2-dev
# Paquetes requeridos a partir de Crystal Clemmyshttps://www.overleaf.com/project/5ae8f4df514dd4664757d7c3
$ sudo apt install libpcre3 libpcre3-dev libxml2-utils
# Paquetes para la ejecución de pruebas
$ sudo -H python3 -m pip install -U argcomplete flake8 flake8-blind-except flake8-builtins flake8-class-newline flake8-comprehensions flake8-deprecated flake8-docstrings flake8-import-order flake8-quotes pytest-repeat pytest-rerunfailures
$ sudo python3 -m pip install -U pytest pytest-cov pytest-runner setuptools
```

Código C.8: Dependencias para ROS2.

ROS2 tiene una estructura de microkernel, como se indicó en la sección 2.4.3. Como resultado tiene múltiples paquetes independientes. La herramienta *vcstool* se encarga de la administración y descarga de las versiones apropiadas y compatibles del código fuente de ROS2. Esto se consigue al ejecutar el conjunto de comandos en el código C.9.

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
wget https://raw.githubusercontent.com/ros2/ros2/release-latest/ros2.repos
vcs import src < ros2.repos
```

Código C.9: Descarga de ROS2.

La compilación de ROS2 es un proceso largo y que computacionalmente requiere de múltiples recursos de procesamiento y almacenamiento. Con el objetivo de reducir el tiempo de compilación se ignoran aquellos paquetes que no van a ser utilizados dentro de la *Raspberry Pi*. Por ejemplo, aquellos paquetes asociados a la simulación gráfica, percepción y navegación. Esto incluye los paquetes de proveedores de tecnología DDS como *OpenSplice* o *Connext*, considerando que en el presente proyecto se usa Fast-RTPS. Para ello se ejecuta el comando declarado en el código C.10. Esto genera un conjunto de archivos con el nombre *COLCON_IGNORE* que le indican a la herramienta de compilación que dichos paquetes no deben ser compilados.

```
$ cd ~/ros2_ws
$ touch \
  src/ros-planning/navigation_msgs/COLCON_IGNORE \
  src/ros-perception/laser_geometry/COLCON_IGNORE \
  src/ros2/kdl_parser/COLCON_IGNORE \
  src/ros2/ros1_bridge/COLCON_IGNORE \
  src/ros2/urdf/COLCON_IGNORE \
  src/ros2/demos/COLCON_IGNORE \
  src/ros2/rmw_connext/COLCON_IGNORE \
  src/ros2/rmw_opensplice/COLCON_IGNORE \
  src/ros2/rosidl_typesupport_opensplice/COLCON_IGNORE \
  src/ros2/rosidl_typesupport_connext/COLCON_IGNORE \
  src/ros2/orocos_kinematics_dynamics/COLCON_IGNORE \
  src/ros2/robot_state_publisher/COLCON_IGNORE \
  src/ros2/rviz/COLCON_IGNORE \
  src/ros2/rcl/rcl/test/COLCON_IGNORE \
  src/ros2/urdfdom/COLCON_IGNORE \
```

```
src/ros2/geometry2/COLCON_IGNORE \
src/ros2/system_tests/COLCON_IGNORE \
src/ros-visualization/rqt/COLCON_IGNORE \
src/ros-visualization/python_qt_binding/COLCON_IGNORE \
src/ros-visualization/rqt_msg/COLCON_IGNORE \
src/ros-visualization/rqt_publisher/COLCON_IGNORE \
src/ros-visualization/rqt_service_caller/COLCON_IGNORE \
src/ros-visualization/rqt_srv/COLCON_IGNORE \
src/ros-visualization/qt_gui_core/COLCON_IGNORE \
src/ros-visualization/rqt_console/COLCON_IGNORE \
src/ros-visualization/rqt_plot/COLCON_IGNORE \
src/ros-visualization/rqt_py_console/COLCON_IGNORE \
src/ros-visualization/rqt_shell/COLCON_IGNORE \
src/ros-visualization/rqt_top/COLCON_IGNORE \
```

Código C.10: Ignorar la compilación e instalación de paquetes.

La herramienta *colcon* [94] es la encargada de compilar todos los paquetes de ROS2. Para ello se ejecutan los comandos que se presentan en el código C.11. Se usa el parámetro *–executor sequential* para compilar un paquete a la vez, es decir, de manera secuencial. Esto evita errores por falta de memoria RAM en dispositivos con limitaciones de de recursos como en el caso del *Raspberry Pi*.

```
$ cd ~/ros2_ws
$ colcon build --symlink-install --executor sequential
$ . install/setup.bash
```

Código C.11: Compilación de ROS2.

C.3. Instalación de Dependencias

A continuación se presenta el proceso de instalación para las dependencias de los dispositivos de *hardware* de *AllexControl*. Esto incluye las librerías del controlador de posición *EPOS4*, la conexión *bluetooth* y la comunicación serial.

C.3.1. Librería para los Controladores de Posición

Se requiere la librería *EPOS LINUX* [92] en su versión 6.4.1.0 para interactuar con los controladores de posición *EPOS4* para motores maxon. Esta librería de comandos está disponible en la dirección `https://www.maxonmotor.de/medias/sys_master/root/8831822430238/EPOS-Linux-Library-En.zip`. Para su instalación se ejecutan los comandos del código C.12.

```
$ unzip ~/EPOS-Linux-Library-En.zip
$ sudo bash ~/EPOS_Linux_Library/install.sh -i
```

Código C.12: Instalación de la librería EPOS.

El protocolo CAN constituye la interfaz de comunicación entre el *Raspberry Pi* y los controladores *EPOS4*. En los puertos GPIO del *Raspberry Pi 3* se debe conectar la tarjeta PiCAN 2 para permitir la comunicación mediante el protocolo CAN.

Para completar la configuración desde el *software* se deben añadir las siguientes líneas al archivo /boot/config.txt para activar la comunicación SPI.

/boot/config.txt

```
...
dtparam=spi=on
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=spi-bcm2835-overlay
enable_uart=1
```

Finalmente, la interfaz de comunicación se activa mediante el comando presentado en el código C.13.

```
$ sudo /sbin/ip link set can0 up type can bitrate 1000000
```

Código C.13: Inicialización de la interfaz CAN.

C.3.2. Comunicación Bluetooth

Para interactuar con el *hardware* del *bluetooth* se requiere instalar controladores para el sistema operativo y librerías para el código de aplicación. Considerando que el bloque *bluetooth* se programa en el lenguaje de programación *python*, se debe instalar la librería *pybluez* como se presenta en el código C.14.

```
$ sudo apt-get install bluetooth libbluetooth-dev
$ sudo python3 -m pip install pybluez
```

Código C.14: Paquetes para interactuar con el módulo bluetooth.

En la versión 5 de *bluez* existe un error por el cual *sdptool* no puede acceder al dispositivo de *bluetooth* señalando el error:
bluetooth.btcommon.BluetoothError: (2, 'No such file or directory')
En este caso se debe ejecutar el *bluetooh* en modo de compatibilidad al reemplazar la línea *ExecStart* del archivo *dbus-org.bluez.service* por:

```
ExecStart=/usr/lib/bluetooth/bluetoothd --compat
```

Código C.15: Modo de compatibilidad de bluetooth.

C.3.3. Comunicación Serial

La comunicación con la batería se realiza mediante el puerto serial. Por defecto los puertos seriales están desactivados en *Raspbian*. Para activar la comunicación serial se ejecuta el comando definido en el código C.16.

```
$ sudo raspi-config
```

Código C.16: Comando para configuración del Raspberry Pi.

Esta acción despliega la interfaz gráfica con un menú para la configuración del *software* del *Raspberry Pi*. En esta pantalla se ejecutan los siguientes pasos:

1. Seleccionar la quinta opción denominada *Interfacing Options* como se muestra en la figura C.1.
2. En el siguiente menú seleccionar la opción *P6 Serial* como se muestra en la figura C.2.
3. Habilitar el puerto de *hardware* para la comunicación serial como se muestra en la pantalla C.3.
4. Finalmente aceptar los cambios realizados.

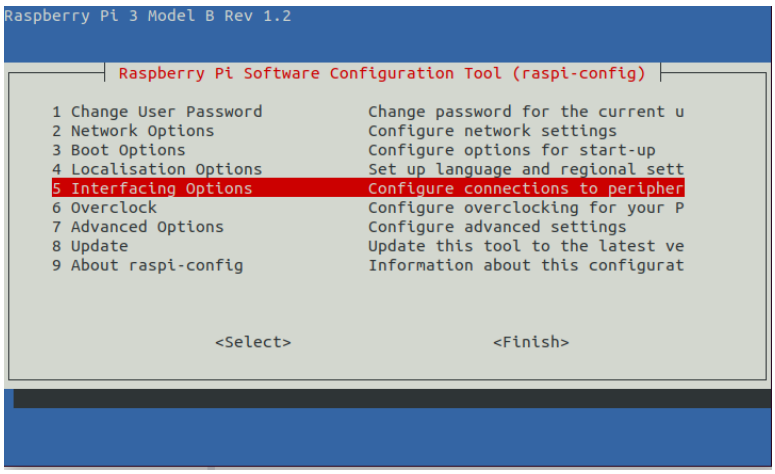


Figura C.1: Opciones de configuración del Raspberry Pi.

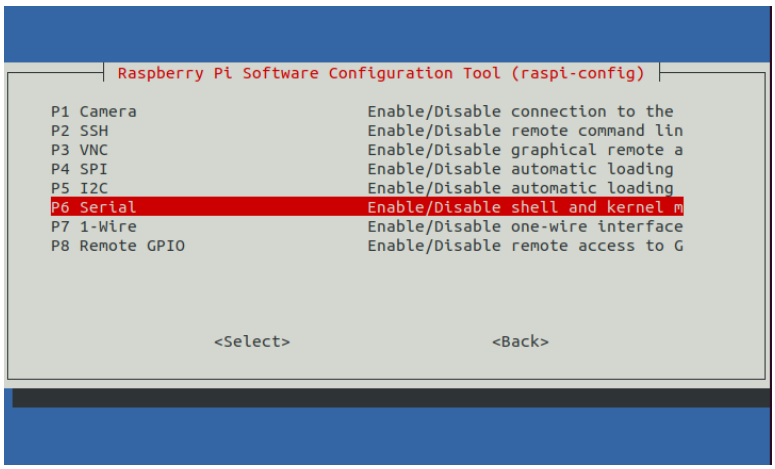


Figura C.2: Seleccionar la configuración del puerto serial.

A continuación, se debe instalar la librería *pyserial* para interactuar con el puerto serial desde el lenguaje de programación *python* como se presenta en el código C.17.

```
1 $ sudo pip3 install pyserial
```

Código C.17: Instalar la librería de interacción con el puerto serial.

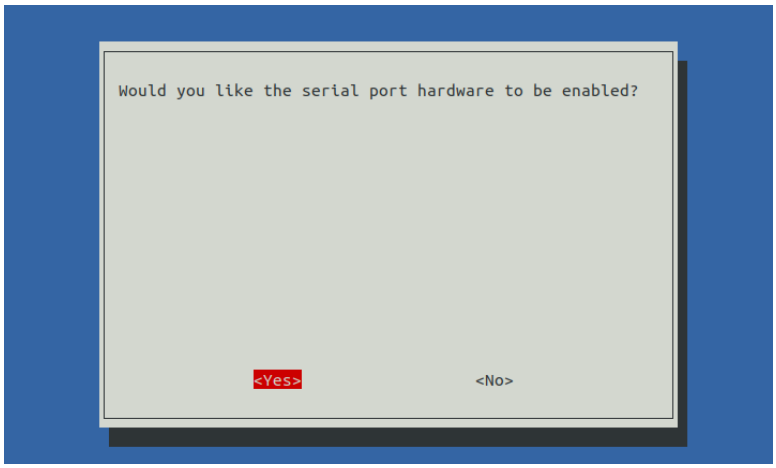


Figura C.3: Activar el hardware para el puerto serial.

Finalmente, se debe evaluar si el puerto serial funciona correctamente. Para ello se instala y ejecuta el paquete *minicom* como se muestra en el código C.18. Se colocan dos cables puente en los puertos 14 y 15 del GPIO del *Raspberry Pi*.

```
$ sudo apt install minicom
$ sudo minicom -D /dev/ttyS0
```

Código C.18: Instalar la librería de interacción con el puerto serial.

Si los puertos están en funcionamiento, la consola debe permitir la escritura de caracteres como se presenta en la figura C.4. En este caso se escribió la palabra *Funciona* y la interfaz gráfica del comando muestra dicho texto, con lo cual se demuestra que la funcionalidad está activa.

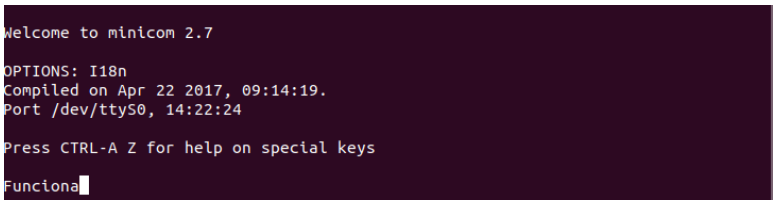


Figura C.4: Interfaz del comando minicom.

ANEXO D

Archivos de Configuración

En el presente anexo se presenta la estructura de un conjunto de archivos de configuración en formato *yaml* que son usados para alterar el comportamiento de la aplicación sin que exista la necesidad de recompilar el código. Los cuatro primeros archivos están asociados a los parámetros para la ejecución de los nodos de *AllexControl* durante el despliegue automatizado. Estos archivos están asociados a la configuración de los nodos del *Sistema de Gestión de Energía* (sección D.1), de los nodos de conexión (sección D.2), los nodos de intención de movimiento (sección D.3) y los nodos de las extremidades (sección D.4). Finalmente se define el archivo para la configuración de las trayectorias de posición angular para cada articulación (sección D.5).

D.1. Configuración para los Nodos de Batería

```yaml
exo_battery: #Nodo de batería
  ros___parameters: #Inicio de parámetros para ROS2
    node_name: exo_battery #Nombre del nodo de batería en ROS2
    serial_port: /dev/ttyS0 #Directorio para el puerto serial
    baudrate: 9600 #Unidades de señal por segundo
    enabled: true #El nodo debe ser ejecutado
    test: true #Determinar si el nodo está en ambiente de prueba
    variable:
      cell_voltage:
        read: true  #Se debe leer el voltaje de las celdas
      temperature:
        read: true #Se debe leer la temperatura
      voltage:
        read: true #Se debe leer el voltaje total del exoesqueleto
      current:
        read: true #Se debe leer la corriente
exo_battery_test: #Nodo para la simulación del Sistema de Gestion de Energía
  ros___parameters:
```

```
    serial_port: /dev/ttyS0
    baudrate: 9600
```

D.2. Configuración de los Nodos de Comunicación

```
exo_btTx: #Nodo para la transmisión de datos por bluetooth
  ros__parameters:
    node_name: exo_btTx
    enabled: true #El nodo debe ser activado
    test: false #El nodo no está en ambiente de prueba
exo_btRx: #Nodo para la recepción de datos por bluetooth
  ros__parameters:
    node_name: exo_btRx
    enabled: true #El nodo debe ser activado
    test: false #El nodo no está en ambiente de prueba
```

D.3. Configuración de los Nodos de Intención de Movimiento

```
exo_intention: #Nodo para la planificación de movimiento
  ros__parameters:
    enabled: true #El nodo debe ser activado
exo_intention_emg: #Nodo para la detección de intención de movimiento por
    electromiografía
  ros__parameters:
    enabled: true #El nodo debe ser activado
    test: true #El nodo está en ambiente de prueba
exo_intention_eeg: #Nodo para la detección de intención de movimiento por
    electroencefalografía
  ros__parameters:
    enabled: false #El nodo debe ser activado
    test: true #El nodo está en ambiente de prueba
```

D.4. Configuración de los Nodos de Extremidades

```
l_limb: #Nodo para la extremidad izquierda
  ros__parameters:
    node_name: l_limb
    enabled: true #El nodo debe ser activado
    test: false #El nodo no está en ambiente de prueba
    device_name: EPOS4 #Nombre del controlador de posición
    protocol_stack_name: CANopen #Protocolo de comunicación con el controlador
      de posición
    interface_name: CAN_mcp251x 0 #Interfaz de comunicación con el controlador
      de posición
    port_name: CAN0 #Puerto de comunicación en el sistema operativo
```

```
    hip_enabled: false #El nodo de la cadera izquierda debe ser activado
    knee_enabled: true #El nodo de la rodilla izquierda debe ser activado
    ankle_enabled: true #El nodo del tobillo izquierdo debe ser activado
    emg_enabled: true #El nodo de EMG debe ser activado
    eeg_enabled: true #El nodo de EEG debe ser activado
l_hip_sensor: #Parámetros del nodo de sensores de la cadera izquierda
  ros__parameters:
    test: false #El nodo no está en ambiente de prueba
    variable:
      position:
        read: true #Se deben leer los datos de posición
      velocity:
        read: true #Se deben leer los datos de velocidad
      current:
        read: true #Se deben leer los datos de corriente
l_knee_sensor: #Parámetros del nodo de sensores de la rodilla izquierda
  ros__parameters:
    test: false
    variable:
      position:
        read: true
      velocity:
        read: true
      current:
        read: true
l_ankle_sensor: #Parámetros del nodo de sensores del tobillo izquierdo
  ros__parameters:
    test: false
    variable:
      position:
        read: true
      velocity:
        read: true
      current:
        read: true
l_hip_actuator: #Parámetros para los motores de la cadera izquierda
  ros__parameters:
    test: false #El nodo no está en ambiente de prueba
l_knee_actuator: #Parámetros para los motores de la rodilla izquierda
  ros__parameters:
    test: false
l_ankle_actuator: #Parámetros para los motores del tobillo izquierdo
  ros__parameters:
    test: false
exo_emg: #Parámetros para la adquisición de datos de electromiografía
  ros__parameters:
    test: true #El nodo está en ambiente de prueba
    variable:
      emg:
        read: true #Se deben leer los datos de electromiografía
exo_eeg: #Parámetros para la adquisición de datos de electroencefalografía
  ros__parameters:
    test: true #El nodo está en ambiente de prueba
    variable:
```

```
    eeg:
      read: true #Se deben leer los datos de electroencefalografía
```

D.5. Configuración de la Trayectoria de Movimiento

```
gait: #Nombre del movimiento, en este caso es la caminata
  id : 0 # Identificador numérico del movimiento
  l_hip: #Trayectoria de la cadera izquierda
    initial_movement: #Movimiento desde la posición inicial hacia la posición
      previa al movimiento cíclico
      target_position : [0, 30] #Posiciones angulares máximas y mínimas para la
      trayectoria
      cycle_percent : [0, 1] #Intervalos de tiempo en los cuales se produce un cambio
       en la trayectoria
    cyclic_movement: #Movimiento cíclico de la terapia de rehabilitación
      target_position : [30, −15, 35, 30]
      cycle_percent : [0, 0.52, 0.87, 1]
    stop_movement: #Movimiento desde el final de la trayectoria cíclica hasta
      una posición de reposo
      target_position : [30, 0]
      cycle_percent : [0, 1]
  l_knee: #Trayectoria de la rodilla izquierda
    initial_movement:
      target_position : [0, −3]
      cycle_percent : [0, 1]
    cyclic_movement:
      target_position : [−3, −13, −3, −65, −3]
      cycle_percent : [0, 0.18, 0.4, 0.69, 1]
    stop_movement:
      target_position : [−3, 0]
      cycle_percent : [0, 1]
  l_ankle: #Trayectoria del tobillo izquierdo
    initial_movement:
      target_position : [0, 4]
      cycle_percent : [0, 1]
    cyclic_movement:
      target_position : [4, 18, −8, 24, 2, 8, 4]
      cycle_percent : [0, 0.08, 0.38, 0.67, 0.79, 0.92, 1]
    stop_movement:
      target_position : [4, 0]
      cycle_percent : [0, 1]
  r_hip: #Trayectoria de la cadera derecha
    initial_movement:
      target_position : [0, 30]
      cycle_percent : [0, 1]
    cyclic_movement:
      target_position : [30, −15, 35, 30]
      cycle_percent : [0, 0.52, 0.87, 1]
    stop_movement:
      target_position : [30, 0]
      cycle_percent : [0, 1]
```

```
r_knee: #Trayectoria de la rodilla derecha
  initial_movement:
    target_position : [0, −3]
    cycle_percent : [0, 1]
  cyclic_movement:
    target_position : [−3, −13, −3, −65, −3]
    cycle_percent : [0, 0.18, 0.4, 0.69, 1]
  stop_movement:
    target_position : [−3, 0]
    cycle_percent : [0, 1]
r_ankle: #Trayectoria del tobillo derecho
  initial_movement:
    target_position : [0, 4]
    cycle_percent : [0, 1]
  cyclic_movement:
    target_position : [4, 18, −8, 24, 2, 8, 4]
    cycle_percent : [0, 0.08, 0.38, 0.67, 0.79, 0.92, 1]
  stop_movement:
    target_position : [4, 0]
    cycle_percent : [0, 1]
```

ANEXO E

Manual de Usuario

En este capítulo se redacta el manual de usuario para el manejo del subsistema de gestión de terapias de rehabilitación (*AllexApp*) que se obtuvo como producto del presente proyecto de titulación. Para ello se presentan las generalidades de su funcionalidad y ambiente de ejecución (sección E.1), así como las interfaces gráficas para la administración de pacientes y control del exoesqueleto (sección E.2).

E.1. Resumen del Sistema

AllexApp es una aplicación que permite gestionar las sesiones de rehabilitación para pacientes que usan el exoesqueleto ALLEX-1 como complemento a su tratamiento. Para conseguir este objetivo, la aplicación provee las siguientes funcionalidades:

- Creación y edición de fichas médicas para pacientes.
- Creación de sesiones de rehabilitación para un paciente.
- Iniciar el proceso de adquisición de datos del exoesqueleto.
- Configurar la ejecución de un movimiento de rehabilitación.

AllexApp se ejecuta únicamente en dispositivos móviles que tengan una interfaz gráfica para el sistema operativo *Android*. Las versiones de *Android* que tienen soporte para la aplicación están en el rango de la versión 4.4 (KitKat) como mínimo y 8.1 (Oreo) como máximo. Para poder usar todas las funcionalidades del sistema se requiere que el dispositivo tenga conexión a una red local inalámbrica. Adicionalmente es necesario que el módulo *bluetooth* del dispositivo móvil esté activo para poder gestionar la conexión con el exoesqueleto. Todos los usuarios que tengan asignadas credenciales de dirección de correo y contraseña pueden usar todas las funcionalidades del sistema.

E.2. Uso del Sistema

La aplicación no está disponible para el público general, es decir, no puede ser descargada desde la tienda de aplicaciones de *Android*. Por tanto, se requiere instalar el archivo de extensión *.apk* de *AllexApp* en el dispositivo móvil a ser usado. Para iniciar la aplicación, el usuario debe presionar en el icono de *AllexApp* que se presenta en la figura E.1.

Figura E.1: Logo de la aplicación *AllexApp*.

E.2.1. Ingreso a la Aplicación

La primera pantalla que se carga al momento de iniciar la aplicación es la de ingreso al sistema que se presenta en la figura E.2 a continuación.

Figura E.2: Pantalla de ingreso a *AllexApp*.

Esta pantalla presenta dos campos para el ingreso de las credenciales del usuario como son su dirección de correo electrónico y el espacio para que ingrese la contraseña asignada por el administrador. Cuando se presiona en el botón *Ingresar* inicia el proceso de validación de las credenciales del usuario. En esta pantalla existen dos escenarios de error. El primero, es aquel en el cual existen campos que no han sido rellenados con texto. Como resultado se obtiene una pantalla similar a la que se presenta en la figura E.3. Los campos que no han sido completados están marcados de color rojo y con un símbolo de advertencia que facilita su identificación y corrección.

Figura E.3: Error por campos vacíos en el formulario de ingreso de *AllexApp*.

Otro caso de error se presenta cuando las credenciales ingresadas no son válidas. En dicho escenario se presenta un cuadro de diálogo que informa que el usuario no está autorizado para acceder a los recursos de la aplicación como se presenta en la figura E.4.

Figura E.4: Error por el ingreso de credenciales incorrectas.

En caso de que todas las credenciales sean correctas el usuario ingresa al menú principal de la aplicación.

E.2.2. Menú Principal de la Aplicación

Es la primera pantalla a la cual el usuario tiene acceso cuando recibe la autorización de ingresar al sistema. Como se presenta en la figura E.5, el menú corresponde a una barra de navegación vertical con las diferentes funcionalidades del sistema.

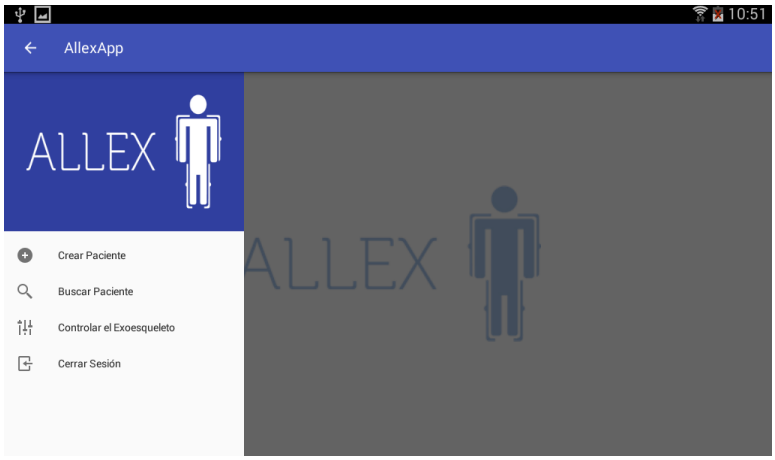


Figura E.5: Menú principal de la aplicación.

En este caso existen cuatro opciones de menú como son:

- *Crear Paciente:* Al presionar en esta opción se carga el formulario para la creación de un paciente como se presenta en la sección E.2.3.
- *Buscar Paciente:* Permite buscar un paciente previamente creado en el sistema. Esta pantalla se describe en la sección E.2.4.
- *Controlar el Exoesqueleto:* El usuario inicia el proceso de conexión con el exoesqueleto como se presenta en la sección E.2.6.
- *Cerrar Sesión:* Al presionar en esta opción, el usuario regresa a la pantalla de ingreso al sistema puesto que las credenciales de autorización de acceso a los recursos han sido revocadas.

E.2.3. Crear Paciente

Corresponde al formulario que permite la creación de un nuevo paciente en el sistema como se presenta en la figura E.6.

Figura E.6: Formulario de creación del paciente.

Este formulario posee campos para el ingreso de caracteres alfanuméricos

como el identificador del paciente (cédula o pasaporte), nombre del paciente y diagnóstico médico. Los campos para la altura y peso del paciente permiten únicamente el ingreso de valores numéricos. La fecha de nacimiento es seleccionada mediante el cuadro de diálogo que se presenta en la figura E.7 y el género usa un menú desplegable. El botón *Cancelar* es usado para que el usuario regrese al menú principal de la aplicación.

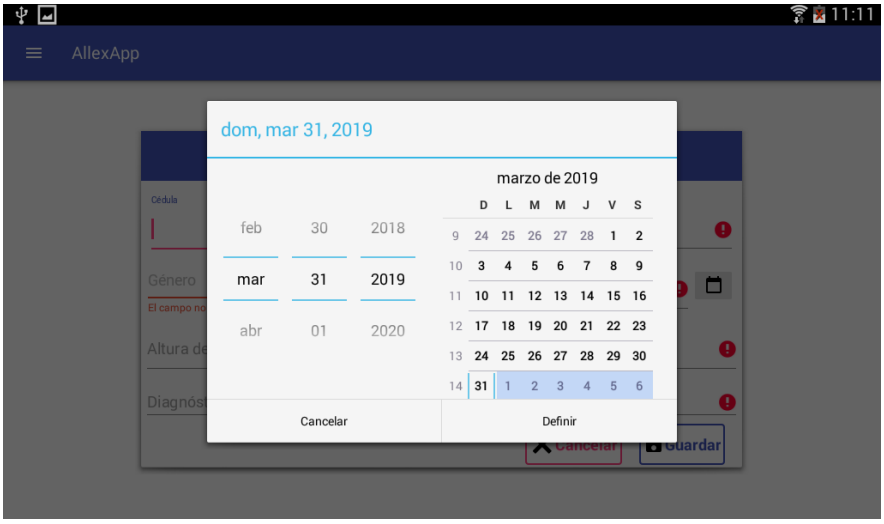


Figura E.7: Cuadro de diálogo para la selección de la fecha.

En esta pantalla existen dos casos de error. El primero se genera cuando existen campos que no tienen valores asignados. En la figura E.8 dichos campos se presentan con un icono de color rojo y etiquetas de ayuda que facilitan la identificación de los errores. El segundo caso se genera cuando ya existe un paciente previamente registrado con el mismo identificador. Si se presenta esta situación se muestra un cuadro de diálogo de error.

Figura E.8: Error por campos vacíos en el formulario de ingreso de *AllexApp*.

E.2.4. Buscar Paciente

Esta pantalla permite ejecutar la búsqueda de pacientes al escribir el identificador o nombre del mismo en el cuadro de texto que se presenta en la esquina superior derecha de la pantalla (figura E.9).

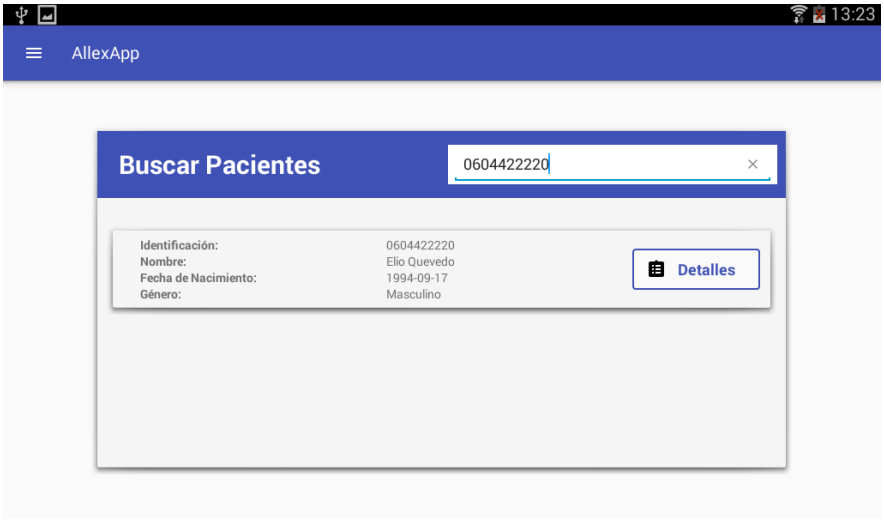


Figura E.9: Pantalla de búsqueda de pacientes.

Si existen coincidencias se muestra un listado de pacientes con un nombre o identificador que es similar al que se está buscando. Cada uno de los elementos de la lista posee información como el nombre del paciente, el género y la fecha de nacimiento. En el extremo derecho de cada elemento de la lista se encuentra el botón *Detalles*. Si el usuario presiona dicho botón, se carga la pantalla de información del paciente.

E.2.5. Pantalla de Información del Paciente

Esta pantalla consta de dos secciones como se representa en la figura E.10. En el lado izquierdo se encuentra el detalle del paciente. En esta zona se muestran aquellos datos que fueron registrados en la pantalla de creación de la sección E.2.3. Por otro lado, en el extremo derecho de la pantalla se encuentra el espacio para las sesiones de rehabilitación ejecutadas.

Figura E.10: Pantalla de información de pacientes.

En la zona de detalle del paciente se encuentra el botón *Editar*. Al presionar en este botón se reemplaza la zona de detalle del paciente por un formulario de edición de datos en donde es posible cambiar el valor de todos los campos con excepción del identificador tal y como se presenta en la figura E.11. Si el usuario no desea almacenar los cambios realizados en el servidor, debe presionar en el botón *Cancelar* que es de color rojo. Caso contrario debe presionar el botón *Guardar* para solicitar al servidor que almacene los cambios solicitados. En ambos casos la aplicación regresa al estado inicial con la pantalla de detalles de la ficha médica.

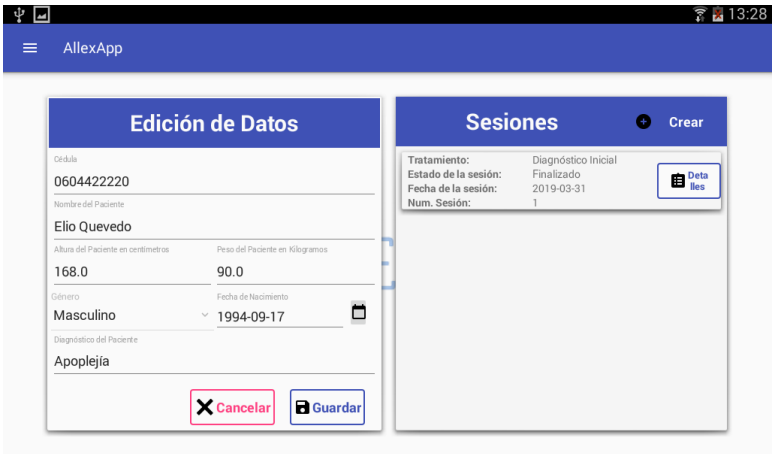


Figura E.11: Pantalla de edición de detalle de pacientes.

El segundo botón de la pantalla se encuentra en la zona del listado de sesiones de rehabilitación y se denomina *Crear*. Al presionar este botón se carga un cuadro de diálogo como el que se muestra en la figura E.12 para crear una sesión de rehabilitación.

Este cuadro de diálogo permite la creación de una sesión de rehabilitación. Para cumplir con esta meta se debe ingresar un título y descripción para la

sesión de rehabilitación. Uno de los botones en este cuadro se denomina *Guardar*, que tiene por objetivo registrar una sesión de rehabilitación en el sistema. Por otro lado, está el botón con el icono de cerrar el cual le permite al usuario cancelar la transacción y regresar a la pantalla de información del paciente. Si la sesión ha sido correctamente creada, inicia el proceso de conexión con el exoesqueleto mediante el protocolo *bluetooth*.

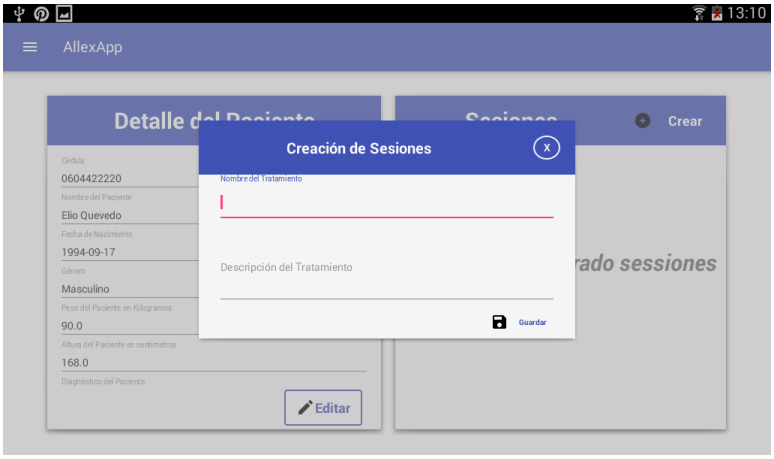


Figura E.12: Cuadro de diálogo para la creación de una sesión de rehabilitación.

E.2.6. Conexión con el Exoesqueleto

La primera pantalla que se muestra al usuario después de crear la sesión de rehabilitación es la pantalla para encender el módulo *bluetooth* del dispositivo móvil. En esta pantalla existen dos botones como se presenta en la figura E.13, los cuales son:

- *Cancelar:* Permite que el usuario regrese a la pantalla de información de paciente.
- *Encender:* Al presionar este botón el sistema operativo genera un cuadro de diálogo en el que advierte al usuario que va a otorgarle a la aplicación el permiso para activar el *bluetooth*. Si el usuario acepta, se ejecuta dicha acción.

Existen casos en los cuales esta pantalla no se carga. Esto se produce cuando el usuario ha activado el módulo *bluetooth* directamente en el menú de configuraciones del sistema operativo. En estos casos la aplicación continúa su ejecución en la pantalla de conexión.

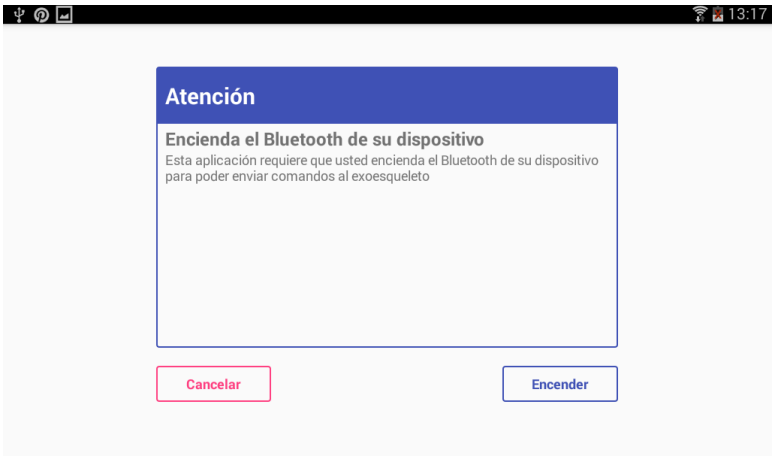


Figura E.13: Autorizar el uso de *Bluetooth* en el dispositivo.

Una vez que el dispositivo ha sido encendido, se muestra la lista de dispositivos *bluetooth* que han sido previamente emparejados como se presenta en la figura E.14. Al presionar el botón *Actualizar* se vuelve a cargar la lista de dispositivos, lo cual es útil en aquellos casos en los que se ha completado el emparejamiento con un nuevo dispositivo.

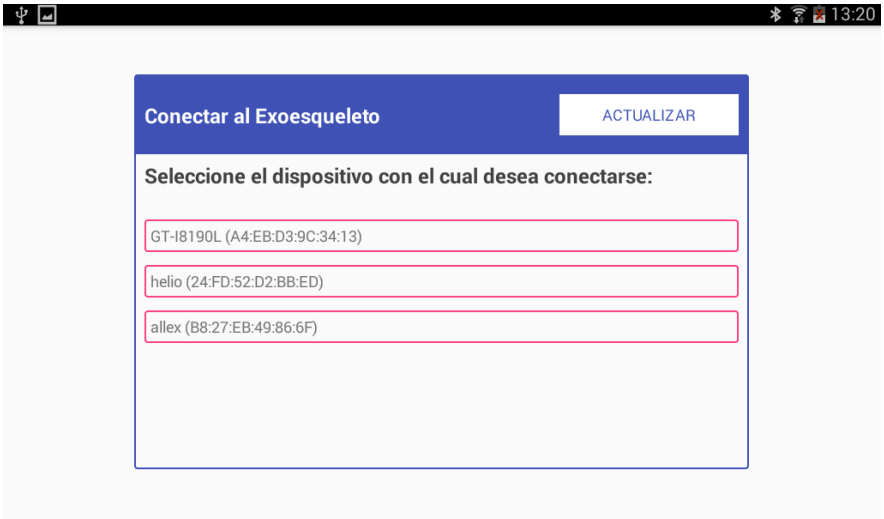


Figura E.14: Pantalla de conexión.

Cada elemento de la lista de dispositivos está representado por su nombre y dirección MAC. Para iniciar la conexión con *AllexControl* el usuario debe seleccionar el dispositivo *allex*. Si la conexión es exitosa se carga la pantalla de control que se presenta en la siguiente sección.

E.2.7. Pantalla de Control

Esta es la pantalla en la cual el usuario envía comandos al exoesqueleto. La pantalla que se presenta en la figura E.15 consta de cuatro secciones:

- En el lado superior izquierdo se encuentra la sección de *Configuración de Movimiento* en el cual se introducen los parámetros para modificar la trayectoria de los motores como el tipo de movimiento, duración del ciclo de movimiento en segundos, número de iteraciones, activar la detección de intención de movimiento por electromiografía o electroencefalografía.
- En la sección inferior izquierda se encuentra un icono para el estado de la batería. La batería puede estar en estado de carga alto, medio o bajo.
- En la esquina superior derecha se encuentran las gráficas de posición angular de los motores respecto al tiempo.
- En la esquina inferior derecha se encuentran dos botones para enviar un comando que es ejecutado por el exoesqueleto:
  - *Lectura:* Envía al exoesqueleto el comando de lectura. De esta manera el exoesqueleto inicia el proceso de adquisición de datos.
  - *Mover:* El exoesqueleto envía la rutina de configuración del movimiento que fue previamente definida por el usuario. Cuando la ejecución del exoesqueleto termina carga la pantalla de previsualización de movimientos.

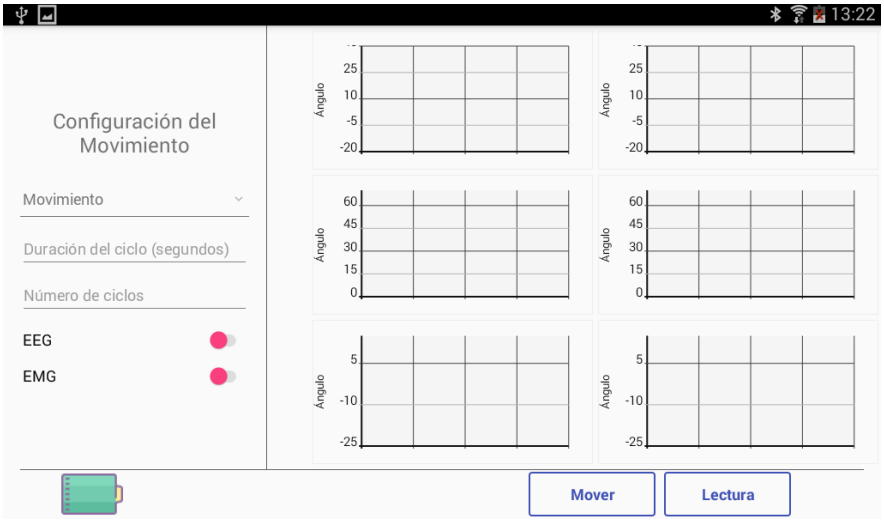


Figura E.15: Pantalla de control del exoesqueleto.

E.2.8. Pantalla de Previsualización

Esta pantalla se carga después de la ejecución de un movimiento de rehabilitación previamente configurado por el usuario y se presenta en la figura E.16. En ella se muestran las trayectorias de movimiento que ejecutó cada articulación del exoesqueleto durante el comando *Mover*. Al final de dicha pantalla existen dos botones:

- *Cancelar:* Con lo cual el usuario regresa a la pantalla de control.
- *Guardar:* Los datos adquiridos durante dicha sesión de rehabilitación son enviados a *AllexServer*.

Figura E.16: Pantalla de previsualización de resultados de una sesión de rehabilitación.

E.2.9. Pantalla de Sesión Finalizada

Es la pantalla que se carga cuando el usuario presiona en el botón *Detalles* de una sesión de rehabilitación finalizada y que se presenta en la figura E.17. Esta está formada por dos zonas:

- En la zona izquierda se presenta información de la sesión de rehabilitación como el título, la descripción, fecha y número de sesión.
- La sección derecha corresponde a las gráficas de posición angular respecto al tiempo de cada articulación.

Figura E.17: Pantalla de sesión finalizada.

Finalmente, puede agregar un diagnóstico de la sesión de rehabilitación al presionar el botón *Diagnóstico funcional* de la zona de detalle de sesiones. Cuando se presiona el botón, se carga un cuadro de diálogo con un campo de texto para el diagnóstico funcional como se presenta en la figura E.18.

Figura E.18: Pantalla para la adición del diagnóstico funcional del paciente en la sesión de rehabilitación.

Bibliografía

[1] P. Sale, M. Franceschini, A. Waldner, and S. Hesse, "Use of the robot assisted gait therapy in rehabilitation of patients with stroke and spinal cord injury," *Eur J Phys Rehabil Med*, vol. 48, no. 1, pp. 111–21, 2012.

[2] C. N. para la Igualdad de Discapacidades (CONADIS), "Estadísticas de Discapacidad." https://www.consejodiscapacidades.gob.ec/estadisticas-de-discapacidad/, 2018.

[3] N. Ozkaya, M. Nordin, D. Goldsheyder, and D. Leger, *Fundamentals of biomechanics*. Springer, 2012.

[4] A. M. Dollar and H. Herr, "Active orthoses for the lower-limbs: challenges and state of the art," in *2007 IEEE 10th International Conference on Rehabilitation Robotics*, pp. 968–977, IEEE, 2007.

[5] G. Morone, S. Paolucci, A. Cherubini, D. De Angelis, V. Venturiero, P. Coiro, and M. Iosa, "Robot-assisted gait training for stroke patients: current state of the art and perspectives of robotics," *Neuropsychiatric disease and treatment*, vol. 13, p. 1303, 2017.

[6] A. Calderón-Bernal, R. Cano-de la Cuerda, I. Alguacil-Diego, F. Molina-Rueda, A. Cuesta-Gómez, and J. Miangolarra-Page, "Terapia robótica para la rehabilitación de la marcha en patología neurológica," *Rehabilitación*, vol. 49, no. 3, pp. 177–192, 2015.

[7] M. Bortole, "Design and control of a robotic exoskeleton for gait rehabilitation," *Systems and Automation Engineering Department*, 2013.

[8] L. Li, J. R. Schnellenberger, M. J. Nandor, S. R. Chang, K. M. Foglyano, R.-D. Reyes, R. Kobetic, M. Audu, R. J. Triolo, and R. D. Quinn, "Embedded control system for stimulation-driven exoskeleton," in *2018 International Symposium on Medical Robotics (ISMR)*, pp. 1–6, IEEE, 2018.

[9] G. Durandau, M. Sartori, M. Bortole, J. C. Moreno, J. L. Pons, and D. Farina, "Real-time modeling for lower limb exoskeletons," in *Wearable Robotics: Challenges and Trends*, pp. 127–131, Springer, 2017.

[10] M. M. Bassa, "Development of the communication system for a lower limb human exoskeleton using the ros middleware," 2015.

[11] A. Walia and N. Kumar, "Powered lower limb exoskeleton featuring intuitive graphical user interface with analysis for physical rehabilitation progress," 2018.

[12] G. Aguirre-Ollinger, J. E. Colgate, M. A. Peshkin, and A. Goswami, "Inertia compensation control of a one-degree-of-freedom exoskeleton for lower-limb assistance: Initial experiments," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 20, no. 1, pp. 68–77, 2012.

[13] Rex Bionics, "Rex for clinical use." https://www.rexbionics.com/rex-for-clinical-use/, 2019.

[14] EksoGT, "Treat Together with EksoGT." https://eksobionics.com/eksohealth/eksogt/clinicians/, 2019.

[15] J. L. Cornejo, J. F. Santana, and S. A. Salinas, "Exoskeleton for gait rehabilitation of children: Conceptual design," in *2017 International Conference on Rehabilitation Robotics (ICORR)*, pp. 452–454, IEEE, 2017.

[16] L. D. Rijcke, "Development of a Graphical User Interface for a Rehabilitation Exoskeleton ," Master's thesis, Vrije Universiteit Brussel, Bélgica, 2013.

[17] J. Blandín and A. Velasco, "Diseño e implementación del sistema de instrumentación y comunicaciones de un exoesqueleto de extremidades inferiores," Master's thesis, Universidad de Cuenca, Ecuador, 2019.

[18] F. Astudillo and J. Charry, "Detección de Intención de movimiento de extremidades inferiores usando métodos de aprendizaje supervisado.," Master's thesis, Universidad de Cuenca, Ecuador, 2019.

[19] M. Mendieta and C. Muñoz, "Diseño y construcción de un sistema de gestión de energía para aplicaciones de asistencia a la movilidad en extremidades inferiores.," Master's thesis, Universidad de Cuenca, Ecuador, 2019.

[20] J. Angeles and J. Angeles, *Fundamentals of robotic mechanical systems*, vol. 2. Springer, 2002.

[21] A. H. Odorico, F. J. Lage, and Z. Cataldi, "Robótica, informática, inteligencia artificial y educación," in *IX Workshop de Investigadores en Ciencias de la Computación*, 2007.

[22] C. Balaguer, A. Barrientos, P. Sanz, R. Sanz, and E. Zalama, "Libro blanco de la robotica," *De la investigación al desarrollo y futuras aplicaciones. CEA-GTROB subencionado por el MEC. Espana*, 2007.

[23] J. Londoño, E. Bravo, and J. García, "Aplicación de tecnologías de rehabilitación robótica en niños con lesión del miembro superior," *Revista Salud UIS*, vol. 49, no. 1, 2017.

[24] International Federation of Roboctics, "Classification of service robots by application areas." https://ifr.org/img/office/Service_Robots_2016_Chapter_1_2.pdf?fbclid=IwAR1K4uXYixgmzNd-bXHqDT8OcBzIhr-DhjlpiWFqnuPaNnhUEBozpp9Jg4Q, 2019.

[25] M. Ben-Ari and F. Mondada, *Elements of robotics*. Springer International Publishing, 2018.

[26] R. Mahoney, "Service robotics case studies in silicon valley," *Silicon Valley Robotics Technical Report*, 2015.

[27] R. H. Taylor, A. Menciassi, G. Fichtinger, P. Fiorini, and P. Dario, "Medical robotics and computer-integrated surgery," in *Springer handbook of robotics*, pp. 1657–1684, Springer, 2016.

[28] D. J. Reinkensmeyer, "Rehabilitation Robot," 2018.

[29] H. Krebs and B. Volpe, "Rehabilitation robotics," in *Handbook of clinical neurology*, vol. 110, pp. 283–294, Elsevier, 2013.

[30] W. Van Dijk, *Human-Exoskeleton Interaction*. PhD thesis, Technische Universiteit Delft, 01 2015.

[31] M. A. Chávez Cardona, F. Rodríguez Spitia, and A. Baradica López, "Exoskeletons to enhance human capabilities and support rehabilitation: a state of the art," *Revista Ingeniería Biomédica*, vol. 4, no. 7, pp. 63–73, 2010.

[32] L. M. Mooney, E. J. Rouse, and H. M. Herr, "Autonomous exoskeleton reduces metabolic cost of human walking during load carriage," *Journal of neuroengineering and rehabilitation*, vol. 11, no. 1, p. 80, 2014.

[33] M. Aach, O. Jansen, M. Sczesny-Kaiser, O. Höffken, R. Meindl, M. Tegenthoff, P. Schwenkreis, Y. Sankai, and T. A Schildhauer, "Voluntary driven exoskeleton as a new tool for rehabilitation in chronic spinal cord injury – a pilot study," *The spine journal : official journal of the North American Spine Society*, vol. 14, 04 2014.

[34] A. M. Spungen, S. Kornfeld, P. K. Asselin, S. Knezevic, and M. Avedissian, "Training Persons with Spinal Cord Injury to Ambulate Using a Powered Exoskeleton," *Journal of Visualized Experiments*, no. 112, 2016.

[35] J. Riedo and K. J. Hunt, "Feedback control of heart rate during robotics-assisted end-effector-based stair climbing," *Systems Science & Control Engineering*, vol. 4, no. 1, pp. 223–234, 2016.

[36] A. J. Young and D. P. Ferris, "State of the art and future directions for lower limb robotic exoskeletons," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 2, pp. 171–182, 2017.

[37] K. Anam and A. A. Al-Jumaily, "Active exoskeleton control systems: State of the art," *Procedia Engineering*, vol. 41, pp. 988–994, 2012.

[38] R. Jimenez-Fabian and O. Verlinden, "Review of control algorithms for robotic ankle systems in lower-limb orthoses, prostheses, and exoskeletons," *Medical engineering & physics*, vol. 34, no. 4, pp. 397–408, 2012.

[39] S. Collado-Vázquez and J. M. Carrillo, "Balzac y el análisis de la marcha humana," *Neurologia*, vol. 30, no. 4, pp. 240–246, 2015.

[40] D. Mariana Haro, "Laboratorio de análisis de marcha y movimiento," *Revista Médica Clínica Las Condes*, vol. 25, no. 2, pp. 237–247, 2014.

[41] A. M. M. Nogueras, J. I. C. Arenillas, J. O. Rodríguez, F. B. Iglesias, and C. S. Sánchez, "Fases de la marcha humana," *Revista iberoamericana de fisioterapia y kinesiología*, vol. 2, no. 1, pp. 44–49, 1999.

[42] T. Pineda and D. Francisco, "Diseño e implementación de un sistema para visualizar la marcha humana biomecánica en la afectación de rodilla ante una gonartrosis," Master's thesis, Escuela Poltécnica Nacional, 2017.

[43] Universidad de Nueva York , *Ortésica del Miembro Inferior*. Digital Resource Foundation, 1986.

[44] J. Perry and J. M. Burnfield, *Análisis de la marcha: función normal y patológica*. Editorial Base, 2015.

[45] F. Romero Sánchez and B. M. Jorge, "El análisis biomecánico de la marcha en personas con discapacidad motora," *Educación Física y Deportes, Revista Digital*, vol. 230, 2017.

[46] D. Albuquerque, J. Castro, S. Ribeiro, and T. Heineck, "Requirements engineering for robotic system: A systematic mapping study," 05 2017.

[47] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i)[tutorial]," *IEEE Robotics & Automation Magazine*, vol. 16, no. 4, pp. 84–96, 2009.

[48] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part ii)," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 100–112, 2010.

[49] D. Brugali, A. Agah, B. MacDonald, I. A. Nesnas, and W. D. Smart, "Trends in robot software domain engineering," in *Software Engineering for Experimental Robotics*, pp. 3–8, Springer, 2007.

[50] I. A. Nesnas, "The claraty project: coping with hardware and software heterogeneity," in *Software Engineering for Experimental Robotics*, pp. 31–70, Springer, 2007.

[51] E. Navarro, P. Letelier, and I. Ramos, "Goals and quality characteristics: Separating concerns," in *Early Aspects*, 2004.

[52] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS component model: a model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1758–1764, ACM, 2013.

[53] V. Dominick, K. Markus, and B. Herman, "The 5c-based architectural composition pattern: lessons learned from re-developing the itasc framework for constraint-based robot programming," *Journal of Software Engineering in Robotics*, vol. 5, no. 1, pp. 17–35, 2014.

[54] R. Dorf and R. Bishop, *Modern Control Systems*. Pearson Prentice Hall, 2011.

[55] P. Albertos and I. Mareels, *Feedback and control for everyone*. Springer Science & Business Media, 2010.

[56] W. Findeisen, "Hierarchical control systems: An introduction," 1978.

[57] R. F. Stengel, "Toward intelligent flight control," *IEEE transactions on Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1699–1717, 1993.

[58] A. Kurt and Ü. Özgüner, "Hierarchical finite state machines for autonomous mobile systems," *Control Engineering Practice*, vol. 21, no. 2, pp. 184–194, 2013.

[59] L.-r. Jen and Y.-j. Lee, "Working group. ieee recommended practice for architectural description of software-intensive systems," in *IEEE Architecture*, Citeseer, 2000.

[60] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *Journal of Systems and Software*, vol. 122, pp. 16–39, 2016.

[61] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2002.

[62] S. Kounev, C. Rathfelder, and B. Klatt, "Modeling of event-based communication in component-based architectures: state-of-the-art and future directions," *Electronic Notes in Theoretical Computer Science*, vol. 295, pp. 3–9, 2013.

[63] Object Management Group (OMG), "System Modeling Language Specification (SysML), Version 1.5." OMG Document Number formal/17-05-01 (https://www.omg.org/spec/SysML/1.5/PDF), 2017.

[64] T. Noergaard, *Embedded systems architecture: a comprehensive guide for engineers and programmers*. Newnes, 2012.

[65] D. P. Möller, "Guide to computing fundamentals in cyber-physical systems," *Computer Communications and Networks. Springer, Heidelberg*, 2016.

[66] AspenCore, "2017 Embedded Markets Study - Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments." https://m.eet.com/media/1246048/2017-embedded-market-study.pdf, 2017.

[67] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating systems for low-end devices in the internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 720–734, 2016.

[68] M. Rabbah, N. Rabbah, H. Belhadaoui, and M. Rifi, "Designing middleware over real time operating system for mobile robot," in *First International Conference on Real Time Intelligent Systems*, pp. 419–425, Springer, 2017.

[69] E. Tsardoulias and P. Mitkas, "Robotic frameworks, architectures and middleware comparison," *arXiv preprint arXiv:1711.06842*, 2017.

[70] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[71] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "A review of middleware for networked robots," *International Journal of Computer Science and Network Security*, vol. 9, no. 5, pp. 139–148, 2009.

[72] G. Magyar, P. Sinčák, and Z. Krizsán, "Comparison study of robotic middleware for robotic applications," in *Emergent Trends in Robotics and Intelligent Systems*, pp. 121–128, Springer, 2015.

[73] A. Elkady and T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography," *Journal of Robotics*, vol. 2012, 2012.

[74] InfluxData, "Time Series Database (TSDB) Explained," 2019.

[75] DB-Engines, "DB-Engines Ranking of Time Series DBMS." https://db-engines.com/en/ranking/time+series+dbms, 2019.

[76] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[77] Object Management Group (OMG), "Data Distribution Service (DDS), Version 1.4." OMG Document Number formal/15-04-11 (https://www.omg.org/spec/DDS/1.4/PDF), 2015.

[78] Object Management Group (OMG), "The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification, Version 2.2." OMG Document Number formal/14-09-01 (https://www.omg.org/spec/DDSI-RTPS/2.2/PDF), 2014.

[79] Object Management Group (OMG), "Interface Definition Language, Version 4.2." OMG Document Number formal/18-01-05 (https://www.omg.org/spec/IDL/4.2/PDF), 2018.

[80] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th International Conference on Embedded Software*, p. 5, ACM, 2016.

[81] P. Letier, G. Fau, U. Mittag, J. Zange, J. Rittweger, M. Jung, J. McIntyre, and A. Runge, "Soleus: Ankle foot orthosis for space countermeasure with immersive virtual reality," in *Wearable Robotics: Challenges and Trends*, pp. 305–309, Springer, 2017.

[82] S. A. Murray, K. H. Ha, C. Hartigan, and M. Goldfarb, "An assistive control approach for a lower-limb exoskeleton to facilitate recovery of walking following stroke," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 3, pp. 441–449, 2015.

[83] W. Ma, X. Zhang, and G. Yin, "Design on intelligent perception system for lower limb rehabilitation exoskeleton robot," in *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 587–592, IEEE, 2016.

[84] G. Durandau, M. Sartori, M. Bortole, J. C. Moreno, J. L. Pons, and D. Farina, "Emg-driven models of human-machine interaction in individuals wearing the h2 exoskeleton," *IFAC-papersonline*, vol. 49, no. 32, pp. 200–203, 2016.

[85] D. Albuquerque, J. Castro, S. Ribeiro, and T. Heineck, "Requirements engineering for robotic system: A systematic mapping study," 05 2017.

[86] N. Nazmi, M. A. Abdul Rahman, S.-I. Yamamoto, S. A. Ahmad, H. Zamzuri, and S. A. Mazlan, "A review of classification techniques of emg signals during isotonic and isometric contractions," *Sensors*, vol. 16, no. 8, p. 1304, 2016.

[87] A. Olczak, "Concept of brain-controlled exoskeleton based on motion tracking and eeg signals analysis," in *International Scientific Conference BCI 2018 Opole*, pp. 141–149, Springer, 2018.

[88] W. H. Organization *et al.*, "Classifying health workers," *Geneva: WHO*, 2010.

[89] "Distribution dashboard." https://developer.android.com/about/dashboards. Accessed: 2019-03-03.

[90] I. Sommerville, *Software engineering*. New York: Addison-Wesley, 2010.

[91] I. Batchkova and I. Antonova, "Improving the software development life cycle in process control using uml/sysml," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 14133–14138, 2011.

[92] maxon motor, "EPOS Positioning Controllers Command Library." https://www.maxonmotor.nl/medias/sys_master/root/8825412714526/EPOS-Command-Library-En.pdf, 2016.

[93] M. Cabello, *Introduccion a las Bases de Datos relacionales*. Editorial Visión Libros, 2010.

[94] D. Thomas, "colcon Documentation." https://media.readthedocs.org/pdf/colcon/latest/colcon.pdf, 2018.

[95] "Ros qos - deadline, liveliness, and lifespan." https://github.com/ros2/design/blob/33431f5396955e3c0993677183f9d2fbfcdcfba5/articles/qos_deadline_liveliness_lifespan.md. Accessed: 2019-03-02.